

UNIVERSIDADE ESTADUAL PAULISTA

Instituto de Geociências e Ciências Exatas

Campus de Rio Claro

**TÉCNICAS DE MODELAGEM 3D APLICADAS A
DADOS PALEOBATIMÉTRICOS
DAS BACIAS DE SANTOS E CAMPOS E À
SIMULAÇÃO DEFORMACIONAL DE
OBJETOS GEOLÓGICOS**

Luca Pallozzi Lavorante

Orientador: Prof. Dr. Hans Dirk Ebert

Dissertação de Mestrado elaborada junto ao
Programa de Pós-Graduação em Geociências -
Área de Concentração em Geologia Regional,
para obtenção do título de Mestre em
Geociências

RIO CLARO (SP)
2005

550.0285 Lavorante, Luca Pallozzi
L414t Técnicas de modelagem 3D aplicadas a dados paleobatimétricos das bacias de Santos e Campos e à simulação deformacional de objetos geológicos / Luca Pallozzi
Lavorante. – Rio Claro : [s.n.], 2005
210 f. : il., figs., mapas + CD-ROM

Dissertação (mestrado) – Universidade Estadual Paulista, Instituto de Geociências e Ciências Exatas
Orientador: Hans Dirk Ebert

1. Geologia – Programas de computador. 2. Visualização 3D. 3. Geomodelagem. 4. Paleobatimetria. 5. Tensor.
I. Título.

Ficha Catalográfica elaborada pela STATI – Biblioteca da UNESP
Campus de Rio Claro/SP

Comissão Examinadora

Prof. Dr. Hans Dirk Ebert - Orientador

Prof. Dr. Ricardo Latgé Milward de Azevedo

Prof. Dr. Cláudio Coelho de Lima

Luca Pallozzi Lavorante
- Aluno -

Rio Claro, 28 de Outubro de 2005

Resultado: Aprovado com louvor e distinção

A Minoru e Helena Irikura,
que propiciaram minha caminhada
em direção ao mundo acadêmico

AGRADECIMENTOS

Gostaria de agradecer a todas as pessoas que contribuíram para a execução deste trabalho.

Em especial, meus sinceros agradecimentos:

- Ao Prof. Dr. Hans Dirk Ebert, por seu acompanhamento e ajuda em todas as etapas deste trabalho.
- Ao geólogo Iata Anderson de Sousa, que sanou minhas dúvidas com admirável paciência.
- A Elias Isler, Consuelo Carolina Perinotto e Gustavo Henrique Teixeira da Silva, integrantes do Laboratório de GeoModelagem 3D do DPM/IGCE/UNESP, cujo trabalho permitiu a resolução de numerosos problemas.
- Ao Programa para a Formação de Recursos Humanos para o Setor de Petróleo e Gás (PRH), conduzido pela Agência Nacional do Petróleo (ANP), pela concessão de uma bolsa de estudo.
- Ao meu pai Giuseppe, por seu incentivo e admirável capacidade de síntese.
- Finalmente a Nena, pelo companheirismo e alegria.

SUMÁRIO

Índice	ii
Resumo	iv
Abstract	v
1 Introdução	1
2 Objetivos	2
3 Métodos	2
4 Área de Estudo	42
5 <i>Tensor3D</i> : um programa para simulação de tensores em três dimensões ...	63
6 Integração das bibliotecas VTK e GTK	132
7 Modelagem 3D da paleobatimetria das bacias de Santos e Campos	139
8 Conclusões	201
9 Referências bibliográficas	205

ÍNDICE

1 Introdução	1
2 Objetivos	2
3 Métodos	2
3.1 Conceitos sobre computação gráfica e visualização científica	4
3.1.1 Conceitos de modelagem e visualização de dados	5
3.2 Introdução às ferramentas de código aberto	9
3.3 O sistema operacional Linux	10
3.4 O sistema VTK	11
3.4.1 Estrutura fundamental do sistema VTK	12
3.4.2 Definição de modelos geométricos tridimensionais	14
3.4.3 Programas compilados e programas interpretados	18
3.4.4 Estrutura de arquivos de dados do VTK	21
3.4.5 Instalação do VTK	24
3.5 Ferramentas para o desenvolvimento de programas	29
3.5.1 O arquivo <i>Makefile</i>	29
3.5.2 O ambiente de desenvolvimento <i>anjuta</i>	31
3.5.3 A biblioteca GTK	33
3.5.4 A ferramenta <i>glade</i>	35
3.6 O programa GOCAD	36
3.6.1 Interação com o programa	36
3.6.2 Objetos disponíveis no GOCAD	38
4 Área de estudo	42
4.1 Contexto regional	42
4.2 Geologia das bacias de Campos e Santos	43
5 <i>Tensor3D</i> : um programa para simulação de tensores em três dimensões	63
5.1 Teoria dos tensores	63
5.2 Notação para o tensor de tensão	64

5.3	Notação para o tensor de deformação	66
5.4	Autovalores e autovetores: eixos principais de tensão e deformação	67
5.5	O círculo de Mohr para a tensão	70
5.6	O círculo de Mohr para a deformação	73
5.7	Obtenção de elipsóides a partir de tensores	74
5.8	Considerações sobre a simetria dos tensores de tensão e deformação	76
5.9	O programa <i>Tensor3D</i>	77
5.9.1	Modelagem de <i>glyphs</i>	78
5.9.2	Descrição das funcionalidades	81
5.9.3	Estrutura do programa	96
5.9.4	Instalação	106
5.9.5	Programas auxiliares: <i>Object Factory 3D</i>	109
5.9.6	Algumas aplicações	113
6	Integração das bibliotecas VTK e GTK	132
6.1	Compilação dos módulos de trabalho	132
6.2	Gerenciamento de múltiplos ciclos de interação	133
6.3	Sincronização e comunicação entre processos de execução (<i>threads</i>)	135
7	Modelagem 3D da paleobatimetria das bacias de Santos e Campos	139
7.1	Digitalização e georreferenciamento das curvas	141
7.2	Criação de Modelos Digitais de Terreno (MDT's)	143
7.3	Modelagem de dados complementares	146
7.4	Visualização 3D dos modelos digitais obtidos	152
7.5	Modelagem de superfícies paleobatimétricas intermediárias	165
7.6	Integração com demais dados geológicos	169
7.7	Modelagem de outros dados batimétricos e paleobatimétricos	185
7.8	Discussão dos resultados	197
8	Conclusões	201
9	Referências Bibliográficas	205

RESUMO

Palavras chave: Geomodelagem, visualização 3D, Bacia de Santos, Bacia de Campos, paleobatimetria, tensor.

As pesquisas em geociências tendem cada vez mais a utilizar grande volume de dados heterogêneos, cuja interpretação integrada é complexa devido ao envolvimento de diferentes parâmetros, bem como de relações temporais e espaciais. As técnicas de computação gráfica e visualização científica assumem importância crescente por permitirem representar e manipular dados geológicos tal como são e estão no espaço, isto é, em 3D. Esta dissertação teve por objetivo utilizar ferramentas computacionais para modelar geometricamente e visualizar dados geológicos. Utilizando o programa GOCAD foram construídas superfícies paleobatimétricas das bacias de Santos e Campos para o meso-Neocretáceo a partir dos dados disponíveis na literatura. Sua integração com dados litológicos e estruturais em um único ambiente de visualização 3D permitiu aumentar o potencial de interpretação dos dados originalmente representados em mapas bidimensionais. Para melhor contextualizar a evolução paleogeográfica destas bacias durante a abertura do Atlântico Sul e representar analogias com ambientes de deposição atuais foram construídas superfícies batimétricas do Atlântico Sul do mesocretáceo ao recente e do Mar Vermelho. A partir da utilização de ferramentas de modelagem e visualização 3D de domínio público (VTK) foi desenvolvido um programa computacional (Tensor3D) para a simulação da deformação de objetos geológicos, desde rochas, estruturas tectônicas, domos de sal até bacias, a partir da modificação dos componentes de cisalhamento simples e puro contidos em tensores de deformação. A capacidade de deformação interativa, aliada à possibilidade de armazenar e ler novamente resultados de etapas intermediárias de deformação tornam o programa uma ferramenta didática ideal para estudo da deformação de corpos geológicos em 3D, além de fornecer um instrumento de auxílio ao entendimento do comportamento de estruturas geométricas complexas durante processos deformacionais.

ABSTRACT

Keywords: Geomodelling, 3D Visualization, Santos and Campos basins, paleobathymetry, tensor.

Research in Geosciences is currently using extensive volumes of heterogeneous data, whose integrated interpretation is complex due to the involvement of different parameters, as well as time and spatial relationships. Computer graphics and scientific visualization techniques are assuming increasing importance as they allow the representation and manipulation of geologic data exactly as they appear in 3D space. The purpose of this work is using computational tools in order to geometrically model and visualize geologic data. Using the GOCAD program, mid-Early Cretaceous paleobathymetric surfaces have been constructed for the Santos and Campos basins, based on published data. Their additional integration with lithologic and structural data in a unified 3D visualization environment, allowed an increase in the interpretative potential of data originally represented using 2D maps. In order to provide a more general context for the paleogeographic evolution of these basins during the opening of the South Atlantic Ocean, and to represent analogies with current depositional environments, paleobathymetric surfaces have been modelled for the South Atlantic Ocean, from mid-Early Cretaceous to present time, and for the present Red Sea. By using 3D *open-source* modelling and visualization tools (VTK), a computational program (*Tensor3D*) has been developed to simulate deformation of geologic objects, from rocks, tectonic structures, salt domes to basins. This process is controlled by modifying simple and pure shear components contained in strain tensors. The consequent real time deformation, along with the feature of storing and later retrieving geometric results from intermediate deformation steps makes the program an ideal tool for the study of geologic bodies' deformation in 3D. The program also represents a useful tool for understanding the behaviour of complex geometric structures during deformation processes.

1 Introdução

As pesquisas em geociências tendem cada vez mais a envolver a construção de modelos geométricos tridimensionais, aperfeiçoando a visualização de grande volume de dados e a interpretação dos resultados. O volume de dados tende a aumentar constantemente e o seu tratamento a ser cada vez mais sofisticado. A crescente necessidade de compreensão do comportamento espacial de corpos e parâmetros geológicos demanda a integração entre dados de superfície (como mapas geológicos bidimensionais) e de sub-superfície (perfis geológicos e geofísicos, sondagens, etc.). A modelagem 3D permite a visualização simultânea e conseqüentemente uma interpretação integrada de diferentes elementos geológicos como topografia, estruturas tectônicas, horizontes estratigráficos e parâmetros geofísicos. As principais vantagens da construção e da manipulação de modelos tridimensionais são:

- diagnosticar erros nas interpretações geométricas, freqüentemente imperceptíveis nos mapas bidimensionais;
- incrementar as capacidades de interpretação, através da:
 - Representação simultânea em um mesmo modelo de informações geométricas e parâmetros físicos ou químicos distintos;
 - Visualização de grande volume de dados;
 - Mudança no ponto de observação, no exagero vertical, no sombreado, nas cores, etc.;
 - Representação de estruturas geológicas que são tridimensionais na natureza, como camadas, falhas, dobras, etc.;
 - Simulação de processos e fenômenos, a partir da variação dos parâmetros adotados com o tempo (animação).

A disponibilidade de antigos dados paleobatimétricos, litológicos e estruturais publicados sobre as bacias de Santos e Campos, constitui importante fonte de informações para a indústria do petróleo. Sua utilização, no entanto, restringe interpretações integradas e avançadas, pois os dados encontram-se representados em mapas bidimensionais impressos e não georeferenciados. Através do seu georeferenciamento e integração em um ambiente de visualização 3D é possível aumentar seu potencial de interpretação quanto à evolução destas bacias e para a aplicação em atividades exploratórias mediante a inserção de dados de poços e perfis sísmicos.

Dentre as aplicações na área da simulação de processos, outro exemplo de complexidade que pode ser melhor caracterizada pelo potencial das ferramentas de computação gráfica e modelagem 3D é a deformação de corpos geológicos, tanto em áreas de embasamento como em coberturas sedimentares. Os regimes tectônicos, em escala regional ou de reservatório, são acomodados por mecanismos de cisalhamento simples e puro que atuam em três dimensões, o que geralmente dificulta sua compreensão por não especialistas no tema.

2 Objetivos

A presente dissertação de mestrado visa demonstrar o potencial da utilização de ferramentas de modelagem e visualização 3D para aperfeiçoar as capacidades de análise e interpretação de dados e processos geológicos mediante sua aplicação a dois estudos específicos:

- a) a reconstrução paleogeográfica e paleobatimétrica das bacias de Campos e Santos, durante o meso-Neocretáceo, a partir de dados fisiográficos, litológicos e estruturais obtidos da literatura e utilizando o programa GOCAD.
- b) a simulação da deformação de corpos geológicos, desde rochas, estruturas tectônicas, domos de sal até bacias, a partir da modificação dos componentes de cisalhamento simples e puro contidos em tensores de deformação. Para tanto foi desenvolvido um aplicativo baseado em código aberto (*open-source*) que permite a deformação interativa de modelos geométricos a partir de parâmetros definidos pelo usuário.

Este projeto constitui um exemplo do potencial de integração de softwares e ambientes computacionais de código fonte aberto com aplicativos comerciais, para suprir necessidades específicas de processamento e visualização científica de dados geológicos em ambientes de pesquisa acadêmica ou exploratória.

3 Métodos

Esta dissertação possui caráter fortemente metodológico, devido à formação do mestrando na área computacional, de forma que o estudo, aplicação e desenvolvimento das ferramentas de computação gráfica 3D constituem seu foco principal. Assim, ocorrerá uma discussão e descrição mais específica sobre estes aspectos do que sobre os temas geológicos abordados.

A aplicação de algoritmos de computação gráfica tridimensional à visualização de dados geológicos abriu novos caminhos de investigação científica aos pesquisadores, possibilitando estudar com maior atenção as relações existentes entre diferentes partes de um conjunto tridimensional de dados. Isto permite ao pesquisador a interação com seus dados, à medida que os computadores lhe fornecem representações tridimensionais das estruturas estudadas.

O processamento e visualização de dados geológicos constituem processos bastante complexos e são caracterizados por etapas intermediárias de trabalho. Embora existam ferramentas computacionais que conseguem solucionar a maioria das problemáticas associadas ao processo, é comum se realizar a integração de aplicativos distintos. Isto é consequência da inexistência de uma ferramenta única que resolva satisfatoriamente todas as necessidades de modelagem e visualização. Atividades de integração desta natureza são comuns no Laboratório de GeoModelagem 3D do DPM/IGCE/UNESP, do qual este mestrando é integrante. Neste caso, faz-se necessário escrever programas para realizar a conversão entre formatos de dados diferentes, bem como integrar aplicativos e bibliotecas especializados na realização de sub-tarefas (pré-processamento, modelagem, visualização, auxílio à interpretação), etc.

Nas seções a seguir serão ilustrados alguns conceitos de computação gráfica e visualização científica que constituem o embasamento do trabalho realizado neste projeto. Em seguida, serão descritos aplicativos para a configuração de um ambiente de trabalho 3D inteiramente “de código aberto”, desde o sistema operacional até as ferramentas de modelagem, visualização e interação com o usuário. A idéia por trás deste ambiente é disponibilizar uma ferramenta econômica de trabalho para os demais pesquisadores do grupo, além de fornecer detalhes de seu funcionamento interno para incentivar a adoção de uma metodologia semelhante durante as várias etapas de processamento e visualização de dados geológicos.

Será também ilustrado o funcionamento de uma importante ferramenta comercial, o programa GOCAD, utilizada em numerosos projetos de visualização do Laboratório de GeoModelagem 3D. Neste Mestrado, o programa foi usado para modelar e visualizar superfícies paleobatimétricas 3D para as bacias de Santos e Campos, a partir de mapas 2D obtidos da literatura.

3.1 Conceitos sobre computação gráfica e visualização científica

A computação gráfica 3D representa a base para a visualização científica. Consiste no processo de geração de imagens fotorrealísticas utilizando computadores (FOLEY et al., 1996). Para tanto, usa uma técnica denominada renderização (*rendering*), cujo objetivo é transformar dados (normalmente, informações numéricas) em primitivos gráficos, isto é, entidades geométricas representáveis em uma imagem criada por computador. A renderização é realizada levando-se em conta a interação entre algumas entidades fundamentais. Estas são referenciadas fazendo-se uma analogia com a indústria cinematográfica: luzes, câmera e atores. Existem fontes de luz utilizadas para iluminar os objetos a serem mostrados. Tais objetos são chamados atores. Atores representam a forma das entidades apresentadas, além de possuir importantes propriedades de visualização tais como cor, eventual índice de transparência, material utilizado, etc. As luzes definidas iluminam os atores, utilizando modelos matemáticos que simulam a iluminação real. Por fim, uma câmera virtual constrói uma imagem projetando os atores em um plano 2D que será exibido na tela do computador. Esta combinação de luzes, câmera e atores é denominada *cena* (2D ou 3D), e constitui o resultado final do processo de renderização.

A visualização científica é o processo de transformar dados científicos (advindos de observação, simulação numérica ou outras fontes) em primitivos gráficos a serem representados em duas ou três dimensões através de técnicas de computação gráfica. A idéia central do processo consiste em mapear informações numéricas de teor científico em entidades gráficas, explorando a facilidade de interação com objetos gráficos própria do ser humano. Por exemplo, informações topográficas de cotas referentes a uma determinada região são de difícil interpretação quando apresentadas sob a forma de tabela numérica (seu formato original). Tal dificuldade aumenta em proporção ao crescimento do volume de dados, e a interpretação se torna praticamente impossível quando várias ordens de informações são consideradas (por exemplo, medidas de grandezas distintas associadas a pontos do espaço).

Esta complexidade, contudo, é reduzida utilizando gráficos 2D ou 3D para exibir os pontos do conjunto de dados. Isto proporciona uma representação visual cujas características mais importantes são imediatamente aferidas pelo observador. Além disto, uma representação gráfica permite realçar determinadas propriedades dos dados como um todo (por exemplo, uma tendência geral do conjunto). Técnicas de computação gráficas, como a utilização de escalas de cores para mapear a variação de valores de uma determinada variável, proporcionam meios para entender rapidamente qual é o comportamento da mesma na região de estudo.

Uma importante vantagem da computação gráfica 3D consiste na possibilidade de *interagir* com os objetos visualizados, modificando o ponto de observação e simulando rotações no espaço para determinar as diversas características espaciais das entidades estudadas. Este caráter interativo do processo representa uma exigência *fundamental* da visualização científica, pois muitas vezes norteia a exploração e interpretação dos dados originais.

3.1.1 Conceitos de modelagem e visualização de dados

A técnica mais difundida de representação de dados geológicos tridimensionais baseia-se na construção de superfícies sombreadas que delimitam corpos geológicos (contatos) ou que representam planos tectônicos (falhas) (PFLUG & HARBAUGH, 1992). Tal técnica baseia-se na definição de objetos através de um conjunto de pontos tridimensionais, isto é, uma lista de coordenadas do tipo (x, y, z). A representação geométrica de uma entidade observada na natureza (como uma superfície topográfica), constitui, na verdade, uma *modelagem computacional*, isto é, uma discretização do mundo real (que é contínuo), simplificando-se os objetos estudados através de entidades finitas (isto é, constituídas de um número finito de pontos geométricos). Isto decorre da natureza discreta dos computadores, que possuem limites finitos quanto à sua capacidade de processamento e armazenamento de informações, e que, portanto, trabalham com aproximações numéricas da realidade. Apesar desta limitação, os modelos matemático-geométricos construídos constituem uma poderosa ferramenta de estudo. Tais modelos apresentam também a vantagem implícita de realçar as características mais importantes do objeto para concentrar a investigação apenas sobre os aspectos de interesse.

Após a definição do conjunto de pontos dos objetos estudados, constrói-se uma malha de polígonos (triângulos ou quadriláteros), denominada *malha vazada*, para identificar a geometria das superfícies geológicas sendo investigadas (por exemplo, falhas ou relevo topográfico). Tal malha é definida especificando de que maneira os pontos geométricos deverão ser interconectados. Um exemplo de malha vazada usada para modelar uma falha geológica é mostrado na Figura 3.1. A técnica de visualização do conjunto de arestas interconectadas é denominada *Wireframe* (FOLEY et al., 1996).

A técnica anterior realça apenas a geometria do objeto modelado. Utilizando recursos de computação gráfica é possível aplicar cores ao objeto, efetuando o seu sombreamento (*Shading*). As cores são aplicadas em cada polígono da entidade. De modo geral, a

intensidade de cor utilizada para cada polígono é o resultado da interação entre uma (ou mais) fonte(s) de iluminação predefinida(s) e a orientação da face em relação à luz. Isto é, polígonos aproximadamente normais à direção da fonte de luz recebem uma tonalidade máxima de cor (cor mais clara), ao passo que polígonos aproximadamente paralelos à direção da fonte de luz aparecem mais escuros. Esta técnica é denominada *Flat Shading* (Figura 3.2).

Por fim, é possível amenizar a aparência discreta do objeto visualizado. Isto poderia ser feito de várias maneiras. A primeira consiste em se utilizar um maior número de pontos para suavizar as arestas do modelo. Isto implica uma nova digitalização das informações originais, procedimento nem sempre possível ou dispendioso. Alternativamente, pontos intermediários podem ser obtidos interpolando as informações disponíveis. Existem numerosos algoritmos de interpolação linear e curvilínea para tanto. A desvantagem deste processo é a adição de pontos cujas características (posição e / ou valor da variável associada) podem ser diferentes das encontradas na realidade, ocasionando problemas de fidelidade dos modelos utilizados. Evidentemente, os algoritmos de interpolação assumem, ao adicionar novos pontos, a continuidade das propriedades geométricas ou da distribuição de valores.

Uma terceira solução consiste na utilização de um artifício gráfico de sombreamento (*Gouraud Shading*, FOLEY et al., 1996). O número de pontos do modelo não é aumentado, mas os polígonos de sua representação 3D são sombreados utilizando alguma forma de interpolação cromática entre as cores determinadas para seus vértices. A diferença fundamental em relação à técnica anterior está no número de cores associadas a cada polígono. No caso da modalidade *Flat Shading*, considera-se uma única cor, cuja tonalidade depende da orientação da face em relação à direção da fonte de luz. Já no sombreamento de tipo *Gouraud*, considera-se uma tonalidade diferente da cor-base para cada vértice do polígono (por exemplo, três tonalidades para um triângulo), sendo que esta informação é determinada considerando-se a orientação dos polígonos adjacentes em relação à direção da fonte de luz. Finalmente, será criada uma interpolação de cores entre as tonalidades associadas aos vértices. Tal interpolação será utilizada para sombrear o polígono em questão (Figura 3.3). É importante ressaltar que a técnica de *Gouraud Shading* não acrescenta nenhuma informação geométrica ao modelo original. Contudo, a interpolação cromática transmite a ilusão de uma maior definição dos dados, proporcionando uma representação aceitável sem comprometer a eficiência da visualização (pois o número de polígonos utilizados continua o mesmo).

Existem muitas técnicas de visualização científica e computação gráfica utilizadas para realçar determinadas características dos dados visualizados, bem como auxiliar na sua

interpretação (utilização de cores, *glyphs*, etc.). Exemplos destas técnicas serão fornecidos na seção destinada aos resultados deste trabalho.

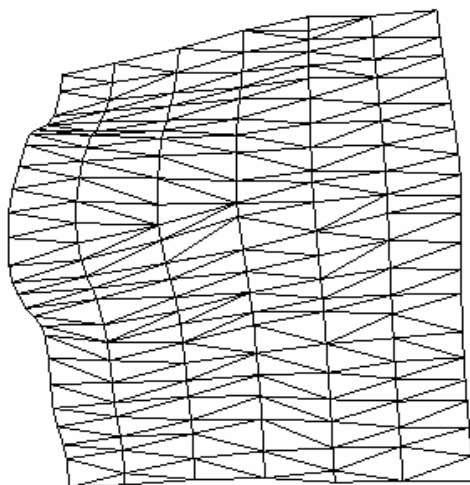


Fig. 3.1 – Visualização 3D de uma falha geológica em modalidade *Wireframe*.



Fig. 3.2 – Representação da mesma superfície em modalidade *Flat Shading*. Cada triângulo recebe uma única cor, cuja intensidade é controlada pela orientação do mesmo em relação à fonte de luz.

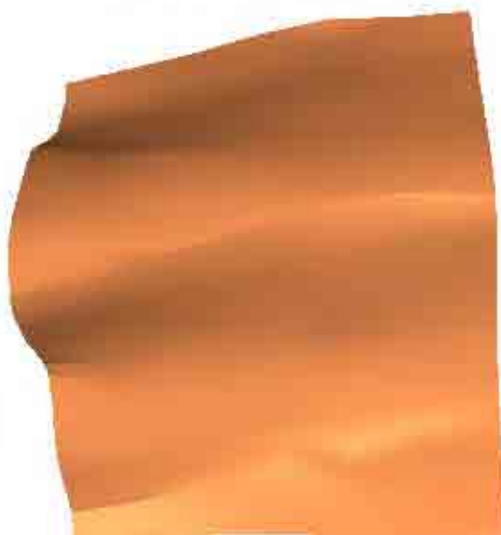


Fig. 3.3 – Modalidade de sombreamento *Gouraud Shading*. Cada triângulo é sombreado de acordo com uma interpolação cromática entre as tonalidades calculadas para seus vértices.

Além de dados constituídos de pontos esparsos conectados através de polígonos, existem conjuntos de informações criados usando uma ou mais variáveis medidas ou calculadas em pontos *regularmente espaçados* ao longo de um plano (*grid*) ou de uma região tridimensional (*grids* empilhados, ou volume). Estes dados são representados em 3D utilizando técnicas de *visualização volumétrica* (SCHROEDER et al., 1996). Como esta categoria de dados contém mais informações do que malhas vazadas, sua visualização é realizada através da criação de uma nuvem semitransparente. Isto permite mostrar as informações contidas em qualquer parte do conjunto de dados volumétricos.

Uma aplicação de visualização volumétrica de dados geológicos é mostrada em Lavorante et al. (2000) (Figura 3.4).

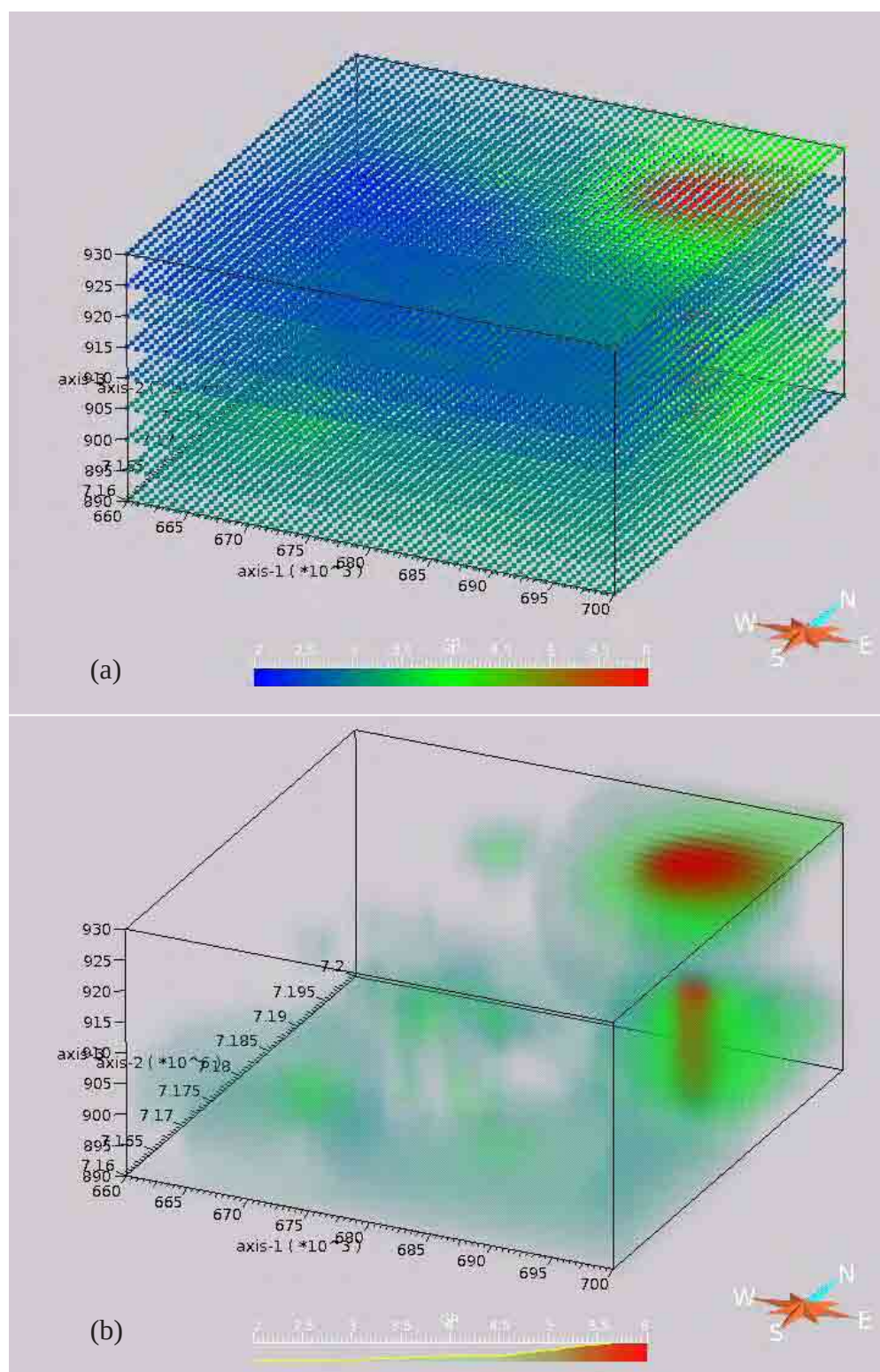


Figura 3.4 – Visualização volumétrica da granulometria do preenchimento da Bacia de Curitiba usando o programa GOCAD. A variável assume valores de 2 a 6 (argilitos a conglomerados, respectivamente). O volume foi obtido sobrepondo *grids* horizontais gerados a partir da interpolação de dados de poços tubulares profundos. (a) Visualização dos dados através de um conjunto de pontos (sombreados usando a escala de cores da região inferior), para destacar o caráter regular da distribuição de valores. (b) Criação de uma nuvem semitransparente usando planos de corte aproximadamente perpendiculares à direção de observação. A visualização das estruturas internas é possível especificando-se uma função de transparência associada ao intervalo de valores considerado (estreitamento da escala de cores da porção inferior).

3.2 Introdução às ferramentas de código aberto

Uma das linhas de trabalho que este mestrando seguiu utiliza ferramentas de código aberto (*Open Source*), desde o sistema operacional até os programas responsáveis pelo desenvolvimento de aplicativos. Tal filosofia baseia-se em uma licença de software denominada GNU (acrônimo de *GNU's not UNIX*, uma fórmula utilizada para diferenciar o caráter aberto do sistema operacional Linux em relação aos sistemas UNIX tradicionais, a pagamento) (GNU, 2005). De acordo com esta licença, o código fonte dos aplicativos disponibilizados pode ser divulgado. Não existem, portanto, patentes aplicadas aos programas sob a licença GNU. Uma vantagem em se utilizar ferramentas de software livre consiste na possibilidade de divulgação do aplicativo resultante com a mesma licença, proporcionando aos demais desenvolvedores a possibilidade de acessar o código fonte do programa. Desta forma, podem surgir novas idéias de trabalho ou um projeto preexistente pode ser melhorado em virtude da contribuição de vários desenvolvedores. A liberdade associada ao uso do código implica a possibilidade de:

- Executar um programa, para qualquer motivo.
 - Estudar como o programa funciona e adaptá-lo a necessidades específicas. O acesso ao código fonte é uma condição imprescindível para isto.
 - Distribuir cópias de forma a ajudar outros usuários.
 - Modificar o programa e disponibilizar eventuais melhorias para a comunidade.
- Novamente, o acesso ao código-fonte é fundamental.

Um programa é considerado software livre se oferecer estes quatro requisitos.

Dentre as vantagens do software livre sobre o software comercial, destacam-se:

- a criação de grupos de trabalho descentralizados que representam importantes exemplos de desenvolvimento cooperativo, já que o código fonte pode ser livremente visionado, alterado e disponibilizado.
- a velocidade de detecção e correção de *bugs* (erros de programação), ressaltando-se que nenhum grupo de trabalho centralizado possui a mesma agilidade encontrada em comunidades espacialmente distribuídas.
- a difusão de técnicas de resolução de problemas comerciais e científicos com a consequência de espalhar o conhecimento.

A ideologia de software livre é muito diferente da ótica tradicional de desenvolvimento de software comercial, o qual é normalmente fechado, no sentido de que a especificação e construção de seus módulos internos permanecem prerrogativa exclusiva de

um núcleo protegido (empresa, centro de pesquisa). Esta característica tem sua justificativa no esforço intelectual e computacional envolvido no desenvolvimento de um software, levando à necessidade de proteção dos recursos utilizados. Além disto, é necessário se manter uma equipe de trabalho para desenvolvimento e correção de erros no aplicativo, e recursos humanos e computacionais possuem um custo de utilização. Desta forma, este custo é repassado para o usuário final.

A adoção de idéias inevitavelmente contrastantes quanto às duas filosofias de desenvolvimento tem gerado inúmeras controvérsias, muitas vezes em detrimento do próprio usuário final. Na prática procura-se ferramentas econômicas para a resolução eficiente de problemas e a atitude do centro de pesquisa neste Laboratório de GeoModelagem 3D tem sido a de integrar ferramentas *open source*, normalmente de aplicação geral, com programas comerciais desenvolvidos para finalidades específicas. Um exemplo do primeiro tipo de ferramenta é o sistema VTK (VISUALIZATION TOOLKIT, 2005), que possui numerosos módulos para modelagem e visualização 3D, mas que, em força de sua generalidade, necessita de adaptações e configurações sob a forma de desenvolvimento de novos módulos, modificação de módulos preexistentes e escrita de programas para a utilização de tais módulos ou conversão de dados em formatos diferentes. Como se vê, trata-se de tarefas que envolvem um certo esforço. Por outro lado, como ferramenta comercial destaca-se o GOCAD (2005), especialmente desenvolvido para modelar e visualizar dados geológicos. Este programa possui também módulos que auxiliam na interpretação dos dados. O nível de excelência que caracteriza tal software, bem como a redução do referido esforço de trabalho, justificam o pagamento de uma licença de uso. Mas é muitas vezes conveniente realizar uma integração entre os dois ambientes (do software livre e do software comercial), já que podem surgir necessidades de modelagem específicas que nem sempre uma única ferramenta consegue preencher.

3.3 O Sistema Operacional Linux

Linux (2005) é um sistema operacional desenvolvido por Linus Torvalds com a ajuda de numerosos colaboradores voluntários localizados em diferentes países. Seu objetivo é proporcionar uma ferramenta de trabalho baseada em aplicativos de código fonte aberto. Foi criado nos moldes dos sistemas operacionais UNIX comerciais (Solaris, AIX, Irix, HPUX, etc.), mas com a intenção de funcionar em computadores com processador Intel de 32 bits (PC's). Atualmente, o sistema pode ser utilizado em outras plataformas de hardware, como a

arquitetura a 64 bits das empresas Intel e AMD, bem como equipamentos SUN, entre outros. Múltiplas instâncias do sistema Linux são também utilizadas em computadores de grande porte da empresa IBM (*mainframes*).

Todos os módulos que constituem este sistema possuem código fonte aberto. O nome Linux, que antes indicava apenas o programa central (*core*, ou *kernel*) (*LINUX KERNEL ARCHIVES*, 2005), responsável pelo gerenciamento de outros programas de usuário, hoje denota um conjunto de aplicativos (ferramentas de Internet, escritório, processamento gráfico, computação científica, etc.) que são disponibilizados em coletâneas chamadas distribuições. As distribuições mais famosas são a da empresa americana *Red Hat* (*RED HAT ENTERPRISE LINUX*, 2005; *FEDORA CORE*, 2005), o *Debian* (2005), o *FreeBSD* (2005), o *Mandrake* (*MANDRAKELINUX*, 2005), o *Conectiva* (2005), etc. A maioria destas distribuições têm em comum o custo nulo de aquisição do software. Cada coletânea contém um certo número de aplicativos de trabalho (Internet, editor de texto, etc.) também de código aberto e gratuitos. Apesar de existir compatibilidade em termos de código binário (programas executáveis) entre as várias distribuições, cada uma possui diferentes aplicativos de configuração e gerenciamento do sistema, o que torna o processo de migração de uma distribuição para outra um tanto trabalhoso. Isto também determina a tendência de um usuário a se especializar na configuração e uso de uma determinada distribuição.

O sistema Linux utilizado neste projeto é denominado *Fedora Core 3* (2005), disponibilizado pela empresa Red Hat e atualmente em sua quarta versão. Sua adoção por parte do Laboratório de GeoModelagem 3D deriva de uma certa tradição na utilização de distribuições da Red Hat, o que fez os pesquisadores deste grupo se acostumarem com suas ferramentas de gerenciamento. A empresa Red Hat possui também uma versão comercial, denominada *Enterprise Linux*, que contém módulos adicionais de segurança e gerenciamento de servidor desenvolvidos internamente. Tanto o *Fedora Core* como o *Enterprise Linux* estão disponíveis para plataformas Intel (ou AMD) de 32 e 64 bits.

3.4 O sistema VTK

O nome VTK (*VISUALIZATION TOOLKIT*, 2005; SCHROEDER et al., 1996) denota um conjunto de algoritmos de código aberto que implementam as principais técnicas de processamento de imagens e visualização científica atualmente utilizadas em ambientes de pesquisa. Estes algoritmos foram desenvolvidos a partir da biblioteca gráfica Open-GL (2005), que por sua vez implementa as principais técnicas de computação gráfica 3D. De maneira geral, pode-se classificar o VTK como uma biblioteca gráfica mais abstrata que

Open-GL, já que o primeiro utiliza as funções de computação gráfica da segunda. Trata-se, na verdade, de um ambiente de desenvolvimento que permite a rápida visualização tridimensional e manipulação de superfícies poligonais e dados volumétricos vindos de diferentes áreas de pesquisa, como a medicina, engenharia, meteorologia, geologia, etc. A versão atual do VTK é a 4.2.

Este sistema foi desenvolvido em C++ e utiliza as principais propriedades dos ambientes de programação orientados a objetos para definir e processar os dados a serem visualizados. Isto é, as várias funcionalidades disponíveis no sistema estão codificadas em módulos distintos mas com grande capacidade de comunicação e de transferência de dados. É também possível utilizar o VTK de maneira interativa, através de uma linguagem interpretada no lugar de C++. Isto permite ao usuário a rápida criação, modificação e visualização de objetos tridimensionais conhecendo apenas algumas funcionalidades do sistema. Desta forma, os conceitos de programação avançada são reservados para aplicações mais complexas.

A representação tridimensional constitui a base do sistema de visualização do VTK, sendo que os principais algoritmos de criação e elaboração de primitivos geométricos estão englobados em procedimentos (métodos) de fácil utilização. Deste modo, estruturas complexas podem ser visualizadas com poucas linhas de código auto-explicativas.

3.4.1 Estrutura fundamental do sistema VTK

Pode-se imaginar o VTK como um sistema de fluxos de dados que transitam entre várias entidades de processamento, desde a elaboração dos vértices que constituem um objeto geométrico, até sua representação numa janela gráfica interativa, sendo possível modificar a posição do ponto de observação e a distância do objeto em qualquer instante. Cada entidade é responsável pela execução de uma diferente etapa no processo de visualização. A estrutura orientada a objetos do VTK permite que diferentes tipos de dados sejam processados pelas mesmas classes genéricas de objetos, cujas instâncias específicas operam nos vários casos.

De acordo com a nomenclatura da programação orientada a objetos, a palavra *classe* denota um conjunto de informações (*atributos*) e de operações que podem ser executadas sobre as mesmas (*métodos*). A palavra *instância* indica a criação de um objeto específico pertencente a determinada classe. Considerando uma aplicação de modelagem não gráfica, a classe *avião* engloba o conjunto de atributos (nome, modelo, peso, velocidade máxima, número e tipo de turbinas, número de assentos, posição atual, etc.) e métodos (decolar, pousar, virar, ajustar potência de determinada turbina, etc.) que uma aeronave em geral deve

ter. Já uma instância desta classe representa determinada aeronave específica. Assim, na modelagem de uma frota aérea, determinada companhia pode possuir, por exemplo, cinco aeronaves (instâncias) de tipo DC-10 (classe). No domínio do sistema VTK, uma classe denota um conjunto de atributos e métodos para determinado objeto gráfico ou processo de manipulação de dados, ao passo que uma instância da classe corresponde a uma cópia de tais informações explicitamente criada pelo programador e armazenada na memória do computador. Por exemplo, dez representações poligonais de esferas no espaço 3D correspondem a múltiplas instâncias de uma mesma classe (*vtkSphereSource*), cada qual com sua definição de posição, raio, número de meridianos, número de paralelos, etc. (atributos). Para cada instância, é possível invocar (executar) métodos para modificar os atributos correspondentes (posição e raio da esfera, etc.).

Todas as classes utilizadas no VTK possuem nomes longos e auto-explicativos e são introduzidas pela sigla “vtk” (*vtkPolyData*, *vtkRenderWindow*, etc). Existem algumas classes fundamentais de objetos cujas instâncias executam todo o processo de visualização:

- *vtkLight*: responsável pela definição de todas as propriedades das luzes aplicadas aos objetos geométricos.
- *vtkCamera*: especifica a maneira pela qual uma câmara virtual deverá visualizar os objetos. Inclui parâmetros importantes, como a posição da câmera, a direção de observação, o tipo de projeção (paralela ou perspectiva), etc.
- *vtkMapper*: especifica as propriedades geométricas dos dados, bem como informações adicionais (atributos) associadas aos pontos. Instâncias desta classe convertem informações poligonais para primitivos gráficos usados pelas bibliotecas de processamento gráfico do sistema computacional (OPEN-GL).
- *vtkActor*: constitui um objeto representado em três dimensões. São também definidas propriedades de iluminação, cor e transparência da superfície do objeto.
- *vtkRenderer*: coordena o processo de visualização envolvendo luzes, propriedades geométricas dos objetos, posição da câmera, etc. O resultado é uma imagem 2D (cena).
- *vtkRenderWindow*: visualiza e atualiza a janela gráfica que exhibe as informações tridimensionais geradas pela classe anterior.

Após a visualização tridimensional de um ou mais objetos, o usuário pode interagir com os dados rotacionando, transladando e variando a distância entre as entidades representadas e o ponto de observação. Para tanto utiliza-se o mouse e o teclado. Esta interação é obtida associando uma instância da classe *vtkRenderWindowInteractor* à janela de visualização. Algumas teclas são utilizadas para modificar as propriedades dos objetos

definidos. Por exemplo, a tecla ‘w’ estabelece uma representação em malha vazada (*Wireframe*, Figura 3.1) para cada objeto na janela de visualização, enquanto ‘s’ determina o uso de superfícies sombreadas (*Flat Shading* ou *Gouraud Shading*, Figuras 3.2 e 3.3, respectivamente). Quando o ponteiro do mouse estiver posicionado sobre algum objeto na janela, a tecla ‘p’ permite efetuar uma seleção individual.

Outra característica relevante da classe *vtkRenderWindowInteractor* é a possibilidade de executar interativamente comandos para criar ou manipular classes durante a visualização, modificando assim as propriedades dos objetos representados. Isto ocorre apertando a tecla ‘u’ durante a visualização, o que mostrará uma janela pronta para aceitar comandos.

Com relação ao movimento dos fluxos de dados no sistema, existem três categorias de classes disponíveis:

- As *fontes* são responsáveis pela definição geométrica dos dados a serem visualizados. Podem ser constituídas por informações criadas matematicamente (cubo, esfera, cone, etc.), ou programas que leiam dados do disco ou da rede. Evidentemente, as fontes apresentam unicamente saídas de dados.
- Os *filtros* elaboram dados preexistentes, gerando saídas de natureza variada. Um exemplo de filtro é dado pela classe *vtkDelaunay2D*, que constrói uma malha de triângulos a partir de um conjunto de pontos de entrada, usando o algoritmo de triangulação Delaunay (WATSON, 1981).
- Os *destinos* recebem mas não criam dados no sistema. Representam as etapas terminais do processamento. Pode-se considerar o exemplo dos *writers* e *exporters*, responsáveis, respectivamente, pela gravação em disco das definições geométricas de um objeto, ou pela gravação do inteiro contexto de visualização, incluindo ponto de observação, fontes de iluminação, etc. Os *exporters* são de grande utilidade para a criação de *scenes* (cenas ou contextos prontos), que podem literalmente ser exportados para outros sistemas de visualização, incluindo os navegadores utilizados para acessar a Internet. Outro exemplo importante de classe do tipo destino é *vtkRenderWindow*, que encerra um fluxo de dados advindo de outras classes através da criação de uma janela de visualização 3D.

3.4.2 Definição de modelos geométricos tridimensionais

Lembrando que a base para a representação geométrica de um objeto é o conjunto de pontos tridimensionais que o constituem, os sistemas de visualização destinam procedimentos especiais à leitura e armazenamento destes pontos. A especificação do conjunto de vértices de

um objeto define genericamente a sua *geometria*. No caso de um cubo, por exemplo, sua geometria é dada pela especificação das coordenadas de oito pontos no espaço 3D.

Uma outra característica indispensável de um objeto é sua *topologia*. É necessário fornecer, além do conjunto de pontos, regras para se estabelecer de que maneira estes pontos estão relacionados (conectados). No exemplo do cubo, sua topologia é especificada assumindo-se seis primitivos geométricos (os quadriláteros que denotam as faces). Para cada primitivo deverá ser fornecido o conjunto de pontos que o constituem. Isto ocorre atribuindo-se aos pontos dos objetos números de identificação (*ids*), referenciados em todos os processos envolvendo topologia.

No sistema VTK, a classe *vtkPolyData* é responsável pela definição de modelos geométricos constituídos de polígonos (Figura 3.5).

A geometria é especificada através de uma lista de coordenadas (x, y, z) de pontos definindo os vértices do objeto. Cada ponto possui um código ou número de identificação (*id*) usado para referenciá-lo na estrutura da superfície (tabela de geometria). A topologia é a lista de todos os polígonos que são construídos usando os pontos referenciados na estrutura geométrica. Cada polígono é definido por uma lista de códigos de pontos especificando os vértices do mesmo, e também possui um código de identificação (*id*) na tabela de topologia.

A geometria de um objeto do tipo *vtkPolyData* é construída usando instâncias da classe *vtkPoints*, que fornece eficientes procedimentos (métodos) para a inserção e remoção de pontos tridimensionais. Para definir a topologia de um objeto utiliza-se a classe *vtkCellArray*. A classe *vtkPolyData* possui métodos para obter informações geométricas e topológicas sobre sua estrutura. Por exemplo, *GetNumberOfCells()* e *GetNumberOfPoints()* informam, respectivamente, o número de primitivos (polígonos) e o número de pontos na correspondente tabela de geometria.

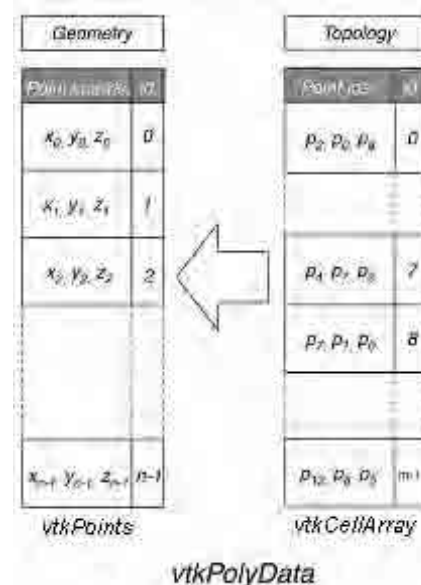


Fig. 3.5 – Estrutura da classe *vtkPolyData* usada para construir superfícies poligonais no sistema VTK. Neste exemplo, o modelo possui n pontos e m triângulos.

No sistema VTK os primitivos possuem dimensão de 0 a 3 e são os seguintes (Figura 3.6):

- *VTK_VERTEX*. É um ponto no espaço.
- *VTK_POLY_VERTEX*. Conjunto de pontos sem informação topológica.
- *VTK_LINE*. Segmento de linha definido por dois pontos.
- *VTK_POLY_LINE*. Linha poligonal, resultado da junção de segmentos adjacentes.
- *VTK_TRIANGLE*. Polígono constituído por três pontos.
- *VTK_TRIANGLE_STRIP*. Uma faixa de triângulos adjacentes. O mecanismo de armazenamento é otimizado de modo que para a especificação de n triângulos são necessários apenas $n+2$ pontos, ao em vez de $3n$. Além disto, a especificação de modelos geométricos através de faixas de triângulos permite uma visualização otimizada, pois as atuais placas gráficas aceleradoras possuem recursos para processar faixas de triângulos de forma eficiente.
- *VTK_POLYGON*. Polígono genérico constituído de três ou mais pontos coplanares.
- *VTK_PIXEL*. Quadrilátero em que segmentos adjacentes formam ângulos de 90 graus entre si (quadrado ou retângulo). Seus lados são paralelos a dois eixos coordenados.
- *VTK_QUAD*. Polígono constituído por quatro pontos, sendo que os ângulos entre segmentos adjacentes podem ser diferentes de 90 graus.
- *VTK_TETRA*. Primitivo tridimensional formado por quatro pontos não coplanares.
- *VTK_VOXEL*. Paralelepípedo alinhado com os eixos coordenados. Neste primitivo o ângulo entre segmentos adjacentes é sempre de 90 graus.
- *VTK_HEXAHEDRON*. Primitivo formado por seis quadriláteros. É um paralelepípedo em que as faces não mantêm necessariamente perpendiculares entre si.
- *VTK_WEDGE*. Prisma com base triangular.
- *VTK_PYRAMID*. Pirâmide cuja base é um quadrilátero com ângulos internos que podem diferir de 90 graus.

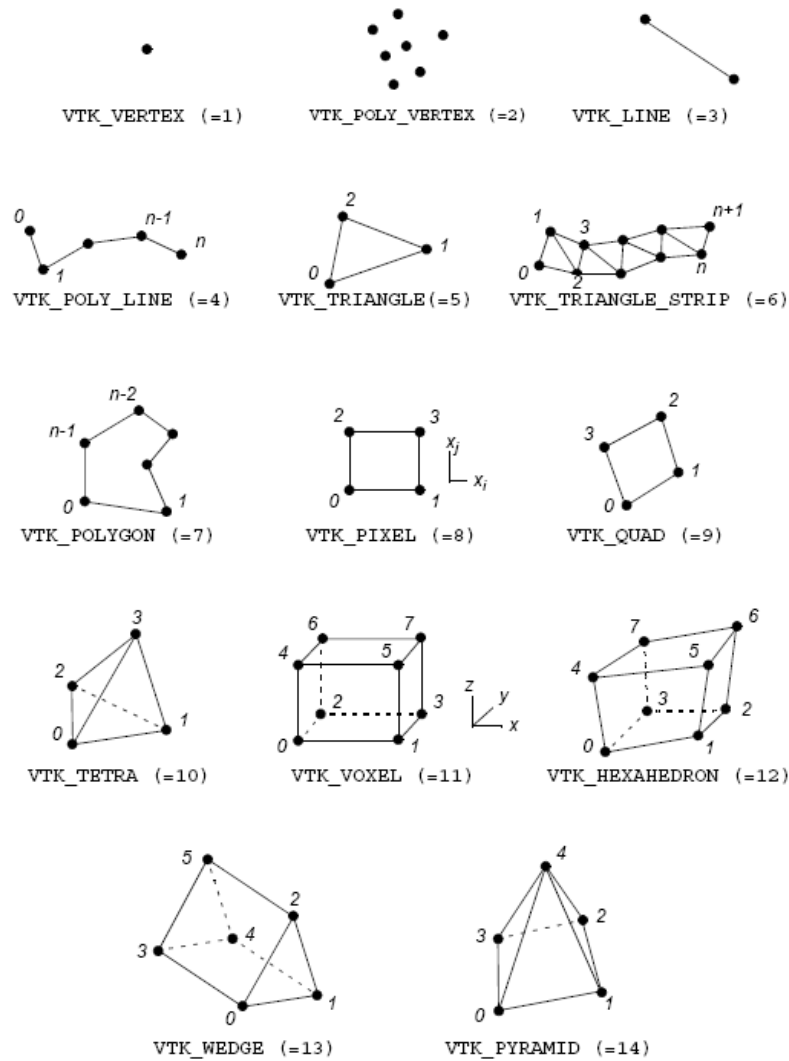


Figura 3.6 – Primitivos gráficos de tipo linear do sistema VTK. Os números ao lado dos pontos determinam a correta ordem de sua referência para efeitos de interação com as fontes de iluminação. Pontos, linhas, polígonos e sólidos possuem dimensão 0, 1, 2 e 3, respectivamente. Primitivos com dimensão de 0 a 2 são armazenados na classe *vtkPolyData*. Para primitivos de dimensão 0 a 3 utiliza-se a classe *vtkUnstructuredGrid*. Fonte: Visualization ToolKit (2005).

Estes primitivos são denominados *lineares*, pois seus pontos são conectados por segmentos de linhas. O VTK possui também primitivos *não-lineares*, cujos pontos são conectados por arcos de curvas. Tais primitivos são utilizados para aplicações mais complexas recentemente implementadas na biblioteca gráfica.

Como foi ilustrado, geometria e topologia representam os dois aspectos fundamentais para a definição de um objeto tridimensional. Estas duas informações constituem o modelo a ser visualizado. Considerando que em Visualização Científica aos pontos no espaço podem ser associadas várias ordens de informação (temperatura, porosidade, velocidade, etc.), surge a necessidade de se acrescentar uma outra classe ao modelo. Trata-se dos *atributos* dos pontos ou dos primitivos gráficos, isto é o conjunto de informações associadas aos pontos ou aos

primitivos da entidade geométrica (a palavra “atributo” é usada aqui com um significado mais abrangente do que no contexto da programação orientação a objetos). As classes usadas para tanto são *vtkPointData* (para pontos) e *vtkCellData* (para primitivos gráficos). A isosuperfície (ou superfície equipotencial) é um exemplo de entidade geométrica cujos pontos possuem a mesma informação (mesmo valor do atributo). Em uma superfície topográfica, o atributo de cada ponto é constituído pela própria coordenada Z (elevação), o que, do ponto de vista do VTK, implica a ausência de atributos (nenhuma instância de *vtkPointData* ou *vtkCellData*).

Existem diferentes tipos de atributos. A cada ponto pode-se associar um valor escalar, um vetor, um vetor unitário (também denominado normal) ou uma matriz 3x3 de escalares (tensor).

3.4.3 Programas compilados e programas interpretados

Como descrito acima, os módulos executáveis que constituem o VTK foram escritos na linguagem C++ (STROUSTRUP, 2000). Isto levou à necessidade de *compilar* tais módulos, ou seja, traduzir o código fonte da linguagem de programação usada para instruções executáveis pelo processador do computador (código de máquina). Efetuar a compilação de uma classe inteira de procedimentos permite otimizar o código executável e proporciona o máximo desempenho possível. Por outro lado, um programa compilado não permite ao usuário a modificação em tempo real do conjunto de instruções e informações geradas. Para isto, é necessário modificar o código fonte e realizar uma *nova compilação*. Além disto, o código compilado para determinado processador não pode ser utilizado em outro, sendo necessária uma compilação para cada plataforma computacional. Para minimizar tal inconveniente, o sistema VTK foi estruturado de maneira a permitir a execução de seus módulos (funções de biblioteca) dentro de programas interpretados. Programas deste tipo diferem dos compilados por serem traduzidos e executados *passo a passo*, isto é, um comando de cada vez. Programas interpretados são às vezes chamados *scripts*. Tal palavra realça a necessidade de traduzir cada instrução para código de máquina *em tempo de execução*. A linguagem interpretada utilizada pelo VTK é chamada *Tcl/Tk* (2005).

Uma importante vantagem das linguagens interpretadas é sua extrema flexibilidade em tempo de execução, já que o usuário pode, depois de iniciada uma sessão de trabalho, digitar e executar interativamente comandos, observando o resultado de cada execução parcial e podendo modificar os comandos utilizados em qualquer momento. No caso da linguagem *Tcl/Tk*, é possível modificar em tempo de execução as propriedades dos objetos visualizados, acrescentar novos dados, criar representações combinadas, etc.

<pre>// Criacao de um cubo em C++ #include "vtkRenderer.h" #include "vtkRenderWindow.h" #include "vtkCubeSource.h" #include "vtkPolyDataMapper.h" #include "vtkActor.h" main () { // Modelagem geometrica vtkCubeSource *cubeSrc = vtkCubeSource::New(); vtkPolyDataMapper *cubeMpr = vtkPolyDataMapper::New(); cubeMpr->SetInput(cubeSrc->GetOutput()); // Visualizacao vtkActor *cube1 = vtkActor::New(); cube1->SetMapper(cubeMpr); vtkRenderer *ren1 = vtkRenderer::New(); ren1->AddActor(cube1); vtkRenderWindow *renWin = vtkRenderWindow::New(); renWin->AddRenderer (ren1); renWin->Render (); cubeSrc->Delete(); cubeMpr->Delete(); cube1->Delete(); ren1->Delete(); renWin->Delete(); } </pre> (a)	<pre># Criacao de um cubo em Tcl/Tk package require vtk package require vtkinteraction # Modelagem geometrica vtkCubeSource cubeSrc vtkPolyDataMapper cubeMpr cubeMpr SetInput [cubeSrc GetOutput] # Visualizacao vtkActor cube1 cube1 SetMapper cubeMpr vtkRenderer ren1 ren1 AddActor cube1 vtkRenderWindow renWin renWin AddRenderer ren1 renWin Render wm withdraw . </pre> (b) <pre> graph TD A[vtkCubeSource] --> B[vtkPolyDataMapper] B --> C[vtkActor] C --> D[vtkRenderer] D --> E[vtkRenderWindow] style A fill:#d3d3d3 style B fill:#ffffff style C fill:#ffffff style D fill:#ffffff style E fill:#d3d3d3 </pre> (c)
--	---

Figura 3.7 – Exemplo de código fonte para a modelagem e visualização de um cubo usando o sistema VTK. (a) Programa em C++. (b) Programa em Tcl/Tk. A segunda linguagem provê um código mais enxuto e que não requer compilação prévia. (c) Diagrama de fluxo de dados entre as classes usadas no programa. Classes com fundo cinza representam a origem ou destino final dos dados.

Uma outra característica interessante das linguagens interpretadas é que um mesmo código fonte não precisa ser explicitamente compilado pelo programador. Tal processo ocorre, obviamente, de forma implícita quando da interpretação do programa, havendo um interpretador para cada plataforma. Contudo, do ponto de vista do programador, o mesmo código pode ser executado em plataformas e sistemas operacionais distintos sem sofrer modificações.

A linguagem Tcl/Tk é composta de dois módulos. O primeiro (Tcl), denota a linguagem propriamente dita: instruções que controlam o fluxo de execução do programa, comandos para a leitura/escrita de arquivos do disco, etc. O segundo módulo (Tk) constitui

uma extensão da linguagem e implementa numerosas componentes de interação gráfica com o usuário através de botões, barras de rolagem e outros objetos usados nos atuais sistemas operacionais. O programa *Tensor3D*, discutido mais adiante, faz amplo uso das componentes gráficas do módulo Tk.

Para evidenciar a facilidade de uso da linguagem Tcl/Tk, será mostrado um programa para a visualização de um cubo (Figura 3.7), tanto em C++ como em Tcl/Tk, lembrando que o primeiro código precisa ser compilado (vide seção 3.5.1), enquanto o segundo pode ser diretamente digitado e executado (usando o interpretador de comandos *vtk*) no sistema de desenvolvimento interativo do VTK.

Os dois programas apresentados utilizam os mesmos objetos do sistema VTK. Segue descrição do programa em C++ (Figura 3.7a). O início do processo de modelagem ocorre pela criação da instância *cubeSrc* da classe *vtkCubeSource* (método *New()*), que define as características geométricas de um cubo posicionado na origem do sistema cartesiano de coordenadas. A instância *cubeMpr* da classe *vtkPolyDataMapper* constitui o segundo estágio do processo e finaliza a fase de definição geométrica do modelo. Este objeto recebe um fluxo de dados de *cubeSrc* através do procedimento *cubeMpr->SetInput(cubeSrc->GetOutput())*. O próximo passo, que inicia a fase de visualização, é a definição de um “ator” genérico tridimensional, *cube1*, que referencia todas as características geométricas e de iluminação do cubo. Este objeto, instância da classe *vtkActor*, recebe o fluxo de dados relativo aos estágios anteriores. Neste ponto, é necessário transformar as informações geométricas associadas ao contexto de visualização em uma imagem bidimensional que será a representação da “cena” na tela do computador. Isto ocorre através de uma instância da classe *vtkRenderer*, *ren1*, a qual recebe as informações relativas ao novo ator cubo através do comando *renWin->AddActor(cube1)*. O fluxo de informações termina na instância *renWin* da classe *vtkRenderer*, responsável pelo gerenciamento da janela que mostrará o resultado do processamento. Antes de finalizar o programa, todas as instâncias de classes criadas precisam ser explicitamente destruídas (métodos *Delete()*) para não comprometer o gerenciamento de memória no sistema operacional.

O código em Tcl/Tk (Figura 3.7b) efetua exatamente as mesmas operações que o anterior, mas a criação e o tratamento das instâncias de classes do VTK é simplificado. A estrutura do programa (Figura 3.7c) mostra como o funcionamento do VTK pode ser representado pela transferência de informações entre módulos de processamento interconectados.

Embora mais flexível que C++, a linguagem Tcl/Tk não apresenta o mesmo desempenho em fase de execução, já que o interpretador deve traduzir e executar seus comandos continuamente, o que exclui a possibilidade de efetuar otimização de código executável. Além disto, os controles de sintaxe precisam ser realizados a cada comando, ao passo que isto só ocorre uma vez na compilação de programas em C++.

Outra desvantagem da linguagem Tcl/Tk é que não dispõe de métodos eficientes para o gerenciamento de estruturas complexas de dados de forma interativa, ao contrário do que ocorre em C++. Isto implica que seu uso deveria ser restrito a aplicações envolvendo a visualização de estruturas geométricas de fácil definição (por exemplo, através da leitura de um arquivo preexistente em disco). Para a execução de tarefas mais complexas, sobretudo aquelas relacionadas com a modificação em tempo real da geometria e topologia do modelo, é aconselhável o uso de programas em C++, menos flexíveis em tempo de execução, porém mais poderosos.

Este mestrando tem desenvolvido aplicativos de modelagem geométrica usando tanto a linguagem C++, como Tcl/Tk. No primeiro caso, foi escrito um programa para recortar malhas vazadas representando falhas geológicas ao longo das linhas de interseção com superfícies topográficas. Este trabalho foi realizado durante as atividades de iniciação científica junto ao Laboratório de GeoModelagem 3D do DPM / IGCE / UNESP (programa *Tricut*, LINDENBECK et al., 2002). Já o segundo trabalho, realizado durante este Mestrado, envolveu o desenvolvimento em Tcl/Tk do programa *Tensor3D* para deformar em tempo real a representação 3D de estruturas geológicas (capítulo 5). Em ambos os casos, a escolha da linguagem dependeu da finalidade dos aplicativos: modelagem geométrica avançada e eficiente (*Tricut*) versus flexibilidade (*Tensor3D*).

3.4.4 Estrutura de arquivos de dados do VTK

O sistema VTK possui numerosos filtros destinados à leitura / importação de arquivos de dados de acordo com os principais formatos usados em computação científica. Destaca-se o formato MOVIE.BYU (MOVIE.BYU FILE FORMAT, 2005), amplamente utilizado no Laboratório de GeoModelagem 3D do DPM / IGCE / UNESP para o intercâmbio de informações entre programas de modelagem / visualização distintos. Os arquivos neste formato possuem extensão *.g ou *.dis.

O sistema VTK possui também um formato de arquivos próprio, extremamente flexível, cuja estrutura será descrita a seguir. Tal formato foi extensivamente utilizado para

criar modelos digitais a serem lidos pelo programa *Tensor3D* e, por ser bem estruturado, pode ser usado como exemplo para a especificação de objetos 3D.

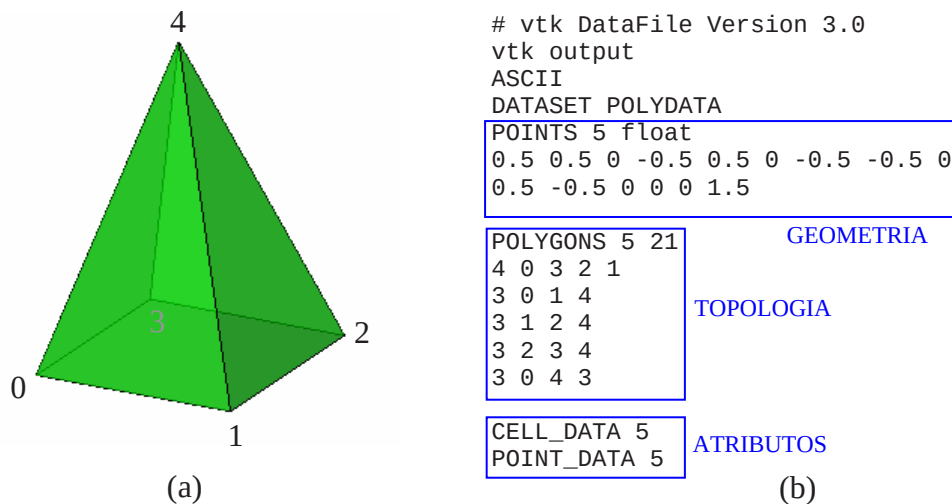


Figura 3.8 – (a) Exemplo de modelo geométrico. (b) Arquivo *.vtk correspondente ao objeto.

A Figura 3.8a mostra um exemplo de modelo geométrico visualizado com o VTK. Trata-se de uma pirâmide com base quadrada, um objeto constituído de cinco pontos e cinco polígonos ou faces. O arquivo *.vtk responsável por sua definição é mostrado na Figura 3.8b. As primeiras duas linhas contêm informações de cabeçalho necessárias para identificar arquivos de dados no formato do VTK. A terceira linha pode conter as palavras reservadas *ASCII* ou *BINARY*, indicando que o arquivo poderá ser de texto legível (*ASCII*), ou que será usada a formatação otimizada de dados (*BINARY*). A segunda opção é muito importante para o armazenamento de grande volume de dados, proporcionando uma considerável economia de espaço utilizado em disco. A quarta linha é introduzida pela palavra reservada *DATASET* e indica o tipo de objeto modelado no arquivo. Um modelo geométrico constituído de polígonos é definido através da palavra reservada *POLYDATA*, que corresponde à classe de armazenamento *vtkPolyData*.

A primeira seção do arquivo logo após as informações de cabeçalho contém a geometria do modelo, isto é, a lista de coordenadas dos pontos que o definem. A seção é introduzida pela palavra reservada *POINTS*, seguida do número de pontos que serão especificados e do formato numérico das coordenadas: *int* para valores inteiros (ocupando 4 bytes cada), *float* para valores decimais em precisão simples (4 bytes cada, 6 a 9 dígitos decimais) e *double* para valores decimais em precisão dupla (8 bytes cada, 15 a 17 dígitos decimais). O resto da seção contém as coordenadas (*x*, *y*, *z*) dos pontos do objeto. Na medida em que são lidos, os pontos recebem um número de identificação crescente não negativo (*id*),

começando pelo valor 0. Desta forma, no exemplo da figura haverá cinco pontos, tendo códigos de 0 a 4, sendo o ponto N° 0 (0.5, 0.5, 0), o N° 1 (-0.5, 0.5, 0) e assim por diante. A ordem de especificação dos pontos é importante e determina os códigos atribuídos aos mesmos.

A seção seguinte do arquivo contém informações de topologia, isto é, conectividade dos pontos. Nesta seção especifica-se de que maneira os pontos são conectados para formar polígonos. A primeira linha consiste na palavra reservada *POLYGONS*, um dos primitivos de modelagem geométrica definidos na seção 3.4.2, seguida do número de polígonos e do número total de valores que serão fornecidos nas linhas seguintes da seção. Esta última informação serve para controlar a consistência das informações topológicas especificadas. É possível verificar que na seção de topologia do arquivo de exemplo, 21 valores seguem a primeira linha.

O resto da seção destina-se à enumeração dos códigos de pontos que definem cada polígono, precedidos pelo número de pontos do mesmo. Por exemplo, a linha *4 0 3 2 1* indica que será inserido um polígono de quatro pontos cujos códigos são 0, 3, 2 e 1, de acordo com os códigos implicitamente definidos na seção de geometria. Este polígono corresponde à base da pirâmide da Figura 3.8a. Os demais polígonos da seção de topologia são triângulos e correspondem às faces laterais da pirâmide. A ordem de enumeração dos pontos de um polígono segue a convenção da biblioteca gráfica OpenGL (2005), usada internamente pelo sistema VTK, e assume que uma sequência de códigos em sentido anti-horário corresponde à definição do polígono visto de “fora para dentro” (em relação à superfície do modelo principal), ao passo que uma sequência em sentido horário define o mesmo de “dentro para fora”. Esta distinção pode parecer irrelevante, mas em computação gráfica (e no sistema VTK) é possível atribuir informações diferentes às duas especificações de um mesmo polígono. Por exemplo, as faces da pirâmide da Figura 3.8a possuem cor verde quando observadas de “fora para dentro” (ordem anti-horária dos pontos especificada na seção de topologia), mas as mesmas faces vistas de um ponto interno ao modelo poderiam ser sombreadas usando uma cor distinta. Esta é uma importante propriedade que permite diferenciar partes internas e externas de um modelo geométrico. Os polígonos do exemplo da figura foram especificados levando em conta a ordem anti-horária dos pontos, de forma que as faces da pirâmide são consideradas externas em relação ao observador.

A última seção do arquivo de dados é opcional e destina-se à especificação de atributos associados aos pontos ou às faces do modelo. Tais atributos podem ser representados por valores escalares, vetores ou matrizes quadradas de tensores (vide seção 3.4.2). No

exemplo da figura, não foram definidos atributos, sendo que a seção limita-se a declarar cinco atributos para os pontos da pirâmide e cinco para suas faces. Neste caso, o sistema VTK preenche os campos de atributos de forma automática, de acordo com informações gráficas associadas ao modelo (cor das faces, vetores normais calculados para os pontos, etc.).

O objeto cuja estrutura foi mostrada nesta seção é extremamente simples. Contudo, as regras de definição de geometria e topologia expostas são as mesmas para modelos mais complexos, tanto em termos de morfologia como de volume de informações. A geometria, topologia e, eventualmente, atributos são armazenados de forma semelhante em formatos de arquivos de programas e bibliotecas gráficas diferentes do VTK. A descrição feita nesta seção visou também proporcionar os conceitos fundamentais para a escrita de programas de conversão de e para o formato de dados do sistema VTK, caso seja necessária uma interação do mesmo com outros formatos não suportados (um exemplo é o formato *.ts do programa GOCAD).

3.4.5 Instalação do VTK

O VTK foi inicialmente desenvolvido para ser utilizado em aplicativos no sistema operacional UNIX. Os autores da biblioteca não disponibilizaram código pronto para ser executado (código binário ou pré-compilado), e sim o conjunto de arquivos em C++ que constitui o código fonte do VTK. A única exceção é a versão pré-compilada para o sistema operacional Windows, desenvolvida para agilizar a execução dos numerosos programas de exemplo na linguagem Tcl/Tk que acompanham a biblioteca. A razão do VTK não possuir versões pré-compiladas para UNIX é que existem muitas variantes do mesmo, sendo que cada instância específica pode sofrer modificações importantes de uma versão para outra. Isto significa que o código fonte precisa ser compilado para a plataforma específica do usuário (Intel, IBM, SUN, etc.), isto é, convertido para o código de máquina (ou binário) do processador. Este mestrando tem utilizado o VTK nos sistemas operacionais Irix (para plataforma Silicon Graphics), Solaris (plataforma SUN) e, recentemente, Linux Fedora Core 3 (plataforma Intel com 32 bits). Para cada sistema, foi necessário realizar um processo de configuração e compilação. Nesta seção serão descritos os procedimentos de instalação para o sistema operacional Linux, utilizado durante todo o período do Mestrado.

Será assumida a utilização do sistema operacional Red Hat Fedora Core 3 em plataforma Intel 32 bits. Todos os aplicativos foram instalados no diretório */usr/local*, normalmente usado para programas independentes do sistema operacional. A estação usada foi um Pentium 4 com 2.4 GHz de frequência de clock, 1 Gbyte de memória RAM, 1 HD de

120 Gbytes (e outro de 80 Gbytes para realizar cópias de segurança de arquivos), e uma placa gráfica aceleradora NVIDIA GeForce FX 5600.

O VTK utiliza a biblioteca gráfica Open-GL, sendo necessária sua instalação prévia. Nas atuais placas gráficas aceleradoras, os recursos da biblioteca Open-GL estão implementados em hardware para obter o máximo desempenho de aplicativos 3D. Portanto, é suficiente instalar o *driver* de controle da placa aceleradora da companhia NVIDIA, o qual contém o software necessário para ativar os módulos Open-GL da placa. O driver para o sistema operacional Linux foi baixado da *homepage* da companhia (NVIDIA, 2005). Na página principal, são acessados os links “Download Drivers”, “Linux, FreeBSD and Solaris Drivers”, e “1.0-6629” na seção *Linux IA-32*. O código 1.0-6629 indica a versão do *driver*. Em seguida, é exibida uma página com a descrição dos procedimentos de instalação. O *driver* está contido no arquivo *NVIDIA-Linux-x86-1.0-6629-pkg1.run*, que pode ser acessado a partir desta página. Este arquivo é válido para todos os modelos de placas da companhia NVIDIA.

Após a gravação do arquivo em disco, é necessário finalizar o gerenciador gráfico do Linux e trabalhar em modalidade de texto (*shell*). Em seguida, como usuário *root*, deve-se executar o seguinte comando a partir do diretório onde foi gravado o arquivo:

```
# bash ./NVIDIA-Linux-x86-1.0-6629-pkg1.run
```

O símbolo “#” é o *prompt* de comandos do superusuário (*root*). Será visualizado um conjunto de informações e instruções para a correta instalação do *driver*. O procedimento não é complexo, sendo apenas necessário selecionar a opção *OK* em todas as janelas exibidas. Em seguida, deve-se modificar o arquivo */etc/X11/xorg.conf* conforme instruções contidas no arquivo *README* (seção *sec-03*) que acompanha o *driver* da placa (arquivo */usr/share/doc/NVIDIA_GLX-1.0/README*, criado após a instalação do *driver*). O arquivo *xorg.conf* contém as configurações do gerenciador gráfico para as distribuições Linux da empresa Red Hat, e é composto por seções (blocos de instruções) destinadas ao controle de cada funcionalidade. Para modificar o arquivo é necessário usar o usuário *root* e executar os seguintes passos (é aconselhável fazer uma cópia do arquivo antes de proceder):

- Na seção “*Device*”, identificar a linha *Driver “nv”* e substituí-la por *Driver “nvidia”*;
- Na seção “*Module*”, remover as linhas *Load “dri”* e *Load “GLcore”*, e acrescentar a linha *Load “glx”*.

O arquivo */usr/share/doc/NVIDIA_GLX-1.0/README* contém uma descrição completa de todas as funcionalidade da placa aceleradora que podem ser configuradas. Sua leitura é importante para obter o máximo desempenho da placa instalada, e contém também

uma lista de soluções para os problemas mais comuns que podem ocorrer durante a instalação do *driver*.

Para voltar ao ambiente gráfico do sistema Linux, é suficiente executar o comando *startx* a partir da *shell* de comandos (não é preciso usar o usuário *root*). Dentro deste ambiente, é possível verificar a instalação do *driver* invocando uma janela de comandos e executando o programa *glxgears*. Trata-se de um aplicativo gráfico 3D que simula a rotação de três engrenagens. Seu correto funcionamento significa que o *driver* da placa (e portanto a biblioteca Open-GL) foi instalado com êxito. O programa *glxgears* exibe a informação de quadros por segundo (*frames per second, FPS*) obtidos pela placa aceleradora e representa uma medida básica de comparação de desempenho entre modelos de placas distintos. Com o microcomputador Pentium 4 utilizado nesta pesquisa foi obtido um valor médio de 1900 quadros por segundo.

As primeiras versões do VTK apresentavam freqüentemente problemas de compilação devido à necessidade de definir manualmente o valor de numerosas variáveis de ambiente dependentes da variante do sistema UNIX utilizada. Ao mesmo tempo, a complexidade da estrutura do sistema tem aumentado, tornando difícil o gerenciamento dos arquivos *Makefile* que controlam sua compilação (seção 3.5.1). Para minimizar tais inconvenientes, as versões mais recentes do VTK utilizam uma ferramenta de auxílio ao processo de compilação, denominada CMake (2005). Este é um programa cuja instalação previa é imprescindível para a correta compilação do VTK, e pode ser obtido acessando sua página principal na rede Internet. Nesta página, deve-se selecionar o link “*download*” e em seguida o arquivo *cmake-2.0.6-x86-linux.tar.gz (latest release)*. O arquivo pode ser gravado no diretório temporário */tmp* do Linux. O sufixo *.tar.gz* indica que o programa é constituído de uma estrutura de diretórios que foi convertida para um arquivo único através do comando *tar*, e em seguida compactada usando o comando *gzip* (isto reduz o tamanho do arquivo principal e agiliza sua transferência em redes de computadores). Para expandir o diretório do programa, deve-se executar os seguintes comandos como usuário *root*:

```
# cd /tmp
# gunzip cmake-2.0.6-x86-linux.tar.gz
# cd /usr/local
# tar xvf /tmp/cmake-2.0.6-x86-linux-files.tar
```

Isto atualizará os diretórios *share*, *doc*, *man* e *bin* (dentro de */usr/local*) com os arquivos do CMake. O arquivo binário do programa é */usr/local/bin/cmake*.

Para utilizar o sistema VTK com a linguagem de programação Tcl/Tk, é necessário verificar que as bibliotecas da mesma tenham sido instaladas no sistema operacional Linux.

Trata-se de quatro módulos de instalação (no formato *.rpm da empresa Red Hat), a saber: *tcl-8.3.5-96.0.1*, *tcl-devel-8.3.5-96.0.1*, *tk-8.3.5-96.0.1* e *tk-devel-8.3.5-96.0.1*. Os números indicados nestes arquivos denotam a versão das bibliotecas e podem ser diferentes dos que foram usados neste Mestrado. Para verificar se tais módulos estão instalados, pode-se usar, sempre como usuário *root*, os comandos

```
# rpm -qa | grep tcl  
# rpm -qa | grep tk
```

que exibem uma lista dos módulos instalados cujo nome contenha as siglas “tcl” e “tk”. Os quatro arquivos do parágrafo anterior devem estar contidos na saída gerada pelos dois comandos. Se os módulos não estiverem instalados, será necessário baixar os mesmos da Internet (RPMFIND, 2005). Em seguida, deverá ser realizada sua instalação manual usando os seguintes comandos (na ordem especificada), a partir do diretório onde os arquivos foram copiados:

```
# rpm -ivh tcl-8.3.5-96.0.1.rpm  
# rpm -ivh tcl-devel-8.3.5-96.0.1.rpm  
# rpm -ivh tk-8.3.5-96.0.1.rpm  
# rpm -ivh tk-devel-8.3.5-96.0.1.rpm
```

Para verificar se as bibliotecas da linguagem Tcl/Tk foram instaladas corretamente, pode-se invocar o interpretador da mesma, usando o comando */usr/bin/wish*. Será visualizada uma janela do módulo Tk e a *shell* de comandos aguardará instruções a serem interpretadas.

Com a instalação da biblioteca OpenGL e do interpretador da linguagem Tcl/Tk, a estação está pronta para a configuração e compilação do VTK. A partir da página principal do sistema na rede Internet (VISUALIZATION TOOLKIT, 2005), deve-se selecionar o *link* “*Get The Software*”. Na nova página exibida, encontra-se uma seção chamada “*Download the official release*”. A versão utilizada neste Mestrado foi a 4.2. Os arquivos *VTK-4.2-LatestRelease.tar.gz* e *vtkData-4.2.tar.gz* devem ser selecionados e gravados em disco. O primeiro contém uma versão compactada de todos os arquivos necessários para compilar o VTK. O segundo arquivo contém dados usados para os programas de exemplos que acompanham a biblioteca. É também possível baixar o arquivo *VTKDocHtml-4.2.tar.gz*, o qual contém uma descrição de todas as classes do sistema, acessível a navegadores *Web*. Tal lista está também disponível na página da biblioteca, na seção “*Manual*”.

Assumindo */usr/local* como diretório de instalação do VTK, os arquivos devem ser descompactados, como usuário *root*, através dos seguintes comandos:

```
# cd /usr/local  
# tar xvfz VTK-4.2-LatestRelease.tar.gz  
# tar xvfz vtkData-4.2.tar.gz
```

Serão criados os diretórios *VTK* e *VTKData-release-4.2* dentro de */usr/local*. O diretório *VTK* contém vários subdiretórios de código fonte em C++, divididos de acordo com as especialidades das classes (processamento de imagens, procedimentos comuns, processamento 3D, etc.). Existe também um arquivo chamado *README.html*, o qual contém um guia de instalação da biblioteca com os procedimentos ilustrados nesta seção. Para iniciar a configuração das variáveis de ambiente necessárias à compilação do VTK, deve-se invocar o programa CMake a partir do diretório */usr/local/VTK*:

```
# cmake -i
```

Será exibida a pergunta “*Would you like to see advanced options?*”. A resposta deve ser “No”. Em seguida, o programa mostrará uma lista de variáveis cujo conteúdo deve ser definido de forma interativa. A maioria delas possui um valor *default* que não precisa ser modificado. Para estes casos, é suficiente apertar a tecla <ENTER>, o que confirmará os valores predefinidos. Contudo, há variáveis cujo conteúdo deve ser especificado explicitamente:

Variável	Valor
<i>BUILD_EXAMPLES</i>	<i>ON</i>
<i>BUILD_SHARED_LIBS</i>	<i>ON</i>
<i>CMAKE_BUILD_TYPE</i>	<i>CMAKE_CXX_FLAGS</i>
<i>VTK_DATA_ROOT</i>	<i>/usr/local/VTKData-release-4-2</i>
<i>VTK_USE_HYBRID</i>	<i>ON</i>
<i>VTK_USE_PATENTED</i>	<i>ON</i>
<i>VTK_WRAP_TCL</i>	<i>ON</i>

Após a definição de todas as variáveis o programa CMake terminará sua execução. A configuração do VTK está completa. O próximo passo é a compilação propriamente dita, que deve ser invocada executando o comando *make* no diretório */usr/local/VTK*. A compilação é demorada: sua duração na máquina descrita acima foi de trinta minutos.

Uma vez finalizado o processo de compilação, o diretório */usr/local/bin/VTK* conterá as bibliotecas do VTK (arquivos com extensão *.so) utilizáveis para a compilação de programas em C++, e o interpretador de comandos do sistema (arquivo *vtk*). As bibliotecas podem ser utilizadas em programas (em C++) compilados, usando arquivos *Makefile* (seção 3.5.1). Já o arquivo *vtk* é um executável que interpreta *scripts* na linguagem Tcl/Tk. O subdiretório *Examples* do VTK contém numerosos exemplos de programas que mostram as

funcionalidades de modelagem e visualização do sistema. Alguns programas estão escritos nas linguagens C++, Python e Java, sendo a grande maioria em Tcl/Tk. Para a execução destes últimos, é suficiente invocar o comando *vtk* usando como argumento o nome do programa desejado, por exemplo “*vtk programa.tcl*”. Como será visto mais adiante, esta é uma forma rápida de desenvolvimento e teste de aplicativos relativamente simples e não requer o processo de compilação a cada alteração do programa-fonte.

3.5 Ferramentas para o desenvolvimento de programas

Nesta seção serão descritas algumas ferramentas de desenvolvimento para a confecção de programas em C/C++ em ambiente Linux. O objetivo é ilustrar as etapas de preparação de um ambiente de trabalho para o desenvolvimento, usando o sistema VTK, de aplicativos de modelagem / visualização complexos. Estes requerem o uso de linguagens compiladas, mais poderosas e eficientes do que as linguagens interpretadas (vide seção 3.4.3). Todas as ferramentas descritas a seguir utilizam a licença GNU e são de código aberto.

O sistema Linux possui numerosos aplicativos de desenvolvimento. O mais importante é provavelmente o compilador *gcc* (2005), que permite compilar programas em linguagem C/C++ e é também uma ferramenta de código aberto. Um compilador de utilização livre constitui um importante passo para promover o desenvolvimento de aplicativos, já que não é necessário pagar nenhuma licença para gerar módulos executáveis (binários) perfeitamente funcionantes a partir de um código fonte em C/C++. O compilador suporta também outras linguagens, como Fortran77 e Java.

3.5.1 O arquivo *Makefile*

Nos sistemas operacionais UNIX (incluindo Linux), a compilação de projetos complexos constituídos de vários módulos de código fonte ocorre através de um arquivo de controle, chamado *Makefile*, o qual contém todas as diretivas e opções de compilação a serem passadas para o compilador. Outra função do arquivo *Makefile* é indicar todas as bibliotecas que serão utilizadas no projeto. Segue um exemplo de arquivo *Makefile*.

```
# Arquivo Makefile para compilacao de programa
# de teste usando bibliotecas do sistema VTK
CC = gcc
INCLUDES = -I/usr/local/include/vtk
CFLAGS = -Wall -O2
LDFLAGS = -L/usr/local/lib/vtk
LIBS = -lvtkCommon -lvtkRendering -lvtkGraphics -lvtkIO
```

```

teste: main.o graphics.o
      $(CC) $(LDFLAGS) $(LIBS) main.o graphics.o -o teste

main.o : main.c
      $(CC) $(INCLUDES) $(CFLAGS) main.c -c

graphics.o : graphics.c
      $(CC) $(INCLUDES) $(CFLAGS) graphics.c -c

all: teste

```

Este exemplo assume que o programa a ser compilado seja constituído de dois arquivos de código fonte em C, *main.c* e *graphics.c*, sendo que o segundo contém comandos que invocam funções da biblioteca gráfica VTK. O programa executável se chama *teste*. Para criá-lo, é necessário executar, dentro de um terminal *shell* do Linux, o comando *make* no diretório que contém o arquivo *Makefile*. Este comando, cujo nome completo é GNU Make (2004), consiste em um interpretador de comandos para arquivos *Makefile*. Trata-se de um programa de código aberto.

Um arquivo *Makefile* é constituído de duas seções: declaração de variáveis e cláusulas. Como mostrado no arquivo de exemplo, o *Makefile* utiliza algumas variáveis definidas internamente. A primeira, CC, indica qual compilador será utilizado (no caso, o compilador *gcc*). Sempre que for necessário invocar o compilador, basta acessar o conteúdo da variável CC, utilizando para tanto a expressão \$(CC). O mesmo procedimento vale para as demais variáveis do exemplo. INCLUDES indica o caminho dos arquivos de cabeçalho das classes do VTK. Tais arquivos fornecem a informação de todas as funções que a biblioteca disponibiliza. A variável CFLAGS contém as opções de compilação, sendo que *-Wall* ativa todas as mensagens de alerta e *-O2* determina a criação de um código executável otimizado (mais rápido). Já a variável LDFLAGS especifica o diretório onde serão encontradas as bibliotecas dinâmicas do VTK (arquivos com extensão *.so) que são utilizadas pelo programa *teste*. A variável LIBS contém uma lista destas bibliotecas (O sufixo “.so” não deve ser indicado).

A segunda parte do arquivo *Makefile*, contém uma lista de cláusulas ou regras que o compilador deve seguir para gerar o executável. Uma cláusula assume a forma

```

<objetivo > : <componentes>
               <comando>

```

sendo <objetivo> o nome do arquivo a ser gerado nesta regra, <componentes> a lista de componentes necessários para tanto e <comando> o meio para a execução deste processo. No arquivo de exemplo, existem três objetivos, isto é, arquivos a serem gerados, sendo o primeiro final (o programa executável *teste*) e os demais intermediários (*main.o* e *graphics.o*, também chamados códigos objetos). Na primeira regra é possível verificar que *teste* necessita dos

arquivos intermediários. De fato, tais arquivos contêm os arquivos-fonte *main.c* e *graphics.c* compilados, isto é, traduzidos para a linguagem de máquina do processador, mas nenhuma referência às funções de biblioteca utilizadas (VTK e biblioteca padrão da linguagem C) “está resolvida”. Isto significa que o compilador ainda não especificou no código objeto onde é possível encontrar as bibliotecas necessárias. O problema é solucionado através de um processo chamado “linkeditação”, realizado por um programa (*linker*) que pode ser invocado a partir do compilador, como no exemplo apresentado. Na primeira cláusula, a opção *-o* do compilador indica qual é o nome do arquivo a ser gerado (que coincide com o nome do objetivo). Já a segunda e terceira cláusulas especificam a regra para criação dos códigos objetos (extensão *.o*) correspondentes aos códigos fontes *main.c* e *graphics.c*. A opção *-c* indica que o arquivo a ser gerado não é executável, e sim objeto, isto é, intermediário. Existe uma quarta cláusula, *all : teste*, a qual especifica qual é o objetivo padrão, isto é, o objetivo requisitado ao se executar o comando *make* sem argumentos. A especificação da componente *teste* para este objetivo indica que o executável corresponde ao objetivo padrão. Objetivos (arquivos) intermediários podem também ser gerados especificando seus nomes como argumentos para o comando *make*. Por exemplo, o arquivo *graphics.o* é criado através do comando *make graphics.o*. Isto é normalmente desnecessário, já que todos os objetivos intermediários são gerados automaticamente a partir do objetivo padrão.

A utilização de arquivos *Makefile* para a compilação de programas de hierarquia complexa simplifica o gerenciamento de múltiplos módulos de código fonte. Além disto, a utilização de variáveis para referenciar e agrupar as entidades relevantes do programa torna mais intuitivo o estudo da estrutura de projetos de desenvolvimento complexos. Por fim, a utilização de um interpretador de arquivos *Makefile* de código aberto, reforça o caráter livre do desenvolvimento de software.

3.5.2 O ambiente de desenvolvimento *anjuta*

O compilador *gcc* converte arquivos de texto contendo programas em C/C++ para código de máquina executável pelo computador. Estes arquivos podem ser criados utilizando qualquer editor. Contudo, quando o projeto a ser desenvolvido incluir numerosos módulos de trabalho (por exemplo, gráficos, de interface, de controle de erro, etc.), a criação de arquivos de código fonte através de um editor de texto torna-se um processo demorado. Além disto, a integração dos vários módulos do programa torna-se mais difícil à medida que aumenta o número de módulos a serem escritos.

