Universidade Estadual de Campinas

Faculdade de Engenharia Elétrica e de Computação

departamento de engenharia de computação e automação industrial

Visualização Científica de Mapas de Poincaré

Autor: **Sidney Pio de Campos**

Orientadora: **Wu, Shin-Ting**

Dissertação submetida à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, para preenchimento dos pré-requisitos parciais para obtenção do Título de Mestre em Engenharia Elétrica.

abril 1998

Esta tese foi defendida no dia 11 de abril de 1997 com a banca examinadora composta por:

Profa. Dra. Wu Shin-Ting	— DCA/FEEC/UNICAMP
Prof. Dr. Clésio Luís Tozzi	— DCA/FEEC/UNICAMP
Prof. Dr. Marcus A.M. Aguiar	— DFESCM/IFGW/UNICAMP

Profa. Dra. Wu Shin-Ting

Resumo

As características de um sistema podem ser imprevisíveis dependendo das condições iniciais e dos parâmetros externos aplicados ao sistema. Essa característica de imprevisibilidade é conhecida como caos determinístico e seu estudo é essencial para várias áreas de pesquisa tais como Física, Biologia, Química, Medicina e Economia. Uma ferramenta útil para análise desses fenômenos é conhecida como mapas de Poincaré.

Nesse trabalho apresentamos uma ferramenta gráfica para a visualização e manipulação dos mapas de Poincaré. O conjunto de pontos que compõem um mapa é convertido em uma imagem binária. Nessa imagem podem ser aplicadas técnicas de Processamento de Imagens a fim de diminuir as descontinuidades e determinar os contornos da regiões normais e caóticas e suas propriedades. Com a ferramenta desenvolvida permite-se também a interpolação de conjuntos de contornos de diferentes mapas e posteriormente a visualização 3D do objeto resultante.

O nosso objetivo é demonstrar que, em certas aplicações, com técnicas relativamente simples de Computação Gráfica podem ser desenvolvidas ferramentas úteis para a análise de dados e auxílio à pesquisa.

Abstract

Depending on the external parameter and the initial conditions applied to a system, its behaviour can be highly unpredicable. This kind of unpredicability is known as deterministic chaos and its study is essential to several research areas such as Physics, Biology, Chemistry, Medicine and Economy. A useful tool to identify this phenomenon is Poincaré maps generated by a simulation.

In this work a prototype for the computer analysis of Poincaré maps is described. We demonstrate that, from the point-of-view of computer graphics, we can process Poincaré maps as noisy images. This approach not only facilitates the partition of Poincaré maps into regular and chaotic regions but also offers possibilities to visualize the continuous evolution of a system by varying the actuator.

Our objective is to show that, in some cases, useful tools for suporting data analysis can be developed with aid of simple techniques of Computer Graphics.

Agradecimentos

A Profa. Dra. Wu Shin-Ting, pela sua orientação, paciência e determinação durante esse trabalho.

Ao Prof. Dr. Marcus Aguiar, pela cooperação e auxílio com os dados utilizados nesse trabalho.

Ao Prof. Dr. Clésio Luís Tozzi, pelas correções e sugestões dadas durante a realização desse trabalho.

A Nilma Lúcia Viguetti Campos, minha esposa, sempre presente em todos os momentos, enfrentando comigo os desafios.

A minha mãe e minha irmã, que sempre incentivaram, mesmo geograficamente distante.

Ao Prof. Dr. Oscar Ferreira de Lima, meu primeiro orientador de iniciação científica.

Aos amigos do DCA/FEEC/UNICAMP, pelas sugestões apresentadas durante a realização desse trabalho.

Aos amigos do IFGW/UNICAMP, pelo apoio recebido no decorrer desse trabalho.

É preciso amar as pessoas
como se não houvesse amanhã
Porque se você parar para pensar,
na verdade não há

Pais e Filhos - Dado Villa-Lobos/Renato Russo/Marcelo Bonfá

Dedico esse trabalho a meu pai, Antonio de Campos.

Sumário

Lista de Figuras

Capítulo 1

Introdução

1.1 Visualização Científica

A Visualização Científica [20] [9] [3] é uma metodologia multidisciplinar que utiliza técnicas de visualização de imagens para auxiliar o pesquisador na observação e análise dos dados experimentais ou simulados, facilitando a compreensão da sua estrutura. Um dos fundamentais objetivos da visualização científica é permitir ao usuário explorar a informação inerente aos dados, fornecendo métodos alternativos de análise e apresentação.

O desenvolvimento de ferramentas gráficas é beneficiada pela disponibilidade de modernas estações de trabalho, com bom desempenho, grandes quantidades de memória e disco e com poderosas capacidades gráficas.

Cientistas e engenheiros já utilizavam a visualização como ferramenta na compreensão de fenômenos complexos no passado. Existem pesquisas históricas que mostram que muitos cientistas notáveis como Faraday e Maxwell utilizavam seu raciocínio visual para o desenvolvimento de seus trabalhos.

O marco recente para a área de Visualização é geralmente atribuída a publicação do *" NSF Report on Visualization in Scientific Computing "* [1]. Nessa publicação são estabelecidas as definições, os domínios e recomendações para a nova área.

A visualização compreende tanto a interpretação quanto a síntese de imagens a partir de complexos conjuntos de dados multidimensionais. A visualização científica integra diversas áreas de computação, por exemplo:

- Computação Gráfica;
- Processamento de Imagens;
- Visão Computacional;
- Projeto Auxiliado por Computador;
- Processamento de Sinais;
- Estudos de Interface com o usuário.

Atualmente possuímos várias fontes que geram grandes quantidades de dados:

- Dados obtidos a partir de simulações em computadores;
- Dados obtidos por satélites no espaço;
- Imagens Médicas tais como os dados de Tomografias e Ressonância Magnética;
- Dados obtidos por instrumentos de medidas geofísicas (temperatura do oceano, abalos sísmicos, etc).

Normalmente os dados acima citados são enviados (ou gerados) em formato numérico. Usando exclusivamente o formato numérico, a análise de um grande conjunto de dados pode ser difícil. Através da visualização científica será possível realizar uma análise com eficiência muito maior. É estimado que 50% dos neurônios do cérebro são associados com visão.

A lista de oportunidades de pesquisa criadas com a área de visualização é grande. Como ilustração podemos citar algumas aplicações:

- Modelagem de Moléculas;
- Imagens Médicas (diagnóstico médico, planejamento de cirurgias, planejamento de tratamento com radiação, estudo do funcionamento e estrutura do cérebro);
- Pesquisa na área matemática;
- Geociências(cartografia, meteorologia, geologia);
- Astrofísica;

- Estudos de Dinâmica dos Fluidos;
- Análise de Elementos Finitos.

É importante diferenciar visualização científica de arte. Na realidade não estamos preocupados com o lado artístico, e sim em facilitar a apresentação e interpretação dos dados. Devemos nos preocupar em desenvolver ferramentas úteis e funcionais.

A visualização científica pode ser definida de acordo com o diagrama apresentado na figura (1.1). Imagens e sinais podem ser capturados por câmeras ou sensores, transformadas por processamento de imagens e apresentadas ou impressas. A abstração dessas representações visuais podem gerar representações simbólicas (símbolos ou estruturas). Utilizando técnicas de computação gráfica, símbolos ou estruturas podem ser geradas dentro de representações visuais.

Como exemplo de sistemas de visualização científica podemos citar:

- DataExplorer desenvolvido pela IBM;
- AVS (Application Visualization System) desenvolvido por Advanced Visual Systems Incorporated [18];
- IRISExplorer desenvolvido pela SGI;
- FAST (Flow Analysis Software Toolkit) desenvolvido por Sterling Federal Systems.

Um aspecto importante da visualização científica é o trabalho com equipes multidisciplinares. A participação de especialistas da área para a qual se deseja desenvolver a ferramenta de visualização é fundamental para o correto desenvolvimento do projeto.

O nosso objetivo é demonstrar que, em certas aplicações, com técnicas relativamente simples de computação gráfica podem ser desenvolvidas ferramentas úteis para a análise de dados e auxílio à pesquisa.

Assim, resolvemos escolher um problema específico no Instituto de Física Gleb Wataghin (IFGW) da UNICAMP e desenvolvemos uma ferramenta de visualização. O problema escolhido foi a visualização científica de mapas de Poincaré. Todas as informações relativas aos mapas de Poincaré foram fornecidas pelo Prof. Dr. Marcus A.M. de Aguiar do Departamento de Física do Estado Sólido e Ciência dos Materias (DFESCM) do IFGW.

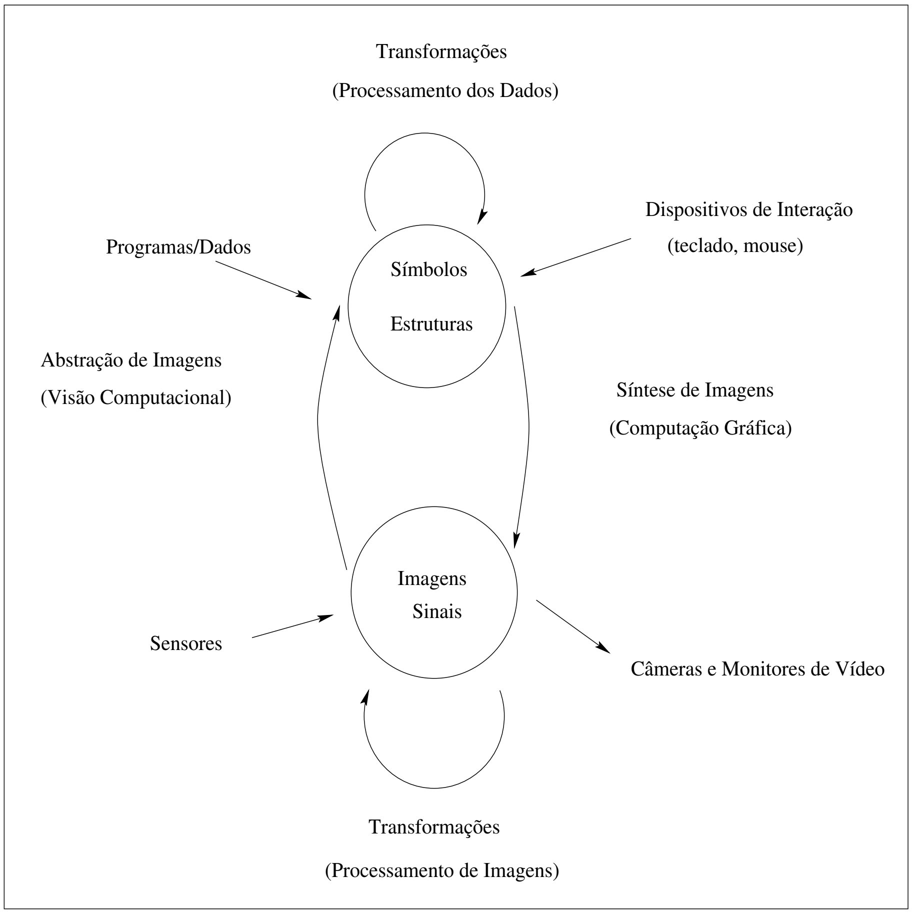


Figura 1.1: Taxonomia da visualização científica

1.2 O Problema

As características de um sistema podem ser imprevisíveis dependendo das condições iniciais e dos parâmetros externos aplicados ao sistema. Essa característica de imprevisibilidade é conhecida como caos determinístico e seu estudo é essencial para várias áreas de pesquisa tais como física, biologia, química, medicina e economia. Uma ferramenta útil para análise desses fenômenos é conhecida como mapas de Poincaré.

Um mapa de Poincaré corresponde a um conjunto de pontos sobre um plano. Cada ponto representa os valores assumidos pelas duas variáveis do sistema de interesse sob determinadas condições. Se o sistema apresenta uma característica de imprevisibilidade, os pontos plotados não formam uma curva geométrica bem definida. Classificamos a região a que eles pertencem como uma região caótica. Se os pontos apresentam uma distribuição regular (isto é, curvas geometricamente bem definidas), as regiões são consideradas regulares ou normais. Nas regiões caóticas existe o fenômeno chamado de " sensibilidade às condições iniciais " que diz que dois pontos iniciais vizinhos separam-se exponencialmente rápido com a aplicação do mapa: $\delta x_n \simeq e^{\lambda n}\delta x_0$. O expoente λ é chamado de *exponente de Lyapunov* .

Uma típica aplicação dos mapas de Poincaré é estudar a evolução do sistema a partir da variação de um parâmetro externo (por exemplo um campo magnético). Cada mapa corresponde a um valor aplicado. Comparando-se os mapas, pode-se inferir o comportamento do sistema para os valores intermediários entre os valores aplicados. Pode-se também utilizar os mapas de Poncaré para auxiliar na análise da evolução de determinada trajetória de uma partícula através da variação de um parâmetro externo. Até o presente momento, essas tarefas são realizadas de maneira arcaica, com auxílio do computador mas sem o auxílio de análises gráficas de imagens.

A distinção entre regiões regulares e caóticas também é uma importante característica para os problemas na área de Física. Algumas técnicas de força-bruta para a exata determinação das regiões requerem um custo computacional tão elevado que as tornam impraticáveis [4] . Na maioria dos casos a determinação é realizada visualmente, através da análise dos mapas.

1.3 Proposta de Visualização

Atualmente os mapas de Poincaré são analisados com o auxílio de um software de domínio público chamado Ace/gr [1]. Este software não foi desenvolvido exclusivamente para a análise de mapas de Poincaré, permitindo a visualização de um conjunto de pontos utilizando um sistema de 2 coordenadas. Desse modo existem limitações para efetuar análises e manipulações mais sofisticadas, como

1. analisar a variação deste conjunto de pontos sob a variação do parâmetro externo. O pesquisador imprime os gráficos correspondentes a vários mapas e interpola esses mapas mentalmente;
2. ajustar finamente os pontos obtidos nas simulações;

[1] Ace/gr - Graphics for Exploratory Data, desenvolvido por Paul J. Turner

3. estimar a percentagem de regiões normais/caóticas em cada mapa. Hoje o pesquisador calcula ponto a ponto o expoente de Lyapunov λ propagando uma trajetória vizinha. Se $\lambda > 0$, o ponto pertence à região caótica. Esse procedimento é muito custoso, no entanto.

Após uma série de reuniões com o Prof. Dr. Marcus A. M. de Aguiar, decidiu-se, como proposta para o trabalho de tese desenvolver uma ferramenta gráfica, que sob ponto de vista de aplicação e sob ponto de vista do sistema:

- seja portável para as diversas arquiteturas Unix disponíveis no IFGW/UNICAMP. A linguagem de programação utilizada será C, com rotinas gráficas para o sistema de janelas X (*X-Window System*);
- auxilie na comparação visual entre o conjunto de dados de vários mapas de Poincaré, correspondendo a distintos valores do parâmetro externo. Hoje eles tentam realizar esse tipo de operação manualmente, através da comparação dos mapas impressos em papel (sobrepondo os gráficos impressos contra a luz, por exemplo), ou gerando outro mapa de Poincaré completo para valores intermediários;
- auxilie na estimação da percentagem de regiões normais e caóticas e identificar " visualmente " o limite entre as regiões normais e caóticas;
- auxilie na visualização do comportamento em estágios intermediários através de interpolação de sucessivos mapas de Poincaré , como por exemplo o corte em um plano paralelo ao plano xy (gerando o que chamamos de **diagrama de bifurcação**).

1.4 Organização do Trabalho

Inicialmente no capítulo 2 apresentaremos os conceitos dos mapas de Poincaré, as áreas de aplicação e exemplos de mapas, Assim apresentamos ao leitor uma noção geral do tipo de problema que estamos propondo resolver.

No capítulo 3 apresentamos sucintamente as técnicas de computação gráfica e processamento de imagens utilizadas no desenvolvimento do projeto. São descritos os conceitos de filtragem (em especial os filtros passa-baixa), técnicas de segmentação, interpolação e visualização.