Para solucionar este problema, existem ambientes de desenvolvimento integrados, os quais permitem gerenciar mais de um módulo de código fonte ao mesmo tempo. Um exemplo de tais aplicativos é o sistema *anjuta DevStudio* (2005). Este é uma ferramenta de código aberto que permite o desenvolvimento de projetos complexos em C/C++ em ambiente Linux. A versão do ambiente utilizada neste trabalho é a 1.2.2. Dentre as funcionalidades do *anjuta* é possível destacar as seguintes:

- Editor de texto com interessantes funções de busca de palavras, coloração de palavras reservadas para agilizar a leitura do código, autopreenchimento das expressões de programação mais utilizadas, bem como de nomes de funções e variáveis declaradas pelo programador.
- Integração de vários arquivos de código fonte em um único projeto, visando facilitar o gerenciamento de aplicações complexas. Todas as facilidades do item anterior estão disponíveis para cada arquivo de texto que o desenvolvedor pretenda gerenciar em um dado instante. Desta forma, não é preciso abrir múltiplas instâncias de um editor de texto para controlar mais de um módulo ao mesmo tempo.
- Assistente de compilação. Todos os comandos mais importantes de compilação podem ser invocados a partir do *anjuta*. O compilador utilizado é o *gcc*, mas o *anjuta* proporciona uma interface para a especificação de arquivos de compilação de projetos complexos (*Makefiles*), bem como de bibliotecas a serem incluídas na aplicação. Existe também a possibilidade de fornecer ao compilador opções avulsas de trabalho.
- Ferramentas de *debugging*. Tais ferramentas permitem rastrear o código fonte do projeto em busca de erros de programação, durante a execução do mesmo. Para tanto, é possível especificar *breakpoints* ou pontos de interrupção da execução do programa para observar o conteúdo de variáveis “suspeitas” (fontes de erros).
- Integração com bibliotecas para criação de interfaces de trabalhos (GUI’s, ou *Graphical User Interfaces*). Esta funcionalidade permite criar interfaces gráficas para controle do aplicativo em desenvolvimento, diretamente no ambiente de trabalho do *anjuta*. Para tanto, utiliza-se a biblioteca GTK (descrita na próxima seção), que fornece um conjunto de componentes gráficas (como botões, barras de rolagem, etc.), visando tornar mais amigável a interface com o usuário do aplicativo.

A Figura 3.9 mostra um exemplo de utilização do *anjuta*. O ambiente possui várias janelas integradas de trabalho. Em particular, a parte superior da janela do programa mostra alguns botões destinados ao gerenciamento de cada módulo de texto. Funções usuais incluem a leitura e gravação de um arquivo em disco, a busca por palavras dentro do texto, etc. A

porção direita da tela (região 1) exibe o conteúdo do arquivo de código fonte (módulo) sendo editado. A porção esquerda (região 2) exibe o conteúdo da janela de projeto (*Project*), a qual permite navegar pelo conjunto de módulos do mesmo. Nesta, é suficiente selecionar o módulo a ser exibido / modificado, para que seu conteúdo seja visualizado na janela à direita.

Por fim, a porção inferior da tela do programa (região 3) mostra todas as mensagens de alerta e eventuais erros ocorridos durante a compilação do projeto. Existem outras janelas que permitem, por exemplo, observar o conteúdo de variáveis durante a execução para detectar eventuais erros de programação.

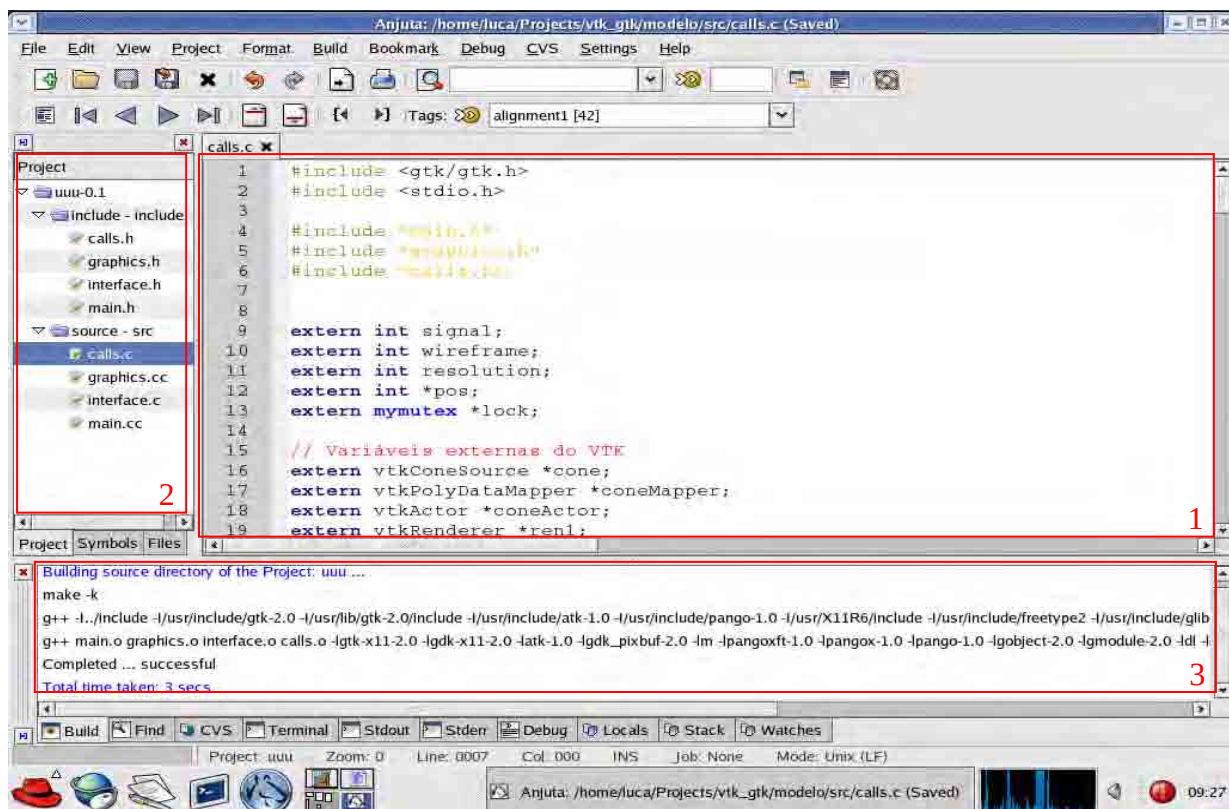


Figura 3.9 – Ambiente de desenvolvimento integrado *anjuta*. 1. Janela para escrita de módulos em C/C++. As palavras reservadas da linguagem recebem cores distintas de acordo com sua categoria. Isto facilita a leitura do código. 2. Gerenciador dos módulos sendo desenvolvidos. A seleção de um item da lista permite editar seu conteúdo na região 1. 3. janela que exibe os comandos executados pelo ambiente para realizar a compilação do projeto, bem como eventuais mensagens de erro e alerta.

3.5.3 A biblioteca GTK

A biblioteca GTK (*GNU TOOLKIT*, 2005) é utilizada para desenvolver todos os aplicativos gráficos no ambiente gráfico do Linux e implementa janelas de interação gráfica através de funções de biblioteca em C. Foi desenvolvida visando uma abordagem modular e oferece um conjunto de objetos gráficos (*widgets*) através dos quais o usuário pode interagir com um aplicativo (manipular botões, inserir informações, etc.). O objetivo da biblioteca é

proporcionar a máxima facilidade de uso dos componentes oferecidos. Neste trabalho, utilizou-se a versão 2.0 da biblioteca GTK.

Segue um exemplo de programa em C com os comandos necessários para a criação da janela da Figura 3.10:

```
#include <gtk/gtk.h>

/* Funcao para imprimir mensagem */
void hello( GtkWidget *widget, gpointer data )
{
    g_print ("Hello World\n");
}

/* Funcao principal */
int main( int argc, char *argv[] )
{
    GtkWidget *window; /* Janela */
    GtkWidget *button; /* Botao definido na janela */

    gtk_init (&argc, &argv);

    /* Cria janela */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    /* Cria botao */
    button = gtk_button_new_with_label ("Hello World");

    /* Indica que a funcao "hello" deve ser executada ao se pressionar
       o botao */
    g_signal_connect (G_OBJECT (button), "clicked",
                      G_CALLBACK (hello), NULL);

    /* Insere botao na janela */
    gtk_container_add (GTK_CONTAINER (window), button);
    gtk_widget_show (button);
    gtk_widget_show (window);

    /* Inicia ciclo de interacao com o usuario */
    gtk_main ();
    return 0;
}
```



Figura 3.10 – Exemplo de aplicação GTK.

O exemplo acima cria uma janela gráfica com um botão central de teste e com todos os botões de gerenciamento gráfico tradicionais (fechar, minimizar, redimensionar). Estes últimos são, na verdade definidos no ambiente de gerenciamento de janelas subjacente ao GTK (no caso de todos os sistemas UNIX, o gerenciador X), mas isto é controlado pela biblioteca de forma transparente ao usuário.

Apesar da relativa simplicidade do código apresentado, o desenvolvimento de interfaces mais elaboradas requer a definição de numerosos *widgets* e funções, tornando a tarefa mais complexa. Para tanto, os desenvolvedores de aplicativos gráficos de código aberto

para a comunidade Linux implementaram uma ferramenta gráfica para a criação de interfaces baseadas na biblioteca GTK.

3.5.4 A ferramenta *glade*

O editor de interfaces *glade* (GTK+ USER INTERFACE BUILDER, 1999) tem como objetivo facilitar a criação de interfaces gráficas de interação com o usuário através da biblioteca GTK. Esta ferramenta encontra-se na versão 2.10.0. As várias janelas que constituem o ambiente são mostradas na Figura 3.11.

O intuito da ferramenta é permitir que o desenvolvedor crie interfaces arrastando e posicionando componentes nas mesmas a partir de uma lista de objetos disponíveis. Além disto, deve ser possível modificar interativamente as propriedades de visualização e comportamento (modalidade de interação com o usuário) de cada componente.

A primeira janela do ambiente, denominada *glade* (em cima à esquerda na Figura 3.11), contém as funções principais para leitura e escrita de interfaces no disco. Cada arquivo de interface possui a extensão *.*glade*. A janela *Palette* (em baixo à esquerda) exhibe todas as componentes (*widgets*) que o desenvolvedor pode inserir em sua janela. A mais importante é a componente *window*, que representa o formulário ou *frame* dentro do qual podem ser inseridos os demais objetos. Dentre os principais *widgets* disponíveis, existem botões, barras de rolagem, seletores de tipo *checkbox* ou *radio button*, campos para a introdução de texto, etc. A janela *Properties* (região central superior) permite modificar o conjunto de propriedades associadas a cada objeto, como seu tamanho padrão em *pixels*, estado (ativado / desativado) e posicionamento em relação aos demais objetos.

Na medida em que componentes são inseridas para criação de uma interface, a ferramenta *glade* cria uma hierarquia de objetos que pode ser visualizada na janela *Widget Tree*. A Figura 3.11 exhibe uma interface (janela *window1*, região central inferior) criada após alguns minutos de interação com o ambiente. A hierarquia de objetos definidos é visualizada na janela *Widget Tree* correspondente (à direita).

Após a confecção de uma interface gráfica, o ambiente *glade* permite salvar em disco o código em C necessário para sua criação. Este código é gravado em um arquivo denominado *interface.c*.

Durante o desenvolvimento de um projeto no ambiente *anjuta*, é possível invocar, através das teclas ATL + “g”, o editor de interfaces gráficas *glade*. Isto permite criar aplicativos gráficos de forma totalmente integrada ao *anjuta*. Uma vez desenvolvida a

interface, será criado um arquivo *interface.c* o qual poderá ser adicionado aos demais arquivos do projeto.

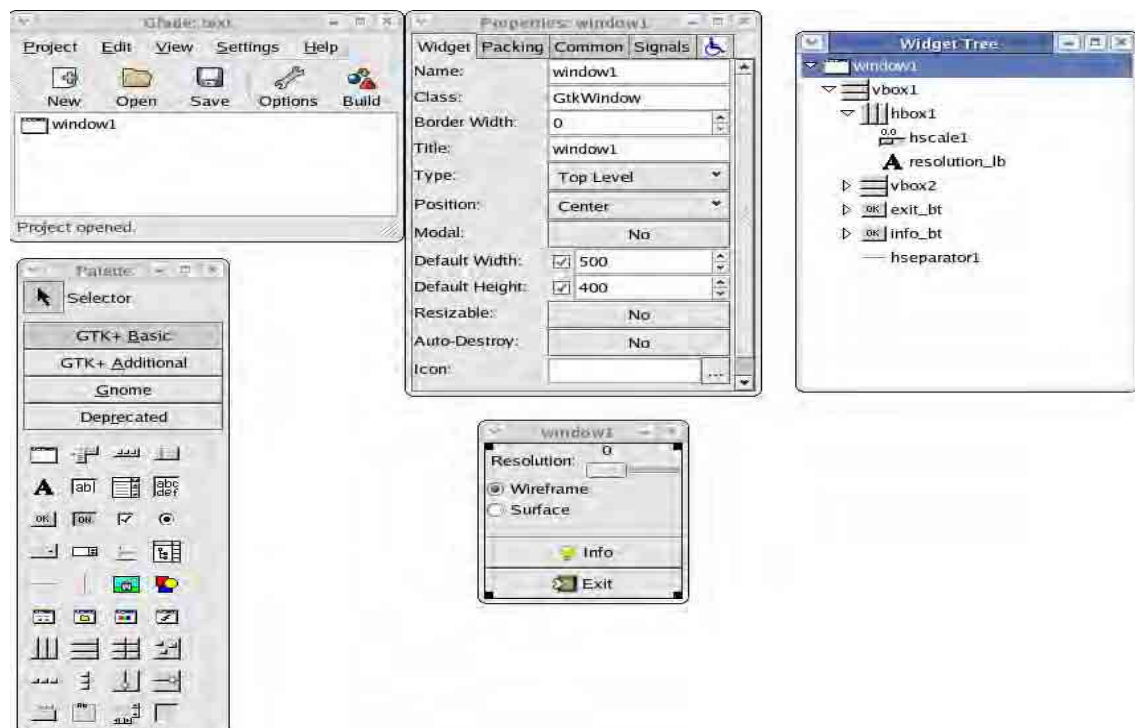


Figura 3.11 – O ambiente *glade* para desenvolvimento de interfaces gráficas.

3.6 O programa GOCAD

O programa GOCAD (2005) é uma ferramenta comercial de visualização e manipulação de dados geológicos desenvolvida pela *ENSGN-Ecole Nationale Supérieure de Geologie* de Nancy, França. É constituído por um conjunto de módulos escritos em linguagem C++. Tais módulos são continuamente atualizados por diferentes grupos de pesquisa, cujas atividades são financiadas por um consórcio internacional de empresas e universidades, que no Brasil inclui a UNESP e a Petrobrás. O programa GOCAD, disponível para plataforma UNIX (incluindo Linux) e Windows, contém módulos que permitem visualizar diferentes tipos de dados, desde poligonais (pontos, linhas e superfícies) até volumétricos. Considerada a complexidade dos algoritmos de computação gráfica implementados, a visualização de dados requer a utilização de uma placa gráfica aceleradora. O programa utiliza a biblioteca gráfica Open-GL.

3.6.1 Interação com o programa

A interação com o GOCAD ocorre principalmente através do mouse. Para isto, o

programa apresenta-se sob a forma de janelas de comandos contendo botões e menus facilmente selecionáveis. A janela principal do GOCAD contém as seguintes regiões (Figura 3.12):

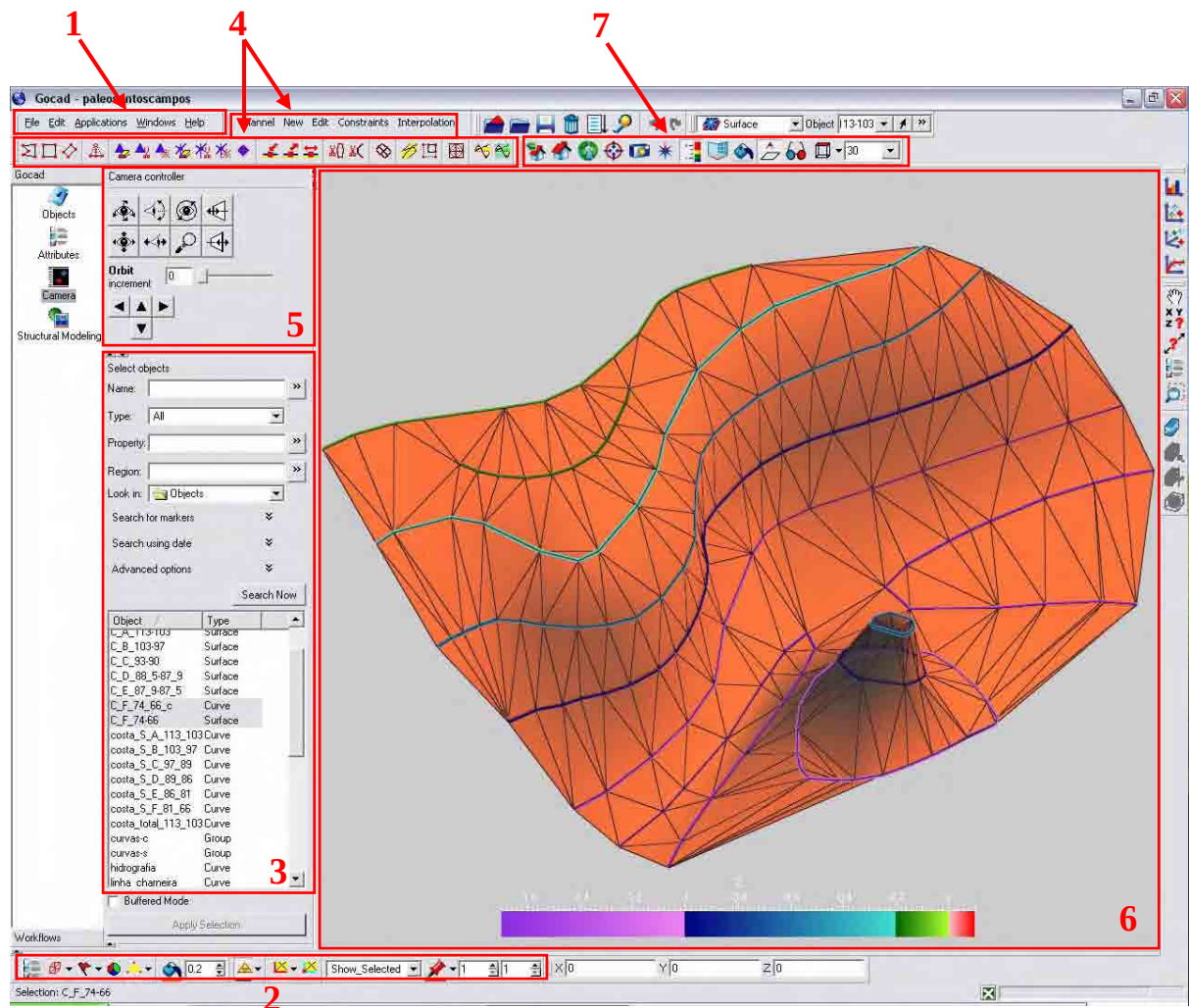


Figura 3.12 – Janela Principal do programa GOCAD. 1. Região de controle principal. 2. Gerenciador de atributos. 3. Gerenciador de objetos. 4. Menu de operações específicas. 5. Ferramenta de controle da câmera. 6. Janela de visualização 3D. 7. Propriedades gerais de visualização

1. **Região de controle principal:** Esta seção contém comandos para manipular objetos independente de sua estrutura específica. São presentes botões para ler do disco, copiar e gravar objetos tridimensionais.
2. **Gerenciador de atributos:** Conjunto de funções para modificar as opções de visualização de um objeto como, por exemplo, a cor de uma superfície, modalidades de sombreamento dos triângulos que a constituem, etc. O conjunto de opções de visualização oferecidas varia de acordo com a natureza dos objetos (curva, superfície, etc.).
3. **Gerenciador de objetos.** Ferramenta utilizada para ativar / desativar a visualização dos objetos. Apresenta uma lista dos nomes das entidades disponíveis, bem como seu tipo.
4. **Menu de operações específicas:** conjunto de comandos específicos associados ao objeto

atualmente selecionado. Por exemplo, é possível, entre outras operações, recortar uma superfície usando outros objetos, fechar uma curva, remover partes de um objeto, etc. Neste menu estão presentes comandos para criar novos objetos a partir de outros, como uma superfície a partir de um conjunto de pontos ou de curvas. O conjunto de comandos disponíveis para utilização é flexível: os submenus e seu conteúdo modificam-se de acordo com a natureza do objeto sobre o qual devem ser aplicados.

5. **Ferramenta de controle da câmera.** Conjunto de funções utilizadas para realizar o ajuste fino das propriedades da câmera, como ponto de vista e direção de observação.
6. **Janela de visualização 3D:** Esta região mostra a visualização 3D de todos os objetos selecionados no gerenciador da região 3. A utilização dos três botões do mouse permite rotacionar, transladar e afastar / aproximar os objetos exibidos. O controle do ponto de observação pode também ser realizado através do teclado. Por exemplo, as setas permitem transladar a imagem na direção especificada pelo usuário e o teclado numérico possui funcionalidades para modificar o ponto de observação simulando um vôo panorâmico em volta dos objetos visualizados.
7. **Propriedades gerais de visualização.** Através destas funções é possível modificar algumas propriedades associadas à janela de visualização da região 6. Por exemplo, pode-se salvar / restaurar a posição e direção de observação da câmera, afastar o ponto de vista de forma a visualizar todos os objetos atualmente selecionados, e gravar em disco imagens da visualização atual. É também possível selecionar a cor do fundo, realizar a seleção entre a projeção paralela e a perspectiva, e habilitar a visão estereoscópica (para a qual deve-se usar um óculos especial. Por fim, o usuário pode especificar visadas predefinidas (em planta, Norte, Sul, etc.) e definir um fator de exagero vertical adequado às suas necessidades.

3.6.2 Objetos disponíveis no GOCAD

O programa GOCAD permite a modelagem e visualização das seguintes entidades tridimensionais:

- **Point Set:** um conjunto de pontos independentes. Não há informação de conectividade (topologia).
- **Curve:** segmentos de linhas conectados ou não entre si. Uma curva pode ser constituída de múltiplas linhas poligonais não conectadas.
- **Surface:** superfície. Representa uma malha vazada de triângulos.

- **Voxel:** malha regular tridimensional constituída por elementos volumétricos denominados *Voxels*.
- **Sgrid:** conjunto de dados estratigráficos. Um *SGrid* é mais sofisticado que um *Voxel* porque o usuário pode adaptar seus limites a superfícies que modelam horizontes estratigráficos e particioná-lo ao longo de superfícies de falha.
- **Channel:** este objeto é definido por uma linha poligonal atravessada por seções poligonais. Utiliza-se esta entidade para representar estruturas longas e meandantes, tais como canais.
- **Well:** objeto que engloba informações geométricas e geológicas associadas a poços.

Entidades de mesma natureza ou heterogêneas podem ser agrupadas em estruturas denominadas *groups*. Isto permite gerenciar de forma simultânea as propriedades de numerosos objetos. Um caso típico é o dos objetos de tipo *Well*.

Algumas entidades são construídas a partir de outras, mais simples. É o caso das curvas, que são criadas a partir de um conjunto de pontos. O mesmo vale para as superfícies nas quais aos pontos tridimensionais são associadas informações de conectividade para gerar malhas de triângulos.

Todo objeto no programa GOCAD recebe um nome (mostrado na região 3 da Fig. 3.12) com o qual será identificado durante a execução. Isto é necessário porque em toda operação de visualização / manipulação deve ser sempre possível identificar corretamente as entidades sobre as quais o processamento será efetuado. Tal identificação poderia ser feita selecionando-se o objeto visualizado na janela 3D. Mas o objeto em questão poderia não estar visível no momento, surgindo assim a necessidade de um código (nome) para qualquer referência. Este nome acompanha o objeto mesmo fora da execução, pois tal informação é armazenada no arquivo de dados a ele associado.

O conjunto de informações e atributos disponíveis para visualização e/ou manipulação varia de acordo com o objeto selecionado e é controlado pelo Gerenciador de Atributos. Alguns atributos são específicos enquanto outros, mais gerais, são utilizados para todos os objetos. Por exemplo, o atributo *Info* fornece informações sobre o número de pontos, triângulos, etc. que constituem o objeto. Nenhuma função de manipulação é oferecida. O atributo *Graphic* controla propriedades como a cor do objeto, a especificação do índice de opacidade, a visualização de sua topologia sob forma de malha vazada (para entidades do tipo *Surface*), etc. O atributo *Property* permite definir escalas de cores de acordo com a variação das coordenadas X, Y ou Z e sombrear o objeto em estudo adequadamente. Por fim, através

do atributo *Contours*, é possível construir e visualizar conjuntos de isolinhas a serem traçadas “sobre” as entidades visualizadas.

Todos os objetos podem ser lidos do disco a partir de arquivos em formato ASCII. Entidades constituídas por grande quantidade de informações geométricas podem ser armazenadas em arquivos binários para obter economia de armazenamento em disco (é o caso dos *Voxets*).

O programa GOCAD representa uma poderosa ferramenta de modelagem e visualização 3D. Possui numerosas funcionalidades de manipulação de dados, sendo possível modificar interativamente os objetos visualizados. Destaca-se o recorte, a decimação de superfícies (redução do número de triângulos) e a modificação de uma superfície de acordo com um conjunto de pontos de controle (vínculos ou *constrains*), através de algoritmos de topologia matemática (MALLET, 1989). Outro algoritmo de computação gráfica 3D utilizado envolve a visualização de dados volumétricos sob forma de nuvem semitransparente (*volume rendering*, Figura 3.4), bem como a sobreposição de imagens (mapas temáticos) sobre Modelos Digitais de Terrenos (MDTs).

Resultados preliminares publicados da aplicação do GOCAD em modelagem 3D de dados geológicos no Brasil têm abordado superfícies estratigráficas e canais de reservatórios turbidíticos (LANZARINI & GUIMARÃES, 2000), modelos físicos/tectônicos experimentais de estruturas halocinéticas (GARCIA, 1999), a paleobatimetria da Bacia de Campos (EBERT & LAVORANTE, 2000), a modelagem 3D das bacias do Recôncavo (MENDONÇA et al., 2001; ANTUNES et al., 2004) e de Camamu-Almada (ANTUNES et al., 2004).

Nesta dissertação serão descritos e apresentados os resultados da utilização de diferentes aplicativos computacionais para modelar dados geológicos. Inicialmente será descrita uma ferramenta de modelagem geométrica 3D para realizar a visualização de objetos deformados de acordo com informações tensoriais (seção 5). Em seguida, será mostrado um exemplo de integração de aplicativos de código aberto para montar um ambiente de visualização de dados geológicos, visando propor uma infra-estrutura de trabalho acessível para ambientes de pesquisa (seção 6). Por fim, será descrito o processo de modelagem e visualização da paleobatimetria das bacias de Santos e Campos utilizando o programa comercial GOCAD (seção 7).

As atividades descritas representam um exemplo de interação entre aplicativos de modelagem computacional heterogêneos junto ao Laboratório de GeoModelagem 3D e demonstram a importância de se manter frentes de trabalho distintas mas integradas entre si.

4 Área de Estudo

Nos últimos anos, as bacias sedimentares ao longo das margens continentais do Atlântico Sul têm despertado um grande interesse quanto às atividades de exploração de petróleo (Figura 4.1). Isto foi ocasionado pela descoberta de campos gigantes em ambas as margens continentais.

As bacias das margens sul-americana e africana têm sido tradicionalmente consideradas independentes. Porém, a expansão e unificação das bases de conhecimento permitiram considerar ambas as margens e suas bacias como parte de um único sistema regional maior, levando em conta a evolução estrutural, tectônica e estratigráfica das estruturas investigadas (MOHRIAK, 2003; Figura 4.2). Estas similaridades decorrem do fato das bacias de ambas as margens terem sofrido evolução tectônica semelhante, passando pelas fases rifte, transicional e de margem passiva durante a abertura e desenvolvimento do Atlântico Sul. (Figuras 4.3 e 4.4).

4.1 Contexto regional

As bacias de Santos e Campos (Figura 4.5), enfocadas neste trabalho, são parte integrante das bacias sedimentares situadas na margem continental que compõe o Sistema Rifte do Leste Brasileiro. Trata-se de bacias de margem continental passiva, resultado da separação da América do Sul e da África durante o Cretáceo. Suas colunas estratigráficas permitem obter informações quanto à evolução tectônica e paleoambiental das bacias. Desta forma, auxiliam na identificação de pulsos tectônicos e deposicionais ocorridos durante a abertura e evolução do Atlântico Sul, proporcionando importantes indicações sobre a paleogeografia deste ambiente ao longo do Cretáceo.

Dentre os autores que estudaram a formação e expansão do Atlântico Sul, Nürnberg & Müller (1991) realizaram uma reconstrução completa de sua evolução tectônica do final do Jurássico até o presente. O modelo assume placas não-rígidas para a América do Sul e a África, descrevendo o gradativo processo de rifteamento entre as mesmas. Para tanto foram utilizados dados de anomalias magnéticas e altimétricos obtidos por satélite (Figuras 4.6a e 4.6b). Schettino & Scotese (2000) desenvolveram um programa computacional que permite gerar reconstruções paleogeográficas usando uma interface disponível na rede Internet (PLATE TECTONICS ONLINE RECONSTRUCTION TOOL, 2001), a partir de idades especificadas pelo usuário. As Figuras 4.7a e 4.7b mostram uma seqüência de mapas gerados para idades contidas no Cretáceo. Sykes et al. (1998) realizaram uma modelagem

computacional da paleobatimetria do hemisfério sul do mesocretáceo (110 Ma) até o presente. A modelagem foi obtida utilizando um banco de dados contendo informações de batimetria atual, idades da crosta oceânica e espessura do pacote sedimentar. Estas informações foram levantadas realizando uma compilação de dados geofísicos. Os mapas gerados modelam a paleobatimetria na ausência de sedimentos e foram criados a intervalos regulares de 5 Ma. Os autores verificaram a fidelidade do modelo utilizando informações de batimetria atual, idades da crosta oceânica e espessura de camadas sedimentares obtidas de 39 poços do DSDP e ODP (*Deep Sea Drilling Project* e *Ocean Drilling Project*; referências citadas no trabalho). A Figura 4.8 mostra alguns exemplos de idades modeladas.

4.2 Geologia das bacias de Campos e Santos

Bacia de Campos

A Bacia de Campos situa-se na costa norte do Estado do Rio de Janeiro e sul do Espírito Santo, entre os paralelos 21° e 23° sul (Figura 4.9). Seu limite norte com a Bacia do Espírito Santo ocorre no Alto de Vitória, e o limite sul com a Bacia de Santos é no Alto de Cabo Frio.

A sequência basal (Figuras 4.10 e 4.12) é representada por derrames basálticos intercalados com conglomerados cinza-esverdeados eocretáceos, denominados Formação Cabiúnas (MIZUSAKI et al., 1988 apud RANGEL et al., 1994).

A Formação Lagoa Feia, dividida em quatro grandes ciclos de sedimentação, é representada por sedimentos de planície aluvial ou margem de lago, constituída por conglomerados, arenitos líticos, siltitos, folhelhos e calcários, recobertos por evaporitos (anidrita e halita), seguidos de fácies transgressivas no topo, representadas por folhelhos lacustres (CASTRO & AZAMBUJA, 1980 apud RANGEL et al., 1994), depositados durante o Barremiano-Aptiano.

A Formação Macaé, depositada entre o Albiano e o eoturoniano, é constituída de calcirruditos, calcarenitos e calcilutitos. Essa formação representa o início da passagem de um ambiente marinho restrito para marinho franco, iniciando-se como plataformaraso, chegando a batial superior, quando num contexto de mar epicontinental (RANGEL et al., 1994).

O Grupo Campos (SCHALLER, 1973 apud RANGEL et al., 1994) apresenta as seguintes formações: Ubatuba, constituída por folhelhos cinza-escuros e esverdeados, argilitos e margas cinza-claras, calcilutitos cinzas e cremes e diamictitos, depositada em ambiente francamente marinho batial e abissal. Formação Carapebus, constituída de camadas

de arenito fino a conglomerático, resultantes da atuação de correntes de turbidez numa situação paleogeográfica de talude-bacia. Formação Emboré, constituída por arenitos e carbonatos impuros, depositados em ambiente flúvio-deltaíco e plataformal.

Esta bacia, assim como a Bacia de Santos, pertence ao Sistema Rifte Leste Brasileiro, que se desenvolveu durante a ruptura do Gondwana, gerando duas margens distintas: a margem norte equatorial desenvolvida em resposta a movimentação “strike-slip” entre América do Sul e África, proporcionando a formação de complexas bacias dominadas por cisalhamento, e a margem leste, iniciando como rift e evoluindo para uma bacia tipo margem passiva em consequência da extensão crustal ortogonal (CHANG et al., 1992).

Bacia de Santos

A Bacia de Santos situa-se entre os paralelos 23° e 28° sul e ocupa cerca de 275.000 km² até a cota batimétrica de 2.000m (Figura 4.9). Seu limite norte com a Bacia de Campos se dá no Alto de Cabo Frio, e o limite sul com a Bacia de Pelotas se dá na Plataforma de Florianópolis.

A Formação Camboriú (OJEDA & ARANHA, 1980) abrange os derrames basálticos eocretáceos sotopostos ao preenchimento sedimentar de praticamente toda a Bacia de Santos (Figuras 4.11 e 4.13). No Barremiano e eoaptiano ocorreu uma deposição em ambiente continental, provavelmente sob a forma de leques aluviais progradantes sobre lagos rasos, representada pela Formação Guaratiba (OJEDA & CESERO, 1973). Esta unidade se caracteriza por conglomerados de matriz argilo-arenosa, arenitos médios a conglomeráticos imaturos e coquinas brancas. A Formação Ariri representa ambiente de deposição marinha restrito, com a formação de planícies de *sabkha*, e se caracteriza por espessos pacotes de halitas, associados a anidritas, calcilutitos, folhelhos e margas.

Os depósitos de leques aluviais albianos são representados pela Formação Florianópolis, constituída por arenitos finos a grossos, folhelhos e siltitos (OJEDA & CESERO, 1973 apud PEREIRA & FEIJÓ, 1994). A Formação Guarujá é constituída de calcarenitos oolíticos gradando lateralmente para calcilutitos, e apresenta um paleoambiente de planície de maré, laguna rasa, plataforma externa e mar aberto, de idade eoalbiana (PEREIRA et al., 1986). O ambiente plataformal raso a profundo e raramente batial superior, é representado pela Formação Itanhaém, e se caracteriza por folhelhos, siltitos, margas, calcilutitos e arenitos (OJEDA & ARANHA, 1980 apud PEREIRA & FEIJÓ, 1994), de idade neo-albiana.

O ambiente deposicional continental-transicional, na forma de leques aluviais, rios entrelaçados e deltas (PEREIRA et al., 1986) é representado pela Formação Santos, de idade cenomaniana-maastrichtiana, e é composto por conglomerados e arenitos líticos, intercalados com folhelhos e argilas. A Formação Juréia é constituída por folhelhos, siltitos, arenitos finos e muito finos e calcilutitos, e reflete um ambiente deposicional de plataforma marinha, com idades santoniana a maastrichtiana. A Formação Itajaí-Açu se caracteriza por uma espessa seção de clásticos finos (folhelhos), em ambiente deposicional de talude, com idade cenomaniana a maastrichtiana (OJEDA & CESERO, 1973).

O ambiente de plataforma carbonática, com influência de leques aluviais nas áreas mais proximais (PEREIRA et al., 1986), é representado pela Formação Iguape, de idade terciária, constituída por calcarenitos e calcirruditos bioclásticos. A Formação Marambaia representa uma deposição de taludes durante o Terciário e é composta por uma espessa seção de folhelhos e margas, com intercalação de arenitos finos turbidíticos. No Quaternário, representado pela Formação Sepetiba, ocorreu deposição em leques costeiros, constituídos de areias grossas a finas, quartzosas, feldspáticas, glauconíticas, e coquinas de moluscos, briozoários e foraminíferos.

A Figura 4.14 mostra a distribuição dos litotipos predominantes durante a evolução geológica da margem continental sudeste brasileira no Cretáceo. Os mapas foram extraídos do trabalho de França et al. (1995), que realiza uma integração de dados geológicos inéditos pertencentes a diferentes companhias petrolíferas da América do Sul. Os mapas refletem o paleoambiente e a paleogeografia da época. A região mostrada contém as bacias de Campos e Santos.

Já Modica e Brush (2004) elaboraram um modelo litoestratigráfico e paleogeográfico para a Bacia de Santos, do mesocretáceo ao Terciário, visando melhorar o entendimento da história de seu preenchimento sedimentar. Para tanto foi utilizado extenso conjunto de dados sísmicos 2D. A interpretação resultante foi verificada através de dados bioestratigráficos. O trabalho divide a história deposicional da bacia em 11 seqüências. As Figuras 4.15 a 4.17 ilustram a evolução da Bacia de Santos durante o meso-Neocretáceo.

Parte desta dissertação ilustra a modelagem 3D da paleobatimetria das bacias de Campos e Santos a partir de mapas de paleoambientes 2D obtidos da literatura (AZEVEDO et al., 1987; VIVIERS, 1986). Estas informações, de natureza micropaleontológica, foram integradas aos demais mapas descritos nos parágrafos anteriores. Foram também utilizados dados de outros trabalhos, visando construir um ambiente unificado de modelagem e

visualização 3D da evolução das bacias de Campos e Santos durante o meso-Neocretáceo. Os resultados serão mostrados na seção 7.



Figura 4.1 – Bacias sedimentares do Atlântico Sul. Fonte: Mohriak (2003).

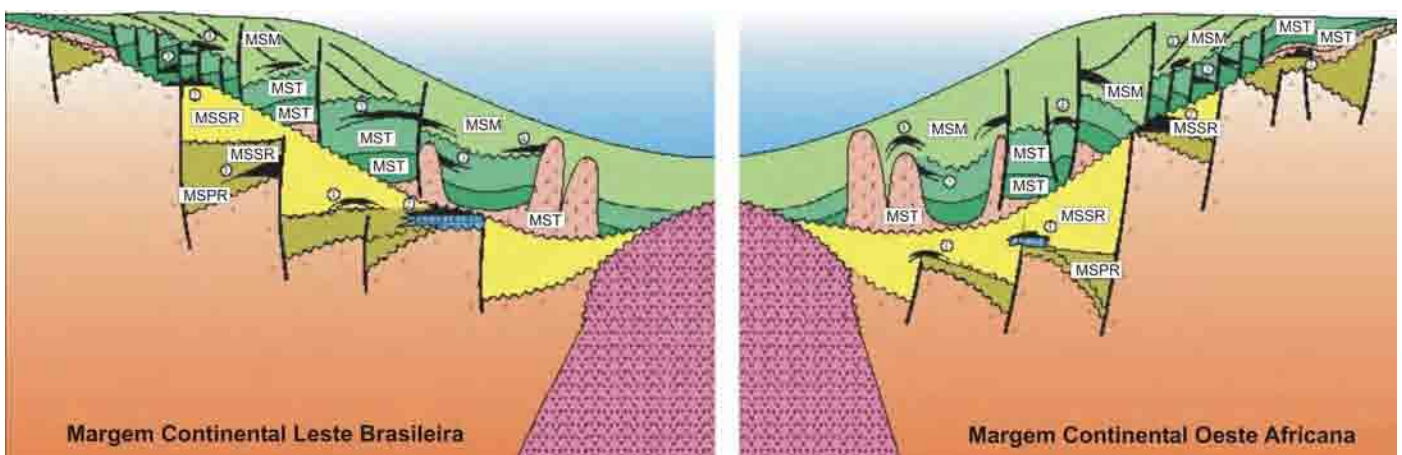


Figura 4.2 – Seção geológica comparando a margem continental leste brasileira e a oeste africana. MSPR: Megasequência Pré-Rifte; MSSR: Megasequência Sin-Rifte; MST: Megasequência Transicional; MSM: Megasequência Marinha. Fonte: Mohriak (2003).

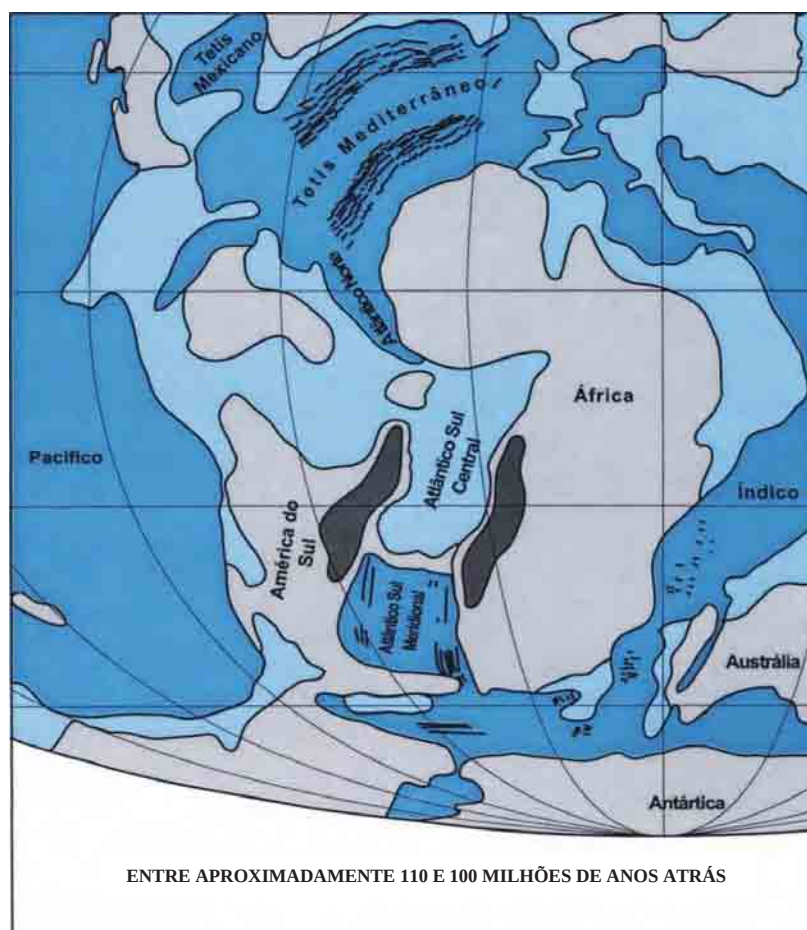


Figura 4.3 – Reconstituição paleogeográfica para o Albiano. Em cor cinza escuro, as montanhas em torno do Atlântico Sul Central; em cor cinza, as áreas continentais; em cor azul claro, os mares rasos; em azul escuro, os oceanos ou mares mais profundos. Fonte: Azevedo (2001).

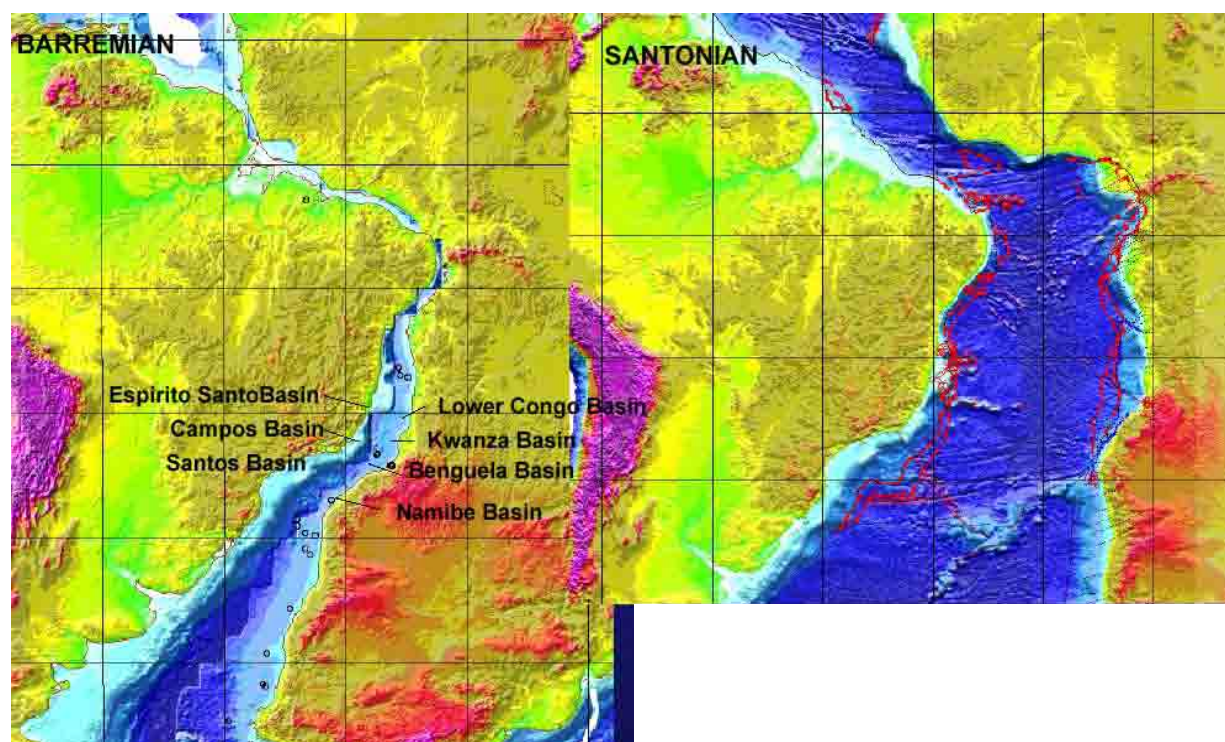


Fig. 4.4 - Reconstituição paleogeográfica das placas sul-americana e africana para o Barremiano (125-119 Ma) e Santoniano (87,5-83 Ma). Fonte: Gjølberg (2003).

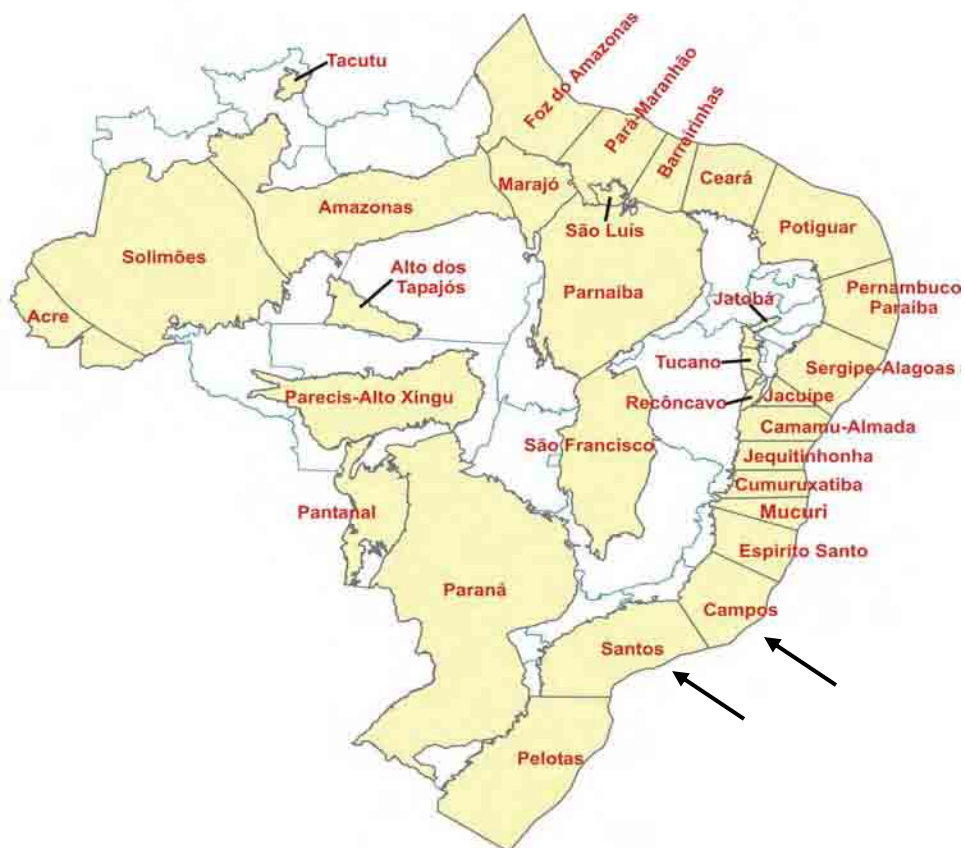


Figura 4.5 – Localização das bacias de Santos e Campos no contexto das bacias sedimentares fanerozóicas do Brasil. Fonte: BDEP/ANP (2004).

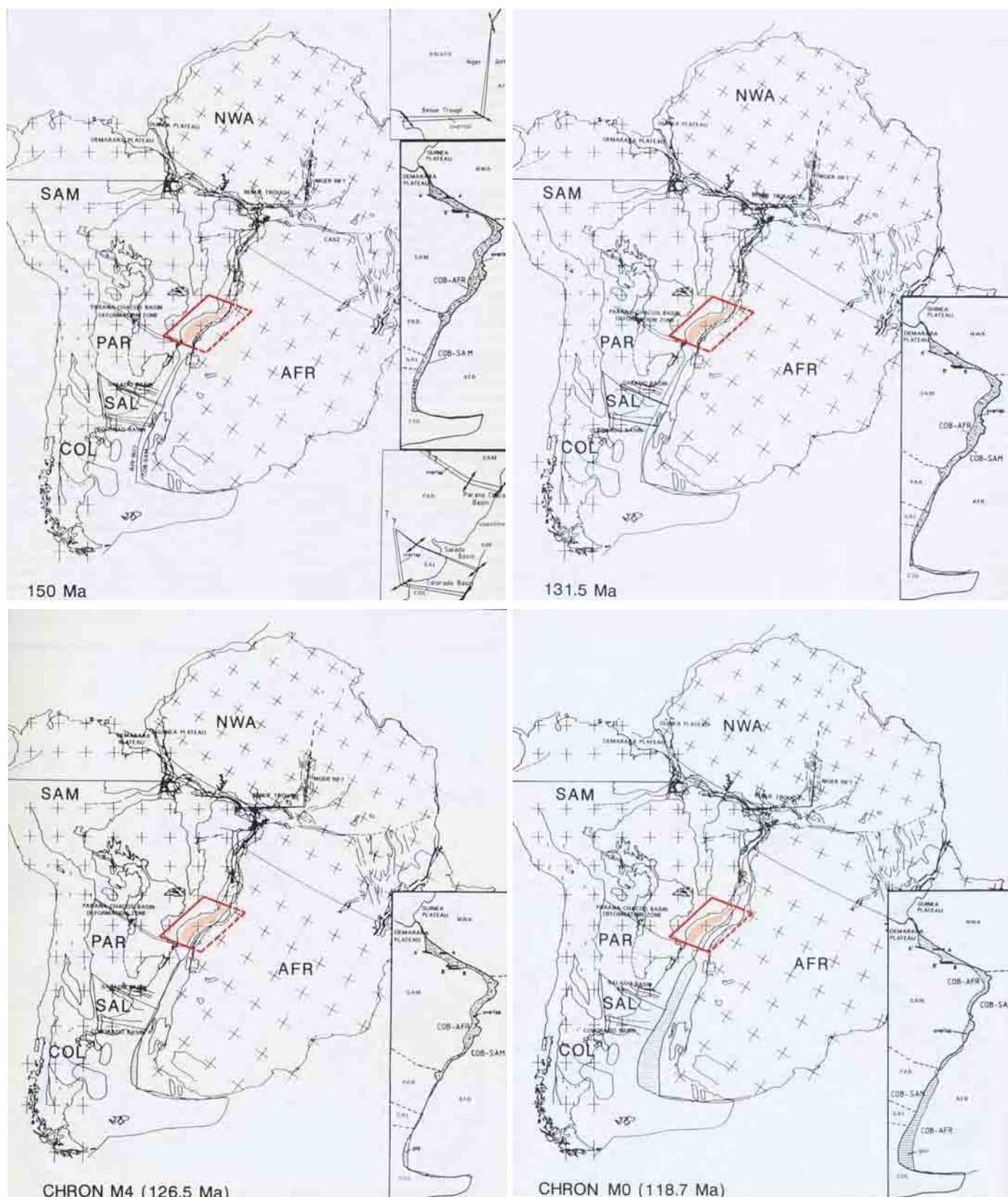


Figura 4.6a – Reconstrução paleogeográfica da abertura e expansão do Atlântico Sul (150-118,7 Ma). O polígono vermelho delimita a área deste estudo e a região em laranja denota a localização da paleobatimetria das bacias de Santos e Campos. As Figuras menores mostram as regiões de sobreposição (*overlap*, hachura pontilhada) e espaçamento (*gap*, hachura em linhas) entre margens continentais ou bordas de placas. Legenda: AFR = África meridional, NWA = África norte-ocidental, SAM = América do Sul, PAR = Placa do Paraná, SAL = Placa do Salado, COL = Placa do Colorado. Fonte: Nürnberg & Müller (1991).

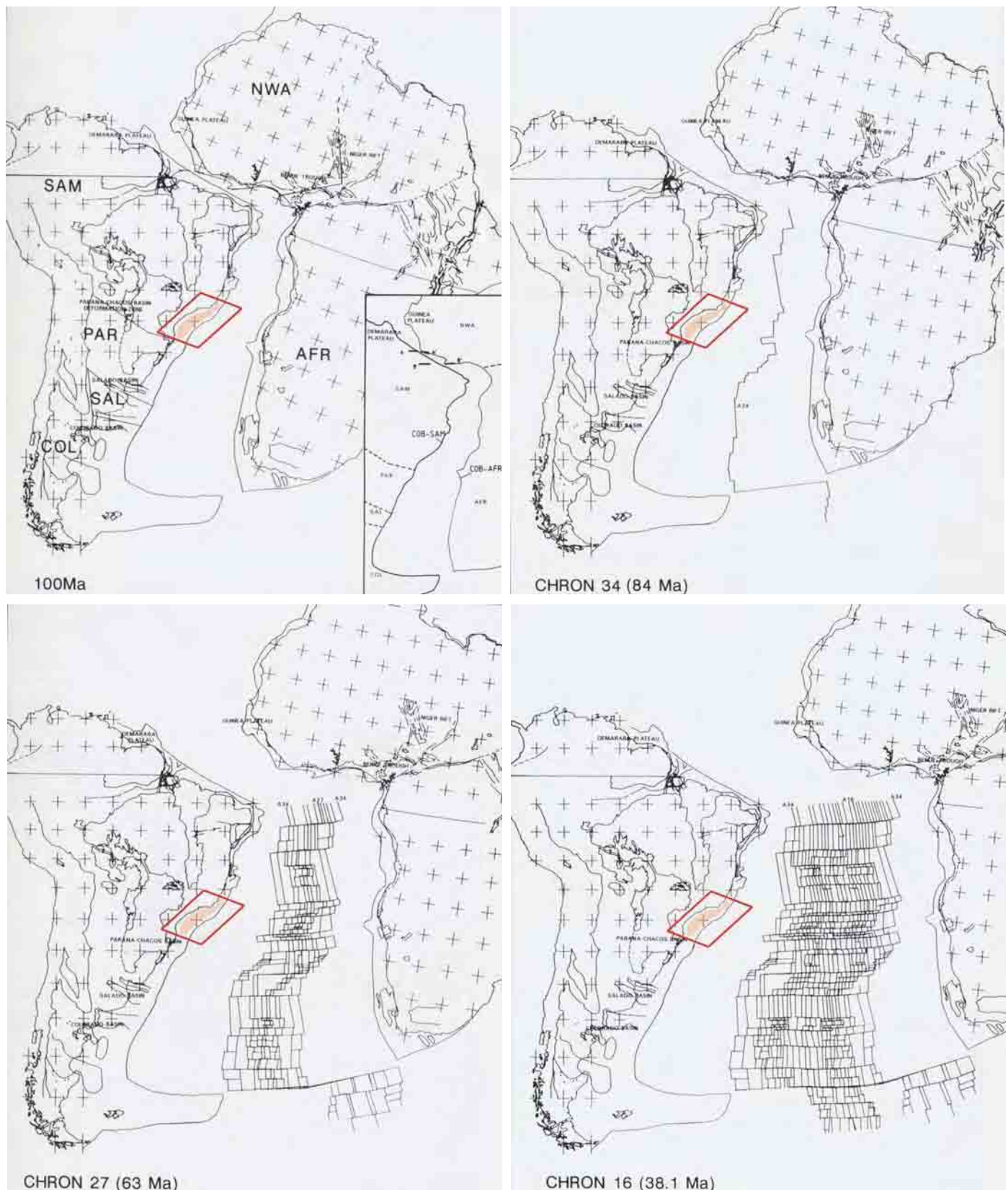


Figura 4.6b – Reconstrução paleogeográfica da expansão do Atlântico Sul entre 100 e 38,1 Ma. A linha externa representa o limite da crosta continental e a linha interna o atual limite de costa como referência. Fonte: Nürnberg & Müller (1991).

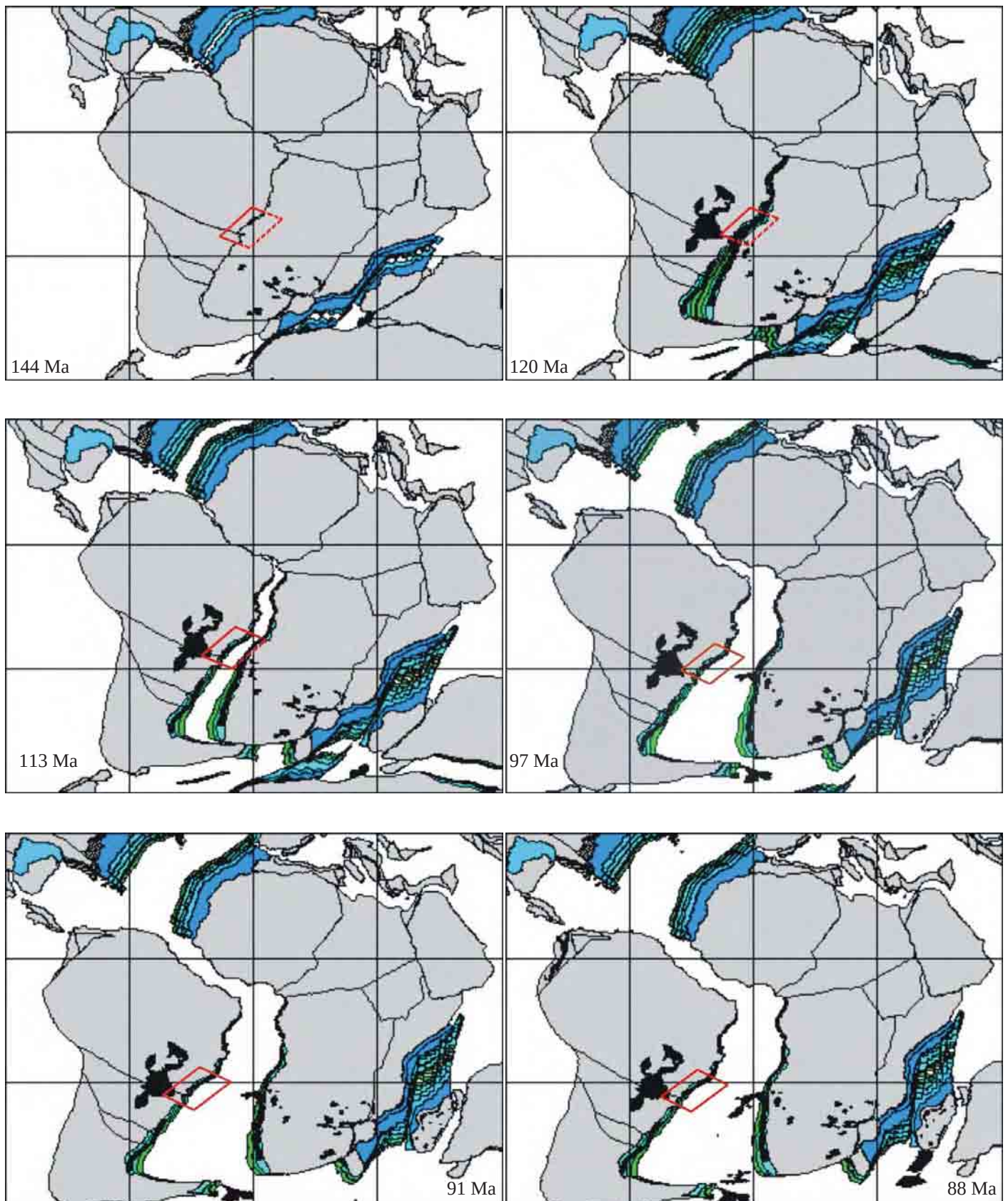


Figura 4.7a – Reconstrução paleogeográfica da abertura e expansão do Atlântico Sul durante o eo-mesocretáceo. O polígono vermelho delimita a área de estudo desta dissertação. Legenda na próxima Figura. Fonte: Schettino & Scotese (2000).

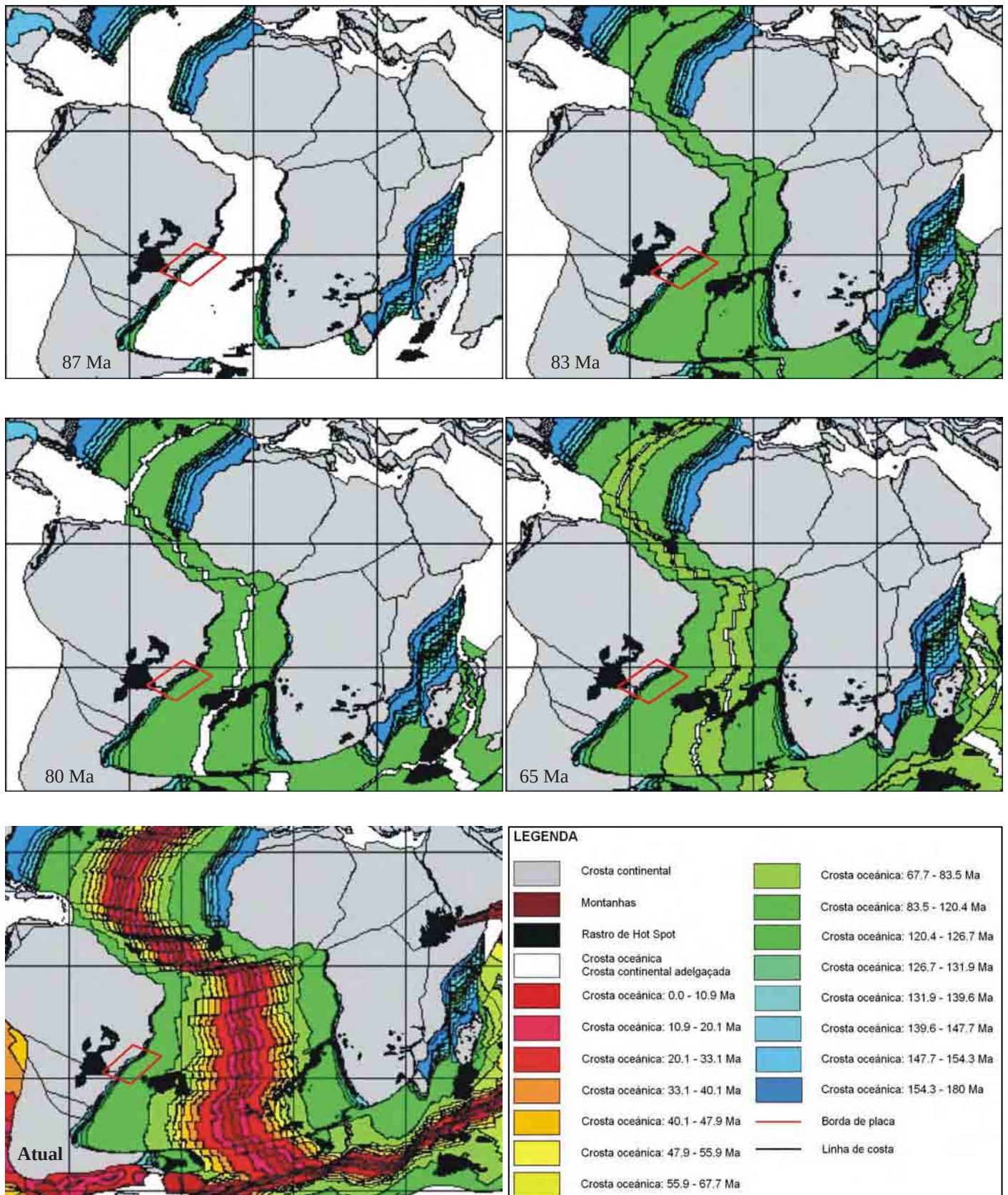


Figura 4.7b – Reconstrução paleogeográfica da expansão do Atlântico Sul durante o Neocretáceo. A última figura representa a situação atual. Fonte: Schettino & Scotese (2000).

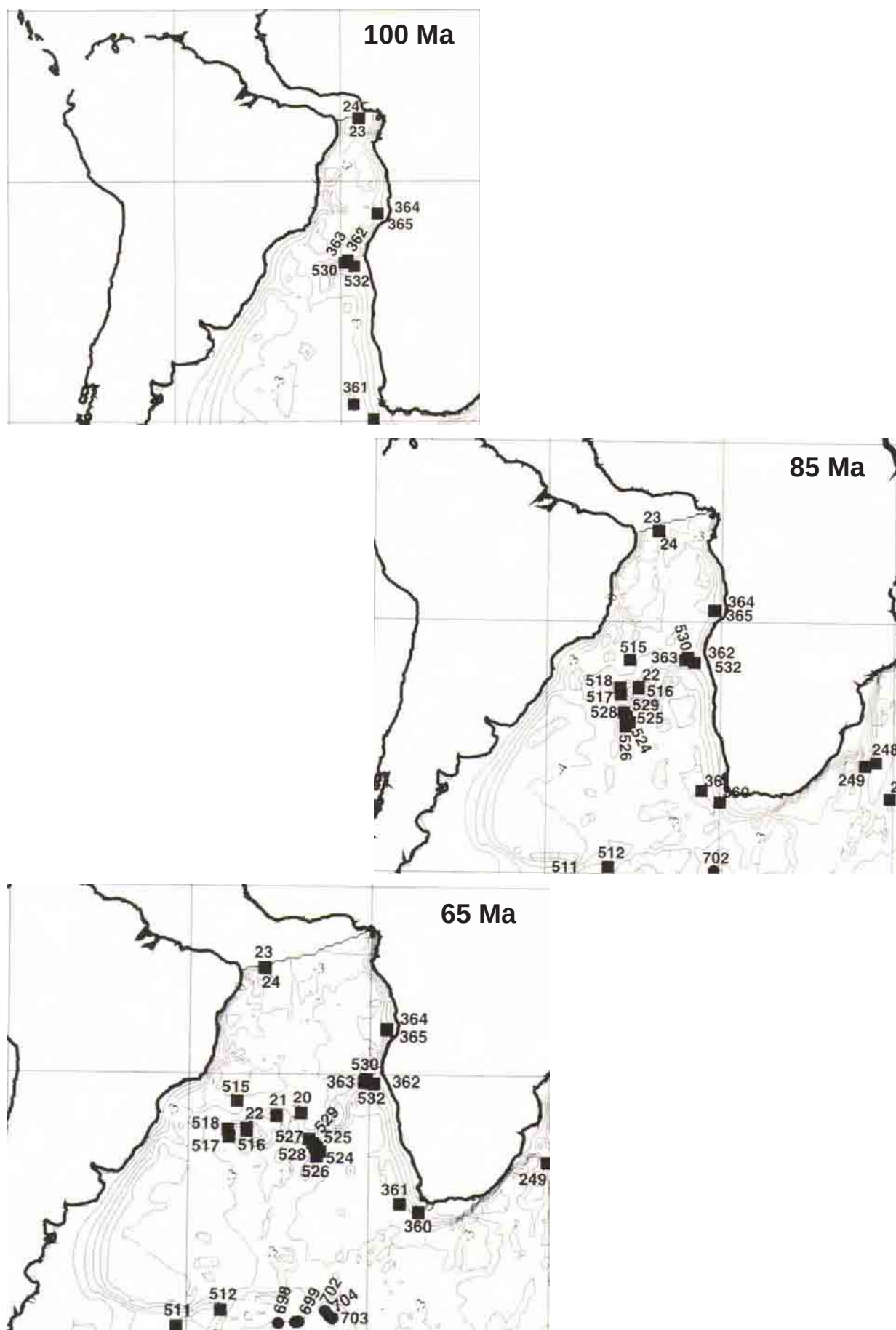


Figura 4.8 – Detalhe de reconstrução paleobatimétrica do hemisfério sul durante o meso-Neocretáceo (100, 85 e 65 Ma). A modelagem computacional não leva em conta a deposição de sedimentos. É também exibida a localização de poços dos projetos DSDP/ODP. Fonte: Sykes et al. (1998).

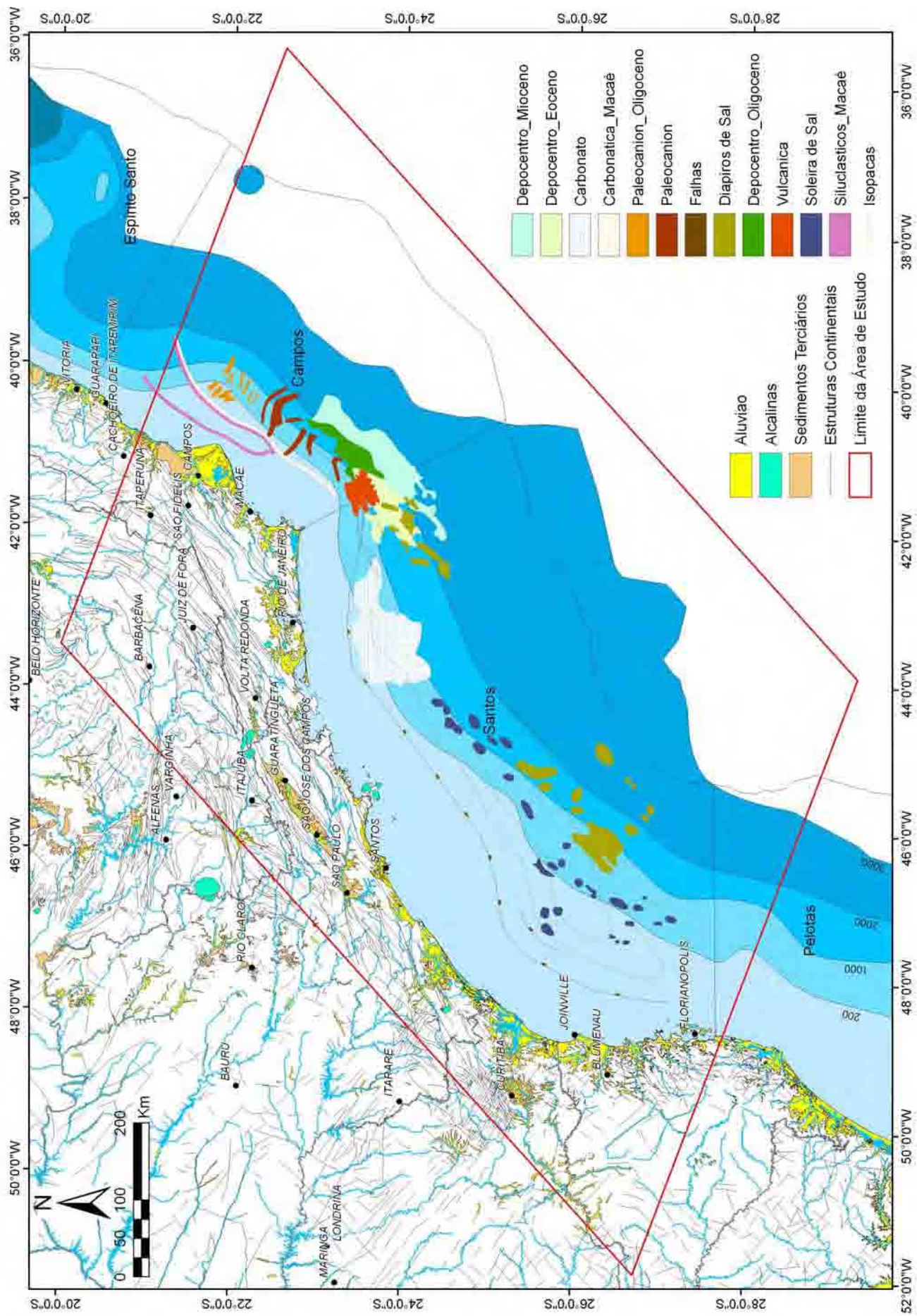


Figura 4.9 – Localização da área de estudo (polígono vermelho). São também exibidas as estruturas oceânicas, continentais, litotipos cenozóicos, rede de drenagens, informações batimétricas, bem como limites das bacias de Espírito Santo, Campos, Santos e Pelotas.

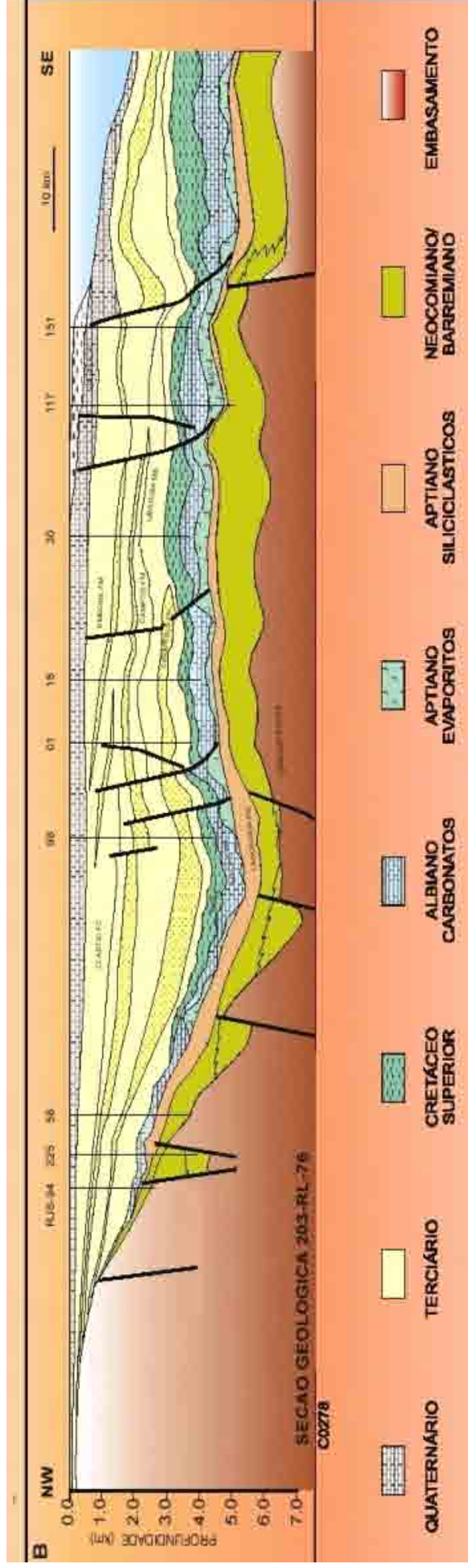


Figura 4.10 – Seção geológica esquemática da Bacia de Campos. Fonte: Mohriak et al. (1990).

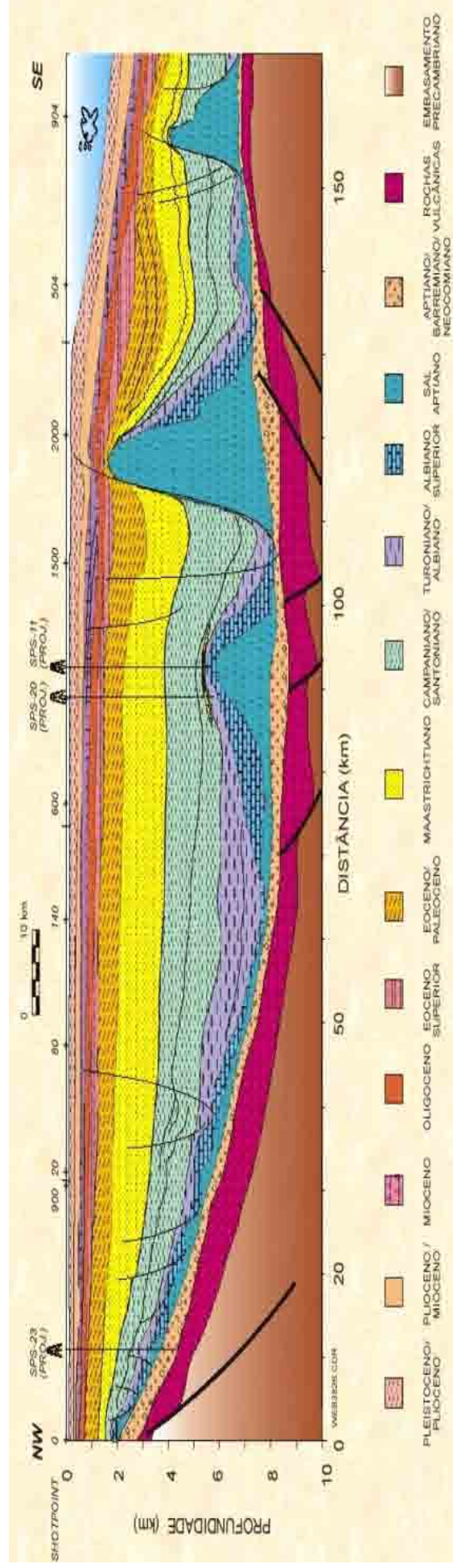


Figura 4.11 – Seção geológica esquemática da Bacia de Santos. Fonte: Cainelli & Mohriak (1998).

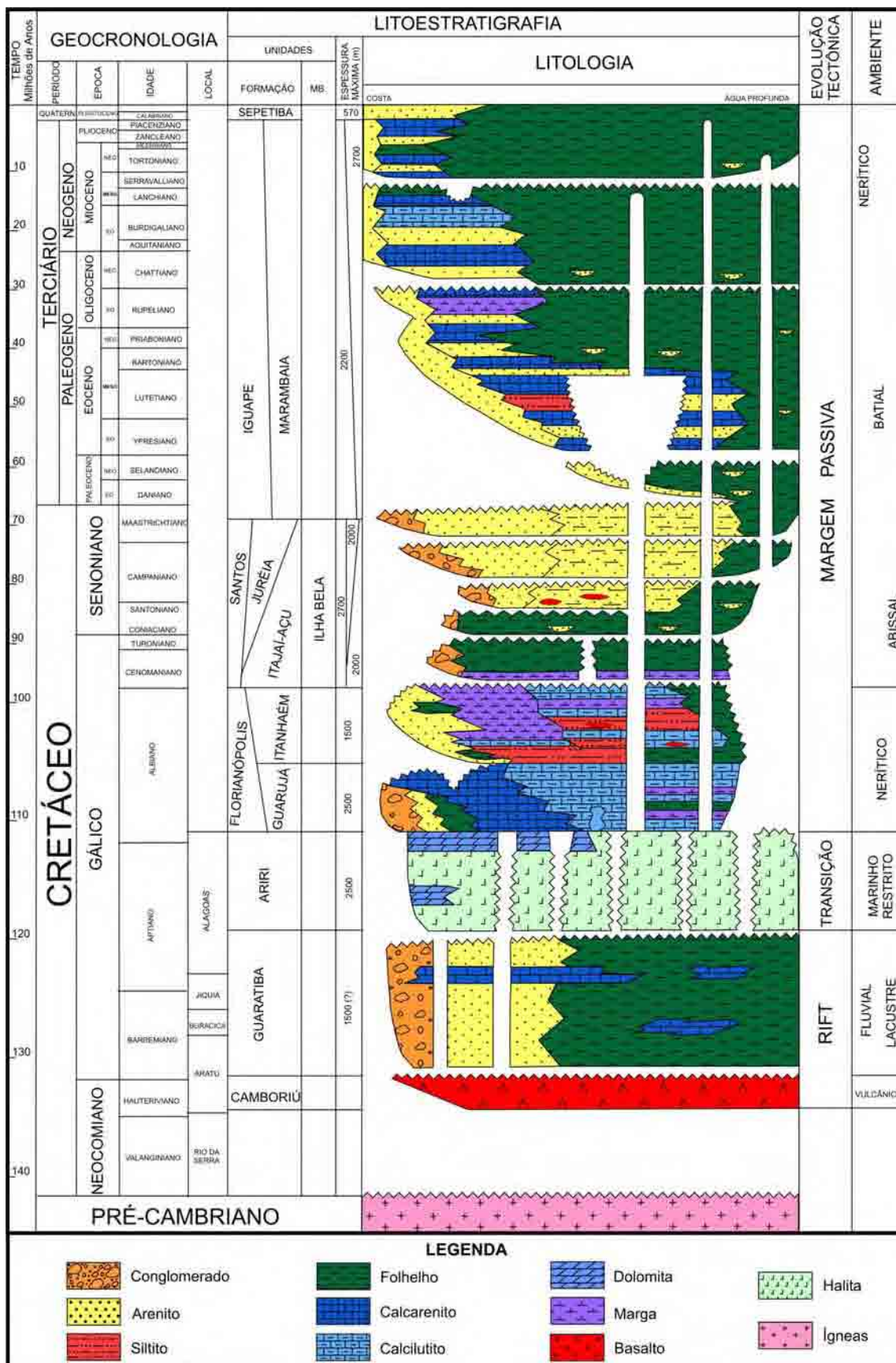


Figura 4.13 - Carta estratigráfica da bacia de Santos. Fonte: Pereira & Feijó (1994).

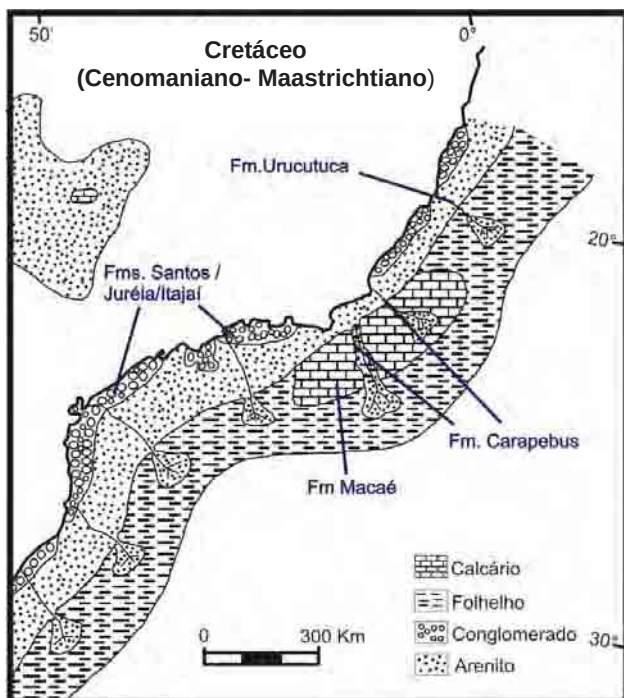
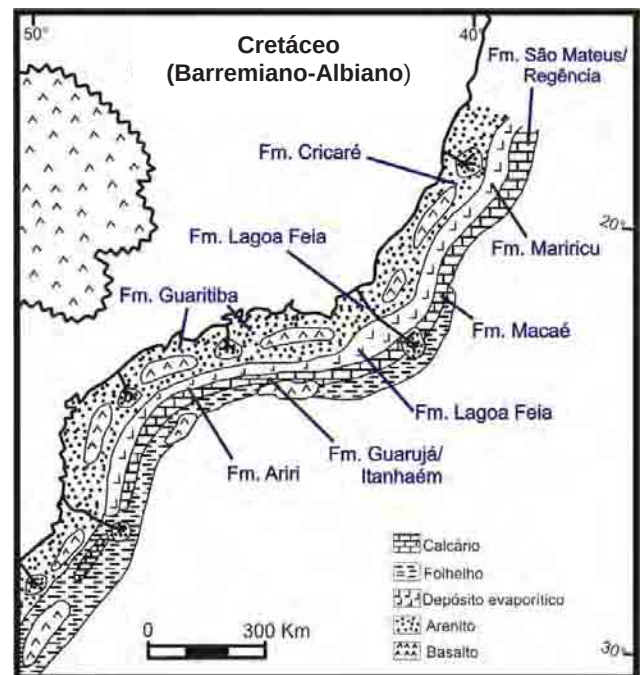
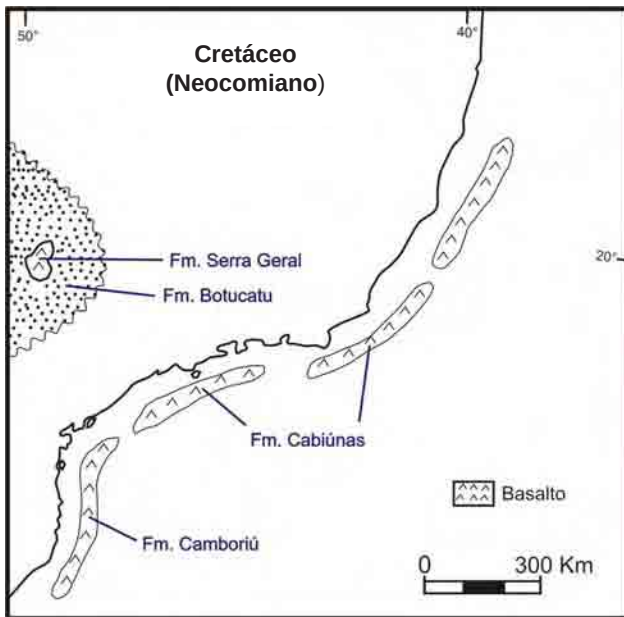
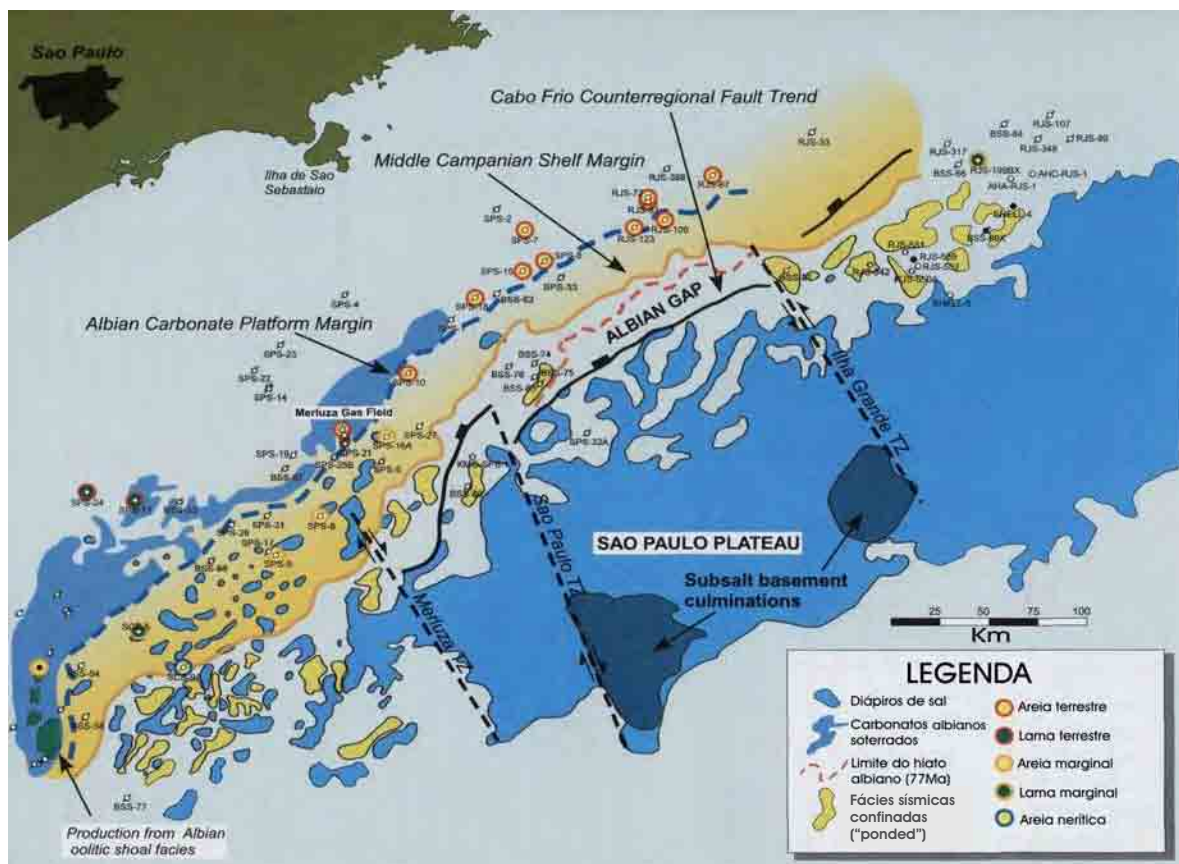
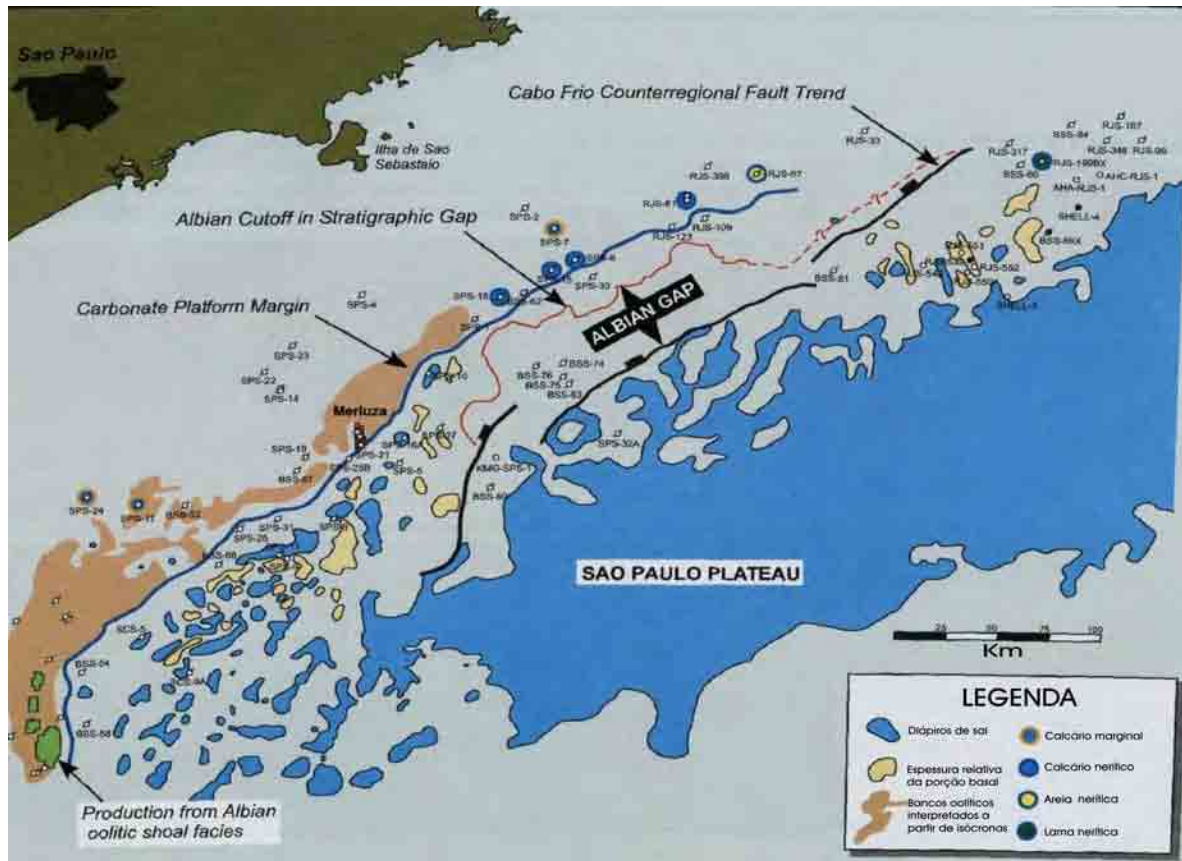


Fig. 4.14 – Mapas geológicos ilustrando a evolução cretácica da margem continental sudeste brasileira.
Fonte: França et al. (1995).