No capítulo 4 apresentamos as decisões de projeto e uma implementação da ferramenta de visualização. São apresentados os módulos que compõem a ferramenta e as técnicas que foram utilizadas na implementação. Nesse capítulo também apresentamos a interface gráfica do protótipo: o esquema geral da interface e as diretrizes utilizadas.

Durante o desenvolvimento do protótipo tentamos aproveitar ao máximo ferramentas de domínio público, evitando reescrever código. Essas ferramentas utilizadas na implementação da ferramenta são descritas no capítulo 5.

Apresentamos alguns resultados obtidos com o uso da ferramenta no capítulo 6.

No capítulo 7 são apresentadas as conclusões e propostas para trabalhos futuros.

Por fim, apresentamos a bibliografia utilizada e os apêndices, que apresentam uma documentação mais detalhada do protótipo, com detalhes de estruturas de dados e algoritmos utilizados. São apresentadas as telas que compõem o protótipo e uma correção realizada no código de interpolação desenvolvido por outro trabalho de pesquisa.

Capítulo 2

Mapas de Poincaré

Nos últimos 30 anos o interesse de várias áreas do conhecimento tem se voltado para a questão do comportamento caótico nas soluções de problemas que anteriormente pareciam simples e sem relevância científica. Na verdade, apesar da palavra *caos* ter sido acrescentada ao vocabulário científico há relativamente pouco tempo, seu conteúdo e existência já eram conhecidos no começo deste século pelo matemático francês Henri Poincaré.

Os mapas de Poincaré tornaram-se uma importante ferramenta no estudo de sistemas dinâmicos em várias áreas de pesquisa. A definição precisa do mapa de Poincaré depende do problema que está sendo resolvido, mas a idéia básica é reduzir o estudo de um fluxo contínuo para um mapa discreto.

Os mapas de Poincaré são compostos por dois tipos de região: uma região normal ou regular, apresentando uma distribuição geometricamente bem definida e regiões caóticas, com características de imprevisibilidade. A seguir apresentaremos três exemplos de sistemas que podem ser analisados através de mapas de Poincaré.

2.1 Sistemas Conservativos (com 2 graus de liberdade)

A idéia inicial dos mapas de Poincaré surgiu para solucionar problemas em sistemas conservativos com dois graus de liberdade. Consideremos uma partícula movendo-se no plano x-y sobre a ação de forças. (campo magnético, por exemplo). Dada a condição inicial $(x_0, y_0, \dot{x}_0, \dot{y}_0)$, uma única solução da equação de Newton é determinada. Para descrever completamente a trajetória da partícula precisamos calcular $x(t)$, $y(t)$, $\dot{x}(t)$ e $\dot{y}(t)$. Isto não é muito prático, desde que não podemos visualizar as quatro dimensões do espaço de fase.

Entretanto, se o movimento é limitado (isto é, mantém-se em uma região finita do espaço de fase) uma representação bidimensional do movimento da partícula é possível fazendo-se $y_0 = 0$ e $\dot{y}_0 > 0$. Propagamos a trajetória até o tempo $t = t_1$ onde $y(t_1) = 0$, novamente com $\dot{y}(t_1) > 0$. Nesse instante salvamos os valores de $x(t)$ and $\dot{x}(t)$.

O mapa de Poincaré para este caso será definido pelo mapeamento $(x_0,\dot{x}_0) \rightarrow (x(t_1),\dot{x}(t_1))$. Facilmente pode-se mostrar que todas as propriedades básicas do fluxo são refletidas no mapa. Em particular, é fácil verificar que as trajetórias periódicas do fluxo correspondem a pontos periódicos do mapa. A característica de estabilidade também é preservada: trajetórias do fluxo estáveis correspondem a órbitas estáveis dos mapas; trajetórias do fluxo instáveis correspondem a órbitas instáveis dos mapas. As trajetórias instáveis são normalmente chamadas de caóticas, uma vez que aparecem nos mapas de Poincaré como uma seqüência de pontos aparentemente aleatórios cobrindo uniformemente certas áreas do plano do mapa. As trajetórias regulares ou estáveis apresentam-se como curvas simples quando mapeadas successivamente.

2.2 Oscilador Harmônico

Como segundo exemplo podemos considerar o caso de um oscilador harmônico bidimensional. O movimento do oscilador é realizado no plano (x,y), estando acoplado a duas molas de constantes k_1 e k_2 e e freqüências de oscilação $w_1 = \sqrt{k_1/m}$ e $w_2 = \sqrt{k_2/m}$. Nesse caso a energia total será a soma das energias em cada modo de oscilação e será definida pela equação (2.1):

$$H(x, y, p_x, p_y) = \frac{{p_x}^2}{2m} + \frac{{p_y}^2}{2m} + \frac{m{w_1}^2x^2}{2} + \frac{m{w_2}^2y^2}{2} \tag{2.1}$$

onde $p_x = m\dot{x}$ e $p_y = m\dot{y}$ são os momentos (quantidade de movimento) . Usando-se as equações de Newton:

$$m\dot{x} = -k_1 x$$

$$m\ddot{y} = -k_2 y \tag{2.2}$$

é fácil mostrar que $\frac{dH}{dt} = 0$, isto é, a " função energia " H permanece constante ao longo do movimento. Nesse caso, o movimento do oscilador fica restrito a 3 dimensões apenas, numa região chamada de " superfície de energia ".

Escolhemos agora o plano y = 0 dentro desse espaço tridimensional e marcamos as sucessivas intersecções das trajetórias com essa superfície, conforme ilustrado na figura (2.1). A superfície onde marcamos os pontos é a seção de Poincaré e as intersecções sucessivas geram o mapa de Poincaré.

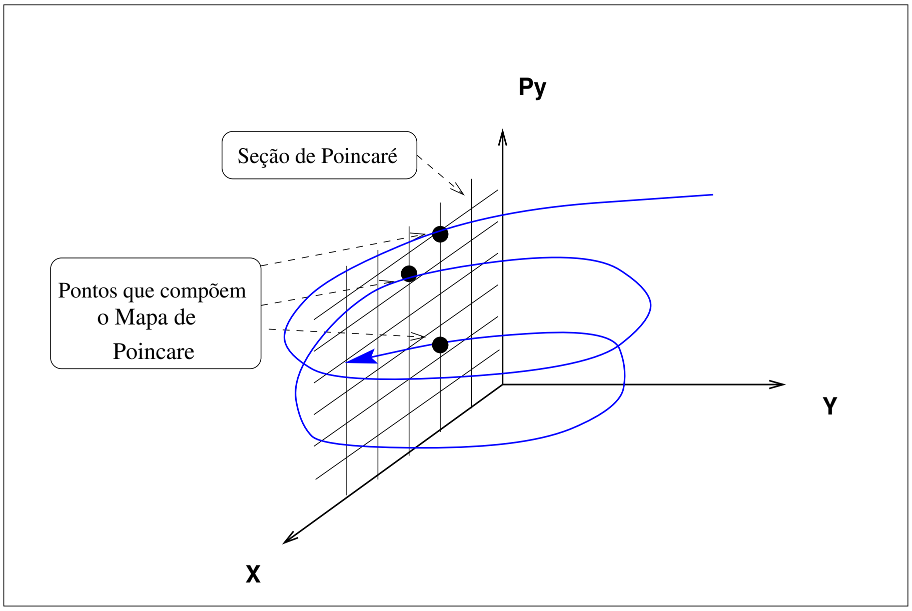


Figura 2.1: Definição de um mapa de Poincaré para Oscilador Harmônico

Para o mapa escolhemos uma trajetória qualquer do sistema que tenha energia E e observamos sua evolução temporal. Toda vez que a coordenada y passar por zero, marcamos um ponto no plano (x, px) definido pelo valor das coordenadas x(t) e px(t) no instante em que y = 0. Depois de algum tempo, quando a coordenada y voltar a se anular, marcamos um novo ponto no plano (x, px). Desse modo conseguimos gerar um mapa de Poincaré desta trajetória escolhida. De fato, como as trajetórias furam a seção nos dois sentidos, os pontos (x, px) só são marcados se $py > 0$, ou seja, só marcamos os pontos quando as trajetórias atravessam a seção em um dos sentidos.

2.3 Mesa de Bilhar

Como terceiro exemplo iremos utilizar um sistema onde uma partícula carregada move-se no plano x-y e está confinada a uma região quadrada de tamanho unitário. Se a partícula não estiver sujeita a forças, ela irá se mover em linhas retas dentro do quadrado e ao entrar em contato com as bordas a reflexão no movimento será especular (ângulo de incidência igual ao de reflexão).

Caso a partícula também esteja sujeita a um campo magnético constante na direção z, o movimento em direção reta será substituído pelo movimento sobre círculos, cujo raio é proporcional ao inverso do campo magnético. Desde que o movimento da partícula entre duas colisões consecutivas é trivial, o mapa de Poincaré é definido observando-se apenas os pontos de colisão: plotaremos o comprimento de arco x do ponto de colisão, medido a partir de uma das bordas, e $\cos\theta$, que corresponde ao ângulo que a partícula gera até a próxima colisão. Uma vez que o sistema é invariante de rotação por $\pi/2$ é suficiente para mostrar que o mapa de Poincaré está definido para $0 \leq x \leq 1$. Na figura (2.2) o sistema e as coordenadas do mapa são apresentados.

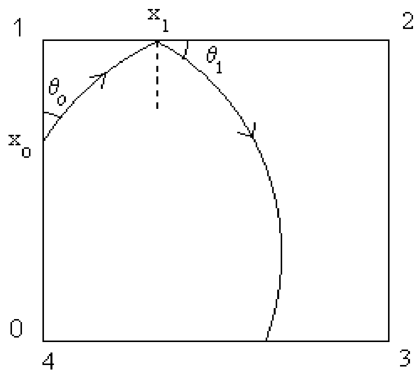


Figura 2.2: Partícula carregada confinada a uma região quadrada e campo magnético constante na direção z

2.4 Aplicações Gerais do Plano

A ferramenta gráfica desenvolvida nesta tese pode ser empregada para estudar aplicações gerais do plano sobre si mesmo. Dadas duas funções suaves f_1 e $f_2 : \Re^2 \to \Re$ definimos a aplicação por:

$$x_{n+1} = f_1(x_n, y_n)$$

$$y_{n+1} = f_2(x_n, y_n) \tag{2.3}$$

Dada uma "condição inicial" (x_0,y_0) a aplicação gera as "órbitas" $(x_0,y_0);(x_1,y_1);(x_2,y_2)...$ Se o Jacobiano da aplicação satisfazer a seguinte igualdade:

$$\frac{\partial(f_1, f_2)}{\partial(x, y)} = \begin{vmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{vmatrix} = 1, \tag{2.4}$$

a aplicação, ou "mapa", será conservativa no sentido que uma região $U \in \Re^2$ de área u seja mapeada na região $U' = f(U)$ com mesma área $u' = u$.

Os mapas de Poincaré exemplificados anteriormente são aplicações conservativas do plano definidas implicitamente através das trajetórias contínuas geradas pelas equações de Newton.

Exemplos de mapas que podem ser escritos implicitamente na forma de (2.3) são:

- mapa conservativo de Henon (α parâmetro)

$$\begin{cases} x_{n+1} = x_n \cos\alpha - (y_n - x_n^2) \sin\alpha \\ y_{n+1} = y_n \sin\alpha - (y_n - x_n^2) \cos\alpha \end{cases} \tag{2.5}$$

- mapa dissipativo de Henon (α , β parâmetros)

$$\begin{cases} x_{n+1} = y_n + 1 - \alpha x_n^2 \\ y_{n+1} = \beta x_n \end{cases} \tag{2.6}$$

Capítulo 3

Técnicas de Computação Gráfica e Processamento de Imagens

Nesse capítulo iremos descrever brevemente as principais técnicas de computação gráficas e processamento de imagens utilizadas na implementação do protótipo. Uma descrição com mais detalhes desses conceitos pode ser verificada em [6] [8] [11] [15] [2].

O modelo de visualização adotado no desenvolvimento do protótipo pode ser dividido nas seguintes etapas:

- Pré-processamento dos dados: Nessa etapa o conjunto de dados é processado, convertendo para um formato adequado para os passos seguintes;
- Filtragem: Os dados originais obtidos pelo processo de simulação consistem de imagens discretas. São utilizadas técnicas de filtragem nos dados originais numa tentativa de obter imagens contínuas a partir de imagens discretas que apresentam descontinuidades. Na imagem contínua poderemos definir as regiões de interesse e as características do conjunto de dados;
- Segmentação:Após a filtragem podemos separar as regiões, segmentando a imagem;
- Interpolação: Os conjuntos de contornos definidos no processo de segmentação podem ser interpolados permitindo a observação do comportamento dos mapas em função da variação do parâmetro externo;

- Rendering: Após a interpolação são utilizadas técnicas a fim de visualizar melhor o comportamento do sistema como um todo e os mapas interpolados (utilizando as características de cor e tonalização, transparência, por exemplo).

Um esquema do processo de visualização pode ser observado na Figura (3.1).

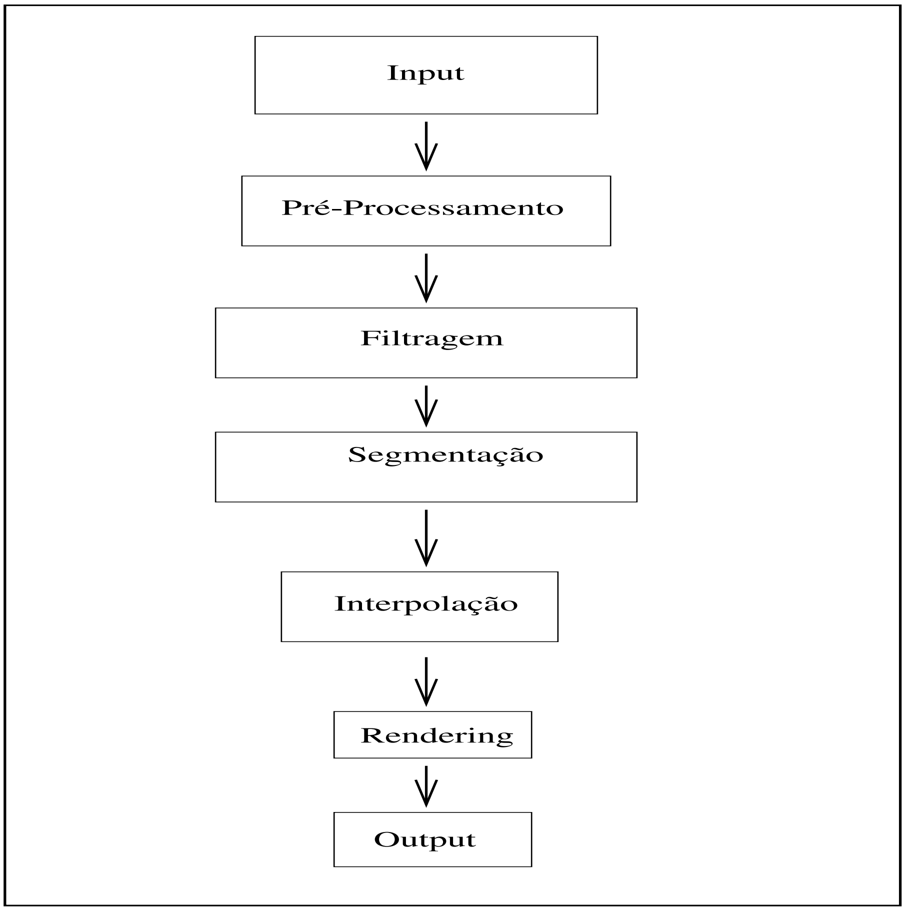


Figura 3.1: Esquema Geral do Processo de Visualização

Nas seções a seguir descreveremos sucintamente, com as devidas referências, algumas possíveis técnicas para a execução de cada uma das etapas apresentadas na Figura (3.1).

3.1 Pré-Processamento

Nessa etapa o conjunto de pontos gerados pela simulação são convertidos do espaço objeto para o espaço imagem. No nosso caso o conjunto de pontos será mapeado em uma matriz bidimensional binária. Os detalhes dessa etapa de pré-processamento são apresentados na seção (4.1.1).

3.2 Filtragem

Um dos objetivos do uso de filtros é realizar um "melhoramento" da imagem. No caso específico dos mapas de Poincaré, tentou-se através do uso de filtros diminuir as descontinuidades das imagens geradas devido às subamostragens.

Podemos dividir os filtros em filtros suavizantes e filtros aguçantes. Os filtros suavizantes são usados para redução de ruídos e para "borrar" a figura (*blurring*). Estes filtros são utilizados para a remoção de pequenos detalhes de uma imagem ou para ligar pequenos "vazios" (*gaps*) em linhas ou curvas.

Os filtros podem ser definidos no domínio espacial ou no domínio da freqüência. Os primeiros são baseados na direta manipulação dos pixels da imagem, enquanto os do domínio de freqüência operam sobre a transformada de Fourier da imagem.

A categoria de filtros no domínio espacial normalmente estão associados ao uso de máscaras. Basicamente a máscara é um pequeno vetor bidimensional, por exemplo 3x3, conforme a figura a seguir:

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Supondo que o centro da máscara está localizado na coordenada (x,y) da figura a ser filtrada, o novo valor do pixel em (x,y) será definido por R:

$$R = w_1 z_1 + w_2 z_2 + ... + w_9 z_9, \tag{3.1}$$

onde $w_1, w_2...w_9$ correspondem aos valores dos pixels da máscara e $z_1, z_2...z_9$ correspondem aos valores dos pixels da imagem centrada em (x,y). Esse procedimento é exemplificado na figura (3.2).

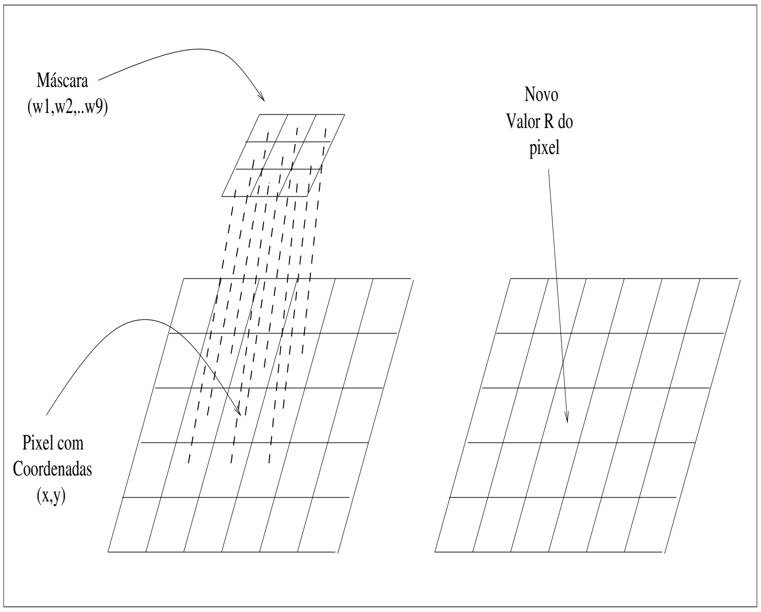


Figura 3.2: Processo de filtragem por máscaras

A máscara move-se para o próximo pixel e o processo é repetido, continuando esse processo até que todos os pixels da imagem tenham sido percorridos.

Como exemplo de filtros espaciais iremos apresentar o filtro da média (seção 3.2.2) e o filtro da mediana (seção 3.2.3). Apresentaremos também na seção 3.2.4 o filtro passa-baixa definido no domínio espacial. Outra categoria de filtros constituem os filtros morfológicos, baseados na morfologia matemática (seção 3.2.5). Antes, porém, introduzimos a noção de conectividade de pixels.

3.2.1 Conectividade de pixels

Um pixel p com coordenadas (x,y) tem quatro vizinhos horizontais e verticais cujas coordenadas são dadas por:

(x+1,y), (x-1,y) (x, y+1), (x, y-1)

Este conjunto de pixels, chamados *4-vizinhos de p* (*4-neighbors of p*) é denotado por $N_4(p)$. O pixel p também possue 4 vizinhos na *diagonal*:

(x+1,y+1), (x+1,y-1) (x-1, y+1), (x-1, y-1)

que são denotados por $N_D(p)$. Esses pontos, juntamente com os *4-vizinhos de p* são chamados *8-vizinhos de p*, denotados por $N_8(p)$.

Para determinar se dois pixels são conectados, devemos determinar se eles são adjacentes por algum critério (*4-vizinhos*, por exemplo) e se satisfazem um critério específico de similaridade (se são iguais, por exemplo). Considerando V o conjunto de valores de cinza utilizados para definir a conectividade, podemos considerar 3 tipos de conectividade:

- *4-conectividade*: Dois pixels p e q com valores no conjunto V são 4-conexos se q está no conjunto $N_4(p)$;
- *8-conectividade*: Dois pixels p e q com valores no conjunto V são 8-conexos se q está no conjunto $N_8(p)$;
- *m-conectividade*: Dois pixels p e q com valores no conjunto V são m-conexos se
  - q está no conjunto $N_4(p)$, ou
  - q está no conjunto $N_D(p)$ e o conjunto $N_4(p) \cap N_4(q)$ é vazio.

Se uma região é definida como 4-conexa, todos os pixels da região podem ser alcançados por uma combinação de movimentos apenas em $N_4(p)$. Em uma região 8-conexa, todos os pixels da região podem ser alcançados por uma combinação de movimentos em $N_8(p)$.

3.2.2 Filtro da Média

O filtro da média é a técnica mais direta para filtragem no domínio espacial. A idéia desse filtro é obter a média dos valores dos pixels contidos em uma vizinhança pré-definida. Podemos definir a seguinte máscara para um filtro da média 3x3:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Devemos notar que, ao passar esse filtro, o que vamos obter na verdade é a média de todos os pixels na área vizinha ocupada pela máscara.

3.2.3 Filtro da Mediana

Neste filtro tomamos para cada pixel p uma vizinhança 8-connectada de p, com n pixels $p_1, p_2, ..., p_n$. Calculamos a mediana mp desse conjunto, ordenando os pixels p_i segundo o seu valor de intensidade. O valor do filtro em p é definido como sendo mp, isto é, o nível de cinza de cada pixel é trocado pela mediana do nível de cinza na vizinhança do pixel. Se a intensidade de um pixel é muito diferente da intensidade dos pixels vizinhos ele certamente não será a mediana dos pixels numa vizinhança, e portanto seu valor, após a filtragem será mudado para o valor de outro pixel da vizinhança. A principal função do filtro da mediana é forçar os pontos com intensidades distintas se tornarem mais próximos de seus vizinhos, realmente eliminando os picos de intensidade que aparecem isolados na área da máscara do filtro.

3.2.4 Filtros Passa-Baixa

Analisando uma imagem no domínio da freqüência, podemos perceber que as arestas e outras transições estreitas (tais como ruídos) na imagem contribuem fortemente para o conteúdo de alta-freqüência. Assim, se pensarmos que desejamos eliminar as transições abruptas na nossa imagem *F(u,v)* , deveremos utilizar um filtro passa-baixa *H(u,v)* no domínio de freqüência, tal que:

$$G(u,v) = H(u,v)F(u,v) \tag{3.2}$$

onde *G(u,v)* corresponde a imagem com as transições eliminadas (imagem filtrada).

Podemos definir a função H(u,v) correspondente ao filtro passa-baixo ideal com freqüência de corte D_0 como:

$$H(u,v) = \begin{cases} 1 & \text{se } D(u,v) \leq D_0 \\ 0 & \text{se } D(u,v) > D_0 \end{cases} \tag{3.3}$$

onde $D(u,v)$ é a freqüência no ponto (u,v) .

Podemos também definir um filtro passa-baixa onde a transição na freqüência de corte não é tão estreita, como o chamado filtro de Butterworth de ordem n, da seguinte maneira:

$$H(u,v) = \frac{1}{1 + [D(u,v)/D_0]^{2n}} \tag{3.4}$$

Na figura (3.3) apresentamos os gráficos relativos aos filtros passa-baixa.

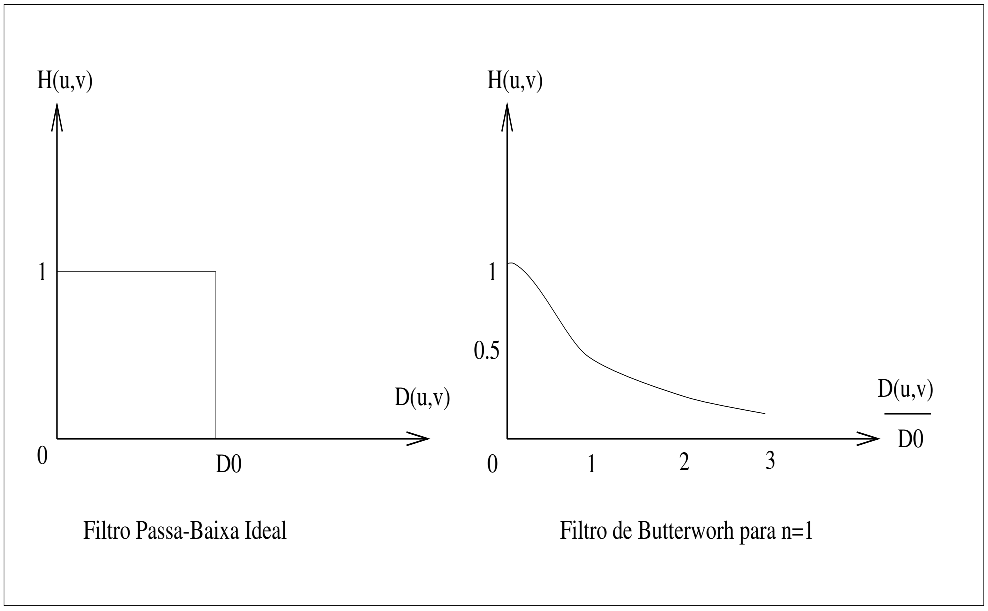


Figura 3.3: Filtros passa-baixa

3.2.5 Filtros Morfológicos

A morfologia matemática [19] compreende a área que estuda as propriedades topológicas e estruturais dos objetos a partir de suas imagens. Assim, as técnicas são semelhantes às técnicas utilizadas em modelagem geométrica.

Os filtros utilizados na morfologia matemática são definidos a partir de um **elemento estruturante**. Esse elemento consiste de um subconjunto do plano que interage com a imagem do objeto de forma a determinar propriedades topológicas e revelar informações estruturais do objeto.

Dois filtros básicos dessa área são os filtros de dilatação e erosão. Inicialmente vamos definir esses filtros no domínio contínuo. Supondo o conjunto B como o elemento estrutural das operações morfológicas. Dado o subconjunto B(x,y) do plano com o ponto de origem (x,y) e um subconjunto X do plano, a erosão do conjunto X por B é o conjunto X $\ominus$ B(x,y), formado por todos os pontos (x,y) $\in$ X tal que B(x,y) $\subset$ X, isto é,

$$X \ominus B(x,y) = (x,y) \in X; B(x,y) \subset X \tag{3.5}$$

A dilatação de um conjunto X pelo elemento estrutural B(x,y), X $\oplus$ B(x,y), é o conjunto dos pontos do plano para os quais o elemento estrutural B intersecta o conjunto X, isto é,

$$X \oplus B(x,y) = (x,y) \in \mathcal{R}^2; B(x,y) \cap X \neq \phi. \tag{3.6}$$

Intuitivamente, a operação de erosão subtrai os pontos do conjunto X, enquanto a operação de dilatação acrescenta pontos ao conjunto. No entanto, essa visão intuitiva pode não corresponder à realidade. Dependendo do elemento estrutural o resultado da dilatação pode não conter nenhum elemento do conjunto inicial.

As definições de operações de dilatação e erosão se aplicam diretamente no caso de imagens binárias. Para as demais imagens (coloridas, por exemplo) a extensão dessas operações não é imediata.

No domínio discreto os filtros de erosão e dilatação são definidos de maneira análoga, trabalhando com conjuntos discretos.

A figura (3.4) ilustra o exemplo de uso dos filtros de dilatação e erosão:

3.2.6 Preenchimento por semente

O algoritmo descrito realiza o preenchimento de regiões a partir de uma semente definida. O algoritmo utiliza uma pilha, que corresponde a um vetor no qual os valores são

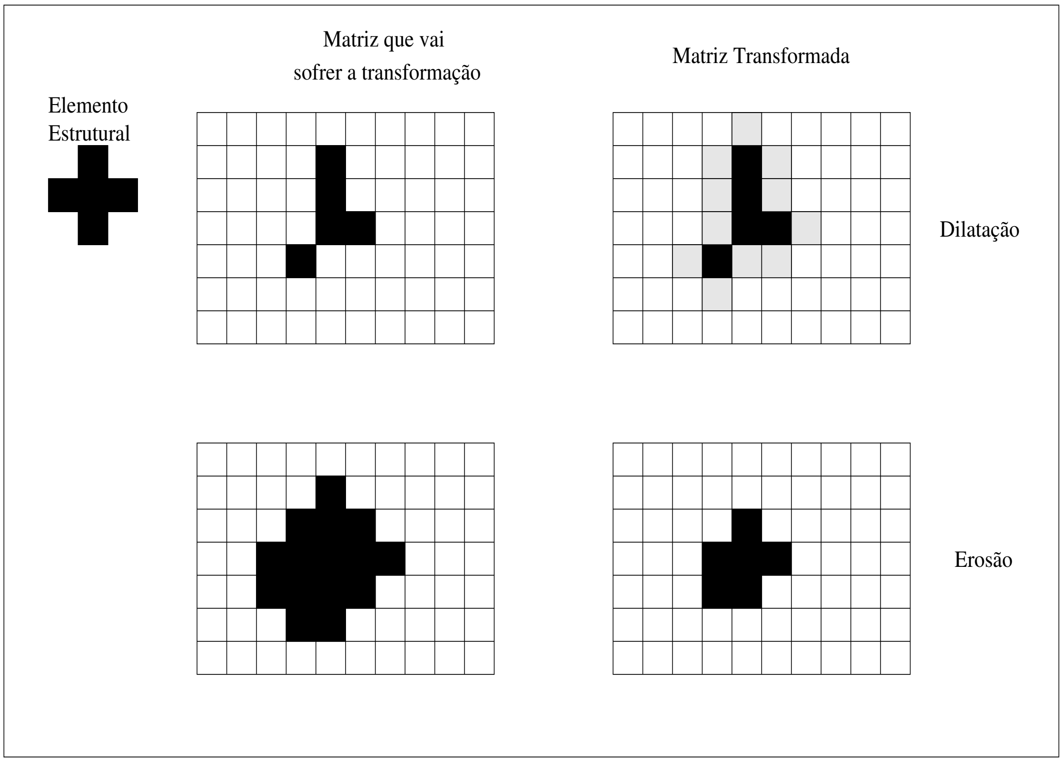


Figura 3.4: Operações de dilatação e erosão

sequencialmente armazenados e removidos. Nessa estrutura segue-se sempre a ordem FILO (*First In Last Out*): O primeiro elemento inserido na pilha é o último a ser retirado.

Em linhas gerais o algoritmo é o seguinte:

1. Empilhe a semente na pilha.
2. Enquanto a pilha não fica vazia:
   (a) Desempilhe a semente da pilha.
   (b) Marque o valor do pixel correspondente com o valor de " preenchido ".
   (c) Para cada um dos pixels adjacentes ao pixel corrente, verifique se pertence a fronteira ou já foi marcado como preenchido. Se não ocorrer nenhum dos dois casos, empilhe o pixel.
   (d) volte para o passo 2.