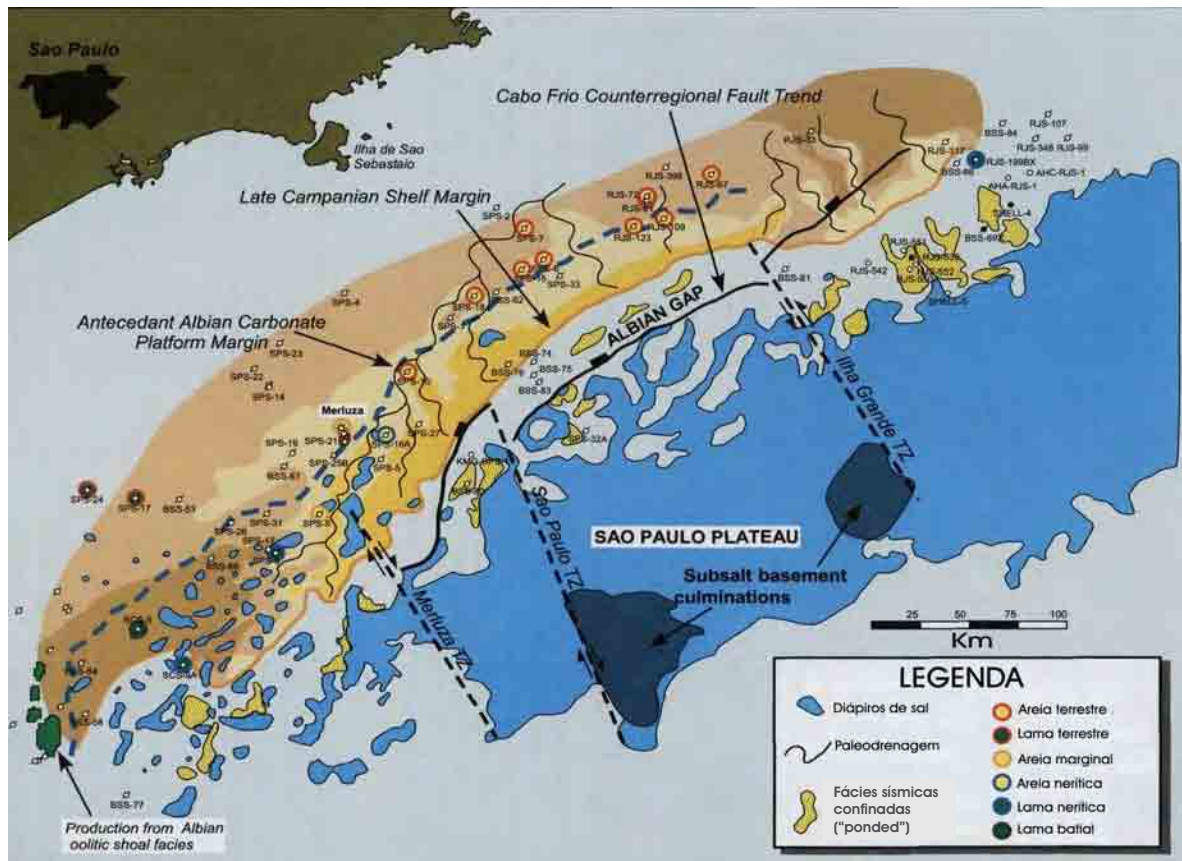


Fig. 4.17 – Bacia de Santos: litologia e paleobatimetria durante o neocampaniano-Maastrichtiano (76-65 Ma). Neste período iniciou-se uma concentração de sedimentos siliciclásticos nas regiões Norte e central da bacia. Este processo está relacionado à organização do sistema do rio Paraíba do Sul (paleodrenagens da região superior). Os tons de marrom denotam as diferentes espessuras da paleoplataforma: marrom escuro = fino, marrom-laranja = espesso. Fonte: Modica & Brush (2004).

5 *Tensor3D*: um programa para simulação de tensores em três dimensões

A deformação natural de rochas na crosta terrestre, não apenas em níveis crustais inferiores, como os expostos nos cinturões orogênicos, mas também em bacias sedimentares invertidas, é em geral de caráter tridimensional. Para facilitar o entendimento e a descrição da deformação em atividades didáticas ou de pesquisa, recorre-se usualmente a definições bidimensionais, como cisalhamento simples e cisalhamento puro. Estas, no entanto, são uma simplificação da realidade e freqüentemente não explicam a complexidade geométrica e cinemática das estruturas dúcteis registradas em rochas deformadas.

O estudo da deformação tridimensional de rochas é também importante para a indústria do petróleo, para abordar problemas como a análise da deformação das bacias sedimentares, a caracterização de reservatórios e de trapas estruturais, etc.

Para facilitar a análise conceitual e a representação dos componentes de cisalhamento simples e puro presentes simultaneamente em cada um dos três planos cinemáticos, foi desenvolvido um programa para simular deformações em três dimensões. Tal ferramenta visa auxiliar tanto alunos de graduação e pós-graduação, como geólogos mais experientes, explorando os benefícios da computação gráfica 3D.

Este aplicativo foi escrito utilizando a linguagem Tcl/Tk em ambiente VTK (descrito na seção 3.4). Possibilita assim a utilização das classes de modelagem geométrica e visualização científica presentes nesta biblioteca. Além disto, o uso de ferramentas de trabalho de código aberto torna esta aplicação ideal para ambientes de pesquisa e fins didáticos. A discussão que segue abordará os seguintes tópicos:

- Revisão conceitual da teoria matemática sobre os tensores de tensão e de deformação;
- Descrição da ferramenta computacional, ou programa *Tensor3D*.

5.1 Teoria dos tensores

Um tensor é uma entidade matemática que pode ser usada para descrever o estado ou as propriedades físicas de um material (HOBBS et al., 1976; TWISS & MOORES, 1992). Esta entidade é representada através de um conjunto de componentes escalares associadas a um sistema de coordenadas específico.

A *ordem* de um tensor indica quantas componentes escalares são necessárias para descrevê-lo totalmente. O número de componentes c é igual à dimensão d do espaço físico elevada ao expoente dado pela ordem r do tensor. Formalmente,

$$c = d^r \quad (5.1)$$

Por exemplo, no espaço tridimensional ($d = 3$) um *escalar* é um tensor de ordem zero ($r = 0$) e portanto possui $3^0 = 1$ componente. Exemplos típicos incluem a temperatura, a massa e o volume. Escalares são definidos simplesmente por sua magnitude e não são afetados por mudanças no sistema de coordenadas.

Um *vetor* é um tensor de ordem 1 ($r = 1$) com $3^1 = 3$ componentes no espaço tridimensional ($d = 3$). Força, velocidade e aceleração são grandezas vetoriais. Vetores são utilizados para descrever quantidades físicas caracterizadas por magnitude e apenas *uma* informação de direção. As componentes do vetor são representadas utilizando índices, como no exemplo $\mathbf{F}_k$, no qual uma força é decomposta nas componentes F_1 , F_2 e F_3 . Cada componente é paralela a um dos eixos coordenados, sendo que os índices 1, 2 e 3 correspondem aos eixos X, Y e Z, respectivamente. Em duas dimensões o índice assume apenas os valores 1 e 2. Portanto, para descrever uma grandeza vetorial é necessário prover d componentes, sendo d a dimensão (2 ou 3) do espaço considerado. O vetor $\mathbf{F}_k$ do exemplo anterior será indicado através da notação

$$\mathbf{F}_k = (F_1, F_2, F_3) \quad (5.2)$$

Um *tensor* de ordem 2 ($r = 2$) no espaço tridimensional ($d = 3$) possui $3^2 = 9$ componentes. Em geologia estrutural, os exemplos mais importantes são a tensão e a deformação. Tensores de ordem 2 são usados para descrever quantidades que possuem magnitude e que são caracterizadas por *duas* informações de direção. Por exemplo, para o tensor de tensão, as duas informações associadas a cada componente são a orientação da normal ao plano sobre o qual tal componente atua e a direção da componente de tensão que atua ao longo do plano.

Tensores de segunda ordem, como a componente de tensão σ_{kl} , são representados através de um símbolo com dois índices. Para o espaço tridimensional, tanto k como l assumem os valores 1, 2, ou 3. Desta forma, para cada valor que k pode assumir, l varia entre 1 e 3. Isto proporciona $3 * 3 = 9$ componentes distintas. Em duas dimensões, k e l podem assumir apenas os valores 1 e 2. Como será descrito a seguir, um tensor de ordem maior do que um é representado utilizando uma notação matricial.

5.2 Notação para o tensor de tensão

Para representar um tensor de tensão no espaço 3D é necessário identificar cada uma das nove componentes que constituem esta grandeza física. Para tanto, os eixos coordenados

considerados são X_1 , X_2 e X_3 , paralelos, respectivamente, aos eixos X , Y e Z . Para cada componente de tensão σ_{kl} , o primeiro índice k é o número do eixo coordenado perpendicular ao plano sobre o qual atua a componente em questão. Por outro lado, o segundo índice l denota o eixo coordenado paralelo à componente de tensão. Por exemplo, a componente σ_{13} é uma tensão cisalhante que age sobre o plano normal ao eixo X_1 (plano X_2X_3 , primeiro índice) e é paralela ao eixo X_3 (segundo índice). A componente σ_{11} representa uma tensão normal que atua sobre o plano normal ao eixo X_1 e é paralela ao mesmo eixo. Trata-se de uma tensão normal, sendo que sua componente cisalhante é nula. Com efeito, as componentes de tensão normal possuem dois índices iguais, ao passo que na tensão cisalhante os índices são distintos.

As componentes de um tensor de tensão são escritas em uma ordem específica para formar uma matriz:

$$\sigma_{kl} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix} \quad (5.3)$$

Nesta representação, os elementos de cada linha indicam as componentes de tensão ao longo de um mesmo plano e na direção dos eixos coordenados. O primeiro índice, que denota o plano de aplicação da tensão, mantém-se fixo. Por outro lado, os elementos de cada coluna indicam as componentes de tensão ao longo de uma direção fixa (representada pelo segundo índice) paralela a um dos eixos coordenados, mas que atuam sobre os três planos coordenados X_2X_3 , X_1X_3 e X_1X_2 . A Figura 5.1 mostra uma representação gráfica de cada componente do tensor de tensão. Os eixos coordenados X_1 , X_2 e X_3 são representados utilizando os símbolos x , y e z .

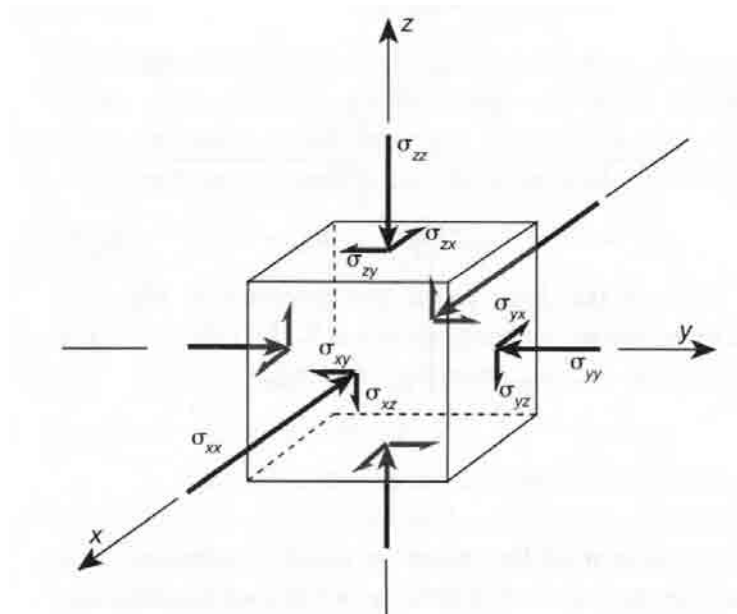


Figura 5.1 – Componentes do tensor de tensão. Fonte: Twiss & Moores (1992).

5.3 Notação para o tensor de deformação

Considerações análogas valem para a definição da deformação de um corpo, isto é, modificações em sua forma e tamanho. Para tanto, utiliza-se um tensor de segunda ordem denominado tensor de deformação.

Para um sistema de coordenadas 3D ortogonal (X_1, X_2, X_3), a *extensão* de um segmento de comprimento inicial L_1 é

$$e_{11} = \frac{\ell_1 - L_1}{L_1} = \frac{\Delta L_1}{L_1} \quad (5.4)$$

sendo que o primeiro índice de e_{11} denota que o segmento é inicialmente paralelo a X_1 , enquanto o segundo índice denota que a mudança de comprimento é também paralela a X_1 . ΔL_1 é a diferença entre o comprimento final (ℓ_1) e inicial do segmento. Relações semelhantes valem para segmentos inicialmente paralelos a X_2 e X_3 , respectivamente (Figura 5.2a).

Para a componente cisalhante da deformação, segmentos inicialmente paralelos aos eixos X_1, X_2 e X_3 (notação maiúscula) são, após a deformação, paralelos a x_1, x_2 e x_3 (notação minúscula), respectivamente (Figura 5.2b). Por exemplo, as duas componentes de deformação cisalhante para a aresta do cubo paralela a X_1 são e_{12} e e_{13} :

$$e_{12} = \frac{1}{2} \tan \psi_{12} \quad e_{13} = \frac{1}{2} \tan \psi_{13} \quad (5.5)$$

Em cada caso, o primeiro índice denota que a deformação cisalhante envolve um segmento inicialmente paralelo ao eixo X_1 , isto é, ocorre no plano X_2X_3 normal ao eixo X_1 .

O segundo índice denota que o cisalhamento ocorre, dentro do plano considerado, nas direções positivas dos eixos X_2 e X_3 , respectivamente. Cada ângulo ψ_{12} e ψ_{13} é a diferença entre 90° e os ângulos “deformados” $x_1 \wedge x_2$ e $x_1 \wedge x_3$, respectivamente. A presença de dois índices em cada componente confirma a necessidade de se especificar *duas* informações distintas de direção, de acordo com a natureza de um tensor de segunda ordem.

Em virtude disto e da observação da Figura 5.2, verifica-se que existem nove componentes de deformação. Utilizando a notação matricial, o tensor de deformação é esquematizado como segue:

$$e_{kl} = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} \quad (5.6)$$

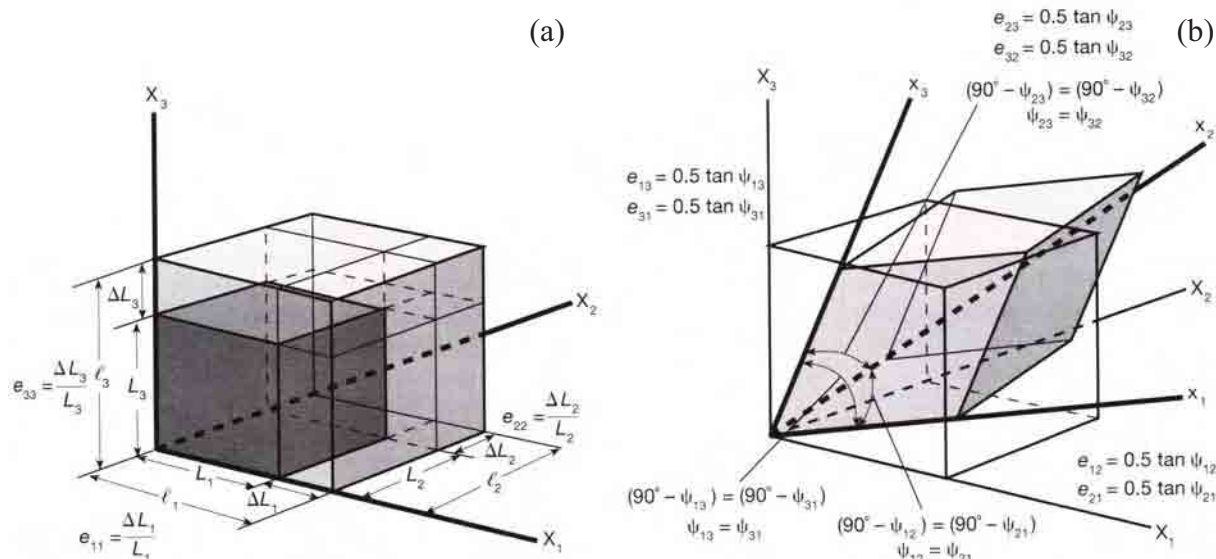


Figura 5.2 – Componentes do tensor de deformação. (a) Componente volumétrica. O cubo menor aumenta de volume em virtude da elongação de todos os seus lados. (b) Componente cisalhante. A deformação cisalhante descreve a mudança de forma do cubo. O sistema de coordenadas considerado antes da deformação é (X1, X2, X3). Após a deformação, os lados do objeto são paralelos aos eixos (x1, x2, x3). A relação entre os dois sistemas de coordenadas é definida através dos ângulos de cisalhamento ψ . Fonte: Twiss & Moores (1992).

As componentes da diagonal principal da matriz, que possuem índices iguais, são as extensões (ou deformações longitudinais, Figura 5.2a). As demais componentes, que possuem índices distintos, constituem as deformações cisalhantes (Figura 5.2b). Os elementos de uma mesma linha denotam todas as componentes de deformação que atuam sobre um plano coordenado normal ao eixo definido pelo primeiro índice. Por exemplo, as componentes e_{11} , e_{12} e e_{13} atuam sobre o plano X2X3 (ou YZ) perpendicular ao eixo X1 (X). Mas enquanto e_{11} indica *extensão* na direção X1, e_{12} e e_{13} denotam *cisalhamento* nas direções X2 e X3, respectivamente. O mesmo raciocínio vale para as demais linhas da matriz. Desta forma, o tensor de deformação proporciona informação suficiente para determinar a deformação normal e cisalhante de um corpo orientado no espaço tridimensional.

5.4 Autovalores e autovetores: eixos principais de tensão e deformação

Uma matriz 3x3 pode ser caracterizada por três vetores em 3D, denominados autovetores (*eigenvectors*); e três escalares, denominados autovalores (*eigenvalues*) da matriz (SCHROEDER et al., 1996). Os autovetores formam um sistema tridimensional de coordenadas cujos eixos são mutuamente perpendiculares. Em algumas aplicações, especialmente em engenharia e geologia estrutural, estes eixos são denominados **eixos principais** e possuem importante significado físico, como será ilustrado em breve. A cada autovetor é associado um autovalor.

Matematicamente, é possível representar autovalores e autovetores como segue. Dada a matriz 3x3 $\mathbf{A}$ correspondente a um tensor de segunda ordem, os autovetores $\mathbf{x}$ e autovalores λ devem satisfazer a relação

$$\mathbf{A} \mathbf{x} = \lambda \mathbf{x} \quad (5.7)$$

Isto ocorre se o determinante da matriz $\mathbf{A} - \lambda \mathbf{I}$ for nulo, sendo $\mathbf{I}$ a matriz identidade de ordem 3x3 (nesta matriz, todos os elementos da diagonal principal são iguais a 1, enquanto os demais elementos são nulos). Isto é:

$$\det |\mathbf{A} - \lambda \mathbf{I}| = 0 \quad (5.8)$$

A expansão desta equação produz um polinômio de terceiro grau em λ cujas raízes são os autovalores. Desta forma existem sempre três autovalores (já que o grau do polinômio é 3), embora nem sempre distintos. Após a determinação dos autovalores, os autovetores correspondentes podem ser encontrados a partir da equação 5.7.

É possível expressar os autovetores de uma matriz 3x3 da seguinte forma:

$$\mathbf{x}_i = \lambda_i \mathbf{e}_i, i = 1, 2, 3 \quad (5.9)$$

sendo $\mathbf{e}_i$ um vetor de comprimento unitário na direção do autovetor $\mathbf{x}_i$ e λ_i o autovalor correspondente. Esta relação mostra que cada autovetor tem como magnitude seu autovalor correspondente.

Ordenando os autovalores de forma que $\lambda_1 \geq \lambda_2 \geq \lambda_3$, os autovetores associados $\mathbf{x}_1$, $\mathbf{x}_2$ e $\mathbf{x}_3$ serão denominados *maior*, *intermediário* e *menor*, respectivamente.

A tensão σ em um ponto qualquer de um corpo geológico é conhecida a partir da determinação das componentes normal σ_n e cisalhante σ_s , atuantes em um plano orientado que contém o ponto. Naturalmente, existe um número infinito destes planos. Logo, é necessário estabelecer qual informação mínima permite determinar as componentes de tensão em qualquer plano. Esta informação é representada graficamente por um elipsóide de tensão (Figura 5.3), cujo centro corresponde ao ponto de aplicação da tensão. Os eixos maior, intermediário e menor do elipsóide são denominados **principais**. Tais eixos, denotados $\hat{\sigma}_1$, $\hat{\sigma}_2$ e $\hat{\sigma}_3$ (sendo que $\hat{\sigma}_1 \geq \hat{\sigma}_2 \geq \hat{\sigma}_3$), são perpendiculares a três planos mutuamente ortogonais, também chamados principais, ao longo dos quais as tensões normais possuem valores extremos e as tensões cisalhantes são nulas. O ponto de aplicação da tensão é representado através de um cubo infinitesimal cujas faces são paralelas aos planos principais e perpendiculares aos eixos principais $\hat{\sigma}_1$, $\hat{\sigma}_2$ e $\hat{\sigma}_3$ (Figura 5.4).

As considerações expostas para a tensão valem também para sua consequência, a deformação. Neste caso, utiliza-se um tensor de deformação para o qual é possível encontrar direções de deformação **principais**. Assume-se que o elipsóide correspondente seja o resultado da deformação de uma esfera na direção dos eixos principais $\hat{e}_1$, $\hat{e}_2$ e $\hat{e}_3$ (sendo que $\hat{e}_1 \geq \hat{e}_2 \geq \hat{e}_3$). $\hat{e}_1$ constitui usualmente o eixo de maior estiramento ($\hat{e}_1 > 1$), $\hat{e}_2$ pode indicar estiramento, encurtamento ou deformação plana ($\hat{e}_2 > 1$, $\hat{e}_2 < 1$ e $\hat{e}_2 = 1$, respectivamente), ao passo que $\hat{e}_3$ indica o eixo de maior encurtamento (Figura 5.5).

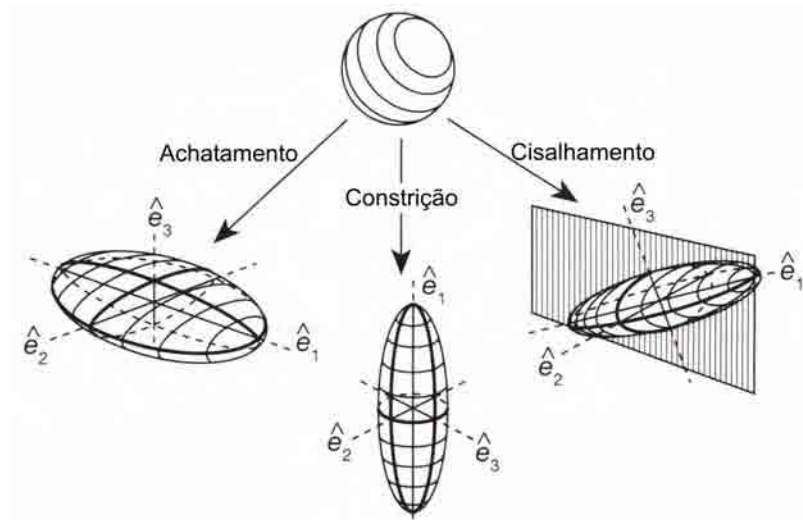


Figura 5.5 – Elipsóide de deformação. O índice 1 dos eixos está associado ao estiramento, enquanto o índice 3 denota encurtamento. O semi-eixo intermediário (índice 2) pode representar estiramento, encurtamento ou deformação plana. Modificado de Twiss & Moores (1992).

5.5 O círculo de Mohr para a tensão

O elipsóide de tensão sugere graficamente que as componentes normais e cisalhantes ao longo de um plano devem variar progressivamente ao se modificar a orientação deste.

No entanto, a relação entre a orientação deste plano e as componentes normais e cisalhantes da tensão é de difícil determinação a partir da observação do elipsóide. Esta relação se torna mais simples quando as referidas componentes forem traçadas em um gráfico, criando o chamado *círculo de Mohr* (Figura 5.6). Este é construído usando um diagrama em que o eixo horizontal fornece o valor da tensão normal σ_n , ao passo que o eixo vertical indica o valor da tensão cisalhante σ_s .

Para uma determinada configuração de tensão, descrita graficamente por um elipsóide, as componentes normais e cisalhantes que atuam sobre um ponto em planos de orientação qualquer são traçadas como pontos de um gráfico de forma a gerar o círculo de Mohr. O centro do círculo encontra-se sobre o eixo da tensão normal. As tensões normais máxima ($\hat{\sigma}_I$)

e mínima ($\hat{\sigma}_3$) possuem valores associados aos interceptos do círculo com o eixo horizontal σ_n . Estes são os únicos pontos em que a tensão cisalhante é nula.

A orientação de um plano P no espaço 3D é especificada por meio de seu vetor normal $\mathbf{n}$ (Figura 5.6a). O plano é posicionado em um sistema de coordenadas cujos eixo $\hat{x}_1$ e $\hat{x}_3$ correspondem aos eixos principais de tensão $\hat{\sigma}_1$ e $\hat{\sigma}_3$. O ângulo θ que sua normal forma com o primeiro eixo serve para definir sua orientação. O plano P é paralelo ao eixo principal $\hat{\sigma}_2$, sendo as componentes de tensão paralelas ao plano $\hat{x}_1\hat{x}_3$ ($\hat{\sigma}_1\hat{\sigma}_3$).

O centro do círculo de Mohr (Figura 5.6b) possui abscissa igual à media aritmética das tensões normais máxima e mínima, ao passo que seu raio é igual à metade da diferença entre as mesmas. A orientação do plano P no espaço 3D, definida pelo ângulo θ , é representada, no círculo de Mohr, usando o dobro desta medida, 2θ . Desta forma, ao variar a orientação do plano, os pontos no diagrama de Mohr percorrem o círculo, fornecendo pares de valores ($\sigma_n^{(P)}$, $\sigma_s^{(P)}$) para as tensões normal e cisalhante que atuam sobre o plano de orientação θ .

Conhecendo tal orientação em relação ao maior eixo principal de tensão, é possível determinar os valores das tensões normal e cisalhante sobre o plano. Analiticamente:

$$\sigma_n = \left[\frac{\hat{\sigma}_1 + \hat{\sigma}_3}{2} \right] + \left[\frac{\hat{\sigma}_1 - \hat{\sigma}_3}{2} \right] \cos 2\theta \quad , \quad \sigma_s = \left[\frac{\hat{\sigma}_1 - \hat{\sigma}_3}{2} \right] \sin 2\theta \quad (5.10)$$

As componentes de tensão discutidas até agora são paralelas ao plano $\hat{x}_1\hat{x}_3$, fornecendo portanto uma informação de natureza bidimensional. Para as tensões nos outros planos principais, é possível traçar os círculos de Mohr correspondentes. Tais círculos podem ser combinados em um mesmo diagrama de forma a proporcionar uma informação pseudo-tridimensional a respeito da distribuição das tensões normais e cisalhantes sobre planos orientados em relação aos três planos coordenados principais $\hat{x}_1\hat{x}_3$, $\hat{x}_1\hat{x}_2$ e $\hat{x}_2\hat{x}_3$. A informação representada não é propriamente tridimensional, pois os planos P considerados pertencem a três grupos bem definidos: os paralelos aos eixos principais de tensão $\hat{x}_2$, $\hat{x}_3$ e $\hat{x}_1$, respectivamente. Dentro de cada grupo, os planos considerados podem ter orientação qualquer, especificada através dos ângulos θ_1 , θ_2 e θ_3 que suas normais formam com os eixos principais de tensão $\hat{x}_2$ e $\hat{x}_1$ (Figura 5.7a).

A informação “tridimensional” de tensão é representada através de três círculos de Mohr, cada qual correspondente ao feixe de planos paralelo a um eixo principal (Figura 5.7b).

No eixo horizontal do diagrama são representados simultaneamente os valores máximo, intermediário e mínimo das tensões normais, $\hat{\sigma}_1$, $\hat{\sigma}_2$ e $\hat{\sigma}_3$. Cada um destes valores corresponde a um ponto comum a dois círculos de Mohr.

As equações dos três círculos são análogas à equação (5.10), com a diferença que devem ser considerados três pares de índices para referenciar as tensões principais: 1 e 3; 1 e 2; 2 e 3.

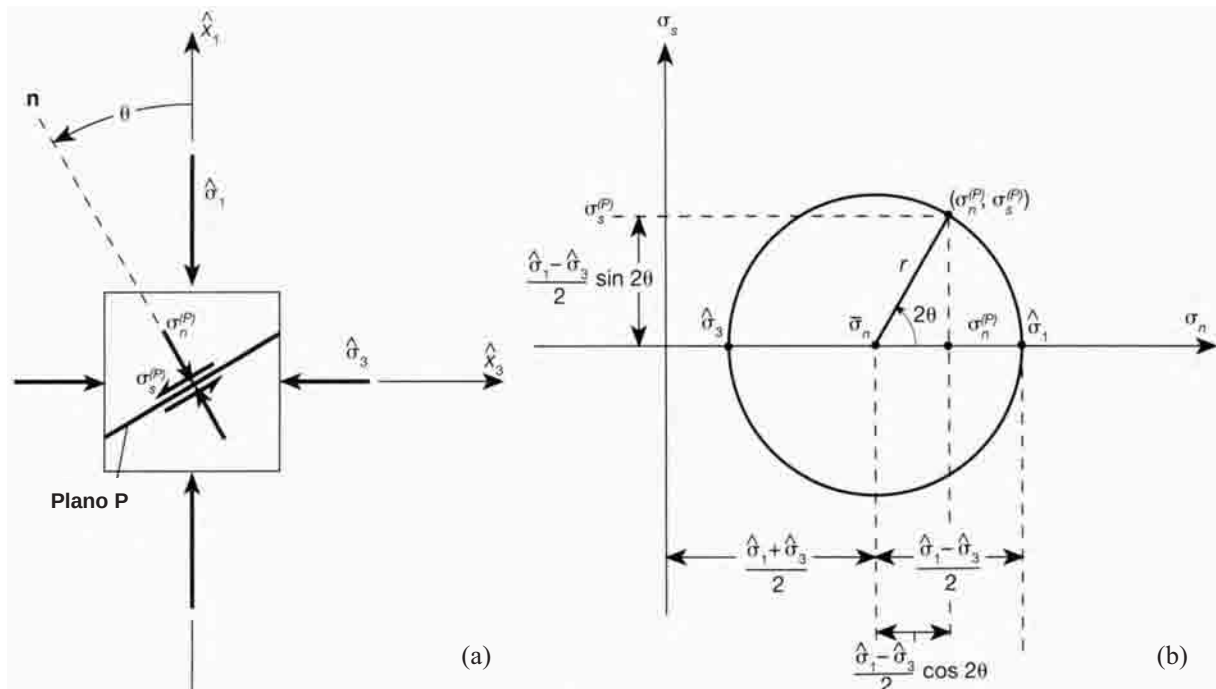


Figura 5.6 – Diagrama de Mohr da tensão aplicada sobre um plano P . (a) Relações entre as componentes de tensão, os eixos coordenados principais correspondentes às tensões normais máxima e mínima, e o plano P com sua normal $\mathbf{n}$. (b) Tensão correspondente representada através do círculo de Mohr. A notação “(P)” indica as componentes de tensão que agem sobre o plano P . Modificado de Twiss & Moores (1992).

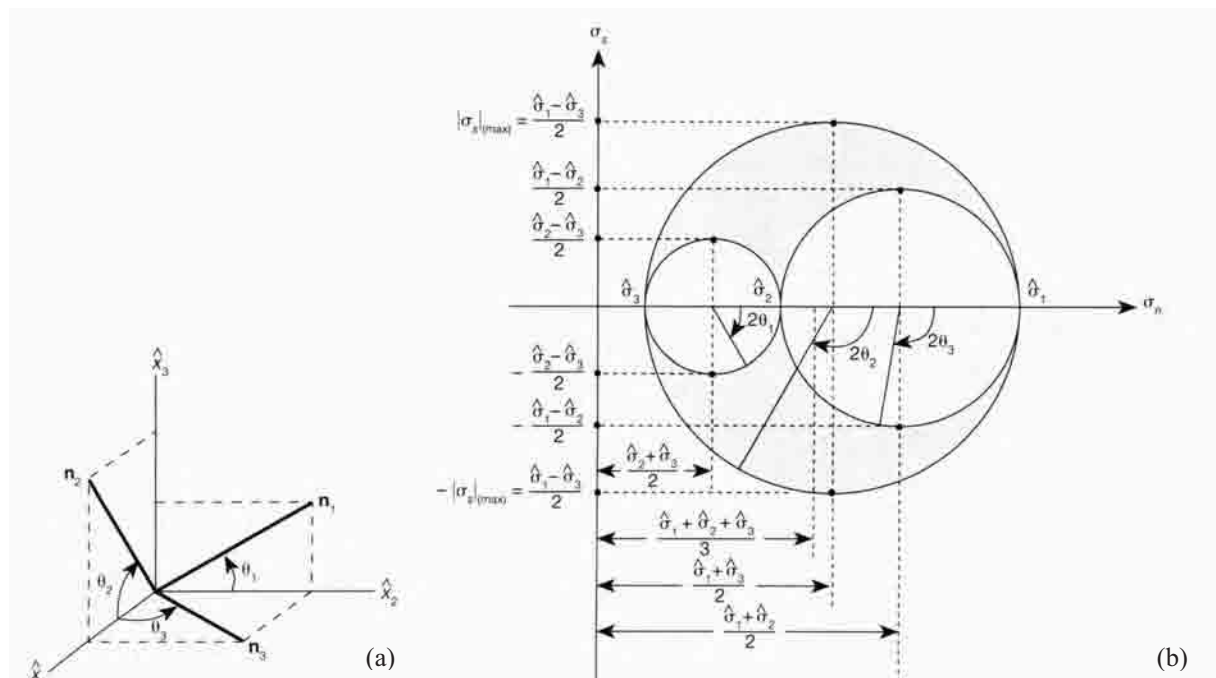


Figura 5.7 – Distribuição tridimensional de tensões representada através do diagrama de Mohr. (a) Gráfico 3D para a orientação dos planos de aplicação da tensão. $\mathbf{n}_1$, $\mathbf{n}_2$ e $\mathbf{n}_3$ são os vetores normais a três planos paralelos aos eixos principais. θ_1 , θ_2 e θ_3 são os ângulos entre os eixos principais e as normais, medidos nos três planos coordenados exibidos. (b) Diagrama de Mohr. Cada círculo de Mohr representa a tensão bidimensional em um dos planos coordenados principais. As componentes de tensão normais e cisalhantes que agem sobre os planos perpendiculares a $\mathbf{n}_1$, $\mathbf{n}_2$ e $\mathbf{n}_3$ são representadas como pontos sobre os círculos de Mohr correspondentes. Fonte: Twiss & Moores (1992).

5.6 O círculo de Mohr para a deformação

O círculo de Mohr para a deformação possui finalidade análoga ao círculo para a tensão e descreve como a deformação normal (extensão) e a deformação cisalhante variam ao se modificar a orientação de um plano em relação ao elipsóide de deformação. Considerando a deformação ao longo do plano cinemático XY (sendo X e Y os eixos maior e menor da elipse de deformação no plano), a construção do círculo requer a determinação de alguns parâmetros de deformação:

$$\lambda'_1 = \frac{1}{(1+e_1)^2} \quad , \quad \lambda'_2 = \frac{1}{(1+e_2)^2} \quad (5.11)$$

Nestas equações, λ'_1 e λ'_2 definem as *elongações quadráticas recíprocas*, ao passo que $(1+e_1)$ e $(1+e_2)$ são, respectivamente, as elongações na direção dos semi-eixos principais maior e menor da elipse de deformação no plano, isto é, o comprimento de tais semi-eixos.

Nesta seção, o índice 2 está associado ao semi-eixo principal menor, e não intermediário, para seguir a convenção de estudo bidimensional utilizada na literatura para o círculo de Mohr de deformação (Ramsay & Huber, 1983). Neste caso, o elipsóide reduz-se a uma elipse e não há semi-eixo intermediário.

Outro parâmetro importante é o ângulo ϕ' entre a direção de um elemento ou objeto no estado deformado e a direção do eixo de maior estiramento X (eixo principal $\hat{e}_1$).

De posse destas informações, é possível traçar o círculo de Mohr para a deformação (Figura 5.8).

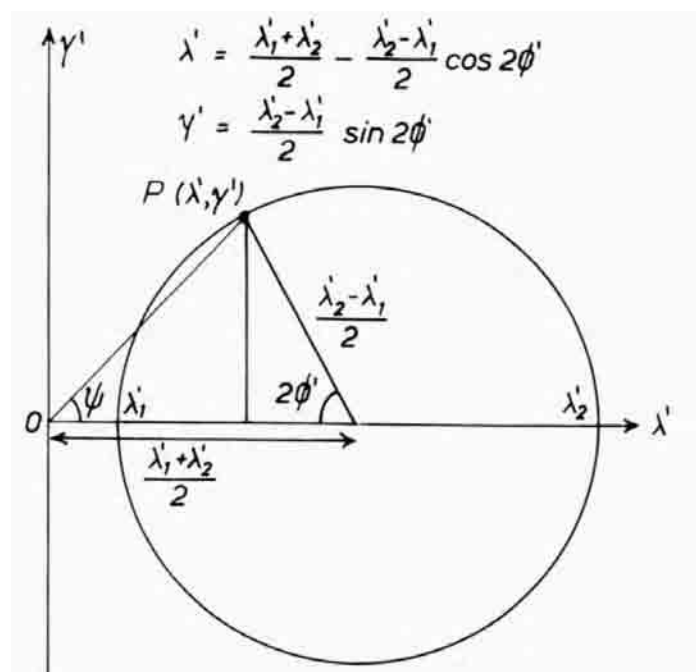


Figura 5.8 – Construção do círculo de Mohr para a deformação. Fonte: Ramsay & Huber (1983).

Verifica-se que as elongações quadráticas recíprocas λ'_1 e λ'_2 , indicadas no eixo horizontal λ' , são utilizadas para calcular o centro e o raio do círculo. Ao se modificar o ângulo ϕ' , um ponto genérico P percorre o círculo de Mohr. As coordenadas do ponto (λ', γ') são proporcionais às componentes de deformação normais e cisalhantes, respectivamente.

É interessante notar que as elongações quadráticas recíprocas, λ'_1 e λ'_2 , são inversamente proporcionais às elongações $(1+e_1)$ e $(1+e_2)$. Estas últimas medem o comprimento dos semi-eixos maior e menor da elipse de deformação no caso 2D, assim como $\hat{\sigma}_1$ e $\hat{\sigma}_3$ medem o comprimento dos semi-eixos maior e menor do elipsóide de tensão.

Mas, se por um lado as tensões normais máxima e mínima são indicadas diretamente no eixo horizontal σ_n do círculo de Mohr para a tensão (Figura 5.7), de forma que a maior tensão normal, $\hat{\sigma}_1$, ocorre na extremidade direita do gráfico; por outro as deformações normais máxima e mínima (elongações $(1+e_1)$ e $(1+e_2)$) no plano considerado correspondem, respectivamente, às extremidades esquerda e direita do círculo de Mohr para deformação ao longo do eixo horizontal λ' . Isto ocorre em virtude da utilização de elongações quadráticas *recíprocas*. Tal diferença deve ser levada em conta ao utilizar o comprimento dos semi-eixos principais de tensão e deformação durante a construção dos círculos de Mohr correspondentes.

As equações do círculo de Mohr para a deformação são as seguintes:

$$\lambda' = \frac{\lambda'_1 + \lambda'_2}{2} - \frac{\lambda'_2 - \lambda'_1}{2} \cos 2\phi' \quad , \quad \gamma' = \frac{\lambda'_2 - \lambda'_1}{2} \sin 2\phi' \quad (5.12)$$

A extensão destes conceitos para o caso tridimensional (passagem de uma elipse com dois semi-eixos para um elipsóide com três semi-eixos), envolve a consideração de três pares de eixos principais ($\hat{e}_1\hat{e}_2$, $\hat{e}_1\hat{e}_3$, $\hat{e}_2\hat{e}_3$), cujas extensões quadráticas recíprocas são utilizadas nas equações 5.11 e 5.12. Neste contexto, o índice 2 está novamente associado ao semi-eixo principal intermediário do elipsóide de deformação. Como no caso da tensão, os três círculos de Mohr construídos se referem à deformação nos três planos principais (isto é, planos perpendiculares aos eixos principais).

5.7 Obtenção de elipsóides a partir de tensores

Os elipsóides de tensão e deformação constituem a representação gráfica das entidades matemáticas tensores. A obtenção dos elipsóides ocorre conforme procedimento a seguir. O primeiro passo consiste na extração dos autovalores e autovetores associados ao tensor. A

técnica analítica de determinação é dada pelas equações 5.7 a 5.9. Como os autovetores são ortogonais, estes formam um sistema local de coordenadas. Estes vetores podem ser associados aos semi-eixos menor, intermediário e maior de um elipsóide. A forma e orientação do elipsóide (em relação ao sistema global de coordenadas) são determinadas, respectivamente, através dos autovalores e dos autovetores do tensor correspondente.

Para determinar o elipsóide associado a um tensor, posiciona-se uma esfera no ponto de aplicação deste (Figura 5.10). A esfera é rotacionada em torno de sua origem de maneira que seu sistema local de coordenadas coincida com os autovetores. Em seguida, utiliza-se os autovalores maior, intermediário e menor para realizar um redimensionamento (*scaling*) da esfera ao longo dos autovetores correspondentes. O resultado é o elipsóide procurado.

Em termos algébricos, estas transformações geométricas são a consequência da multiplicação do conjunto de vértices (coordenadas (x, y, z) representadas como vetores coluna) que define a esfera por matrizes 4x4. Em computação gráfica, uma matriz de transformação 4x4 é responsável pelo deslocamento de pontos tridimensionais que constituem um objeto (FOLEY et al., 1996). O resultado da aplicação de tais transformações sobre todos os vértices (ou pontos) do objeto consiste em translações, rotações ou modificações de escala da entidade como um todo. Existem também deformações mais complexas que podem ser definidas por matrizes. O cisalhamento é um exemplo disto.

A transformação final T resulta da multiplicação das seguintes matrizes:

$$T = T_T T_R T_S \quad (5.13)$$

T_T é a matriz de translação do centro inicial da esfera para o ponto de aplicação do tensor, T_R a matriz de rotação que faz coincidir os eixos coordenados do sistema local da esfera com os eixos principais das direções dos autovetores; e T_S a matriz de deformação da esfera rotacionada para o elipsóide procurado. Tal deformação ocorre por procedimento de *scaling* nas direções dos novos eixos de referência da esfera (o sistema de coordenadas dos eixos principais, ou autovetores). É importante lembrar que não é possível identificar de forma completa um elipsóide no espaço 3D usando apenas T_R ou T_S , pois a primeira matriz indica somente orientação, enquanto a segunda indica modificação na forma da esfera (encurtamento ou estiramento nas direções de três eixos ortogonais *quaisquer*). Um elipsóide está totalmente determinado após fornecer informação de orientação (autovetores da matriz do tensor) e de deformação (autovalores da mesma matriz). A remanescente ordem de informação, a posição no espaço 3D, está embutida na matriz de translação T_T .

5.8 Considerações sobre a simetria dos tensores de tensão e deformação

As relações 5.3 e 5.6 mostram a esquematização das componentes dos tensores de tensão e deformação através de uma notação matricial. As representações utilizadas são:

$$\sigma_{kl} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix} \text{ e } e_{kl} = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix}$$

Nestas relações, tensores de segunda ordem no espaço 3D são representados através de matrizes quadradas 3x3, com um total de nove componentes para cada tensor.

Na verdade, a possibilidade de alterar de forma independente cada componente da matriz do tensor representa uma abstração didática que não ocorre em aplicações práticas. Isto é consequência das considerações expostas a seguir.

Para o tensor de tensão aplicada em algum ponto de uma rocha, assume-se a hipótese realista de que o elemento se encontre em equilíbrio mecânico em relação aos demais elementos da rocha (caracterizados por outros tensores de tensão). Isto significa que a resultante das forças que atuam sobre o ponto de aplicação do tensor deve ser nula (TWISS & MOORES, 1992; MORAES, 2000). Esta condição é garantida através das seguintes igualdades:

$$\sigma_{12} = \sigma_{21}, \sigma_{23} = \sigma_{32}, \sigma_{31} = \sigma_{13} \quad (5.14)$$

cujas observações implicam que o tensor de tensão é *simétrico* em relação à sua diagonal principal, havendo apenas seis graus de liberdade (ao invés de nove) na especificação de suas componentes. É importante notar que a simetria envolve somente as componentes cisalhantes.

Considerações análogas valem para o tensor da deformação estudada em algum ponto de uma rocha. Assumindo, por exemplo, dois segmentos paralelos a x1 e x2 (Figura 5.2b), o ângulo de cisalhamento ψ_{12} de x1 em relação a x2 é o mesmo que o ângulo ψ_{21} de x2 em relação a x1. Portanto, na representação matricial do tensor de deformação, mantêm-se as seguintes condições:

$$e_{12} = e_{21}, e_{23} = e_{32}, e_{31} = e_{13} \quad (5.15)$$

isto é, o tensor é simétrico em relação à sua diagonal principal e existem apenas seis componentes independentes em sua representação matricial. Como no caso anterior, a simetria envolve somente as componentes cisalhantes.

Como foi ilustrado, a utilização de tensores de segunda ordem simétricos em relação a sua diagonal principal reduz de nove para seis o número de componentes que podem ser manipuladas de forma independente. Esta condição, que do ponto de vista matemático assume o caráter de verdadeira limitação, é, por outro lado, fundamental para se trabalhar com realismo em geologia estrutural. Este é um caso típico de diferença entre abrangência da ferramenta matemática e sua utilização em áreas aplicadas.

Além disto, os algoritmos de álgebra linear utilizados para determinar os eixos principais de tensão e deformação para matrizes de tensores (e não, simplesmente, para matrizes quadradas genéricas), têm como pressuposto a simetria destas. Tal exigência foi observada tanto no ambiente de visualização do VTK, como em bibliotecas externas utilizadas para esta finalidade. Este ponto será abordado em detalhe durante a explicação do funcionamento do programa *Tensor3D*.

5.9 O programa *Tensor3D*

O programa *Tensor3D* é um aplicativo de computação gráfica desenvolvido neste projeto para visualizar a deformação de objetos de acordo com as informações contidas na matriz 3x3 associada a um tensor. As exigências fundamentais consideradas durante seu desenvolvimento foram:

- Elevado grau de interatividade. O usuário pode modificar cada componente do tensor e observar *instantaneamente* a deformação ocorrida no objeto estudado.
- Visualização tridimensional. A informação de deformação normal ou cisalhante embutida na matriz do tensor e manipulada pelo usuário é aplicada a um objeto visualizado *em 3D*, sendo possível modificar em tempo real importantes parâmetros de exibição, como o ponto de observação, a distância, etc. Esta técnica visa realçar as relações espaciais que podem ser aferidas através da observação em três dimensões da modificação de parâmetros de tensão e deformação.

O programa foi desenvolvido usando a linguagem Tcl/Tk (2005) que independe do sistema operacional utilizado (Windows, Linux, etc.), e fazendo uso do conjunto de classes de modelagem geométrica e visualização científica do VTK (VISUALIZATION TOOLKIT, 2005).

A escolha da linguagem Tcl/Tk foi ditada pela necessidade de se desenvolver uma ferramenta didática de estudo das deformações em geologia estrutural, que fosse facilmente utilizável em plataformas computacionais diferentes, independente do ambiente de

desenvolvimento inicial. Esta facilidade de utilização foi particularmente observada em ambiente Windows, sendo apenas necessário realizar a instalação dos módulos pré-compilados do interpretador de comandos Tcl/Tk e da biblioteca VTK. Outro argumento a favor desta linguagem é o seu conjunto de ferramentas gráficas (botões, barras de rolagem, *checkboxes*, etc.) de interação com o usuário. A integração de tais ferramentas em programas interativos apresenta um baixo nível de complexidade, o que agiliza o processo de desenvolvimento.

Analogamente, a escolha do VTK como ambiente de modelagem geométrica e de visualização foi orientada pela facilidade de desenvolvimento de aplicações 3D que este sistema proporciona, sendo possível escrever programas relativamente complexos com algumas dezenas de linhas de código. Como foi discutido anteriormente (seção 3.4), isto é consequência da facilidade de instanciação (criação) e manipulação de objetos e processos predefinidos (modelos 3D, algoritmos de modelagem geométrica e de visualização, etc.) que o VTK oferece. Além disto, os conhecimentos sobre VTK deste mestrando, iniciados e sedimentados ao longo de três anos de atividades como bolsista de iniciação científica junto ao Laboratório de GeoModelagem 3D do DPM / IGCE / UNESP, apontaram como escolha natural a utilização desta biblioteca.

Por fim, a junção de ferramentas de trabalho de integração relativamente simples permite realizar a prototipação de aplicações, isto é, o desenvolvimento rápido de programas de computação científica com elevado grau de *feedback*, sendo possível observar de forma quase instantânea o resultado das modificações realizadas durante o ciclo de desenvolvimento. Obviamente, este estágio de desenvolvimento não leva em conta problemáticas de robustez e tolerância a falhas dos programas escritos, dando-se preferência para a construção de protótipos funcionantes. Em consequência disto, tais exigências de robustez devem ser reservadas para ciclos posteriores de refinamento.

5.9.1 Modelagem de *glyphs*

Embora o programa *Tensor3D* seja o resultado da inter-relação (conexão) entre numerosos módulos ou classes da biblioteca VTK, destaca-se a classe *vtkTensorGlyph*, que utiliza as informações matemáticas contidas na representação matricial de um tensor para deformar um modelo geométrico 3D (no ambiente VTK, uma instância da classe *vtkPolyData* que contém informações geométricas e topológicas, vide seção 3.4.2).

Antes de considerar a classe *vtkTensorGlyph*, será abordado o conceito genérico de *glyph*. É comum em visualização científica dispor de um conjunto de dados distribuídos no espaço tridimensional. A cada ponto (x, y, z) do conjunto considerado podem estar associados, por exemplo, atributos de natureza escalar (pressão, temperatura, porosidade, etc.), vetorial (velocidade, aceleração, etc.) ou tensorial (tensão, deformação, etc.). Algumas técnicas de computação gráfica utilizadas em visualização científica visam representar tais atributos através de objetos tridimensionais posicionados nos pontos correspondentes. Tais objetos são denominados *glyphs* e sua forma, escala, orientação e cor podem ser modificadas de acordo com as informações associadas a cada ponto do conjunto de dados. Por exemplo, uma distribuição de temperaturas medidas em pontos do espaço tridimensional poderia ser representada através de um conjunto de pequenas esferas posicionadas nos pontos conhecidos.

Em termos de representação 3D dos dados, o simples fato de identificar visualmente uma esfera em determinado ponto seria indício de um registro de temperatura presente *naquele ponto*. Mas as propriedades de visualização do objeto gráfico considerado podem variar de acordo com outras informações. Continuando o exemplo das temperaturas, cada esfera (*glyph*) poderia receber uma cor associada à temperatura do ponto correspondente, de acordo com uma escala de cores do azul para o vermelho que acompanhasse a variação absoluta de temperaturas do conjunto. Portanto, duas ordens de informação numérica (coordenadas e temperatura) teriam correspondência direta com duas propriedades gráficas (posição e cor de uma esfera). Mas os *glyphs* possuem outras propriedades, como o tamanho relativo (escala). No exemplo considerado, as esferas poderiam ser visualizadas com tamanhos diferentes, levando em conta quão recente é a informação de temperatura associada ao ponto correspondente: para informações mais recentes poderiam ser utilizadas esferas de maior raio. O exemplo poderia continuar levando em conta outros atributos associados aos pontos. O que se deseja destacar é a poderosa capacidade de visualizar dados multidimensionais através de *glyphs*, associando-se a um atributo (escalar, vetorial, tensorial, etc.) uma ou mais propriedades dos objetos 3D considerados.

O sistema VTK possui uma classe, chamada *vtkGlyph3D* (SCHROEDER et al., 1996), que permite implementar *glyphs* copiando uma representação geométrica em cada ponto de um conjunto de dados. A definição do objeto ocorre criando instâncias da classe *vtkPolyData* (descrita na seção 3.4.2) e o *glyph* pode ser orientado de acordo com informações vetoriais e/ou redimensionado de acordo com informações escalares (incluindo magnitudes de vetores).

A classe *vtkGlyph3D* é especializada na visualização de informações escalares e vetoriais, não sendo suportados dados tensoriais. A Figura 5.9 mostra dois exemplos de

utilização de *glyphs* em visualização científica. No primeiro caso, é exibida a triangulação *Delaunay* de um conjunto de pontos no plano (WATSON, 1981). Os pontos foram destacados utilizando-se a representação de pequenas esferas (informação de posição). No segundo caso, utilizou-se um conjunto de pirâmides para representar um campo vetorial ao longo da superfície de uma esfera. As pirâmides estão orientadas na direção dos vetores (informação de posição e de orientação) do campo. As pirâmides poderiam sofrer mudança de escala ou de cor de acordo com a intensidade do campo (magnitude dos vetores correspondente).

Para grandezas tensoriais, como informações de tensão e deformação em determinado ponto do espaço, utiliza-se a classe *vtkTensorGlyph*, que representa a extensão da classe *vtkGlyph3D* para informações matriciais. Como esta, a primeira copia uma representação geométrica (novamente especificada através de uma instância da classe *vtkPolyData*) em cada ponto de aplicação do tensor. O objeto considerado é deformado (isto é, redimensionado e rotacionado) de acordo com as componentes normais e cisalhantes do tensor considerado. A mudança de escala e a rotação são controladas através dos autovalores e autovetores da matriz 3x3 do tensor, de acordo com o procedimento ilustrado a seguir (vide seção 5.7).

A Figura 5.10, obtida utilizando o programa *Tensor3D*, mostra a deformação de um *glyph* (esfera semitransparente) usando a matriz de um tensor 3x3. Em (a) é possível identificar a esfera antes da deformação, no sistema de coordenadas *global* XYZ. Para o tensor considerado, a classe *vtkTensorGlyph* calcula os autovalores e seus autovetores correspondentes. Em seguida, os pares (autovalor, autovetor) são ordenados segundo o valor decrescente dos autovalores. A Figura 5.10b mostra o sistema de eixos principais calculado para o tensor do exemplo. Tais eixos possuem comprimentos diferentes, proporcionais aos autovalores encontrados. Desta forma é possível identificar os eixos de maior estiramento ($\hat{e}_1$), intermediário ($\hat{e}_2$) e de maior encurtamento ($\hat{e}_3$) do elipsóide de deformação associado ao tensor. A Figura 5.10c mostra o resultado da etapa final do processo. O maior autovetor redimensiona o *glyph* (a esfera) na direção X, o intermediário na direção Y e o menor na direção Z. Em seguida, o objeto é rotacionado de forma que o eixo X de seu sistema *local* de coordenadas coincida com o eixo principal maior $\hat{e}_1$, o eixo Y com o eixo principal intermediário $\hat{e}_2$ e o eixo Z com o eixo principal menor $\hat{e}_3$. A figura não mostra o sistema de coordenadas local do objeto, sendo XYZ o sistema global. O objeto redimensionado (deformado) e rotacionado da figura coincide com o elipsóide de deformação do tensor, já que sua forma inicial era a de uma esfera. Objetos diferentes podem ser utilizados, conforme será mostrado mais adiante.

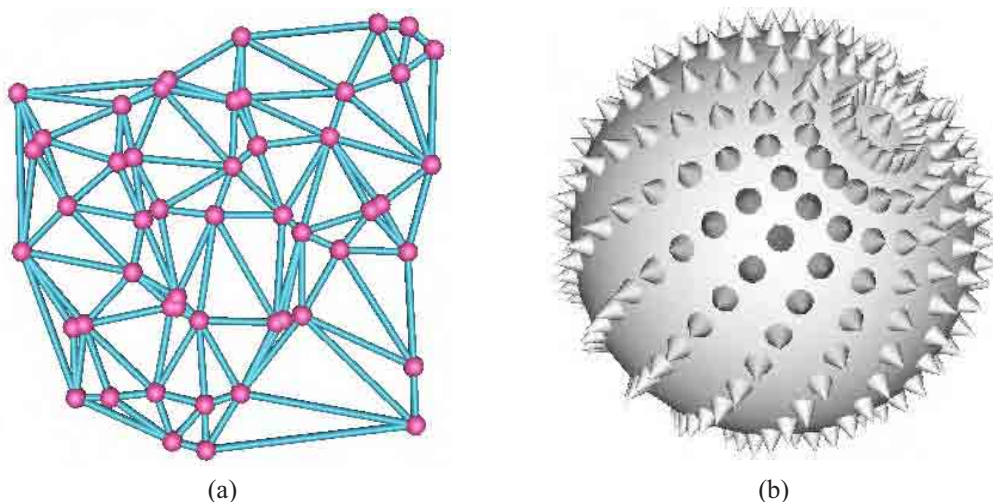


Figura 5.9 – Utilização de *glyphs* para representar diferentes ordens de informações (atributos) associadas a pontos tridimensionais. (a) Esferas realçam a posição dos pontos em uma malha de triângulos. (b) Pirâmides são posicionadas nos pontos de aplicação de grandezas vetoriais e orientadas de acordo com as mesmas.

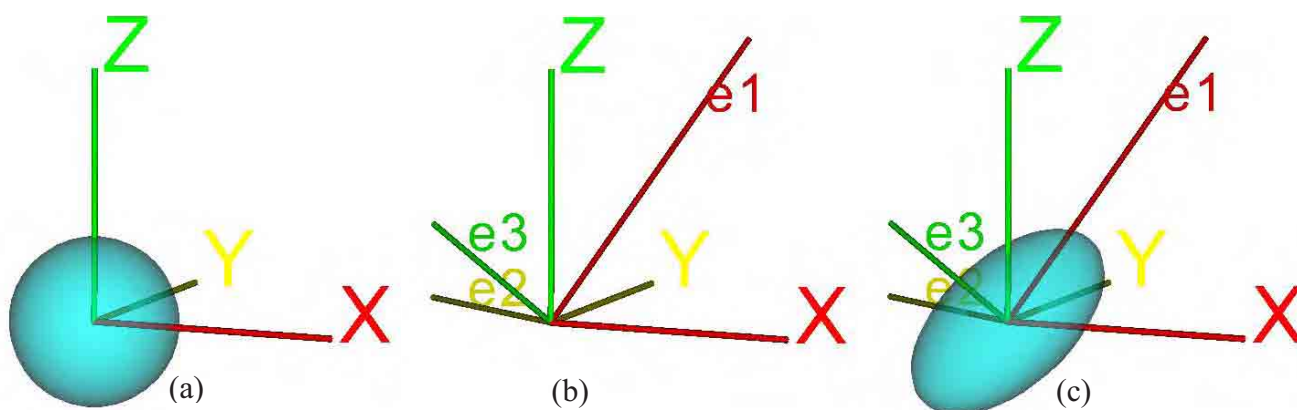


Figura 5.10 – Exemplo de uso da classe *vtkTensorGlyph*. (a) Esfera visualizada no sistema XYZ de eixos coordenados antes da deformação. (b) Visualização dos eixos principais $\hat{e}_1\hat{e}_2\hat{e}_3$ do tensor. (c) Visualização da esfera deformada e orientada (elipsóide) usando as informações encontradas.

5.9.2 Descrição das funcionalidades

A interface de execução do programa *Tensor3D*, exibida na Figura 5.11, é constituída por um conjunto de barras de rolagem destinadas a modificar as componentes de um tensor 3×3 (região 1), vários botões usados para controlar as principais funções do aplicativo (região 2), bem como uma janela de visualização 3D do objeto deformado (região 3). A seguir serão detalhadas as funcionalidades do programa exibidas na figura.

Seletores das componentes do tensor. A região 1 da Figura 5.11 mostra um conjunto de barras de rolagem alinhadas de forma a lembrar a estrutura de uma matriz de elementos 3×3 . Com efeito, tal região destina-se à especificação dos valores das nove componentes do tensor de tensão/deformação utilizado no programa. O usuário pode modificar o valor de cada componente “arrastando” o indicador (retângulo central) contido na barra de rolagem correspondente até alcançar um valor satisfatório. Tal valor é visualizado na parte superior da

barra. Esta forma de interação não permite realizar um ajuste fino da componente do tensor. Para tanto, é possível selecionar com o mouse a região à direita ou esquerda do indicador. Isto resultará no ajuste do valor exibido com incrementos de 0.001 unidades para mais ou para menos.

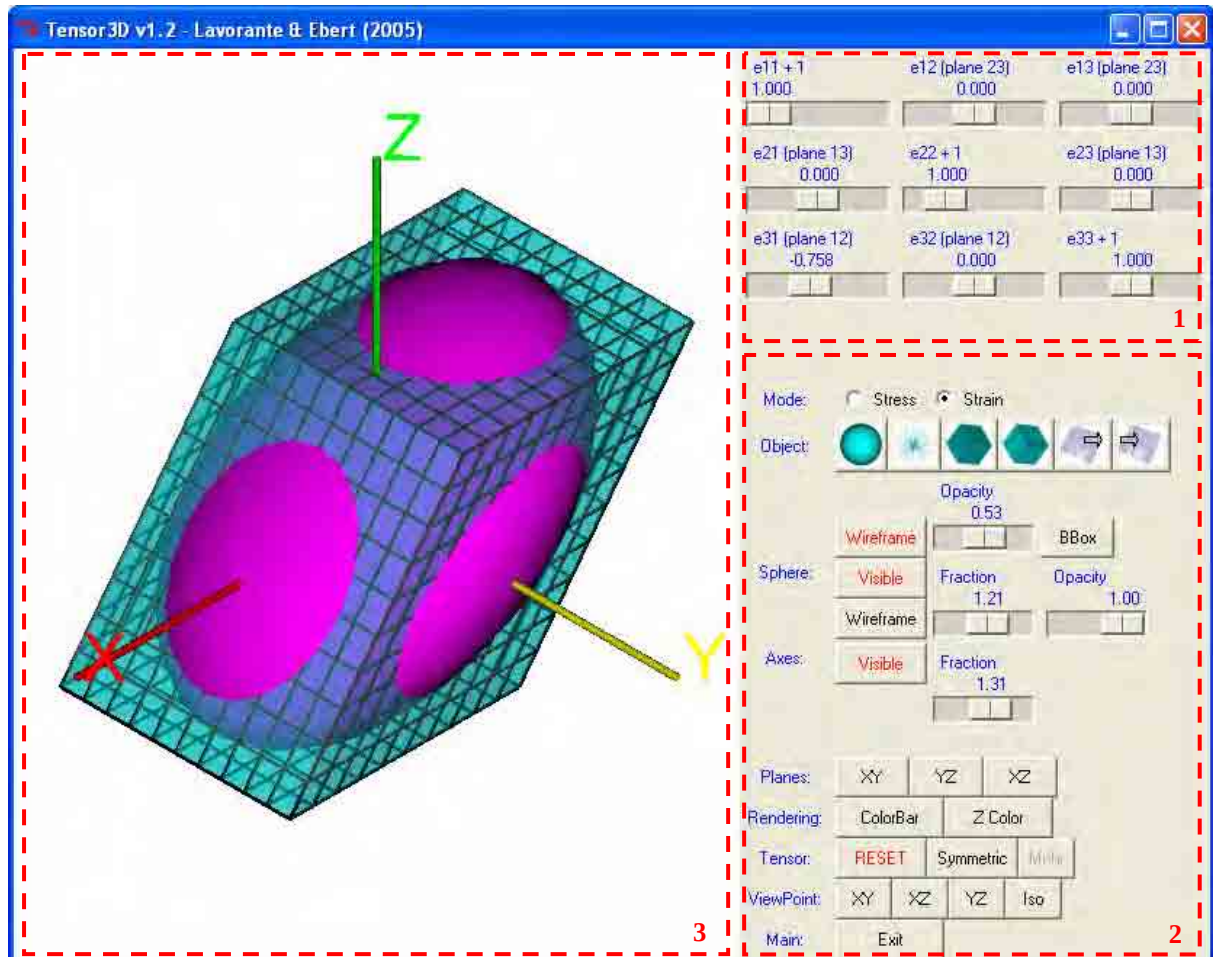


Figura 5.11 – Janela de execução do programa *Tensor3D*. A região 1 mostra barras de rolagem para modificação dos valores das componentes de um tensor 3x3. Abaixo (região 2), um conjunto de botões para o controle das funcionalidades do programa. À esquerda (região 3), a visualização 3D de um cubo deformado de acordo com as informações do tensor. Foi realizado um cisalhamento simples no plano XY e na direção negativa do eixo X (a componente “e31” da matriz foi modificada de 0 para -0.758). O elipsóide em roxo mostra a direção e entidade da deformação.

Os valores iniciais das componentes do tensor não são iguais e dependem da modalidade de utilização do programa: tensão ou deformação. A modalidade é selecionada pelo usuário, conforme será descrito mais adiante. No caso da tensão, os valores iniciais do tensor serão todos nulos. Já na modalidade de deformação, os valores serão inicializados com a seguinte matriz, que coincide com uma matriz identidade 3x3:

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.16)$$

Em ambos os casos, as componentes cisalhantes (fora da diagonal principal) recebem, inicialmente, valor nulo. Já na modalidade de deformação, as componentes normais (ao longo da diagonal principal) recebem o valor 1.0. Isto denota uma condição de estiramento / encurtamento nulo que condiz com os conceitos de *scaling* utilizados em Computação Gráfica (FOLEY et al., 1996). Segundo tal notação, um *scaling* (ou redimensionamento) unitário em determinada direção significa que o objeto *não* sofreu deformação naquela direção, isto é, não sofreu estiramento nem encurtamento. Por outro lado, um *scaling* maior do que 1 denota estiramento, ao passo que um valor menor do que 1 indica encurtamento.

A descrição de cada componente do tensor na janela do programa varia de acordo com a modalidade de utilização. No caso da tensão, cada componente recebe o nome “sAB”, sendo que A e B podem assumir os valores 1, 2 e 3 (correspondentes aos eixos coordenados X, Y e Z). De forma análoga, na modalidade de deformação cada componente do tensor recebe o nome “eAB”, sendo que A e B variam de 1 a 3.

Cada componente embute *duas* informações de direção, conforme definição do tensor de segunda ordem. Por exemplo, na região 1 da Figura 5.11, a componente e11 indica uma deformação normal através de um estiramento na direção do eixo X do sistema de eixos coordenados. Isto significa que a componente da deformação considerada atua no plano YZ perpendicular ao eixo X (primeiro índice de e11) e, ao longo do plano, na direção do mesmo eixo (segundo índice de e11). Outro exemplo é a componente e31. Esta indica uma deformação cisalhante que atua no plano XY perpendicular ao eixo Z (primeiro índice de e31) e na direção do eixo X (segundo índice de e31). A deformação resultante é mostrada na região 3 da Figura 5.11. Para as componentes cisalhantes, o plano ao longo do qual as mesmas atuam está indicado na região superior das barras de rolagem correspondentes.

Funções geométricas e de visualização. A região 2 da Figura 5.11 contém o conjunto de botões que controlam todas as funcionalidades do programa *Tensor3D*. Tais funcionalidades estão agrupadas de acordo com sua categoria (geométrica, gráfica, etc.) e são reconhecíveis através de um *label* (nome) azul. Segue uma discussão sobre os grupos de botões disponíveis.

Mode. Este seletor permite escolher entre as modalidades geométricas de tensão e de deformação. Várias funcionalidades do programa são influenciadas pela modalidade de trabalho. Em primeiro lugar os eixos principais são identificados com nomes diferentes de acordo com a modalidade. Além disto, os valores iniciais das componentes do tensor variam de forma adequada. Outra funcionalidade afetada diz respeito à visualização do círculo de Mohr e será descrita mais adiante. A Figura 5.12a mostra um exemplo de visualização de uma esfera deformada em modalidade de tensão. Além do elipsóide, são exibidos os eixos principais de tensão, “s1”, “s2” e “s3”, que neste exemplo não possuem a mesma orientação que os eixos coordenados XYZ. Já a Figura 5.12b mostra um cubo (com círculos desenhados em cada face) em modalidade de deformação usando as mesmas componentes do tensor da figura anterior. Os eixos principais são agora denotados pelos símbolos “e1” “e2” e “e3”. O elipsóide semitransparente no centro do objeto mostra graficamente a entidade da deformação.

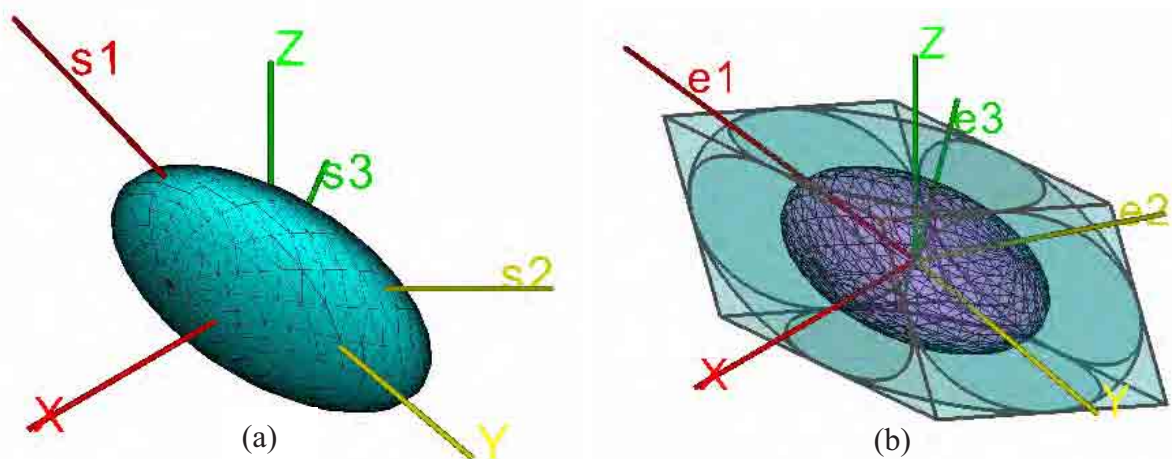


Figura 5.12 – Utilização do programa *Tensor3D* em modalidade de tensão (a) e deformação (b). Para deformar os objetos visualizados foram modificadas tanto as componentes normais como as cisalhantes do tensor de tensão / deformação.

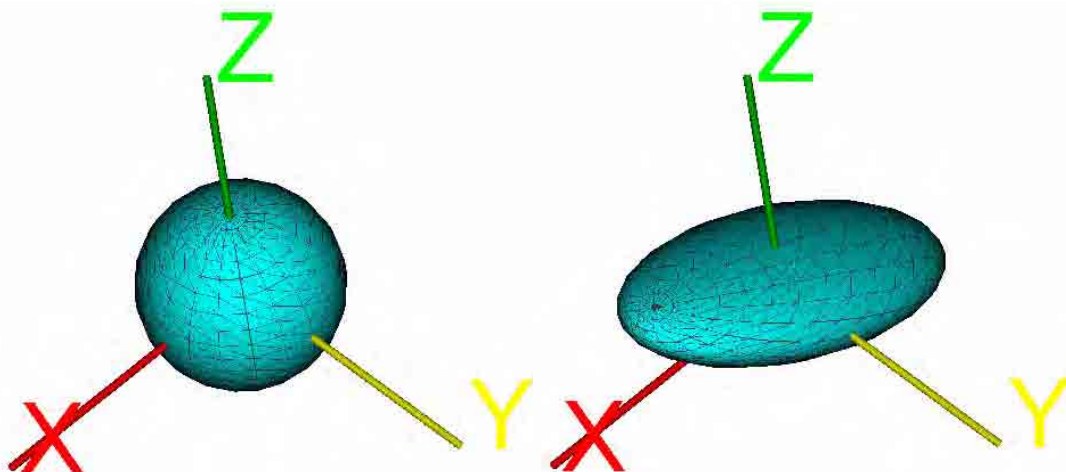


Figura 5.13 – Deformação de uma esfera por cisalhamento simples. O resultado da deformação é um elipsóide. O cisalhamento ocorreu no plano XY perpendicular ao eixo Z e na direção do eixo X. Para tanto, a componente e31 do tensor foi modificada de 0 para 0.515.

Object. Permite selecionar objetos predefinidos que serão deformados de acordo com as componentes do tensor. É possível selecionar uma esfera cuja deformação resultará em um elipsóide (Figuras 5.12a e 5.13); uma “estrela”, isto é, um conjunto de objetos lineares com diferentes orientações iniciais (Figura 5.14); um cubo cujas faces são grades formadas por feixes de linhas paralelas com ângulos de 90 graus (Figura 5.15); e um cubo com círculos desenhados em cada face (Figuras 5.12b e 5.16).

Além dos objetos diretamente selecionáveis na interface gráfica, existem numerosos modelos que podem ser lidos a partir do disco. Tais objetos, armazenados no formato padrão ASCII do VTK (seção 3.4.4) podem ser carregados utilizando o primeiro botão à esquerda com o desenho de um disquete magnético (Figura 5.11). Exemplos de objetos prontos são um feixe de planos paralelos (Figura 5.17) e o modelo de uma concha (Figura 5.18).

Objetos deformados podem também ser gravados no disco, sendo possível, por exemplo, salvar resultados intermediários de um processo deformacional para gerar animações, ou aplicar diferentes tipos de deformação de maneira incremental. Para tanto, é suficiente selecionar o segundo botão à esquerda com o desenho do disquete. Os objetos salvos podem ser relidos e novamente deformados.

A possibilidade de deformar objetos lidos no formato do VTK proporciona um elevado grau de flexibilidade ao programa *Tensor3D*, já que modelos advindos de diferentes ambientes de modelagem geométrica podem ser importados (prévia conversão entre os formatos). Um exemplo é representado pelos modelos digitais gerados com o programa GOCAD, que serão ilustrados na seção 5.9.6.

A seção *Object* contém também algumas funcionalidades destinadas a controlar as propriedades de visualização do objeto deformado. O botão *Wireframe* ativa / desativa a visualização da malha vazada do objeto, independente de sua representação de superfície (Figura 5.19). Já a barra *Opacity* permite selecionar o índice de opacidade da representação de superfície do objeto. A seleção varia entre os valores 0 (objeto totalmente transparente) e 1 (objeto totalmente opaco). Esta função é extremamente útil quando há necessidade de se visualizar detalhes internos ao modelo (Figura 5.20).

Por fim, o botão *Bbox* exibe a caixa 3D que envolve o modelo (*Bounding Box*). Trata-se de um paralelepípedo cujos limites correspondem às coordenadas mínimas e máximas dos pontos do objeto. A caixa sofre deformação junto com o modelo e a observação de sua mudança de forma permite obter uma indicação visual da intensidade da deformação sofrida pelo objeto original (Figura 5.21).

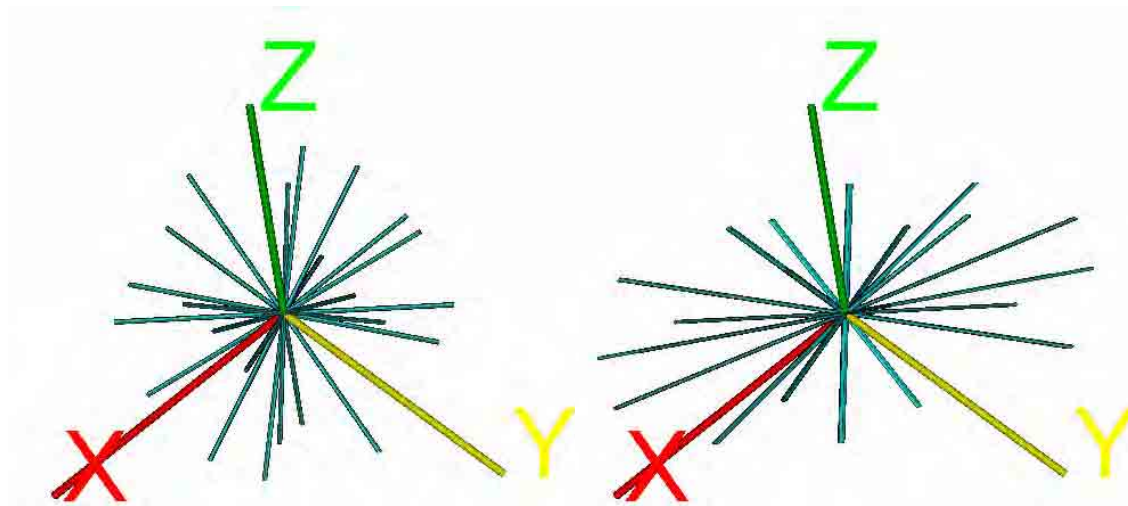


Figura 5.14 – Deformação de uma “estrela” por cisalhamento simples. O cisalhamento é o mesmo considerado na Figura anterior. O objeto inicial é constituído de um conjunto de linhas formando ângulos de 45 graus.

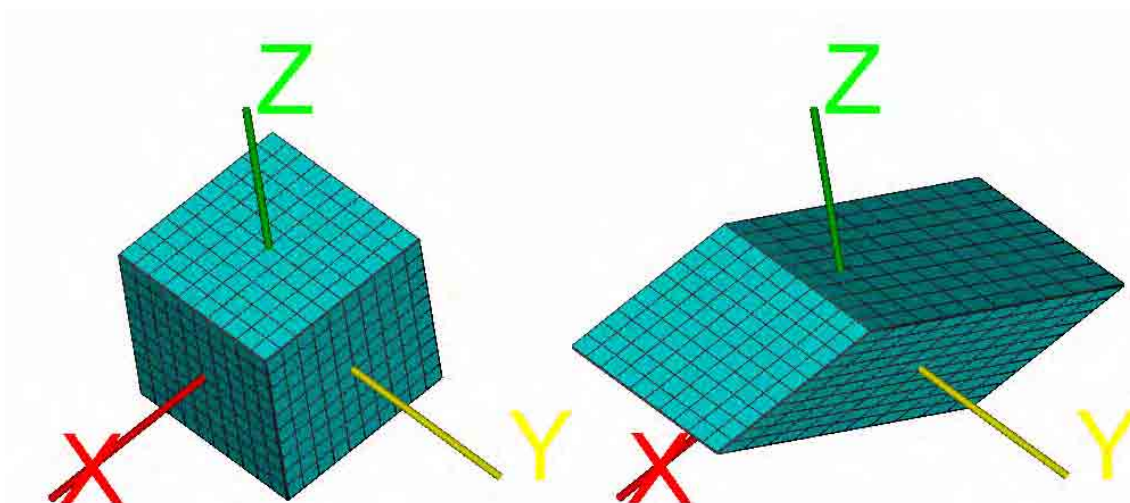


Figura 5.15 – Deformação de um cubo por cisalhamento simples. O objeto inicial possui faces contendo dois feixes de linhas paralelas, com ângulo de 90 graus. Este objeto é interessante para estudar a variação de ângulo entre as linhas após a deformação.

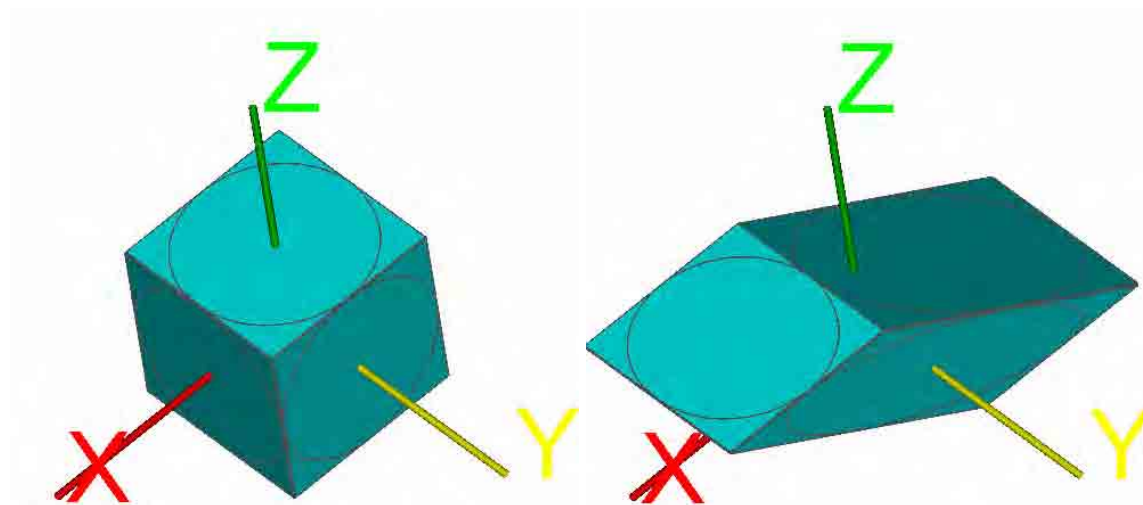


Figura 5.16 – Deformação de um cubo por cisalhamento simples. O objeto inicial é um cubo cujas faces são círculos que se tornam elipses após a deformação. Este objeto é útil para observar a deformação ao longo dos planos cinemáticos.

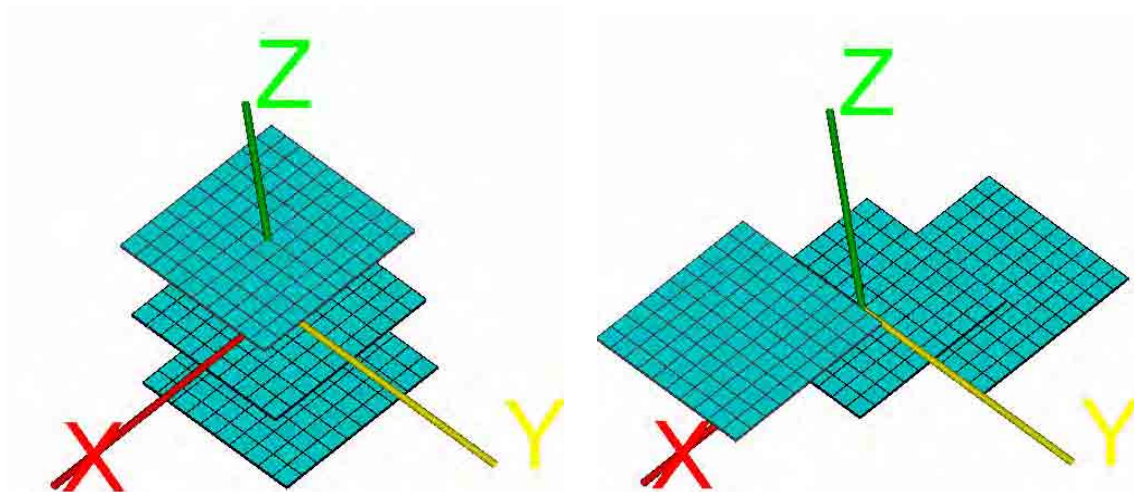


Figura 5.17 – Deformação de um feixe de planos por cisalhamento simples. O objeto inicial é constituído de três planos paralelos entre si e perpendiculares ao eixo Z. A deformação deste objeto mostra claramente que o cisalhamento ocorreu no plano XY e na direção do eixo X (deslocamento do primeiro e terceiro planos).

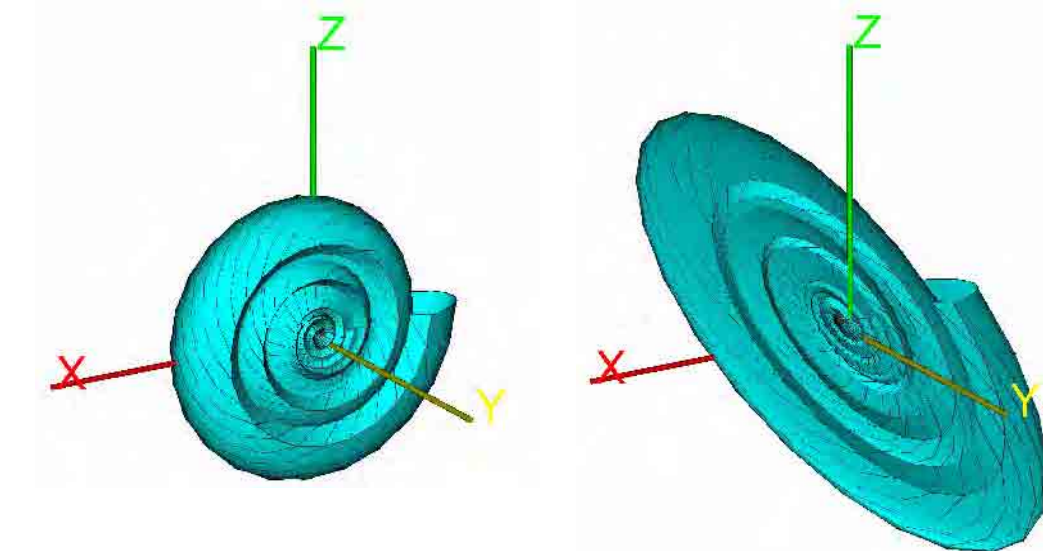


Figura 5.18 – Deformação do modelo digital de uma concha (classe *Cephalopoda*, subclasse *Nautiloidea*) por cisalhamento simples. O modelo foi gerado analiticamente. O cisalhamento ocorreu no plano XY e na direção negativa do eixo Y.

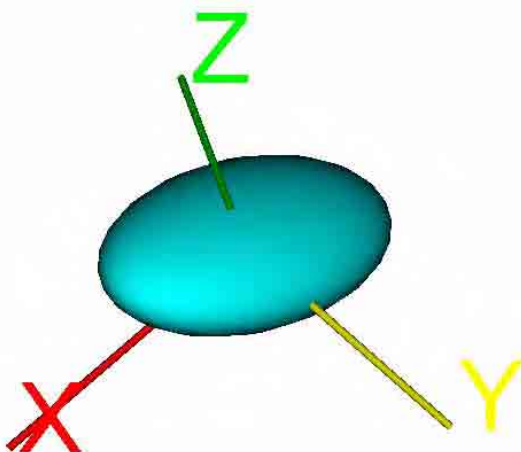


Figura 5.19 – Visualização de um elipsóide de deformação com a modalidade *Wireframe* desativada.

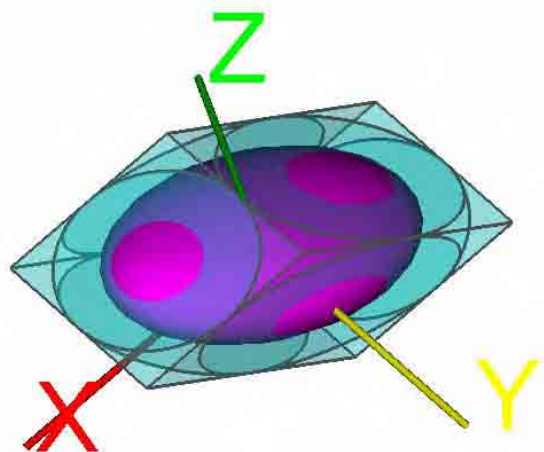


Figura 5.20 – Cubo semitransparente deformado (em azul claro) junto com seu elipsóide de deformação (em roxo).

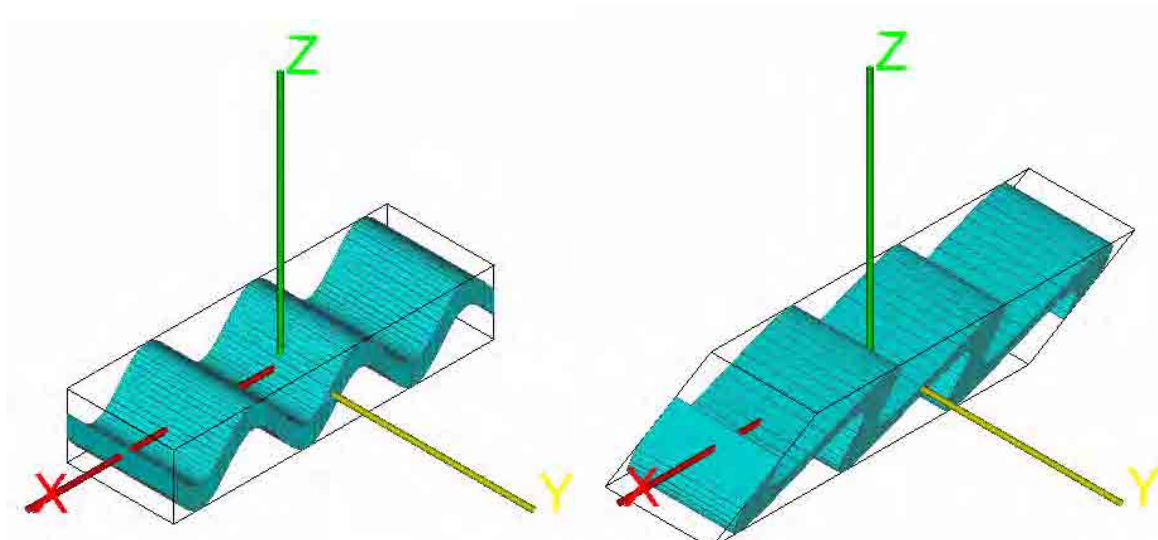


Figura 5.21 – Deformação de uma dobra por cisalhamento simples. O modelo foi gerado analiticamente. O cisalhamento ocorreu no plano XY e na direção negativa do eixo X. A variação de forma da caixa de limites mostra a intensidade da deformação ocorrida.