Um exemplo da aplicação desse algoritmo é apresentado na figura (3.5). Na figura (3.5a) apresentamos a região definida pelas fronteiras e a semente inicial S. Na figura (3.5b) apresentamos o resultado com a região preenchida.

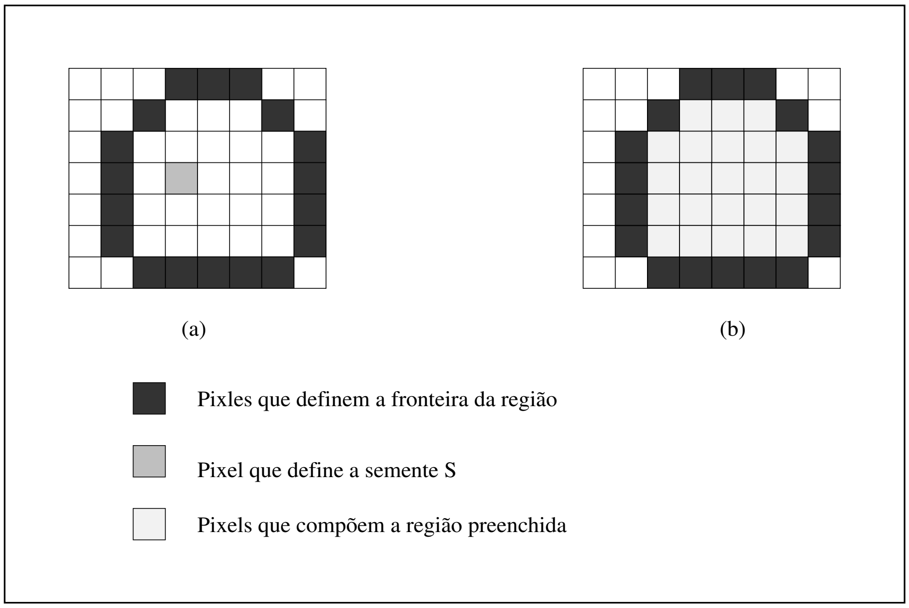


Figura 3.5: Algoritmo de preenchimento por semente clássico

No caso de imagens que apresentem descontinuidades nas fronteiras (figura (3.6a)), o algoritmo pode não funcionar corretamente, conforme ilustrado na figura (3.6b).

Para resolvermos esse caso podemos realizar uma pequena modificação no algoritmo, permitindo o uso de uma semente com dimensões maiores do que a de um pixel. Vamos definir uma semente composta por m x n pixels (figura 3.7).

Se todos os m x n pixels que compõem a semente não pertencerem à região de fronteira, a semente é marcada como percorrida e os pixels que compõem a semente são empilhados, continuando a iteração até a pilha ficar vazia. O algoritmo modificado será:

1. Empilhe a semente na pilha.
2. Enquanto a pilha não fica vazia:
   (a) Desempilhe a semente da pilha.
   (b) Marque o valor do pixel correspondente com o valor de “ preenchido ”.

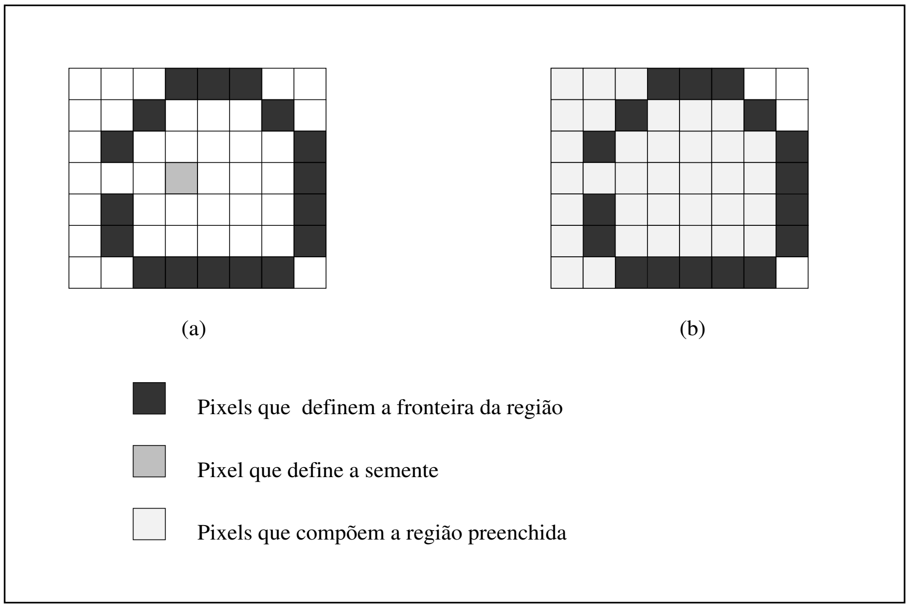


Figura 3.6: Preenchimento por semente em imagens com descontinuidade

(c) Verifique os vizinhos de acordo com a semente definida. Se todos os pixels vizinhos pertecerem não pertencerem a fronteira , empilhe todos os pixels que não foram empilhados ainda para serem processados como futuras sementes.

(d) volte para o passo 2.

É importante verificar que o resultado é dependente do número de pixels que definem a semente. Por exemplo, na figura (3.7a) a semente foi definida com 1 pixel na direção y e 2 pixels na direção x. O resultado desse preenchimento não será correto (figura (3.7b)). Um possível resultado correto (figura (3.7d)) pode ser obtido a partir da definição da semente da figura (3.7c). (2 pixels na direção y e 1 pixel na direção x).

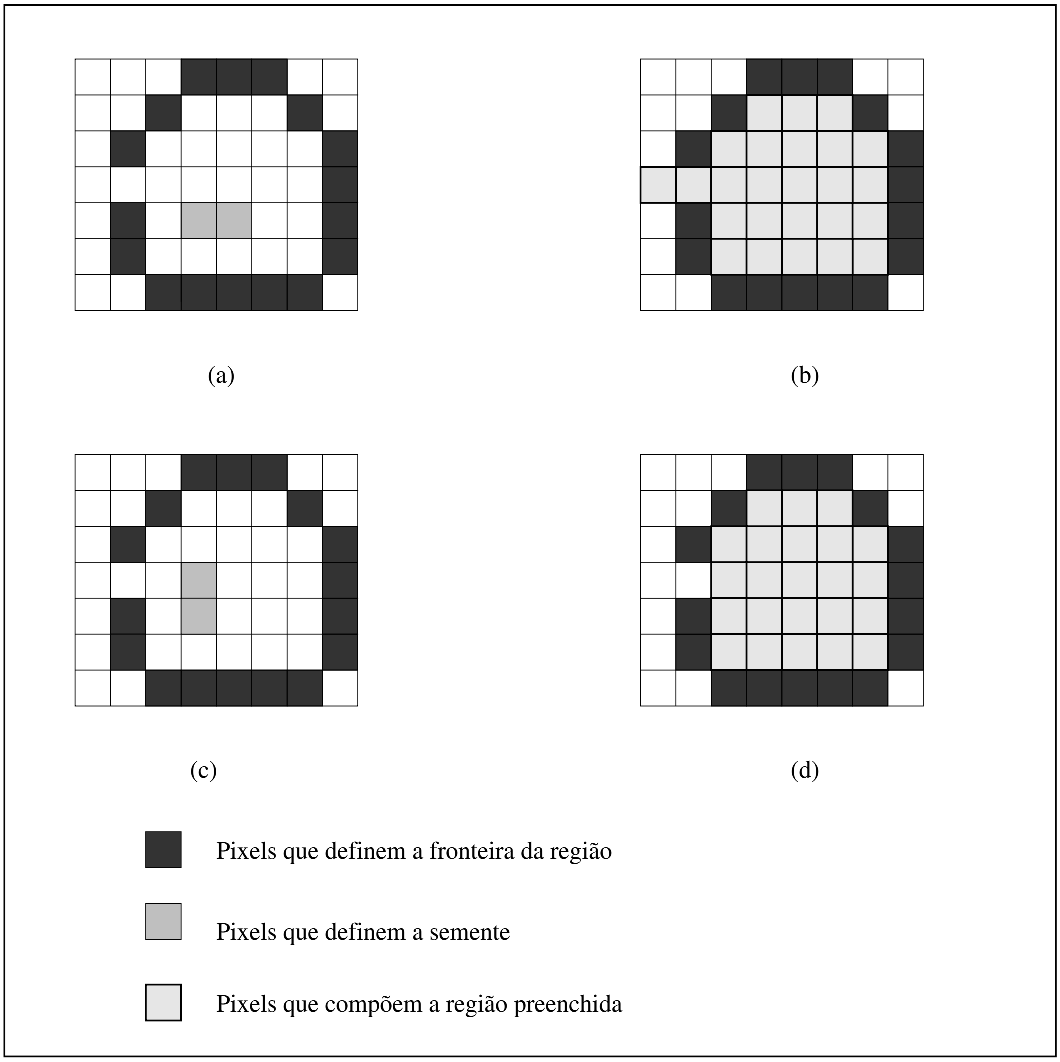


Figura 3.7: Algoritmo de preenchimento modificado

3.3 Segmentação da Imagem

Após o processo de filtragem devemos realizar a segmentação da imagem, isto é, realizar a divisão da imagem em regiões. No nosso caso o interesse é na seqüência de segmentos que definem a borda de uma região. Portanto, distingüimos as seguintes funções na segmentação:

- deteção de bordas;
- ordenamento dos pixels;
- ajuste por segmentos.

3.3.1 Detecção de Bordas

Outro objetivo do uso das técnicas de processamento de imagens é a determinação da envoltória das regiões (determinação das bordas das regiões). Após a distinção das regiões e a determinação das bordas, podemos realizar as operações apenas com os contornos das regiões. Os contornos serão úteis para o processo de interpolação das regiões, apresentado na seção (3.4).

Método Gradiente

O método mais comum para detecção de bordas é o método do gradiente.

Dada uma função *f(x,y)*, o gradiente de *f* nas coordenadas *(x,y)* é definido como o vetor:

$$\nabla \mathbf{f} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{3.7}$$

A magnitude desse vetor é dada por:

$$\nabla f = \left[\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2 \right]^{\frac{1}{2}} \tag{3.8}$$

Vamos considerar uma região composta pelos seguintes pixels:

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

Os valores z_i ($i \in \{1, 2, ...9\}$) correspondem aos valores de nível de cinza. a equação (3.8) pode ser aproximada no ponto z_5 de várias formas. A maneira mais simples é utilizar a diferença $(z_5 - z_8)$ na direção x e $(z_5 - z_6)$ na direção y, obtendo assim:

$$\nabla f \approx \left[(z_5 - z_8)^2 + (z_5 - z_6)^2\right]^{\frac{1}{2}} \tag{3.9}$$

Ao invés de usarmos potências de 2 e retirar a raiz quadrada, podemos obter valores similares usando os valores absolutos:

$$\nabla f \approx |z_5 - z_8| + |z_5 - z_6| \tag{3.10}$$

Outro modo de aproximar a equação (3.8) é usar as diferenças cruzadas:

$$\nabla f \approx \left[(z_5 - z_9)^2 + (z_6 - z_8)^2\right]^{\frac{1}{2}} \tag{3.11}$$

Usando valores absolutos teremos:

$$\nabla f \approx |z_5 - z_9| + |z_6 - z_8| \tag{3.12}$$

Essas operações podem ser implementadas utilizando-se máscaras 2x2, chamadas operadores de gradiente de Roberts:

1	0
0	-1

0	1
-1	0

Uma aproximação da equação (3.8) no ponto z_5 utilizando uma vizinhança 3x3 é dada por:

$$\nabla f \approx |(z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)| + |(z_3 + z_6 + z_9) - (z_1 + z_4 + z_7)| \tag{3.13}$$

A diferença entre a terceira e a primeira linha da região 3x3 aproxima a derivada na direção x, e a diferença entre a terceira e a primeira coluna aproxima a derivada na direção y. Essa equação pode ser implementada através do chamado operador de Prewitt:

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

Existem outras máscaras que podem ser utilizadas para a detecção de bordas, tais como:

- Sobel

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

- Isotropic

-1	-$\sqrt{2}$	-1
0	0	0
1	$\sqrt{2}$	1

-1	0	1
-$\sqrt{2}$	0	$\sqrt{2}$
-1	0	1

Método Morfológico

Os filtros morfológicos básicos (dilatação e erosão) podem ser utilizados para gerar outros tipos de filtros, por exemplo filtros para determinação de bordas de uma imagem. Podemos definir a borda de uma imagem como:

$$\text{bordas}(X) = X - (X \ominus B)$$

para um elemento estrutural B escolhido de forma conveniente.

A figura (3.8) ilustra o exemplo de uso dos filtros morfológico para a operação de deteção de bordas:

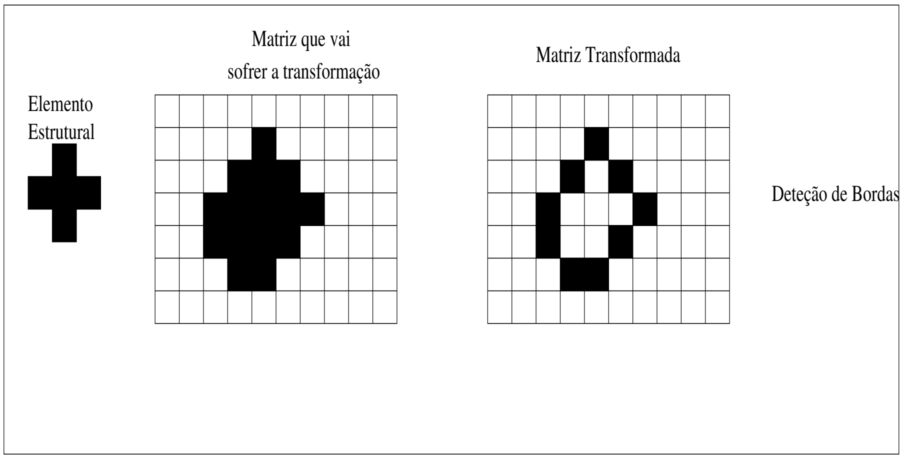


Figura 3.8: Deteção de bordas pelo método morfológico

3.3.2 Ordenamento sucessivo

Após a fronteira ser determinada, é necessário realizar o ordenamento dos pixels que compõem a fronteira. Assim a fronteira será definida por um contorno de pontos ordenados que será utilizado no processo de interpolação dos mapas.

Um algoritmo para realizar o ordenamento é o seguinte [8]:

1. Escolha um ponto inicial e uma direção inicial - no nosso caso, escolhemos o ponto com menor coordenada x e no caso de existir mais de um, o que possui menor coordenada y.
2. Vire a esquerda a caminhe até o próximo pixel se o ponto que está sendo analisado pertence a curva; caso contrário, vire à direita e caminhe até o próximo pixel.
3. Repita o passo 2 até chegar ao ponto inicial.

Na figura (3.9) apresentamos um resultado obtido com esse algoritmo.

Esse simples algoritmo pode gerar um traçado não muito linear, com alguns pixels aparecendo mais de uma vez, mas esse problema pode ser resolvido aproximando-se os segmentos obtidos por retas, fato que será discutido a seguir.

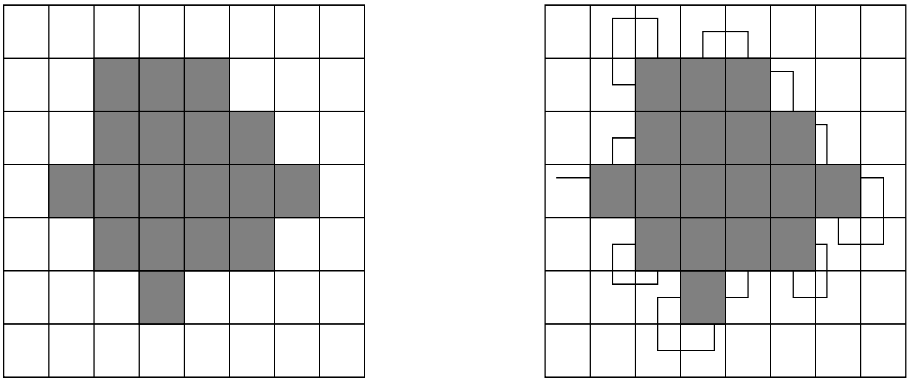


Figura 3.9: Ordenamento das fronteiras das regiões

3.3.3 Ajuste dos Contornos por Segmentos de Reta

Nas etapas sucessivas do processo de visualização (interpolação, por exemplo) o número de pontos irá influenciar significativamente o tempo de resposta do programa. Desse modo, desenvolveu-se um processo de simplificação do número de pontos gerados, aproximando os contornos por segmentos de retas. Além disso, esse algoritmo vai gerar um traçado linear para o contorno da região.

Para a definição do algoritmo vamos considerar que nosso contorno é definido pelo conjunto C_1, C_2, ... , C_n. Vamos considerar também uma variável *limite* que corresponde à distância máxima permitida entre o segmento de reta e um pixel para o ajuste.

O algoritmo é o seguinte:

1. Escolha dois pontos iniciais, que denominaremos C_i e C_{i+1}.
2. Calcule a distância D da reta formada pelos pontos C_i e C_{i+1} ao ponto seguinte, denominado C_{i+2}.
3. Se a distância for maior que o limite permitido, armazene os pontos C_i e C_{i+1} que formam a reta e repita o procedimento 2, utilizando como aproximação a reta formado pelos pontos C_{i+1} e C_{i+2}. A distância calculada será em relação ao ponto C_{i+3}
4. Se a distância for menor que o limite permitido, leia o próximo C_{i+3} ponto e repita o passo 2
5. Repita o procedimento até que todos os pontos tenham sido processados

Na figura (3.10) apresentamos 3 pontos e a distância D entre eles:

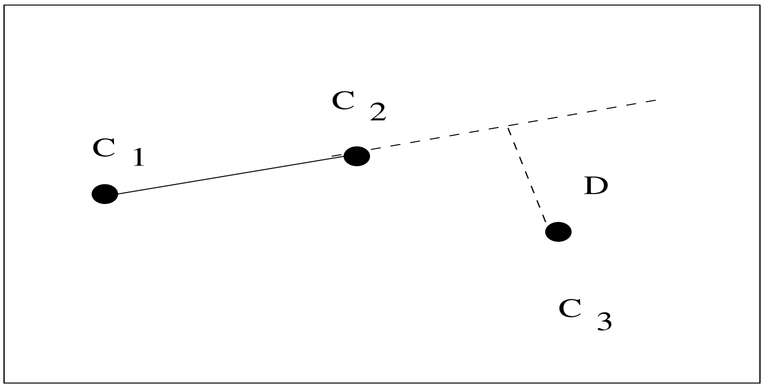


Figura 3.10: Cálculo da distância para ajuste dos contornos por segmentos de reta

3.4 Interpolação

Os conjuntos de contornos definidos no processo de segmentação podem ser interpolados permitindo a observação do comportamento dos mapas em funcão da variação do parâmetro externo. Para realizarmos a interpolação entre os mapas de Poincaré vamos utilizar um processo de reconstrução 3D.

O processo de reconstrução 3D consiste, basicamente, em se obter uma representação tridimensional do objeto permitindo além da visualização a compreensão de sua estrutura através de parâmetros geométricos do objeto. Um método de reconstrução 3D é através da observação seqüencial das imagens bidimensionais, cujas formas e dimensões são individualmente avaliadas e mentalmente reconstruídas. Quando as distâncias entre os cortes sucessivos forem pequenas, a reconstrução pode ser obtido pela sobreposição na ordem correta. No caso de distâncias maiores, os espaços vazios devem ser interpolados, processo que apresenta uma série de dificuldades (precisão, baixa confiabilidade, grande número de imagens). Desse modo, a existência de um processo automático para reconstrução e visualização torna-se importante.

Vários métodos de reconstrução tem sido propostos na literatura e, basicamente, podem ser classificados em duas categorias principais: métodos baseados em volume e métodos baseados em superfície. Os métodos baseados em volume estão normalmente associados à visualização de estruturas dimensionais internas a um objeto e, devido a este fato, são freqüentemente relacionadas à área de visualização científica. Esses métodos apresentam resultados satisfatórios quando o espaçamento entre os cortes transversais é relativamente reduzido.

Os métodos baseados em superfície são melhor aplicados na reconstrução de superfícies externas dos objetos e quando há interesse em se determinar uma representação geométrica dos objetos (determinando-se arestas e vértices que compõem o objeto reconstruído, por exemplo). Nesse caso, a reconstrução dos objetos é realizada através da aproximação das superfícies entre contornos pertencentes às secções transversais dos objetos.

O método de reconstrução utilizado no projeto de tese é através da aproximação das superfícies entre os contornos existentes nas seções transversais. Esse método utiliza um algoritmo heurístico baseado na técnica de triangulação cuja superfície dos objetos é formada por uma coleção de faces triangulares para todo par de contornos consecutivos.

Com o processo de triangulação obtemos uma representação tridimensional do objeto reconstruído. A partir dessa representação, pode-se determinar, com certa precisão, diversos parâmetros geométricos do objeto, tais como área superficial, volume, centro de massa.

Uma completa documentação do algoritmo de reconstrução desenvolvido por Pedrini é apresentada em [13].

O método utilizado apresenta as seguintes vantagens:

- Os contornos apresentando complexas concavidades são corretamente manipulados;
- As dificuldades em relação à ligação de contornos com formas e tamanhos não similares são solucionados;
- O problema de ramificação de contornos também é tratado satisfatoriamente;
- O método pode operar automaticamente, isto é, sem a necessidade de interação com o usuário durante o processo de reconstrução.

Como o objeto pode apresentar mais de um contorno por seção transversal é necessário realizar um mapeamento entre os contornos, identificando-se quais contornos estão ligados entre si para a formação das superfícies dos objetos.

No processo de mapeamento podem ocorrer os seguintes casos (Figura (3.11)):

- **Caso 1:1** - um contorno é ligado com outro contorno;
- **Caso 1:n** - ligação de um contorno com vários contornos;
- **Caso n:m** - ligação de múltiplos contornos em ambas as seções transversais.

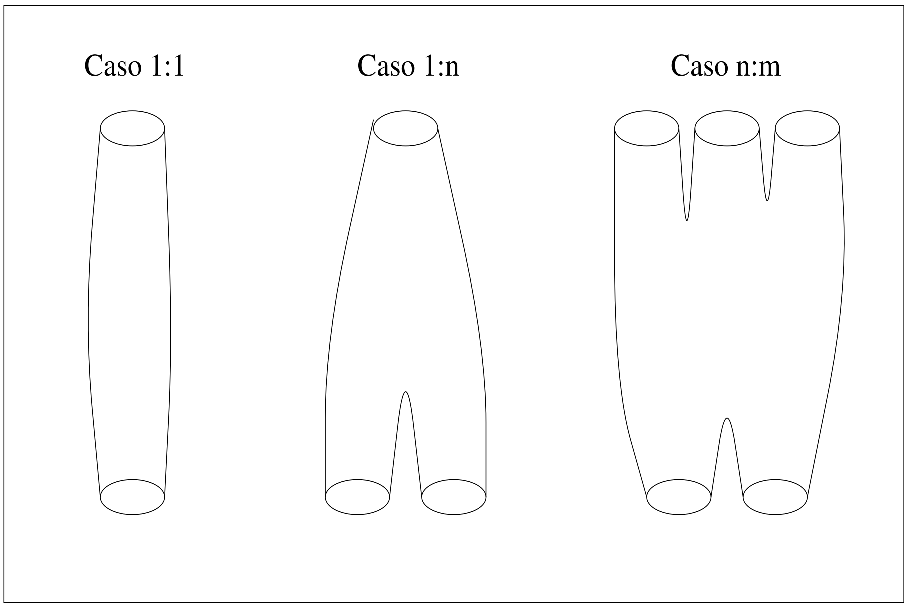


Figura 3.11: Casos de Mapeamento

Após a decisão de mapeamento é realizada a triangulação entre os contornos. A Figura (3.12) apresenta as diferentes situações adotadas pelo algoritmo. A seguir vamos detalhar cada um dos diferentes casos de mapeamento.

3.4.1 Caso 1:1

Neste caso um contorno é ligada com um contorno de outra seção transversal. O algoritmo utilizado é o de Ekoule, Peyrin e Odet [5]. A heurística desse algortimo é baseada no critério de aresta de comprimento mínimo. Os algoritmos baseados nessa heurística possui limitações quanto à manipulação de contornos que não sejam convexos ou que não apresentem formas similares. Para o caso de contornos não convexos é realizada uma transformação a fim de podermos utilizar o algoritmo.

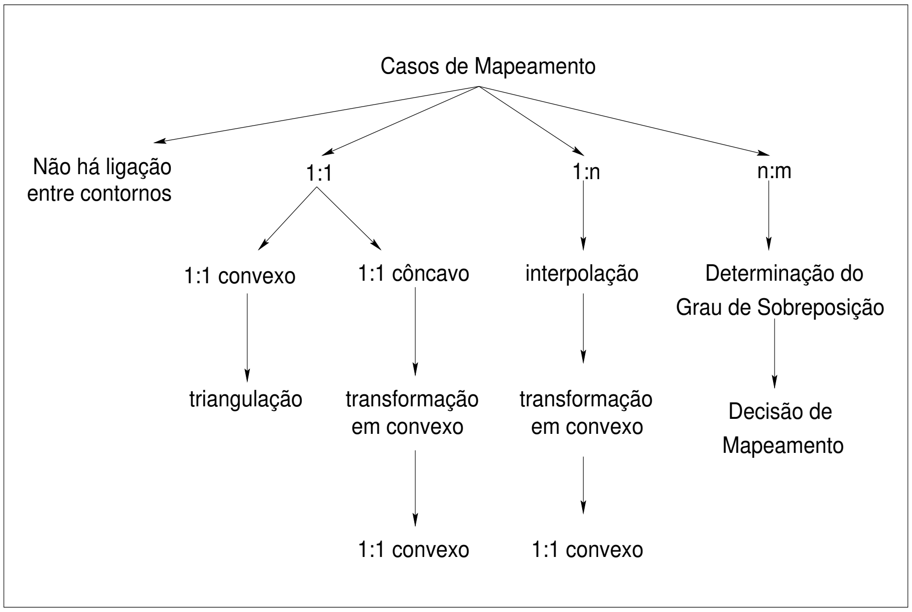


Figura 3.12: Árvore de decisões do processo de reconstrução

A fim de garantir o correto funcionamento do algoritmo de Ekoule, Peyrin e Odet é necessário que os contornos estejam mutuamente centrados. Dessa forma, os contornos são mapeados em um quadrado unitário com o centro do quadrado coincidindo com o centróide de cada contorno. Após o cálculo do centróide, os demais vértices que formam o contornos podem ser mapeados no quadrado unitário e, em seguida, a triangulação do par de contornos pode ser realizada.

Para o caso em que os contornos não são convexos realizamos uma etapa de pré-processamento onde um contorno C apresentando concavidades é transformado em um contorno convexo C', sem perda da informação relativa à distribuição de pontos no contorno original. O método baseia-se na hipótese de que o fecho convexo (*convex hull*) dos contornos, ou seja, o menor conjunto convexo contendo uma seqüência de vértices do contorno são similares em forma e orientação. A transformação do contorno C em um contorno convexo é realizada através da projeção de cada vértice de C em seu fecho convexo.

A análise do tipo de contorno (côncavo ou convexo) é baseada no algoritmo de Cyrus-Beck para recorte (*clipping*) [15]. Através do sinal obtido pelo cálculo do produto vetorial entre pares de segmentos de contorno (não colineares) podemos ter duas situações:

- polígono convexo: todos os produtos vetoriais possuem sinal positivo ou todos possuem sinal negativo;
- polígono côncavo: alguns produtos vetoriais possuem sinal positivo enquanto outros possuem sinal negativo.

O algoritmo utilizado na determinação do fecho convexo é baseado na eliminação sucessiva dos vértices responsáveis pelas concavidades do contorno, sendo estes identificados através do sinal resultante do produto vetorial.

3.4.2 Caso 1:n

Este caso corresponde ao caso de um contorno que deve ser ligado a vários contornos (um contorno é ligado a várias " ramificações "). Este problema será solucionado através da redução a um conjunto de problemas do caso 1:1.

A idéia do método é considerar todas as ramificações como sendo um único contorno fechado. Este processo envolve dois passos:

1. introduzir um novo vértice entre os vértices mais próximos das ramificações. Cada novo vértice deve ser criado no nível intermediário entre os dois níveis
2. renumerar os vértices de cada ramificação levando-se em conta a inclusão dos novos vértices, de modo que eles representem apenas um contorno. Dessa maneira, os vértices incluídos e os seus vizinhos (anterior e posterior) possuirão dois números vizinhos.

O procedimento é finalizado através da triangulação entre este contorno interpolado e o contorno de nível adjacente.

3.4.3 Caso n:m

Nese caso um conjunto de r de contornos fechados $C_{k-1,1}, C_{k-1,2}, ..., C_{k-1,r}$ no nível Z_{k-1} deve ser ligado a um conjunto de s de contornos fechados $C_{k,1}, C_{k,2}, ..., C_{k,s}$ no nível Z_k. A idéia consiste em automaticamente se dividir o problema em procedimentos 1:1 e

1:n. Deve-se realizar um mapeamento para determinar como os contornos devem ser ligados. Este mapeamento é baseado no grau de sobreposição de contornos entre fatias adjacentes.

Para dois contornos $C_{k-1,i}$ e $C_{k,j}$ pertencentes a seções adjacentes de níveis Z_{k-1} e Z_k, este valor poderia representar o cálculo da área de interseção entre estes contornos. Entretanto, utiliza-se uma aproximação desta área, requerendo menor custo computacional. Esta aproximação é realizada a partir das áreas dos retângulos (com lados paralelos aos eixos x e y) que englobam cada contorno em planos adjacentes. Um exemplo desse procedimento é observado na figura (3.13) onde apresentamos dois contornos (C_1 e C_2). O retângulo ABCD engloba o contorno C_1, enquanto o retângulo EFGH engloba o contorno C_2. A comparação será realizada entre esses dois retângulos.

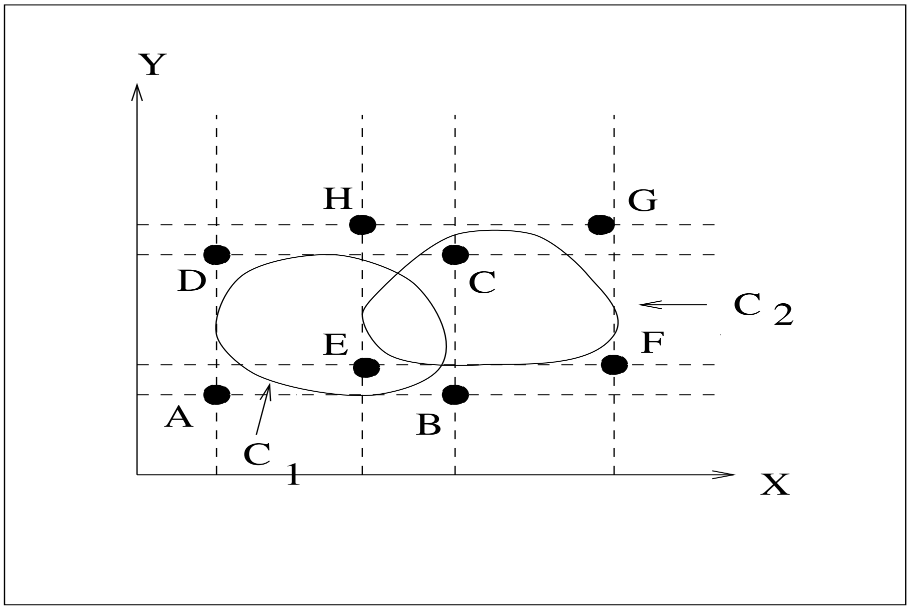


Figura 3.13: Sobreposição de dois retângulos

O grau de sobreposição dos dois contornos é definido pela área de intersecção entre estes dois retângulos. O valor do grau de sobreposição varia entre 0 e 1, indicando desde um mapeamento nulo até um mapeamento completo entre os contornos. Como forma de estabelecer um limite mínimo de sobreposição de contornos, foi criado um limiar t tal que se o grau de sobreposição for maior que o limiar então haverá ligação entre os contornos.