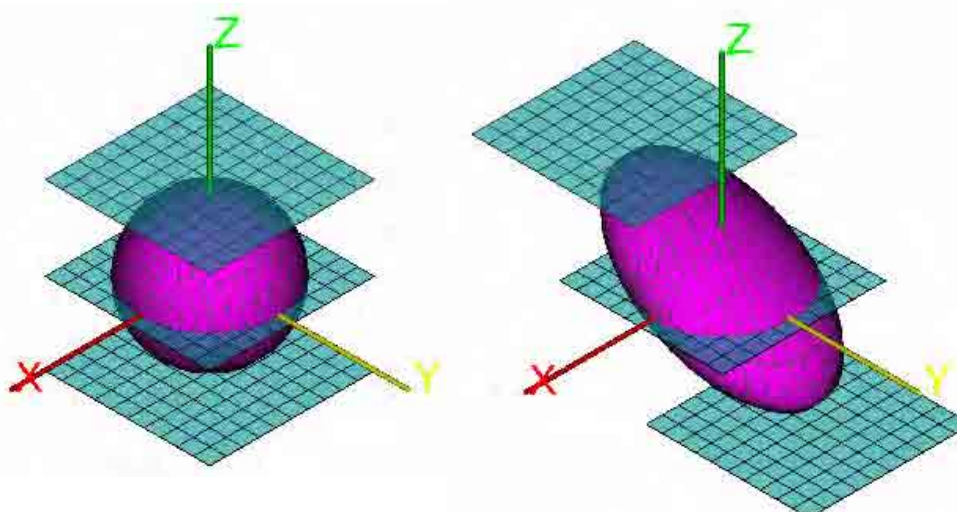


Figura 5.22 – Cisalhamento simples de três planos paralelos entre si e perpendiculares ao eixo Z. A deformação ocorreu no plano XY e na direção negativa do eixo Y. A esfera em roxo (esquerda) sinaliza o estado indeformado e torna-se um elipsóide ao final do processo.

Sphere. Esta seção contém botões para controlar a visualização de uma esfera deformada junto com o objeto selecionado. Isto providencia uma informação do nível de deformação do objeto principal, além de indicar as direções de estiramento / encurtamento máximos (*elipsóide de deformação*). A utilização de uma esfera deformada é também importante para aferir o nível de deformação sofrido por objetos cujo estado indeformado não seja facilmente identificável por simples observação (Figura 5.24).

A esfera é exibida em roxo. O botão *Visible* ativa / desativa sua exibição. O botão *Wireframe* permite visualizar a malha vazada da esfera. Já o botão *Fraction* controla suas dimensões como fração do tamanho do objeto principal. Por fim, o botão *Opacity* controla o

índice de opacidade da esfera. As Figuras 5.11, 5.20, 5.22 e 5.24 mostram exemplos de visualização da esfera deformada.

Axes. Esta seção controla a visualização dos eixos coordenados XYZ e dos eixos principais do tensor, tanto em modalidade de tensão como de deformação. O botão *Visible* habilita / desabilita a exibição de todos os eixos. Já com a barra de rolagem *Fraction* é possível modificar o comprimento e espessura dos eixos relativamente às dimensões do objeto a ser deformado. O tamanho das anotações dos eixos (“X”, “Y”, “Z”, etc.) em relação ao observador é também afetado. Esta funcionalidade é útil quando for necessário visualizar objetos com escalas diferentes.

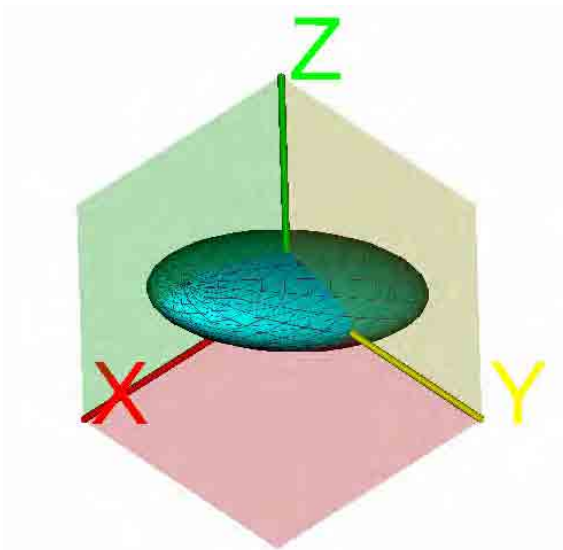


Figura 5.23 – Visualização combinada de uma esfera deformada e dos três planos coordenados semitransparentes.

Planes. Conjunto de botões utilizados para visualizar planos coordenados semitransparentes XY, YZ e XZ (Figura 5.23).

Rendering. Este grupo contém dois botões responsáveis pelo sombreamento do objeto deformado de acordo com uma escala de cores que acompanha a variação de cotas ao longo do eixo Z. O botão *ZColor*, ativa / desativa esta modalidade de visualização. O botão *ColorBar* exibe a legenda que mapeia a variação de cotas para a escala de cores. A Figura 5.24 mostra um exemplo de utilização desta modalidade.

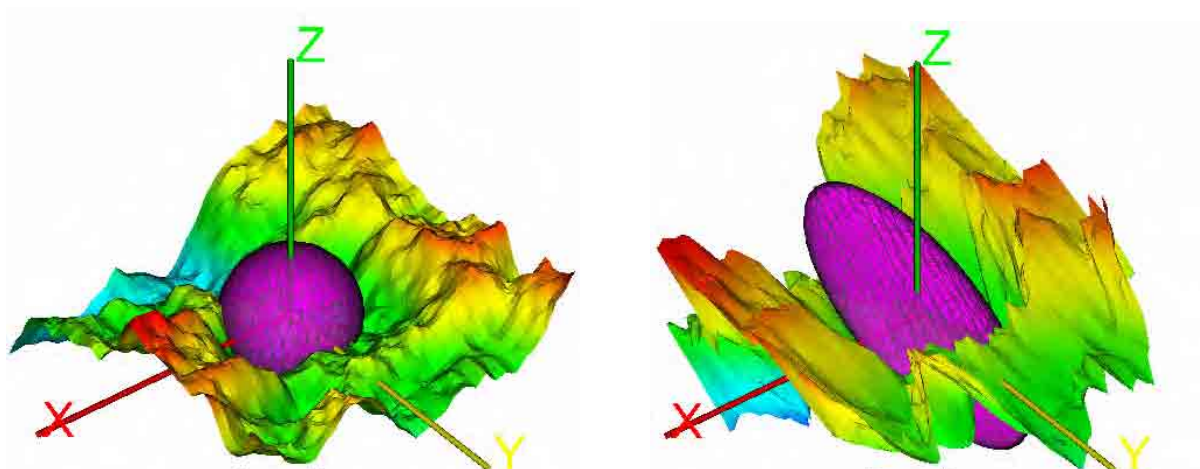


Figura 5.24 – Cisalhamento simples (plano XY, direção negativa do eixo Y) de uma superfície sintética (gerada usando valores aleatórios), sombreada de acordo com uma escala de cores, do azul para o vermelho, que acompanha a variação topográfica ao longo do eixo Z. É também visível o elipsóide de deformação correspondente.

Tensor. Este grupo contém botões que controlam as componentes numéricas do tensor de tensão / deformação utilizado no programa.

O primeiro botão, *RESET*, inicializa as componentes do tensor com valores nulos (modalidade de tensão) ou com os valores da matriz identidade exibidos na relação 5.16 (modalidade de deformação). Esta funcionalidade é extremamente útil, pois permite restaurar simultaneamente os valores originais de todas as componentes do tensor a qualquer momento.

O botão *Symmetric* ativa a modalidade de simetria do tensor. Isto significa que a alteração de uma componente cisalhante σ_{ij} ou e_{ij} ($i \neq j$) determinará automaticamente a alteração correspondente da componente σ_{ji} ou e_{ji} , simétrica em relação à diagonal principal. Como foi descrito anteriormente, a componente simétrica possui os índices trocados. Por exemplo, utilizando um tensor simétrico de deformação, a modificação do elemento e_{31} (deformação cisalhante no plano XY e na direção do eixo X) para 0.455 determinará a alteração automática do elemento simétrico e_{13} (deformação cisalhante no plano YZ e na direção do eixo Z), e *vice-versa*. Em termos de interface gráfica, o deslocamento da barra de rolagem associada a uma componente cisalhante do tensor determinará o movimento automático da barra da componente simétrica em relação à diagonal principal.

As Figuras 5.25 e 5.26 ilustram esta situação para a modalidade de deformação. O caso da tensão é análogo. A ativação da modalidade de cálculo do tensor simétrico determina a visualização dos eixos principais $\hat{e}_1$, $\hat{e}_2$ e $\hat{e}_3$ do elipsóide associado ao tensor. Isto decorre das considerações da seção 5.8 a respeito da simetria de um tensor. Com efeito, os algoritmos de álgebra linear utilizados no VTK para a determinação dos autovalores e autovetores de um tensor assumem que o mesmo seja simétrico. Como os eixos principais de um elipsóide de tensão / deformação são paralelos aos autovetores do tensor correspondente (vide seção 5.7), só faz sentido visualizá-los quando seja possível calcular tais autovetores, isto é, quando o tensor é simétrico.

A visualização dos eixos principais de deformação (notação $\hat{e}_1$, $\hat{e}_2$ e $\hat{e}_3$) é exemplificada nas Figuras 5.25 e 5.26. Na primeira, a modalidade de simetria do tensor é ativada antes de modificar qualquer componente. O programa exibe o valor numérico do comprimento dos três semi-eixos do elipsóide de deformação associado ao tensor ($(1+e_1)$, $(1+e_2)$ e $(1+e_3)$ na Figura). Todos os valores são unitários, já que ainda não ocorreu deformação alguma. A janela de visualização 3D (região esquerda) mostra os eixos principais do elipsóide de deformação, “e1”, “e2” e “e3”. Tais eixos coincidem, em direção e comprimento, com os eixos coordenados X, Y e Z. Isto confirma graficamente que não há deformação cisalhante.

A Figura 5.26 mostra uma situação diferente. A componente de tensão cisalhante e_{31} foi modificada de 0 para 0.454 e a componente simétrica, e_{13} , assumiu o mesmo valor. O programa calcula e exibe, em ordem decrescente, a informação numérica dos autovalores associados a este tensor simétrico: 1.454, 1 e 0.546. Tais autovalores correspondem ao comprimento dos semi-eixos principais do elipsóide de deformação associado ao tensor, e são exibidos na região esquerda da janela. “e1” e “e3”, possuem agora uma nova orientação, visto que a deformação cisalhante implica rotação. Além disto, os semi-eixos principais possuem um novo comprimento, sendo que o de e1 é proporcional a 1.454 (estiramento, pois $1.454 > 1$), o comprimento de e2 é proporcional a 1 (deformação plana) e o de e3 é proporcional a 0.456 (encurtamento, pois $0.456 < 1$). Devido às direções escolhidas para a deformação (ao longo do plano XZ), o eixo principal e2 continua paralelo ao eixo coordenado Y.

A habilitação / desabilitação da modalidade de cálculo simétrico do tensor proporciona grande flexibilidade. Contudo, tal funcionalidade deve ser utilizada com critério. O uso de um tensor não simétrico permite explorar todos os graus de liberdade possíveis na especificação algébrica de um tensor e possui maiores potencialidades em termos didáticos. Por outro lado, a necessidade de se trabalhar com situações realistas estudadas em Geologia Estrutural conduz à utilização de tensores simétricos. A escolha da modalidade de trabalho dependerá das finalidades do estudo.

Uma importante funcionalidade da seção *Tensor* está associada ao botão *Mohr*, o qual permite visualizar os três círculos de Mohr correspondentes aos planos formados pelos eixos principais de tensão ou deformação. O botão está incluído nesta seção pois os círculos somente podem ser desenhados na modalidade de cálculo simétrico do tensor. Com efeito, as equações de determinação dos círculos dependem dos valores de tensões ou deformações normais máxima, intermediária e mínima. Estes valores, por sua vez, correspondem ao comprimento dos semi-eixos dos elipsóides de tensão ou deformação e podem ser determinados apenas para tensores simétricos.

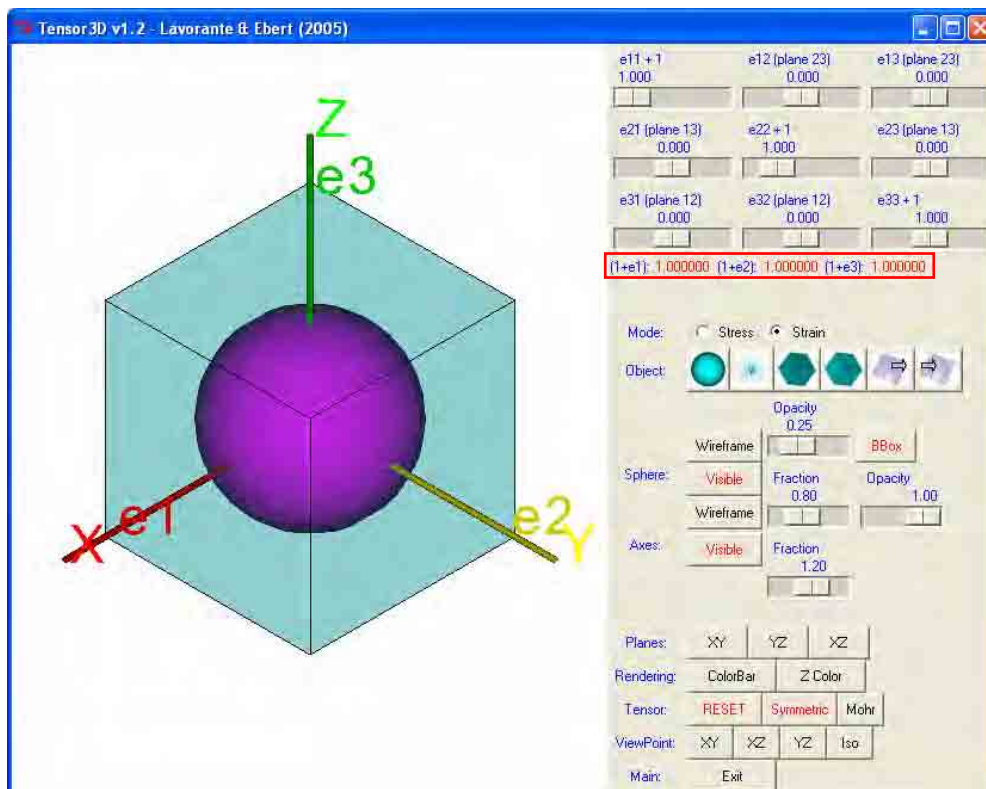


Figura 5.25 – Deformação de um cubo semitransparente usando um tensor simétrico. Na estado indeformado, a ativação do botão *Symmetric* determina a exibição de valores unitários para os comprimentos dos semi-eixos do elipsóide de deformação (retângulo vermelho) e a visualização destes em 3D. Inicialmente, os eixos principais do elipsóide, $e_1e_2e_3$, possuem mesma orientação e comprimento que os eixos coordenados XYZ.

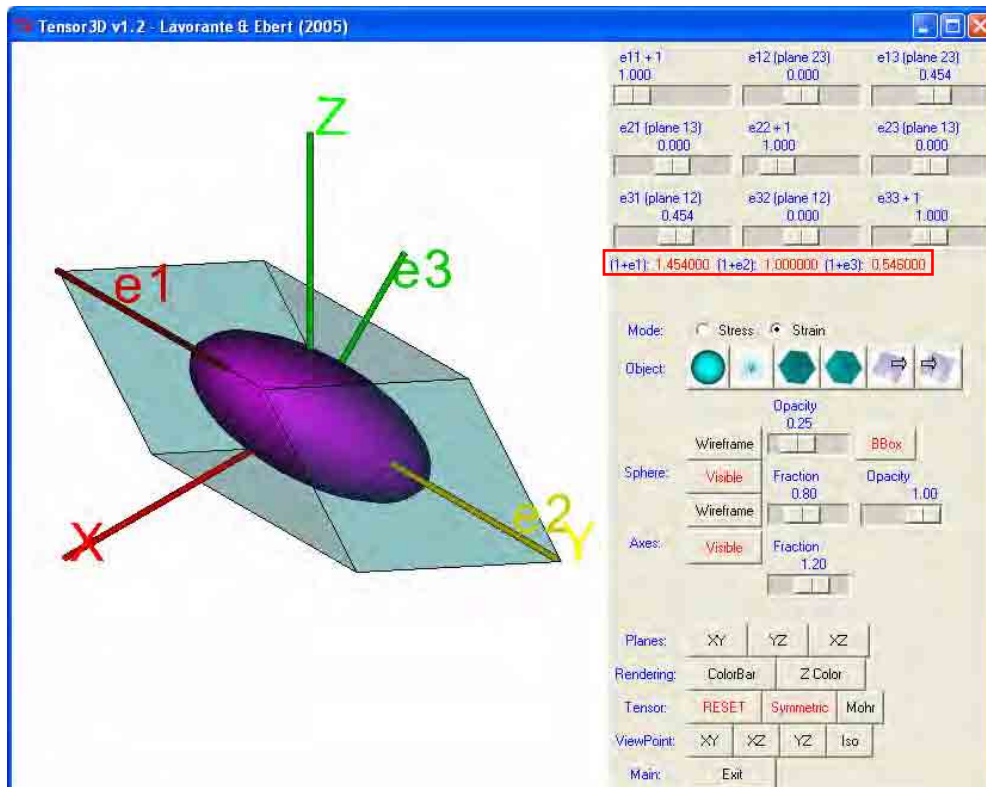


Figura 5.26 – Deformação de um cubo semitransparente usando um tensor simétrico. A especificação de uma deformação cisalhante ao longo do plano XZ (modificação das componentes e_{31} e e_{13} do tensor) determina o cálculo de novos valores para o comprimento dos semi-eixos do elipsóide e sua exibição (retângulo vermelho). Graficamente, esta nova situação é indicada pela modificação de orientação e comprimento dos eixos principais e_1 e e_3 do elipsóide associado ao tensor.

Os três círculos de Mohr são desenhados tanto para a modalidade de tensão como para a de deformação. Em ambos os casos, o ponto de partida consiste nos três autovalores calculados e exibidos pelo programa (Figuras 5.25 e 5.26).

No caso da tensão, os três valores, ou tensões normais máxima, intermediária e mínima $\hat{\sigma}_1$, $\hat{\sigma}_2$ e $\hat{\sigma}_3$, são usados para calcular o centro e raio dos três círculos de acordo com as indicações da seção 5.5 e a equação 5.10. Os três círculos são desenhados usando cores diferentes, sendo o mais externo ($\hat{\sigma}_1\hat{\sigma}_3$) vermelho, o da direita ($\hat{\sigma}_1\hat{\sigma}_2$) vermelho claro e o da esquerda ($\hat{\sigma}_2\hat{\sigma}_3$) cinza (Figura 5.27).

No caso da deformação, os três valores, ou elongações $(1+e_1)$, $(1+e_2)$ e $(1+e_3)$, são usados de forma indireta para calcular o centro e raio dos três círculos. Com efeito, utiliza-se para o cálculo as elongações quadráticas recíprocas da equação 5.11 (seção 5.6). Trata-se do inverso do quadrado das elongações. Portanto, a elongação quadrática recíproca (λ'_1) correspondente ao semi-eixo maior do elipsóide (comprimento $(1+e_1)$) terá valor mínimo, ao passo que para o semi-eixo menor (comprimento $(1+e_3)$) seu valor (λ'_3) será máximo. Como no caso da tensão, os três círculos são desenhados usando cores distintas: vermelho ($\lambda'_3\lambda'_1$), vermelho claro ($\lambda'_3\lambda'_2$) e cinza ($\lambda'_2\lambda'_1$) (Figura 5.28).

ViewPoint. Este grupo de botões permite selecionar visadas perpendiculares aos planos coordenados XY, XZ e YZ. Existe também um botão, denominado *Iso* (representação isométrica), o qual posiciona o ponto de vista ao longo da direção cujos pontos são equidistantes dos eixos coordenados. (Figuras 5.21 e 5.25).

Main. Seção destinada ao controle de execução do programa. Atualmente, a única funcionalidade implementada é o botão *Exit* que termina a execução do *Tensor3D*.

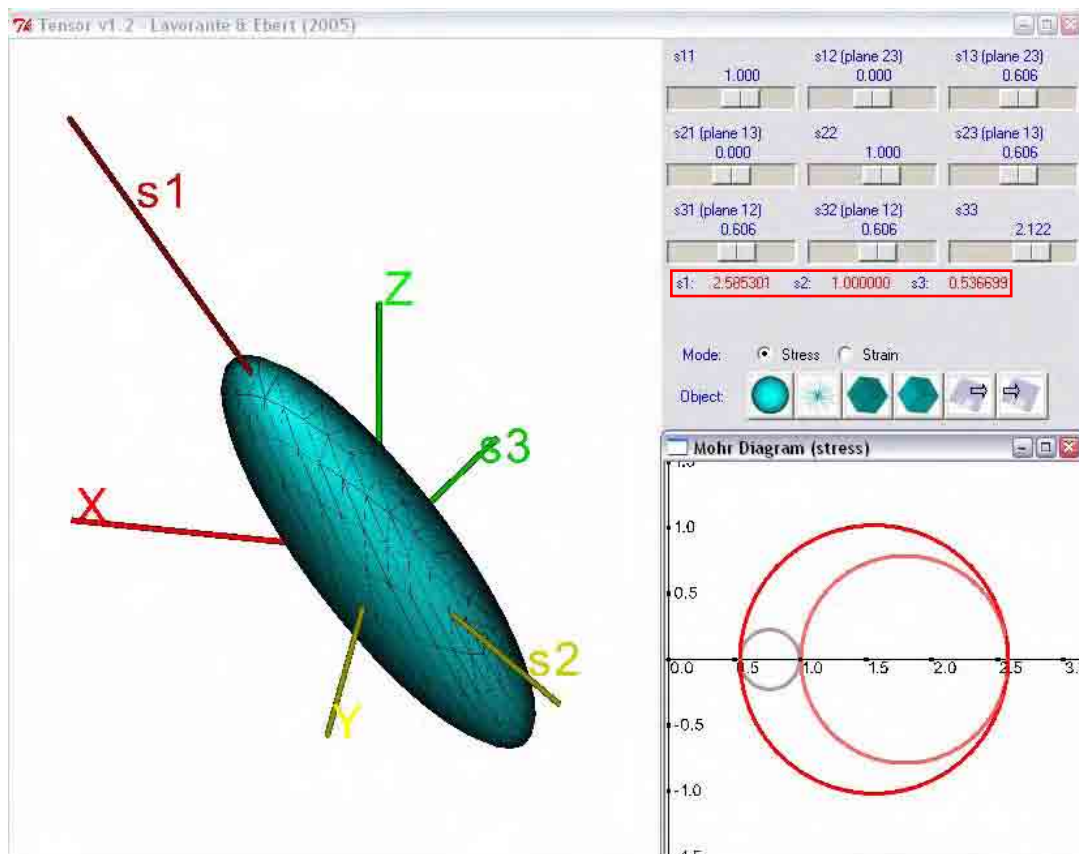


Fig. 5.27 – Círculos de Mohr para a tensão representada graficamente pelo elipsóide azul. Os centros e raios dos círculos são determinados usando o comprimento dos semi-eixos principais “ s_1 ”, “ s_2 ” e “ s_3 ” (valores no retângulo vermelho à direita) do elipsóide de tensão (esquerda).

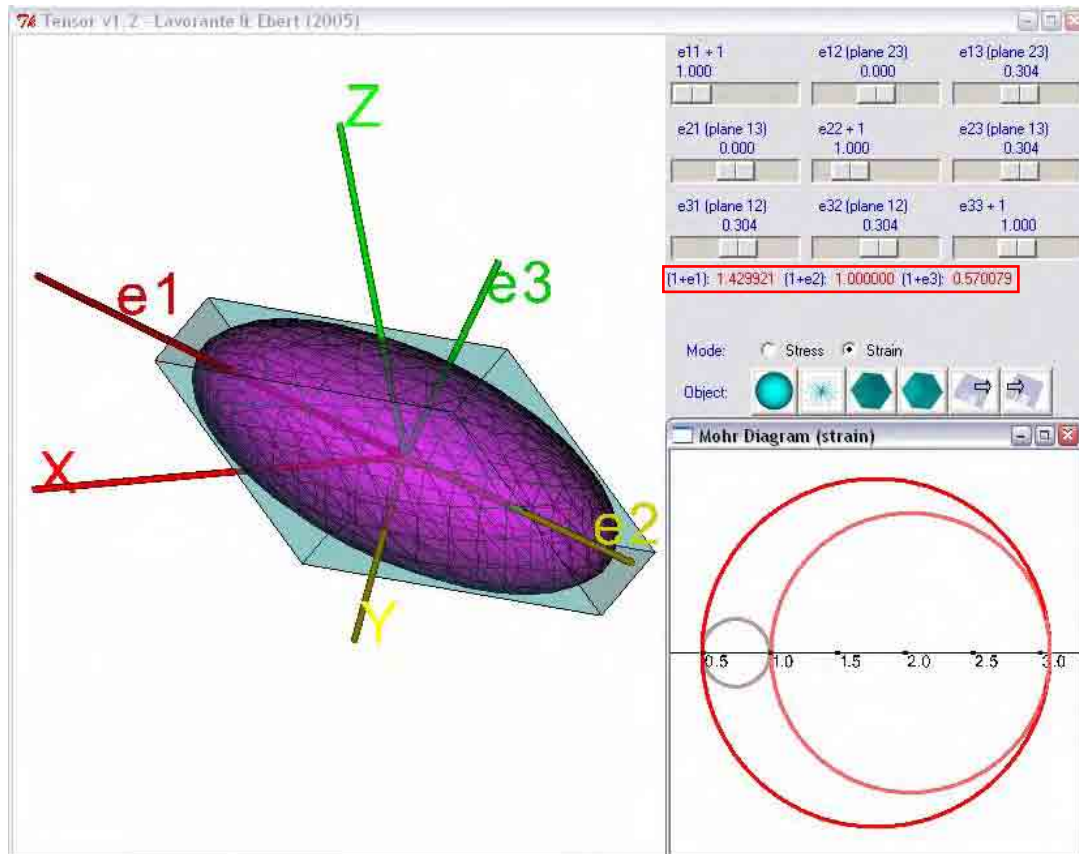


Fig. 5.28 – Círculos de Mohr para a deformação de um cubo semitransparente. Os centros e raios dos círculos são determinados usando o inverso do quadrado do comprimento dos semi-eixos principais “ e_1 ”, “ e_2 ” e “ e_3 ” (valores no retângulo vermelho à direita) do elipsóide de deformação (esquerda).

Janela de visualização 3D. A região 3 da Figura 5.11 e todas as figuras seguintes mostram o conteúdo de uma janela 3D de visualização inserida na porção esquerda da janela principal do programa *Tensor3D*. Esta janela tem como objetivo exibir objetos deformados em 3D de acordo com as especificações do usuário. As alterações realizadas nas componentes do tensor são *instantaneamente* exibidas sob a forma de uma nova deformação do objeto selecionado. Como foi possível observar nas figuras anteriores, o programa visualiza os seguintes objetos:

- Eixos coordenados globais. Adotou-se a convenção de utilizar a cor vermelha para o eixo X, amarela para o eixo Y e verde para o eixo Z. A anotação de cada eixo é realizada usando seu símbolo (X, Y ou Z), visualizado de forma a ser sempre legível pelo usuário independente do ponto de vista selecionado. Em outras palavras, independente da orientação dos eixos, as anotações dos mesmos serão sempre mostradas de frente para o observador.
- Objeto principal. É o objeto que será deformado de acordo com as componentes do tensor. É visualizado usando a cor azul. Normalmente, são exibidas duas representações simultâneas: uma superfície e uma malha vazada (em cinza escuro), que realça os contornos e as características morfológicas do modelo.
- Elipsóide de deformação (Figura 5.22). Trata-se de um objeto secundário, visualizado em roxo, que sugere visualmente a intensidade da deformação do objeto selecionado, além das direções de estiramento / encurtamento máximos. No estado indeformado o elipsóide se reduz a uma esfera posicionada no centro do sistema de coordenadas.
- Caixa de limites (*Bounding Box*, Figura 5.21). Trata-se de um paralelepípedo construído usando as coordenadas mínimas e máximas dos pontos do objeto, nas três direções dos eixos coordenados. Este objeto, cujas arestas são pretas, sofre a deformação junto com o modelo principal. Sua exibição também permite obter uma informação visual da intensidade da deformação normal ou cisalhante ocorrida.
- Eixos principais do tensor (Figuras 5.27 e 5.28). Trata-se dos semi-eixos maior, intermediário e menor do elipsóide de tensão ou deformação associado ao tensor, de acordo com a modalidade selecionada na seção *Mode*. Na janela 3D, são denotados pelos símbolos “s1”, “s2” e “s3” (tensão) ou “e1”, “e2” e “e3” (deformação). Sua orientação depende dos autovetores extraídos da matriz 3x3 do tensor, enquanto seu comprimento é proporcional aos autovalores correspondentes. Na ausência de tensão ou deformação cisalhante, estes eixos possuem a mesma orientação que os eixos coordenados. Neste

caso, seu comprimento depende apenas das componentes normais do tensor. Os eixos principais são exibidos somente na modalidade de simetria, ativada através do botão *Symmetric*.

A janela 3D permite vários graus de interação com o usuário. Estas funcionalidades são implementadas na classe *vtkRenderWindowInteractor*, ilustrada na seção 3.4.1. Através do mouse, é possível rotacionar, transladar e afastar / aproximar (*zoom*) o objeto visualizado. Para tanto utiliza-se, respectivamente, os botões esquerdo, central e direito. Caso haja apenas dois botões, o botão central poderá ser emulado usando a tecla SHIFT e o botão esquerdo. As funções de rotação, translação e *zoom* são ativadas em proporção à distância do ponteiro do mouse do centro da janela 3D. Por exemplo, a seleção do botão esquerdo quando o ponteiro estiver posicionado exatamente no centro da janela não produz rotação alguma. Afastando-se gradualmente o ponteiro do centro (e mantendo-se o botão pressionado), haverá uma lenta rotação do objeto na direção selecionada. Afastando-se o ponteiro até o limite da janela 3D, a velocidade de rotação será máxima.

Através do teclado, são ativadas outras funcionalidades. Destaca-se a tecla 'w', que permite visualizar todos os objetos da janela em modalidade *Wireframe* (malha vazada, Figura 3.1); 's' que habilita a modalidade de superfícies sombreadas (*Flat Shading*, Figura 3.2); 't' e 'j' que controlam a sensibilidade do mouse quanto à rotação do objeto; e 'q' ou 'e' que terminam a execução do programa.

5.9.3 Estrutura do programa

O funcionamento do programa *Tensor* ocorre em virtude de uma troca de informações entre módulos (classes) do sistema VTK especializados na execução de tarefas dedicadas: obtenção de informações do usuário, processamento dos dados obtidos, modificação correspondente de entidades geométricas, visualização das mesmas, etc. O tráfego de informações ocorre ao longo de um sistema de fluxos de dados (*pipeline*) que se inicia nas classes responsáveis pela criação / leitura do disco da geometria e topologia de objetos 3D, e termina nas classes destinadas à atualização do conteúdo da janela de visualização do programa. Esta seção descreve as classes principais do VTK utilizadas para tanto, bem como suas interconexões.

O diagrama de fluxo de informações e interconexão entre as classes utilizadas é mostrado na Figura 5.29. Existem sete blocos de processamento principais, delimitados por linhas azuis pontilhadas. Segue a descrição dos mesmos:

(1) Criação e deformação de objetos de acordo com informações matemáticas do tensor. Este bloco lê do disco as informações geométricas e topológicas do objeto a ser deformado, além de criar os modelos 3D para a esfera acessória que se tornará um elipsóide e a caixa de limites do objeto principal (Figuras 5.21 e 5.26). Objetos armazenados em disco são lidos pela classe *vtkPolyDataReader*, ao passo que a esfera é criada através da classe *vtkSphereSource*. A representação interna de todos os objetos 3D é armazenada na classe *vtkPolyData* (não representada no diagrama da Figura 5.29, mas usada internamente), que é o formato de troca de informações entre *vtkPolyDataReader* / *vtkSphereSource* e a classe *vtkTensorGlyph*. Esta realiza a deformação dos objetos de acordo com a especificação matemática de um tensor. Tal especificação ocorre definindo-se simplesmente um vetor (lista) de nove números em ponto flutuante (as componentes do tensor) na classe *vtkFloatArray*. Uma classe intermediária entre *vtkFloatArray* e *vtkTensorGlyph* é *vtkStructuredPoints*, normalmente utilizada para armazenar informações volumétricas, isto é, dados posicionados em malhas de intervalos regulares nas direções X, Y e Z. Neste caso, considera-se como entrada da classe *vtkTensorGlyph*, um volume composto por um único ponto, localizado no centro do sistema de coordenadas cartesianas e contendo a informação matricial das nove componentes escalares do tensor.

A classe *vtkTensorGlyph* realiza a deformação do objeto e sua saída é enviada para dois filtros distintos: *vtkExtractEdges* e *vtkElevationFilter*. O primeiro extrai as arestas do objeto deformado para posterior visualização da malha vazada do mesmo. Já *vtkElevationFilter* realiza um mapeamento entre a variação de elevação de coordenadas do objeto (ao longo do eixo Z) e uma escala de cores predefinidas do azul para o vermelho.

Outra instância da classe *vtkTensorGlyph* recebe as mesmas informações tensoriais armazenadas em *vtkStructuredPoints* e a especificação geométrica / topológica de uma esfera para criar um elipsóide de deformação. Por fim, uma terceira instância da classe *vtkTensorGlyph* recebe como entrada a caixa de limites (*Bounding Box*) do objeto lido do disco, além das informações tensoriais advindas de *vtkStructuredPoints*. A caixa de limites é gerada pelo filtro *vtkOutlineFilter*, diretamente conectado à classe de leitura *vtkPolyDataReader*. A classe *vtkOutlineFilter* cria um objeto poligonal correspondente ao *Bounding Box* do modelo. A caixa, inicialmente um paralelepípedo, será deformada de acordo com as componentes do tensor.

As classes *vtkExtractEdges*, *vtkElevationFilter* e *vtkTensorGlyph* (duas instâncias) enviam seu processamento para instâncias da classe *vtkPolyDataMapper*, que transformam as informações geométricas em primitivos gráficos a serem utilizados pelo *hardware* dedicado

(placa aceleradora), para visualização posterior. Em seguida, tais primitivos são repassados para instâncias da classe *vtkActor*, as quais definem as propriedades finais de visualização de cada objeto (cor, índice de opacidade, relação com as fontes de luz, etc). Mais especificamente, as propriedades dos objetos são definidas através de instâncias da classe *vtkProperty* (não representadas no diagrama), havendo uma instância para cada ator. Tal instância é criada automaticamente pelo VTK, junto com o ator correspondente.

(2) Criação dos eixos coordenados. O sistema VTK possui uma classe pronta, *vtkAxes*, para criar a representação geométrica de um sistema de eixos coordenados XYZ. Cada eixo recebe uma cor distinta (vermelho para o eixo X, amarelo para o eixo Y e verde para o eixo Z). Como os eixos são representados por linhas, sua visualização pode ser prejudicada quando a câmera não estiver posicionada próximo das mesmas. Para resolver este problema, constroem-se objetos cilíndricos em volta dos eixos coordenados, que desta forma adquirem uma aparência sólida (“tubos”) e são mais facilmente visíveis. A classe utilizada para tanto, é *vtkTubeFilter*. Esta permite especificar a resolução lateral dos cilindros (*tubes*), isto é o número de suas faces laterais. O programa *Tensor3D* usa oito faces laterais. Finalmente, as informações geométricas geradas para os eixos são repassadas para uma instância da classe *vtkPolyDataMapper* e em seguida é criado o ator correspondente.

(3) Criação dos eixos principais do tensor. Os eixos principais do tensor, isto é, os semi-eixos do elipsóide de tensão / deformação que possuem a mesma orientação que os autovetores do tensor, constituem um sistema de coordenadas diferente do sistema global XYZ. Sua criação não pode ser realizada através da classe *vtkAxes*, já que esta não permite orientar os eixos separadamente em relação ao sistema global. Portanto, cada eixo foi criado através de uma instância da classe *vtkLineSource*. Esta define o ponto inicial (a origem do sistema de coordenadas) e final de um segmento de linha no espaço 3D. Inicialmente, quando não há deformação normal nem cisalhante especificadas no tensor do programa, os eixos principais “s1”, “s2” e “s3” ou “e1”, “e2” e “e3”, coincidem com os eixos coordenados XYZ (Figura 5.25). Mas durante a execução do programa, a orientação dos eixos principais pode mudar (Figura 5.26), sendo necessário modificar o ponto final definido em cada instância de *vtkLineSource*. Tais segmentos de linha são transformados em cilindros como foi ilustrado no caso do bloco (2), para os eixos coordenados, mas antes é necessário realizar um redimensionamento das linhas criadas de acordo com o fator de escala utilizado no programa.

Para tanto, a saída da classe *vtkLineSource* é enviada para a classe

vtkTransformPolyDataFilter, a qual permite realizar rotação, translação e *scaling* de um objeto *vtkPolyData* (no caso, os segmentos de linha dos eixos principais). A transformação utilizada é definida em uma classe separada, *vtkTransform*, que contém a matriz 4x4 a ser empregada. O redimensionamento aplicado aos eixos principais do tensor resulta em um aumento de comprimento. O fator de escala para todos os eixos é proporcional às dimensões do modelo e pode ser modificado pelo usuário.

Os eixos principais possuem cores distintas, ligeiramente mais escuras que as cores utilizadas para os eixos coordenados: vermelho para s_1 / e_1 , amarelo para s_2 / e_2 e verde para s_3 / e_3 .

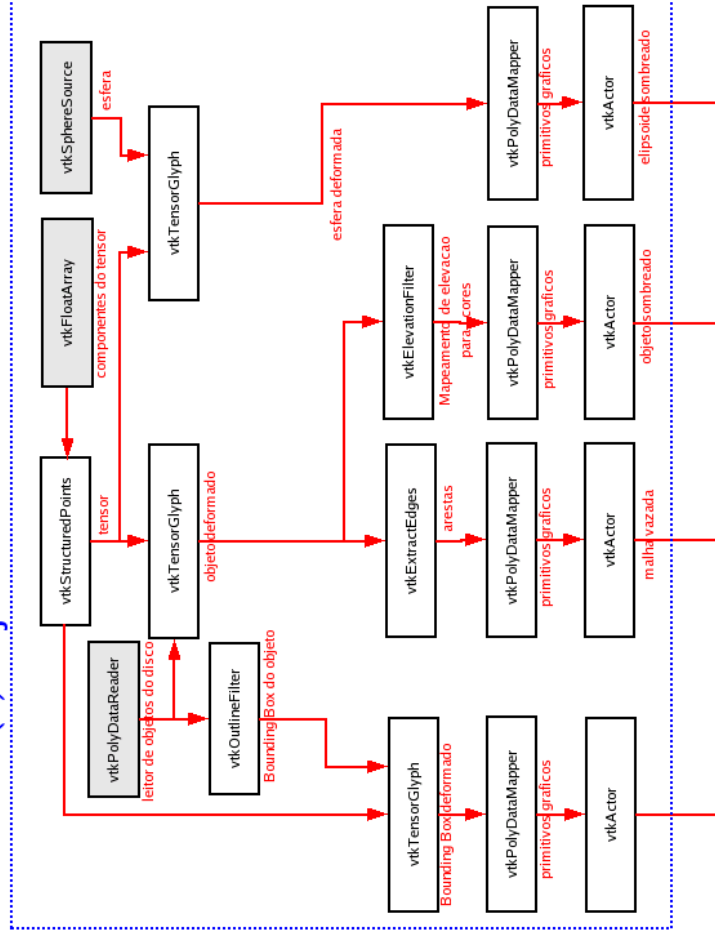
Novamente, após a criação dos cilindros em torno dos eixos (através de instâncias da classe *vtkTubeFilter*), as informações geométrica são convertidas em primitivos gráficos (classe *vtkPolyDataMapper*) e são gerados três atores (*vtkActor*) para a visualização.

(4) Criação de anotações para os eixos coordenados. Como foi visto, a cada eixo coordenado é associada uma cor específica. Além disto, a exibição de uma letra (ou anotação: “X”, “Y” e “Z”) na janela de visualização 3D permite a identificação imediata de cada eixo (Figura 5.11 e seguintes). Para criar a representação geométrica das anotações dos eixos, foram utilizadas instâncias da classe *vtkVectorText*, a qual permite transformar facilmente *strings* de texto (seqüências de caracteres) em objetos poligonais.

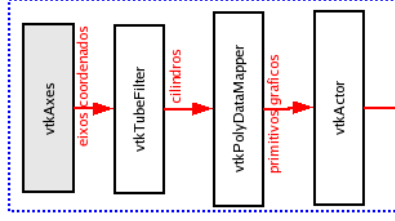
Um problema das anotações “X”, “Y” e “Z” geradas pela classe *vtkVectorText* é que sua orientação é fixa no espaço tridimensional. Isto significa que modificando seu ponto de observação no espaço, o usuário pode olhar “atrás” de cada anotação, já que a mesma é, na verdade, um objeto contido em um plano. A orientação dinâmica de cada anotação, de forma que a mesma exiba sempre a mesma face para o observador, é realizada através da classe *vtkFollower*, que substitui a classe *vtkActor*.

(5) Criação de anotações para os eixos principais do tensor. O conjunto de classes utilizadas neste bloco é o mesmo que foi descrito no bloco anterior. As *strings* de anotação utilizadas para os eixos principais de tensão são “ s_1 ”, “ s_2 ” e “ s_3 ” (Figura 5.27), enquanto para os eixos principais de deformação as anotações são “ e_1 ”, “ e_2 ” e “ e_3 ” (Figura 5.28).

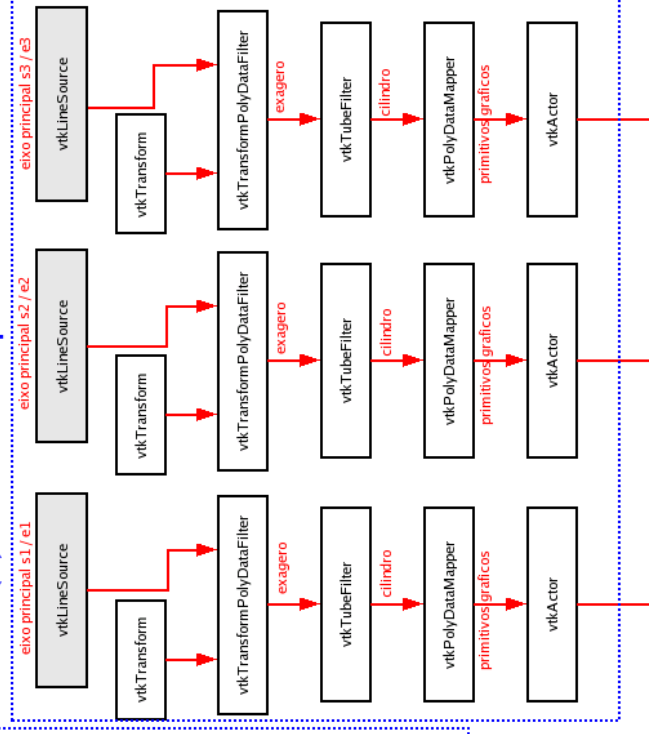
(1) Objeto e esfera deformados



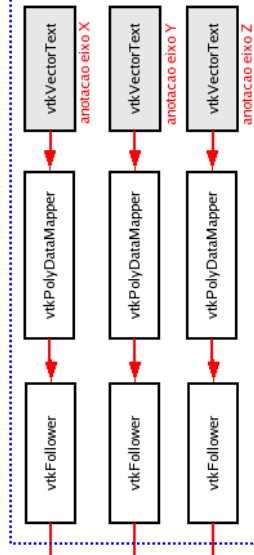
(2) Eixos Coordenados



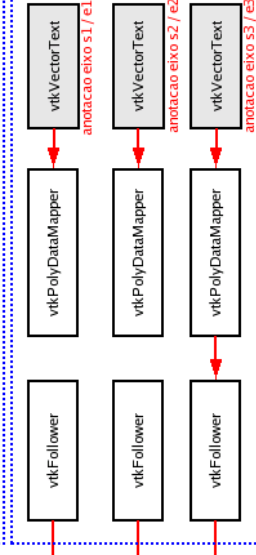
(3) Eixos Principais do tensor



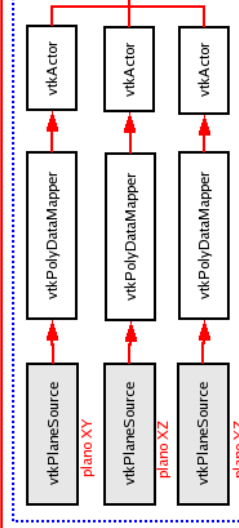
(4) Anotações eixos coordenados



(5) Anotações eixos principais



(6) Planos Semitransparentes



(7) Visualização 3D

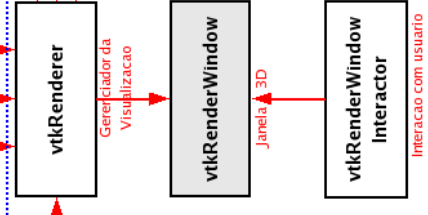


Figura 5.29 – Diagrama de interconexão de classes do programa *Tensor3D*. As caixas pretas indicam as classes do VTK utilizadas. Classes com fundo cinza são fontes ou destinos finais de informações trafegando no *pipeline*. As setas vermelhas mostram a interconexão e a direção do fluxo de informações entre as classes. Os blocos delimitados por linhas pontilhadas azuis identificam os grupos de classes responsáveis pela criação dos objetos principais e sua visualização.

(6) Criação de planos semitransparentes. Os três planos coordenados XY, XZ e YZ, mostrados na Figura 5.23, foram criados utilizando três instâncias da classe *vtkPlaneSource*. Esta classe permite especificar a orientação de um plano no espaço através de seu vetor normal. O vetor normal ao plano XY é (0, 0, 1), o eixo Z. O vetor normal ao plano XZ é (0, 1, 0), isto é, o eixo Y. Por fim, o vetor normal ao plano YZ é (1, 0, 0), o eixo X. A classe *vtkPlaneSource* gera a representação geométrica de um plano (na estrutura da classe *vtkPolydata*) a partir de sua definição matemática. A opacidade de cada plano coordenado foi definida utilizando métodos específicos da classe *vtkProperty* associada ao ator correspondente. Adotou-se uma opacidade de 30%, isto é, uma transparência de 70%.

(7) Visualização 3D. Os blocos de processamento descritos até agora terminam com a criação de “atores” (seção 3.4.1), que devem ser exibidos em três dimensões. Como mostra a Figura 5.29, as informações contidas em cada ator confluem para uma importante classe, chamada *vtkRenderer*. Esta cria uma cena 3D a partir da definição geométrica de atores, propriedades de visualização dos mesmos, posição e propriedades da câmera (ponto de observação do usuário), posição e propriedades das fontes de iluminação definidas, etc. Todas estas informações são gerenciadas e integradas de forma a obter a visualização final.

A cena criada é exibida em uma janela gráfica através da classe *vtkRenderWindow*, que contém todos os métodos e atributos necessários para gerenciar janelas gráficas no sistema operacional escolhido. Por fim, a interatividade proporcionada pelo programa *Tensor3D*, isto é, a possibilidade de modificar a cena exibida interagindo com o teclado ou mouse, é implementada pela classe *vtkRenderWindowInteractor*. Esta é um gerenciador de eventos, que aguarda comandos do usuário e invoca métodos de atualização do *pipeline* de fluxo de dados (na direção contrária à das setas que confluem para a classe *vtkRenderer*), provocando a redefinição do conteúdo da janela 3D de visualização.

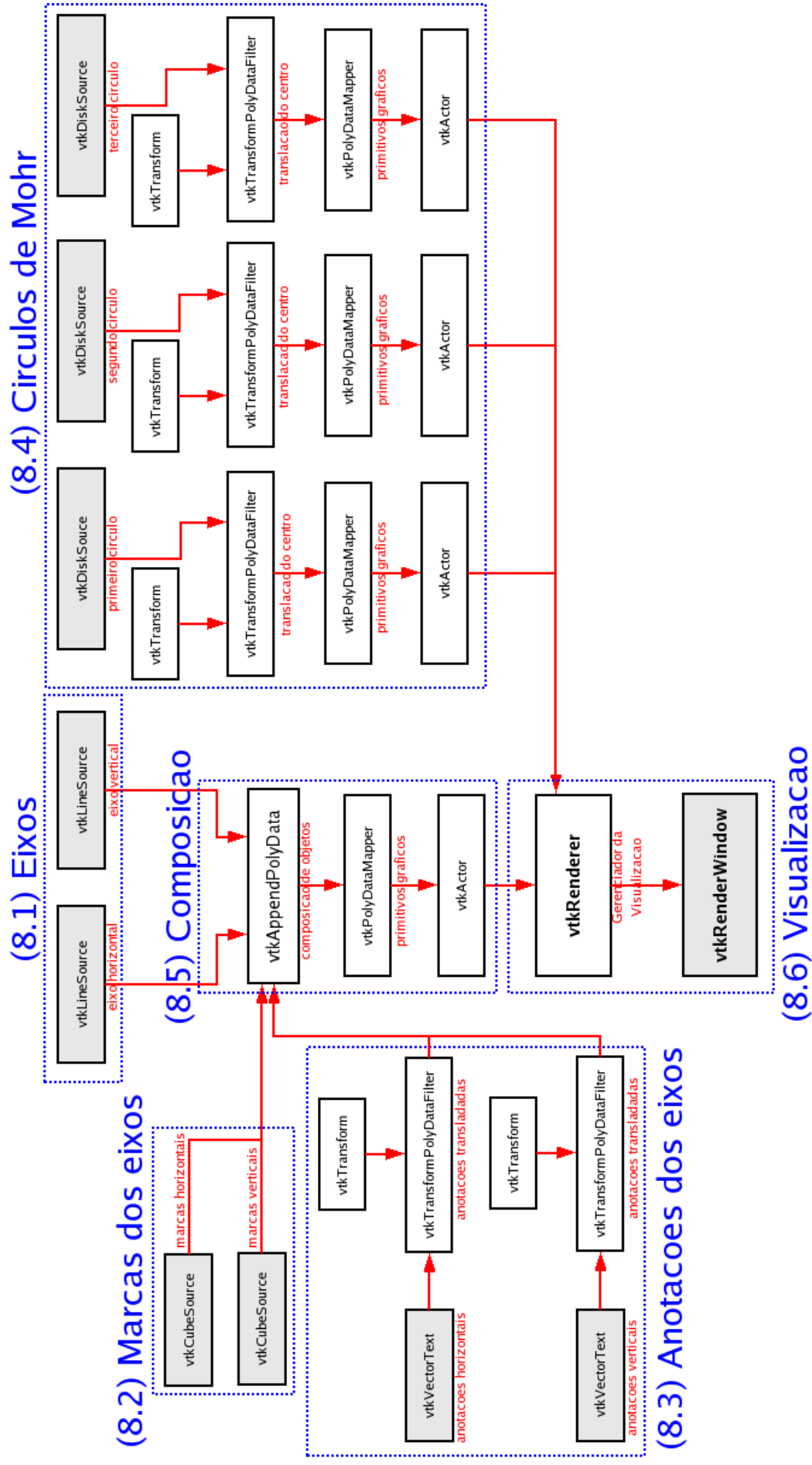


Figura 5.30 – Diagrama de interconexão de classes do módulo para visualização dos círculos de Mohr.

(8) O subconjunto de classes do VTK destinado à exibição dos círculos de Mohr representa um *pipeline* separado de dados (Figura 5.30), cujo destino final é uma outra janela gráfica. O sistema VTK possui uma classe completa para a criação de gráficos 2D, *vtkXYPlotActor*. Contudo este mestrando optou pela construção *ex novo* do diagrama de Mohr para ter um maior controle sobre suas componentes. O resultado, exibido nas janelas das Figuras 5.27 e 5.28, é uma visualização em planta (no plano XY do sistema global de coordenadas) de objetos que na verdade são tridimensionais. Na Figura 5.30, os blocos delimitados por linhas azuis pontilhadas agrupam as classes responsáveis pela criação das principais componentes do gráfico.

(8.1) Eixos coordenados. Os eixos foram construídos usando duas instâncias da classe *vtkLineSource*, definindo-se a representação geométrica de uma linha horizontal e outra vertical. Os eixos do gráfico correspondem às informações de tensão / deformação normal (eixo horizontal) e cisalhante (eixo vertical), respectivamente.

(8.2) Marcas dos eixos. Trata-se de um conjunto de marcas utilizadas para indicar valores ao longo dos eixos do diagrama. As marcas possuem a aparência de pequenos quadrados, mas na verdade estes são cubos criados usando instâncias da classe *vtkCubeSource*. A Figura 5.30 exibe apenas duas instâncias (uma horizontal e outra vertical) por simplicidade de representação, mas ao todo foram utilizadas 25 instâncias para cada eixo, sendo que os intervalos de valores representados variam de -6.0 a 6.0, com incremento de 0.5 unidades. Os cubos foram posicionados corretamente ao longo dos eixos utilizando métodos próprios da classe *vtkCubeSource*.

(8.3) Anotações dos eixos. As anotações são indicações de valores associadas às várias marcas posicionadas sobre os eixos. Sua representação poligonal foi criada usando a classe *vtkVectorText*. O texto criado é sempre posicionado no centro do sistema de coordenadas, o que levou à necessidade de se realizar uma translação do mesmo para efetuar seu alinhamento com as marcas dos eixos. A translação é definida através da classe *vtkTransform*, e é realizada por instâncias de *vtkTransformPolyDataFilter*, que reposicionam adequadamente as anotações das marcas. A Figura 5.30 mostra apenas duas instâncias da classe *vtkVectorText*, mas na verdade foram utilizadas 49 instâncias, uma para cada marca.

(8.4) Círculos de Mohr. Os três círculos do diagrama foram construídos usando instâncias da classe *vtkDiskSource*, a qual especifica o modelo poligonal de um disco (círculo preenchido) que possui dois raios, um interno e outro externo. Quando o valor dos raios coincidir, o disco construído é totalmente preenchido, ao passo que um raio interno menor que o externo permite criar um “buraco” circular no disco, sendo assim possível modelar apenas uma circunferência de espessura variável. Esta foi a técnica utilizada para construir os círculos de Mohr. A classe *vtkDiskSource* possui métodos para especificar o tamanho dos discos, mas não sua posição. Mais uma vez, foi necessário efetuar translações através das classes *vtkTransform* e *vtkTransformPolyDataFilter*. As posições e raios dos círculos de Mohr dependem do comprimento dos semi-eixos do elipsóide de tensão / deformação e são modificados dinamicamente a cada alteração das componentes do tensor na janela do programa *Tensor3D*.

(8.5) Composição de objetos. Como é possível verificar na Figura 5.30, de todas as entidades mencionadas até agora, apenas os três círculos de Mohr são processados até o nível de atores, criando-se três diferentes instâncias da classe *vtkActor*. Os demais objetos (eixos coordenados, marcas ao longo dos mesmos, anotações numéricas correspondentes) não possuem atores. A razão disto é que todos estes objetos foram juntados em um *único* modelo. A composição é realizada pela classe *vtkAppendPolyData*, que aceita múltiplas instâncias da classe *vtkPolyData* como entrada e gera apenas uma instância na saída. Isto permitiu criar um único ator para todos os objetos de posição estática (eixos, marcas e anotações), já que os mesmos não sofrem modificações de escala ou translação ao se alterar as componentes do tensor. A utilização da classe *vtkAppendPolyData* simplifica a modelagem de numerosas entidades geométricas, não sendo necessário criar dezenas de atores diferentes. Isto proporciona economia de utilização de memória. O preço a ser pago é a impossibilidade de modificar separadamente as propriedades geométricas dos objetos. Por isto foi necessário definir atores distintos para os círculos de Mohr, cuja escala e posição são variáveis.

(8.6) Visualização. As informações geométricas e de visualização dos atores (círculos de Mohr e composição de eixos, marcas e anotações numéricas) confluem, no diagrama da Figura 5.30, para a classe *vtkRenderer*, que organiza todos os dados e cria uma imagem que será exibida dentro de uma janela gráfica (*vtkRenderWindow*). A porção do *pipeline* destinada à visualização não inclui a classe *vtkRenderWindowInteractor*, já que não é necessária a interação do usuário com o gráfico gerado.

Além do conjunto de classes da biblioteca VTK ilustradas até agora, o programa *Tensor3D* conta com todas as funcionalidades da linguagem de programação Tcl/Tk. Na verdade, trata-se de um programa *escrito* nesta linguagem, que importa as funcionalidades do VTK na forma de módulos ou bibliotecas. O módulo do VTK não foi escrito em Tcl/Tk, pois seus algoritmos de computação gráfica precisam usufruir da otimização de execução proporcionada pela utilização de código compilado (binário). Apesar disto, tais funcionalidades podem ser facilmente ativadas e manipuladas a partir de programas em Tcl/Tk, através de uma biblioteca de interface (ou *wrapping*) que acompanha o VTK.

A linguagem Tcl/Tk provou ser uma ferramenta extremamente flexível para o desenvolvimento de aplicativos. Contudo, por ser uma linguagem interpretada (cujas instruções são convertidas para código binário passo a passo. Vide seção 3.4.3), houve, durante o desenvolvimento do programa *Tensor3D*, uma preocupação constante em relação à velocidade de processamento numérico realizado pelo aplicativo. Tal processamento está relacionado essencialmente ao cálculo dos autovalores e autovetores da matriz 3x3 do tensor matemático embutido no programa, para determinar a orientação e comprimento dos eixos principais de tensão / deformação. Para cada modificação nas componentes do tensor, tais cálculos são executados quatro vezes. As primeiras três ocorrem internamente às instâncias da classe *vtkTensorGlyph* (Figura 5.29), que realizam a deformação de objetos mas não permitem o acesso aos autovalores e autovetores calculados. Tais informações são também necessárias para a correta visualização dos eixos principais e dos círculos de Mohr. Para tanto, os cálculos são executados novamente, fazendo uso de outra biblioteca de funções, denominada *LA* (HUME LINEAR ALGEBRA TCL PACKAGE, 2005). Esta contém rotinas de álgebra linear em Tcl (parte lógica e não gráfica da linguagem Tcl/Tk) e, apesar de não ser um módulo compilado, observou-se uma velocidade de execução satisfatória em plataformas com processador Pentium 4. A biblioteca *LA* é de domínio público e foi obtida através da rede Internet. Suas funções estão contidas no *script la.tcl* presente no mesmo diretório que o programa *Tensor3D*.

Outra característica importante da linguagem Tcl/Tk é a disponibilidade de numerosos componentes gráficos (ou *gadgets*) tais como barras de rolagem, botões, campos para introdução de valores ou exibição de informações, janelas prontas para a leitura / gravação de arquivos, etc. A utilização destes componentes (associados à porção “Tk” do nome da linguagem) foi fundamental para o desenvolvimento de uma interface gráfica “amigável” de interação com o usuário no programa *Tensor3D*. Os componentes gráficos são de fácil criação / manipulação e uma referência importante para seu conhecimento foi o texto de Welch

(1997), bem como o acesso ao grupo de discussão na rede Internet sobre a linguagem Tcl/Tk (TCL/TK DISCUSSION GROUP, 2005), *comp.lang.tcl*.

5.9.4 Instalação

Esta seção descreve os procedimentos para instalação do programa *Tensor3D* em ambiente Windows. Embora o programa tenha sido desenvolvido e exaustivamente testado em ambiente Linux, a instalação do mesmo é mais simples no sistema operacional Windows. A utilização de uma linguagem de programação multiplataforma (Tcl/Tk, 2005; WELCH, 1997) permite executar o programa em sistemas operacionais distintos sem a necessidade de adaptar o código fonte. As versões do Windows utilizadas para testes foram 98, 2000 e XP.

O programa está contido no CD que acompanha esta dissertação. Ao acessar o conteúdo do CD em ambiente Windows, será visível um diretório ou pasta, *Tensor3D*, contendo por sua vez as pastas *programa* e *auxiliares*. Para o correto funcionamento do aplicativo, é necessário realizar a instalação prévia do interpretador de comandos da linguagem Tcl/Tk e da biblioteca gráfica VTK. Estes módulos estão contidos na pasta *auxiliares* e sua instalação é ilustrada a seguir.

No diretório *auxiliares*, clicar duas vezes sobre o arquivo *tcl832.exe* usando o botão esquerdo do mouse. Selecionar a opção *Next* em todas as janelas que o instalador do Tcl/Tk visualizará em seguida. Aguardar a finalização da instalação e reiniciar o computador.

De forma análoga, para instalar os binários da biblioteca VTK, acessar novamente o conteúdo do CD e, sempre no diretório *auxiliares*, clicar duas vezes sobre o arquivo *vtk42-LatestRelease.exe*. Selecionar a opção *Next* em todas as janelas exibidas. A biblioteca VTK será instalada de forma automática, sendo necessário reiniciar o computador mais uma vez.

Finalmente, acessar a pasta *Tensor3D* do CD e copiar o conteúdo do diretório *programa* em local adequado no disco do computador. Após a cópia, acessar o diretório *programa*. Clicar duas vezes sobre o arquivo *tensor.tcl*. Será exibida uma mensagem de erro, “*Can’t find package vtk*”, devido à necessidade de associar arquivos com extensão *.tcl, isto é programas na linguagem Tcl/Tk, ao interpretador de comandos da biblioteca VTK. Para tanto, clicar com o botão direito do mouse sobre o arquivo *tensor.tcl*, mantendo pressionada a tecla SHIFT do teclado. Será exibido um menu do tipo *pop-up*, com algumas opções. Dentre estas, selecionar “*Abrir com...*”. Será mostrada uma janela para seleção do aplicativo a ser invocado quando da execução do script *tensor.tcl*. Selecionar o botão *Outros* em baixo à direita. Na nova janela exibida, selecionar o caminho *C:\Arquivos de programa\vtk42\bin* e, dentro deste, o arquivo *vtk.exe*, que é o interpretador necessário, e clicar no botão *Abrir*. Na janela

remanescente, selecionar a opção “*Usar sempre este programa para abrir este tipo de arquivo*” e clicar no botão *OK*.

O interpretador de comandos da biblioteca VTK será invocado e executará o programa *Tensor3D*. A execução posterior do programa poderá ser ativada clicando duas vezes sobre o arquivo *tensor.tcl* com o botão esquerdo do mouse.

A associação de *scripts* com extensão *.tcl ao interpretador do VTK não compromete a execução de programas em Tcl/Tk que não utilizem esta biblioteca, já que o VTK redireciona automaticamente instruções da linguagem Tcl/Tk para o interpretador correspondente. Aliás, o próprio programa *Tensor3D* utiliza comandos e funções de ambos os interpretadores (VTK e Tcl/Tk).

O diretório *programa* do *Tensor3D* possui os seguintes arquivos e subdiretórios:

- *tensor.tcl*. É o programa principal
- *la.tcl*. Biblioteca de funções algébricas utilizada pelo programa para calcular os autovalores e autovetores da matriz 3x3 do tensor matemático (HUME LINEAR ALGEBRA TCL PACKAGE, 2005).
- *objfact.tcl*. Trata-se do programa auxiliar *Object Factory 3D* para a criação de modelos geométricos complexos a partir de entidades mais simples. Suas funcionalidades serão ilustradas na próxima seção.
- *imgs*. Diretório que contém os arquivos de imagens associadas a alguns botões da interface gráfica do programa.
- *objs*. Este diretório contém os modelos geométricos que podem ser lidos pelo programa. Os modelos, no formato *.vtk (seção 3.4.4), estão organizados em dois subdiretórios: *main*, que contém os objetos mais importantes ou que resultam da composição de objetos mais simples; e *aus*, que contém tais objetos menores, bem como modelos utilizados nas primeiras etapas de testes de deformação no ambiente do *Tensor3D*.
- *aus*. Este diretório contém alguns programas (*scripts*) no formato *.tcl que foram usados para testar separadamente funcionalidades implementadas no programa *Tensor3D*. Outros *scripts* realizam a modelagem geométrica de alguns dos objetos contidos no diretório *objs*. Tal modelagem envolve a criação de objetos gráficos simples disponíveis no VTK (planos, esferas, discos, etc), bem como sua rotação, translação, redimensionamento e junção de forma a gerar conjuntos de objetos didaticamente adequados para a deformação. O estudo dos programas deste diretório fornece ao iniciante informações importantes sobre as principais ferramentas de modelagem implementadas na biblioteca VTK.

As instruções de instalação desta seção visaram deixar o usuário em condições de utilizar o *Tensor3D* no sistema operacional Windows. Neste ambiente é possível instalar o programa e explorar suas funcionalidades de forma rápida. A instalação no sistema operacional Linux é mais complexa, pois envolve a configuração e compilação da biblioteca gráfica VTK, procedimento que foi descrito na seção 3.4.5.

Uma consideração final diz respeito à versão do VTK utilizada para executar o programa *Tensor3D*. Durante as fases de desenvolvimento e testes do aplicativo, a versão adotada foi a 4.2. Trata-se da versão *estável* mais atual até a data de entrega desta dissertação. Contudo, novas versões estão prestes a serem disponibilizadas na *homepage* da biblioteca (VISUALIZATION TOOLKIT, 2005). Em particular, está previsto um conjunto de importantes modificações na iminente versão 5.0, para adoção da qual foi criada uma versão 4.4 de transição. Dentre as modificações anunciadas, destaca-se a eliminação da classe *vtkPolyData* e sua junção com a classe *vtkUnstructuredGrid* para unificar e otimizar a representação interna de informações geométricas e topológicas de modelos que usam primitivos gráficos 1D e 2D (pontos, linhas e polígonos, classe *vtkPolyData*) e 3D (tetraedros, prismas, etc., classe *vtkUnstructuredGrid*). Diante disto, é muito provável que o programa *Tensor3D* apresente malfuncionamentos ao se tentar executá-lo com a versão 5.0 do VTK. A solução imediata para o problema consiste na utilização da versão da biblioteca para a qual o programa foi escrito, isto é, 4.2. Para tanto os binários pré-compilados do VTK para o sistema operacional Windows estão contidos no CD que acompanha esta dissertação.

A escolha da versão do interpretador de comandos da linguagem Tcl/Tk é menos importante, já que o nível de compatibilidade entre versões diferentes da linguagem, que o programa *Tensor3D* requer, é garantido pelos desenvolvedores da mesma. O usuário pode instalar a versão 8.3.2 do Tcl/Tk contida no CD.

5.9.5 Programas auxiliares: *Object Factory 3D*

Numerosos modelos disponíveis para deformação no programa *Tensor3D* foram desenvolvidos usando programas auxiliares, *scripts* em Tcl/Tk. Estes programas estão contidos no subdiretório *aus* do CD do programa *Tensor3D*. A deformação de alguns destes modelos será ilustrada na próxima seção. Os programas do diretório *aus* utilizam funcionalidades de modelagem geométrica da biblioteca VTK, permitindo criar objetos compostos a partir de entidades mais simples.

O mais importante de tais *scripts* auxiliares, chamado *Object Factory 3D* permite criar modelos que simulam agregados mono- e poliminerálicos cuja orientação preferencial e escala podem ser controladas. A finalidade deste programa é modelar agregados minerais de diferentes formas (prismáticas, placóides) e orientações, bem como graus de isotropia iniciais.

O aplicativo surgiu como programa auxiliar para a criação de entidades complexas para o *Tensor3D*, mas as funcionalidades implementadas tornaram-no uma ferramenta de modelagem geométrica à parte. Os modelos são exibidos em 3D e podem ser gravados no formato *.vtk (seção 3.4.4), o que permite sua leitura e deformação no programa *Tensor3D*.

O programa *Object Factory 3D* está presente no mesmo diretório que o *Tensor3D* e corresponde ao *script objfact.tcl*. Seu nome visa justamente realçar as potencialidades de criar diferentes modelos geométricos. O programa representa mais um exemplo de integração das componentes gráficas da linguagem Tcl/Tk (botões, barras de rolagem, etc.) e dos módulos de visualização científica da biblioteca VTK. Tal integração requer um esforço de programação mínimo e permite a criação rápida de aplicativos interativos de modelagem geométrica. A Figura 5.31 mostra a janela de execução do programa.

A porção direita da figura mostra um conjunto de componentes gráficos (da linguagem Tcl/Tk) utilizados para modificar as propriedades dos objetos modelados. Já a porção esquerda consiste na visualização 3D dos objetos realizada pela biblioteca VTK. Segue uma breve descrição das funcionalidades do programa.

O objeto modelado é uma malha regular de entidades mais simples posicionadas ao longo de linhas paralelas aos eixos coordenados. A resolução da malha, isto é, o número de objetos ao longo de cada direção coordenada pode ser definido na seção *Mesh Size*. Desta forma, é possível, por exemplo, criar malhas de 4x4x4 (=64), 5x5x5 (=125) objetos e assim por diante. A resolução pode assumir valores de 1 a 7.

Os objetos fundamentais que podem ser selecionados são cones (pirâmides cuja base é um polígono regular com 20 lados) ou cilindros orientados na direção do eixo coordenado X, cubos, esferas, ou uma composição aleatória dos mesmos. Tal seleção é realizada na seção

Object, sendo que o primeiro botão *Random* à esquerda permite ativar a composição aleatória de objetos. O tamanho destes pode ser definido usando a barra de rolagem *Section*. O botão *Random* à direita atribui valores aleatórios ao tamanho dos objetos modelados.

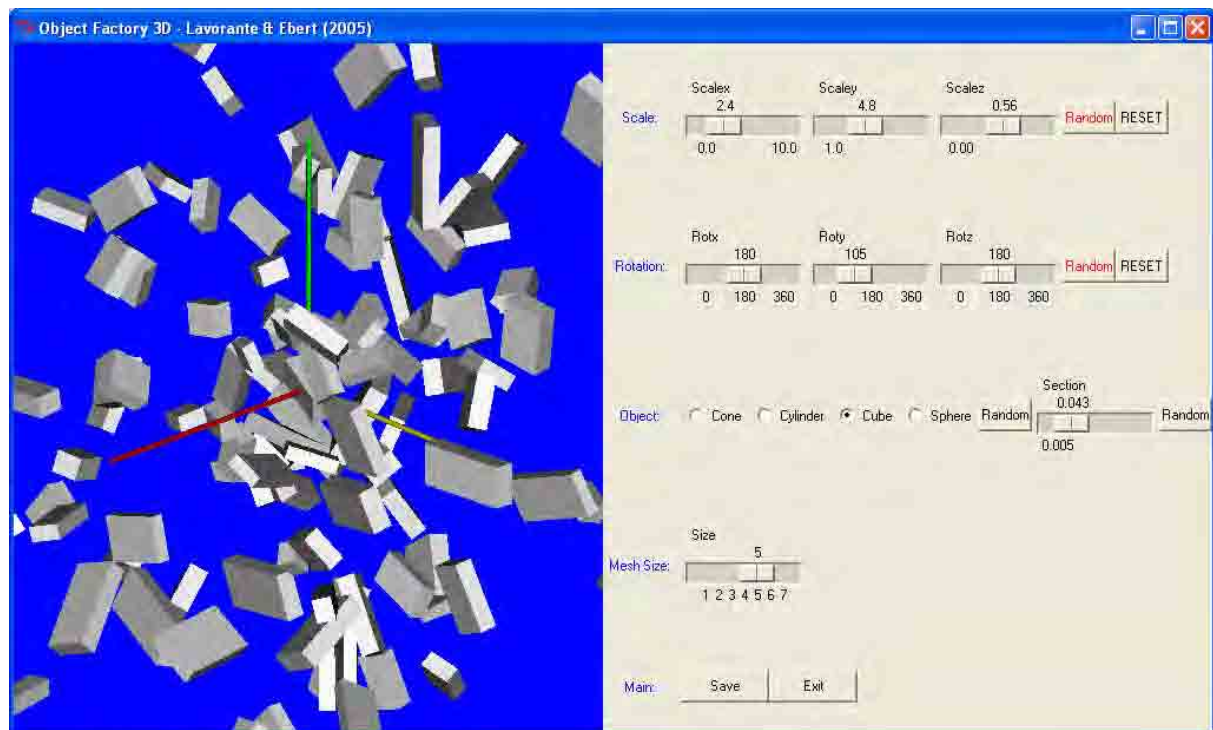


Figura 5.31 – Exemplo de utilização do programa *Object Factory 3D*. O objeto original, uma malha 5x5x5 de cubos alinhados ao longo dos eixos coordenados, foi modelado gerando um agregado de paralelepípedos de orientação aleatória. Para tanto, os cubos iniciais foram redimensionados e rotacionados em torno dos eixos coordenados, usando valores aleatórios para os fatores de escala e ângulos (botões *Random* ativados em vermelho, à direita).

A seção *Scale* controla o fator de escala dos objetos posicionados na malha 3D. A escala pode ser modificada ao longo de cada eixo coordenado *local* ao objeto considerado. Os sistemas de coordenadas local e global possuem a mesma orientação na ausência de rotação do objeto (próximo parágrafo). Os fatores de escala correspondentes ao objeto indeformado são (1, 1, 1), sendo que valores menores que 1 indicam encurtamento ao longo dos eixos X, Y ou Z, ao passo que valores maiores que 1 indicam estiramento. O botão *Random* ativa a utilização de fatores de escala aleatórios para os objetos. O botão *RESET* inicializa os fatores com os valores (1, 1, 1). A Figura 5.32 mostra a modificação de escala para uma malha regular de 5x5x5=125 cubos. O resultado é a modelagem de paralelepípedos.

A seção *Rotation* permite rotacionar cada objeto em torno dos eixos coordenados locais ao mesmo. Os ângulos são expressos em graus, sendo possível especificar separadamente suas medidas para cada eixo coordenado. O botão *Random* define valores aleatórios para os ângulos, sendo que os valores variam entre 0 e a medida especificada pelo

usuário. O resultado é uma orientação diferente para cada objeto da malha, sendo possível controlar o grau de isotropia do conjunto. O botão *RESET* inicializa os ângulos de rotação com o valor 0 (rotação nula). A Figura 5.33 mostra um exemplo de rotação de uma malha regular de 125 cilindros inicialmente alinhados com o eixo X. É também ilustrada a ativação da modalidade de orientação aleatória.

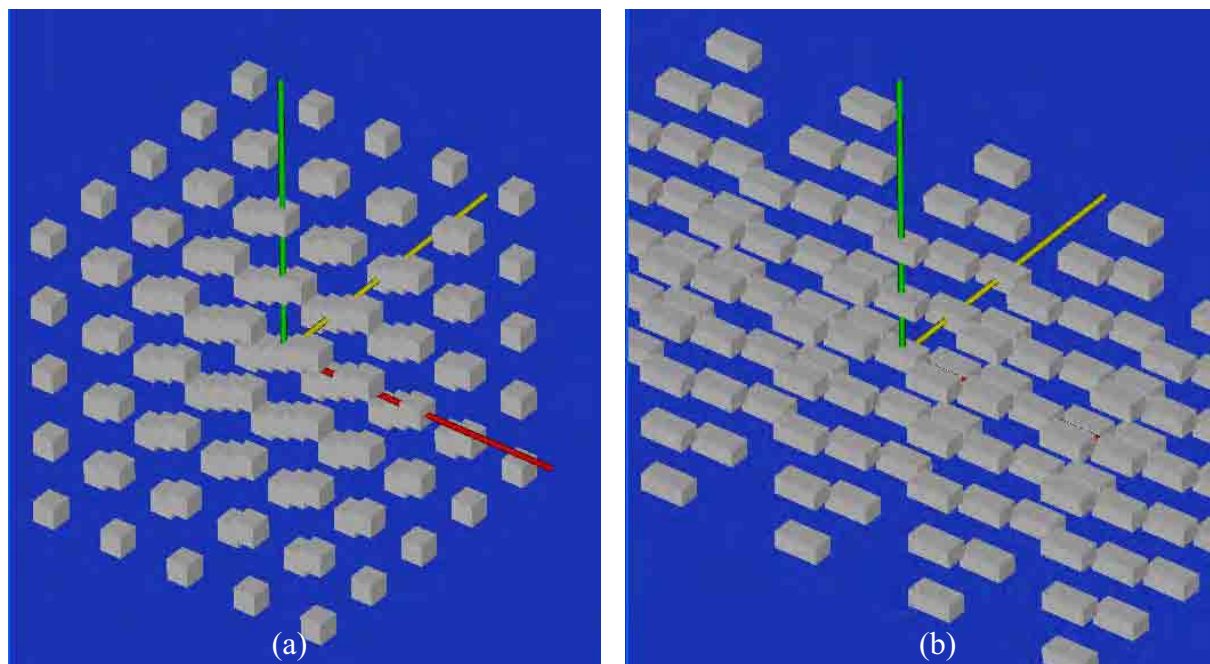


Figura 5.32 – Modificação do fator de escala para uma malha regular de $5 \times 5 \times 5 = 125$ cubos. (a) Configuração inicial dos objetos. As cores dos eixos coordenados globais são vermelho (X), amarelo (Y) e verde (Z). (b) modificação do fator de escala de (1, 1, 1) para (2, 1, 0.8), de forma que os cubos foram estirados na direção X e encurtados na direção Z.

As funcionalidades de modificação de escala e rotação dos objetos podem ser combinadas para construir modelos de agregados minerais estudados em geologia. Um exemplo é a Figura 5.31, que mostra 125 cubos deformados (por mudança do fator de escala) e rotacionados usando parâmetros aleatórios. A simulação pode ser ainda mais realista combinando os vários objetos disponíveis e especificando um tamanho aleatório para cada elemento (Figura 5.34).

Uma vez criado o agregado de objetos ideal para as finalidades do usuário, o mesmo pode ser armazenado em disco no formato *.vtk. Para tanto, é suficiente usar o botão *SAVE* da seção *Main* do programa. Os objetos criados podem assim ser lidos e deformados no âmbito do programa *Tensor3D*. Alguns dos objetos descritos na próxima seção foram modelados desta forma.

O programa *Object Factory 3D* mostrou ser uma ferramenta extremamente poderosa e flexível. Suas capacidades de modelagem podem ser expandidas através da inserção de novos objetos ou de regras de mudança de escala / rotação pré-definidas, visando construir modelos mais fieis aos agregados minerais estudados na prática.

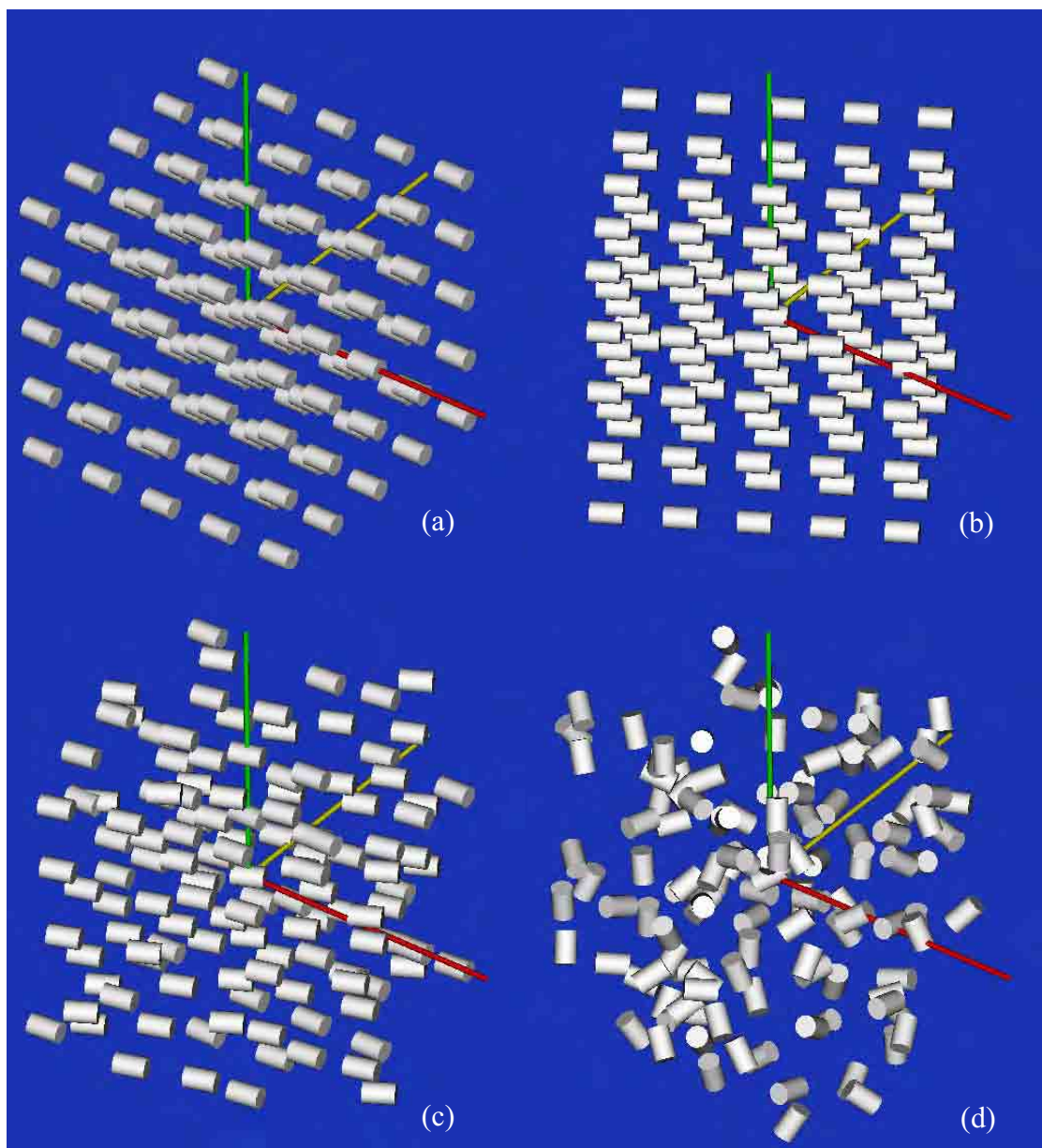


Figura 5.33 – Definição da orientação de 125 cilindros posicionados em uma malha regular. (a) Os objetos são inicialmente paralelos ao eixo X (em vermelho). (b) Especificação de uma rotação de 30 graus em torno do eixo verde Z (parâmetro *Rotz* da janela do programa). Todos os cilindros sofrem a mesma rotação. (c) Ativação da modalidade *Random*. Cada cilindro é rotacionado usando um ângulo aleatório entre 0 e 30 graus. Apesar das diferentes orientações, os objetos ainda mostram uma direção preferencial (cerca de 15 graus), devido ao baixo ângulo de rotação. (d) Utilização da modalidade *Random* com o valor de 180 graus especificado para os três parâmetros de rotação. Os cilindros formam um objeto composto de aparência isotrópica, mascarando a natureza regular da malha original.

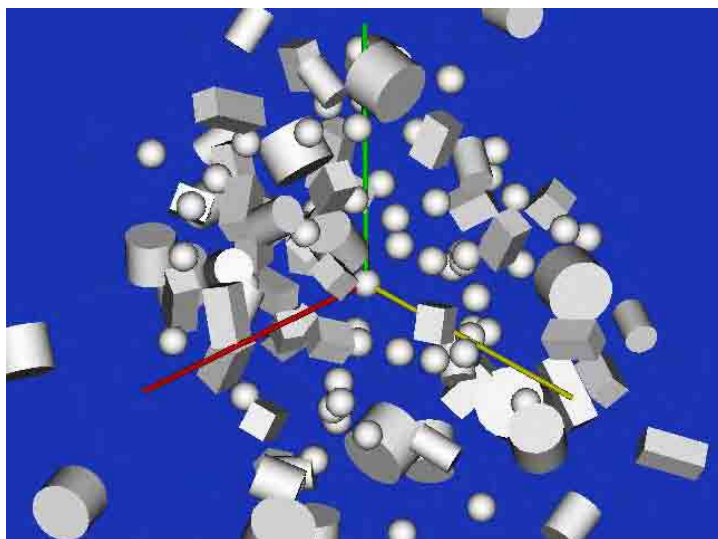


Figura 5.34 – Conjunto de cubos, esferas e cilindros com orientação, fator de escala e tamanho aleatórios (a escala das esferas não foi modificada).

5.9.6 Algumas aplicações

Nesta seção serão mostrados exemplos de deformação progressiva de objetos que modelam estruturas geológicas ou de marcadores cuja análise auxilia no entendimento e quantificação dos processos deformacionais. Os objetos usados foram criados através da biblioteca VTK ou importados de outros ambientes de modelagem. A deformação foi realizada com o programa *Tensor3D*, modificando as componentes da matriz 3x3 do tensor.

Agregado poliminerálico. Este objeto, cuja deformação é mostrada na Figura 5.35, foi criado usando o programa *Object Factory 3D*. O modelo consiste de um conjunto de esferas, paralelepípedos (originalmente cubos) e cilindros de orientações, escalas e tamanhos aleatórios. O arquivo de dados é *objs/main/agregado_poli_iso3.vtk* (de acordo com a estrutura de diretórios ilustrada na seção 5.9.4). A entidade da deformação é destacada através da visualização do elipsóide de deformação e da caixa de limites (*Bounding Box*) do modelo. A orientação constante do elipsóide e do *Bounding Box* ao longo do eixo X mostra que não ocorre cisalhamento simples, isto é, a deformação é coaxial. Contudo, observa-se a rotação de marcadores lineares (cilindros) internos ao objeto durante o processo.

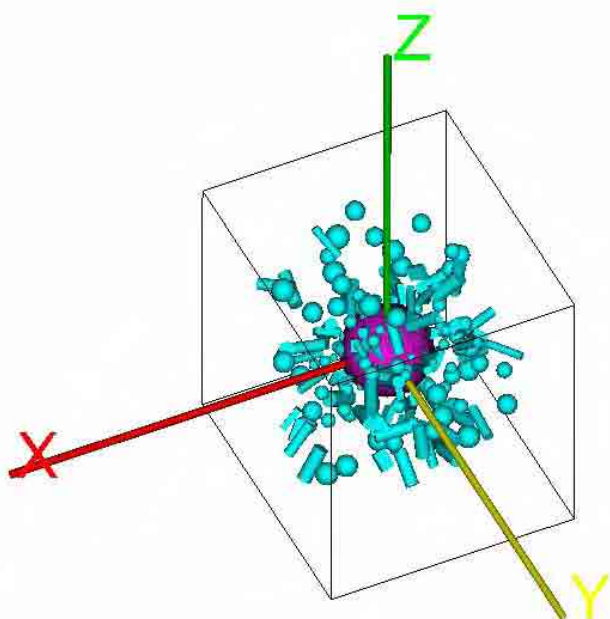
Simulação de boudinagem. Neste exemplo, foi realizado o cisalhamento puro de uma malha regular de $5 \times 5 \times 5 = 125$ cubos construída com o programa *Object Factory 3D* (arquivo *objs/aus/cubes.vtk*). O tamanho dos cubos foi especificado de forma a minimizar o espaço entre os mesmos, para criar um modelo cuja deformação coaxial pudesse fornecer um exemplo de boudinagem de um corpo geológico. O resultado é mostrado nas Figuras 5.36 e

5.37. Estas exibem a janela inteira do programa *Tensor3D* para destacar as componentes do tensor matemático que foram modificadas (elipses vermelhas). No estado indeformado, as componentes da diagonal principal do tensor possuem valor unitário. Tais componentes são modificadas ao longo do processo: aumento na direção dos eixos coordenados X e Y, diminuição na direção do eixo Z. Durante a deformação, o espaçamento entre os paralelepípedos aumenta devido à modificação do fator de escala.

Conchas fósseis. Trata-se de uma composição de três modelos digitais da concha da classe *Cephalopoda*, subclasse *Nautiloidea*, já deformada na Figura 5.18. A concha foi construída usando uma modelagem matemática. O modelo de três conchas foi criado usando o *script seashell3.tcl* contido no diretório *aus* de programas auxiliares do *Tensor3D*. O *script* lê a informação geométrica e topológica da concha através da classe *vtkPolyDataReader* (arquivo *obj/aus/seashell.vtk*). Este objeto está contido no plano coordenado XZ (Figura 5.18). Em seguida são criados três pares de instâncias *vtkTransform* / *vtkTransformPolyDataFilter*, os quais realizam a translação da primeira concha no plano XZ, o posicionamento da segunda no plano YZ através de uma rotação do primeiro objeto em torno do eixo Z (-90 graus); e o posicionamento da terceira concha no plano XY através de uma rotação da primeira em torno do eixo X (-90 graus). As conchas rotacionadas são também transladadas visando o afastamento de seu centro da origem do sistema de eixos coordenados. Por fim, os três modelos são compostos em um único objeto do tipo *vtkPolyData* usando a classe *vtkAppendPolyData*. O modelo é armazenado em disco no formato do VTK (arquivo *objs/main/3seashells.vtk*), através da classe *vtkPolyDataWriter*.

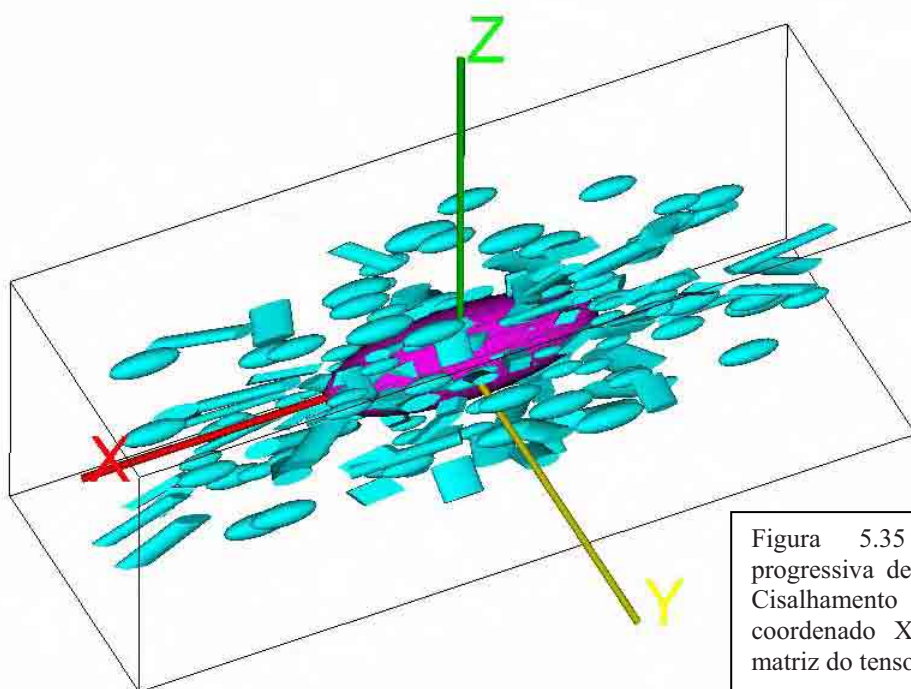
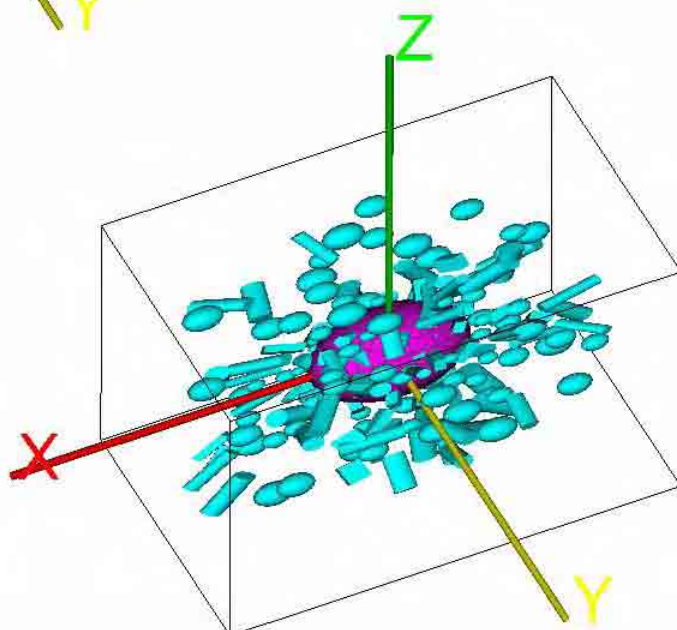
A Figura 5.38 mostra uma deformação coaxial progressiva do objeto composto. Cada concha está contida em um plano coordenado para estudar a deformação ao longo dos três planos cinemáticos. A figura mostra os planos coordenados semitransparentes, bem como o elipsóide de deformação (na origem do sistema). São também visíveis os eixos principais de deformação. Estes se mantêm paralelos aos eixos coordenados, o que confirma o caráter coaxial da deformação.

A Figura 5.39 mostra um exemplo de deformação não coaxial progressiva do mesmo conjunto de conchas. Já na Figura 5.40 foi realizada a deformação coaxial de um agregado de 27 conchas com posições e orientações aleatórias. Tal agregado, armazenado no arquivo *objs/aus/seashells.vtk*, foi criado utilizando uma versão modificada do programa *Object Factory 3D*. Neste, uma malha regular de $3 \times 3 \times 3 = 27$ objetos sofreu rotações com ângulos aleatórios para simular a orientação anisotrópica dos fósseis.



$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



$$T = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figura 5.35 – Deformação coaxial progressiva de um agregado poliminerálico. Cisalhamento puro na direção do eixo coordenado X. É também representada a matriz do tensor correspondente.

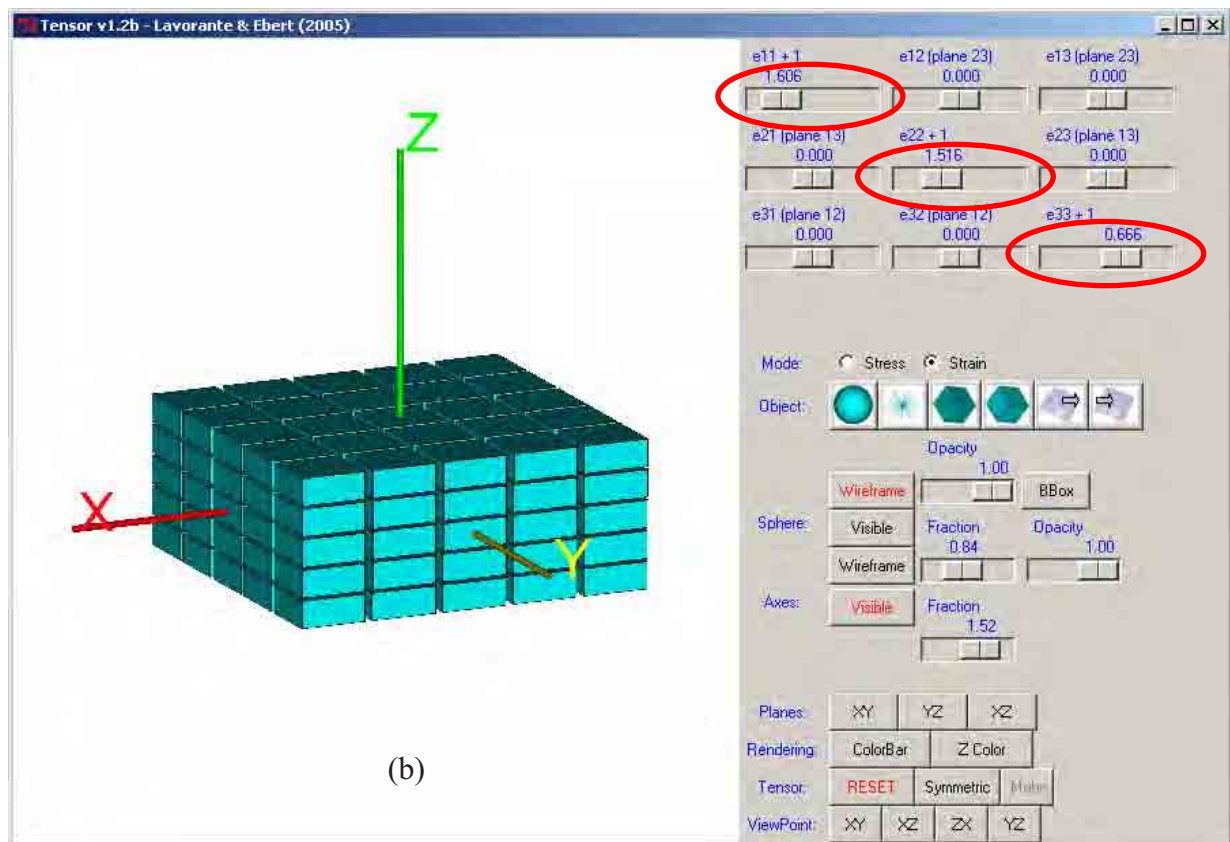
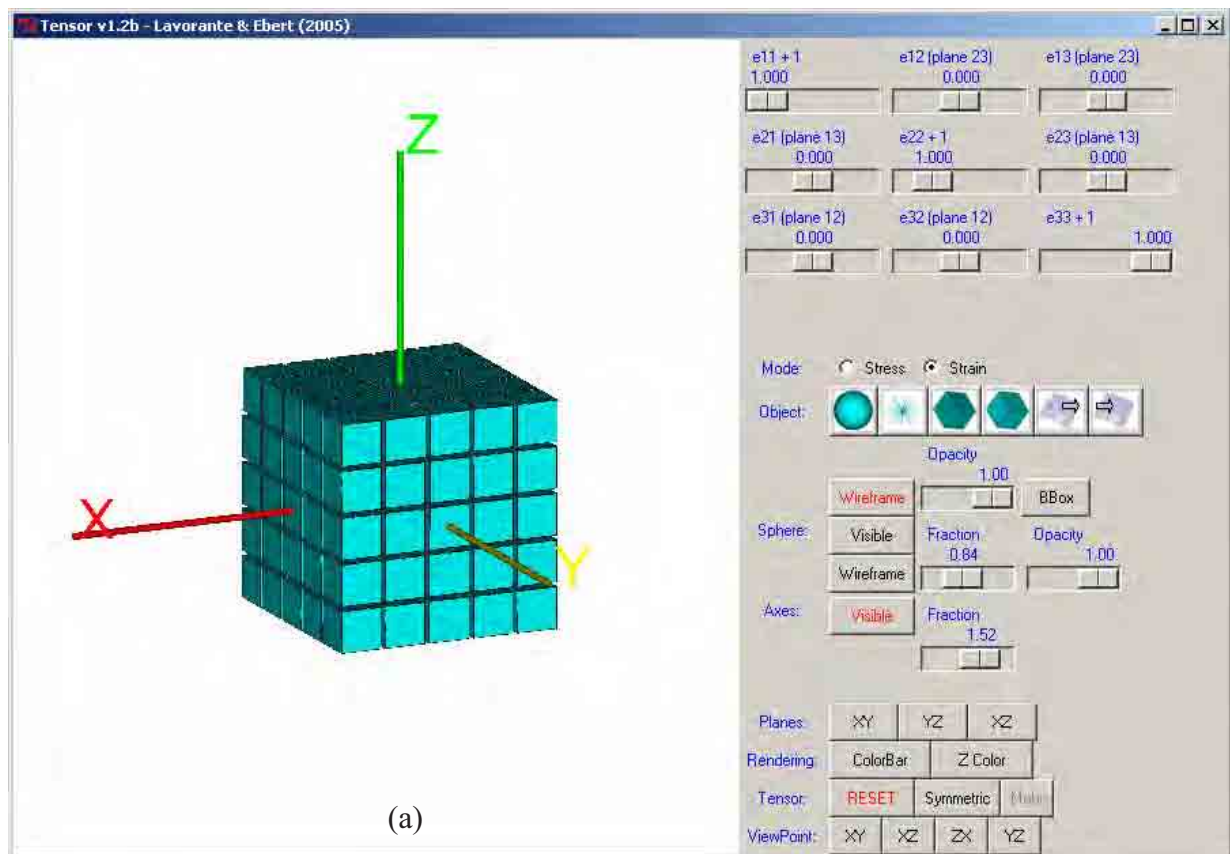


Figura 5.36 – Deformação coaxial progressiva de um agregado de cubos para simular boudinagem. Cisalhamento puro ao longo dos eixos coordenados X, Y e Z por valores, respectivamente, de (1, 1, 1) (a) e (1.6, 1.5, 0.67) (b). As componentes do tensor modificadas são destacadas em vermelho.

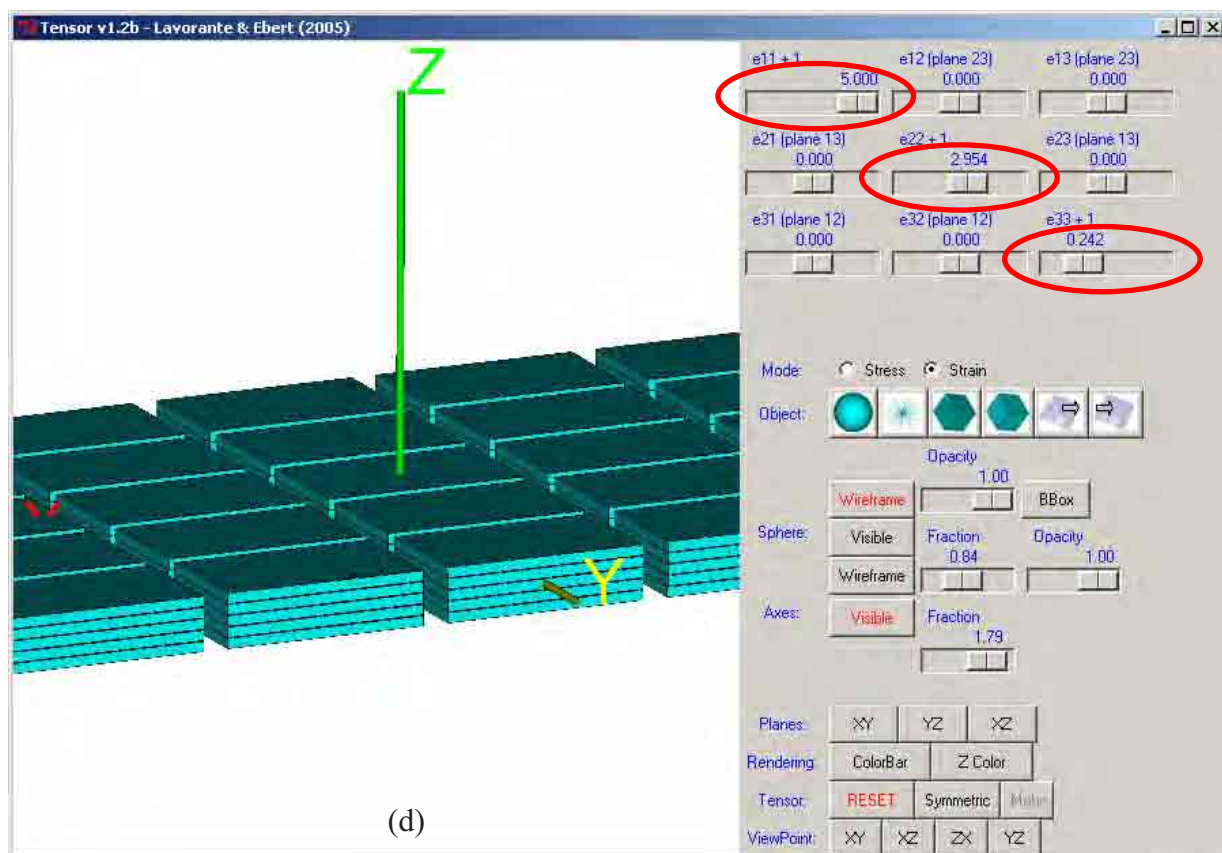
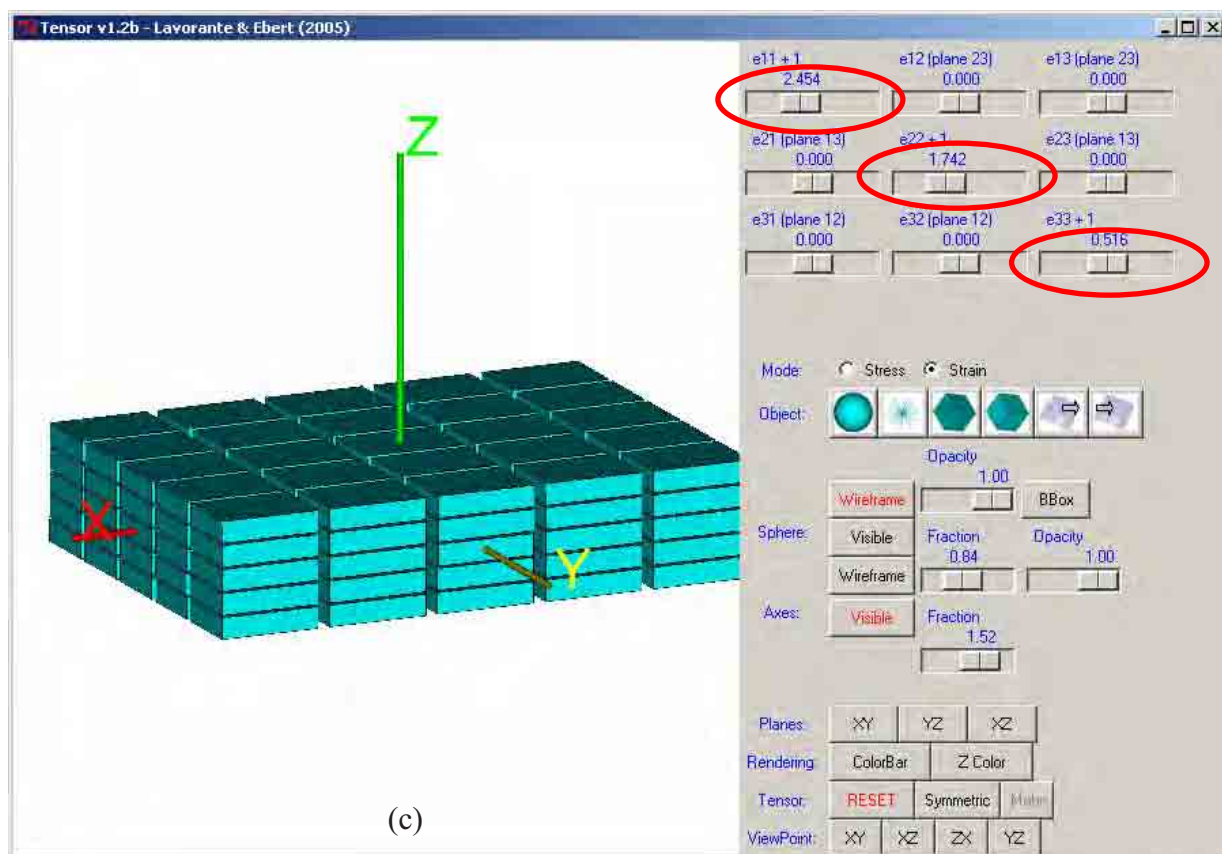
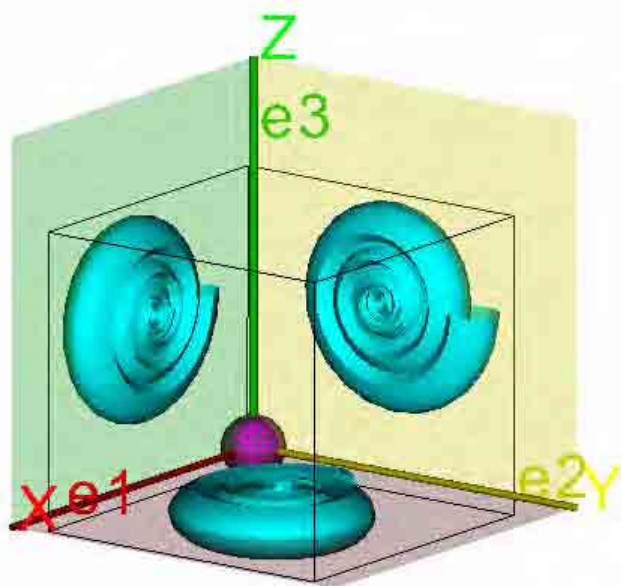
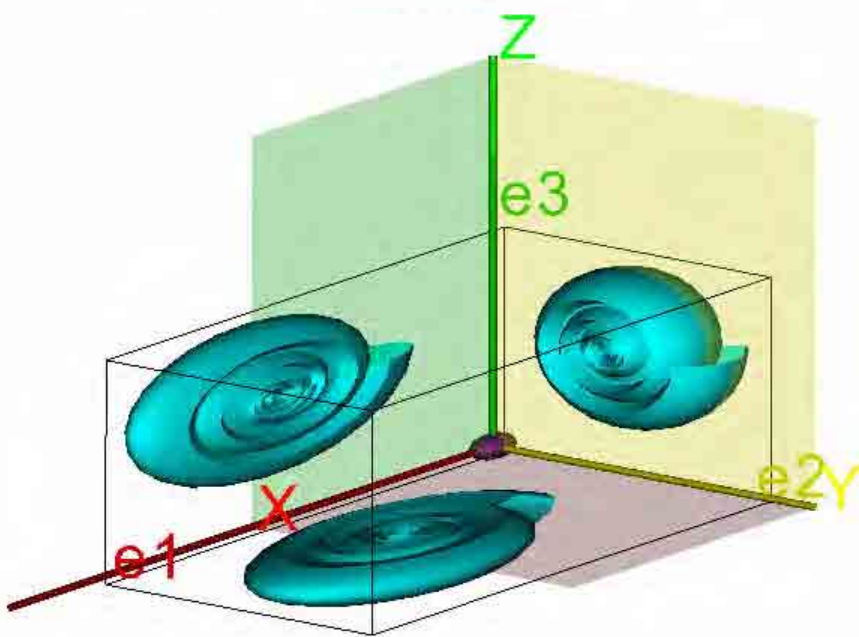


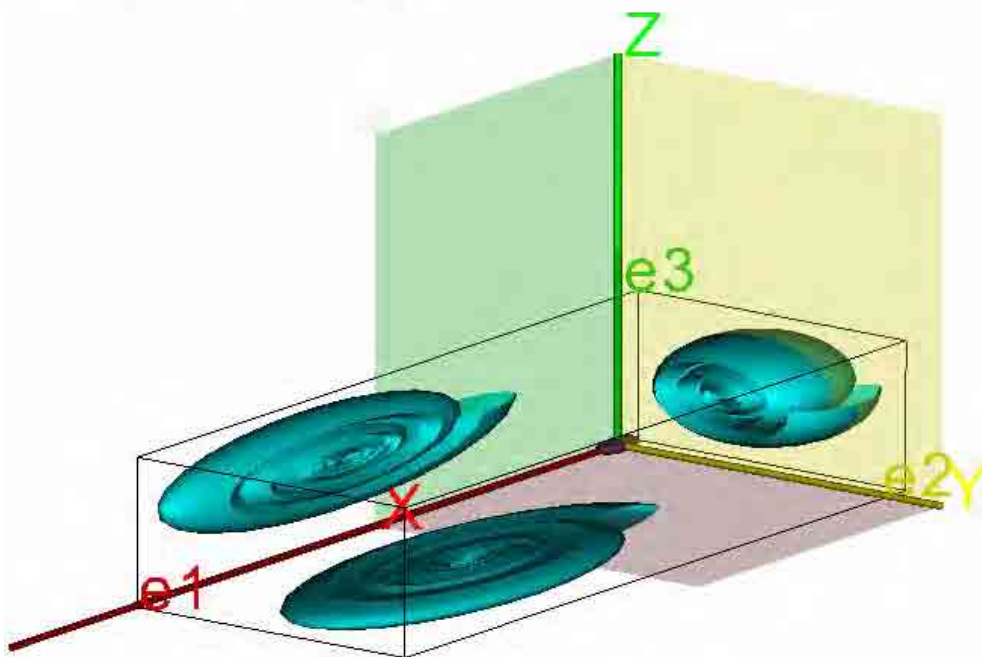
Figura 5.37 – Continuação da deformação coaxial progressiva do objeto da figura anterior para os seguintes valores de deformação longitudinal a X, Y e Z: (2.4, 1.7, 0.5) (c) e (5, 2.9, 0.2) (d).



$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

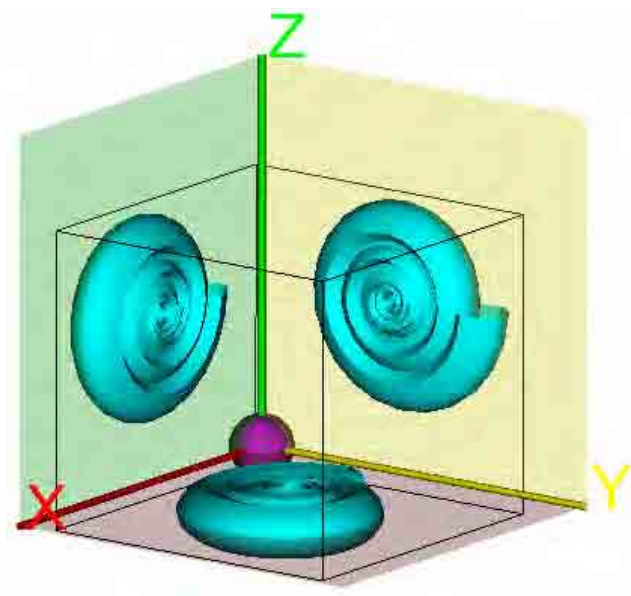


$$T = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.75 \end{bmatrix}$$

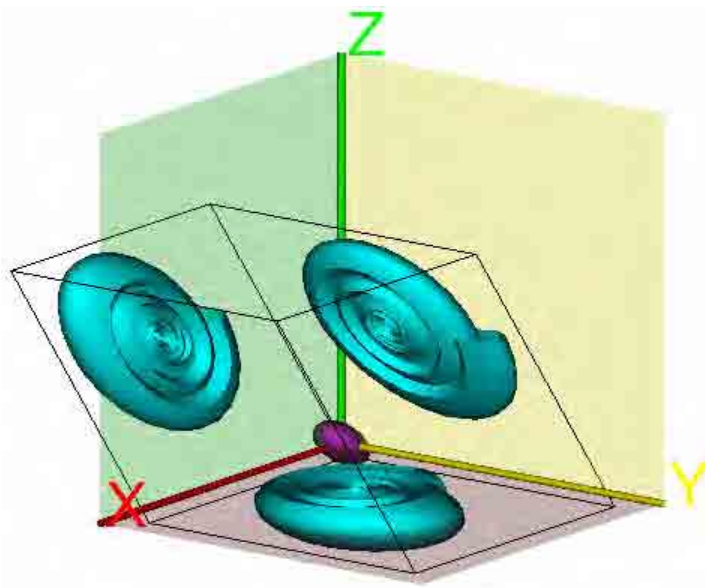


$$T = \begin{bmatrix} 2.5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}$$

Figura 5.38 – Deformação coaxial progressiva de três conchas (classe *Cephalopoda*, subclasse *Nautiloidea*) alinhadas com os planos coordenados semitransparentes. São também visíveis o elipsóide de deformação e o *Bounding Box* do conjunto.



$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.75 & 0 & 1 \end{bmatrix}$$

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1.75 & 0 & 1 \end{bmatrix}$$

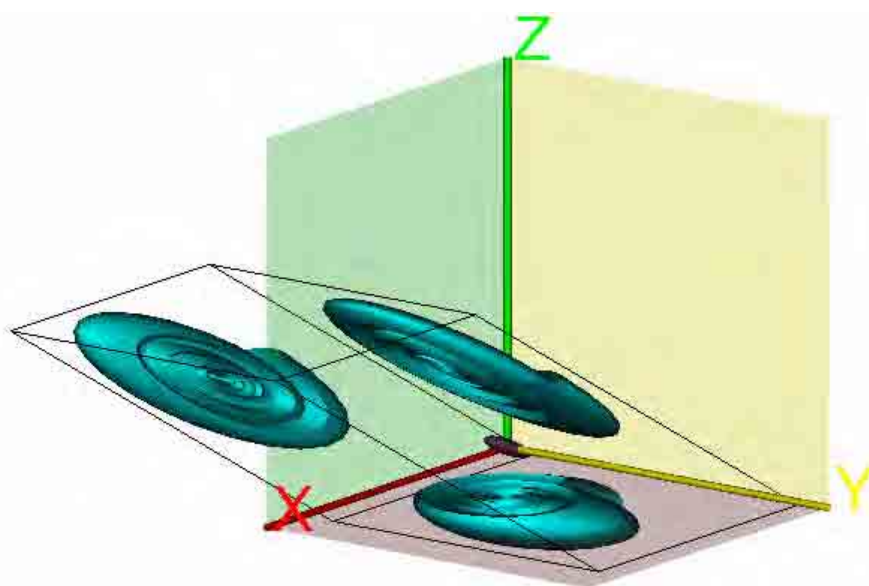


Figura 5.39 – Deformação não coaxial progressiva de três conchas. Cisalhamento simples no plano XY e na direção do eixo X (componente e_{31} do tensor)

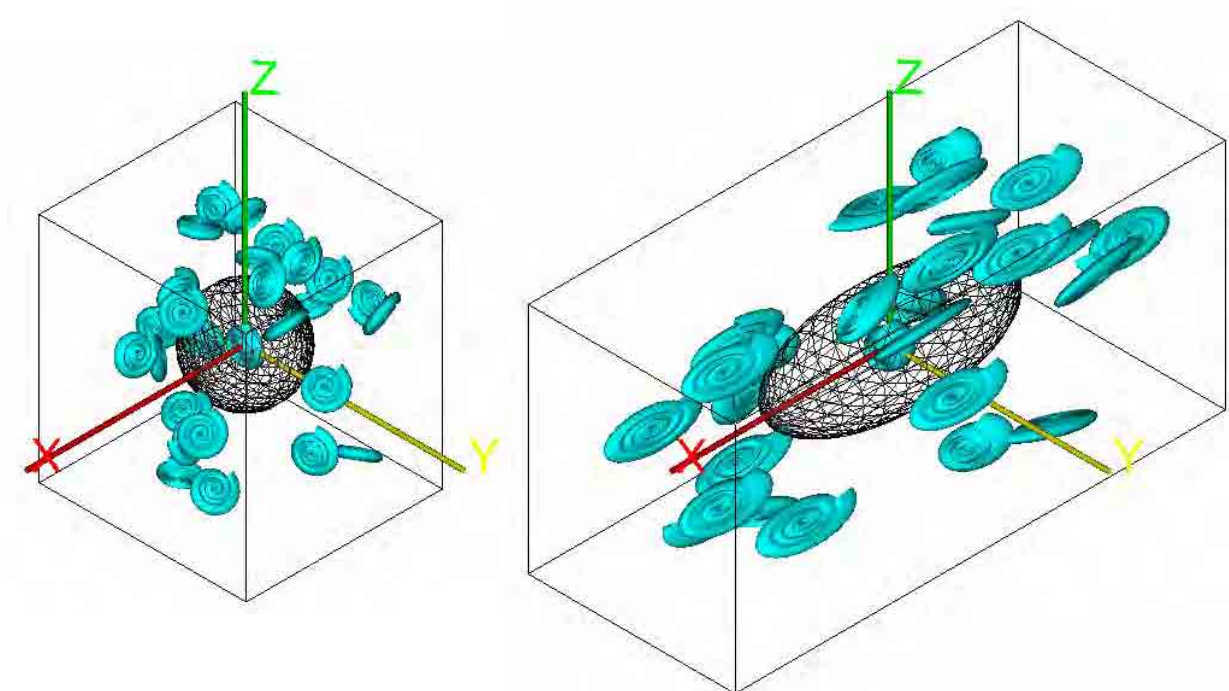


Figura 5.40 – Deformação coaxial de um agregado de 27 conchas *Cephalopoda* (*Nautiloidea*) com posições e orientações aleatórias. Estiramento na direção X por um fator de 2.5 (componente $e_{11}+1$ do tensor). Apesar da deformação ser coaxial, nota-se a rotação de algumas conchas ao final do processo. É também visualizado o elipsóide de deformação (centro do modelo).

Modelo de dobra. A deformação não coaxial deste modelo foi mostrada na Figura 5.21. Trata-se da modelagem de um corpo dobrado obtida matematicamente (Figura 5.41), usando a biblioteca VTK. O primeiro passo consistiu na criação da seção da superfície dobrada no plano XZ. Tal curva foi obtida traçando o gráfico da função $z = \sin(x * 2\pi)$, sendo que x varia de 0 a 3 (três ciclos completos da função seno). O gráfico foi construído como *Polyline* (conjunto de segmentos interconectados), linha poligonal armazenada em uma classe *vtkPolyData* (Figura 5.41a). Em seguida, a representação do objeto foi passada para uma instância da classe *vtkLinearExtrusionFilter*, que realizou uma extrusão da curva na direção Y. Com isto foi criada uma superfície dobrada (Figura 5.41b). Finalmente, o corpo sólido em formato de dobra foi obtido passando o resultado da extrusão anterior para outra instância da classe *vtkLinearExtrusionFilter*, considerando como direção de extrusão o eixo vertical Z (Figura 5.41c).

O modelo de dobra foi gerado por dois *scripts* distintos, ambos contidos no diretório *aus* do CD do programa *Tensor*. O primeiro, *fold.tcl*, cria a superfície dobrada, armazenada no arquivo *objs/aus/fold.vtk*. O segundo *script*, *solidfold.tcl*, cria o modelo sólido do corpo dobrado, armazenado no arquivo *objs/aus/solidfold.vtk*.

O modelo de corpo dobrado criado constitui um exemplo interessante de modelagem matemática de estruturas geológicas usando a biblioteca VTK. Os parâmetros principais do

modelo (número de ciclos da função seno, espessuras lateral e vertical) podem ser facilmente modificadas nos *scripts fold.tcl* e *solidfold.tcl*, permitindo a criação de modelos diferentes. Além disto, a aparência regular de um objeto criado matematicamente poderia ser amenizada usando filtros de interpolação ou adicionando ruídos (translações de entidade aleatória) às coordenadas dos pontos constituintes.

Outro exemplo de deformação do modelo de dobra é mostrado na Figura 5.42. Neste caso, foi realizada uma deformação coaxial.

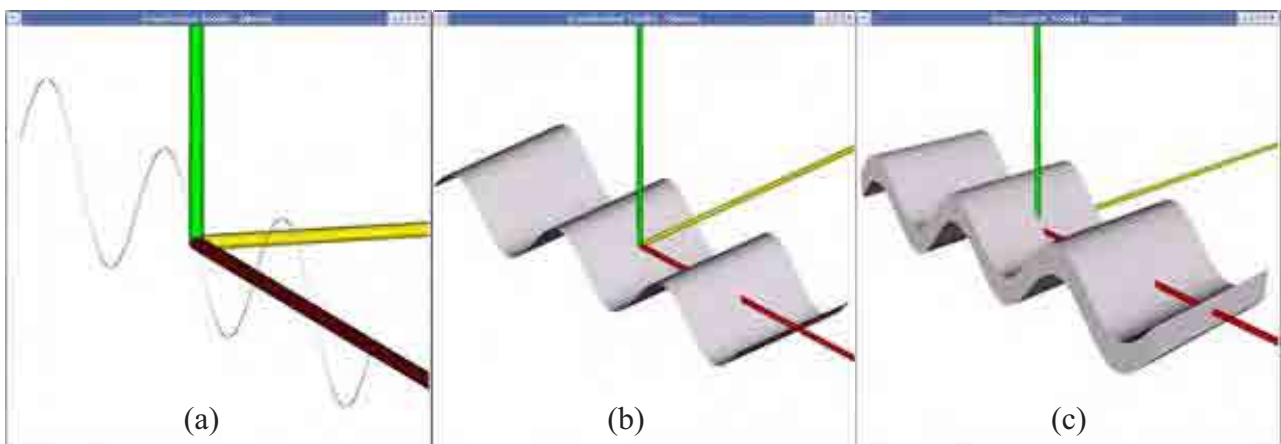


Figura 5.41 – Procedimento geométrico para a modelagem de um corpo dobrado no VTK. (a) Linha poligonal obtida unindo os pontos de três ciclos completos da função seno ao longo do eixo X (em vermelho). (b) Extrusão da linha na direção Y (eixo amarelo) para gerar uma superfície dobrada. (c) Nova extrusão na direção do eixo Z (em verde) para gerar um corpo sólido dobrado.

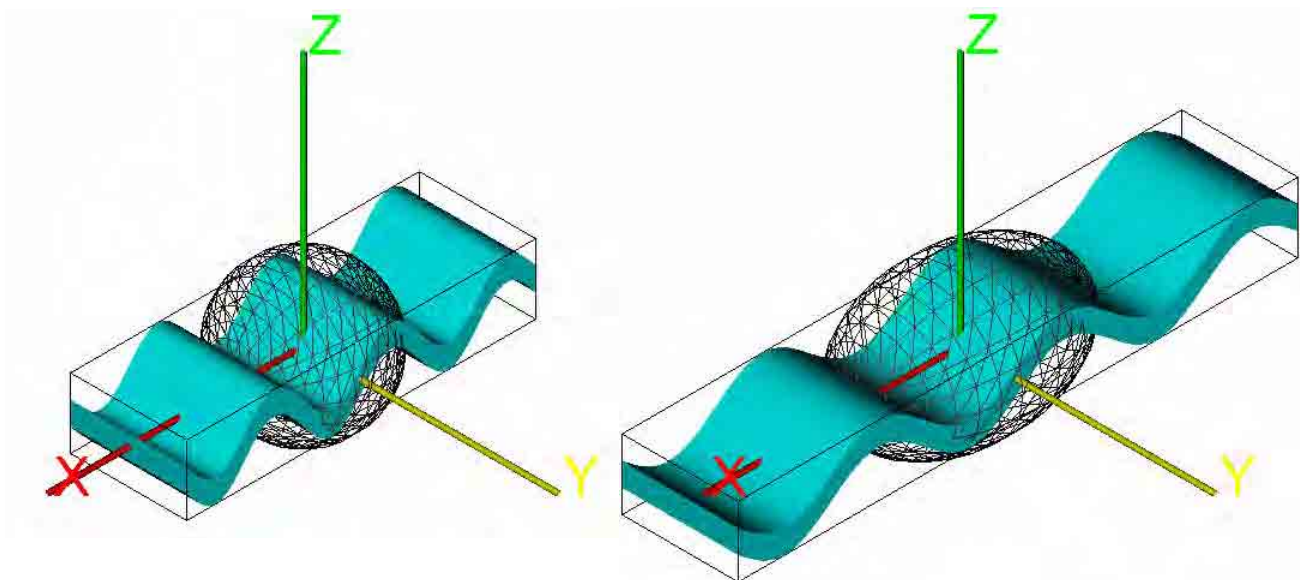


Figura 5.42 – Deformação coaxial de um modelo matemático de dobra. Estiramento na direção X por um fator de 1.6 (componente $e_{11}+1$ do tensor). No centro do sistema, o elipsóide de deformação.

Modelo de Estrela. Este modelo foi construído usando classes do VTK e é constituído de um conjunto de diâmetros de uma esfera, sendo que estes formam ângulos de 45 ou de 30 graus em cada plano coordenado. A Figura 5.14 contém um exemplo de deformação da estrela com ângulos de 45 graus. A finalidade do modelo em estrela é compor marcadores lineares cuja rotação em relação ao eixo de maior estiramento possa ser observada ao longo do processo deformacional.

O modelo de estrela com ângulos de 30 graus é gerado pelo *script aus/star30.tcl* e cria o arquivo *objs/aus/star30.vtk*. O procedimento de construção é o seguinte. Dentro do plano XY, é criado um conjunto de diâmetros com distância angular mútua de 30 graus. Para tanto, utiliza-se instâncias da classe *vtkLineSource*, rotacionando cada linha através de classes *vtkTransform* / *vtkTransformPolyDataFilter*. O objeto obtido está contido no plano XY. Em seguida, são criadas múltiplas instâncias do mesmo, cada qual rotacionada, em relação à anterior, de 30 graus considerando X como eixo de rotação. A Figura 5.43a mostra duas instâncias (planos) formando um ângulo de 90 graus em relação ao eixo X. Todas as instâncias de linhas coplanares rotacionadas são passadas como entrada para a classe *vtkAppendPolyData*, que gera uma única representação poligonal *vtkPolyData*. Sua saída é fornecida para a classe *vtkTubeFilter*, a qual cria a representação de prismas (ou “tubos”) em torno das linhas. Desta forma, as linhas se tornam mais visíveis no processo de visualização 3D. Por fim, o arquivo de dados correspondente é criado usando a classe *vtkPolyDataWriter*. O resultado final é a estrela da Figura 5.43b.

As Figuras 5.44 e 5.45 mostram a deformação não coaxial progressiva do modelo de estrela. A janela de comandos do programa *Tensor3D* é mostrada para destacar as componentes de deformação cisalhante modificadas.

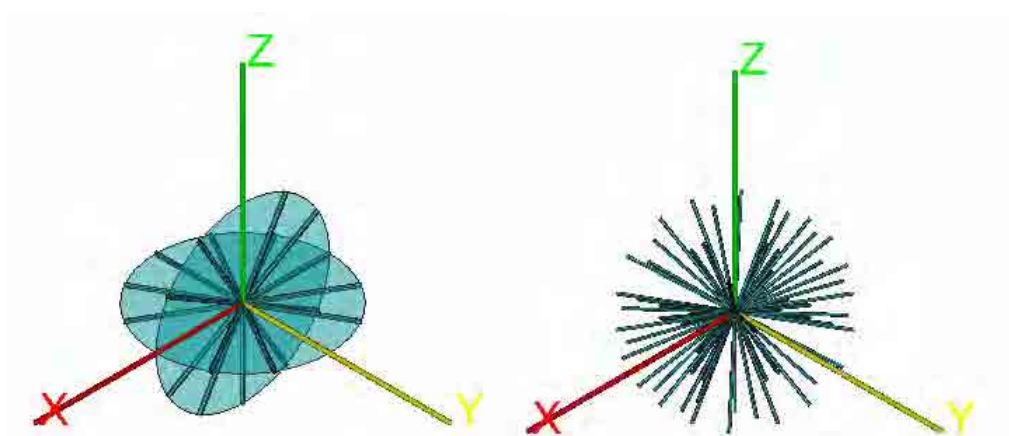


Figura 5.43 – Construção do modelo de uma “estrela” cujas linhas são diâmetros de uma esfera. O ângulo entre os diâmetros é de 30 graus. (a) Exemplo de dois conjuntos de linhas coplanares. (b) Modelo final da estrela.

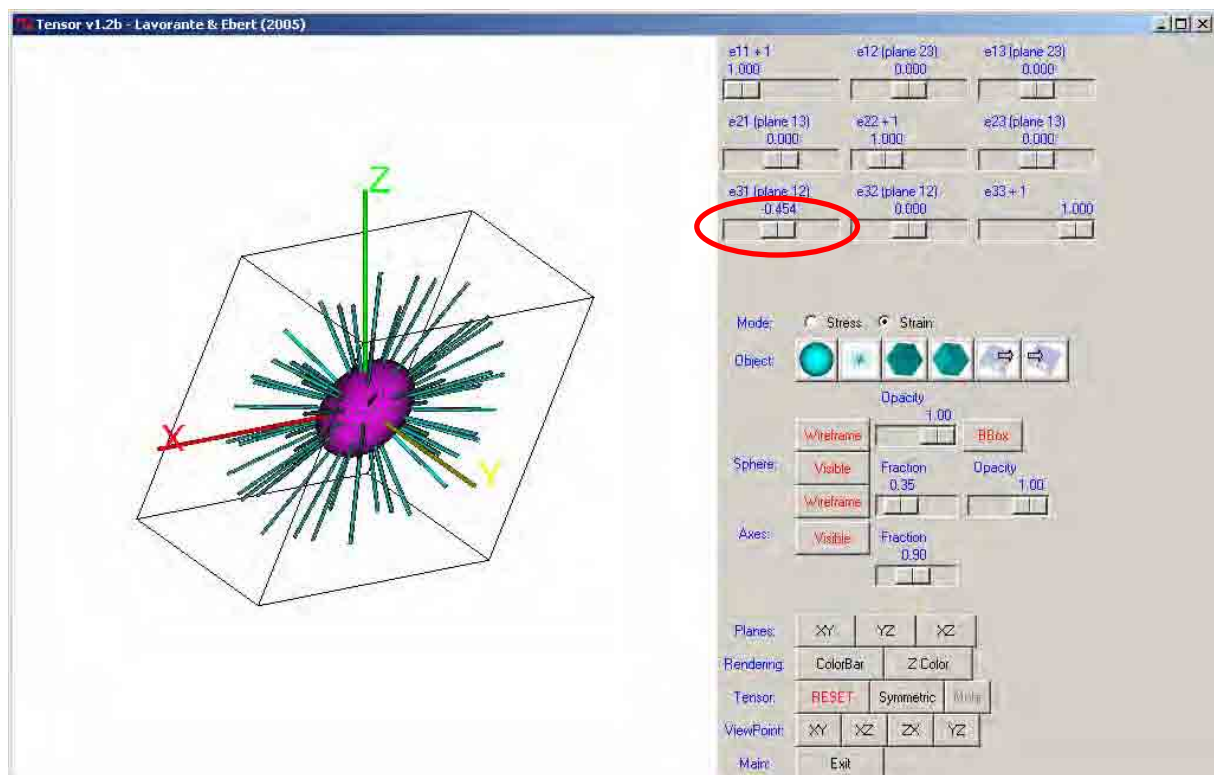
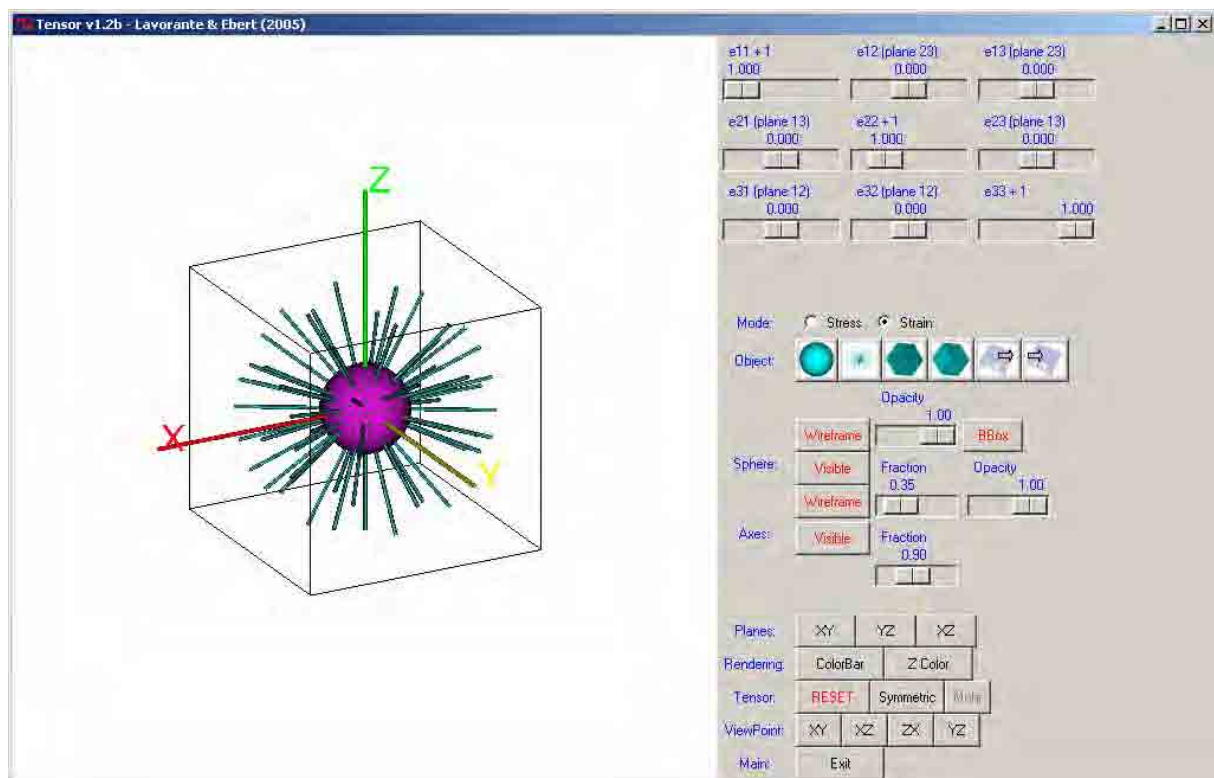


Figura 5.44 – Deformação não coaxial progressiva de um conjunto de marcadores lineares formando ângulos de 30 graus. Cisalhamento simples no plano XY e na direção negativa do eixo coordenado X. A componente do tensor modificada é indicada com elipses vermelhas (direita). A entidade da deformação é realçada exibindo o elipsóide no centro do modelo, bem como seu *Bounding Box* (esquerda).

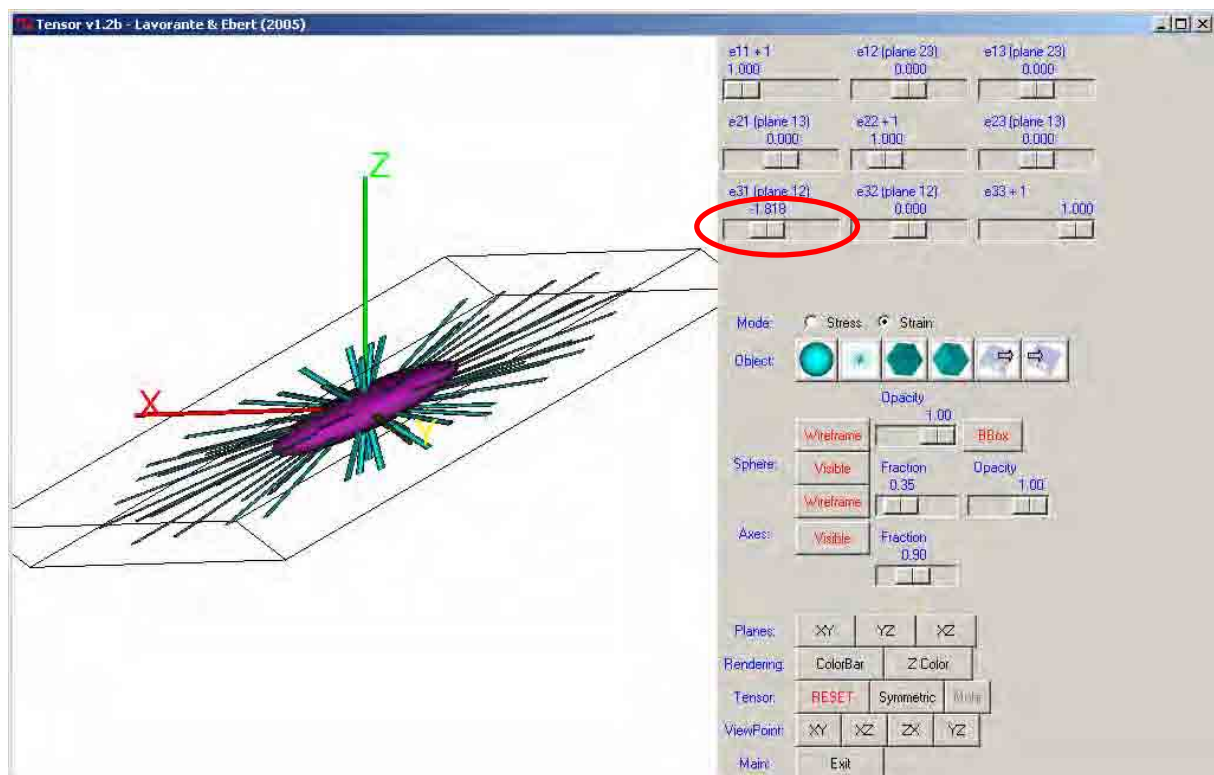
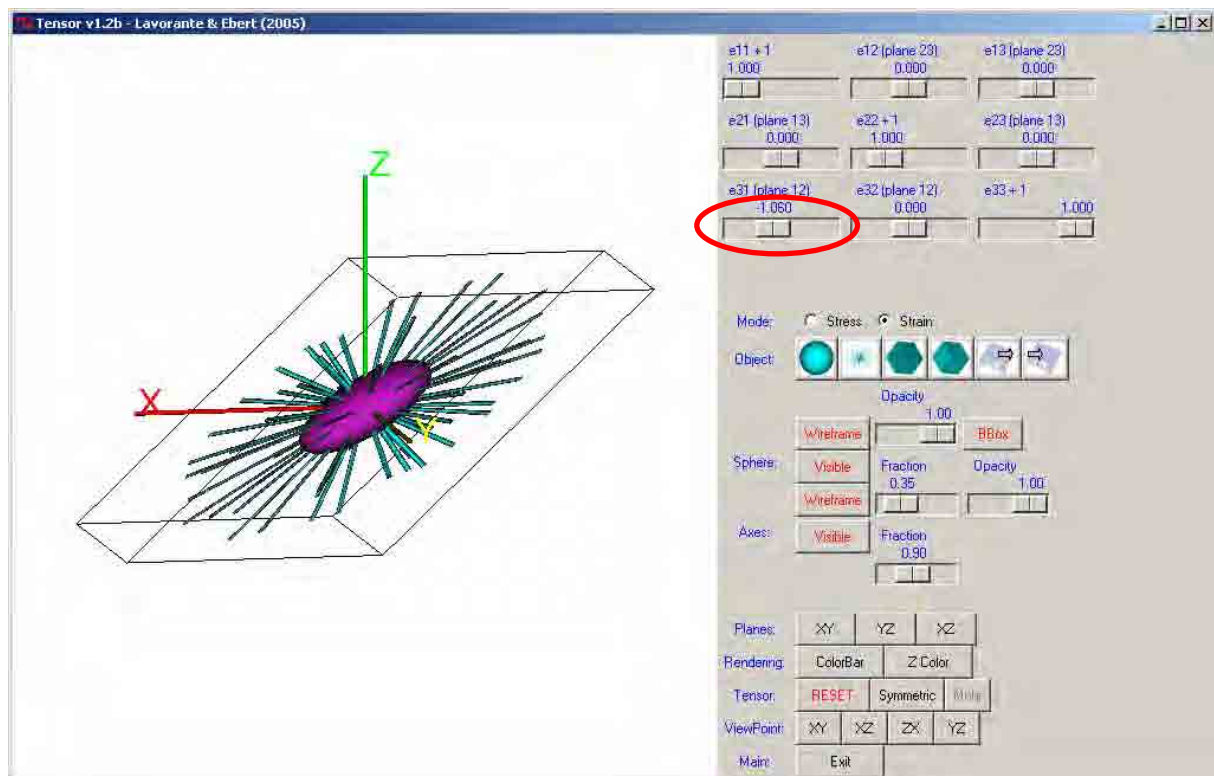


Figura 5.45 – Continuação da deformação não coaxial progressiva do objeto da figura anterior.

Superfície topográfica sintética. Trata-se de um modelo digital de terreno sintético, isto é, criado usando valores aleatórios gerados por computador. Para tanto foi utilizada a classe *vtkPerlinNoise*, a qual gera uma seqüência contínua e não periódica de números aleatórios (ruído). Isto permite modelar processos e objetos do mundo real. O modelo digital está contido no arquivo *objs/main/topo.vtk*. A Figura 5.46 mostra um exemplo de deformação não coaxial progressiva do objeto. É visível a formação de um sistema de dobras em bainha ao longo do processo.

Bacia de Taubaté. Este é o modelo digital da Bacia de Taubaté, cuja morfologia foi obtida através de técnicas sísmicas. A superfície poligonal foi criada usando o programa GOCAD (seção 3.6). O arquivo de dados foi exportado para o formato *.ts do programa, sendo necessária sua conversão para o formato *.vtk do *Tensor3D*. Para tanto, utilizou-se o programa *ts2dis*, que realiza uma conversão para o formato intermediário MOVIE.BYU (MOVIE.BYU FILE FORMAT, 2005), com extensão *.dis. O programa *ts2dis* foi desenvolvido por este mestrando durante seu período como bolsista de iniciação científica junto ao laboratório de GeoModelagem 3D do DPM / IGCE / UNESP. O arquivo *.vtk da bacia foi criado usando um *script* em Tcl/Tk que usa as classes *vtkBYUReader* (leitura do arquivo *.dis) e *vtkPolyDataWriter* (gravação do arquivo *.vtk). O formato interno de troca de informações geométricas e topológicas entre os dois filtros é o da classe *vtkPolyData*.

Na Figura 5.47 é mostrada uma deformação coaxial do modelo digital da Bacia de Taubaté. O modelo é sombreado usando uma escala de cores que acompanha a variação de elevação. É também visualizado o elipsóide de deformação para cada etapa. O último estágio do processo representa o ponto de partida para outra deformação, não coaxial, ilustrada na Figura 5.48. O resultado é interessante pois mostra a modificação simultânea de componentes do tensor normais e cisalhantes. Esta composição comprova também a extrema flexibilidade de um tensor matemático. Os exemplos anteriores mostram modificações de até três componentes da matriz, separando as deformações normal e cisalhante para fins didáticos. Contudo, em aplicações reais, é possível combinar modificações das nove (ou seis, em caso de simetria) componentes do tensor para simular processos deformacionais mais complexos. Finalmente, a utilização de modelos digitais importados de outro ambiente de trabalho amplia as potencialidades do programa *Tensor3D*, podendo-se utilizar dados reais, além dos modelos matemáticos gerados com a biblioteca VTK. A importação pode envolver vários modelos criados e visualizados com o programa GOCAD neste Laboratório de GeoModelagem 3D, sendo possível controlar e observar interativamente a simulação de sua deformação.

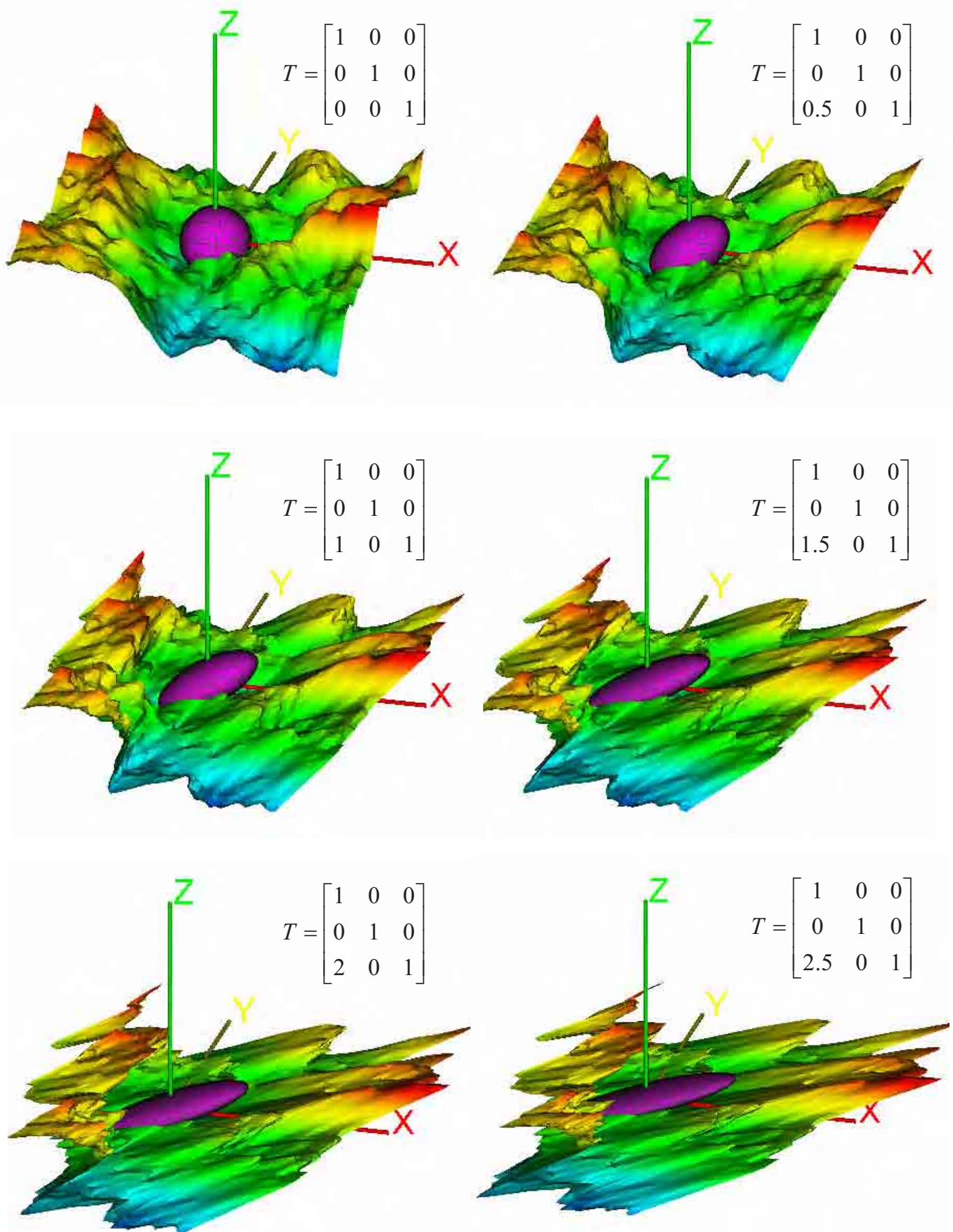


Figura 5.46 – Deformação não coaxial progressiva de um modelo digital de terreno gerado por computador. Cisalhamento no plano XY e na direção do eixo X. As matrizes mostram as componentes do tensor a cada etapa. O elipsóide de deformação é também exibido.

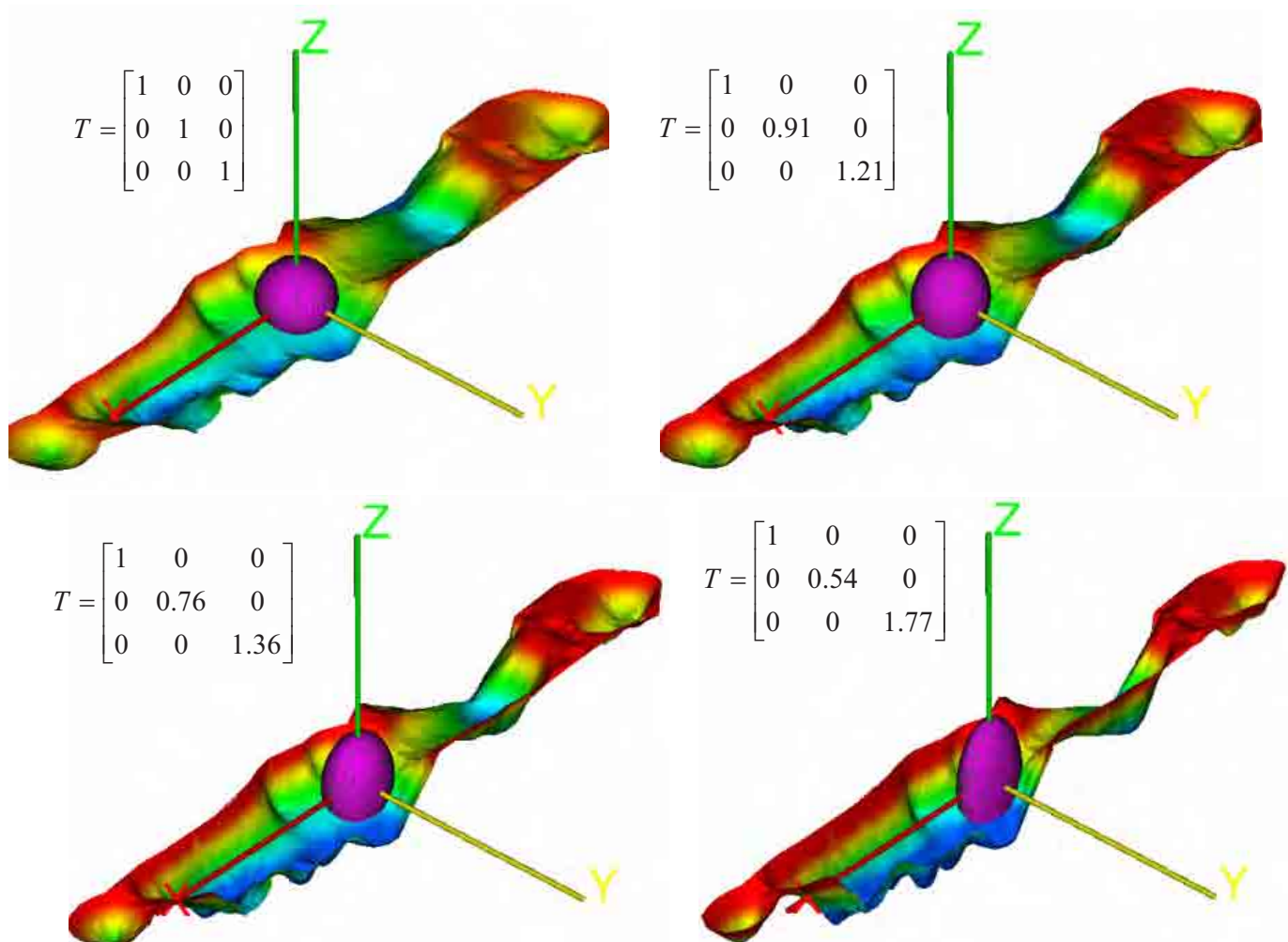


Figura 5.47 – Deformação coaxial progressiva do modelo digital da Bacia de Taubaté. O valor das componentes do tensor modificadas é mostrado a cada etapa. A deformação envolve um encurtamento na direção do eixo coordenado Y e um estiramento ao longo do eixo Z. A região central mostra o elipsóide de deformação.

Diápiro de sal. Este modelo foi criado deformando uma esfera no ambiente do GOCAD usando o algoritmo DSI de interpolação do programa (Mallet, 1989). Um modelo poligonal de esfera de 40 meridianos e 40 paralelos foi gerado com a classe *vtkSphereSource* do VTK, gravado em formato *.dis (MOVIE.BYU) e convertido para o formato *.ts do GOCAD (programa *dis2ts* desenvolvido por este mestrando) (Figura 5.49a). Neste programa, foram adicionados pontos de controle externos para orientar a interpolação do objeto (Figura 5.49b). Em seguida, a esfera foi deformada usando o conjunto de pontos de controle (Figura 5.49c). Por fim, o objeto no formato *.ts foi convertido, via programa *ts2dis*, de volta para o formato *.vtk (arquivo *objs/main/diampir.vtk* na estrutura de diretórios do programa *Tensor3D*).

A Figura 5.50 mostra a sequência de uma deformação coaxial (primeiras três etapas) e outra não coaxial (última etapa) para o modelo do diápiro de sal. A modelagem geométrica deste objeto é um exemplo de integração de ferramentas comerciais (GOCAD) e de código aberto (VTK), por meio de programas auxiliares de conversão.

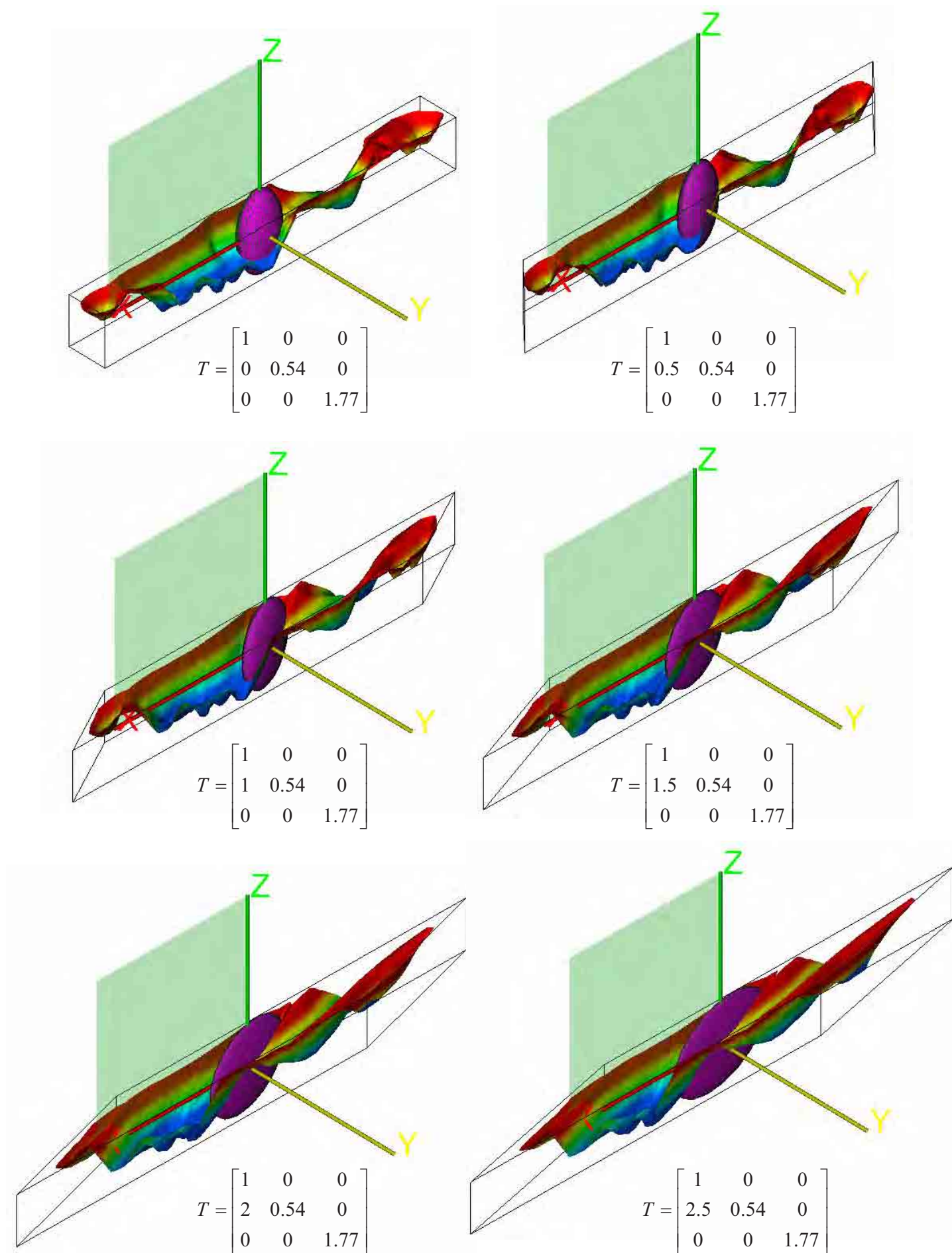


Figura 5.48 – Deformação não coaxial progressiva do modelo digital da Bacia de Taubaté. O processo começa com a etapa final da deformação coaxial da figura anterior. Cisalhamento simples no plano XZ (vertical) e na direção do eixo X, paralelo ao eixo maior da bacia.

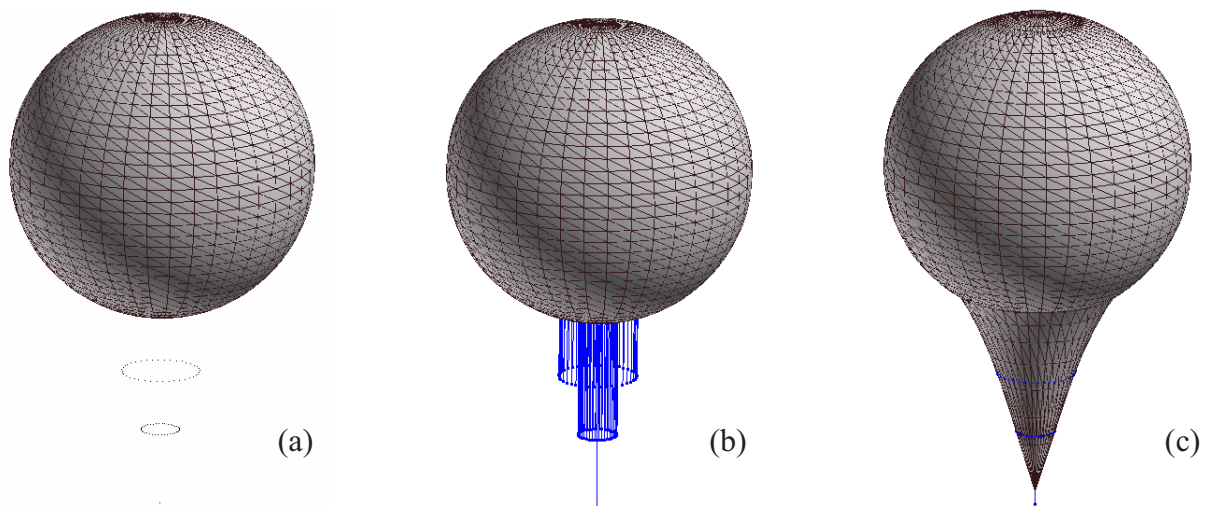


Figura 5.49 – Modelagem de um diápiro no ambiente GOCAD. (a) Importação da representação poligonal de uma esfera criada usando o VTK e adição de pontos auxiliares (região inferior). (b) Definição dos pontos como vínculos (em azul) para a deformação do objeto. (c) Interpolação (algoritmo DSI do GOCAD).

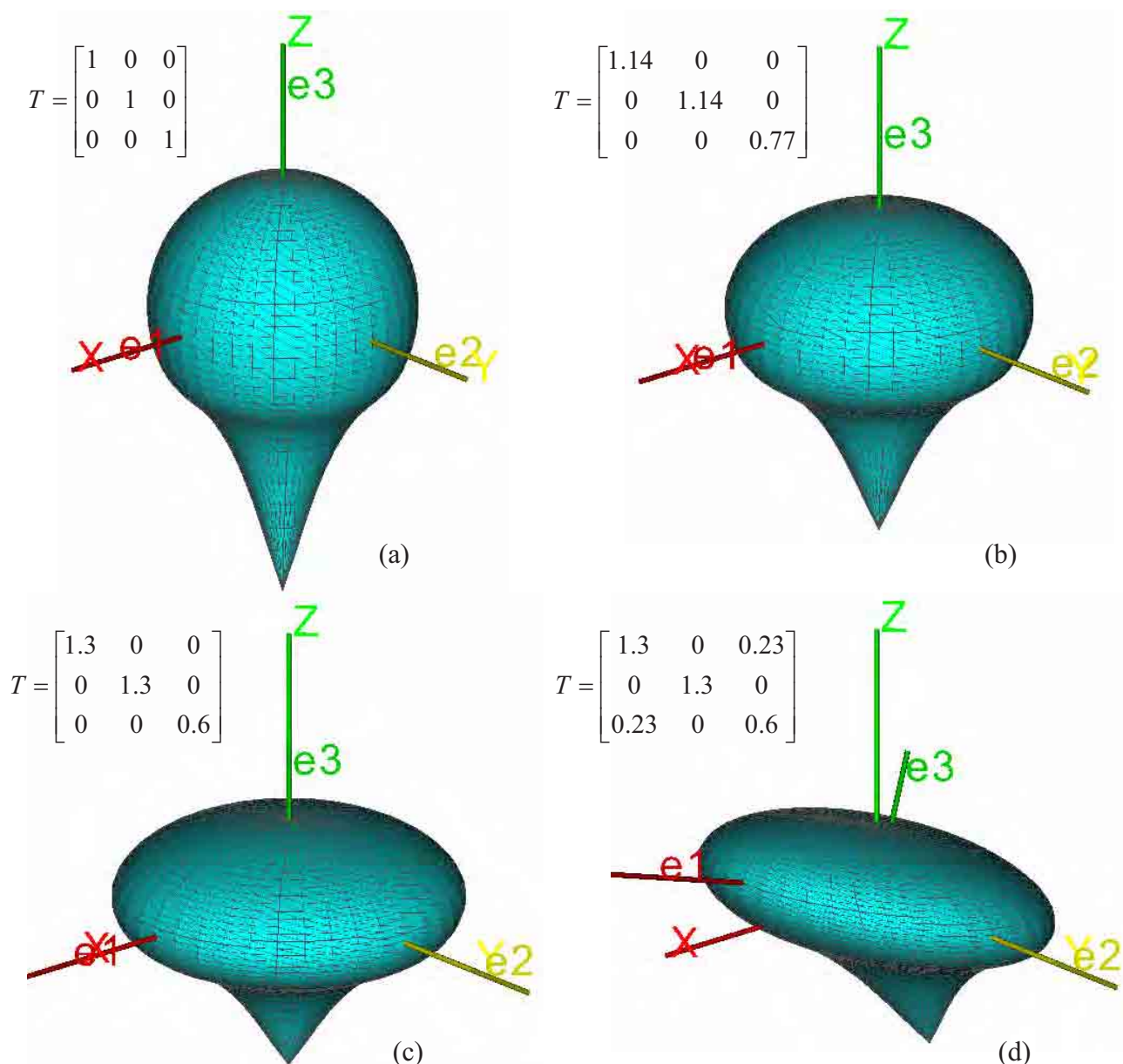


Figura 5.50 – Deformação progressiva do modelo de um diápiro em modalidade de simetria do tensor. (a), (b) e (c) Deformação coaxial com modificação simultânea das componentes normais do tensor para simular volume constante do objeto. (d) Deformação não coaxial (plano XZ) a partir da última etapa do processo anterior. Os eixos principais de deformação $\hat{e}_1$ e $\hat{e}_3$ deixam de ser paralelos a X e Z, respectivamente.

Cubos achatados. Este objeto foi criado usando a biblioteca VTK (*script aus/biscuits.tcl*), sendo composto de três grupos de cubos achatados e alinhados com os planos coordenados (Figura 5.51a). Cada grupo contém 16 cubos. O primeiro grupo a ser criado foi o conjunto no plano XY. Para tanto, foram usadas instâncias da classe *vtkCubeSource*. O fator de escala dos cubos foi especificado usando métodos desta classe e corresponde a 1:1:0.5. Todas as instâncias foram combinadas usando a classe *vtkAppendPolyData*. O conjunto de 16 cubos achatados do plano XY foi passado como entrada para duas instâncias da classe *vtkTransformPolyData*, as quais operaram, através da classe *vtkTransform*, duas rotações distintas: uma de 90 graus em torno do eixo X e outra de -90 graus em torno do eixo Y. Desta forma foram criados os conjuntos de cubos alinhados aos eixos XZ e YZ, respectivamente. O objeto final resulta de uma composição realizada através de outra instância da classe *vtkAppendPolyData*, e foi gravado em disco com o nome *biscuits.vtk* (diretório *objs/main*).

No exemplo da Figura 5.51 foram realizadas uma deformação coaxial e outra não coaxial, antes separadamente e em seguida de forma combinada.

Além dos modelos ilustrados nesta seção, o diretório *objs* do programa *Tensor3D* contém outros objetos usados para testar determinadas funcionalidades do aplicativo. Existem também outros agregados de objetos criados usando o programa *Object Factory 3D*. Por exemplo, o diretório *objs/main* contém modelos de agregados monominerálicos isotrópicos: os objetos usados são cilindros (arquivo *agregado_cil_iso.vtk*), pirâmides (*agregado_cones_iso.vtk*), cubos deformados em paralelepípedos (*agregado_cubes_iso.vtk*) e esferas deformadas em elipsóides (*agregado_esf_iso.vtk*).

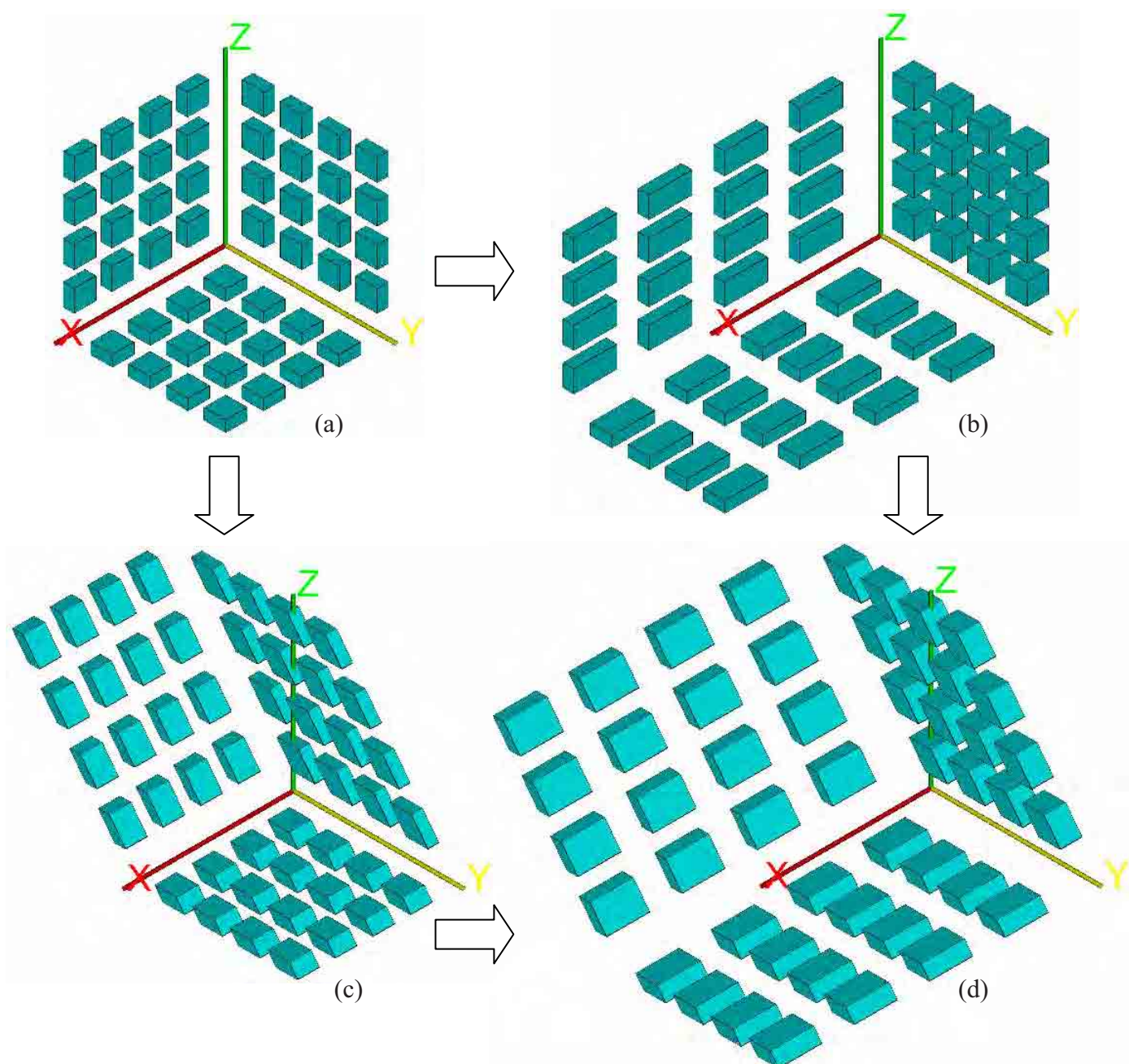


Figura 5.51 – Deformação de um conjunto de cubos achatados. Os objetos estão alinhados com os planos coordenados XY, YZ e XZ. (a) Estado indeformado. (b) Deformação coaxial: estiramento na direção do eixo X por um fator de 2.0. (c) Deformação não coaxial a partir do estado (a): cisalhamento simples no plano XY e na direção negativa do eixo Y ($e_{32} = -0.762$). (d) composição das duas deformações anteriores.

6 Integração das bibliotecas VTK e GTK

Como foi mostrado nas seções anteriores, o sistema VTK oferece numerosos módulos flexíveis que implementam os principais algoritmos de visualização científica. O resultado do processamento é visualizado em uma janela tridimensional. Para aplicações complexas, onde seja necessária a introdução de informações pelo usuário, é importante dispor de janelas de interação constituídas por botões, barras de rolagem, campos para introdução de texto e números, etc. Tais funcionalidades são proporcionadas pela biblioteca GTK (programas compilados) e linguagem Tcl/Tk (programas interpretados).

Considerando a possibilidade de construir aplicações de visualização mais complexas inerente aos programas *compilados* baseados na biblioteca VTK (seção 3.4.3), e objetivando preparar um ambiente de trabalho para aplicações tridimensionais interativas, este mestrando conduziu estudos no sentido de integrar as bibliotecas VTK e GTK no mesmo projeto. O ambiente de desenvolvimento escolhido foi o *anjuta* (seção 3.5.2), que possui numerosas facilidades para o programador.

6.1 Compilação dos módulos de trabalho

A integração de VTK e GTK envolveu o desenvolvimento de aplicações híbridas, isto é, baseadas nas linguagens C++ (utilizada pela biblioteca gráfica VTK) e C (utilizada para o desenvolvimento da biblioteca GTK). Esta integração, que aparentemente deveria ocorrer de forma simples devido às ferramentas de apoio oferecidas pelo *anjuta*, tornou-se um problema justamente dentro deste ambiente de desenvolvimento. Em particular, o *anjuta* não permitia a compilação de simples programas de teste em C++ que integrassem as bibliotecas mencionadas.

O programa *anjuta* cria um arquivo de *Makefile* dentro do diretório de projeto, o que permitiu examinar seu conteúdo e tentar executar a compilação manualmente através do comando *make*. Infelizmente, a estrutura deste arquivo é excessivamente complexa para permitir uma análise, e a invocação do comando *make* gerava os mesmos erros de compilação exibidos no ambiente do *anjuta*.

Após numerosas tentativas de compilação, foram consultadas listas de discussão na página Web do *anjuta* sobre o problema de integração de códigos em C e C++, o que confirmou a incapacidade do sistema em gerenciar simultaneamente módulos nas duas linguagens. Em particular, os desenvolvedores da ferramenta informaram que o módulo de

criação do arquivo de *Makefile* no *anjuta* apresenta falhas e prometeram uma revisão na próxima versão do *software*.

Sendo o ambiente *anjuta* uma ferramenta de desenvolvimento muito prática, que permite acelerar o gerenciamento de projetos complexos, este mestrando decidiu escrever manualmente arquivos de *Makefile*. Apesar desta prática contrastar com a facilidade de uso do *anjuta*, trata-se de uma operação a ser executada uma única vez para cada projeto, sendo que um mesmo arquivo de *Makefile* pode ser utilizado para outros projetos com pequenas modificações.

Após a criação de um arquivo de *Makefile*, a ferramenta *anjuta* conseguiu compilar um programa de teste com êxito. A Figura 6.1 mostra o resultado da execução de um programa compilado que exibe as janelas do VTK e do GTK simultaneamente. Alguns aspectos da integração entre estas ferramentas serão descritos nas próximas seções.

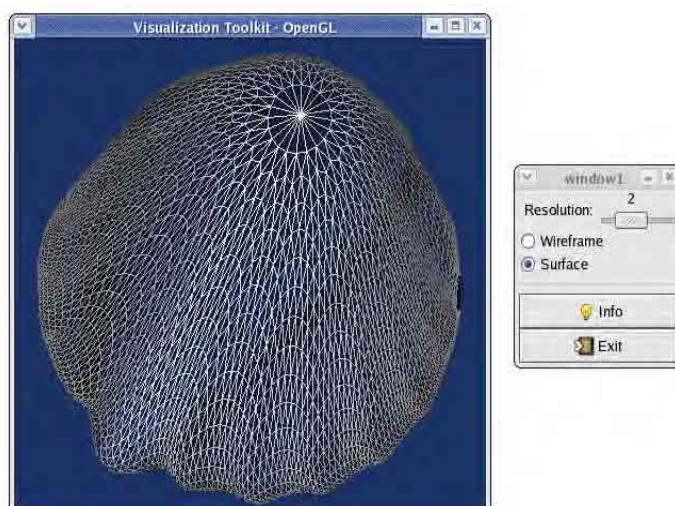


Figura 6.1 – Exemplo de integração da biblioteca gráfica VTK (esquerda) com a ferramenta de interface GTK. A interação do usuário com a janela do GTK (direita) permite modificar as propriedades geométricas e de visualização do objeto exibido.

6.2 Gerenciamento de múltiplos ciclos de interação

A compilação de programas com as bibliotecas VTK e GTK permitiu iniciar o estudo de integração destas ferramentas dentro de uma mesma aplicação. Um problema considerado desde o início foi a presença de um evento principal de execução (*main loop*) para cada biblioteca. Este conceito será explicado a seguir. Para tanto, é necessário descrever brevemente as modalidades de execução das duas bibliotecas.

O uso da biblioteca VTK subteme instanciação de classes (criação de objetos de determinados tipos) e *interligação* das mesmas, de forma que a saída de processamento de uma seja a entrada para o processamento da classe seguinte, e assim por diante. Após a

construção de um *pipeline*, ou fluxo de dados entre classes, a interação do usuário com a janela 3D de visualização somente é possível através do módulo *vtkRenderWindowInteractor*, o qual aguarda eventos associados ao mouse e ao teclado e atualiza o conteúdo da janela 3D adequadamente. Para isto, o módulo precisa executar um ciclo (ou *loop*) contínuo de instruções, durante o qual “monopoliza” a execução do programa, isto é, não permite a execução de instruções externas ao ciclo. Tal ciclo é virtualmente infinito, no sentido de que dura enquanto o programa estiver em execução. Isto não representa um problema contanto que a visualização pelo VTK seja o único módulo do programa em funcionamento.

Programas baseados na biblioteca GTK apresentam a mesma característica. Após a criação e visualização de todas as componentes de interação do usuário com a interface (botões, campos para informações, etc.), um programa baseado nesta biblioteca necessita “entrar” em um ciclo de execução virtualmente infinito. Durante o mesmo, a biblioteca verifica continuamente se o usuário usou o mouse sobre alguma componente (“clicar” sobre algum botão, “deslizar” uma barra de rolagem, etc.) e atualiza a interface adequadamente.

Este ciclo de interação (*main loop*) é executado até o usuário finalizar a aplicação e, de forma semelhante à classe *vtkRenderWindowInteractor*, monopoliza o programa dentro do qual estiver executando.

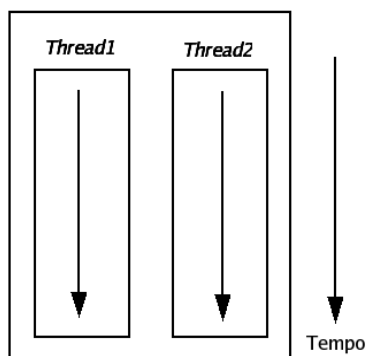


Figura 6.2 – Exemplo de *threads* contidas em um processo.

Para promover a integração das bibliotecas VTK e GTK em uma mesma aplicação, foram implementadas diferentes *threads* de execução. Uma *thread* pode ser vista como um processo distinto, apesar de representar uma unidade de execução computacionalmente menos dispendiosa (em termos de utilização de recursos do sistema operacional) que um processo tradicional em ambiente UNIX. Um mesmo processo pode conter várias *threads* que são executadas simultaneamente e de forma independente (Figura 6.2). Isto permite a realização em paralelo de tarefas computacionais que têm pouca ou nenhuma relação entre si.

Considerado que as bibliotecas VTK e GTK possuem ciclos de interação independentes, a execução das mesmas foi atribuída a duas *threads* distintas, sendo a primeira responsável pelo processamento gráfico, e a segunda pelo gerenciamento da interface com o usuário. Após a inicialização de suas componentes, cada *thread* ingressa seu ciclo de eventos, aguardando a interação com o usuário. Tal interação pode ocorrer com a janela 3D do VTK (Figura 6.1), promovendo a atualização do ponto de observação, *zoom*, etc.; ou com a janela

de interface do GTK, permitindo a utilização dos botões e demais componentes presentes na mesma.

No ambiente Linux é possível definir *threads* de execução através da biblioteca *pthread*, que foi utilizada neste trabalho. Tal biblioteca, desenvolvida em C, contém numerosas funções para controlar a criação, execução e destruição de *threads*, bem como mecanismos de sincronização entre as várias unidades de processamento.

6.3 Sincronização e comunicação entre processos de execução (*threads*)

Em uma aplicação *multi-threaded*, isto é, constituída de várias *threads* independentes que executam em paralelo, é muitas vezes necessário que as unidades em execução troquem informações entre si. Um exemplo disto é um programa de visualização científica com a opção de ler dados geométricos a partir do disco. Normalmente, durante a leitura dos dados, o usuário pode interagir com a janela principal de visualização, modificando algumas propriedades gráficas, o que indica que a execução dos módulos de visualização e leitura de dados ocorre através de *threads* distintas. No entanto, terminada a leitura do disco, o módulo de visualização deve proceder à construção dos primitivos gráficos correspondentes. O momento em que isto ocorre é controlado por algum mecanismo de comunicação entre as *threads* da aplicação. Uma maneira de garantir tal comunicação consiste na utilização de variáveis globais, isto é, variáveis de estado cujo conteúdo possa ser acessado por todas as *threads*. No exemplo do programa anterior, a *thread* de visualização poderia consultar periodicamente o estado de uma variável booleana (com valores 1 ou 0) para verificar se é necessário realizar a atualização do conteúdo da janela 3D. Contudo, já que a execução das *threads* de leitura e de visualização ocorre de forma paralela, é possível que a segunda leia o conteúdo da variável exatamente no momento em que a primeira estiver modificando seu valor. Isto pode provocar falhas graves de execução.