Esse grau de sobreposição é avaliado para cada par de contornos entre níveis adjacentes, determinando-se o tipo de ligação. Após a determinação do tipo de ligação a triangulação final pode ser realizada utilzando-se o procedimento adequado à situação.

3.5 Rendering

Rendering pode ser definido como a transformação de um conjunto de modelos computacionalmente processáveis em imagens pictoriais. Nessa seção apresentaremos as transformações geométricas sobre as representações tridimensionais dos mapas interpolados e os modelos de iluminação e tonalização para visualização dessa representação.

3.5.1 Modelos de Iluminação e Tonalização

Quando a luz encontra uma superfície ela pode ser absorvida, refletida ou trasmitida. Uma parte da energia da luz incidente é absorvida e convertida em calor. O restante é refletida ou trasmitida. Este restante que é refletido ou transmitido é que torna o objeto visível. Se toda a energia da luz incidente é absorvida, o objeto é invisível. A quantidade de energia absorvida, refletida, ou transmitida depende do comprimento de onda da luz. Se a intensidade de luz incidente é aproximadamente igual para todos os comprimentos de luz então o objeto, iluminado com uma fonte de luz branca (que contém todos os comprimentos de onda) aparece cinza. Se aproximadamente toda a luz é absorvida, o objeto aparece preto. Se apenas uma pequena fração é absorvida, ele aparece branco. Se alguns comprimentos de onda são seletivamente absorvidos, a luz que sai do objeto tem uma diferente distribuição de energia e o objeto apresentará cores. A cor do objeto será definido pelos comprimentos de onda seletivamente absorvidos.

A característica da luz refletida da superfície do objeto depende da composição, direção e geometria da fonte de luz, da orientação da superfície e das propriedades da superfície do objeto. A luz refletida de um objeto também é dividida em difusa ou especular. A luz refletida difusa é a luz I que se reflete igualmente em todas as direções, ou seja:

$$I = I_l K_d \cos\theta \qquad 0 \leq \theta \leq \pi/2, \;\; onde \tag{3.14}$$

I_l é a intensidade incidente de um ponto de luz, K_d é a constante de reflexão difusa $(0 \leq K_d \leq 1)$, e θ é o ângulo entre a direção da luz e a normal da superfície (figura (3.14)).

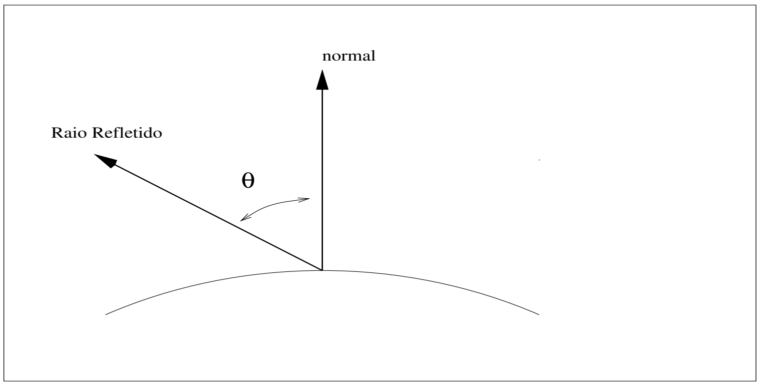


Figura 3.14: Definição do ângulo θ

A luz refletida especular é a que reflete em uma direção preferencial. A intensidade da luz refletida especularmente depende do ângulo de incidência, do comprimento de onda da luz incidente e das propriedades do material. A reflexão especular é direcional. Para uma superfície de reflexão especular perfeita, o ângulo de reflexão é igual ao de incidência. Desse modo, apenas um observador localizado exatamente nesse ângulo observará qualquer luz refletida especular. Isto significa que o vetor de visão S (figura (3.15) é coincidente com o vetor de reflexão R, isto é, o ângulo α é zero. Para superfícies com reflexão imperfeita a quantidade de luz que chega ao observador depende da distribuição espacial da luz refletida especular, representada por $\cos^n \alpha$.

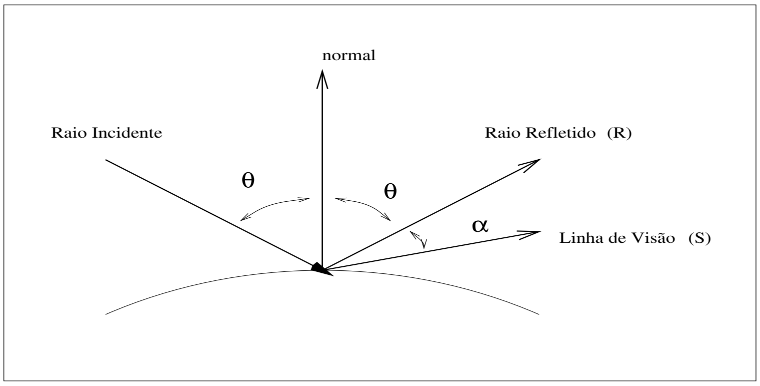


Figura 3.15: Reflexão Especular da Luz

Por causa das complexas características físicas da luz refletida especularmente, um modelo empírico de Bui-Tuong Phong é normalmente usada para o modelo de iluminação (equação (refeq4))

$$I = K_s \cos^n \alpha \tag{3.15}$$

onde K_s corresponde a uma constante experimental e $\cos^n \alpha$ representa uma função de distribuição espacial. Valores grandes de n estão associados a materiais com características de metais e outras superfícies brilhantes, enquanto valores pequenos correspondem a materias com pouco "brilho", por exemplo papel.

O modelo pode levar em conta também a luz ambiente. Desse modo a equação ficará definida como:

$$I = I_a k_a + I_l(K_d \cos\theta + K_s \cos^n \alpha) \qquad 0 \leq \theta \leq \pi/2, \;\; onde \tag{3.16}$$

I_a é intensidade da luz ambiente e k_a é a constante de reflexão difusa do ambiente $(0 \leq K_a \leq 1)$.

Se o modelo de iluminação considerar que a intensidade de luz diminui inversarmente com o quadrado da distância das fontes de luz, então:

$$I = I_a k_a + \frac{(K_d \cos\theta + K_s \cos^n \alpha)}{d + K} \qquad 0 \leq \theta \leq \pi/2, \;\; onde \tag{3.17}$$

K é uma constante arbitrária e d a distância entre os objetos.

Se múltiplas fontes de luz estão presentes, os efeitos são linearmente adicionados, resultando na equação 3.18:

$$I = I_a k_a + \sum_{j=1}^{m} \frac{I_l}{d + K}(K_d \cos\theta + K_s \cos^n \alpha) \qquad 0 \leq \theta \leq \pi/2 \tag{3.18}$$

Essa equação é normalmente utilizada para determinar a intensidade de cada ponto do objeto ou cada pixel apresentado. Por questões de eficiência, utilizam-se técnicas de tonalização (*shading*) para interpolar as intensidades obtidas pela equação (3.18). Na técnica de Flat Shading considera-se que a intensidade em cada face é constante. Uma aparência mais suave pode ser obtida usando a técnica de tonalização desenvolvida por Gourard. Nesta técnica determina-se a intensidade para cada vértice do polígono e a intensidade dos demais pixels é obtida através da interpolação. Uma terceira técnica de tonalização é conhecida como Phong shading. Enquanto Gourard shading interpola os valores de intensidade em cada vértice do polígono, Phong shading interpola os vetores normais. O modelo de iluminação é então aplicado a cada pixel, usando o vetor normal interpolado para determinar a intensidade. Esta técnica apresenta uma aproximação local melhor para a curvatura da superfície e um rendering melhor para a superfície. Em particular, objetos especulares aparecem mais realísticos.

3.5.2 Transparência

Alguns objetos apresentam a propriedade de transmitir a luz, por exemplo vidros, janelas de automóveis. Quando a luz atravessa meios diferentes, por exemplo do ar para água, o raio de luz é refratado.

A lei de *Snell* (figura (3.16)) define a relação entre o ângulo de incidência, o ângulo de refração e os índices de refração dos meios:

$$n_1 \sin\theta = n_2 \sin\alpha \tag{3.19}$$

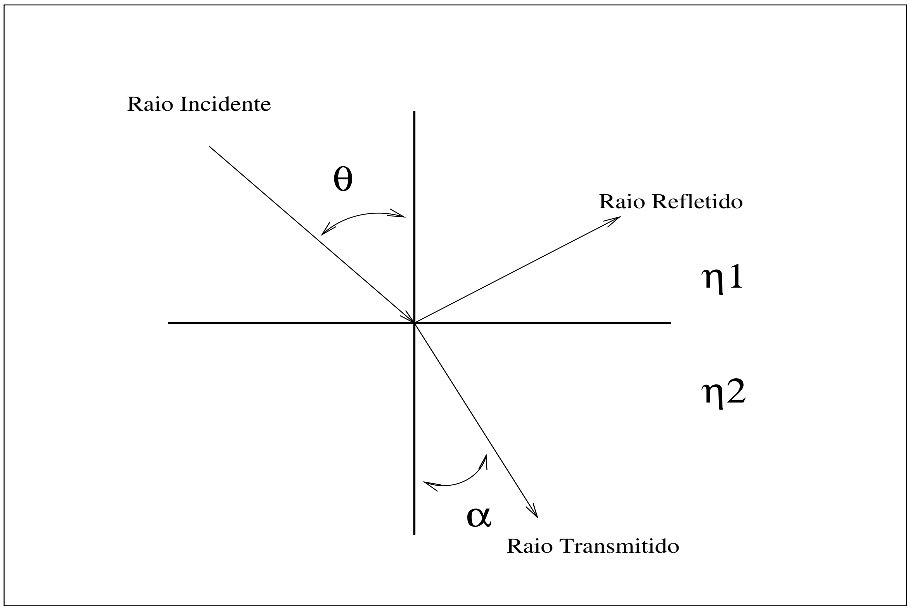


Figura 3.16: Lei de Snell

A transparência refrativa pode ser um processo computacionalmente custoso. Por esse motivo, na maioria das implementações são consideradas as transparências não refrativas.

Um dos métodos comumente utilizados é a transparência interpolada. Para descrever a transparência interpolada vamos considerar os polígonos 1 e 2 e um observador na posição O, conforme apresentado na figura (3.17).

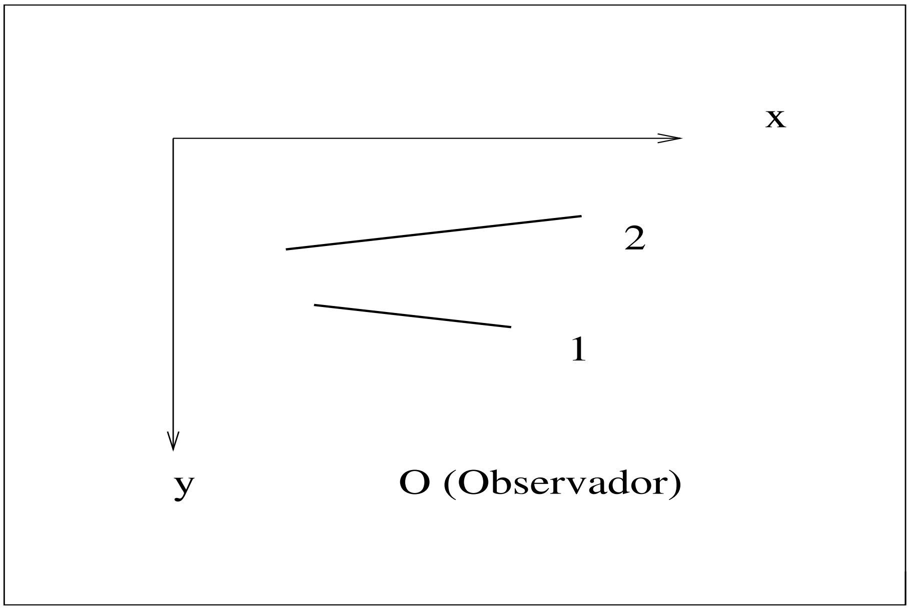


Figura 3.17: Transparência interpolada

A transparência interpolada é dada pela equação:

$$I_{\lambda} = (1 - K_{t1})I_{\lambda 1} + K_{t1}I_{\lambda 2}. \tag{3.20}$$

O coeficiente de transmissão K_{t1} representa a transparência da superfície 1 ($0 \leq K_{t1} \leq 1$). Quando K_{t1} é igual a 0, a superfície 1 é opaca e não transmite nenhuma luz. Quando K_{t1} é igual a 1, a superfície é completamente transparente.

Uma aplicação de transparência é a visualização do interior de objetos complexos.

3.5.3 Transformações Geométricas

As transformações geométricas são operações matemáticas realizadas em um objeto que permitem a visualização do objeto em diferentes posições ou escalas. As transformações mais comuns são a translação, rotação e a mudança de escala. Cada transformação pode ser representada por uma matriz 4x4. As transformações normalmente são realizadas através de multiplicação de uma matriz (que representa as coordenadas dos pontos que compõem um objeto) e pela matrize de transformação correspondente. Resumidamente, apresentaremos as matrizes referentes a cada uma das transformações:

- Translação:

$$\mathrm{T} = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Rotação:
  - Rotação de um ângulo θ em torno do Eixo X:

$$\mathrm{Rx} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - Rotação de um ângulo θ em torno do Eixo Y:

$$\mathrm{Ry} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - Rotação de um ângulo θ em torno do Eixo Z:

$$\mathrm{Rz} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Mudança de escala:

$$S = \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Capítulo 4

Decisões de Projeto e uma Implementação

A implementação do protótipo foi realizado em linguagem C (gcc). O desenvolvimento foi realizado em um micro computador 586 com 16 MB de RAM utilizando sistema operacional Unix (FreeBSD 2.1.0). Para os testes no IFGW utilizamos uma estação de trabalho Sun (SparcStation 10 com 96 MB de RAM) e sistema operacional SunOS 4.1.3B. O programa foi desenvolvido com o intuito de permitir fácil porte para as diversas arquiteturas Unix do IFGW (SunOS, Solaris, OSF/1, AIX, FreeBSD).

O programa foi desenvolvido em vários módulos. A nossa expectativa é que esses módulos sejam aperfeiçoados e possam ser facilmente substituídos.

Nessa seção apresentaremos o algoritmo desenvolvido para realizar a visualização e manipulação dos mapas de Poincaré. Serão apresentados os módulos e suas funções, técnicas e algoritmos usados no protótipo. Os detalhes da implementação estão apresentados no apêndice A.

O protótipo pode ser dividido em 3 grandes módulos:

1. **Módulo para processamento e análise de um mapa de Poincaré:** Esse módulo é o responsável pela análise inicial dos mapas. Os mapas de Poincaré normalmente são formados por um conjunto de pontos gravados em arquivos textos. Nesse módulo serão realizadas as seguintes tarefas:

- Os arquivos de dados serão lidos do disco, armazenados em uma estrutura de dados conveniente e apresentados para o usuário em um gráfico xy (corresponde à etapa de Pré-processamento);
- O usuário poderá interagir com esses dados, setando as cores desejadas para os pontos, as escalas, os títulos para os eixos e para o gráfico gerado, eliminando curvas não desejadas;
- O usuário deverá interagir com o programa realizando a determinação das regiões normais e caóticas. Após a distinção das regiões, o contorno das regiões será gerado (corresponde às etapas de Filtragem e Segmentação).

2. **Módulo para análise de um conjunto de mapas de Poincaré:** Após a realização das tarefas acima descritas, o usuário poderá realizar operações com o mapa gerado, analisando propriedades básicas de cada mapa ou operações entre os mapas (subtrair um mapa do outro, por exemplo)
3. **Módulo para visualização de mapas de Poincaré interpolados:** O usuário poderá realizar a interpolação de mapas de Poincaré a partir dos contornos definidos pelo usuário. Após a interpolação o usuário poderá visualizar os mapas interpolados, realizando operações de rotação, translação e cortes em planos paralelos ao plano xy (corresponde às etapas de Interpolação e Rendering).

Apresentamos na figura (4.1) um esquema com os três módulos básicos.

A seguir apresentamos a descrição detalhada dos módulos, com as técnicas utilizadas, bem como as ferramentas e os algoritmos.

4.1 Módulo para processamento e análise de um mapa de Poincaré

4.1.1 Leitura dos pontos que compõem um mapa

O primeiro passo para a análise dos mapas de Poincaré corresponde à leitura dos pontos que compõem cada mapa. Esses pontos são gravados em arquivos do tipo texto com duas colunas que se referem às coordenadas (x,y) e várias linhas. Cada par de coordenadas

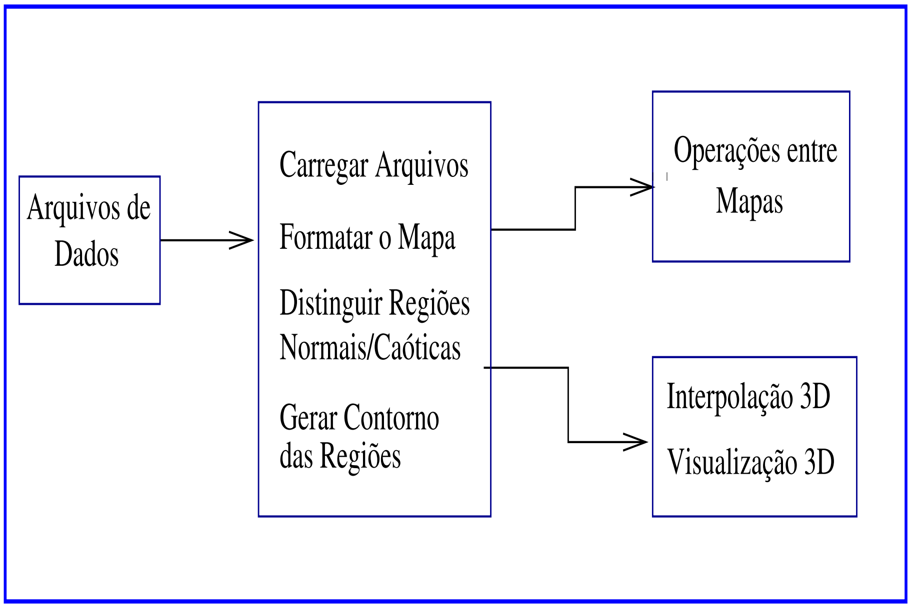


Figura 4.1: Módulos do protótipo de visualização e manipulação

corresponde a um ponto no mapa de Poincaré. O número de linhas varia de acordo com a simulação, mas geralmente é acima de 1000 linhas. Geralmente um mapa de Poincaré será formado por mais de um arquivo (devido ao processo de geração).

É importante observar que a proximidade nos pontos dos arquivos de dados não representa uma proximidade geométrica dos pontos que compõem o mapa de Poincaré. A ordem na qual os pontos aparecem nos arquivos de dados é uma consequência do processo de simulação, não gerando nenhuma informação adicional para a análise.

O número de arquivos gerados normalmente está relacionado com o número de pontos iniciais que são usados para gerar o mapa. Normalmente esse número de pontos iniciais pode ser aumentado ou diminuído, dependendo do tipo de curva gerada e da análise realizada pelo pesquisador. Não há maiores problemas em gerar sempre com o mesmo número de pontos, mas esse fato não deve se tornar uma restrição do programa.

Não existe uma relação direta entre o número de arquivos e regiões. Uma região pode ser formada por mais de um arquivo de dados e em um arquivo de dados pode existir mais de uma região (esse fato só ocorre com regiões regulares - um arquivo pode conter 2 regiões regulares, mas nunca uma região regular e outra caótica).

Outro aspecto que deve ser considerado é que a ordem em que os pontos são dispostos nos arquivos não traz nenhuma informação adicional. Assim não perderemos nenhuma informação ao desprezar a ordem dos pontos.

Desde que nosso objetivo é desenvolver técnicas que ajudam em um análise visual dos mapas, decidiu-se tratar os mapas como imagens compostas por regiões preenchidas e não preenchidas. Assim a primeira tarefa é converter o conjunto de pontos representados no espaço objeto para o espaço imagem.

O programa que realiza a conversão leva em conta os seguintes parâmetros:

- Valores máximos e mínimos para os eixos x e y
- Dimensão da imagem gerada (tamanho da matriz gerada)

Adotamos para nossos testes imagens com dimensão 512x512, um valor razoável para processamento e suficiente para os dados apresentados.

Como principal conseqüência dessa conversão, utilizamos algoritmos de processamento de imagens para segmentar os mapas de Poincaré, distinguindo as regiões.

Outra vantagem dessa conversão é que, embora os mapas de Poincaré possam envolver milhares de pontos, a complexidade dos algoritmos de processamento subseqüentes é apenas dependente de um tamanho de imagem pré-definido.

Apesar do volume de pontos que compõem um mapa de Poincaré, ainda existem descontinuidades na imagem considerando-se o conceito de região 4 ou 8-conexas. Essas descontinuidades representam um problema quando distinguirmos as regiões. Esse assunto será tratado em detalhes na seção (4.1.3).

4.1.2 Eliminando curvas

No processo de geração dos pontos que compõem os mapas de Poincaré, algumas curvas são geradas e não trazem nenhuma informação adicional para a análise. Como a identificação destas curvas é dependente do contexto, permitimos ao usuário especialista (pesquisador) que essa curva seja selecionada e eliminada, utilizando-se o *mouse*.

Quando um ponto inicial utilizado no processo de simulação pertence a uma região normal, os pontos calculados irão gerar uma curva fechada bem comportada. O interesse do pesquisador não é na curva fechada propriamente dita e sim na determinação da fronteira entre as regiões. Assim a curva bem comportada apenas significa que estamos em uma região normal e não necessariamente na fronteira da região. Para facilitar a compreensão vamos observar um exemplo na figura (4.2):

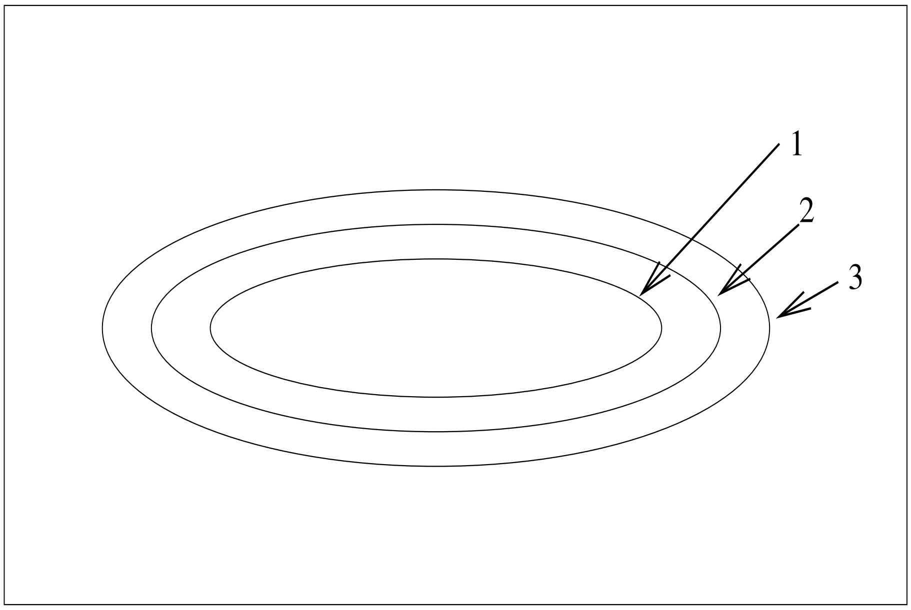


Figura 4.2: Eliminando curvas redundantes

As curvas 2 e 3 apenas reforçam a idéia de que a região a que pertencem é uma região normal, mas não pertencem a fronteira da região normal. Nesse caso, uma possível fronteira seria determinada através da curva 1 (considerando-se apenas as 3 curvas apresentadas). No nosso processo marcamos a região normal e posteriormente determinamos as bordas. Se não eliminarmos as curvas 2 e 3 o processo pode não atingir o resultado desejado. Assim a correta eliminação de curvas " redundantes " é fundamental para as demais etapas da análise.

4.1.3 Distinção das regiões normais e caóticas

Mesmo após a conversão para uma matriz bidimensional ainda há um grande número de pontos, tornando a interpolação inviável. Além disso, os pontos apresentados não estão ordenados. Numa tentativa de diminuir o número de pontos para posteriores manipulações, surgiu a idéia de determinar a envoltória das regiões.

Realizamos testes com as técnicas apresentadas na seção (3.2), numa tentativa de automatizar o processo de segmentação. Inicialmente aplicamos filtros suavizantes numa tentativa de diminuir a descontinuidade na imagem gerada. O resultado obtido com esses filtros não foi satisfatório, pois os padrões das regiões normais e caóticas em um mapa podem diferir bastante, dependendo do número de amostras e das condições iniciais consideradas. Além disso, as imagens geradas apresentavam características diferentes das desejadas, normalmente apresentando sobreposição das regiões, causada pela suavização.

Resolvemos optar por um processo de filtragem morfológica interativa, onde permitimos que as regiões sejam classificadas corretamente através de uma interface amigável com o usuário.

A idéia básica é apresentar um mapa de Poincaré e permitir que as regiões normais sejam identificadas. As regiões restantes serão consideradas regiões caóticas. Com relação à região caótica, o interesse do pesquisador é na envoltória da região. Assim com a experiência do pesquisador ele consegue analisar que, mesmo sem ter sido gerada uma curva, existe uma curva que fica na transição entre a região caótica e normal. O pesquisador gostaria que tal "curva" fosse "detectada" pelo programa, realçando ainda mais a necessidade de interação no programa. Utilizando a técnica de crescimento por semente (seção (3.2.6)), o usuário poderá determinar o tamanho da semente a ser utilizada através da interface e utilizará o mouse para posicionar a semente inicial, participando interativamente da determinação das regiões. Após o processamento a partir da semente, o resultado é apresentado para o usuário. Caso o resultado obtido não corresponda ao desejado, o usuário pode selecionar uma opção para desfazer a operação (*undo*), voltando ao estado anterior e selecionando novas dimensões para a semente. Além da opção de *undo*, o usuário pode desmarcar as regiões selecionadas, simplesmente selecionando a opção *desmarcar* e posicionando o mouse sob a região a ser desmarcada.

4.1.4 Determinação dos pontos que compõem as fronteiras das regiões

Após a distinção das regiões é necessário determinar os contornos dessas regiões, realizando o ordenamento dos pontos das bordas. Utilizamos o algoritmo descrito na seção

(3.3.1). Uma vez que esse algoritmo pode gerar um traçado não muito linear o contorno da região será ajustado por retas (seção (3.3.2)).

4.1.5 Propriedades

Uma vez determinadas as regiões podemos gerar um relatório com informações referentes ao mapa, tais como :

- Arquivos que compõem um determinado mapa e o número de pontos de cada arquivo, com o objetivo de permitir ao pesquisador uma informação precisa sobre o conjunto de dados que está sendo analisado;
- Arquivos cujos pontos foram " eliminados " pelo usuário para realizar a análise. O processo de eliminação de curvas é baseado na experiência do pesquisador. Assim é importante deixar registrado quais curvas foram consideradas " redundantes " pelo pesquisador, permitindo que o mesmo resultado de análise seja reproduzido por outro pesquisador (funcionando como um arquivo de registros sobre as operações realizadas com os mapas);
- Percentagem da região normal e caótica. A estimativa de como as regiões estão aumentando ou diminuindo é uma importante informação para o pesquisador.

4.2 Módulo para análise de um conjunto de mapas de Poincaré

Nesse módulo podem ser analisados conjuntos de mapas, realizando operações como a subtração de dois mapas (para verificar a variação entre dois mapas adjacentes, por exemplo). O algoritmo utilizado é bastante simples, pois os mapas estão mapeados em uma matriz, bastando realizar a subtração entre os elementos da matriz.

4.3 Módulo para interpolação e visualização de mapas de Poincaré

4.3.1 Interpolação dos mapas de Poincaré

O processo de geração de mapas de Poincaré pode ser computacionalmente custoso. Pretendemos, com a interpolação, apresentar uma " idéia aproximada " do comportamento dos mapas, baseado nos mapas adjacentes.

Algumas propriedades dos mapas podem ser observadas:

- Os contornos de uma dada região tem uma certa coerência com relação à evolução no espaço do parâmetro externo. Assim se um contorno A é interior ao contorno B, não vai existir um mapa a ser interpolado tal que o contorno B esteja no interior de A. Na figura (4.3) mostramos um exemplo desse tipo de coerência.

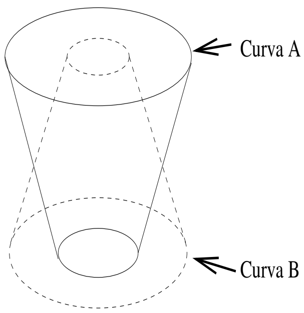


Figura 4.3: Coerência dos contornos dos mapas de Poincaré

- Os contornos podem se subdividir. Por exemplo podemos ter uma região formada por apenas um contorno e este se subdividir em outros dois contornos (figura (4.4)).

Para a interpolação dos mapas utilizamos a técnica de reconstrução 3D apresentada na seção 3.4.

A implementação original foi desenvolvida em vários módulos:

- Um módulo principal em linguagem C com rotinas em xview e chamadas de shell scripts, realizando o desenvolvimento da interface com usuário, mapeamento dos contornos e chamada dos demais módulos (reconstr3d.c);

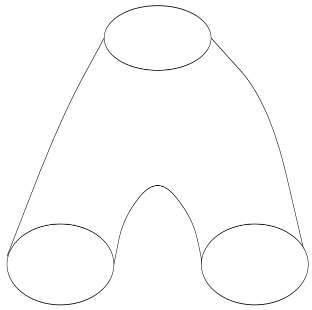


Figura 4.4: Subdivisão dos contornos dos mapas de Poincaré

- módulo em linguagem Pascal responsável pela separação dos contornos (separa_cont.pas) no arquivo de entrada dos dados;
- um módulo em linguagem Pascal responsável pela triangulação entre os contornos (triangula.pas);
- Um módulo em linguagem Pascal responsável pela determinação das medidas do objeto reconstruído (medidas.pas).

A comunicação entre os diversos módulos é realizada através de arquivos com formatos específicos

Para a visualização do resultado obtido foi utilizada a ferramenta PROSIM (Prototipação e Síntese de Imagens Foto-Realistas), desenvolvido pelo Grupo de Computação de Imagens do Departamento de Engenharia de Computação e Automação Industrial (DCA) da FEEC-UNICAMP.