Para garantir o correto funcionamento de uma aplicação *multi-threaded* onde os vários módulos de execução trocam informações entre si, é necessário assegurar que, enquanto uma *thread* estiver escrevendo uma informação compartilhada, as demais não tentarão realizar sua leitura. Para isto, utiliza-se *ferramentas de sincronização* providas pelo sistema operacional.

Com este método, sempre que uma *thread* estiver prestes a escrever em uma variável global, é ativado um bloqueio, ou *lock*, o qual impede que as demais *threads* em execução concorrente realizem sua leitura. Na prática, a execução destas *threads* de leitura é suspensa até que a *thread* de escrita abandone o bloqueio sobre a variável compartilhada. Tal

comportamento deve ser codificado de forma explícita pelo programador. Desta forma, o pseudo-código para uma *thread* de escrita será:

```
Ative_lock;  
    Atualize_variável;  
Desative_lock;
```

O código das *threads* que executam a leitura da variável compartilhada será:

```
Ative_lock;  
    Leia_variável;  
Desative_lock;
```

A indentação (espaçamento) indica que as instruções dentro dos comandos de gerenciamento de *lock* estão contidas em uma região crítica de código, isto é, uma região onde somente pode “entrar” (ser executada) uma *thread* de cada vez.

O exemplo de aplicação ilustrado acima é relativamente simples, mas existem casos muito mais complexos de utilização de bloqueios para *threads*. Bloqueios são também definidos de forma distribuída, isto é, transpondo os limites físicos de um computador e utilizando redes de computadores para a comunicação entre processos em máquinas distintas. Por exemplo, em uma aplicação de bancos de dados distribuídos é necessário garantir que um registro não seja lido enquanto algum processo (em outro computador remoto) estiver atualizando suas informações. Um sistema de visualização complexo constituído de múltiplos nós de processamento constitui uma aplicação típica.

Voltando ao problema de integração das bibliotecas VTK e GTK, espera-se que o usuário possa modificar propriedades gráficas do objeto visualizado utilizando botões e demais componentes de interação da janela do GTK. Por exemplo, a Figura 6.1 mostra que o usuário pode modificar o aspecto de uma superfície, selecionando o sombreado de tipo *Wireframe* ou *Gouraud* (seção 3.1.1). Tais informações precisam ser notificadas para a *thread* do VTK, de forma que esta realize as atualizações gráficas necessárias. Uma variável global chamada *wireframe*, compartilhada entre as *threads* do VTK e do GTK, poderia assumir o valor 1 em caso de visualização da malha vazada e o valor 0 para a modalidade *Gouraud*.

Para garantir a sincronização de acesso à variável compartilhada, foi utilizado um mecanismo de bloqueio denominado *mutex*, o qual faz parte do conjunto de ferramentas oferecidas pela biblioteca *pthread* do sistema operacional Linux. Dentre as funções de gerenciamento de *mutex* (ou *locks*) que a biblioteca oferece, pode-se destacar:

- *pthread_mutex_init*. Providencia a criação de um bloqueio.
- *pthread_mutex_destroy*. Destrói um bloqueio não mais necessário.
- *pthread_mutex_lock*. Tenta obter exclusividade de acesso através de um bloqueio. Caso o bloqueio já tenha sido obtido por outra *thread*, a *thread* atual será suspensa.
- *pthread_mutex_unlock*. Abandona o bloqueio permitindo o acesso às demais *threads*.
- *pthread_mutex_trylock*. Semelhante à função *pthread_mutex_lock*, mas se o bloqueio já estiver ativado, a *thread* atual não é suspensa e continua a execução das instruções seguintes. A *thread* poderá tentar o bloqueio na próxima execução.

Como foi explicado acima, os ciclos principais de execução do VTK e do GTK (*main loops*) precisam acessar periodicamente o conteúdo das variáveis compartilhadas para trocarem informações (por exemplo, se o usuário selecionou uma modalidade de visualização diferente na interface GTK). Um problema grave, neste sentido, é que o VTK não possui mecanismos deste tipo e, após ingressar no ciclo principal de eventos da classe *vtkRenderWindowInteractor*, não é possível invocar funções externas de forma periódica.

Existem funções associadas a eventos gerados na janela de visualização, como uso do teclado e do mouse, para os quais é possível definir o acesso às variáveis compartilhadas com a *thread* que executa o GTK. Mas todas estas funções têm sua execução subordinada à ocorrência de algum evento *externo* ao gerenciador *vtkRenderWindowInteractor*, não sendo possível gerar eventos *internos* (independentes) de forma periódica. Este problema implica em uma situação de incomunicabilidade da *thread* associada ao VTK com as demais *threads* de execução. De fato, não sendo possível a comunicação entre os módulos de visualização e de interface com o usuário, todas as componentes oferecidas pela biblioteca GTK se tornariam inúteis.

Uma busca aprofundada no grupo de discussão do VTK na rede Internet (VTK MAILING LIST, 2004), para uma possível integração das duas bibliotecas mostrou que isto ainda não ocorreu. Alguns desenvolvedores do grupo aconselharam escrever um gerenciador de eventos híbrido, que contivesse funções VTK e GTK, eliminando também a necessidade de se trabalhar com *threads* distintas. A escrita de tal gerenciador é uma tarefa bastante complexa que este mestrando preferiu evitar por causa do tempo de desenvolvimento envolvido.

Não podendo forçar o gerenciador da biblioteca VTK a gerar periodicamente eventos internos para consultar o estado das variáveis de comunicação com as demais *threads*, optou-se pela utilização dos eventos *externos* (não provocados pelo VTK) que a biblioteca pode gerenciar.

Toda classe do VTK invoca dois eventos, *StartEvent* e *EndEvent* correspondentes, respectivamente, ao início e fim de seu processamento. Quando alguma janela se sobrepõe à janela principal de visualização (por exemplo, arrastada pelo usuário utilizando o mouse), o conteúdo desta é atualizado automaticamente. Além disto, são invocados os eventos *StartEvent* e *EndEvent* da classe *vtkRenderer*. Esta classe cria o conteúdo da janela 3D a partir das informações obtidas de módulos antecedentes do *pipeline* de visualização (fontes de dados, filtros, atores, posição da câmera, etc.). Para forçar a execução de um evento de atualização da janela 3D, as funções associadas às componentes da interface GTK foram implementadas de forma a criar e destruir uma pequena janela (dimensão de 10x10 pixels) sobre a janela do VTK. Isto provoca a execução de todos os módulos do *pipeline* responsáveis pela visualização, atualizando o conteúdo da janela 3D. Outro efeito é a execução de funções associadas aos eventos *StartEvent* e *EndEvent* para a classe *vtkRenderer*. Estas funções foram codificadas de forma a consultarem o conteúdo das variáveis de controle entre as *threads* do VTK e do GTK.

O mecanismo de comunicação entre *threads* implementado não constitui a forma mais elegante de promover a troca de informações de controle entre os gerenciadores de eventos das bibliotecas VTK e GTK. Apesar disto, a implementação escolhida mostrou-se eficaz e isto permitiu alocar mais tempo para a execução das demais atividades de desenvolvimento.

O CD que acompanha esta dissertação contém o código fonte de todos os módulos desenvolvidos para integrar as bibliotecas VTK e GTK. Para acessar o código, é suficiente localizar o diretório *VTK_GTK*. Dentre os vários arquivos, destacam-se:

- *Makefile*. Arquivo responsável pela compilação do aplicativo (seção 3.5.1).
- *calls.c*. Conjunto de funções que controlam o comportamento associado às componentes gráficas (botões e barra de rolagem) da biblioteca GTK (seção 3.5.3).
- *graphics.cc*. Conjunto de funções destinadas à criação e manipulação de classes do sistema VTK.
- *interface.c*. Contém funções para criar a estrutura da interface gráfica do GTK. É gerado automaticamente após a interação com a ferramenta *glade* (seção 3.5.4).
- *support.c*. Arquivo auxiliar também gerado pela biblioteca *glade*.
- *main.cc*. Arquivo que contém a função principal de execução do programa.
- *vtk_gtk*. Código executável do aplicativo de integração (Figura 6.1).

A compilação do aplicativo requer a execução do comando *make* a partir do diretório que contém os módulos de código fonte.

7 Modelagem 3D da paleobatimetria das bacias de Santos e Campos

Nesta seção serão ilustradas técnicas de computação gráfica 3D utilizadas para modelar dados paleobatimétricos das bacias de Campos e Santos. A Figura 7.1 mostra a localização da área de estudo e os limites da paleobatimetria modelada para a região. Através destas técnicas, objetivou-se ampliar as capacidades de interpretação de mapas bidimensionais gerados em investigações paleoambientais do meso-Neocretáceo (do Albiano ao Maastrichtiano, 113 a 65 Ma). A ferramenta de modelagem e visualização empregada foi o programa GOCAD (2005) em sua versão 2.1.2.

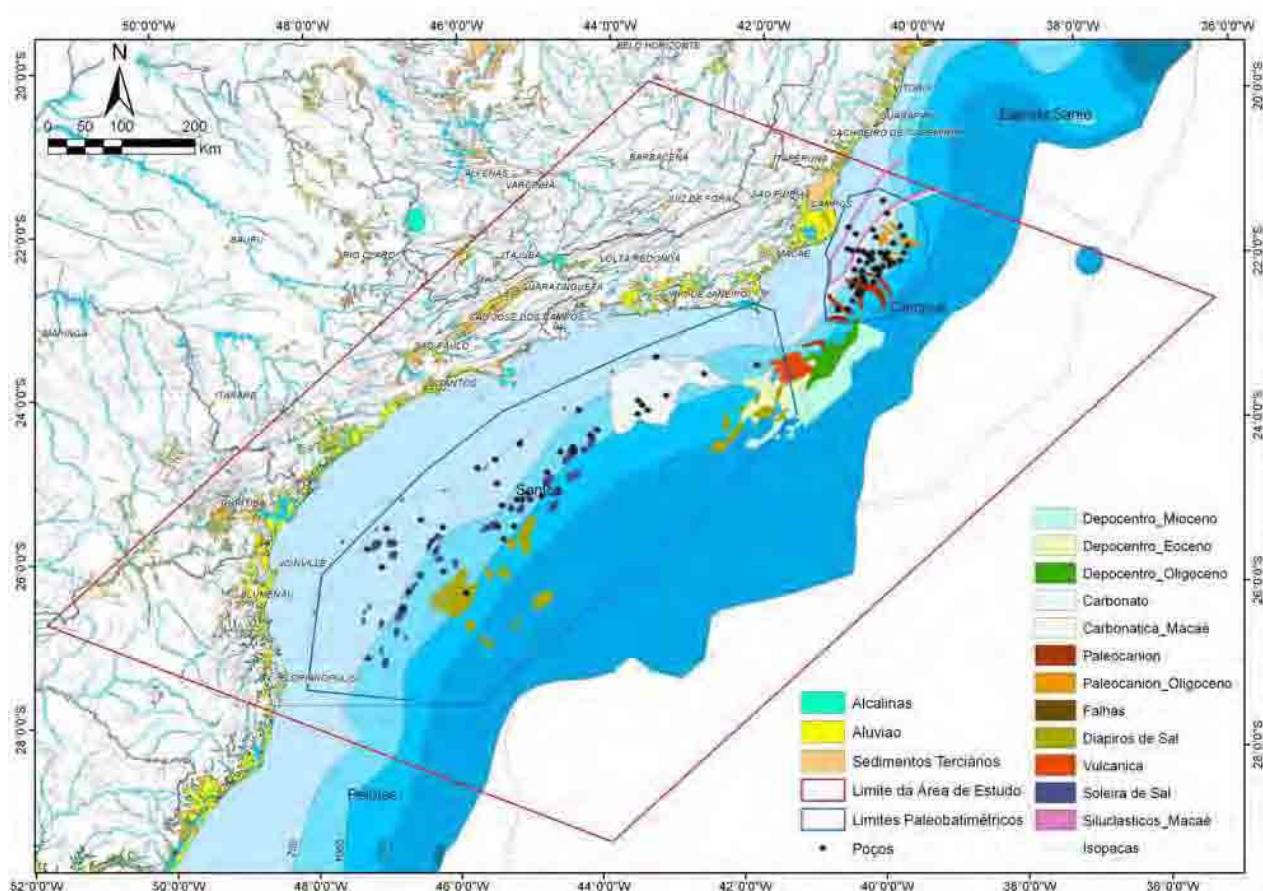


Figura 7.1 – Localização da área de estudo (polígono vermelho), limites das paleobatimetrias modeladas neste trabalho (polígonos azuis) e poços utilizados para confecção dos mapas de Viviers (1986) e Azevedo et al. (1987). São também exibidas as estruturas oceânicas, continentais, litotipos cenozóicos, rede de drenagens, informações batimétricas, bem como limites das bacias de Espírito Santo, Campos, Santos e Pelotas.

Resultados preliminares referentes à paleobatimetria da Bacia de Campos encontram-se em Ebert & Lavorante (2000). Neste trabalho, foram digitalizadas as curvas de batimetria da bacia de Campos, utilizando as folhas topográficas de Macaé e Campos na escala 1:250.000. Também foram digitalizadas as curvas paleobatimétricas da Bacia de Campos, do Albiano ao Maastrichtiano, estimadas por Azevedo et al. (1987), visando analisar a variação morfológica da bacia ao longo da história geológica que passa do eo-mesoalbio, neo-

albiano, eoturoniano, eoconiaciano, neo-santoniano até o neomaastrichtiano. No ambiente de visualização do programa GOCAD foi possível representar todas as superfícies tridimensionais simultaneamente, atribuindo-lhes uma coloração de acordo com uma escala de cores baseada na altitude relativa de cada modelo.

O ponto de partida do processo de modelagem realizado neste mestrado é representado pelos trabalhos de Viviers (1986) e Azevedo et al. (1987), os quais objetivaram estabelecer um arcabouço bioestratigráfico e paleobatimétrico para as bacias de Santos e Campos, respectivamente, no contexto da exploração petrolífera. Estes trabalhos, de natureza micropaleontológica, contêm mapas paleobatimétricos associados a topos de seqüências deposicionais identificadas, tendo como base o estudo de amostras de calhas e de testemunhos obtidos a partir de 39 poços para a Bacia de Santos e de 49 poços para a Bacia de Campos. A delimitação das biozonas correspondentes, aliada a informações litoestratigráficas e sismoestratigráficas, permitiu inferir a evolução paleoambiental das bacias para o intervalo Albiano-Maastrichtiano, em seis momentos distintos (Figura 7.2).

ERA	PERÍODO	ÉPOCA	IDADE		IDADE (m.a.)	BACIA DE SANTOS	BACIA DE CAMPOS
MESOZÓICA	CRETÁCEO	SUP.		MAASTRICHTIANO	65		
			NEO	CAMPANIANO	73		
			MESO				
			EO				
			NEO	SANTONIANO	82,9		
			MESO				
			EO				
			NEO	CONIACIANO	87,5		
			MESO				
			EO				
			NEO	TURONIANO	88,5		
			MESO				
			EO				
			NEO	CENOMANIANO	91		
			MESO				
			EO				
		INF.	NEO	ALBIANO	97,5		
			MESO				
			EO				
					113		

Figura 7.2 – Posição temporal aproximada (linhas vermelhas) dos mapas paleobatimétricos para as bacias de Santos e Campos (de acordo com Viviers, 1986; e Azevedo et al., 1987). Idades de acordo com Harland et al. (1982).

Em trabalho posterior, Viviers e Azevedo (1988) subdividiram a história deposicional da porção sudeste da margem continental brasileira em cinco seqüências, a partir das quais obtiveram mapas paleobatimétricos integrados, visando homogeneizar os resultados que haviam obtido separadamente. As idades dos mapas consideradas neste trabalho correspondem às dos mapas que Azevedo (1987) havia definido para a Bacia de Campos. Na presente dissertação, optou-se pela utilização dos mapas paleobatimétricos dos trabalhos separados, que apresentam mais detalhes morfológicos que os mapas integrados. Além disto, foram encontradas similaridades com os mapas confeccionados em separado, sobretudo no tocante à paleobatimetria da Bacia de Campos. Por fim, a maioria dos mapas apresenta um consistente hiato de direção EW do norte na Bacia de Santos ao Sul da Bacia de Campos, tornando justificável, em termos de potencial descritivo, o uso de modelagens distintas para as bacias.

7.1 Digitalização e georreferenciamento das curvas

Os mapas paleobatimétricos foram rasterizados e importados no programa AutoDesk Map (2002). Para georreferenciar os mesmos, surgiu a necessidade de identificar ao menos duas coordenadas geográficas que servissem como base. Para tanto, foram utilizadas coordenadas da batimetria e de poços extraídos do *site* do BDEP/ANP (2004).

Inicialmente os mapas foram georreferenciados com as coordenadas de dois pontos da linha de costa (Figura 7.3). No caso específico, as coordenadas estavam na projeção UTM (Universal Transversa de Mercator), SAD 69, zona 23 sul. Para isso utilizou-se o comando *Align*, selecionando-se um ponto conhecido, e inserindo os valores das coordenadas correspondentes. Em seguida foram utilizados os valores das coordenadas dos poços no mesmo sistema de projeção.

Com o mapa georreferenciado, iniciou-se o trabalho de digitalização das curvas paleobatimétricas e dos poços (39 para a Bacia de Santos e 49 para a Bacia de Campos) (Figura 7.4). Para isso utilizou-se a opção *Polyline*.

Os ambientes de deposição considerados durante a digitalização foram o continental-parálico (10-0 m de altitude), plataformar (raso: 0 a 40 m de profundidade; médio: 40-100 m; profundo: 100-200 m) e batial (superior: 200-500m de profundidade; médio: 500-1000 m; inferior: 1000-2000 m). Isto permitiu atribuir elevações diferentes às isolinhas que denotam a transição entre ambientes adjacentes. A Figura 7.5 mostra os valores de batimetria associados aos limites dos ambientes considerados. Cada ambiente é identificado através de um padrão de cores que foi associado à variação batimétrica das superfícies modeladas.

Os conjuntos de curvas obtidos foram exportados no formato DXF, gerando-se seis arquivos para cada bacia, correspondentes aos momentos cretácicos indicados na Figura 7.2. Nas legendas das figuras referentes às várias idades de visualização paleobatimétrica, o termo “final” está entre aspas em razão do diacronismo do topo do intervalo estratigráfico.

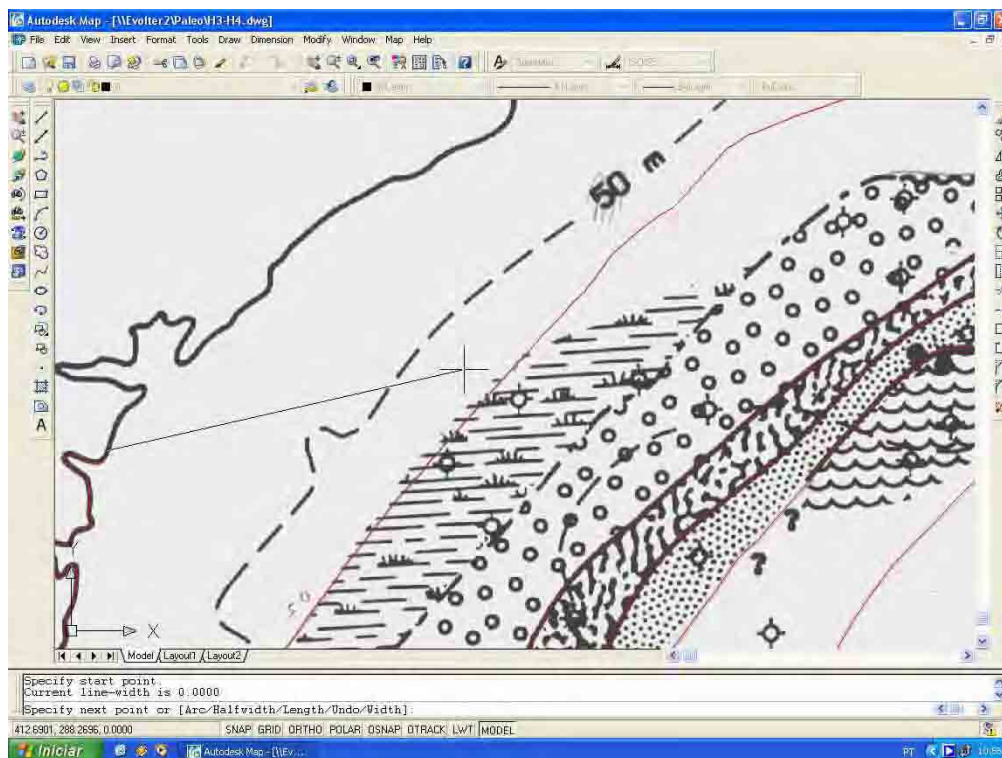


Figura 7.3 – Digitalização, no ambiente AutoDesk Map, da linha de costa atual (esquerda) para realizar o georreferenciamento dos mapas paleobatimétricos 2D. Bacia de Santos, situação ao “final” do neo-albiano (97 Ma). Mapa original: Viviers (1986).

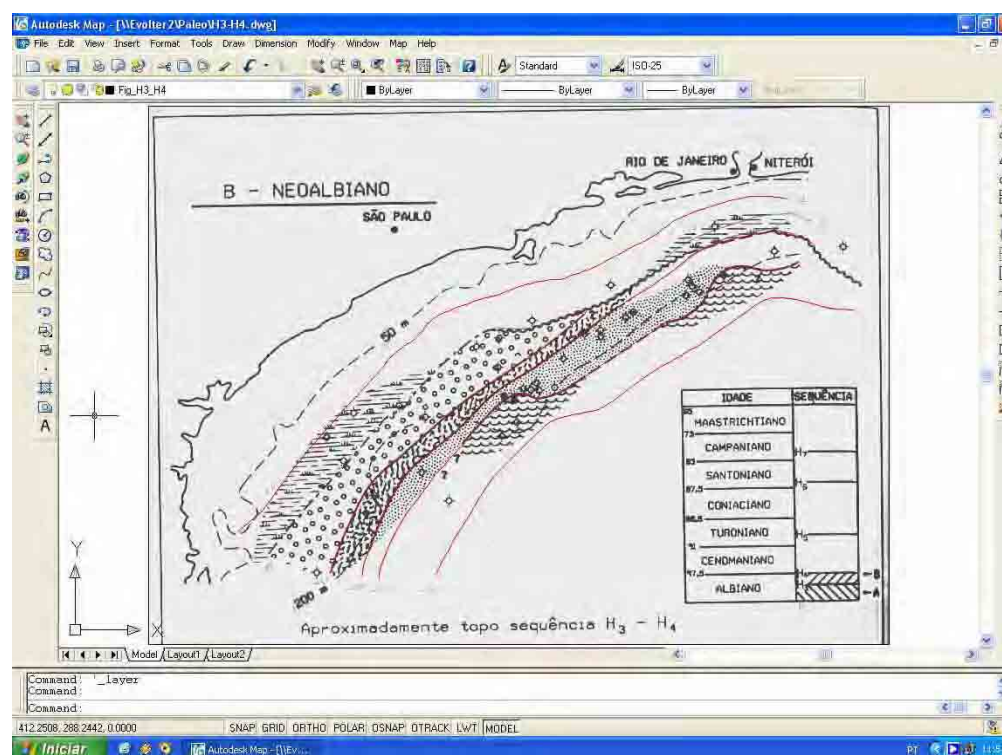


Figura 7.4 – Digitalização das curvas paleobatimétricas (em vermelho) e da localização dos poços. Bacia de Santos, situação ao “final” do neo-albiano.

7.2 Criação de Modelos Digitais de Terreno (MDT's)

Nesta etapa foram construídas malhas vazadas 3D (ou MDT's) para cada conjunto de curvas paleobatimétricas, utilizando o programa GOCAD.

Os arquivos DXF gerados em ambiente AutoDesk Map foram importados no GOCAD sob a forma de objetos do tipo *Curve* (conjuntos de linhas poligonais). A Figura 7.6 mostra um exemplo de curvas paleobatimétricas coloridas de acordo com uma escala de cores associada à variação batimétrica dos ambientes deposicionais considerados (do continental para o batial inferior, tendo-se como referência a Figura 7.5). Utilizou-se uma mesma escala do vermelho para o roxo para sombrear os modelos de todas as superfícies de modos a permitir uma melhor comparação entre elas.

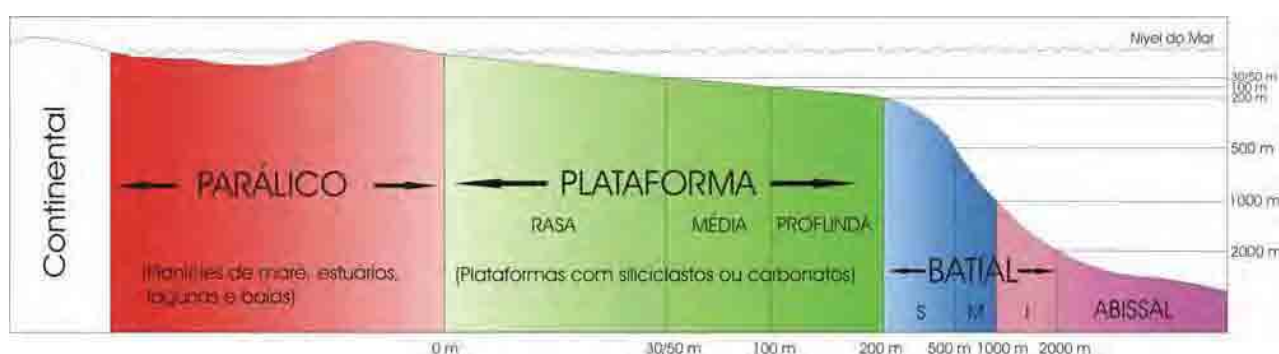


Figura 7.5 – Modelo esquemático dos ambientes deposicionais considerados para representar a variação paleobatimétrica nos modelos digitais construídos. As diferentes cores identificam ambientes distintos. Adaptado de Koutsoukos (1987).

Em seguida, a partir de cada conjunto de curvas, foram extraídos os pontos correspondentes (obtidos durante a digitalização dos mapas 2D). No ambiente GOCAD, estes pontos são entidades do tipo *PointSet*. Esta operação é essencial para a correta construção de malhas vazadas.

A partir das curvas de isoprotundidades, foi também construída uma nova curva, denominada “feixe convexo” (*Convex Hull*, Figura 7.7.). Tal curva é o menor conjunto de segmentos interconectados que contém as curvas paleobatimétricas. Sua utilização é importante para controlar a qualidade de malhas vazadas criadas a partir de pontos avulsos.

Utilizando o conjunto de pontos de cada idade e seu feixe convexo, foram construídas malhas vazadas 3D através do algoritmo de triangulação *Delaunay* (WATSON, 1981). Trata-se de um procedimento de construção de malhas de triângulos a partir de pontos *no plano*. Este algoritmo é a técnica mais bem sucedida para a confecção de modelos digitais de terrenos, pois minimiza os inconvenientes da criação de triângulos muito estreitos, característica indesejável na modelagem geométrica de superfícies. Para tanto, utiliza o

critério do *circuncentro*, segundo o qual três pontos do conjunto de entrada somente podem ser usados como vértices de um novo triângulo se o circuncentro deste não contiver outros pontos do conjunto (Figura 7.8). O algoritmo de triangulação *Delaunay* está implementado no programa GOCAD e para sua execução é suficiente criar uma nova superfície (objeto *Surface*) a partir de pontos e curvas. As curvas consistem no feixe convexo criado anteriormente. Sua utilização é necessária para prover a orientação aproximada de um plano sobre o qual os pontos serão projetados, reduzindo a dimensão (número de coordenadas) destes de 3 para 2. Tal exigência deriva da natureza bidimensional do algoritmo de triangulação. A Figura 7.9 mostra a malha vazada da paleobatimetria ao “final” do eo-mesoalbiano para a Bacia de Santos, obtida a partir das curvas exibidas na Figura 7.7.

Finalmente, a Figura 7.10 mostra a mesma superfície sombreada de acordo com a escala de cores construída na Figura 7.5. A utilização de cores fornece uma indicação visual imediata da variação batimétrica ao longo da superfície e dos ambientes de deposição registrados.

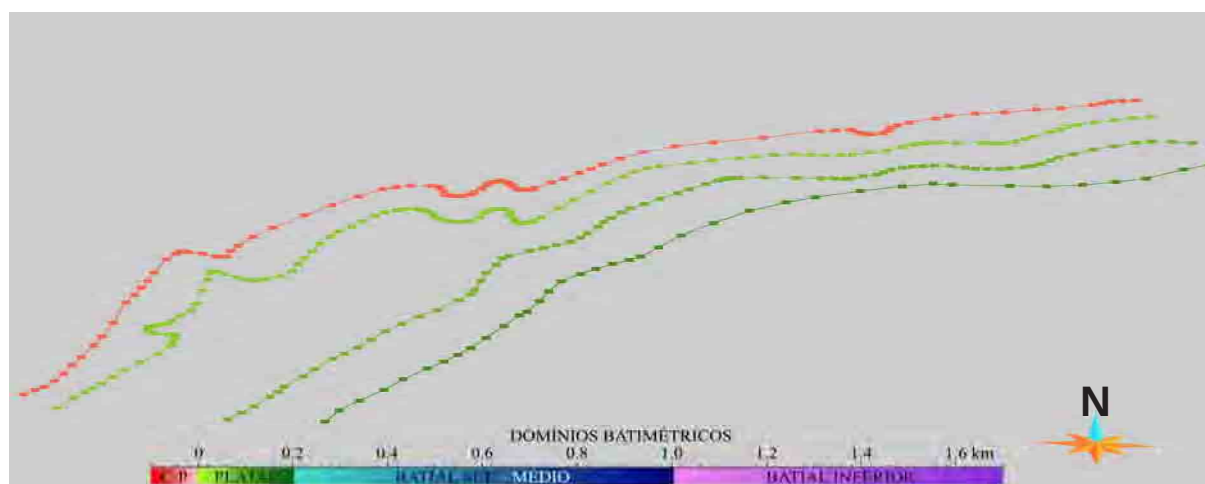


Figura 7.6 – Bacia de Santos: visualização das curvas paleobatimétricas ao “final” do eo-mesoalbiano no ambiente GOCAD. São visíveis os pontos digitalizados e uma escala de cores que acompanha a variação da batimetria. A sigla “C-P” denota o ambiente continental-parálico.

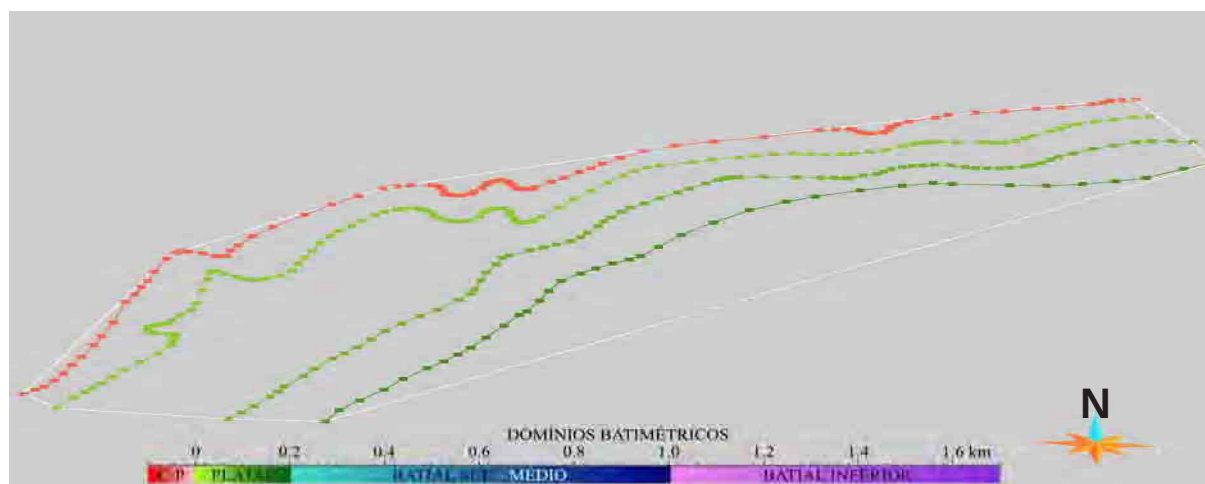


Figura 7.7 – Visualização do feixe convexo (linha branca) do conjunto de curvas da figura anterior.

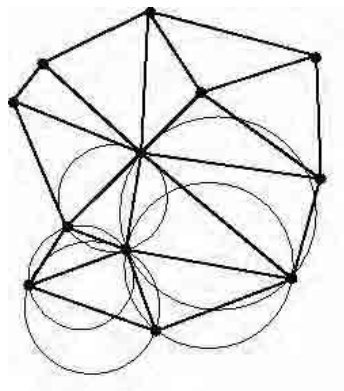


Fig. 7.8 – Triangulação *Delaunay* de um conjunto de pontos no plano. O circuncentro de cada triângulo gerado pode conter apenas os três vértices deste.

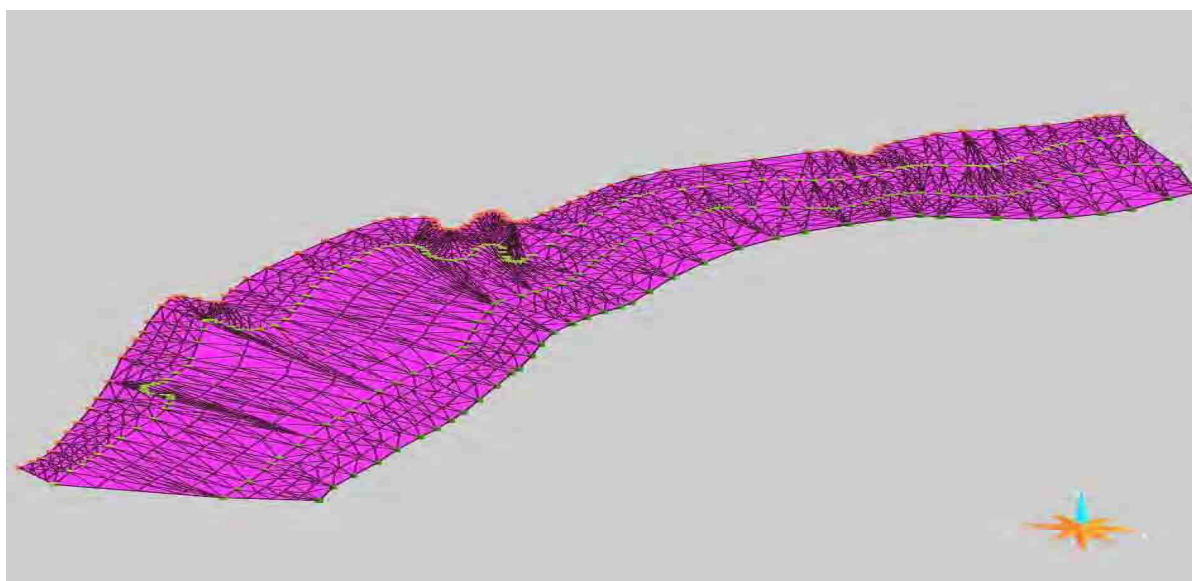


Figura 7.9 – Construção de uma malha vazada de triângulos (superfície paleobatimétrica) a partir dos pontos e do feixe convexo da Figura 7.7. O fator de exagero vertical é 20.

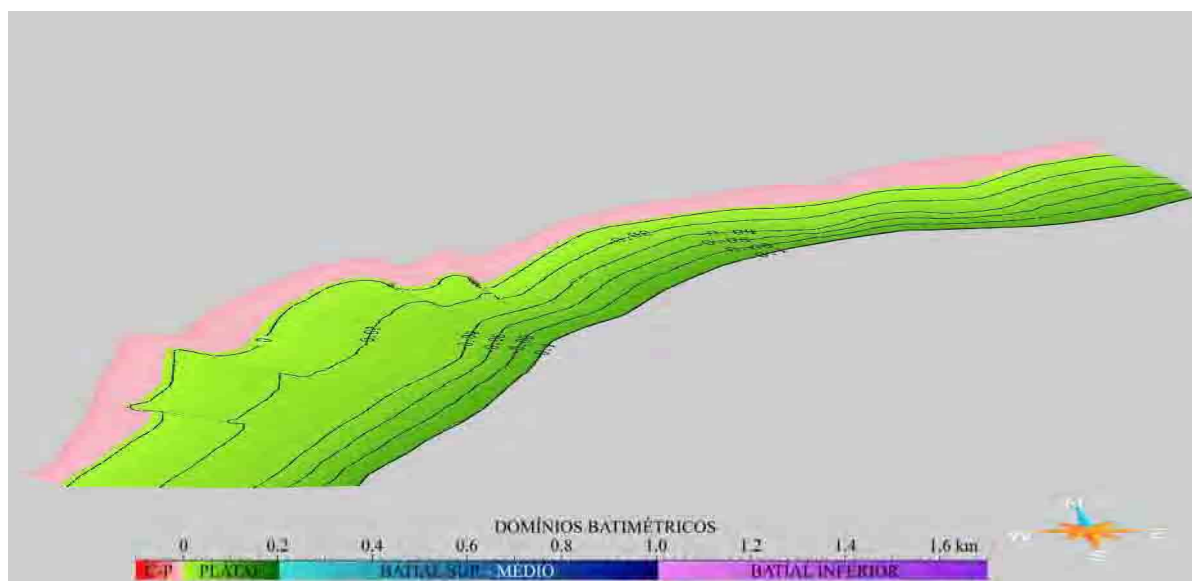


Figura 7.10 – Sombreamento da superfície paleobatimétrica do “final” do eo-mesoalbian (Bacia de Santos), obtido usando uma escala de cores que mapeia as variações de batimetria dos ambientes deposicionais. A coloração aplicada a esta superfície destaca um ambiente parálico e plataformal raso-médio. Espaçamento vertical das isolinhas pretas: 200 m.

7.3 Modelagem de dados complementares

Visando aumentar o potencial de interpretação associado às superfícies paleobatimétricas criadas, foram utilizadas informações complementares para modelar dados auxiliares da plataforma continental do Sudeste do Brasil. Segue uma descrição das principais entidades modeladas:

Poços. Durante a etapa de digitalização dos mapas paleobatimétricos 2D, foi também executada a digitalização e georreferenciamento dos poços utilizados para a confecção dos referidos mapas. O georreferenciamento foi realizado escolhendo-se, respectivamente, 4 e 7 poços para as bacias de Campos e Santos; e obtendo suas coordenadas a partir do Banco de Dados de Exploração e Produção, disponível na Internet (BDEP, 2004).

A finalidade desta modelagem consiste na visualização da localização dos poços indiretamente utilizados para a criação das superfícies paleobatimétricas. As Figuras 7.3 e 7.4 mostram parte do conjunto de poços originais exibidos no mapa paleobatimétrico 2D da Bacia de Santos para o “final” do neo-albiano.

As informações de localização dos poços foram exportadas, no ambiente AutoDesk Map, para o formato ASCII, criando-se um arquivo para cada bacia. Cada linha deste arquivo contém uma informação de nome e localização do poço correspondente (coordenadas X, Y e Z). A cota das plataformas correspondentes aos poços ilustrados foi considerada igual a zero, já que as sondas encontram-se na superfície do mar.

Os arquivos ASCII foram importados no programa GOCAD, criando-se uma entidade do tipo *Well* para cada poço. A entidade correspondente é representada por um símbolo de poço amarelo, a partir do qual é traçada uma trajetória vertical retilínea em vermelho até a profundidade padrão de 4000 m. Tal profundidade representa uma convenção utilizada para coerência de visualização. Para facilitar o gerenciamento múltiplo das propriedades de visualização dos poços de cada bacia, estes foram reunidos em estruturas de manipulação denominadas *groups*. Desta forma, foi criado um grupo para cada bacia. As Figuras 7.11 e 7.12 mostram a localização dos poços nas bacias de Santos e Campos, respectivamente, com as superfícies paleobatimétricas do “final” do mesoalbiano.

Linha de costa atual. Esta informação foi acrescentada para facilitar a localização das superfícies paleobatimétricas em relação ao relevo continental atual e demais estruturas modeladas. Serve também para destacar eventos transgressivos / regressivos ocorridos durante o período de estudo. Foi obtida do CPRM (2000) e importada no GOCAD como uma entidade *Curve*. As Figuras 7.11 e 7.12 contêm exemplos de visualização da linha de costa. A Figura 7.13 mostra, entre outras entidades, a linha de costa atual com as superfícies

paleobatimétricas das duas figuras anteriores. É importante destacar que tal figura constitui o primeiro exemplo de visualização simultânea dos dados de paleobatimetria referentes às duas bacias, em virtude da possibilidade de exibir várias entidades distintas ao mesmo tempo no programa GOCAD.

Topografia atual. Trata-se do MDT do relevo continental atual do Sudeste do Brasil, obtido através do projeto GTOPO30 (2004). A superfície topográfica original foi “recortada” ao longo de uma linha poligonal fechada que delimita a área de estudo (Figura 7.1). Este recorte permitiu reduzir a complexidade do modelo digital através da remoção de partes do relevo topográfico não relevantes para a visualização das informações paleobatimétricas das bacias de Santos e Campos. A topografia foi sombreada usando uma escala de cores do azul para o branco, acompanhando a variação altimétrica (Figura 7.13).

Hidrografia atual. A hidrografia associada ao relevo continental complementa as informações fisiográficas. As curvas de hidrografia foram obtidas a partir de dados do CPRM integrados com informações da Agência Nacional de Águas (ANA, 2004). Um exemplo é mostrado na Figura 7.13. A rede de drenagens modelada cobria uma região mais extensa que a exibida nas figuras anteriores. Sua adaptação envolveu o recorte, em ambiente GOCAD, das regiões correspondentes à porção de topografia removida.

Linha de charneira das bacias de Santos e Campos. Esta informação é representada por uma linha poligonal azul (Figura 7.13). A linha de charneira foi digitalizada a partir de um mapa de estruturas oceânicas e continentais (SHOBBENHAUS & CAMPOS, 1984 apud CAINELLI & MOHRIAK, 1998) que será mostrado mais adiante. O processo foi executado diretamente no ambiente GOCAD, mas antes foi necessário georreferenciar a imagem com o programa ER Mapper (EARTH RESOURCES MAPPER, 2005).

Batimetria atual. O modelo digital da batimetria atual para as regiões das bacias de Santos e Campos foi obtido a partir do projeto TOPEX (2004). O conjunto de pontos estava no sistema de coordenadas LAT-LONG e foi convertido para o sistema UTM/23 SUL. A partir dos pontos foi criada uma malha vazada e a superfície foi sombreada utilizando uma escala de cores do azul para o vermelho (Figura 7.14). Para realçar a variação morfológica ao longo do modelo, foram também traçadas isolinhas com espaçamento vertical de 100 m.

Plano do nível do mar. Esta entidade foi obtida criando, no programa GOCAD, uma superfície plana (quadrilátero). Em seguida, o objeto foi recortado usando a curva da linha de costa atual e a curva que delimita a área de estudo. A superfície coincide com o plano XY do sistema global de coordenadas, possuindo elevação igual a zero. O plano recebeu um índice de transparência de 60% para revelar as estruturas subjacentes (Figura 7.15).

Limite entre a crosta oceânica e continental. Trata-se de uma curva digitalizada a partir de um mapa topográfico / batimétrico do trabalho de Paula & Mohriak (2005). O limite entre crostas oceânica e continental utilizado pelos autores inclui a região das bacias de Campos e Santos (Figura 7.16).

Curvas de evolução da linha de costa. A informação indicativa das linhas de costa associadas às várias idades estudadas foi determinada, para a bacia de Santos, considerando-se a curva de transição entre os ambientes continental e parálico (isto é, a curva de elevação igual a zero) nas várias superfícies paleobatimétricas. Foram criadas seis linhas, uma para cada superfície. Nos mapas paleobatimétricos originais, o limite continental / parálico foi identificado apenas nas últimas duas idades consideradas, sendo que para as demais adotou-se o limite continental / plataformal (Figura 7.2). As curvas obtidas são entidades interessantes implicitamente contidas nos conjuntos de curvas paleobatimétricas digitalizadas, e que podem ser usadas para efeito de comparação com a linha de costa atual. A Figura 7.17 mostra um exemplo de linha de costa indicativa para o “final” do eo-mesoalbiano. A sequência de linhas geradas para o inteiro período do Albiano ao Maastrichtiano será mostrada na próxima seção.

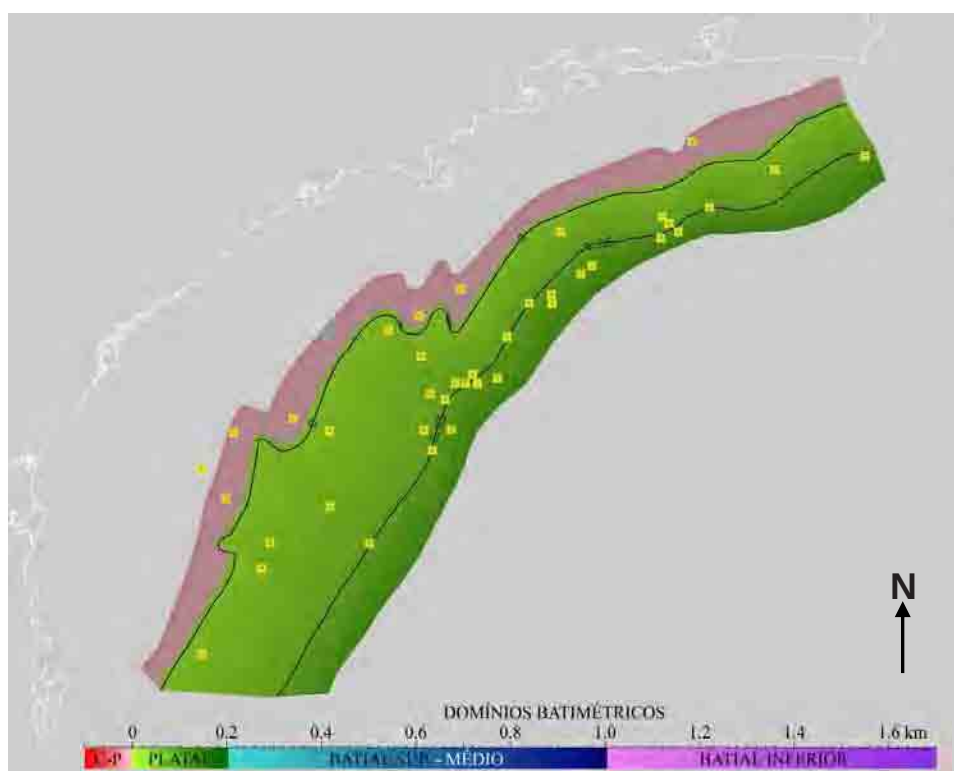


Figura 7.11 – Visualização dos poços utilizados para a confecção dos mapas paleobatimétricos da Bacia de Santos. São também representadas a superfície batimétrica ao “final” do mesoalbiano e a linha de costa atual (em branco). Isolinhas pretas traçadas a intervalos de 50 m. O fator de exagero vertical é 20. Vista em planta.

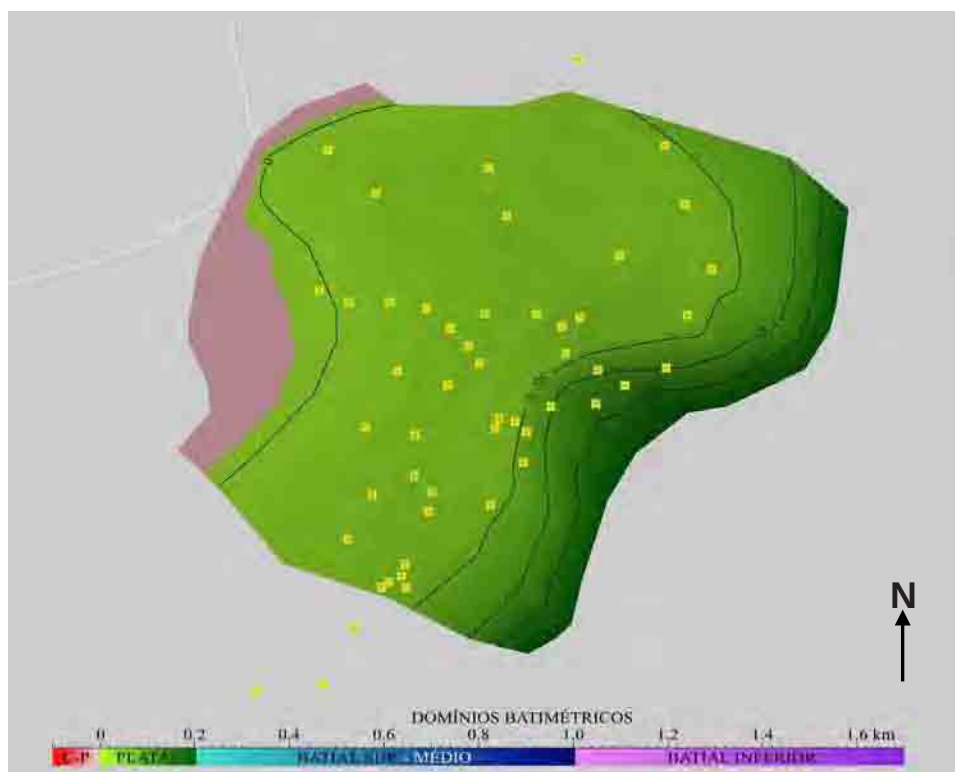


Figura 7.12 – Visualização dos poços e superfície batimétrica ao “final” do mesoalbio para a Bacia de Campos. A noroeste, um detalhe da linha de costa. Isolinhas pretas traçadas a intervalos de 50 m. O fator de exagero vertical é também 20. Vista em planta.

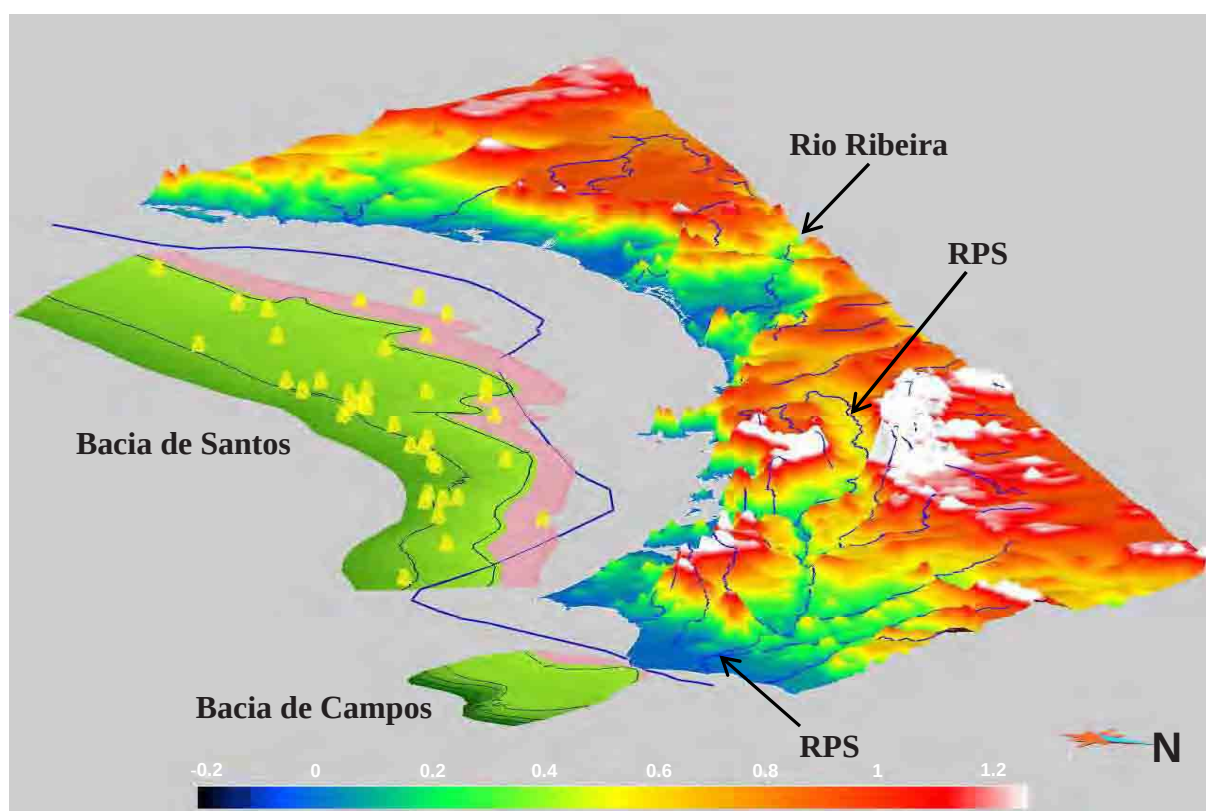


Figura 7.13 – Visualização simultânea das superfícies paleobatimétricas das bacias de Santos e Campos ao “final” do mesoalbio (103 Ma). São também exibidos o relevo topográfico atual, a rede de drenagens, a linha de costa atual (em branco), a linha de charneira das bacias (em azul) e os poços. A escala de cores acompanha a variação de altitude da topografia, passando pelas cores azul (nível do mar), verde (200 m), amarela (600 m), vermelha (1000 m) e branca (1500 m). RPS = Rio Paraíba do Sul. O fator de exagero vertical é 20 para todas as entidades. Visada para sudoeste.

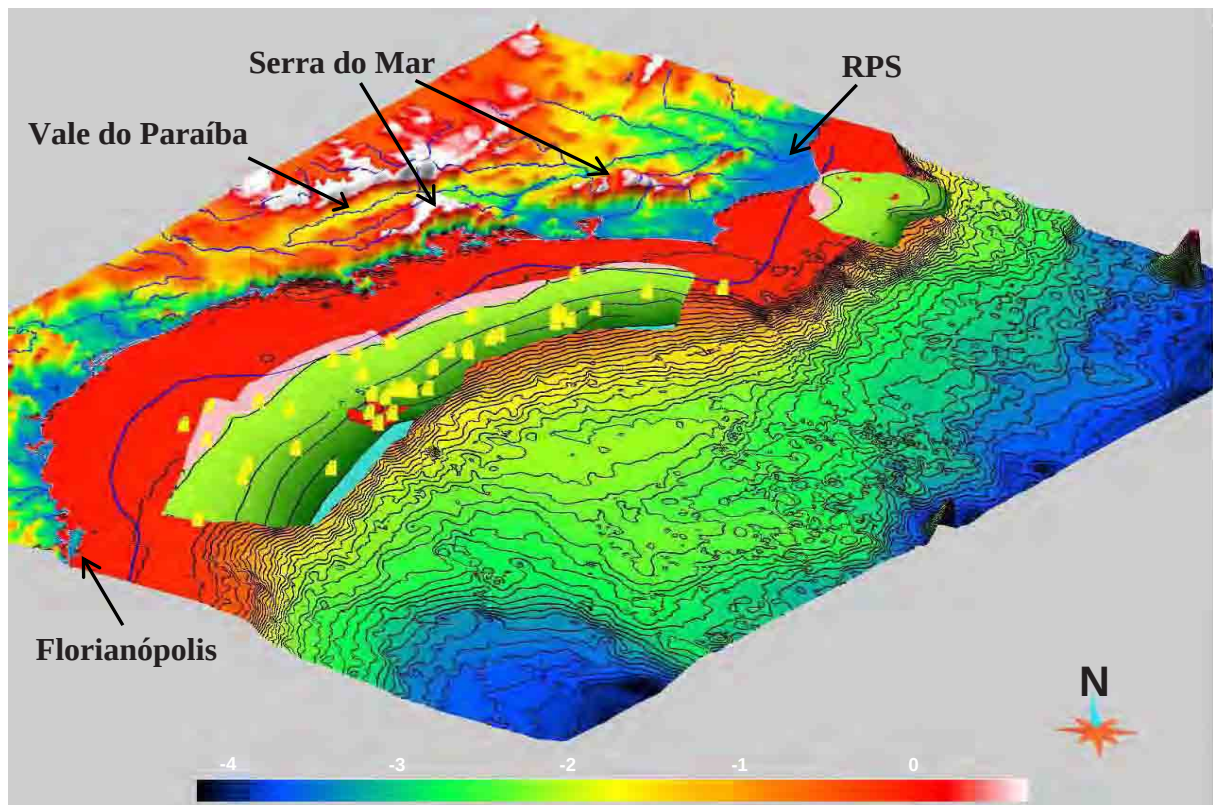


Figura 7.14 – Batimetria atual acrescentada ao conjunto de entidades da Figura anterior. A superfície foi sombreada usando uma escala de cores que passa pelo azul escuro (4000 m de profundidade), verde escuro (3000 m), verde claro (2000 m), laranja (1000 m) e vermelho (nível do mar). O espectro de cores acompanha a variação batimétrica. Fator de exagero vertical: 20. Visada para Norte.

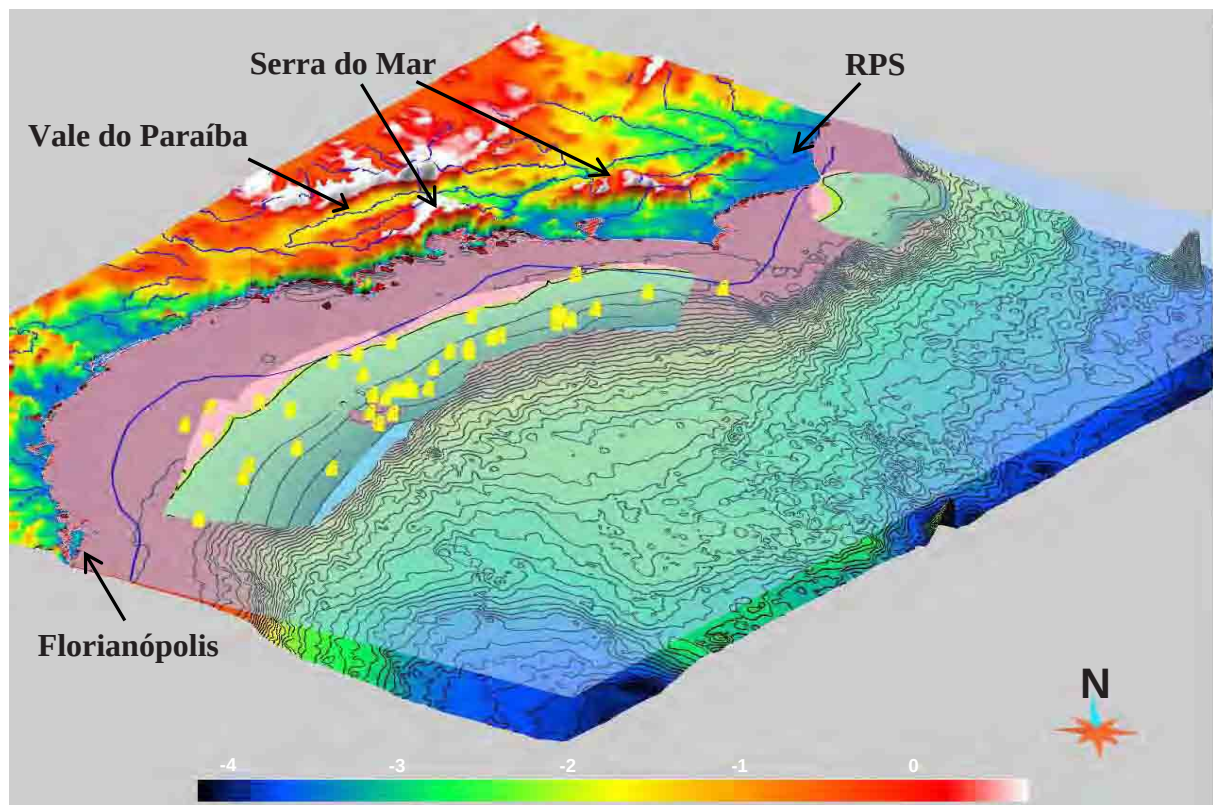


Figura 7.15 – Adição de plano semitransparente para indicar o atual nível do mar. A escala de cores exibida refere-se à batimetria atual. Visada para Norte.

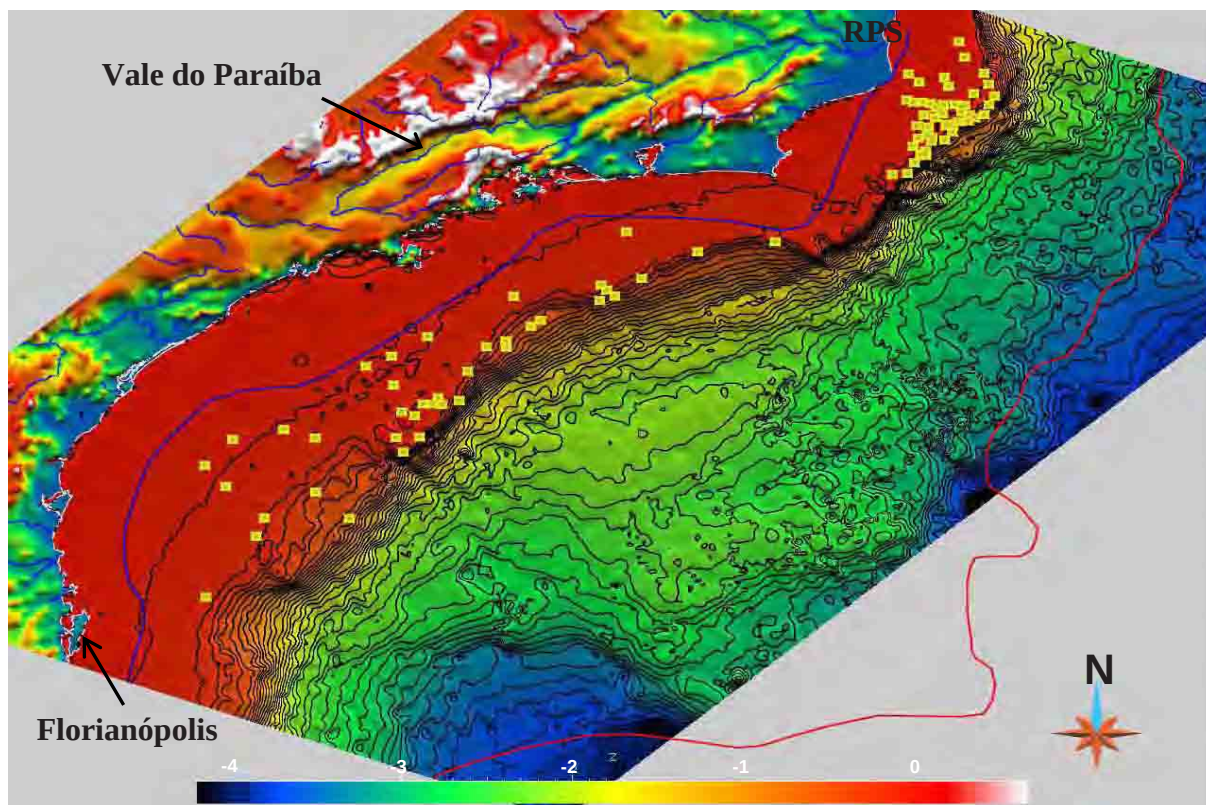


Figura 7.16 – Limite entre a crosta oceânica e continental indicado pela linha vermelha (região inferior). Em azul, a linha de charneira das bacias de Santos e Campos. Vista em planta.

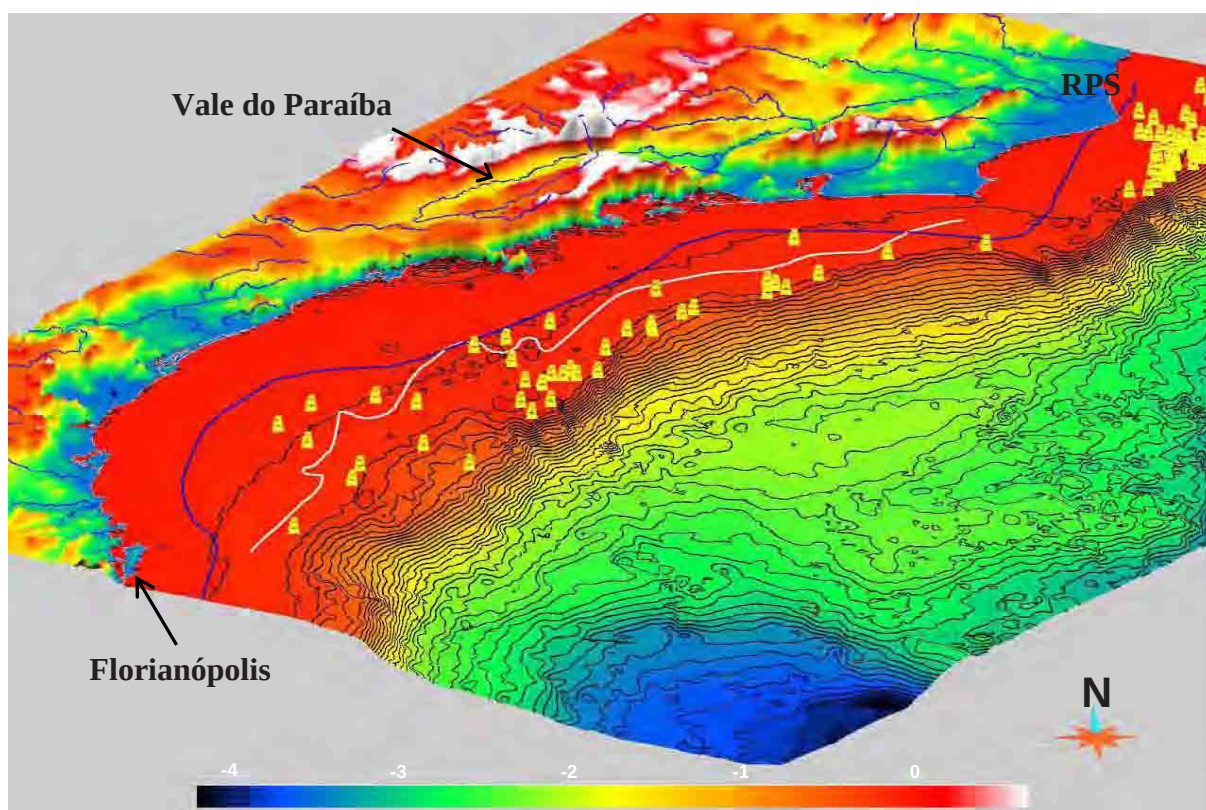


Figura 7.17 – Bacia de Santos: visualização (em branco) da linha de costa ao “final” do mesoalbio. Esta linha foi obtida considerando-se o início do ambiente plataformal na superfície paleobatimétrica correspondente. O fator de exagero vertical é 20. Visada para Norte.

7.4 Visualização 3D dos modelos digitais obtidos

Como foi mostrado acima, as superfícies paleobatimétricas modeladas foram visualizadas em 3D com o programa GOCAD (Figuras 7.9 e seguintes). Neste ambiente, técnicas de visualização científica permitiram enriquecer o conjunto de informações exibidas. Destaca-se a utilização de uma escala de cores associada à variação batimétrica de cada superfície, com limites predefinidos, para proporcionar uma informação visual imediata de sua elevação (Figura 7.10).

Outra técnica amplamente usada em representações 2D é a visualização de isolinhas associadas a níveis de batimetria predefinidos. Esta ferramenta foi também utilizada ao longo da visualização 3D. As isolinhas foram traçadas sobre as superfícies paleobatimétricas.

A evolução morfológica da Bacia de Santos do Albiano ao Maastrichtiano é mostrada na Figura 7.18. As superfícies paleobatimétricas foram coloridas usando a escala de cores ilustrada anteriormente, visando fornecer uma indicação intuitiva da variação batimétrica dentro de cada ambiente deposicional. Sobre as superfícies foram também traçadas isolinhas de cor preta, com espaçamento vertical de 50 metros. Dentre as entidades complementares, foram visualizadas a topografia, hidrografia e a linha de costa atuais, bem como a linha de charneira da bacia e a localização dos poços utilizados na preparação dos mapas 2D originais (VIVIERS, 1986). Todas as entidades mencionadas foram exibidas simultaneamente usando uma única janela 3D de visualização.

De maneira análoga, a Figura 7.19 mostra a evolução da batimetria da Bacia de Campos para o mesmo período. Todas as superfícies foram sombreadas usando a mesma escala de cores adotada para a Bacia de Santos. Além disto, sobre cada modelo foram traçadas isolinhas com intervalos de 50 metros (primeira, segunda e terceira superfícies) e de 100 metros (demais superfícies). A Figura exhibe também um detalhe da topografia local e a localização dos poços utilizados para confeccionar os mapas 2D originais (AZEVEDO et al., 1987).

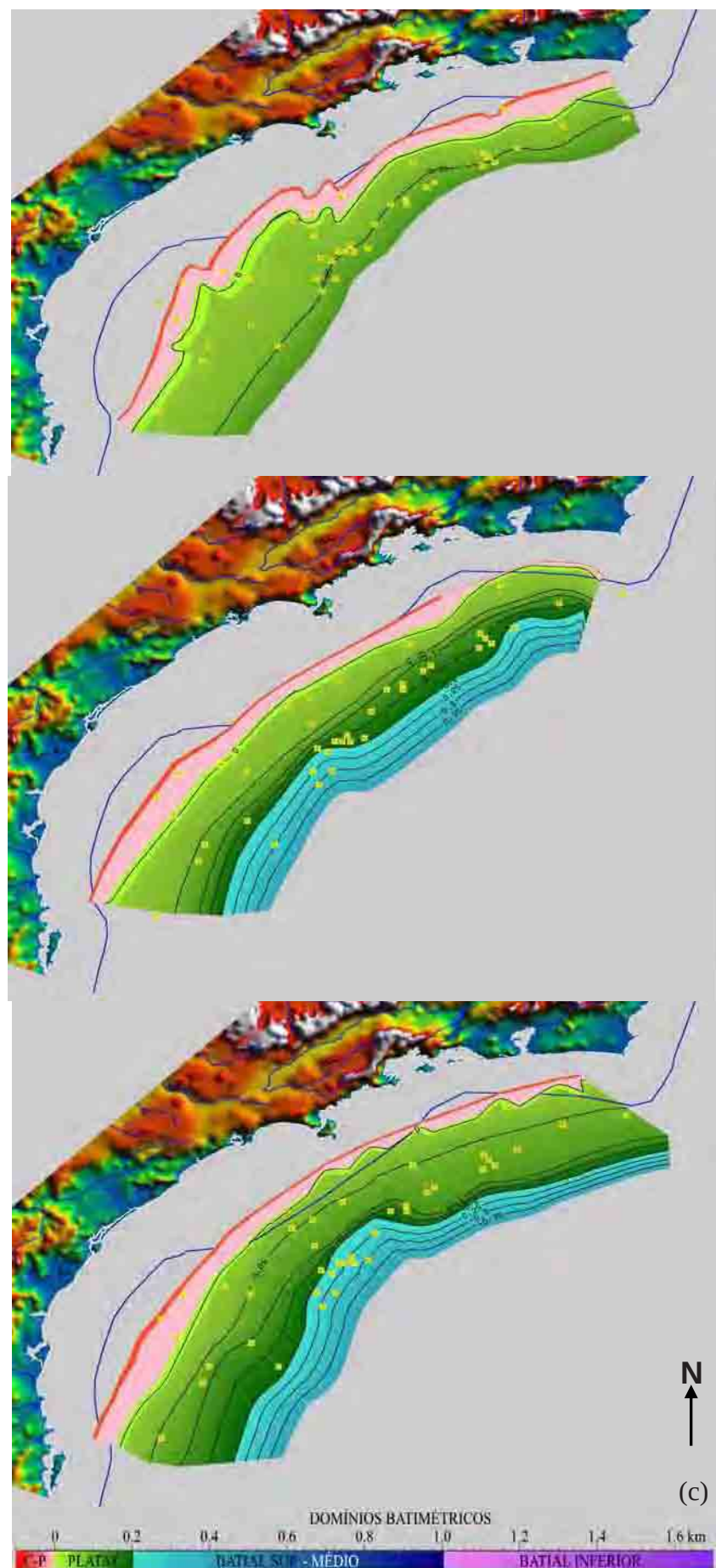


Figura 7.18a-c – Bacia de Santos: superfícies batimétricas ao “final” do eo-mesoalbio (a) (103 Ma), neo-albio (b) (97,5 Ma) e mesoturoniano (c) (89 Ma). São também representadas a topografia, hidrografia e linha de costa atuais, os poços e a linha de charneira (em azul). Neste período, observa-se o gradual aumento da profundidade, com a deposição em ambiente plataformar até batial superior. Fator de exagero vertical: 20. Vista em planta.

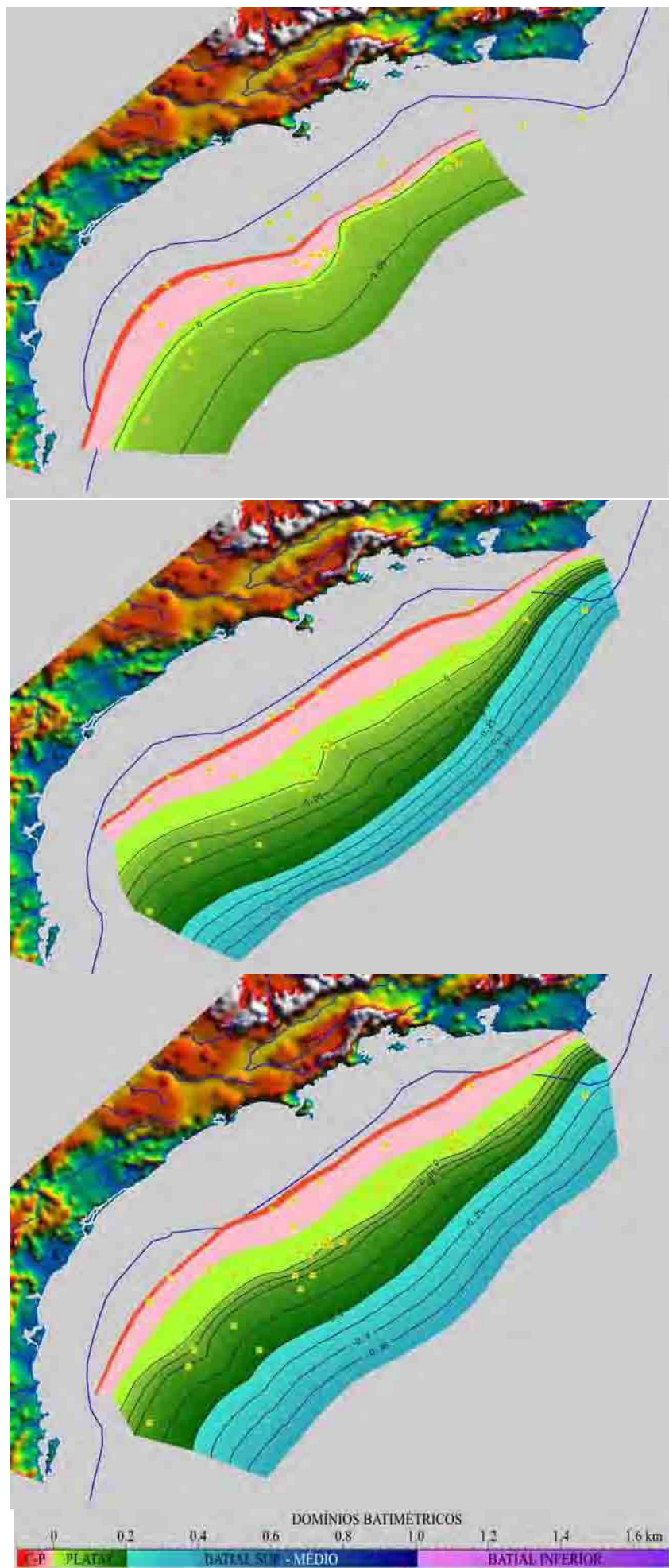


Figura 7.18d-f – Bacia de Santos: superfícies batimétricas ao “final” do eosantoniano (d) (86 Ma), eocampaniano (e) (81 Ma) e Maastrichtiano (f) (65 Ma). Uma fase progradante, que caracterizou ambientes de deposição mais rasos (até plataformar médio), foi seguida por novo estabelecimento de condições mais profundas (plataformar profundo até batial superior). Houve também expansão do ambiente parálico na região central da bacia.

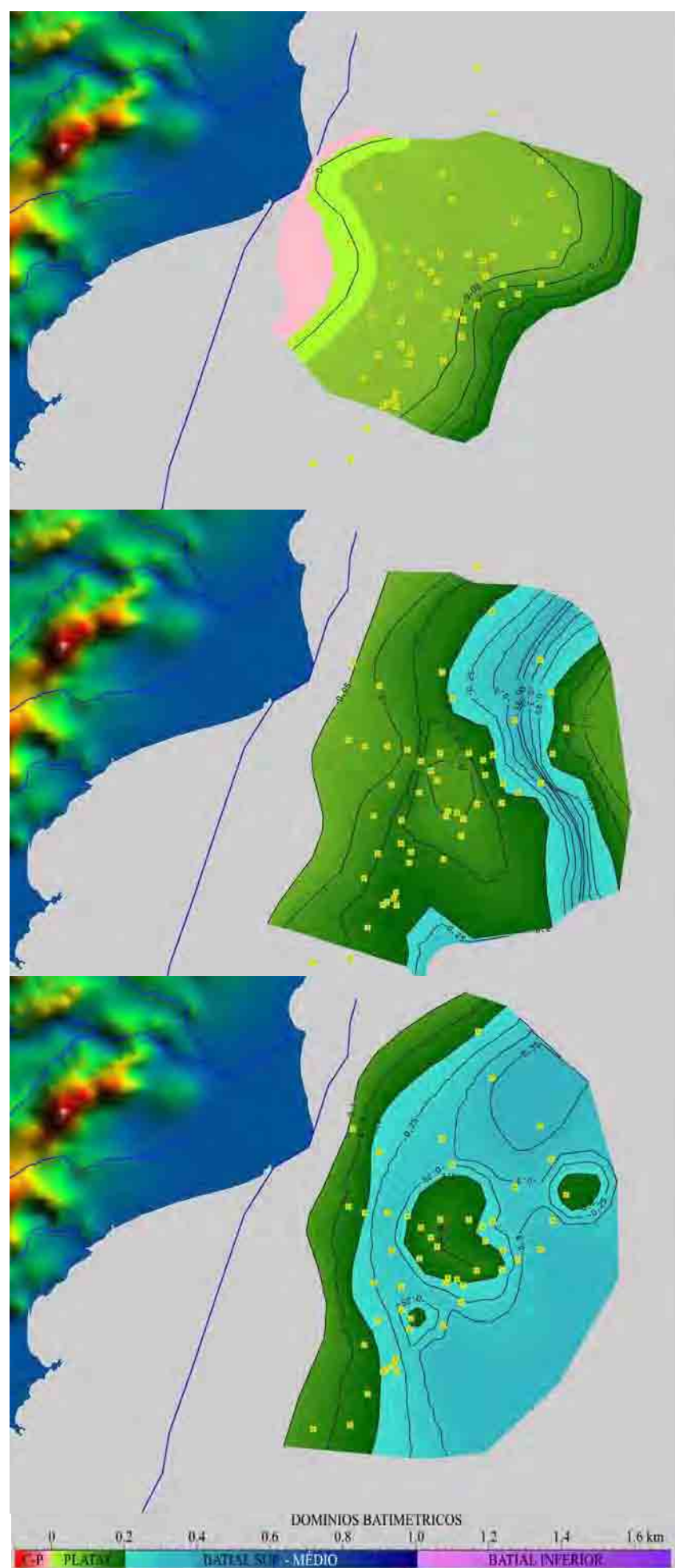


Figura 7.19a-c – Bacia de Campos: superfícies batimétricas ao “final” do eo-mesoalbio (a) (103 Ma), neo-albiano (b) (97,5 Ma) e eoturoniano (c) (90 Ma). Como na Bacia de Santos, ocorreu progressivo aumento da profundidade, com ambiente plataformar médio-profundo, de inclinação crescente, e batial superior. Este ambiente estava bem mais difundido que na Bacia de Santos (Figura 7.18c). RPS = Rio Paraíba do Sul.

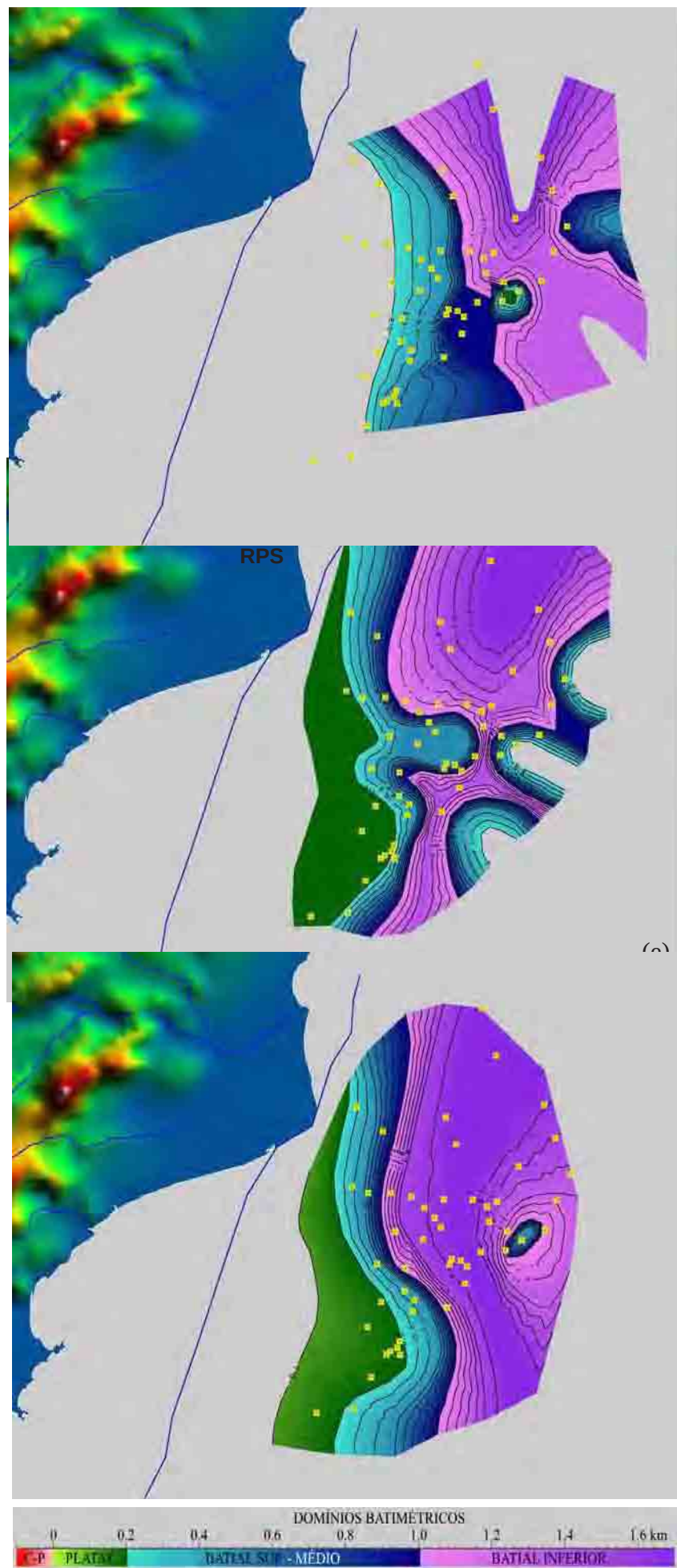


Figura 7.19d-f – Bacia de Campos: superfícies batimétricas ao “final” do eoconiaciano (d) (88 Ma), Santoniano (e) (83 Ma) e Maastrichtiano (f) (65 Ma). Neste período observa-se o domínio do ambiente batial (superior a inferior) sobre a região, ocorrendo crescente aumento da profundidade.

Ambientes de visualização 3D de dados como o programa GOCAD proporcionam outras funcionalidades extremamente poderosas. Além dos recursos já mencionados, é possível destacar os seguintes:

Modificação interativa do ponto de observação. Tal característica é consequência do caráter tridimensional da visualização de dados. As entidades são exibidas em uma janela 3D, que é o resultado da “captura” de uma cena por uma câmera virtual. O ponto de observação do usuário representa a câmera, sendo possível modificar a direção de observação bem como a distância dos objetos visualizados, fator de exagero vertical, etc. Para ilustrar um exemplo, compare-se as Figuras 7.19 e 7.20. A primeira mostra uma vista em planta das superfícies paleobatimétricas da Bacia de Campos. Tal direção de observação representa um caso “particular” de visualização em um ambiente 3D. A Figura 7.20 mostra as mesmas entidades geométricas utilizando uma direção de observação e fator de exagero vertical diferentes. Além disto, a tonalidade das cores aplicadas às superfícies foi ajustada de forma a considerar a posição das mesmas em relação a uma fonte de luz virtual, melhorando a noção de profundidade das entidades. A representação 3D utilizada permite identificar com maior facilidade as variações morfológicas da bacia ao longo do tempo: ocorrência de ambiente plataformal raso durante o eo-mesoalbianiano, expansão do meio plataformal médio-profundo e batial superior até o eoturoniano, profundidades crescentes nos períodos seguintes, com a consolidação do meio batial superior a inferior. É também possível observar a formação de altos na porção leste da bacia (registrados ao “final” do Santoniano) e de um talude relativamente íngreme durante o Maastrichtiano.

A modificação interativa do ponto de observação do usuário é um recurso fundamental para se manter a impressão visual 3D das superfícies estudadas. Com efeito, o conteúdo de uma janela de visualização 3D é apenas uma imagem exibida em uma tela bidimensional (superfície do monitor). Para se manter a percepção de profundidade, é necessário rotacionar continuamente os modelos visualizados. Tal percepção é facilitada em computação gráfica através de recursos como a utilização de uma escala de cores, a visualização da malha vazada dos objetos (conjunto de arestas), o traçado de curvas de isopropfundidades, a projeção geométrica em perspectiva, a oclusão de parte de objetos por entidades mais próximas do observador, etc. (FOLEY et al., 1996). Uma técnica mais realista consiste na visualização de uma imagem composta a partir de duas imagens obtidas de pontos de observação ligeiramente diferentes, simulando a composição das imagens dos olhos esquerdo e direito realizada pelo cérebro dos seres humanos (visão estereoscópica, SCHROEDER et al., 1996). A utilização de

um equipamento especial de realidade virtual (óculos estereoscópicos) permite separar as duas imagens compostas, para “enganar” o cérebro, expondo cada olho a uma imagem diferente. O programa GOCAD suporta tal equipamento e visualizações de entidades geológicas complexas foram realizadas com êxito no Laboratório de GeoModelagem 3D do DPM / IGCE / UNESP.

Visualização combinada de modelos distintos. Tal funcionalidade consiste na visualização simultânea de vários modelos na mesma cena 3D. Isto permite aumentar o potencial de interpretação dos dados investigados, já que características de entidades distintas podem ser exibidas na mesma cena para realçar relações de semelhanças / diferenças, entre outras. Exemplos de visualização combinada de modelos distintos (superfícies paleobatimétricas e modelos auxiliares) foram mostrados nas Figuras 7.11 e seguintes. Outro exemplo é a exibição simultânea da evolução paleoambiental das bacias de Santos e Campos (Figura 7.21). Tal recurso permite identificar padrões de evolução comuns às duas bacias. Esta integração de dados é uma característica importante dos ambientes de modelagem e visualização 3D. Outros exemplos de integração serão mostrados mais adiante.

Produção de animações. Consiste na possibilidade de gerar seqüências de imagens referentes a idades geológicas distintas, com a finalidade de prover uma ferramenta de estudo da evolução das entidades investigadas ao longo de intervalos temporais predefinidos. As Figuras 7.18 a 7.21, que mostram a evolução batimétrica das bacias de Santos e Campos do Albiano ao Maastrichtiano, constituem um exemplo de animação.

Utilizando os dados paleobatimétricos, é possível construir outras animações interessantes. Por exemplo, considere-se, em cada superfície da Bacia de Santos, a curva de nível correspondente à transição entre os ambientes continental e parálico (Figura 7.5). Tal curva faz parte do conjunto de curvas de isopropfundidades digitalizadas nos mapas 2D de Viviers (1986) e Azevedo et al. (1987), e pode ser indicativa do nível do mar nas idades consideradas. Visualizando separadamente cada curva e ordenando as imagens obtidas de acordo com uma seqüência temporal, é possível produzir uma animação da evolução da linha de costa durante o meso-Neocretáceo. (Figura 7.22). A seqüência de imagens constitui uma ferramenta de auxílio para a caracterização de eventos transgressivos / regressivos ocorridos durante o período estudado.

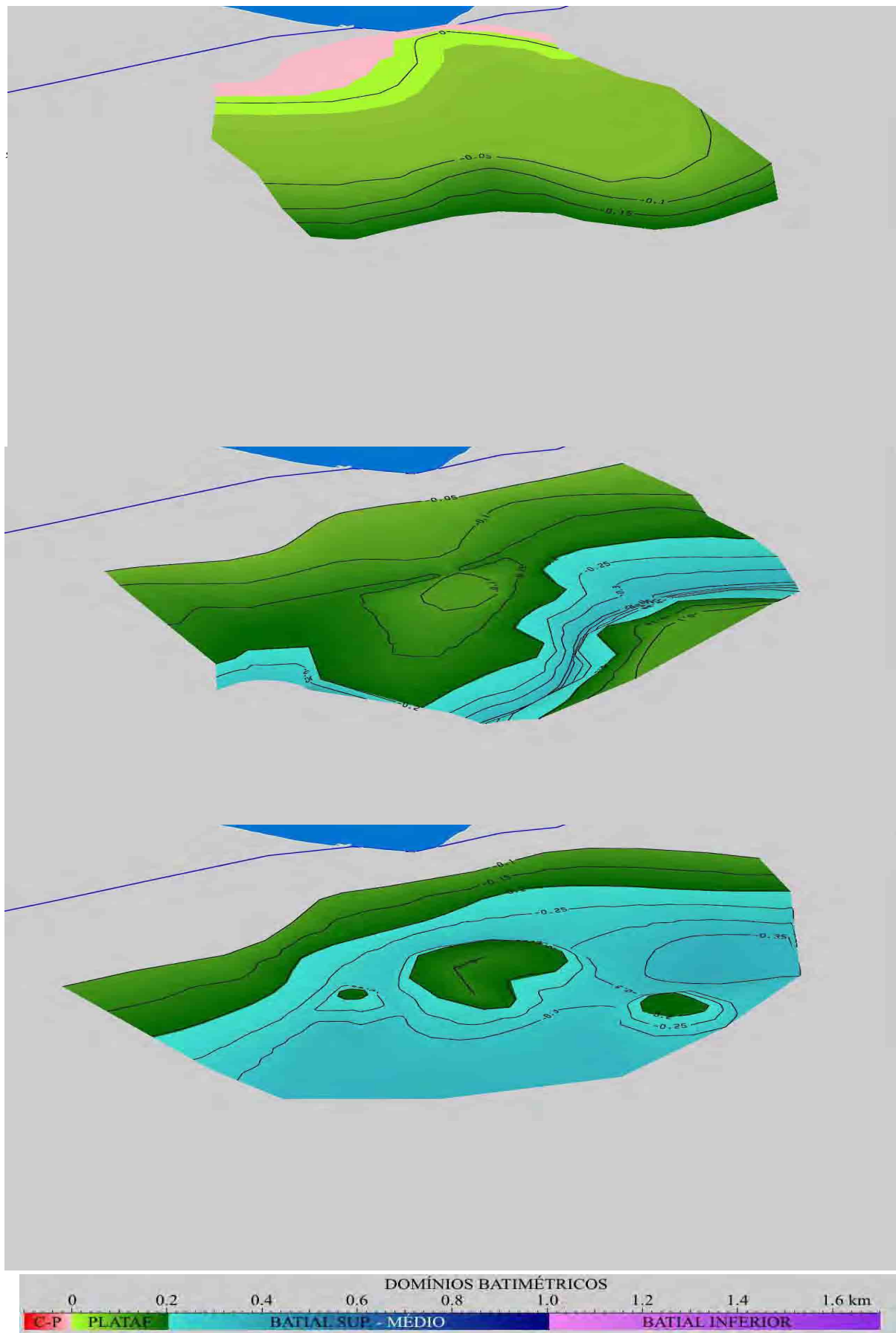


Figura 7.20a-c – Bacia de Campos: superfícies batimétricas ao “final” do eo-mesoalbio (a), neo-albio (b) e eoturoniano (c). O fator de exagero vertical é 50. Direção de observação de SE para NW. As entidades são agora visualizadas levando em conta seu caráter tridimensional. Verifica-se a evolução de um ambiente plataformar de baixo gradiente para o meio batial superior, com regiões de subsidência mais rápida.

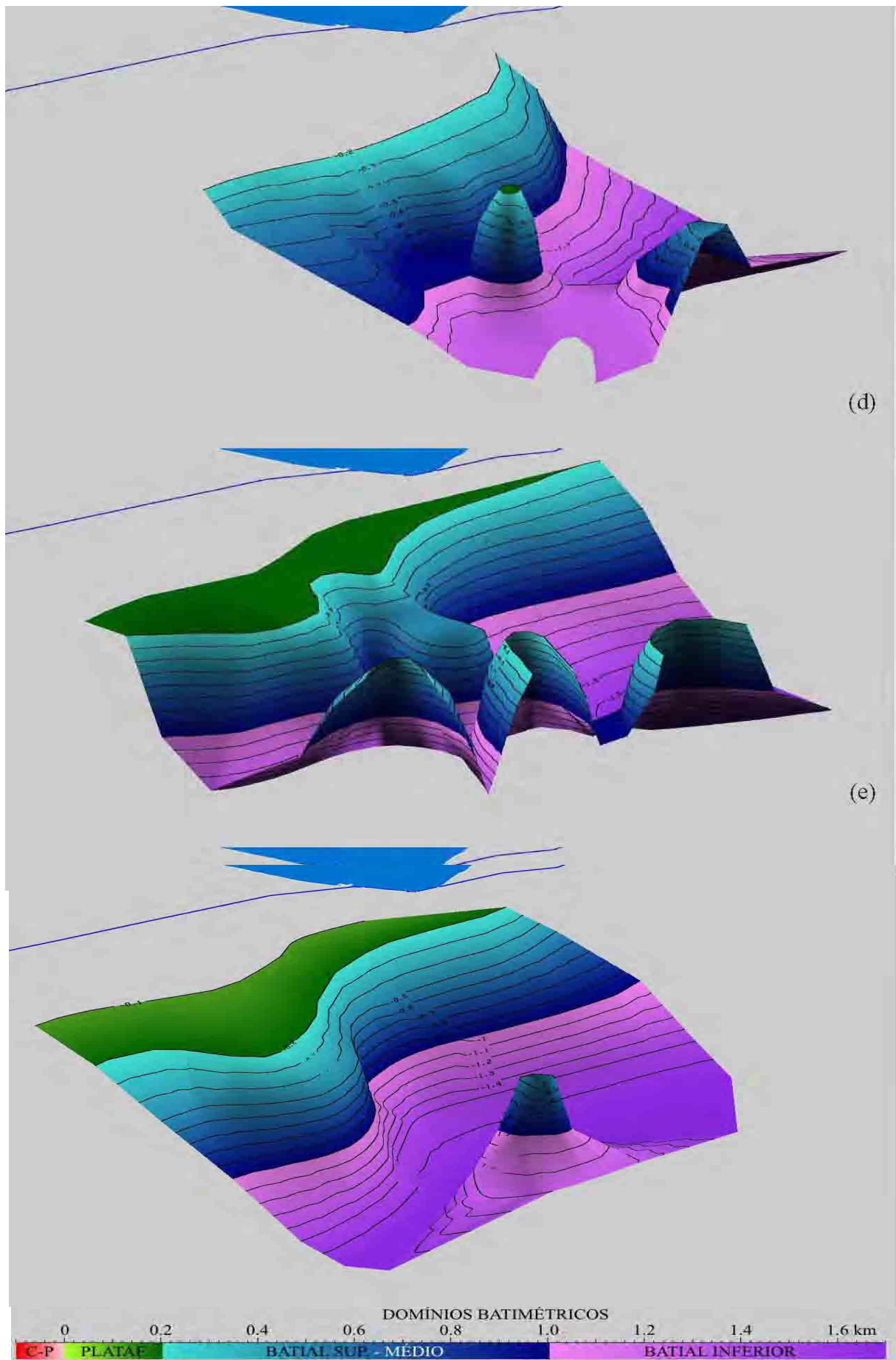


Figura 7.20d-f - Bacia de Campos: superfícies batimétricas ao “final” do eoconiaciano (d), Santoniano (e) e Maastrichtiano (f). Esta direção de observação permite destacar as variações morfológicas ocorridas em relação ao período Albiano-Eoturoniano.

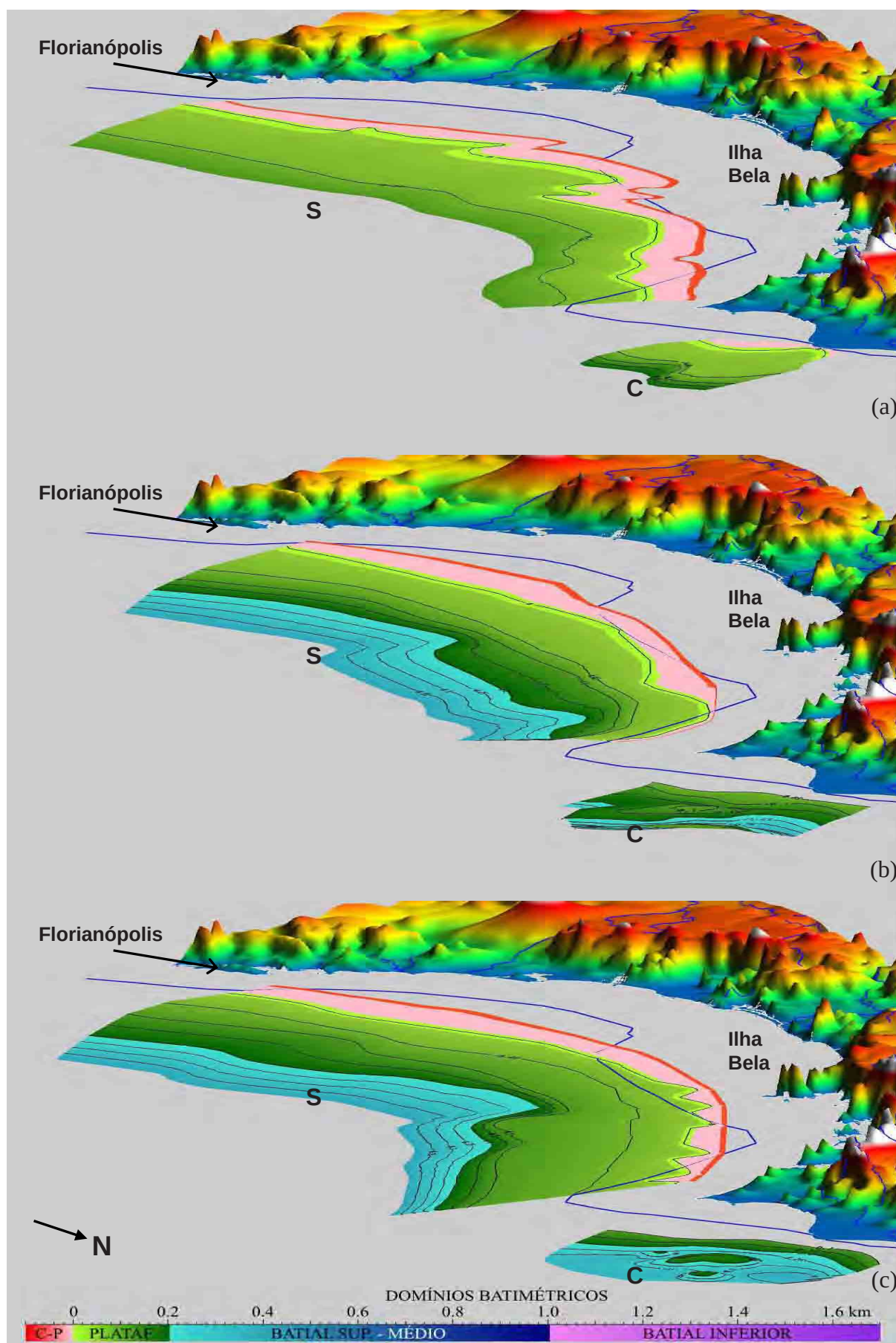


Figura 7.21a-c – Visualização simultânea da evolução paleobatimétrica das bacias de Santos (S) e Campos (C), mostrando um padrão de evolução comum que atinge o meio batial superior. Idades consideradas: (a) “final” do eo-mesoalbio, (b) neo-albiano, (c) eoturoniano (Bacia de Campos) / mesoturoniano (Bacia de Santos). Posição temporal dos mapas: vide Figura 7.2. Fator de exagero vertical: 20. Direção de observação de NE para SW.

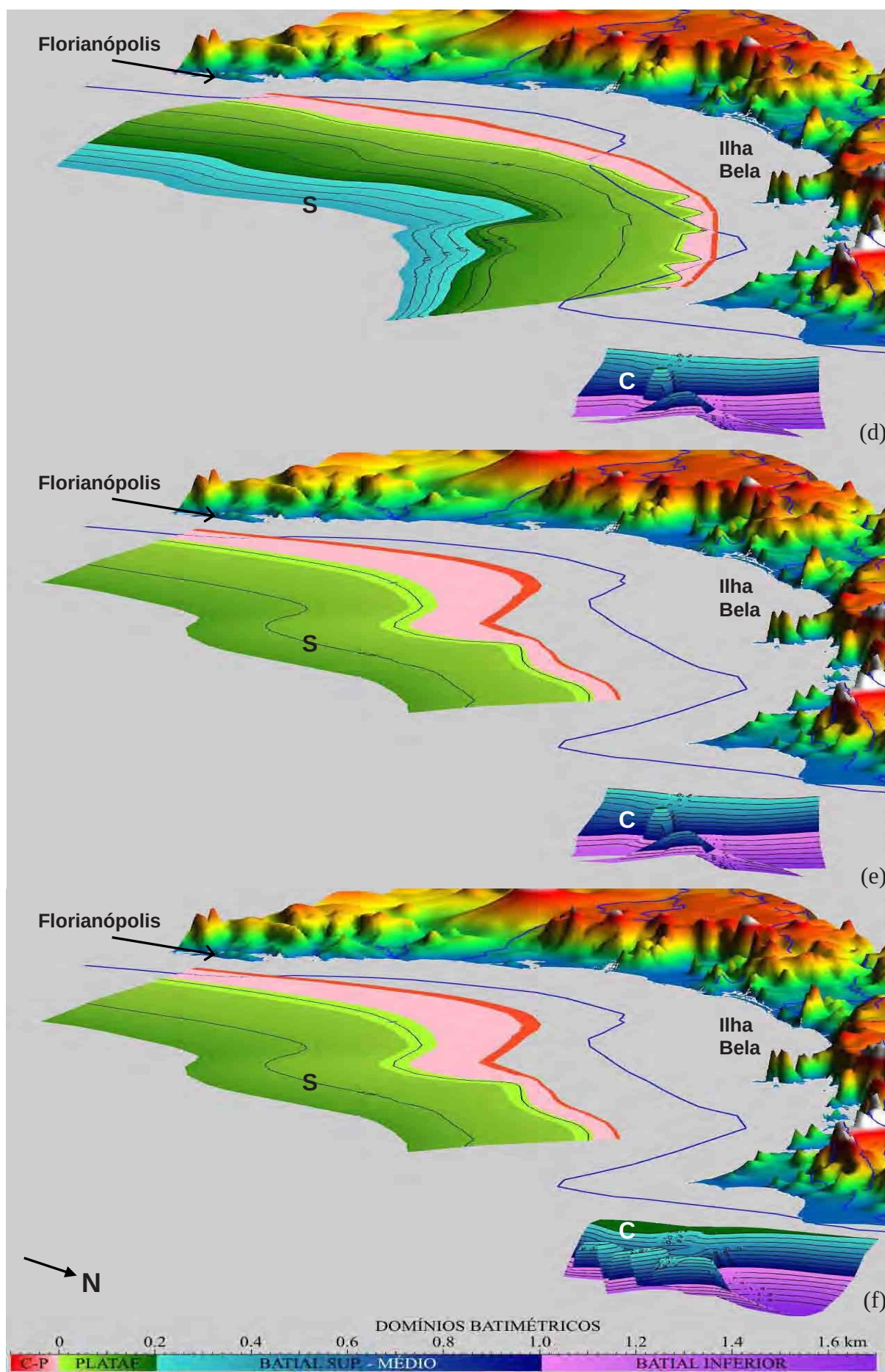


Figura 7.21d-f – Visualização simultânea da evolução paleobatimétrica das bacias de Santos e Campos. (d) “Final” do eoconiaciano (Bacia de Campos) e mesoturoniano (Bacia de Santos). (e) “Final” do eoconiaciano (Campos) e eosantoniano (Santos). (f) “Final” do Santoniano (Campos) e eosantoniano (Santos). A Bacia de Santos voltou a ser caracterizada por ambiente plataformal raso-médio, enquanto o ambiente batial dominava na Bacia de Campos, que deveria apresentar uma plataforma continental muito estreita.

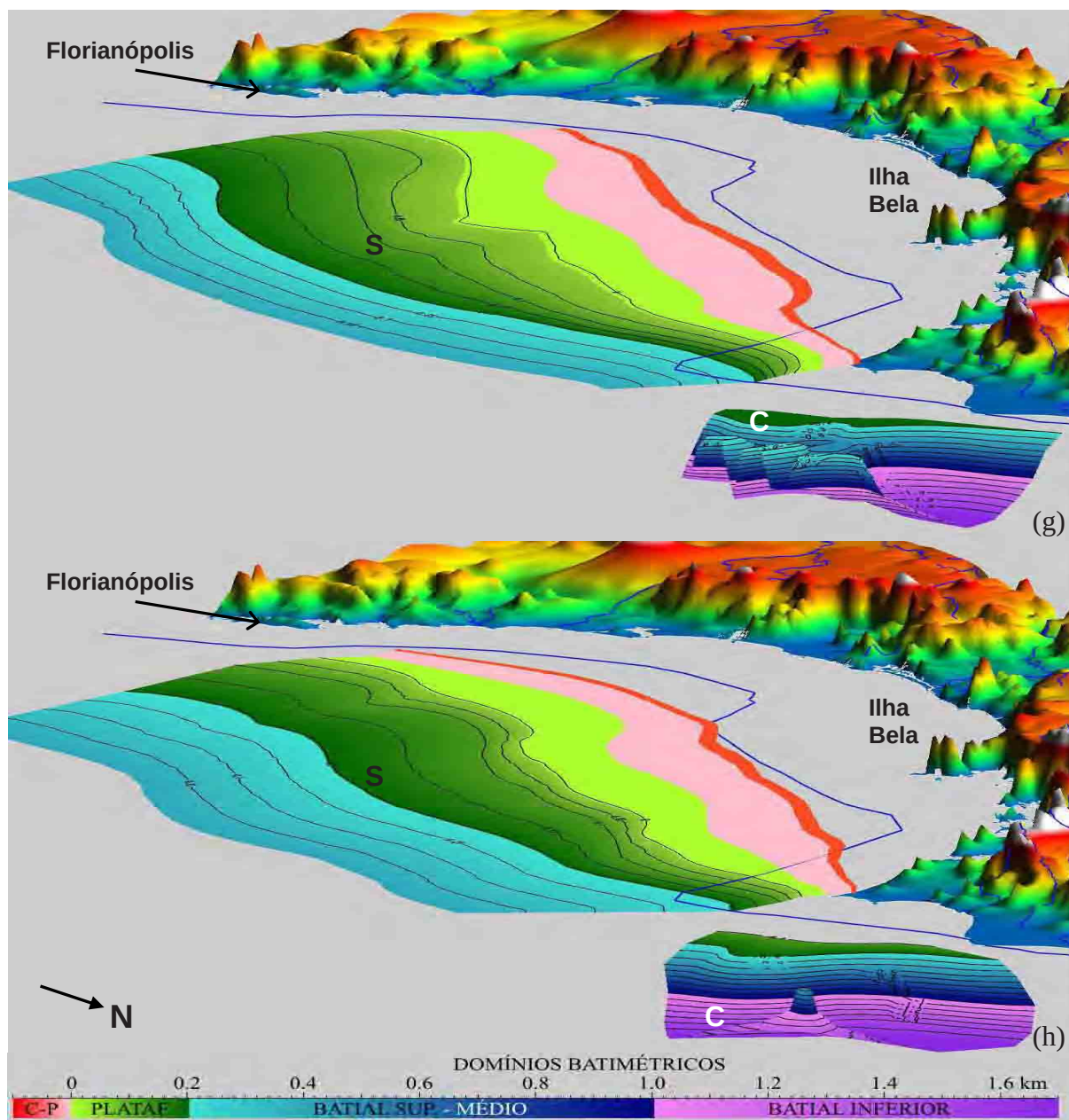


Figura 7.21g-h – Visualização simultânea da evolução paleobatimétrica das bacias de Santos e Campos. (g) “Final” do Santoniano (Bacia de Campos) e eocampaniano (Bacia de Santos). (h) “Final” do Maastrichtiano (ambas as bacias). A Bacia de Santos apresentava um ambiente relativamente mais profundo (até batial superior), além da formação de extensa porção parálica. Por outro lado, na Bacia de Campos ocorriam ambientes de deposição bem mais profundos (meio batial inferior) e íngrimes, com plataforma continental muito estreita.

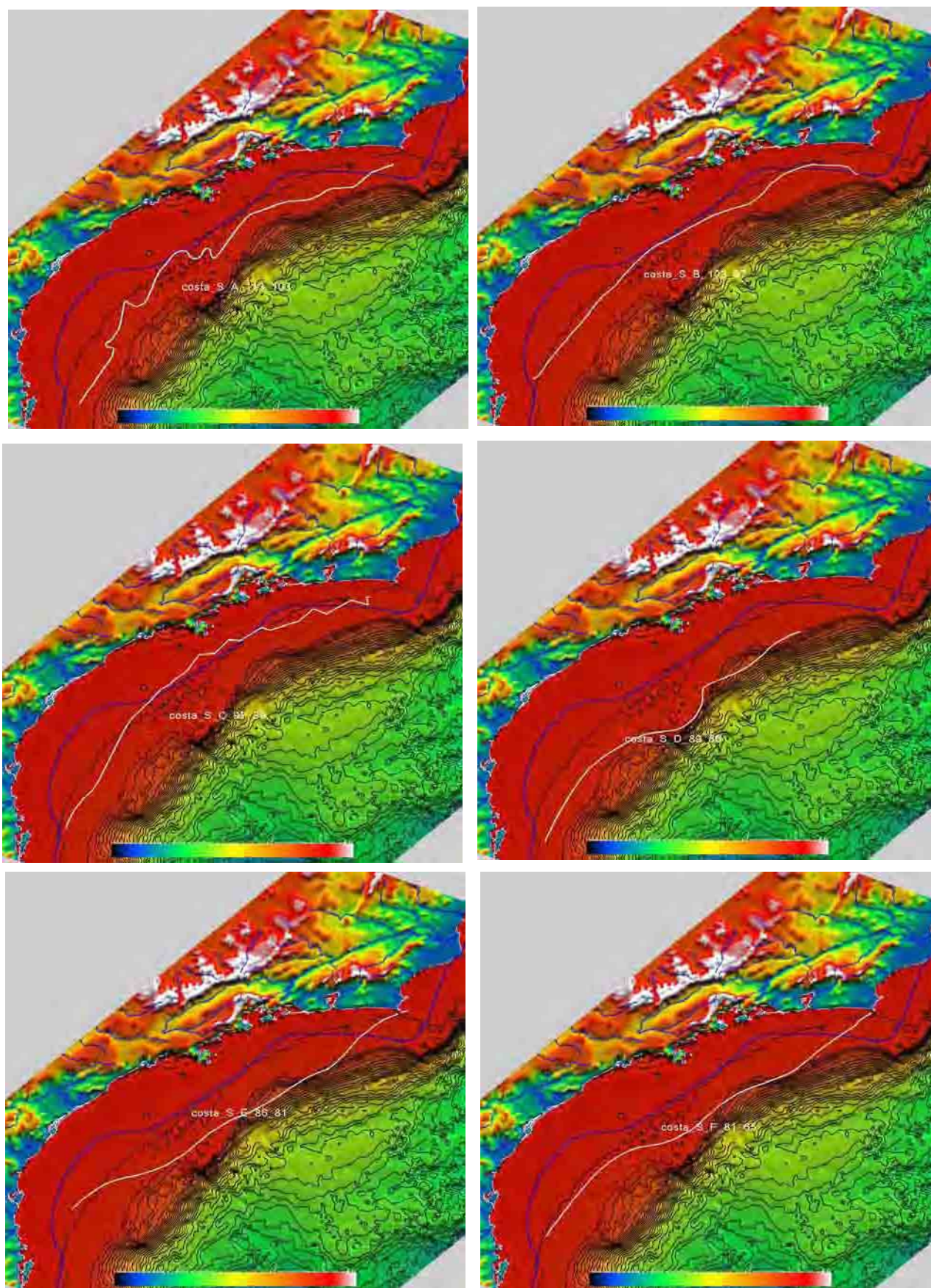


Figura 7.22 – Bacia de Santos: posicionamento da linha de costa (em branco) durante o meso-Neocretáceo. São também visualizadas a topografia, batimetria e linha de costa atuais, além da linha de charneira da bacia (em azul). A escala de cores refere-se à variação da batimetria atual. Idades consideradas: “Final” do eo-mesoalbiano (a), neoalbiano (b), mesoturoniano (c), eosantoniano (d), eocampaniano (e) e Maastrichtiano (f). Vista em planta.

7.5 Modelagem de superfícies paleobatimétricas intermediárias

Para permitir a realização de simulações mostrando dinamicamente, e de forma mais realista, a evolução do relevo batimétrico com o passar do tempo, foram modeladas superfícies paleobatimétricas “intermediárias”. Esta técnica consiste na utilização de ferramentas de modelagem geométrica 3D para criar entidades morfologicamente intermediárias entre os modelos digitais das paleobatimetrias de Santos e de Campos. Para tanto, foram geradas malhas vazadas entre pares de superfícies de idades sucessivas ao longo do período Albiano-Maastrichtiano. Por exemplo, é possível criar um modelo intermediário entre as superfícies referentes às idades neo-albiana e eoturoniana da batimetria da bacia de Campos (Figura 7.23a e b). As superfícies intermediárias possuem este nome por apresentarem uma geometria de transição entre duas superfícies de origem, permitindo modelar a evolução entre as mesmas. A técnica representa uma expansão dos procedimentos de *morphing* usados em computação gráfica 2D, os quais permitem criar imagens intermediárias entre duas imagens-base. A extensão para a modelagem 3D leva em conta a construção de superfícies cujos pontos possuam elevação (coordenada Z) intermediária em relação às superfícies-base.

O procedimento de *morphing* foi realizado no ambiente do programa GOCAD e consiste dos seguintes passos:

- Criação de duas cópias das superfícies-base com o comando *Surface Mode/New/From Surfaces*.
- Atribuição de um único valor para Z (Z=1, por exemplo) em ambas as superfícies-cópia, usando *Compute/On Object*. Em consequência disto, as superfícies assumem uma aparência “achatada” (Figura 7.23c).
- Construção de uma “curva” de interseção entre as superfícies com o comando *Curve Mode/New/From Surfaces/Intersection/With Surfaces* (Figura 7.23d). Trata-se, na verdade, de um conjunto de segmentos delimitados pela interseção das bordas das superfícies-base e considerando-se a direção do eixo vertical Z.
- Extração de pontos a partir da curva com o comando *PointSet Mode/New/From PointSet, Curve or Surface* (Figura 7.23e).
- Criação de uma curva que delimita os pontos: comando *Curve Mode/New/From PointSet/Convex Hull/Of Object*. Trata-se do fecho convexo descrito na seção 7.2.

- Ajuste do feixe convexo aos pontos: comando *Curve Mode/Edit/Fit to Points/Globally*. Este comando pode ser executado repetidamente, o que melhora a qualidade do ajuste de forma incremental (Figura 7.23f).
- Construção de uma nova superfície a partir da curva (feixe convexo) e pontos usando o comando *Surface Mode/New/From PointSets/PointSet and Curve* (Figura 7.23g).

Neste momento, está criada a superfície intermediária, faltando definir apenas os valores das coordenadas Z de seus pontos. Para tanto, executa-se os seguintes passos:

- Criação de duas novas propriedades para a superfície intermediária (chamadas “ZA” e “ZB”, por exemplo): comando *Property/Create*.
- Uso da ferramenta *Object*, arrastando a propriedade “Z” de cada uma das superfícies originais para as novas propriedades criadas na superfície intermediária. A modalidade de integração das propriedades é *Paint Vertically On*.
- Uso do comando *Compute/On Object*, selecionando a nova superfície e aplicando a fórmula $Z = (Z_A + Z_B) / 2$. Esta fórmula ilustra o caráter “intermediário” da superfície, cujos pontos possuem elevação igual à média aritmética dos pontos das superfícies-base.

Após a criação das superfícies intermediárias, é necessário proceder ao refinamento manual de sua geometria, acrescentando ou removendo triângulos dos modelos. A Figura 7.23h mostra o resultado final deste procedimento.

É possível verificar que a técnica de *morphing* descrita nesta seção é, na verdade, a composição de operações mais elementares suportadas pelo programa GOCAD. Utilizando esta técnica, foram modeladas dez superfícies intermediárias (cinco para cada bacia), o que permitiu criar um modelo de evolução morfológica mais gradual. A título de exemplo, a Figura 7.24 exhibe a evolução batimétrica da bacia de Campos do neo-albiano ao eoturoniano levando em conta a superfície intermediária gerada. Evidentemente, tal superfície apresenta uma morfologia intermediária indicativa, já que considera apenas a influência geométrica das feições das superfícies-base. Isto significa que não é possível atribuir à mesma uma idade intermediária exata entre as idades conhecidas para as duas superfícies utilizadas, sendo necessário levar em conta os fatores geológicos que influenciaram a evolução morfológica da Bacia de Campos durante o intervalo de tempo associado às superfícies de origem.

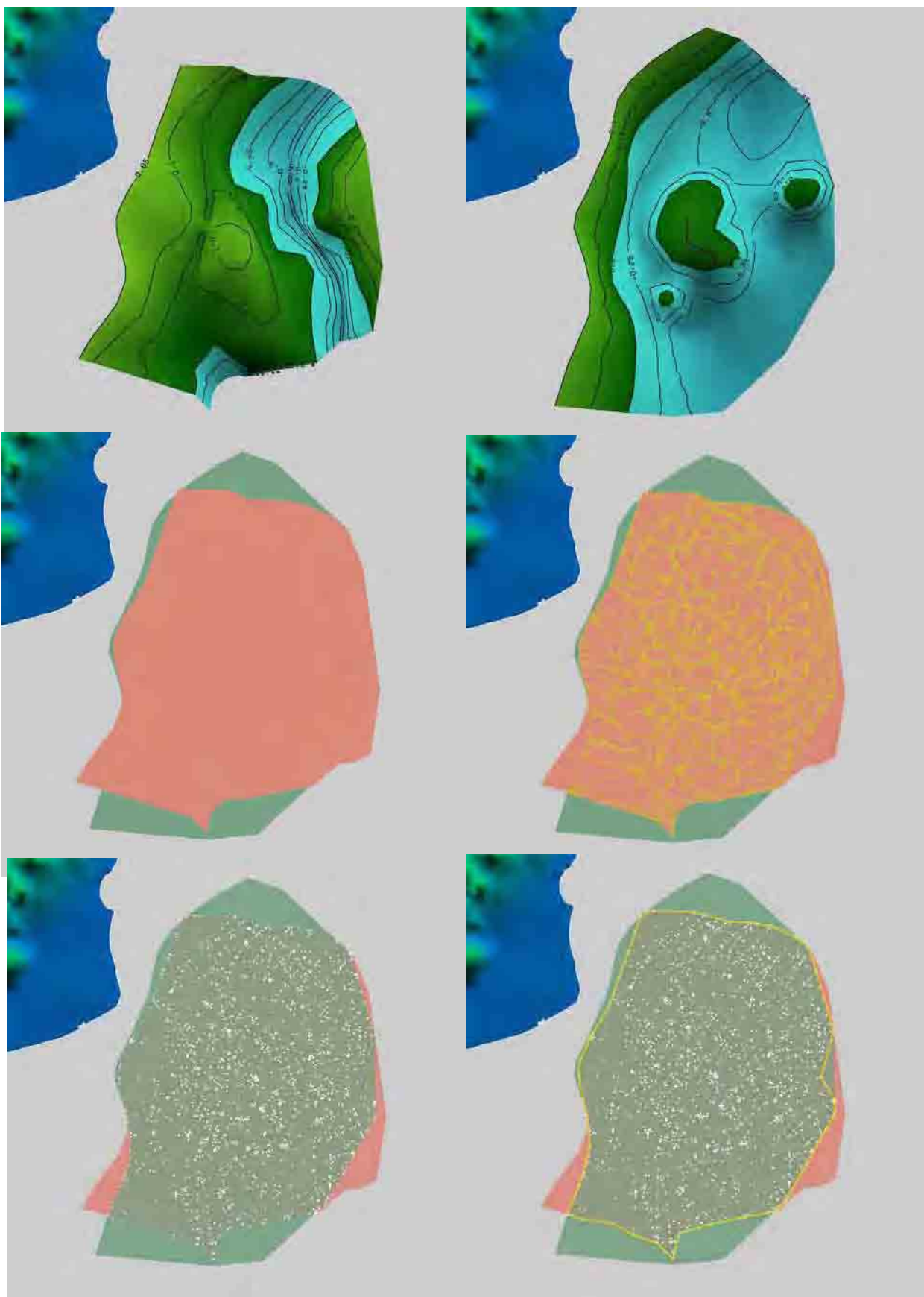


Figura 7.23a-f – Criação de superfície intermediária simulando a técnica de *morphing* no programa GOCAD. (a) e (b) Superfícies-base: paleobatimetrias neo-albiana e eoturoniana da Bacia de campos. (c) Criação de duas cópias com valor Z=1. (d) Criação de uma “curva” de interseção entre as mesmas. (e) Extração dos pontos da curva. (f) Criação e ajuste de uma curva de feixo convexo aos pontos (em amarelo).

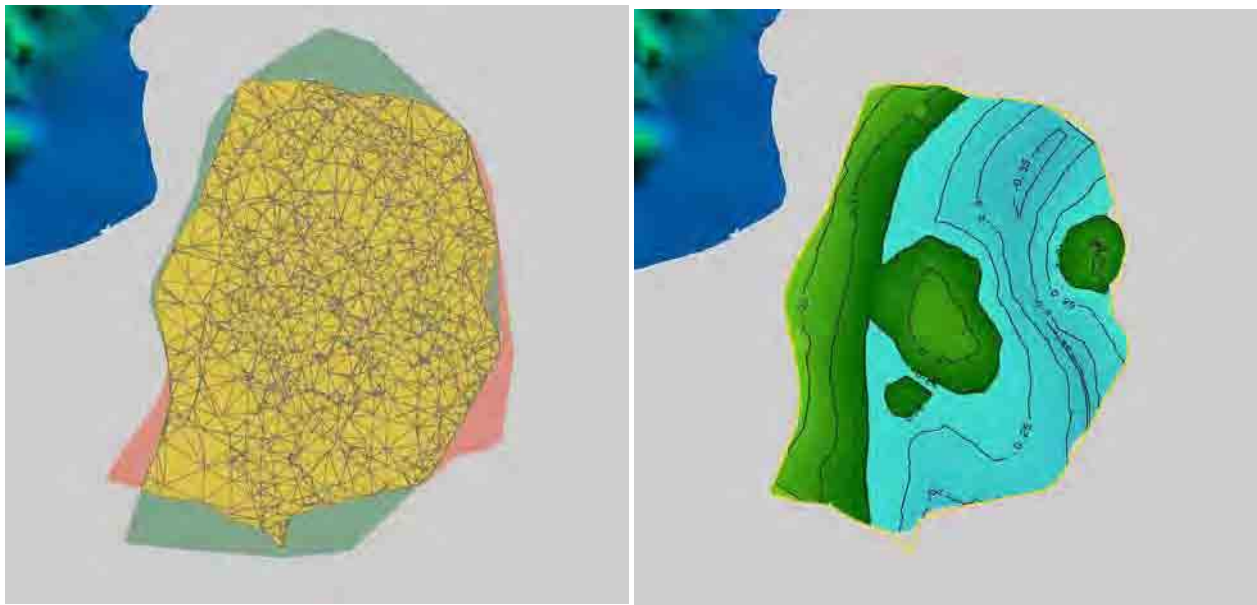


Figura 7.23g-h – Continuação da figura anterior. (g) Criação de uma superfície a partir dos pontos e feixe convexo da Figura 7.23f. (h) Visualização da superfície intermediária resultante.

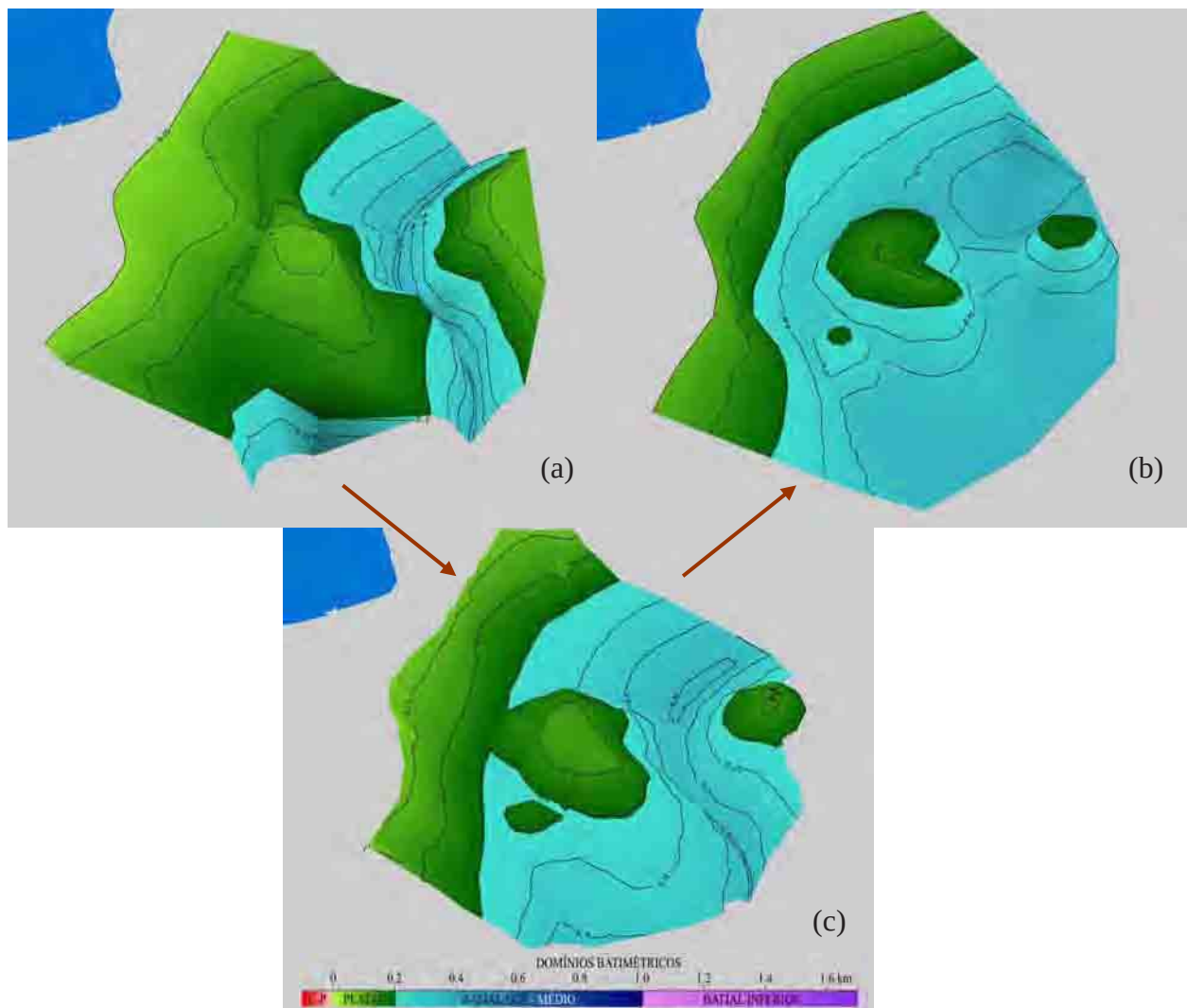


Figura 7.24 – Evolução da Bacia de Campos enriquecida usando superfícies intermediárias. (a) “Final” do neo-albiano (97,5 Ma). (b) “Final” do eoturoniano (90 Ma). (c) Superfície intermediária entre os modelos anteriores. O fator de exagero vertical é 100. Visada para Norte.

7.6 Integração com demais dados geológicos

Para integrar mais informações geológicas aos modelos paleobatimétricos construídos e aumentar assim seu potencial de interpretação, foram utilizadas outras informações obtidas da literatura. Estes dados, no formato de mapas bidimensionais, foram importados no ambiente do programa GOCAD. Com efeito, este aplicativo permite a integração de dados advindos de fontes distintas, proporcionando uma visualização unificada das informações modeladas, bem como o cruzamento das propriedades exibidas. A utilização de mapas 2D auxiliares ocorre, normalmente, através de uma técnica de computação gráfica denominada *Texture Mapping* (FOLEY et al., 1996). Utilizando tal procedimento, os mapas, previamente digitalizados e devidamente georreferenciados, são “sobrepostos” às malhas vazadas que modelam a batimetria atual ou a paleobatimetria das bacias de Santos e Campos. Desta forma, é possível estabelecer relações entre a morfologia das superfícies e a informação contida nos mapas sobrepostos. Mapas utilizados para esta finalidade são denominados “temáticos” e sua aplicação será ilustrada nas figuras a seguir. A técnica de *Texture Mapping* já foi utilizada para sombrear as superfícies paleobatimétricas usando a escala de cores da Figura 7.5. Outros exemplos envolveram as superfícies da topografia e batimetria atuais (Figuras 7.13 e 7.14). Nestes casos, o programa GOCAD cria uma imagem bidimensional usando as cores da escala considerada e em seguida realiza sua sobreposição (ou mapeamento) às superfícies.

Um primeiro exemplo de mapas auxiliares (FRANÇA et al., 1995) foi mostrado na Figura 4.14. Trata-se de três mapas que mostram a distribuição dos sedimentos depositados na margem continental sudeste brasileira durante o Cretáceo. Os períodos considerados são o Neocomiano (144-125 Ma), Barremiano-Albiano (125-97,5 Ma) e Cenomaniano-Maastrichtiano (97,5-65 Ma). Os mapas, originalmente em preto e branco, foram digitalizados e georreferenciados no ambiente do programa AutoDesk Map, usando a informação da linha de costa. Em seguida, nas imagens foram criadas hachuras coloridas para gerar texturas (mapas temáticos) a serem utilizadas no programa GOCAD (Figura 7.25). A Figura 7.26 mostra a sobreposição dos três mapas à superfície da batimetria atual. Embora os mapas não possam ser associados ao relevo batimétrico atual, as figuras valem como primeira contextualização dos litotipos em relação às principais feições batimétricas. Uma aplicação mais adequada consiste na sobreposição dos referidos mapas às superfícies paleobatimétricas descritas anteriormente. Considerando o período de referência das mesmas (meso-Neocretáceo), foram utilizados dois dos três mapas gerados, sendo que o primeiro refere-se às duas superfícies correspondentes ao Albiano.

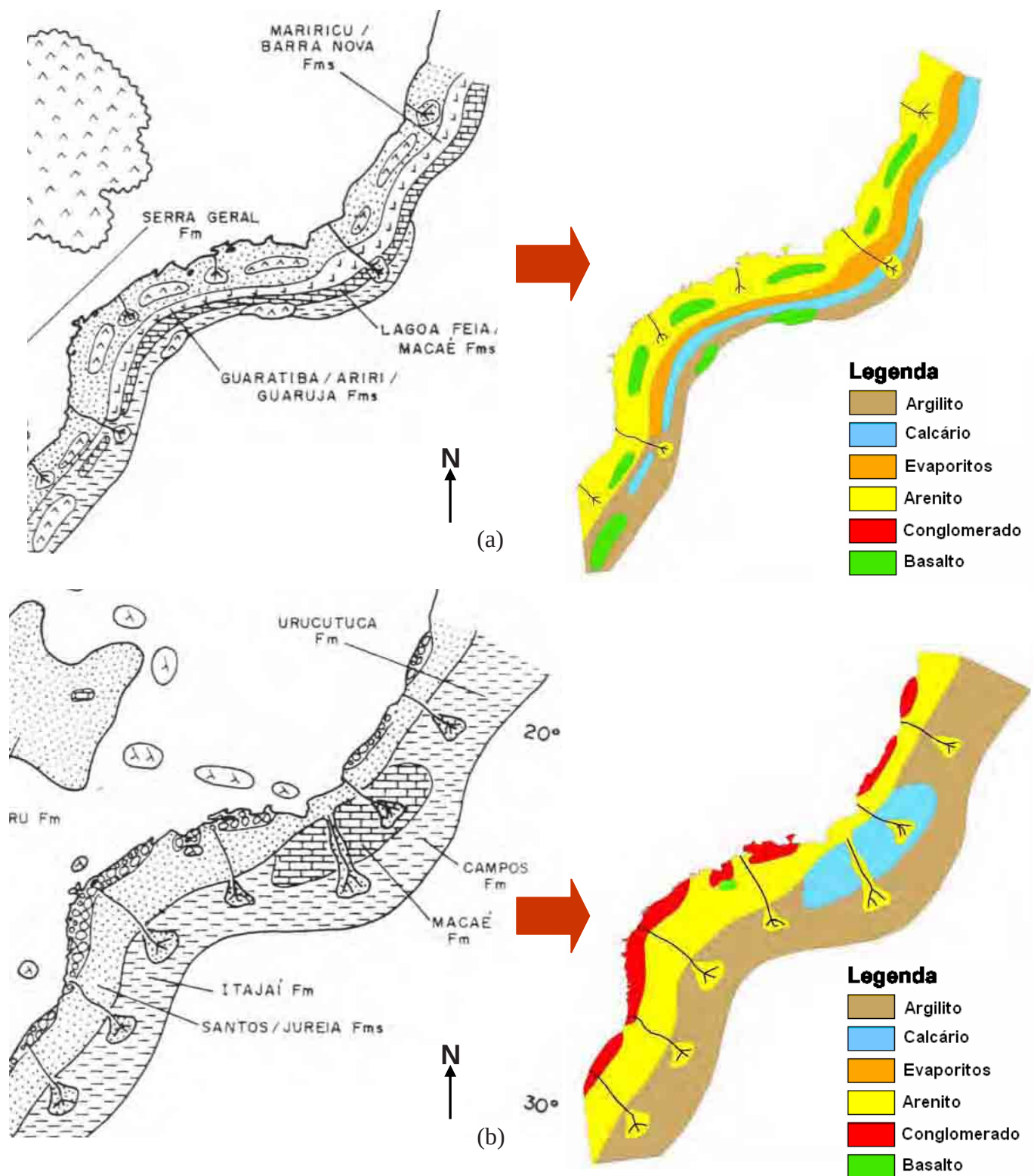


Figura 7.25 – Criação de hachuras para os mapas de litotipos da margem continental sudeste brasileira (França et al., 1995). (a) Barremiano-Albiano (125-97,5 Ma). (b) Cenomaniano-Maastrichtiano (97,5-65 Ma).

Para as demais superfícies (período Cenomaniano-Maastrichtiano) foi utilizado o segundo mapa. A Figura 7.27 mostra a evolução simultânea das bacias de Santos e Campos utilizando os mapas ilustrados acima. A textura com a escala de cores correspondentes aos vários ambientes deposicionais foi substituída pelos mapas de litotipos referentes ao Cretáceo. Uma desvantagem desta substituição é a impossibilidade de estimar a variação batimétrica das superfícies a partir das novas texturas utilizadas. Para amenizar tal inconveniente, sobre as superfícies foram também traçadas isolinhas com intervalos de 100 e 200 m (bacias de Santos e Campos, respectivamente), que também permitem realçar a morfologia das superfícies visualizadas. Evidentemente, a utilização de apenas dois mapas temáticos para seis idades de evolução paleobatimétrica ao longo do meso-Neocretáceo representa uma simplificação da distribuição de litotipos, devendo os mapas em questão ser considerados como de tendência, já que abrangem um período muito longo. A disponibilidade destes mapas para um número maior de intervalos de idades, geraria uma visualização mais realista e representativa.

Outro mapa utilizado contém as principais feições tectônicas da região das bacias de Campos e Santos (SHOBENHAUS & CAMPOS, 1984 apud CAINELLI & MOHRIAK, 1998) (Figura 7.28). Neste mapa estão representadas algumas estruturas oceânicas, bem como a idade dos principais sedimentos depositados na região. O mapa foi digitalizado e georreferenciado usando o programa ER Mapper. Em seguida, a imagem resultante foi importada no programa GOCAD como entidade *Voxet*.

Visualizando a imagem em planta (direção perpendicular ao eixo Z), foram digitalizadas algumas entidades acessórias. Trata-se de curvas criadas diretamente sobre a imagem, digitalizando pontos através do mouse. Um exemplo é a linha de charneira das bacias de Campos e Santos (Figura 7.13 e seguintes). Outras curvas foram digitalizadas em correspondência das zonas de fraturas de Florianópolis, Curitiba e Rio de Janeiro. A partir destas três curvas, foram criados planos de falha (superfícies). Para tanto, as curvas iniciais foram posicionadas na elevação 0, três cópias foram criadas e posicionadas no valor batimétrico -3 (três km de profundidade). Por fim, os planos de falha foram criados usando a função *Surface Mode/New/ From Several Curves*. Para as superfícies obtidas foi definido um índice de transparência de 30%. A Figura 7.29 exhibe as entidades modeladas a partir do mapa de estruturas oceânicas. Tal mapa foi também sobreposto ao relevo da batimetria atual, usando o procedimento de *Texture Mapping*. Os planos de falha correspondentes às zonas de fraturas facilitam a identificação de tais estruturas em visadas propriamente 3D (direção de observação não paralela ao eixo Z).

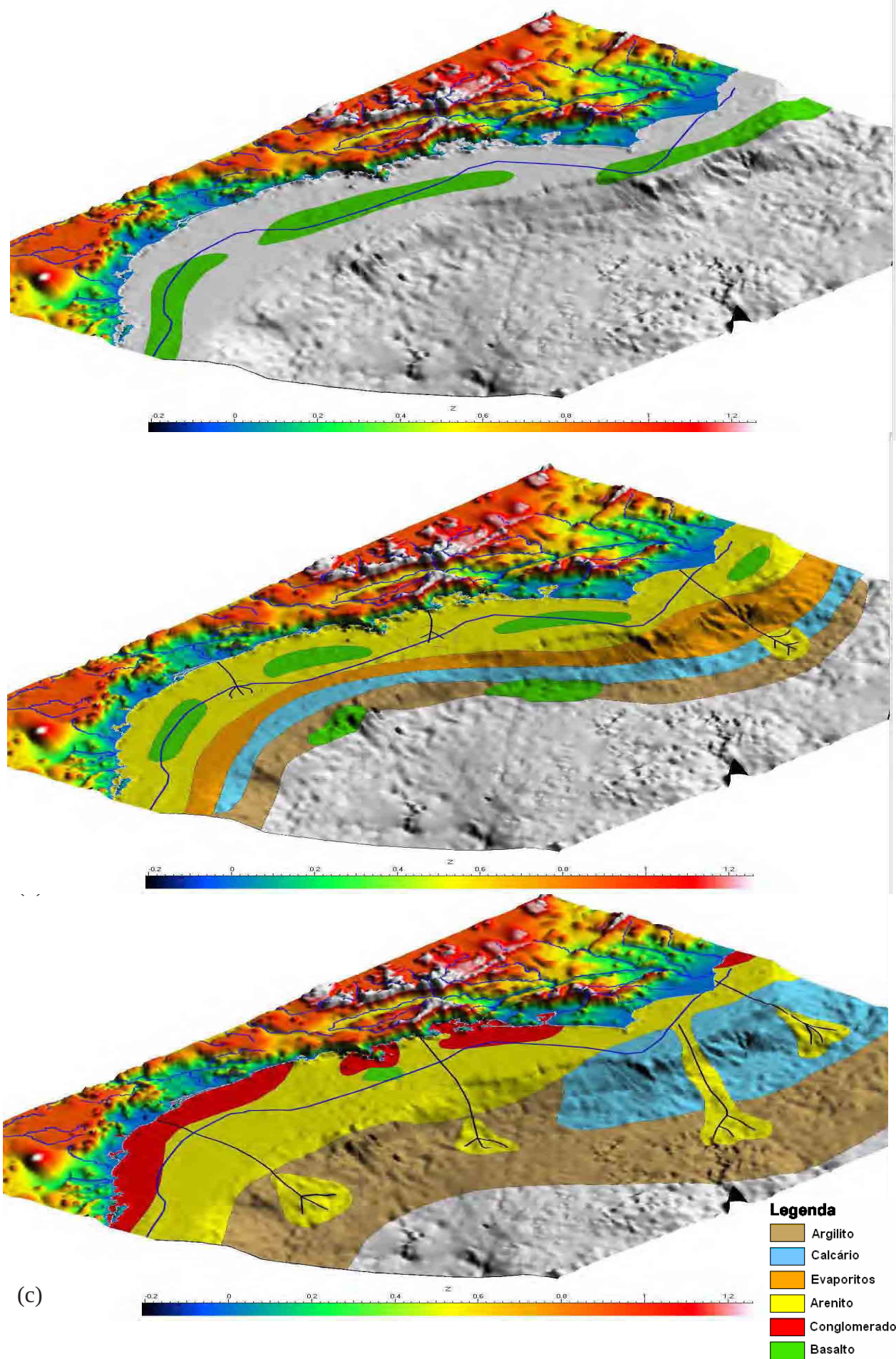


Figura 7.26 – Sobreposição de mapas de litotipos à superfície da batimetria atual. A escala de cores acompanha a variação altimétrica da topografia. Linha de charneira em azul. (a) Neocomiano (144-125 Ma). (b) Barremiano-Albiano (125-97,5 Ma). (c) Cenomaniano-Maastrichtiano (97,5-65 Ma). Visada para Norte. Exagero vertical: 30.

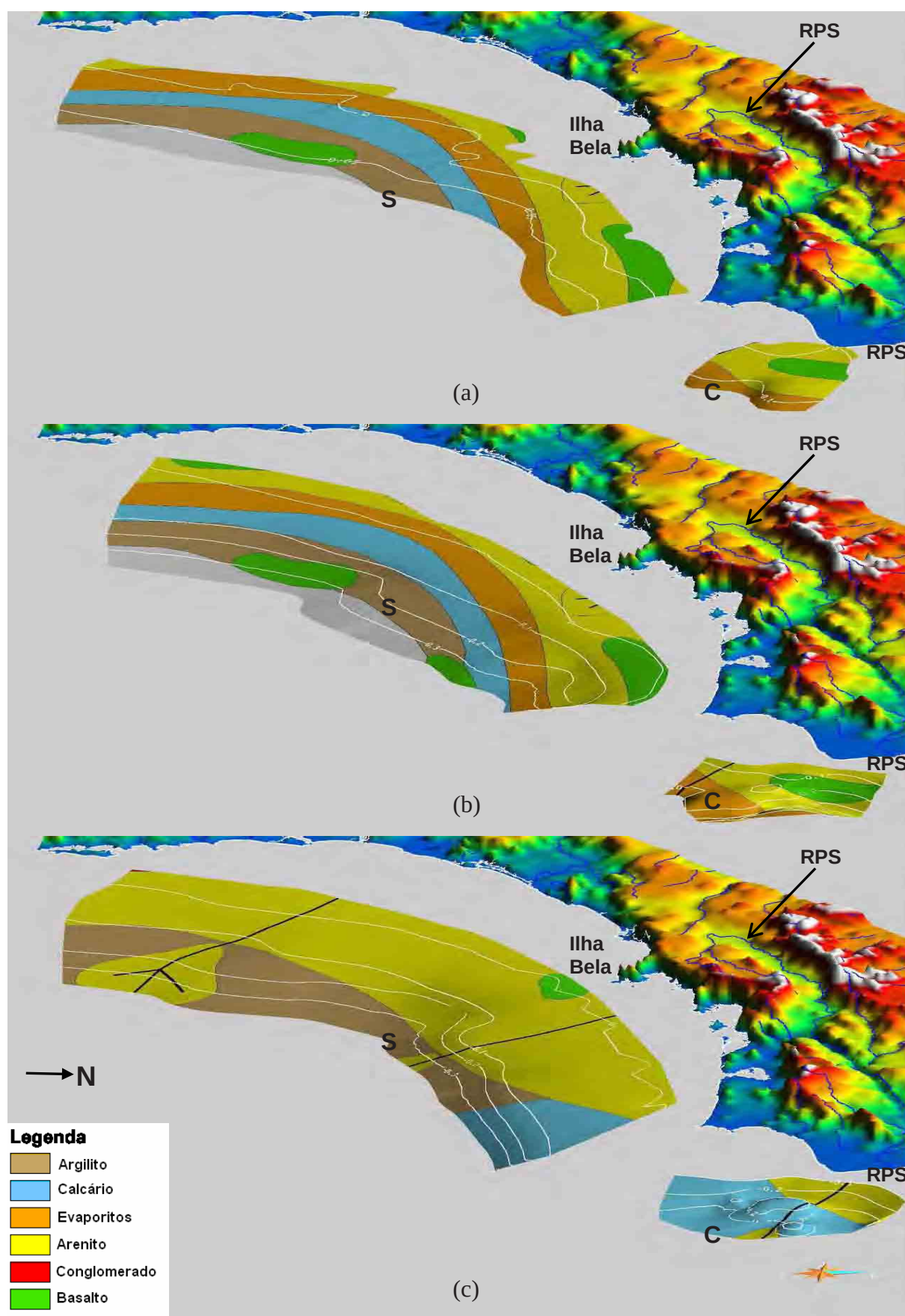


Figura 7.27a-c – Evolução da batimetria das bacias de Santos (S) e Campos (C) com sobreposição de mapas de litotipos de França et al. (1995). Curvas isobatimétricas traçadas em branco. Posição temporal dos mapas: vide Figura 7.2. (a) “Final” do eo-mesoalbio. (b) “Final” do neo-albiano. (c) “Final” do mesoturoniano (Bacia de Santos) e eoturoniano (Bacia de Campos). RPS = Rio Paraíba do Sul. Visada para Oeste. Fator de exagero vertical: 50 (20 para a topografia).

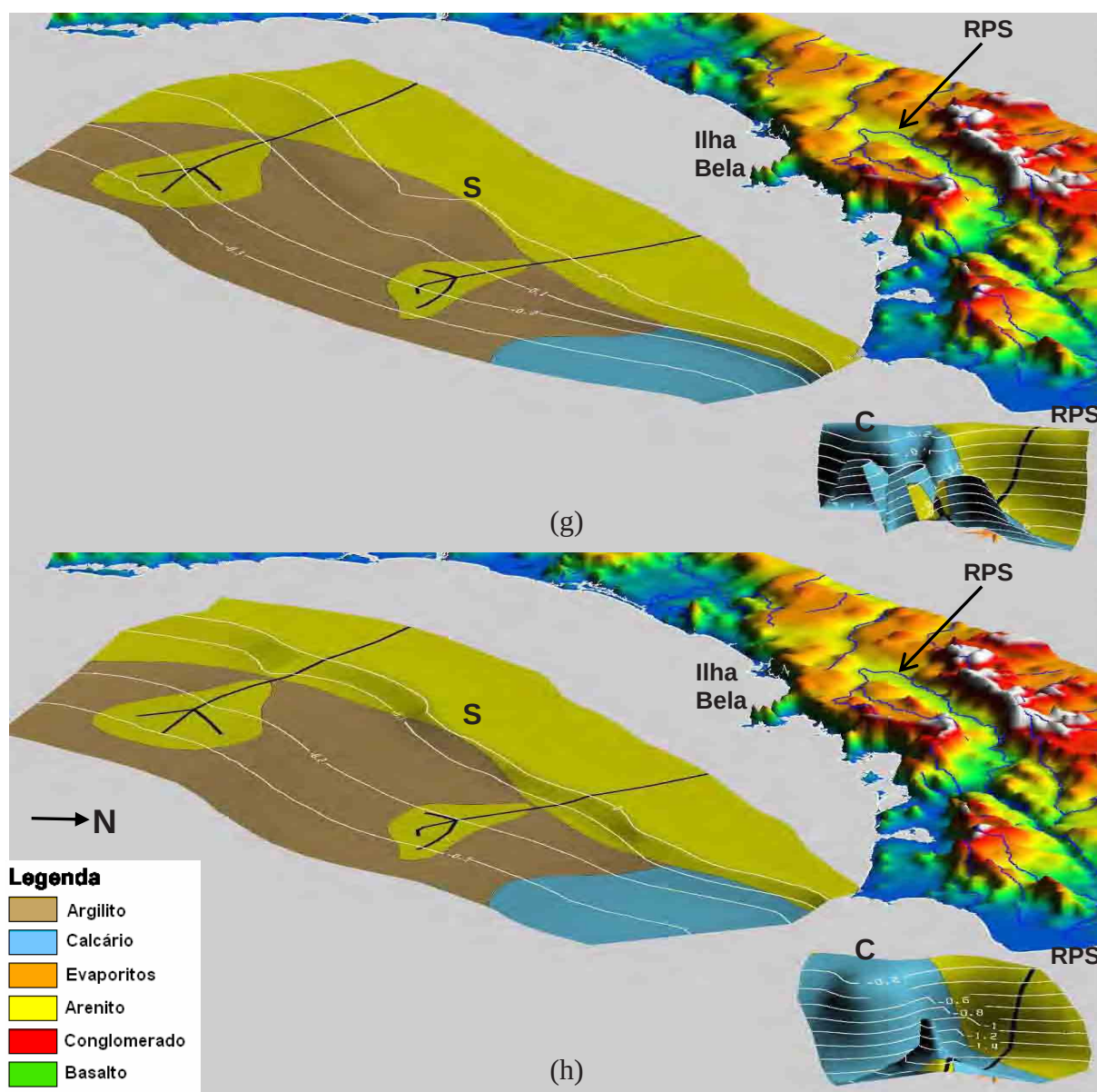


Figura 7.27g-h – Continuação da figura anterior. (g) “Final” do eocampaniano (Bacia de Santos) e neo-santoniano (Campos). (h) “Final” do Maastrichtiano.

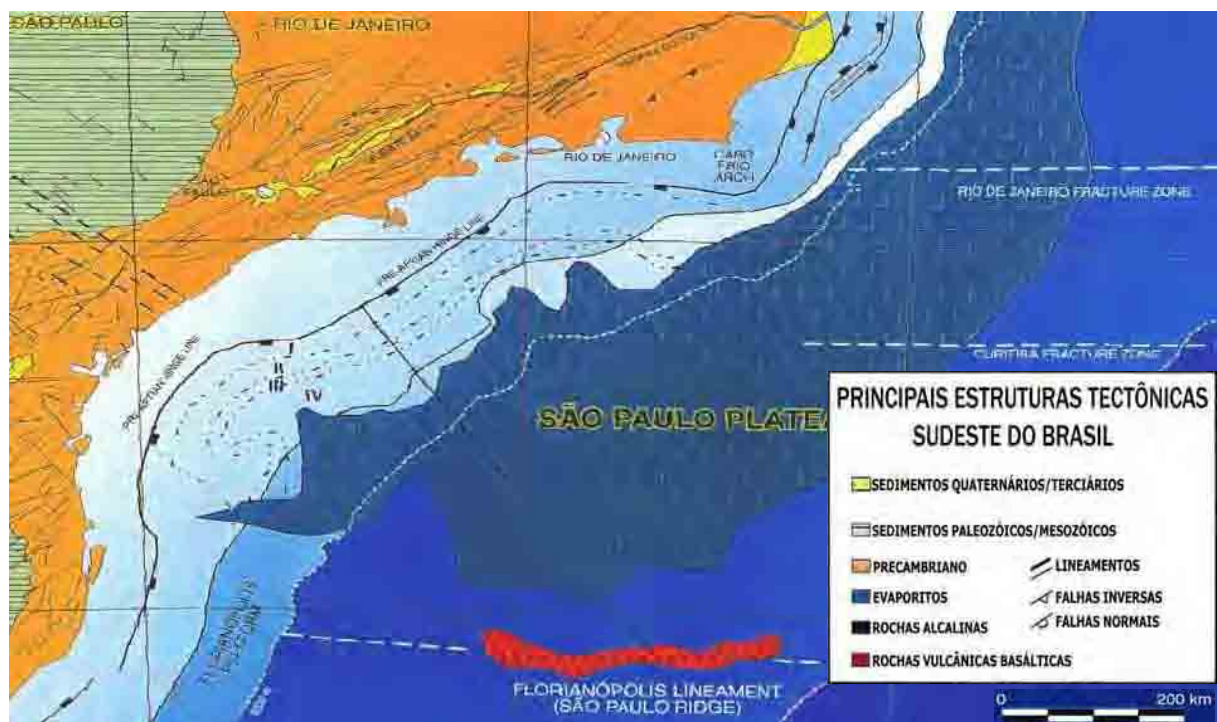


Figura 7.28 – Mapa das principais feições tectônicas da margem continental sudeste brasileira. Fonte: Cainelli & Mohriak (1998).

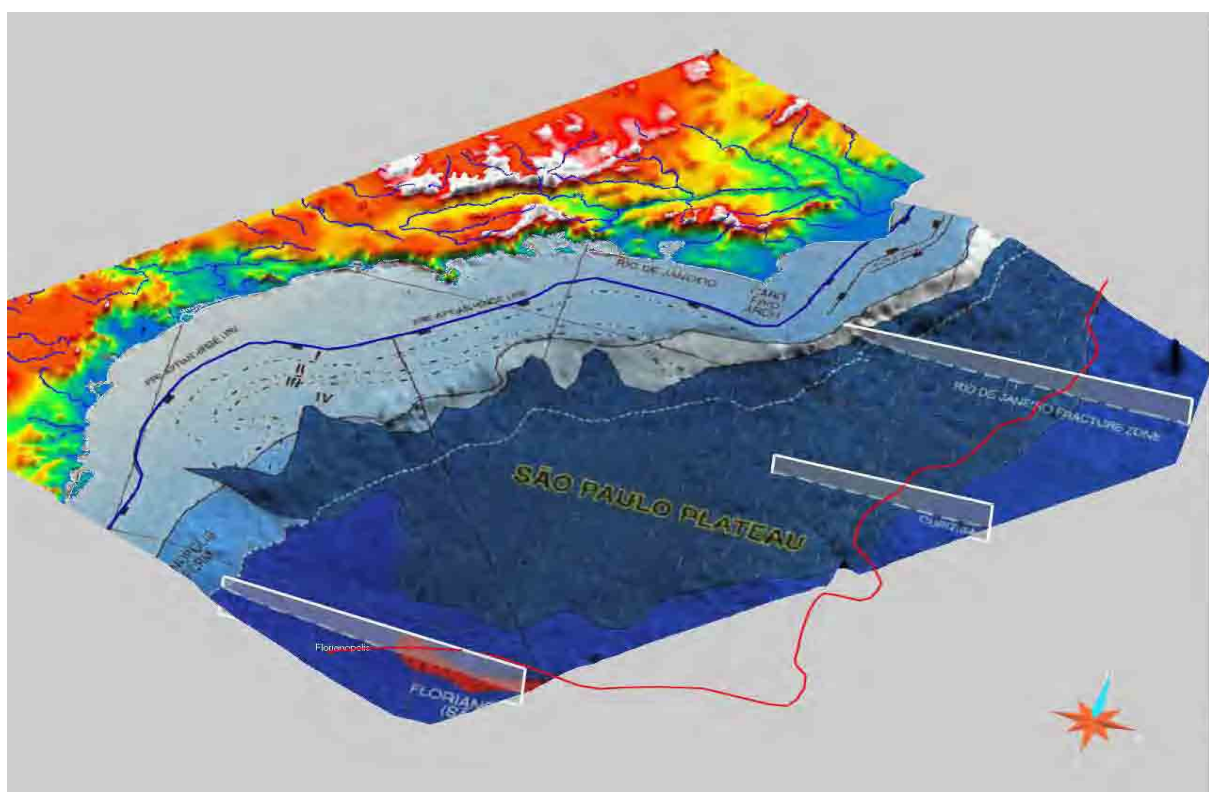


Figura 7.29 – Mapa das estruturas oceânicas sobreposto à batimetria atual. Planos de falhas semitransparentes para as principais zonas de fraturas. Linha de charneira em azul. Limite crosta continental/oceânica em vermelho. Fator de exagero vertical: 20.

Outra fonte de mapas temáticos utilizados para sombrear os modelos digitais da paleobatimetria foi o trabalho de Modica & Brush (2004), que contém diversos mapas litológicos e paleobatimétricos para a Bacia de Santos e referentes ao Cretáceo-Terciário. Os três mapas confeccionados para o meso-Neocretáceo foram mostrados nas Figuras 4.15 a 4.17, sendo os períodos considerados o Albiano (113-97,5 Ma), Coniaciano-mesocampaniano (88,5-76 Ma) e neocampaniano-Maastrichtiano (76-65 Ma). A Figura 7.30 exibe a evolução da Bacia de Santos durante o período de estudo, sendo que às superfícies paleobatimétricas anteriormente construídas foram sobrepostos os referidos três mapas. Novamente, a informação batimétrica e morfológica foi preservada utilizando-se isolinhas com espaçamento vertical de 50 (Figura 7.30a e d) e 100 m.

Adicionalmente, a Figura apresenta duas novas ordens de entidades. A primeira consiste no grupo de três planos de falhas semitransparentes representando às zonas de transferências de Merluza, São Paulo e Ilha Grande (Figuras 4.16 e 4.17). Outra entidade mostra a localização das principais cidades da região de estudo. Para tanto, foram utilizados objetos do GOCAD do tipo poço (*Well*). A coluna de cada “poço” foi usada para indicar a localização exata de uma cidade. O conjunto de poços foi importado a partir de outro projeto de modelagem de estruturas da região Sudeste realizado neste Laboratório.

O trabalho de Modica & Brush forneceu também a oportunidade de integrar informações de um perfil geológico ao ambiente de visualização 3D do GOCAD (Figura 7.31). A imagem correspondente ao perfil foi rasterizada e em seguida georreferenciada. A visualização simultânea do perfil e das demais entidades 3D modeladas é mostrada nas Figuras 7.32 a 7.34. São integradas informações referentes a três momentos distintos da evolução da bacia de Santos. A possibilidade de integrar dados de superfície e de subsuperfície constitui uma das potencialidades mais importantes da ferramenta, e decorre da capacidade de visualizar de forma simultânea objetos modelados a partir de fontes distintas.

Um último exemplo de sobreposição de imagens a superfícies 3D envolveu a utilização de um mapa de localização dos blocos de exploração e produção da Agência Nacional do Petróleo. O mapa, obtido do BDEP (2005), foi sobreposto à superfície da batimetria atual. A Figura 7.35 mostra a localização dos blocos nas regiões das bacias de Santos e Campos. É também exibida a localização dos poços cujos dados deram origem aos mapas utilizados para modelar a paleobatimetria das referidas bacias.

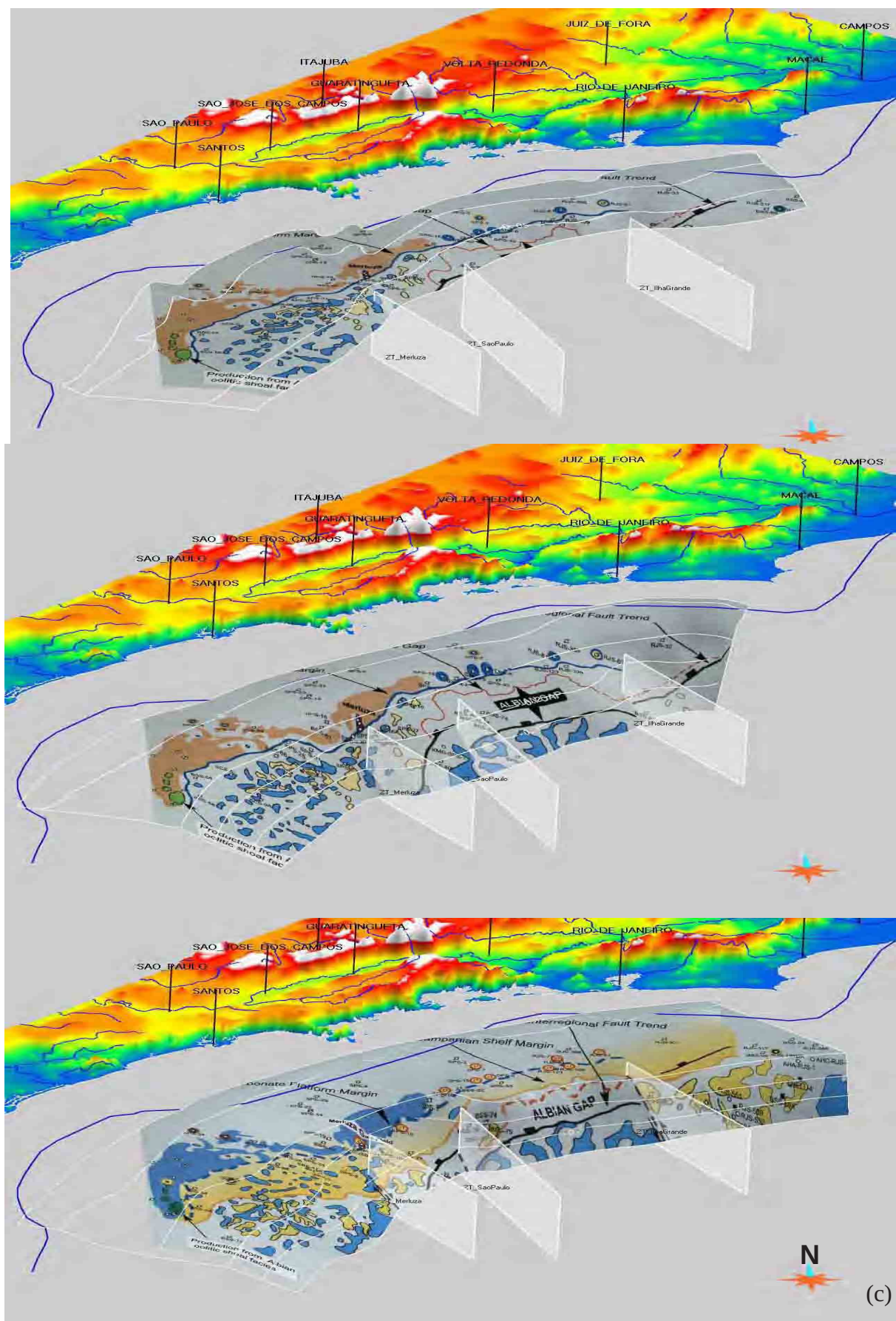


Figura 7.30a-c – Evolução da batimetria da Bacia de Santos usando superfícies sombreadas com mapas de litotipos (Modica & Brush, 2004). Planos semitransparentes das zonas de transferências de Merluza, São Paulo e Ilha Grande. (a) “Final” do mesoalbian (103 Ma). (b) “Final” do neo-albian (97,5 Ma). (c) “Final” do mesoturoniano (89 Ma). Fator de exagero vertical: 200 (paleobatimetria) e 20 (topografia e demais entidades). Visada para Norte.

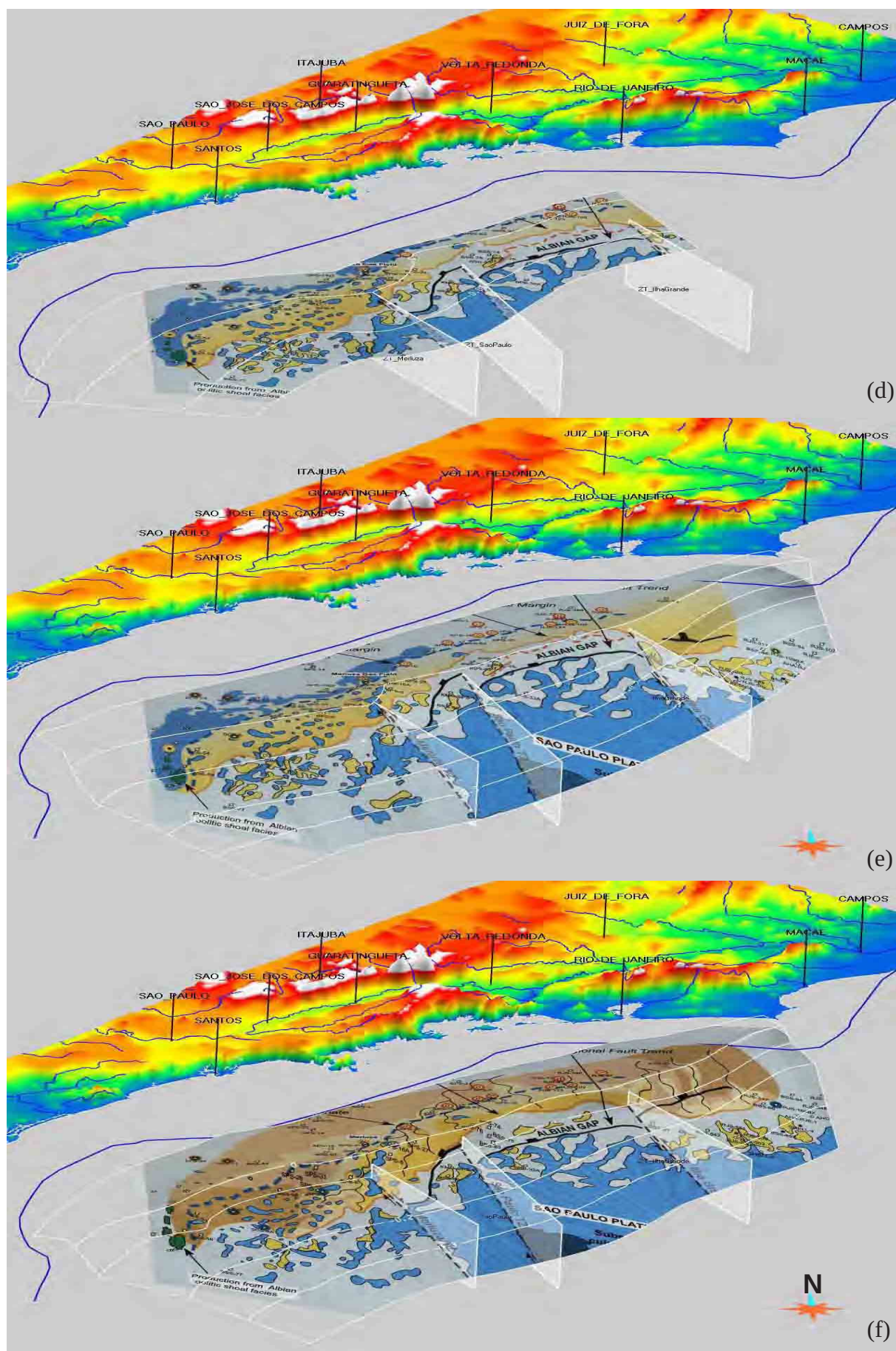


Figura 7.30d-f – Continuação da figura anterior. (d) “Final” do eosantoniano (86 Ma). (e) “Final” do eocampaniano (81 Ma). (f) “Final” do Maastrichtiano (65 Ma).

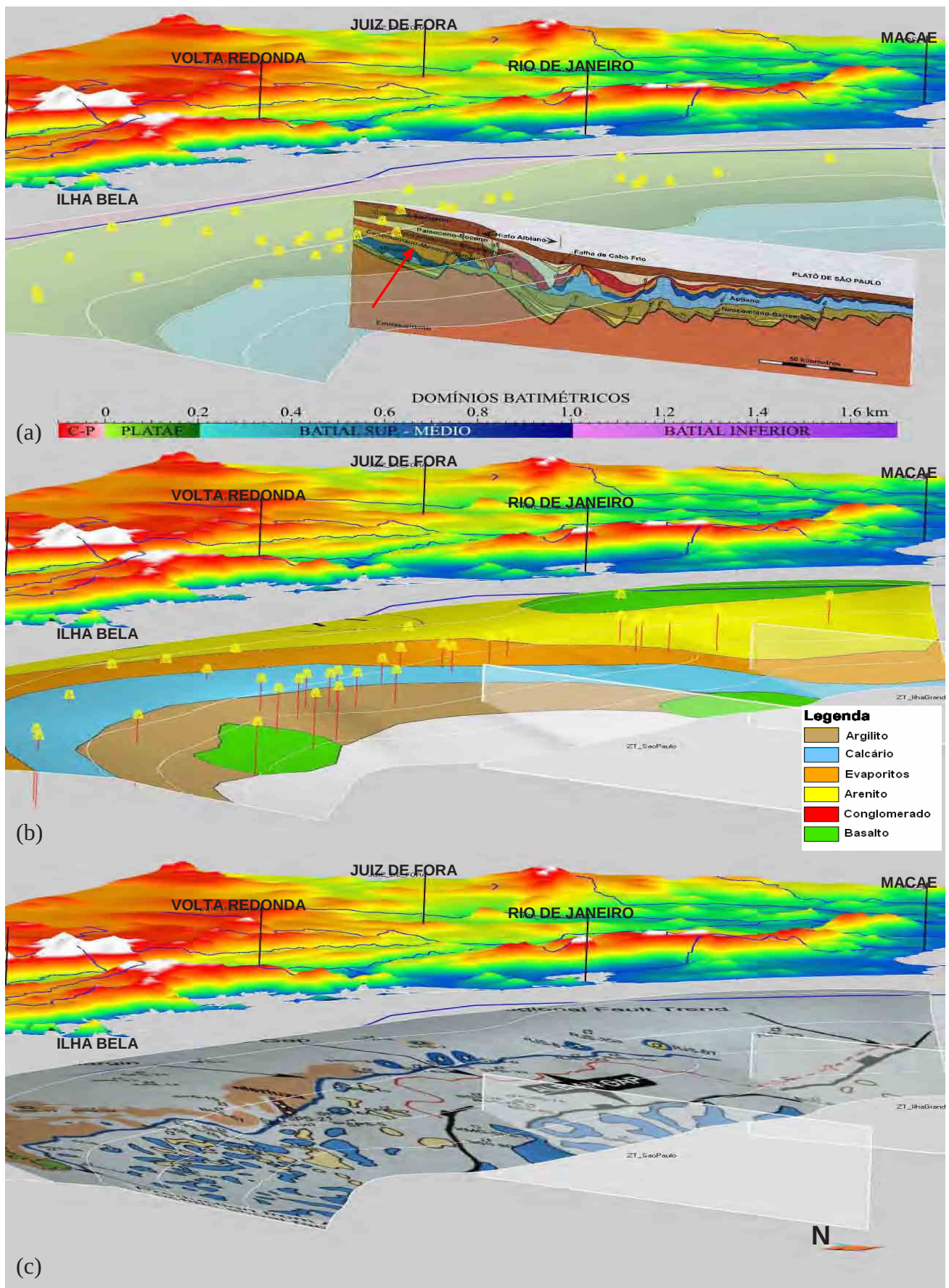


Figura 7.32 – Bacia de Santos: integração de dados sísmicos, paleobatimétricos e litológicos advindos de fontes distintas. (a) Paleobatimetria ao “final” do neo-albiano (97,5 Ma) com transparência para realçar um perfil geológico (Modica & Brush, 2004). A profundidade do topo da sequência neo-albiana (azul escuro) é indicada pela seta. (b) e (c) Sobreposição de mapas de litotipos de França et al. (1995) e de Modica & Brush (2004), respectivamente. São também exibidas as zonas de transferência de São Paulo e Ilha Grande (planos semitransparentes). Fator de exagero vertical: 100 para a paleobatimetria e 10 para as demais entidades (incluindo o perfil).

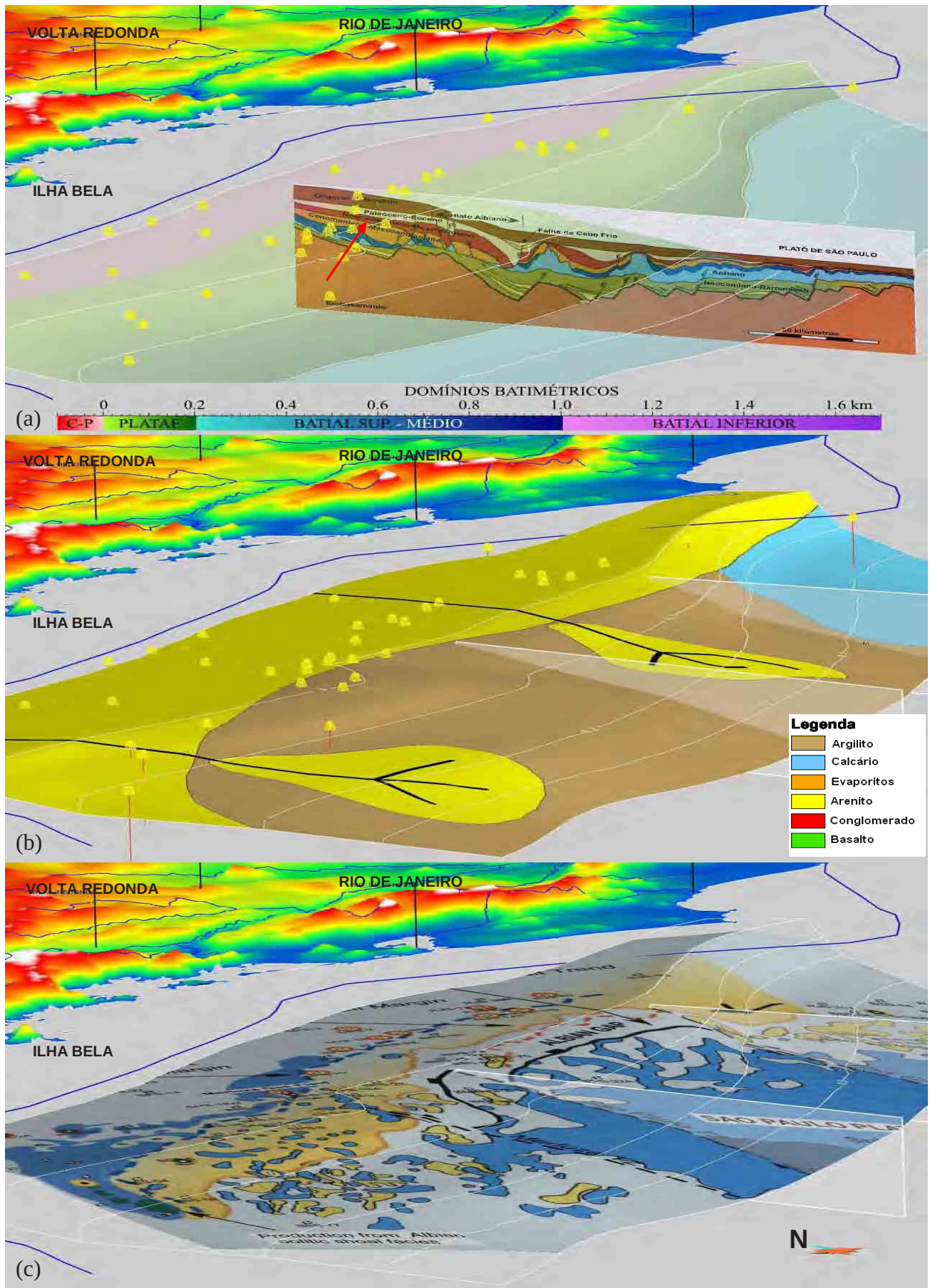


Figura 7.33 – Bacia de Santos: integração de dados sísmicos, paleobatimétricos e litológicos. (a) Paleobatimetria ao “final” do eocampaniano (81 Ma). No perfil, a profundidade atual do topo da seqüência mesocampaniana (cor laranja) é indicada pela seta. (b) e (c) Sobreposição de mapas de litotipos de França et al. (1995) e de Modica & Brush (2004).

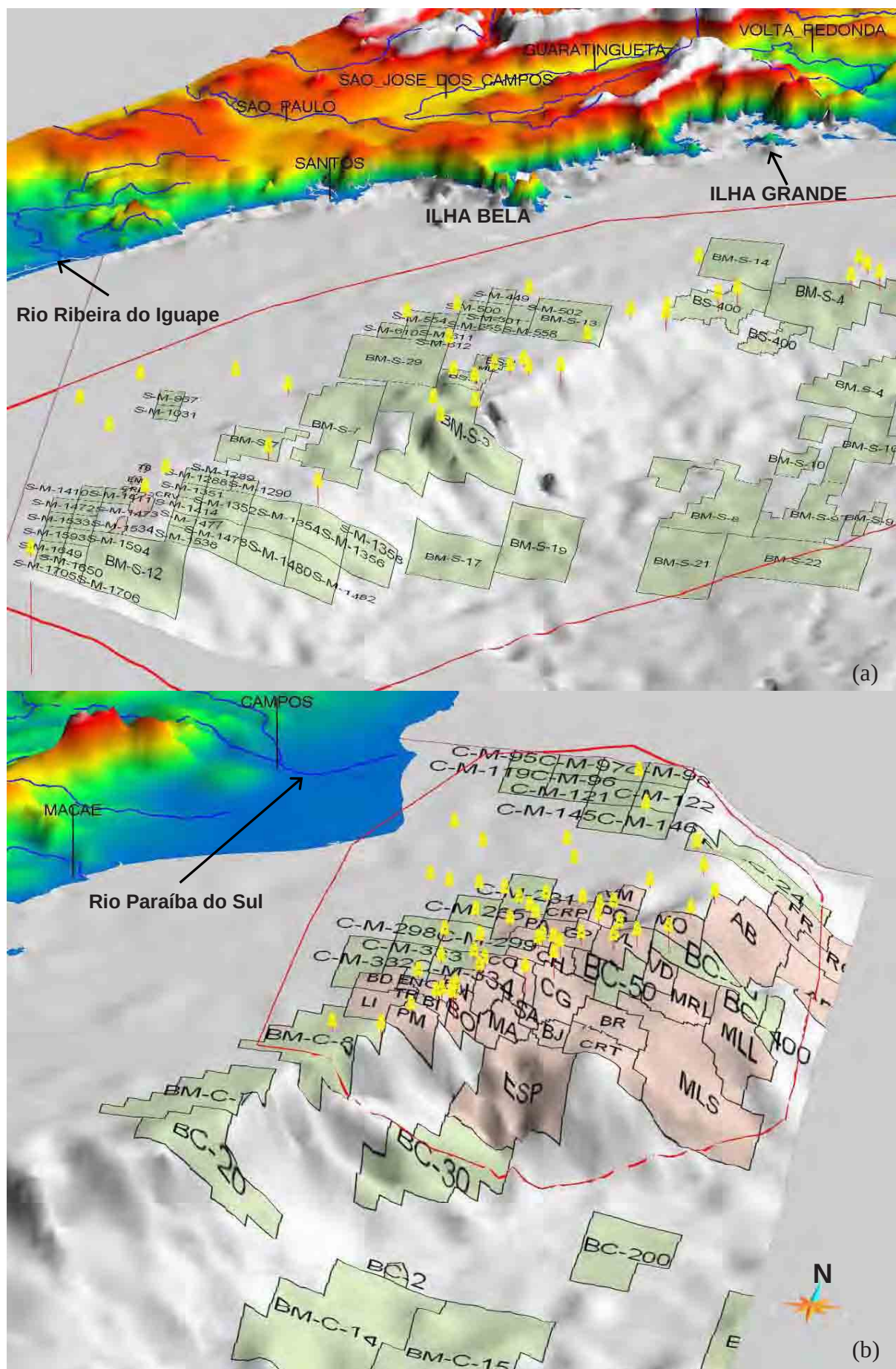


Figura 7.35 – Sobreposição do mapa dos blocos de exploração (cor verde) e produção (cor rosa) da Agência Nacional do Petróleo à superfície batimétrica atual. São também exibidos os poços referenciados nos trabalhos de Viviers (1986) e Azevedo et al. (1987), bem como os limites das superfícies paleobatimétricas modeladas (em vermelho). (a) Bacia de Santos. (b) Bacia de Campos. Fator de exagero vertical: 20. Fonte: BDEP (2005).

7.7 Modelagem de outros dados batimétricos e paleobatimétricos

Para melhor contextualizar a evolução das bacias de Campos e Santos durante a expansão do Atlântico Sul, foram modelados ainda dois conjuntos de dados.

A primeira modelagem envolveu o conjunto de mapas paleobatimétricos confeccionados por Sykes et al. (1998), gerados por simulação computacional, que não levam em conta a espessura do pacote sedimentar (Figura 4.8). Conforme foi ilustrado na seção 4.1, estes autores realizaram uma modelagem da paleobatimetria do hemisfério Sul de 110 Ma atrás até o presente, com intervalo de 5 Ma. Os mapas correspondentes às idades 110, 100, 85 e 65 Ma foram escolhidos por ilustrarem a evolução da fisiografia oceânica durante o afastamento entre a América do Sul e a África dentro do período de estudo considerado.

A modelagem 3D dos referidos dados iniciou-se com a digitalização das curvas de nível de cada mapa paleobatimétrico, no ambiente AutoDesk Map. Para cada idade foram criados dois arquivos, um em formato *.dat, contendo as coordenadas XYZ dos pontos digitalizados; e outro *.dxf com a especificação geométrica e topológica da linha de costa de parte da América do Sul e da África.

Em seguida, os arquivos foram importados no ambiente do programa GOCAD. A partir do conjunto de pontos, foi criada uma curva de fecho convexo e com a mesma foi possível gerar uma malha vazada para modelar a paleobatimetria usando os pontos da idade em questão. Além disto, usando o arquivo *.dxf contendo a linha de costa correspondente foi criada uma entidade do tipo curva. Esta foi visualizada e utilizada para remover da superfície paleobatimétrica todos os triângulos pertencentes às regiões continentais, bem como reposicionar eventuais triângulos não alinhados com a linha de costa na malha vazada criada pelo programa. Após este processo de filtragem, estava disponível uma superfície paleobatimétrica delimitada pelas linhas de costa dos continentes americano e sul-africano.

Para amenizar a geometria dos triângulos relativamente estreitos desta malha vazada, a mesma sofreu um processo de refinamento, sendo que os triângulos foram modificados de forma a assumirem uma aparência aproximadamente equilátera (ângulos internos iguais a 30 graus). No GOCAD, este processo é realizado com a ferramenta *Beautify*.

Em seguida, para modelar os continentes foi criado um plano poligonal paralelo ao plano XY e com elevação igual a zero. Seus limites foram definidos usando os extremos da paleobatimetria e linha de costa correspondentes. A superfície do plano foi “recortada” com a linha de costa e as partes da porção marinha (batimetria) foram removidas.

Ao final deste processo, dispunha-se de duas malhas vazadas para cada idade de modelagem: uma para a paleobatimetria e outra para a topografia “achatada”. As duas

superfícies foram juntadas e a entidade resultante sombreada usando uma escala de cores que acompanha a variação paleobatimétrica do Atlântico Sul conforme dados de elevação dos mapas originais (de 6 Km de profundidade até o nível do mar). Por fim, sobre cada superfície foi traçado o conjunto de isolinhas com espaçamento vertical de 0.5 Km, para realçar as estruturas do fundo oceânico (Figura 7.36).

As malhas vazadas construídas foram também sombreadas usando a escala de cores dos domínios batimétricos da Figura 7.21. Com isto, foi possível comparar as regiões de ocorrência dos vários ambientes deposicionais com as informações correspondentes visualizadas para as bacias de Santos e Campos. Houve a necessidade de se “ampliar” a escala de cores incluindo o ambiente abissal. Desta forma, novas cores foram acrescentadas e associadas às profundidades de 2000 a 6000 m (limite inferior dos mapas originais). A visualização resultante da evolução do Atlântico Sul durante o meso-Neocretáceo (Figuras 7.37 a 7.40) utiliza dois pontos e distâncias de observação diferentes. Considera-se, para cada idade, uma visada mais regional seguida de uma aproximação para destacar a área objeto da modelagem paleobatimétrica da região das bacias de Santos e Campos. As superfícies paleobatimétricas de Sykes et al. são mais profundas que as de Viviers e Azevedo et al. deste período, pois não levam em conta a espessura do pacote sedimentar que foi aumentando cada vez mais com o passar do tempo, o que resultou em morfologias cada vez mais discrepantes.

Por fim, visando facilitar a correlação das informações paleobatimétricas do Atlântico Sul com informações lito-cronoestratigráficas das bacias de Santos e Campos, as idades de reconstrução consideradas (110, 100, 85 e 65 Ma) foram posicionadas sobre as respectivas cartas estratigráficas (Figuras 4.12, 4.13 e 7.41a e b). Além disto, as seqüências estratigráficas correspondentes foram indicadas no perfil da Bacia de Santos (Figura 7.41c).

As informações paleobatimétricas de abertura e expansão do Atlântico Sul, obtidas da literatura e modeladas em 3D no GOCAD, constituem importante conjunto de dados para investigações sobre a evolução paleogeográfica da margem continental sudeste brasileira e das bacias de Santos e Campos. A comparação de imagens geradas para idades distintas permite acompanhar a formação e modificação de estruturas oceânicas durante o período de estudo. Por exemplo, a partir de 110 Ma ocorreu a formação de um alto estrutural que surgiu na margem africana e evoluiu para a região central do oceano, em correspondência do Alto de Florianópolis. Os demais mapas do trabalho de Sykes et al. também foram processados e modelados em 3D, para se criar uma animação sobre a evolução da morfologia do fundo oceânico de 110 Ma até o presente.

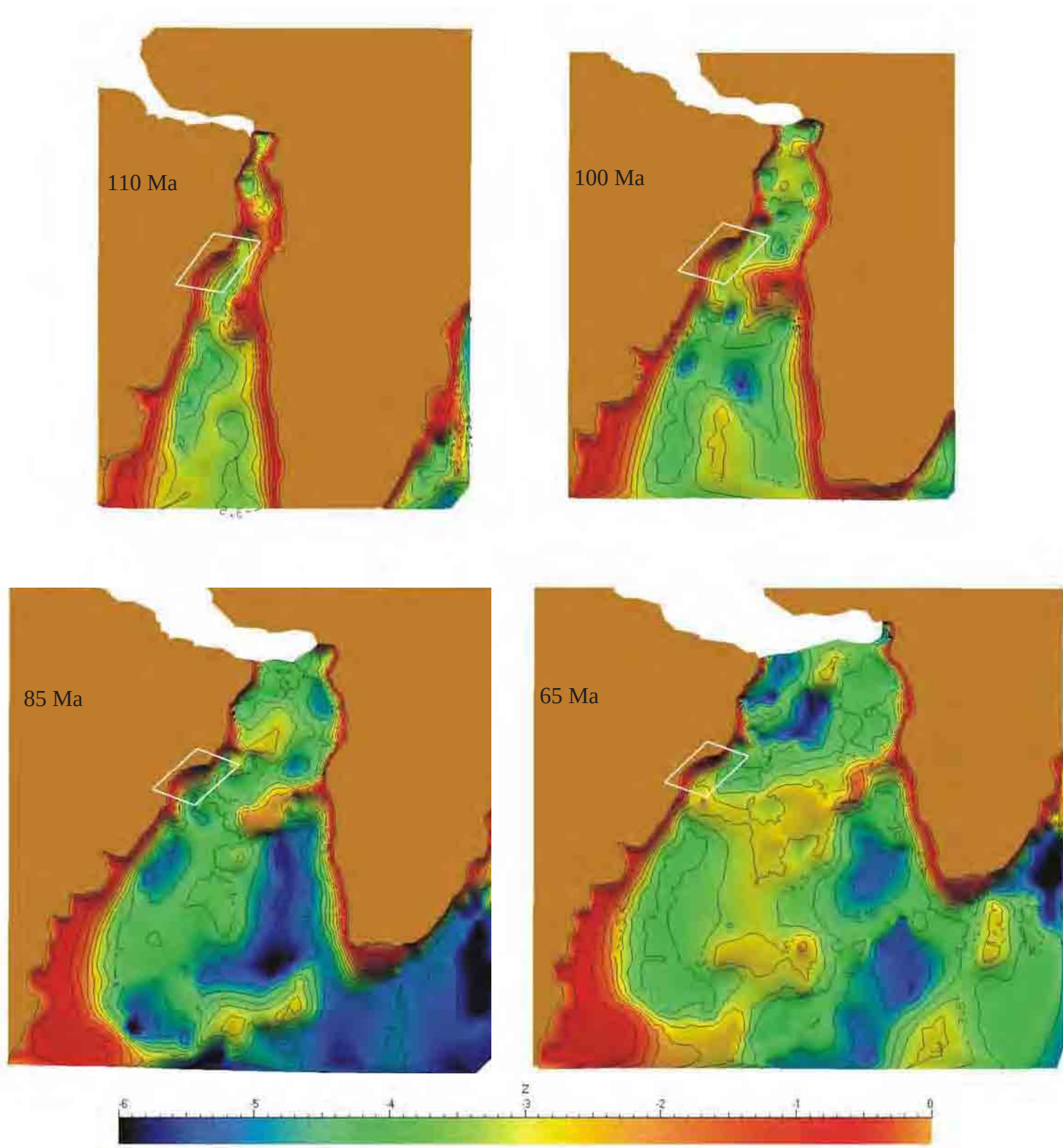


Figura 7.36 - Modelagem 3D da reconstrução computacional da paleobatimetria do Atlântico Sul para o meso-Neocretáceo. As superfícies foram sombreadas usando uma escala de cores do azul escuro até a cor laranja. O polígono branco delimita a região das bacias de Santos e Campos estudada. Fonte dos dados: Sykes et al. (1998).

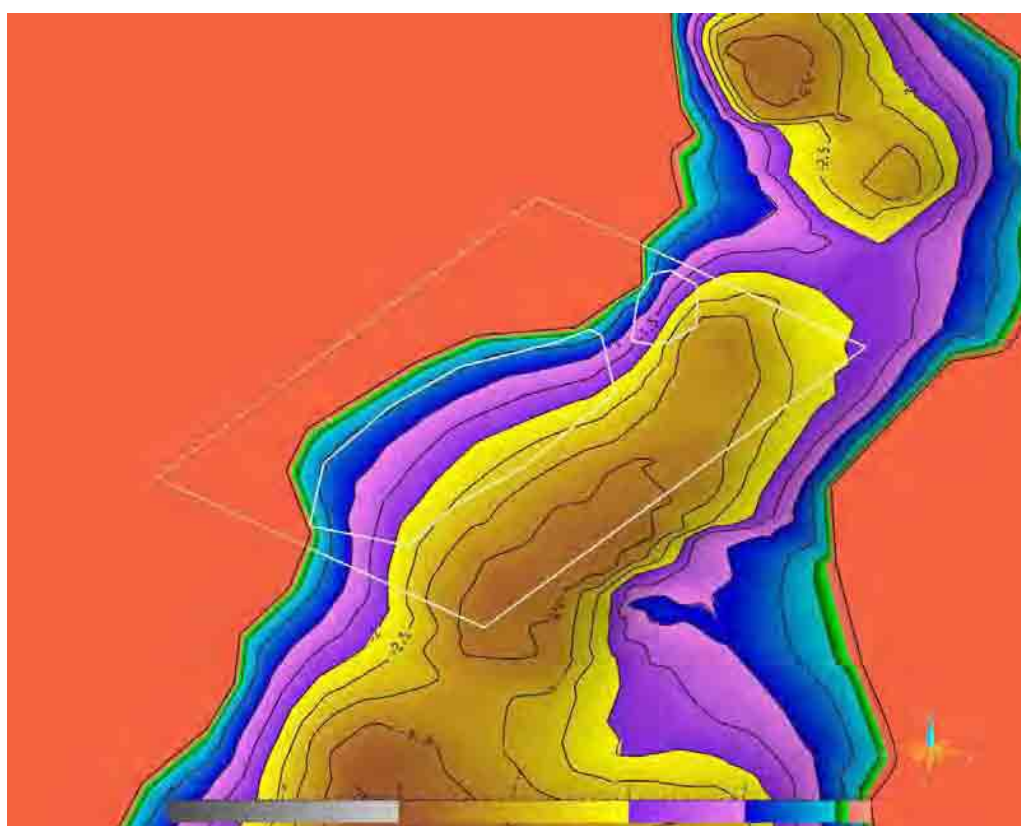
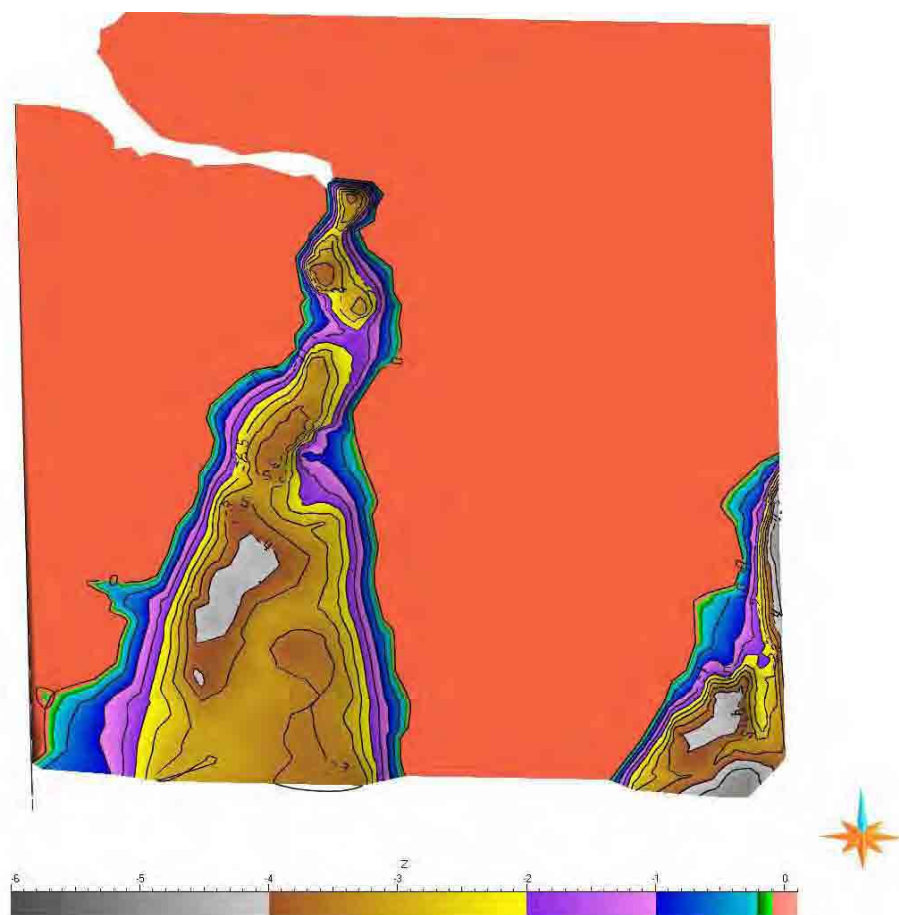


Figura 7.37 – (a) Visualização 3D da paleobatimetria do Atlântico Sul no início do Albiano (110 Ma). (b) Detalhe com o limite da área de estudo e das superfícies paleobatimétricas das bacias de Santos (símbolo “S”) e Campos (símbolo “C”). Adotou-se a escala de cores da Figura 7.21 com inclusão do domínio abissal: do amarelo (-2000 m) para o marrom / cinza claro (-4000 m) até o cinza escuro (-6000 m). Visada para Norte.

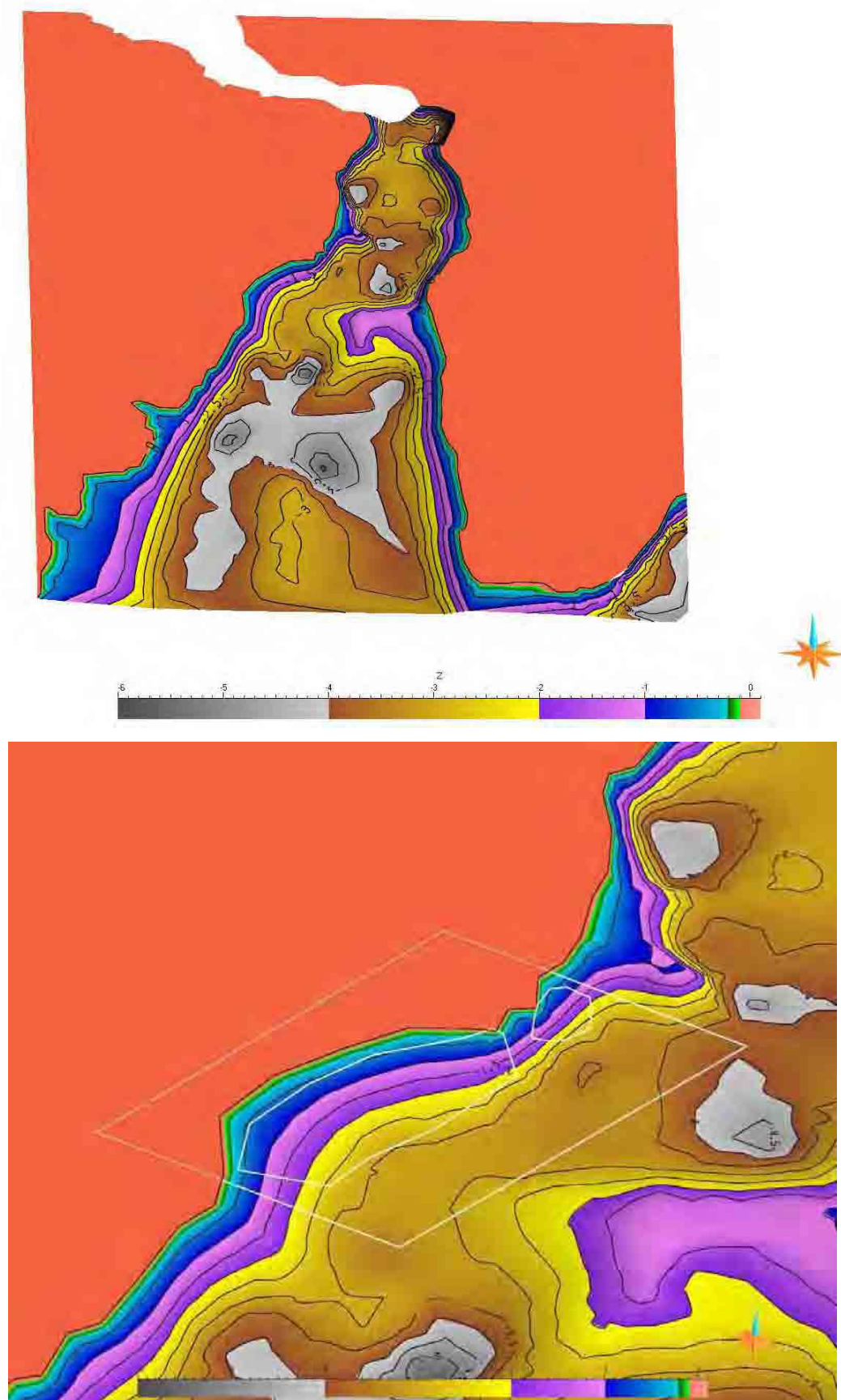


Figura 7.38 – (a) Visualização 3D da paleobatimetria do Atlântico Sul ao “final” do eo-mesoalbian (100 Ma). (b) Detalhe com o limite da área de estudo e das paleobatimetrias modeladas das bacias de Santos e Campos.

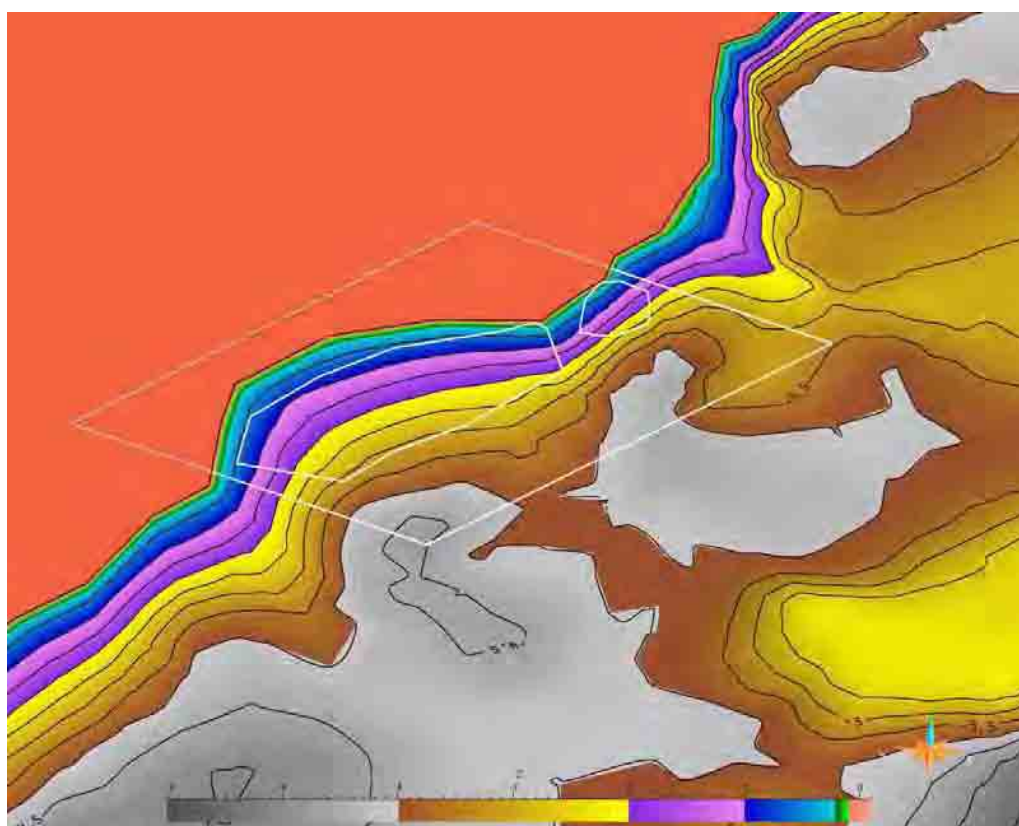
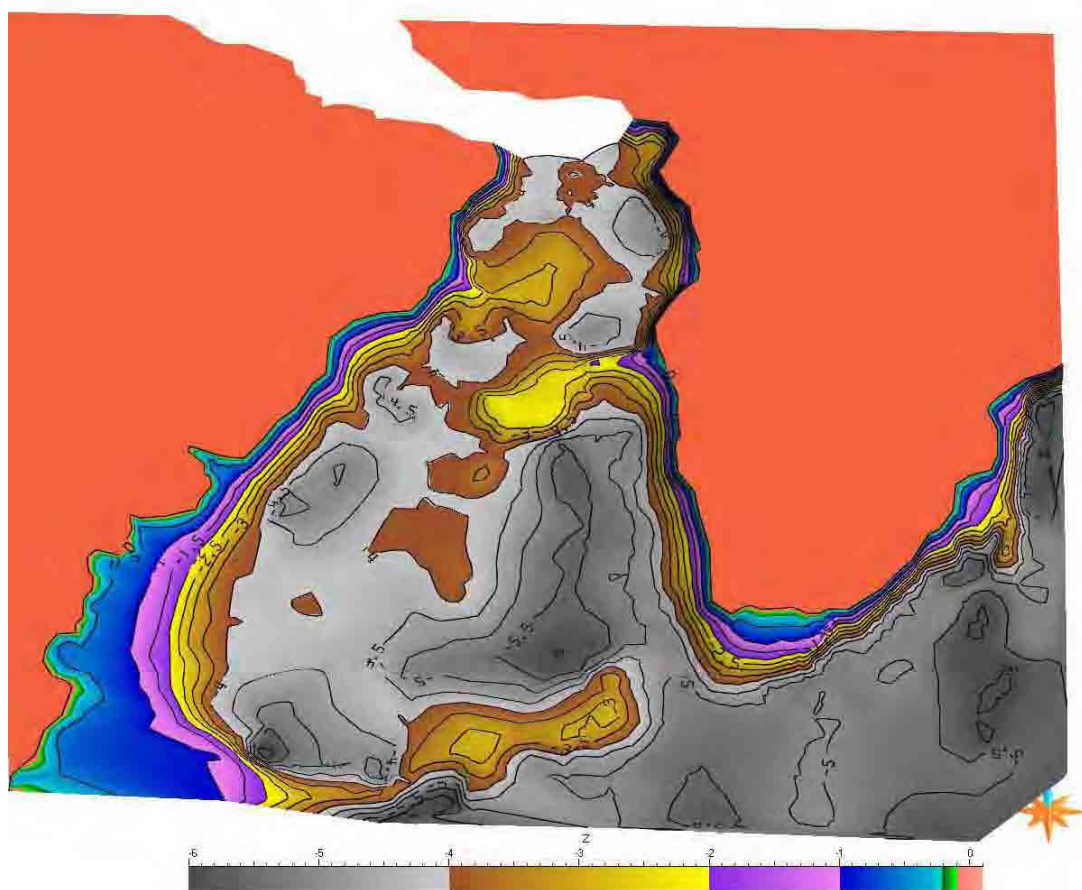


Figura 7.39 – (a) Visualização 3D da paleobatimetria do Atlântico Sul ao “final” do eosantoniano (85 Ma). (b) Detalhe com o limite da área de estudo e das paleobatimetrias modeladas das bacias de Santos e Campos.

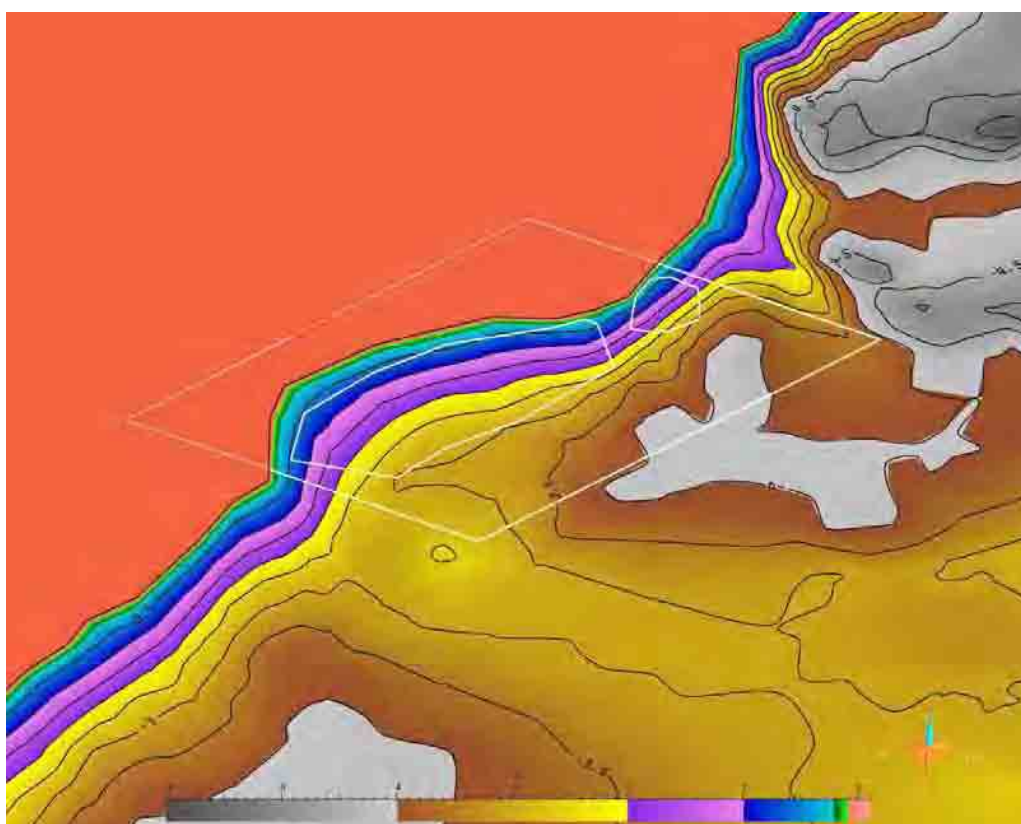
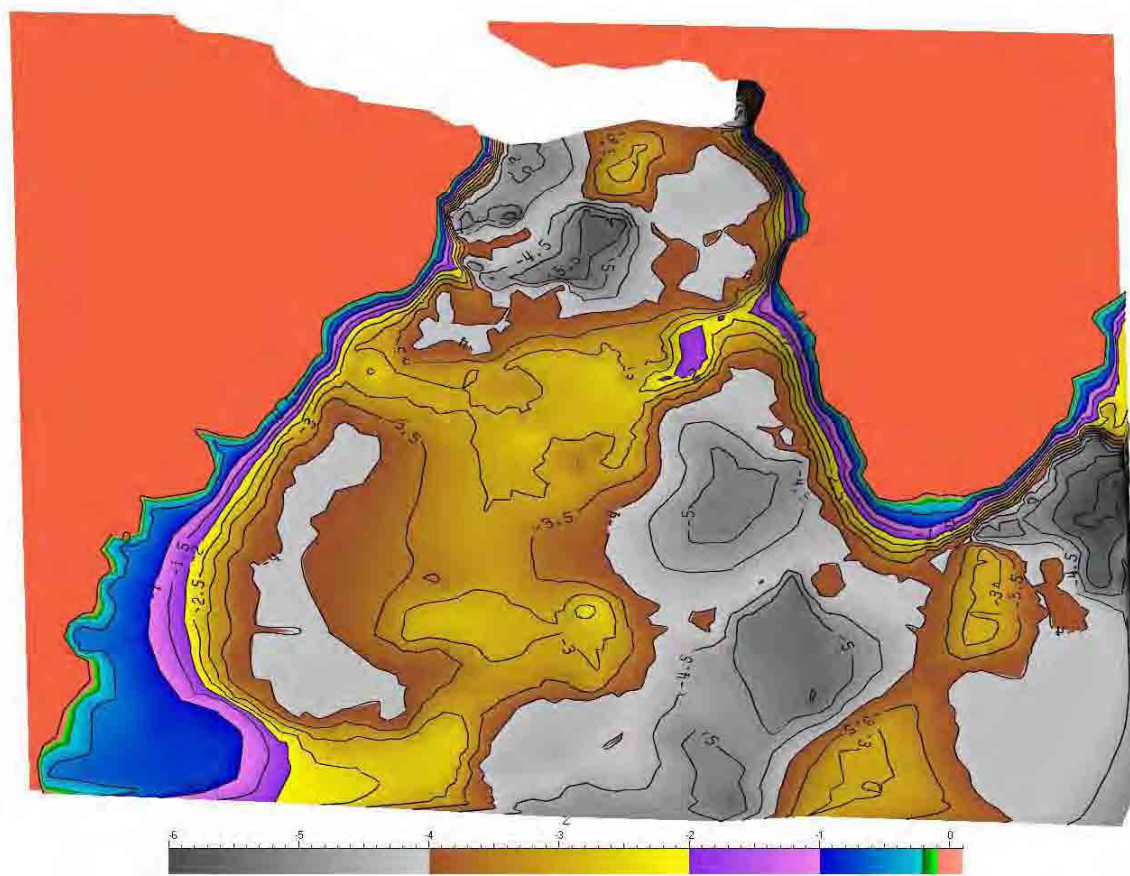


Figura 7.40 – (a) Visualização 3D da paleobatimetria do Atlântico Sul ao “final” do Maastrichtiano (65 Ma). (b) Detalhe com o limite da área de estudo e das paleobatimetrias modeladas das bacias de Santos e Campos.

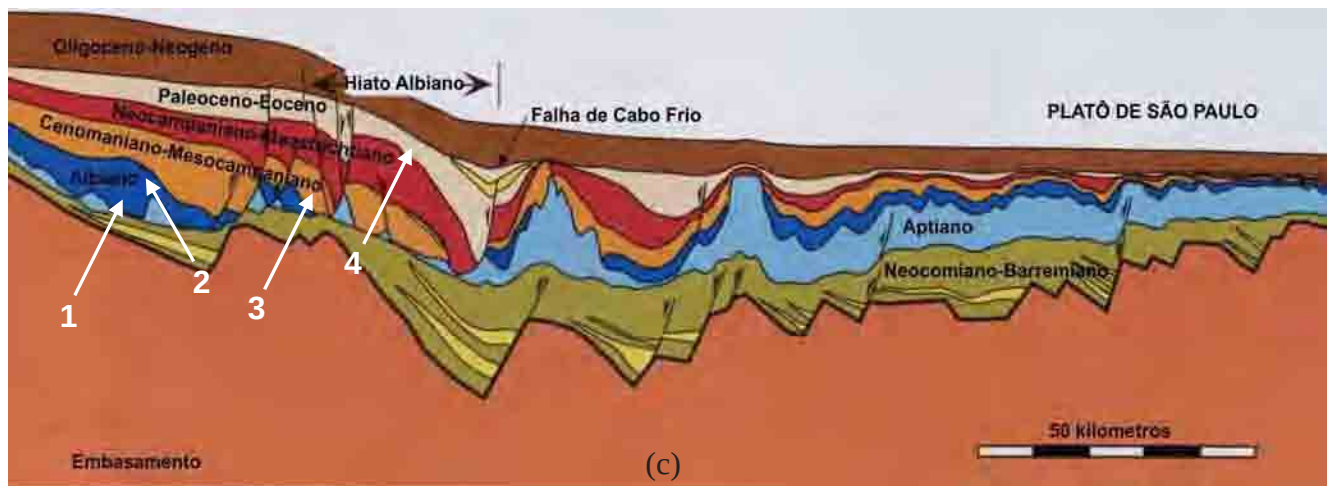
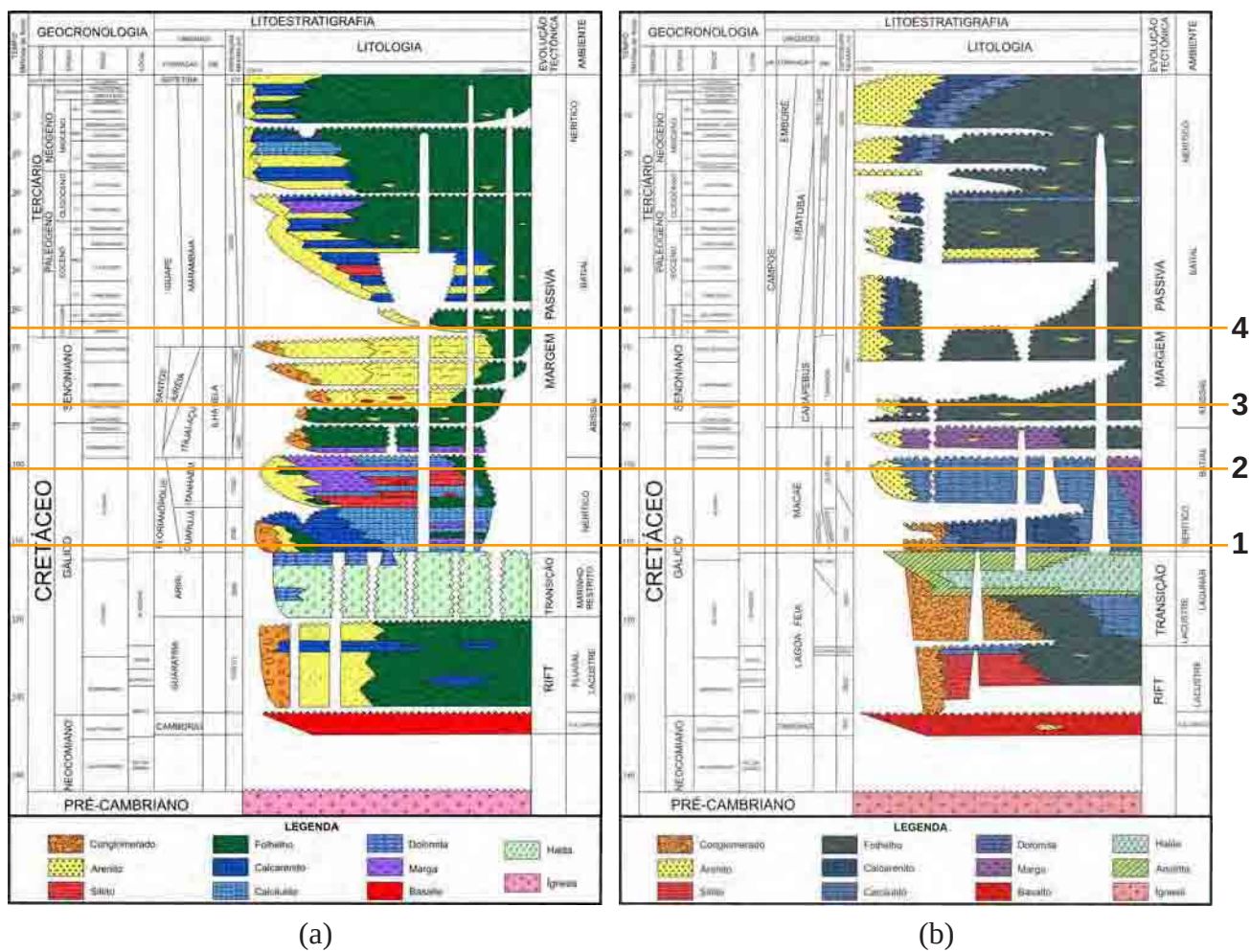


Figura 7.41 – Posicionamento das idades de reconstrução paleobatimétrica do Atlântico Sul (Sykes et al., 1998) sobre as cartas estratigráficas das bacias de Santos (a) e Campos (b). Os números de 1 a 4 correspondem às idades de 110, 100, 85 e 65 Ma. (c) Indicação nas seqüências deposicionais correspondentes no perfil geológico da Bacia de Santos (Modica & Brush, 2004).

Outro conjunto de dados modelados foi um *grid* de pontos de batimetria e topografia da região do Mar Vermelho. Tal conjunto, obtido da rede Internet (projeto TOPEX, 2005), foi escolhido para construir uma analogia em termos morfológicos e deposicionais entre o Mar Vermelho e a situação batimétrica das bacias de Santos e Campos durante o Albiano. Os dados de pontos correspondentes à região foram importados no GOCAD e a partir dos mesmos gerou-se uma malha vazada. Foi também criado um plano horizontal semitransparente para identificar o nível do mar junto à superfície. Esta foi sombreada usando uma escala de cores para destacar as variações altimétricas (Figura 7.42). As elipses representadas na figura correspondem aproximadamente à dimensão e aos ambientes deposicionais das superfícies paleobatimétricas das bacias de Santos e Campos, em comparação com a batimetria do Mar Vermelho.

Para facilitar a comparação dos ambientes deposicionais registrados nas bacias de Santos e Campos durante o período de estudo, a escala de cores das Figuras 7.18 a 7.21, utilizada para destacar os diferentes ambientes nas superfícies paleobatimétricas das referidas bacias, foi também empregada para sombrear a batimetria do Mar Vermelho (Figura 7.43). As elipses foram posicionadas nas áreas que apresentam a melhor semelhança, em termos de ambientes deposicionais, com as bacias de Santos e Campos, deixando de considerar similaridades em termos de extensão. A Bacia de Santos, caracterizada por extensa plataforma na direção distal, é identificada, por analogia, com a região sul do Mar Vermelho, onde o ambiente plataformar é dominante e o gradiente é baixo. A observação das elipses correspondentes na visualização da batimetria do Mar Vermelho sugere que a Bacia de Santos deve ter apresentado ambientes de deposição mais profundos que os identificados por Viviers (1986), que não abrangeram a dimensão total da bacia, uma vez que os poços disponíveis para os estudos na época não atingiram sua porção mais distal.

Já as demais elipses, posicionadas em uma porção mais setentrional do Mar Vermelho, indicam uma região fisiograficamente equivalente a Bacia de Campos, caracterizada por um ambiente plataformar estreito que passa rapidamente para batial superior-médio até atingir o batial inferior, próximo ao limite da crosta oceânica.

Outra similaridade é indicada pelo relevo topográfico adjacente ao Mar Vermelho, em relação à margem continental sudeste brasileira. O relevo é mais alto na porção sul, analogamente à Serra do Mar adjacente à Bacia de Santos (Figura 7.14), e menos pronunciado no setor central do Mar Vermelho, analogamente à porção NW adjacente à Bacia de Campos. Caberia um estudo mais detalhado para pesquisar uma relação genética entre o relevo mais elevado e uma plataforma adjacente mais rasa e com preenchimento sedimentar mais espesso.

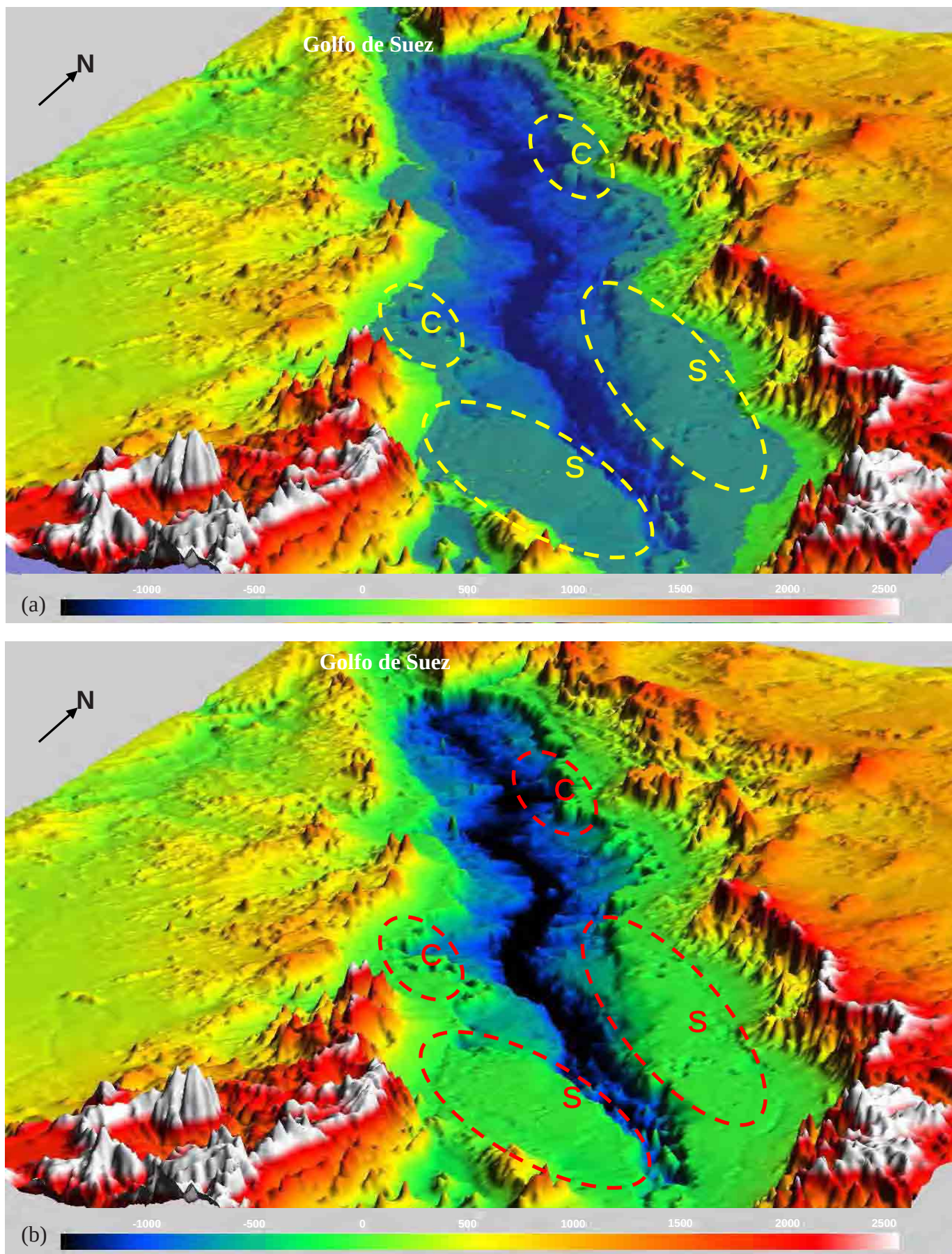


Figura 7.42a-b – Modelagem 3D do relevo e da batimetria atuais do Mar Vermelho com (a) e sem (b) representação do nível do mar. As elipses tracejadas indicam a dimensão relativa dos modelos paleobatimétricos das bacias de Campos (C) e Santos (S) e também delimitam regiões com situação deposicional semelhante. Visada para NW. Fonte dos dados: TOPEX (2005).

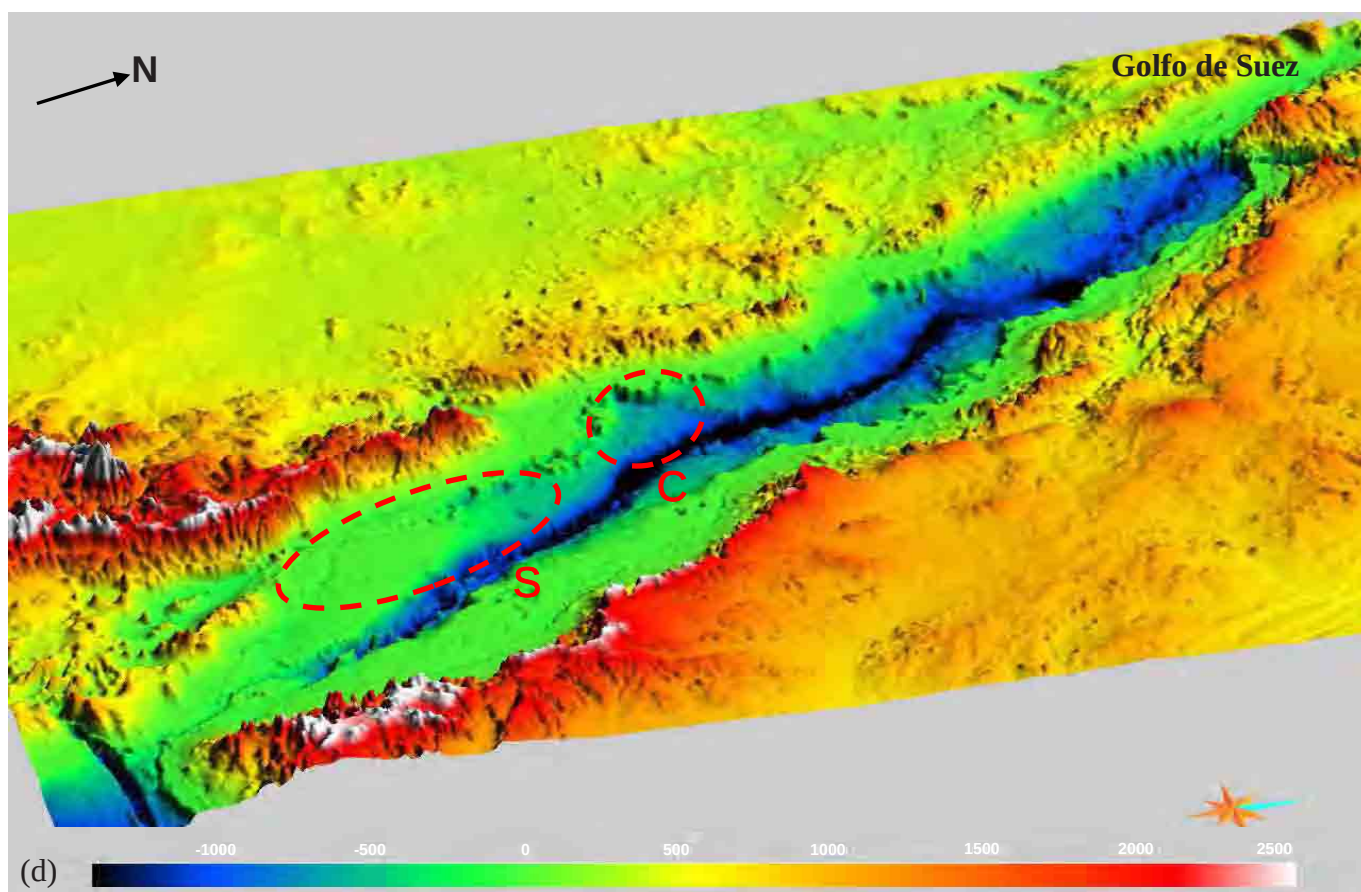
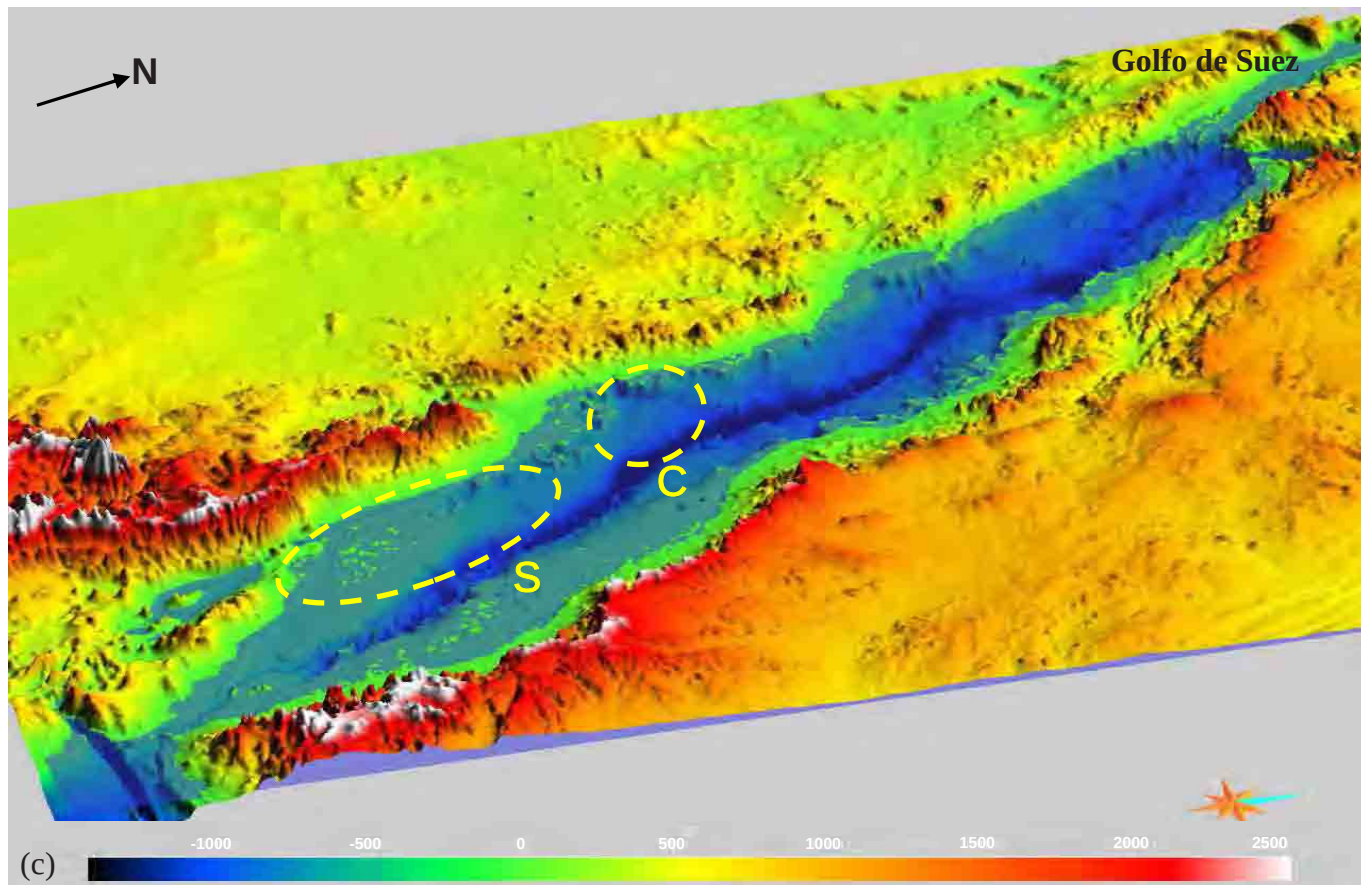


Figura 7.42c-d – Continuação da figura anterior. Visada para W com (c) e sem (d) representação do nível do mar. Além da morfologia batimétrica distinta entre os domínios S e C, a topografia da parte emersa adjacente é também contrastante.

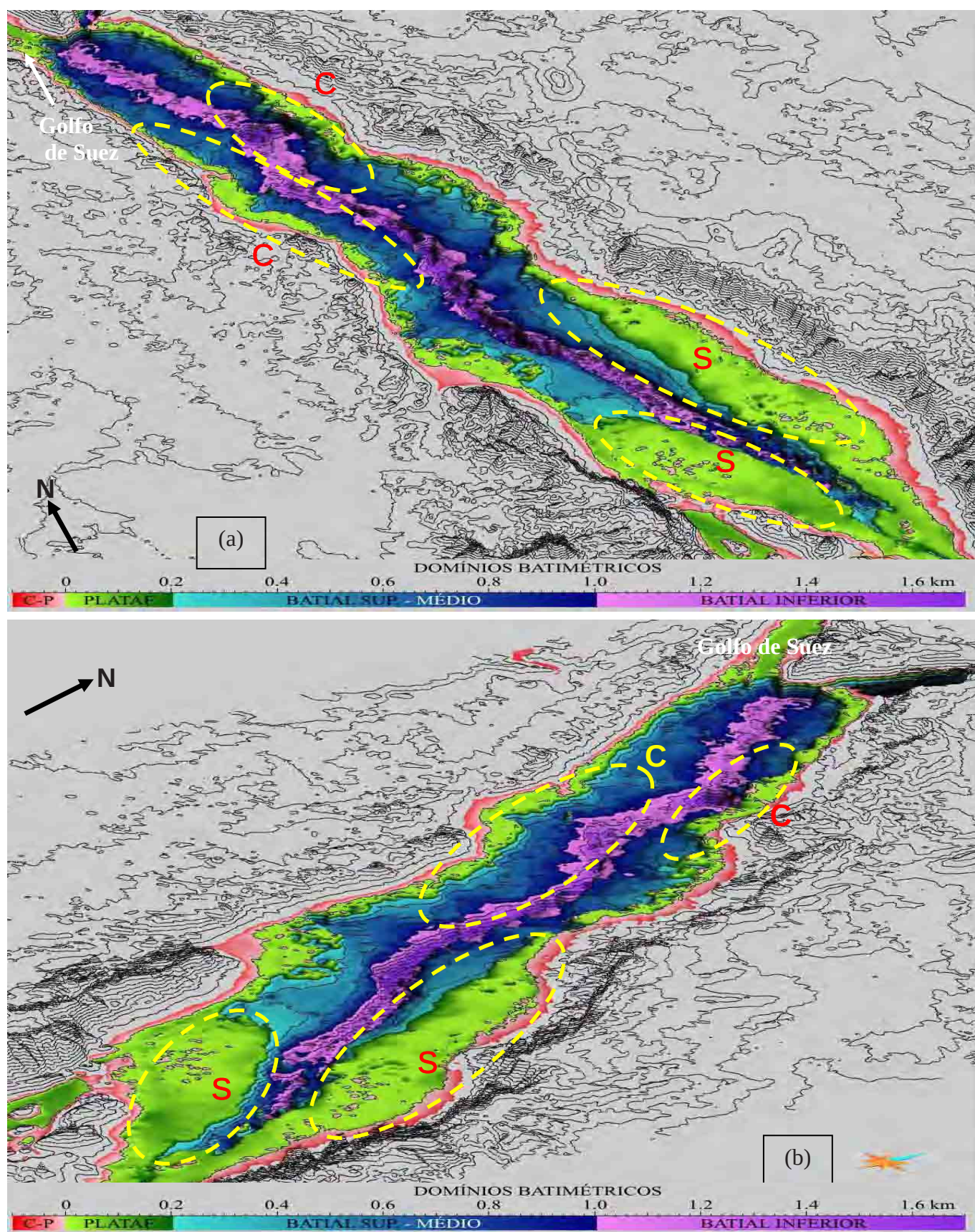


Figura 7.43 – Batimetria e parte do relevo continental com a mesma escala de cores aplicada aos modelos paleobatimétricos das bacias de Campos (C) e Santos (S) para melhor comparar os ambientes de deposição. Legenda das cores: C-P = continental-parálico. Isolinhas no continente com espaçamento vertical de 200 m. As elipses denotam ambientes correlatos aos das bacias de Santos e Campos para o meso-Neocretáceo. Visada para NE (a) e para NW (b).

7.8 Discussão dos resultados

A utilização do programa de modelagem e visualização GOCAD permitiu transpor para um ambiente de estudo 3D os mapas paleobatimétricos confeccionados por Viviers (1986) e Azevedo et al. (1987). Tais informações foram integradas com outros dados tectônicos e litológicos obtidos da literatura, o que permitiu realizar um cruzamento de informações e aumentar a capacidade de interpretação dos conjuntos de dados considerados separadamente. As imagens mostradas nas figuras anteriores contêm elementos para auxiliar no entendimento da história deposicional da porção sudeste da margem continental brasileira durante o meso-Neocretáceo, logo após a fase rift, quando o ambiente de sedimentação passou de continental para marinho inicialmente raso e gradativamente mais profundo (fase drift). Esta evolução foi interrompida na Bacia de Santos pela intensa sedimentação e diminuição da espessura da lâmina d'água a partir do neoturoniano.

Em geral todo o meso-Neocretáceo da região foi caracterizado pela evolução de uma margem continental passiva, sob a influência de eventos tectônicos e variações relativas do nível do mar. A seguir será ilustrada a evolução paleogeográfica e paleoambiental da região em estudo com respaldo nas considerações de Viviers & Azevedo (1988) e na visualização 3D dos dados geológicos modelados.

Durante o eo-mesoalbio (113-103 Ma) ocorreu a sedimentação de carbonatos em ambiente de águas rasas, quentes e oxigenadas, caracterizadas por hipersalinidade (DIAS-BRITO, 1987). Azevedo (2004) também descreve as condições do paleoclima neste período, tecendo considerações distintas para os dois mares restritos Atlântico Sul Central e Atlântico Sul Meridional, localizados, respectivamente, ao norte e ao sul da barreira constituída pelo Alto de Florianópolis e Dorsal de São Paulo (Figura 4.3). No Atlântico Sul Meridional ocorriam temperaturas relativamente mais frias e salinidades mais baixas. A batimetria das bacias de Santos e Campos foi caracterizada pelo ambiente plataformal raso-médio que apresenta um padrão comum às duas bacias (Figura 7.21a). A distribuição dos litotipos correspondentes (Figura 7.27a) indica a presença de arenitos na região proximal e argilitos e calcários na porção distal.

A sequência estratigráfica ao “final” do neo-albiano (97,5 Ma) contém calcilutitos e margas, além de arenitos e conglomerados (porção proximal da bacia de Santos) (Figura 7.41a). Neste período ocorreu a conexão definitiva entre o Atlântico Sul Central e Atlântico Sul Meridional (AZEVEDO, 2004). O ambiente deposicional de ambas as bacias foi o plataformal médio-profundo (Figura 7.21b), sendo que em alguns pontos da Bacia de Santos e sobretudo na Bacia de Campos iniciaram-se condições mais profundas (batial superior). Uma

comparação das figuras que exibem as condições deposicionais deste período com o anterior mostra um deslocamento das curvas de isobatimetria para oeste, fato interpretado pelos autores em termos de um aumento relativo do nível do mar.

Neste período, a bacia de Santos foi caracterizada por condições de deposição mais rasas, devido ao maior aporte de sedimentos terrígenos. A Figura 7.32 exibe uma integração das informações batimétricas e litológicas para a Bacia de Santos, a partir dos trabalhos discutidos neste capítulo. Em particular, a Figura 7.32a permite comparar a posição do horizonte estratigráfico neo-albiano (região azul escura), estimado utilizando o perfil sísmico interpretado da Figura 7.31 (Modica & Brush, 2004), com a profundidade que a plataforma carbonática deve ter apresentado na idade em questão (dados paleobatimétricos de Viviers, 1986). Tal integração constitui poderoso recurso oferecido por ferramentas de visualização integradas como o GOCAD.

Durante o Cenomaniano-mesoturoniano (97,5-89 Ma) ocorreu deposição de margas e folhelhos em condições batimétricas de 100-300 m. Os ambientes encontrados em ambas as bacias são o plataformais-batiais superiores, embora na região nordeste (Bacia de Campos) tenha se verificado ampliação do ambiente batial superior, em comparação com o período anterior (Figura 7.21c).

Na Bacia de Santos as condições relativamente mais rasas foram explicadas como consequência de um maior aporte de sedimentos em relação a sua taxa de subsidência, sobretudo na região sudoeste (Figura 7.27c). Isto promoveu o estabelecimento de condições prográdantes, apesar da elevação generalizada do nível do mar. Ambas as bacias sofreram deposição em condições anóxicas, o que favoreceu a formação de matéria orgânica (Arai, 1987).

Durante o período neoturoniano-eocampaniano (89-81 Ma) as histórias deposicionais das bacias de Santos e Campos divergem. A partir do neoturoniano, o aporte sedimentar na bacia de Santos ultrapassou ou compensou sua taxa de subsidência. Neste período ocorreu a deposição de grande quantidade de sedimentos clásticos na região, relacionada ao soerguimento da Serra do Mar. Estabeleceu-se, assim, um ambiente plataformais raso-médio (Figura 7.21e-f) e em seguida médio-profundo até batiais superiores (Figura 7.21g). Em consequência do elevado aporte sedimentar, formaram-se amplas regiões de ambiente parálico. A Figura 7.33 mostra nova integração de dados batimétricos e litológicos para a bacia. A paleobatimetria exibida refere-se ao “final” do eocampaniano, enquanto o perfil geológico mostra o topo do horizonte estratigráfico mesocampaniano (em laranja). Na Bacia de Campos, cuja estratigrafia para o período é representada por folhelhos, margas e arenitos

turbidíticos, as condições batimétricas já eram bem mais profundas, com estabelecimento progressivo de ambiente batial superior-médio e até inferior (Figura 7.21d-g).

A Figura 7.27d-g exibe a sobreposição do mapa de litotipos às superfícies paleobatemétricas do neoturoniano-eocampaniano. Observa-se a concentração de arenito na porção proximal da Bacia de Santos, sendo a região caracterizada por baixo gradiente. Este litotipo é também encontrado em regiões mais distais, em consequência do aporte sedimentar promovido por sistemas deltáicos e/ou correntes de turbidez. A restante porção distal é caracterizada por argilitos. A Bacia de Campos, com fisiografia bem mais íngreme, apresenta também arenitos na porção proximal e folhelhos na região distal, o que não está representado no mapa de França et al. (1995). Os calcários e margas representados neste mapa e projetados sobre as superfícies paleobatemétricas do neoturoniano-eocampaniano, em ambientes batiais médio-inferior, cessaram sua deposição ao final do Cenomaniano. A inconsistente representação de rochas carbonáticas sobre as paleosuperfícies mais profundas decorre do fato do mapa geológico mencionado abranger um período muito longo e não mostrar com mais detalhe a variação temporal da sedimentação.

A diferença no padrão de sedimentação das duas bacias permaneceu durante o neocampaniano-Maastrichtiano (76-65 Ma). Na Bacia de Santos, o elevado aporte de sedimentos clásticos continuou, embora as condições batimétricas fossem ligeiramente mais profundas em relação ao período anterior. Além disto, o deslocamento para oeste das curvas isobatemétricas mostra ambientes de deposição mais íngremes. A Figura 7.34 exibe a integração de dados litológicos e paleobatemétricos para o “final” do Maastrichtiano.

Na Bacia de Campos, por outro lado, consolidou-se ambiente de deposição mais profundo, predominando o meio batial médio-inferior (7.21h). A seqüência estratigráfica correspondente ao período é caracterizada essencialmente por folhelhos e, de forma menos expressiva, arenitos na porção proximal (Figura 7.41b).

Os modelos gerados com os dados paleobatemétricos de Sykes et al. (1998) para as mesmas idades e usando a mesma escala de cores (Figuras 7.37 a 7.40) indicam, desde o início, condições batimétricas mais profundas que os modelos descritos acima por considerarem as profundidades dos substratos das bacias e não as do topo de cada seqüência depositada na época. Estes modelos foram reproduzidos nesta dissertação com a finalidade de comparar o estágio do afastamento entre a América do Sul e a África para cada idade considerada nas superfícies paleobatemétricas.

Por fim, a modelagem geométrica do Mar Vermelho (Figuras 7.42 e 7.43) permitiu criar uma possível analogia entre alguns de seus compartimentos e a paleogeografia das

bacias de Santos e Campos para as idades modeladas. Permitiu mostrar também que, no caso da Bacia de Santos, o relevo deveria ter atingido profundidades maiores para cada idade do que pôde ser reconstituído a partir dos dados obtidos na época através de plataformas de perfuração mais restritas às porções rasas da bacia.

8 Conclusões

Esta dissertação de mestrado procurou demonstrar o potencial da aplicação de técnicas de modelagem e visualização 3D à representação de dados geológicos e simulações, em especial em estudos tectônicos e de bacias. Destaca-se a importância de ferramentas que permitem a interação do pesquisador com seus dados. Isto pode envolver a manipulação em tempo real de estruturas modeladas, com a qual é possível aferir propriedades e relações entre partes de um conjunto de difícil identificação nos tradicionais ambientes de trabalho 2D. Foram obtidos resultados em três frentes de trabalho como sumarizado a seguir.

A preparação de um ambiente de desenvolvimento de aplicativos de modelagem e visualização 3D usando programas e bibliotecas *open source* constituiu uma importante etapa para a construção de ferramentas de processamento e visualização de dados geológicos em ambientes de pesquisa. Poupano-se recursos financeiros para o pagamento de licenças de utilização de sistemas operacionais ou de aplicativos, os recursos à disposição da instituição podem ser destinados exclusivamente à aquisição dos equipamentos computacionais. Tal situação não é ideal apenas para ambientes de pesquisa acadêmicos, mas também para a própria indústria do petróleo.

Além disto, o caráter aberto das ferramentas de software usadas, bem como daquelas implementadas, proporciona a divulgação dos ambientes computacionais e dos programas utilizados em diferentes centros de trabalho, agilizando a troca de informações e a discussão dos resultados parciais obtidos.

O paradigma baseado em soluções abertas demanda certo esforço no sentido de integrar as ferramentas preexistentes ou criar novos aplicativos. Mesmo assim, o grau de flexibilidade alcançável permite obter resultados extremamente satisfatórios, já que todas as funcionalidades das ferramentas implementadas podem ser ajustadas em detalhe, conforme as necessidades do pesquisador. Estas ferramentas podem também ser amplamente empregadas para a resolução de problemas não previstos em aplicativos comerciais, mesmo de elevado custo de utilização.

Exemplo de aplicação desenvolvida utilizando um ambiente totalmente *open source*, o programa *Tensor3D* permitiu simular a representação em 3D da deformação de objetos geométricos a partir da utilização de tensores de tensão e deformação. A modificação interativa das componentes de um tensor matemático definido pelo usuário e a conseqüente

deformação em tempo real de objetos tridimensionais representando corpos geológicos possibilitam observar, estudar e distinguir a atuação das componentes de cisalhamento simples e puro sobre entidades 3D, em cada um dos planos cinemáticos (XY, XZ e YZ).

A modelagem de diferentes objetos complexos construídos simulando agregados mono- e poliminerálicos como elementos esféricos, segmentos de linhas e elipses com orientações diversas, fornece um instrumento valioso de observação dos processos deformacionais que envolvem corpos geológicos. Para tanto, foram empregadas ferramentas de modelagem geométrica que permitem o estudo do comportamento de diferentes entidades em regimes de deformação simulados no contexto do programa *Tensor3D*. Tais recursos também foram aplicados a entidades que simulam a morfologia de bacias sedimentares, construídas a partir de dados sísmicos e importadas de outros ambientes de trabalho 3D. Além disto, a possibilidade de armazenar e restaurar estados intermediários de deformação dos objetos visualizados permite isolar e estudar as várias etapas que caracterizam a deformação progressiva de rochas.

Finalmente, o alto grau de interatividade proporcionado na visualização e deformação de modelos geométricos torna o programa uma ferramenta adequada tanto para finalidades didáticas, como para o entendimento das deformações *em três dimensões* em projetos de pesquisa que envolvem a deformação de rochas, tanto do embasamento cristalino como de bacias sedimentares, inclusive em escala de reservatórios.

A modelagem 3D de informações paleobatimétricas das bacias de Santos e Campos, obtidas a partir de mapas de isovalores para o meso-Neocretáceo, realizada com o programa GOCAD, aumentou as capacidades de análise e interpretação de dados dispersos e disponíveis na literatura. Visualizando várias superfícies simultaneamente e com sombreamentos adequados para diferenciar os distintos ambientes deposicionais, foi possível identificar relações entre várias idades de evolução para cada bacia. Foram também empregadas outras ferramentas de visualização científica e computação gráfica (tais como escala de cores, definição de exagero vertical, modificação do ponto de observação, etc.) para realçar a variação morfológica das estruturas investigadas ao longo do tempo.

As superfícies paleobatimétricas referentes às duas bacias foram também visualizadas de forma simultânea, o que permitiu identificar semelhanças e diferenças no padrão de sedimentação destes ambientes ao longo do tempo geológico.

Foram também extraídas entidades geométricas a partir das superfícies paleobatimétricas modeladas. Isto permitiu confeccionar animações que mostram, por

exemplo, o deslocamento da linha de costa do Albiano ao Maastrichtiano, fornecendo um instrumento de auxílio ao estudo das variações paleoambientais ocorridas na região.

A aplicação de ferramentas de modelagem geométrica 3D a superfícies paleobatimétricas de diferentes idades permitiu a construção de modelos de morfologia “intermediária”, visando criar animações que mostram uma variação mais gradual da batimetria nas bacias de Santos e Campos. Estas animações permitirão uma análise mais completa dos processos e da evolução tectônica da margem continental em apreço.

As informações paleobatimétricas foram integradas com outros dados de ordem litológica, batimétrica e estrutural obtidos da literatura, visando prover um ambiente unificado de análise da evolução paleoambiental da margem continental sudeste brasileira, onde também poderão ser incorporados perfis sísmicos e poços. Esta integração aumenta o potencial de interpretação dos dados originais, devido à facilidade de junção e gerenciamento de informações advindas de fontes distintas, e da possibilidade de interagir com informações visualizadas em 3D.

A construção de superfícies batimétricas do Atlântico Sul do mesocretáceo ao recente e do Mar Vermelho permite contextualizar melhor a evolução paleogeográfica destas bacias durante a abertura do Atlântico Sul e representar analogias em ambientes de deposição atuais. Estas informações podem ser também utilizadas para tecer considerações a respeito dos paleoambientes que caracterizavam as bacias de Santos e Campos em regiões mais distais, para as quais não havia disponibilidade de poços nos trabalhos originais de Viviers (1986) e Azevedo et al. (1987) utilizados na modelagem paleobatimétrica.

Em síntese, a modelagem 3D destes dados permitiu realçar:

- A diferente história deposicional que caracterizou as bacias de Santos e Campos a partir do neoturoniano, sendo que a Bacia de Campos apresentou relevo mais profundo e íngreme até o final do Maastrichtiano.
- Uma compartimentação da batimetria atual do Mar Vermelho semelhante à fisiografia das bacias de Santos e Campos durante o meso-Neocretáceo. Em particular, a região sul é caracterizada por plataforma mais extensa e batimetria mais suave, ao passo que a porção norte apresenta plataforma mais estreita e fisiografia íngreme.
- Uma compartimentação do relevo emerso do Mar Vermelho semelhante à da margem continental sudeste brasileira. O relevo é mais alto na porção sul, analogamente à Serra do Mar adjacente à Bacia de Santos, que durante o meso-Neocretáceo foi caracterizada por elevado aporte sedimentar; e menos pronunciado na região central, analogamente à porção NW adjacente à Bacia de Campos, onde o volume de sedimentos depositados foi menor.

- A provável existência de ambientes de deposição mais profundos (batial inferior) na Bacia de Santos, baseado na analogia com regiões fisiograficamente semelhantes da batimetria do atual Mar Vermelho.

As técnicas de visualização científica descritas e aplicadas nesta dissertação, ao permitirem a modelagem e visualização de dados geológicos, demonstram a importância das ferramentas computacionais para subsidiar investigações científicas baseadas em conjuntos de dados extensos e diversificados. A complexidade dos dados obtidos em campo ou simulados, aliada à crescente exigência de integração entre informações distintas, destaca a importância de se implantar ambientes computacionais de integração e estudo, além de consolidar as ferramentas de modelagem e visualização 3D como instrumento imprescindível à análise de processos, estruturas e relações de caráter geológico.

9 Referências bibliográficas

- ANA. Disponível em: <<http://www.hidroweb.ana.gov.br/HidroWeb>>. Acesso em: 15 dez. 2004. HidroWeb: Sistema de Informações Hidrológicas.
- ANJUTA DEVSTUDIO. Disponível em: <<http://anjuta.sourceforge.net>>. Acesso em: 15 jul. 2005. Ambiente *open source* de desenvolvimento integrado de programas em C/C++ para a plataforma Linux.
- ANTUNES, P. T. S.; BENGALY, A. P.; CONCEIÇÃO, D. L. R. Three-dimensional geometric modeling of sedimentary basins using GOCAD. In: GOCAD MEETING, 24., 2004, Nancy. **Proceedings...** Nancy: Nancy School of Geology, 2004. p. 1-15.
- ARAI, M. Geochemical reconnaissance of the Mid-cretaceous anoxic event in the Santos Basin, Brazil. **Revista Brasileira de Geociências**, São Paulo, v. 18, n. 3, p. 273-282, 1988.
- ARCGIS. Disponível em: <<http://www.esri.com/software/arcgis>>. Acesso em: 4 ago. 2005. Programa para implementação de Sistemas de Informações Geográficas.
- AUTODESK MAP. Disponível em: <www.autodesk.com/map>. Acesso em: 10 ago. 2005. Digitalização e integração de dados em Sistemas de Informação Geográficos.
- AZEVEDO, R. L. M.; GOMIDE, J.; VIVIERS, M. C. Geo-história da Bacia de Campos, Brasil, do Albiano ao Maastrichtiano. **Revista Brasileira de Geociências**, São Paulo, v. 17, n. 2, p. 139-146, 1987.
- AZEVEDO, R. L. M. **O Albiano no Atlântico Sul: estratigrafia, paleoceanografia e relações globais**. 2001. 2 v. Tese (Doutorado em Geociências) – Instituto de Geociências, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2001.
- AZEVEDO, R. L. M. Paleoceanografia e a evolução do Atlântico Sul no Albiano. **Boletim de Geociências da Petrobras**, Rio de Janeiro, v. 12, n. 2, p. 231-249, 2004.
- BDEP. Disponível em: <<http://www.bdep.gov.br>>. Acesso em: 13 nov. 2004. Banco de dados de Exploração e Produção.
- CAINELLI C.; MOHRIAK, W. U. Geology of Atlantic Eastern Brazilian basins. In: AAPG INTERNATIONAL CONFERENCE AND EXHIBITION, 1998, Rio de Janeiro. **Short Course Notes...** Rio de Janeiro: AAPG, 1998. p. 1-67.
- CHANG, H. K.; KOWSMANN, R.; FERREIRA, A. M.; BENDER, A. A. Tectonics and stratigraphy of the East Brazil Rift System: an overview. **Tectonophysics**, Amsterdam, v. 213, p. 97-138, 1992.
- CMAKE. Disponível em: <<http://www.cmake.org>>. Acesso em: 19 jul. 2004. Ferramenta multi-plataforma para compilação de projetos.

CONECTIVA LINUX. Disponível em: <<http://www.conectiva.com.br>>. Acesso em: 19 jul. 2005. Distribuição brasileira do sistema Linux.

COREL DRAW. Disponível em: <<http://www.corel.com>>. Acesso em: 15 dez. 2004. Aplicativo para editoração e digitalização eletrônica.

CPRM. Disponível em: <<http://www.cprm.gov.br>>. Acesso em: 15 dez. 2004. Serviço Geológico do Brasil.

DEBIAN. Disponível em: <<http://www.debian.org>>. Acesso em: 19 jul. 2005. Distribuição do sistema Linux voltada para desenvolvimento e pesquisa em sistemas operacionais.

DIAS-BRITO, D. A Bacia de Campos no Mesocretáceo: uma contribuição à paleoceanografia do Atlântico Sul primitivo. **Revista Brasileira de Geociências**, São Paulo, v. 17, n. 2, p. 162-167, 1987.

EARTH RESOURCES MAPPER. Disponível em: <<http://www.ermapper.com/ermapper>>. Acesso em: 25 ago. 2005. Aplicativo para junção e processamento de imagens de satélite.

EBERT, H. D.; LAVORANTE, L. P. Visualização 3D da paleobatimetria da Bacia de Campos com o GOCAD. In: SIMPÓSIO DE QUANTIFICAÇÃO EM GEOCIÊNCIAS, 8., 2000. Rio Claro. **Boletim de Resumos Expandidos...** Rio Claro: UNESP, 2000. p. 81-85.

FEDORA CORE 4. Disponível em: <<http://fedora.redhat.com>>. Acesso em: 19 jul. 2005. Distribuição gratuita do sistema Linux mantida pela empresa Red Hat.

FOLEY, J. D.; van DAM, A.; FEINER, S. K.; HUGHES, J. F. **Computer Graphics: principles and practice**. Boston: Addison-Wesley, 1996. 1150 p.

FRANÇA, A. B.; MILANI, E. J.; SCHNEIDER, R. L.; LÓPEZ, O. P., LÓPEZ, J. M.; SUÁREZ R. S.; SANTA ANA, H.; WIENS, F.; FERREIRO, O.; ROSSELLO, E. A.; BIANUCCI, H. A.; FLORES, R. F. A.; VISTALLI, M. C.; FERNANDEZ-SEVESO, F.; FUENZALIDA, R. P.; MUÑOZ, N. Phanerozoic correlation in southern South América. In: TANKARD, A. J.; SUÁREZ SORUCO, R.; WELSINK, H. J. (Eds.). **Petroleum Basins of South America**. Tulsa: AAPG, 1995. p. 129-162. (AAPG Memoir, 62)

FREEBSD. Disponível em: <<http://www.freebsd.org>>. Acesso em: 19 jul. 2005. Distribuição do sistema Linux voltada para segurança de sistemas.

GARCIA, S. F. M. **Estudo tridimensional de efeitos da halocinese em margens passivas**. 1999. 1 v. Dissertação (Mestrado em Evolução Crustal e Recursos Minerais) – Escola de Minas, Universidade Federal de Ouro Preto, Ouro Preto, 1999.

GCC. Disponível em: <<http://gcc.gnu.org>>. Acesso em: 15 jul. 2005. Compilador de código aberto para as linguagens C / C++ em ambiente UNIX.

- GJELBERG, J. Morphology of Oligocene and Miocene Turbidite Channel Systems: Block 434, OffShore Angola. In: MOHRIAK, W.U. (Ed.) **Rifted sedimentary basins of South Atlantic: turbidite reservoir, sedimentation and tectonic process**. Rio de Janeiro: Petrobrás;UERJ, 2003. 1 CD-ROM.
- GNU. Disponível em: <<http://www.gnu.org>>. Acesso em: 19 jul. 2005. Filosofia de desenvolvimento de software de código aberto.
- GNU MAKE. Disponível em:
<http://www.gnu.org/software/make/manual/html_mono/make.html>. Acesso em: 15 jul. 2005. Aplicativo de código aberto para a compilação de projetos complexos usando arquivos de controle *Make*.
- GNU TOOLKIT. Disponível em: <<http://www.gtk.org>>. Acesso em: 15 jul. 2005. Biblioteca de código aberto para desenvolvimento de interfaces gráficas.
- GTK+ USER INTERFACE BUILDER. Disponível em < <http://glade.gnome.org>>. Acesso em 15 jul. 2005. Editor de interfaces gráficas para as bibliotecas GTK / GTK+.
- GOCAD. Disponível em: <<http://www.earthdecision.com>>. Acesso em: 13 jul. 2005. Programa para modelagem e visualização de dados geológicos.
- GTOPO30. Disponível em <<http://edcdaac.usgs.gov/gtopo30/gtopo30.asp>>. Acesso em 15 dez. 2004. Global 30 Arc Second Elevation Data Set.
- HARLAND, W. B.; COX, A. V.; LLEWELLYN, P. G.; PICKTON, C. A. G.; SMITH, A. J.; WALTERS, R. **A geologic time scale**. Cambridge: Cambridge University Press, 1982. 130 p.
- HUME LINEAR ALGEBRA TCL PACKAGE, LA. Disponível em < <http://www.hume.com/la>>. Acesso em 20 jul. 2005. Biblioteca de funções de álgebra linear em Tcl.
- HOBBS, B. E.; MEANS, W. D.; WILLIAMS, P. F. **An outline of structural geology**. New York: John Wiley & Sons, 1976. 560 p.
- KOUTSOUKOS, E. A. M. A área noroeste da Bacia de Campos, Brasil, do Mesocretáceo ao Neocretáceo: evolução paleoambiental e paleogeográfica pelo estudo de foraminíferos. **Revista Brasileira de Geociências**, São Paulo, v. 17, n. 2, p. 168-172, 1987.
- LANZARINI, W. L.; GUIMARÃES, A. O. Modelagem geológica 3D de reservatórios turbidíticos canalizados com os programas GOCAD e PETBOOL. In: SIMPÓSIO DE QUANTIFICAÇÃO EM GEOCIÊNCIAS, 8., 2000, Rio Claro. **Boletim de Resumos Expandidos...** Rio Claro: UNESP, 2000. p. 113-116.

- LAVORANTE, L. P.; EBERT, H. D.; SALAMUNI, E. Visualização de dados volumétricos com o programa VIS5D. In: SIMPÓSIO DE QUANTIFICAÇÃO EM GEOCIÊNCIAS, 8., 2000, Rio Claro. **Boletim de Resumos Expandidos...** Rio Claro: UNESP, 2000. p. 121-123.
- LINDENBECK, C.; EBERT, H. D.; ULMER, H.; LAVORANTE, L. P.; PFLUG, R. Tricut: a program to clip triangle meshes using the Rapid and Triangle libraries and the Visualization Toolkit. **Computers & Geosciences**, New York, v. 28, n. 7, p. 841-850. 2002.
- LINUX. Disponível em: <<http://www.linux.org>>. Acesso em: 19 jul. 2005. Sistema operacional de código aberto baseado em sistemas UNIX comerciais.
- LINUX KERNEL ARCHIVES. Disponível em: <<http://kernel.org>>. Acesso em: 18 jul. 2005. Módulo principal do sistema operacional Linux.
- MALLET, J. L. Discrete Smooth Interpolation. **ACM-Transactions on Graphics**, New York, v. 8, n. 2, p. 121-144. 1989.
- MANDRAKELINUX. Disponível em: <<http://www.mandrivalinux.com>>. Acesso em: 19 jul. 2005. Distribuição francesa do sistema operacional Linux.
- MENDONÇA, J. M.; ARAGÃO, M. A. N. F.; MORAIS, A. Visualização 3D da Bacia do Reconcavo. In: SIMPÓSIO NACIONAL DE ESTUDOS TECTÔNICOS, 8., Recife. **Anais...** Recife: SBG, 2001. p. 367-370.
- MODICA, C. J.; BRUSH, E. R. Postrift sequence stratigraphy, paleogeography and fill history of the deep-water Santos basin, offshore southeast Brazil. **AAPG Bulletin**, Tulsa, v. 88, n. 7, p. 923-945, 2004.
- MOHRIAK, W. U.; BARROS, A. Z.; FUJITA, A. Magnetismo e tectonismo cenozóicos na região de Cabo Frio, RJ. In: CONGRESSO BRASILEIRO DE GEOLOGIA, 37., Natal. **Anais...** Natal: SBG, 1990. v. 6, p. 2873-2885.
- MOHRIAK, W. U. Rift architecture and salt tectonics of the South Atlantic sedimentary basins. In: MOHRIAK, W.U. (Ed.) **Rifted sedimentary basins of South Atlantic: turbidite reservoir, sedimentation and tectonic process**. Rio de Janeiro: Petrobrás;UERJ, 2003. 1 CD-ROM.
- MORAES, A. **Mecânica do contínuo para a geologia estrutural**. Rio de Janeiro: CENPES/Petrobrás, 2000. 88 p.
- MOVIE.BYU FILE FORMAT. Disponível em: < <http://lc.cray.com/doc/movie>>. Acesso em: 19 jul. 2005. Formato para representação de modelos geométricos poligonais.

- NÜRNBERG, D.; MÜLLER, R. D. The tectonic evolution of South Atlantic from Late Jurassic to present. **Tectonophysics**, Amsterdam, v. 191, p. 27-53, 1991.
- NVIDIA. Disponível em: <<http://www.nvidia.com>>. Acesso em: 19 jul. 2005. Página de acesso aos *drivers* de controle das placas gráficas aceleradoras da companhia NVIDIA.
- OJEDA, H. A. O.; ARANHA, L. G. F. **Bacia de Santos, integração geológica regional**. Rio de Janeiro: Petrobrás, 1980. Relatório Interno.
- OJEDA, H. A. O.; CESERO, P. **Bacias de Santos e Pelotas, geologia e perspectivas petrolíferas**. Rio de Janeiro: Petrobrás, 1973. Relatório Interno.
- OPEN-GL. Disponível em: <<http://www.opengl.org>>. Acesso em: 19 jul. 2005. Biblioteca gráfica em linguagem C contendo os principais algoritmos de computação gráfica.
- PAULA, O. B.; MOHRIAK, W. U. Interpretação estrutural das megafeições da Bacia de Santos. In: SIMPÓSIO NACIONAL DE ESTUDOS TECTONICOS, n. 10, 2005, Curitiba. **Boletim de Resumos Expandidos...** Curitiba: SBG-PR, 2005. p. 94-96.
- PEREIRA, M. J.; BARBOSA, C. M.; AGRA, J.; GOMES, J. B.; ARANHA, L. G. F.; SAITO, M.; RAMOS, M. A.; CARVALHO, M. D.; STAMATO, M.; BAGNI, O. Estratigrafia da Bacia de Santos: análise das seqüências, sistemas deposicionais e revisão litoestratigráfica. In: CONGRESSO BRASILEIRO DE GEOLOGIA, n. 34., 1986, Goiânia. **Anais...** Goiânia: SBG, 1986, v.1. p. 65-79.
- PEREIRA, M. J.; FEIJÓ, F. J. Bacia de Santos. **Boletim de Geociências da Petrobrás**, Rio de Janeiro, v. 8, n. 1, p. 219-234, 1994.
- PFLUG, R.; HARBAUGH, J. W. **Three-dimensional computer graphics in modeling geologic structures and simulating geologic processes**. Berlim: Springer Verlag, 1992. 298 p.
- PLATE TECTONICS ONLINE RECONSTRUCTION TOOL. Disponível em: <<http://www.itis-molinari.mi.it/Intro-Reconstr.html>>. Acesso em: 4 ago. 2005. Programa de reconstrução paleogeográfica em tempo real.
- RAMSAY, J. G.; HUBER, M. I. **The techniques of modern structural geology: strain analysis**. London: Academic Press, 1983. v.3. 306 p.
- RANGEL, H. D.; MARTINS, F. A. L.; ESTEVES, F. R.; FEIJÓ, F. J. Bacia de Campos. **Boletim de Geociências da Petrobrás**, Rio de Janeiro, v. 8, n. 1, p. 203-217, 1994.
- RED HAT ENTERPRISE LINUX. Disponível em: <<http://www.redhat.com/software/rhel>>. Acesso em: 13 jul. 2005. Distribuição comercial do sistema Linux mantida pela empresa Red Hat.

- RPMFIND. Disponível em: <<http://www.rpmfind.com>>. Acesso em: 19 jul. 2005.
Repositório de módulos do sistema operacional Linux no formato da empresa Red Hat.
- SCHROEDER, W. J.; MARTIN, K.; LORENSEN, B. **The Visualization Toolkit**. 2nd ed. Upper Saddle River: Prentice Hall, 1996. 826 p.
- SCHETTINO, A.; SCOTese, C. R. A Synthetic APWP for Africa (Jurassic-Present) and Global Plate Tectonic Reconstructions. **Eos Trans. AGU**, v. 81, n. 19. 2000
- STROUSTRUP, B. **The C++ programming language**. 3rd ed. Boston: Addison-Wesley, 2000. 911 p.
- SYKES, T. J. S.; ROYER, J. Y.; RAMSAY, A. T. S.; KIDD, R. B. Southern emisphère palaeobathymetry. In: CRAMP, A.; MACLEOD, C. J.; LEE, S.V; JONES, E. J. W. (Ed.). **Geological Evolution of Ocean Basins: Results from the Ocean Drilling Program**. London: Geological Society, 1998. p. 3-42. (Special Publication, 131)
- TCL/TK. Disponível em: <<http://www.tcl.tk>>. Acesso em: 18 jul. 2005. Linguagem interpretada para prototipação e desenvolvimento de projetos.
- TCL/TK DISCUSSION GROUP.
Disponível em: <<http://groups.google.com/groups?q=comp.lang.tcl>>. Acesso em: 20 jul. 2005. Grupo de discussão sobre a linguagem Tcl/Tk.
- TOPEX. Disponível em: <<http://topex-www.jpl.nasa.gov>>. Acesso em: 15 dez. 2004. Ocean Surface Topography From Space.
- TWISS, J.; MOORES, E. M. **Structural Geology**. New York: W. H. Freeman & Company, 1992. 530 p.
- VISUALIZATION TOOLKIT. Disponível em: <<http://www.vtk.org>>. Acesso em: 19 jul. 2005. Ambiente de visualização científica de código aberto.
- VIRTUAL REALITY MODELLING LANGUAGE. Disponível em: <<http://www.w3.org/MarkUp/VRML>>. Acesso em: 15 jul. 2004. Modelagem de ambientes virtuais descentralizados na rede Internet.
- VTK MAILING LIST. Disponível em: <<http://www.kitware.com/searchpublic.html>>. Acesso em: 18 dez. 2004. Lista de discussão da biblioteca VTK.
- VIVIERS, M. C. Bioestratigrafia e evolução paleoambiental do Meso-Neocretáceo da Bacia de Santos, Brasil. In: CONGRESSO BRASILEIRO DE GEOLOGIA, 34., 1986. Goiânia. **Anais...** Goiânia: SBG, 1986, v.1, p. 50-64.
- VIVIERS, M. C.; AZEVEDO, R. L. M. The southeastern area of the brazilian continental margin: its evolution during the Middle and Late Cretaceous as indicated by

paleoecological data. **Revista Brasileira de Geociências**, São Paulo, v. 18, n. 3, p. 291-298, 1988.

WATSON, D. F. Computing the n-dimensional Delaunay Tessellation with application to Voronoy polytopes. **The Computer Journal**, London, v. 24, n. 2, p. 167-172, 1981.

WELCH, B.B. **Practical programming in Tcl and Tk**. Upper Saddle River: Prentice Hall, 1997. 630 p.