Para a integração com a nosso protótipo, realizamos algumas alterações na implementação original:

- O programa utilizava o *toolkit xview*. No nosso caso estamos utilizando o *toolkit Xforms*. Desse modo, eliminamos toda a referência as bibliotecas xview;
- Existiam módulos desenvolvidos utilizando a linguagem Pascal. Com o objetivo de permitir grande portabilidade, convertemos todo o código em Pascal para linguagem C, usando o conversor p2c [1] . Em alguns casos realizamos alterações no código convertido;

[1] Pascal to C translator ,desenvolvido por Dave Gillespie (e-mail daveg@synaptics.com)

- Para a visualização não utilizamos o PROSIM. Desse modo, alguns scripts utilizados para converter no formato B-Rep não foram mais utilizados. No nosso caso, podemos utilizar a saída do programa de triangulação para realizar a visualização;
- Foram alterados alguns parâmetros para permitir um número maior de contornos por seção transversal;
- Foram verificados e corrigidos alguns problemas implementação do algoritmo de reconstrução. Os problemas detectados bem como as soluções estão apresentados no apêndice C.

Os dados utilizados para reconstrução são passados através de arquivos textos, mantendo-se o formato original do programa de reconstrução. A saída do programa de reconstrução também é através de arquivos texto, que são lidos pela rotinas de visualização 3D.

4.3.2 Visualização dos mapas interpolados

O módulo de visualização utiliza os dados gerados a partir do programa de interpolação. O resultado do programa de interpolação corresponde a um conjunto de faces triangulares. A partir dessas faces são calculados os vetores normais de cada vértice.

A imagem dos mapas interpolados é apresentada ao usuário que pode realizar operações de rotação, mudança de escala, mudança de cores e cortes em planos na direção z.

A implementação das rotinas de visualização foram realizadas utilizando-se a biblioteca *Mesa* que será apresentada na seção (5.3). O resultado do processo de interpolação é um conjunto de arquivos que correspondem às faces do objeto interpolado. Esses vértices são lidos dos arquivos e armazenados em listas de vértices para utilização pelas rotinas do OpenGL/Mesa.

O usuário pode visualizar o objeto de diferentes posições, realizando transformações geométricas (rotação, translação e escala). É permitido ao usuário realizar a alteração nas cores do objeto visualizado, alterando as fontes de luz e propriedades do objeto (transparência, por exemplo).

Também é permitido ao usuário realizar operações de cortes nos mapas interpolados, utilizamos planos adicionais de recorte (*clipping*). Uma vez definido o ponto e a direção do plano de corte, determinamos um plano de corte e visualizamos o objeto resultante.

4.4 Interface com o usuário

A usabilidade dos sistemas tem se tornado cada vez mais importante. A usabilidade está relacionada com a facilidade de aprendizado, a alta velocidade para o desempenho de tarefas, a baixa taxa de erros e a satisfação subjetiva do usuário. Trata portanto, da efetividade e eficiência da interface do usuário e da relação do usuário com a interface. A naturalidade da interface também é um aspecto importante da usabilidade.

Para o usuário de um sistema interativo, a forma de comunicação com o sistema é tão importante quanto a computação. Para usuários, a interface é o sistema. Mesmo com todos os recursos de botões, *menus*, cores, gráficos, os problemas de interação homem-maquina ainda não foram resolvidos. Esse fato pode ser observado nas próprias experiências do dia-a-dia.

Um elemento muito importante no protótipo foi a interação do usuário com as rotinas de manipulação dos mapas. O desenvolvimento de uma interface amigável e funcional é fundamental para o sucesso do programa.

A interface foi desenvolvida com o *toolkit Xforms* (apresentada na seção 5.1), mas a simples utilização do *toolkit* não garante uma boa interface. Não existe uma regra exata para a construção de interfaces com alto grau de usabilidade, mas algumas diretrizes podem orientar no processo de desenvolvimento. Um dos mais conhecidos é a coleção de 944 diretrizes compiladas por Smith e Mosier no *MITRE Corporation* [16]. Para o desenvolvimento da interface do protótipo, consideramos as seguintes diretrizes:

- O projeto é centrado no usuário;
- Existe suporte aos erros do usuário;
- As operações do usuário são optimizadas;
- A interface é consistente e simples;

- Uso de "feedback" para as ações do usuário;
- Apresentamos ao usuário o *status* atual;
- Uso de termos específicos nas mensagens de erro;
- As ações do usuário são facilmente reversíveis (*undo*);
- O display é mantido " inerte ", conservando a mesma seqüência de janelas;
- As janelas são organizadas com o objtivo de diminuir a complexidade e facilitar o uso do programa.

A interface desenvolvida pode ser esquematizada do seguinte modo:

- Região dos botões principais: ativam os 3 módulos principais, além da opção de *Sair* e *Ajuda.*
- Região dos botões de comandos para cada um dos módulos
- Janela de *Status* corrente
- Janela de Feedback
- Janela de Gráficos
- Janela de Ajuda, que é aberta quando pressiona-se o botão de *Ajuda*
- Janelas para visualização dos dados tridimensionais

A figura (4.5) apresenta um esquema geral da interface do sistema.

Foi dedicada atenção na interação do usuário com os botões, desabilitando-os quando não devem ser usados, colocando os feedbacks e confirmacoes para eventos que necessitam maior atenção.

No apêndice B são apresentadas diversas telas do protótipo onde as características acima citadas podem ser observadas

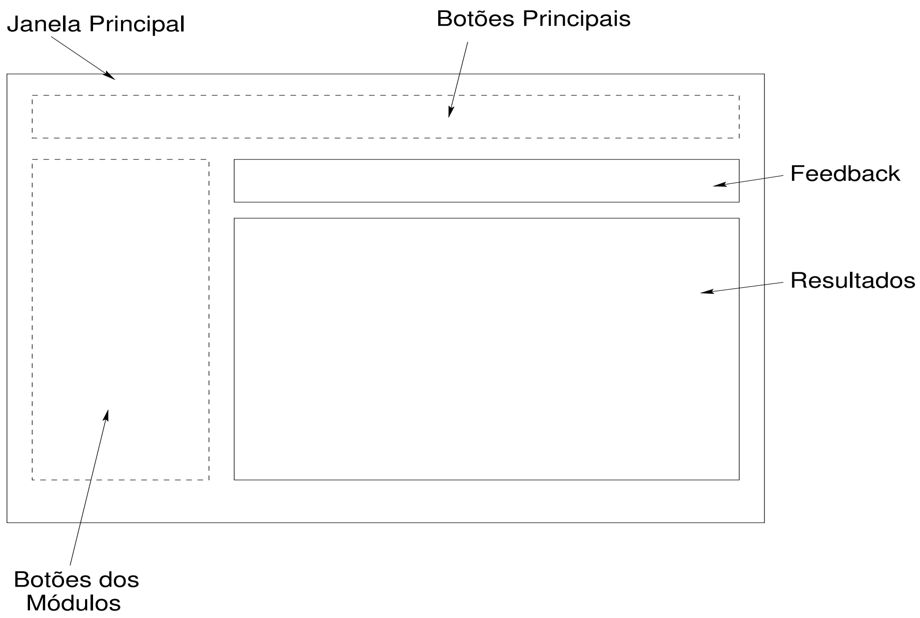


Figura 4.5: Esquema geral da interface do protótipo

Capítulo 5

Ferramentas de Suporte

No desenvolvimento do propótipo tentamos aproveitar ao máximo pacotes de domínio público, evitando reescrever código. Os seguintes pacotes foram utilizados:

- Xforms: Para o desenvolvimento da interface gráfica;
- Morph: Para operações de morfologia;
- Mesa: Para a visualização 3D;
- Xwd, Xpr: Para converter as imagens geradas em tela para arquivos em formato Postscript (impressão).

Apresentaremos a seguir os pacotes utilizados mais detalhadamente.

5.1 Xforms

As interfaces gráficas baseadas em janelas estão se tornando uma característica necessária para a maioria dos sistemas de computação e como resultado os usuários esperam que os programas apresentem uma interface gráfica amigável. Como a tarefa de construção de interfaces normalmente consome um tempo considerável, vários programas que auxiliam a construir interfaces gráficas para usuários (*Graphical User Interface* - GUI) tem sido desenvolvidos. Esse programas são conhecidos como *GUI Toolkits*.

No projeto utilizou-se um *toolkit* chamado *Xforms* desenvolvido por T.C.Zhao (Departamento de Física da Universidade de Wisconsin-Milwaukee - EUA) e Mark Overmars (Departamento de Ciência da Computação da Universidade de Utrecht - Holanda) [17].

A biblioteca *Xforms* foi desenvolvida para o sistema de janelas X, utilizando as rotinas *Xlib*. Possue um rico conjunto de *widgets* tais como botões, sliders e menus integrados em um modelo de execução de rotinas de retorno (*callback*) que permitem uma construção rápida e fácil de aplicações em X. A biblioteca consiste de um conjunto flexível de rotinas em C que podem ser utilizadas em programas em C ou C++ (no futuro pretende-se disponibilizar para outras linguagens).

A biblioteca é eficiente e portável, podendo ser utilizada na maioria das plataformas Unix, incluindo Sun, SGI, HP, DEC Alpha/OSF, IBM RS6000, Convex, Cray, i386/Linux, i386/FreeBSD, i386/NetBSD, SCO, Unixware, DecSystem(mips)/Ultrix e outras.

Juntamente com a biblioteca existe uma ferramenta para projeto de interface chamada fdesign (*Forms Library User Interface Designer*) e é baseada no princípio WYSIWYG (*what you see is what you get* - o que você vê é o que você obtém) permitindo a geração de código em C correspondente à interface projetada.

A principal noção na biblioteca *XForms* é o que chamamos *form*. Um *form* é uma janela na qual diferentes objetos são colocados. Quando o usuário modifica o estado de um particular objeto em algum *form* a aplicação é notificada e uma ação apropriada pode ser executada (definida pela rotina de *callback*).

A aplicação desenvolvida tem grande controle sobre os objetos que são apresentados no *forms* (cor, forma, estilo, tamanho e cor do texto, etc).

A biblioteca *Xforms* é extensível, permitindo que novos objetos (*free objects*) sejam criados pelo programador. Nesse caso deve-se utilizar a arquitetura definida pela biblioteca *Xforms*, permitindo-se que esses novos objetos sejam utilizados em conjunto com os demais.

Algumas rotinas que são muito utilizadas são fornecidas para facilitar o desenvolvimento de interfaces. Desse modo, podemos com apenas uma linha de comando (chamada da rotina) realizar operações normalmente complicadas. Um exemplo é o objeto fselector (*file selector* - selecionador de arquivos) que permite ao usuário navegar em um diretório e selecionar o nome de um arquivo desejado.

A seguir faremos uma breve descrição sobre os objetos disponíveis na biblioteca *Xforms*.

- Objetos Estáticos
  - **Box:** Áreas retangulares para visualmente agrupar os objetos. São utilizados para dar uma aparência mais agradável a interface
  - **Frame:** Corresponde a um Box com a região interna não preenchida. Também é utilizada para agrupar visualmente os objetos
  - **Text:** Consistem de Textos que podem ser colocados em um Box, por exemplo.
  - **Bitmap:** Apresenta um bitmap (X11) no forms
  - **Pixmap:** Apresenta um pixmap usando a biblioteca xpm no forms
  - **Clock:** Apresenta um relógio no forms
  - **Chart:** Permite fácil apresentação de diferentes tipos de charts como bar-charts, pie-charts, line-charts, etc.
- Botões

  Estão disponíveis vários tipos de botões que podem ser pressionados pelo usuário com o mouse ou através de teclas de acesso rápido. Diferentes tipos de botões podem existir: botões que retornam a posição normal quando o usuário solta o mouse, botões que permanecem pressionados até novamente sejam pressionados (*checkbutton*) e botões que "deselecionam" todos os demais do mesmo grupo (*radiobuton*).

  Os botões podem apresentar variadas formas (retangulares, redondos, ovais), características (com labels do tipo bitmap ou pixmap, transparentes).
- Objetos para entrada de valores
  - **Slider:** São úteis para o usuário indicar um valor entre fronteiras fixadas. O usuário pode mudar esse valor movimentando o slider com o mouse.
  - **Dial:** São " discos " onde o usuário pode colocar em determinada posição indicando o valor
  - **Positioner:** Permite o usuário indicar uma posição (x,y) com o mouse
  - **Counter:** Permite o usuário apresentar um valor através de botões que fazem os incrementos no valor

- Objetos para entrada de dados

  São normalmente utilizados para obter entradas do tipo texto pelo usuário, por exemplo um nome de arquivo.

- Objetos para escolhas

  - **Menu:** Os menus são utilizados para o usuário escolher a partir de diferentes possibilidades. Quando o usuário pressiona o botão do mouse dentro do box do menu (ou move o mouse para o topo do box) um menu "pop-up' aparece. O usuário pode fazer então uma seleção a partir do menu
  - **Choice:** É um objeto que permite o usuário escolher entre várias opções.
  - **Browser:** É um box que contém várias linhas de texto. Se o texto não cabe inteiramente no box, um scrollbar é automaticamente adicionado.

- Outros Objetos

  - **Timer:** São utilizados para apresentar um relógio com contagem regressiva, o qual indica o final do tempo "piscando" ou retornando para a aplicação.
  - **XYPlot:** Apresenta um gráfico do tipo XY a partir de uma função ou conjunto de dados. Os pontos podem ser interativamente manipulados e recuperados
  - **Canvas:** São janelas X gerenciáveis. Em particular existe um objeto especializado. chamado **glcanvas** que são canvas especias para serem utilizados junto com OpenGL ou Mesa, permitindo a integração dessas bibliotecas.

5.2 Morph

Para implementar as operações de morfologia utilizamos uma biblioteca de rotinas de morfologia desenvolvida por Richard Alan Peters II da Vanderbielt University School of Engineering [7]. O pacote inclui uma série de operações morfológicas 2D e 3D para imagens binárias e com níveis de cinza.

As rotinas da biblioteca podem ser utilizadas de duas maneiras: através de um programa que opera com imagens no formato *rasterfile* ou chamando as subrotinas diretamente num programa em linguagem C, onde os valores a serem utilizados saõ armazenados em um vetor de elementos.

No protótipo de visualização as subrotinas de morfologia foram chamadas diretamente, sem a necessidade de se converter para um formato específico, evitando a geração de um arquivo temporário em disco, diminuindo o tempo gasto para as operações de morfologia.

5.3 OpenGL/Mesa

Como o interesse em áreas de computação gráfica vem crescendo, é desejado ser capaz de escrever aplicações que possam ser utilizadas em várias plataformas com uma variedade de capacidades gráficas. Um padrão gráfico facilita essa tarefa eliminando a necessidade de reescrever o código para cada plataforma na qual a aplicação será utilizada.

Para gráficos 3D vários padrões tem sido propostos, mas nenhum ainda tem ganho aceitação total. Um sistema relativamente bem conhecido é o PHIGS (*Programmer's Hierarchical Interactive Graphics System*), baseado em GKS (*Graphical Kernel System*). PHIGS realiza o encapsulamento da descrição e dos atributos dos objetos em uma lista (*display list*). A vantagem da lista é que um objeto complexo pode ser descrito apenas uma vez mesmo sendo apresentado várias vezes. Uma desvantagem é que requer um esforço considerável se o objeto for modificado (por exemplo através da interação do usuário). Outra desvantagem do PHIGS é sua carência por recursos avançados de *rendering* tais como mapeamento de textura.

Outro padrão desenvolvido é o PEX, projetado especialmente para X. Entre outras extensões, PEX adiciona automaticamete o modo rendering ao PHIGS, permitindo que objetos sejam apresentados ao mesmo tempo em que são descritos ao invés de terem que esperar completar a lista toda. Uma dificuldade com PEX é a falta de padronização por parte dos diversos fornecedores, tornando a portabilidade do programa um problema.

O sistema gráfico OpenGL [10] [14] é uma poderosa interface de software para hardware gráfico que permite produzir imagens coloridas de alta qualidade de objetos 2D e 3D. O OpenGL pode ser incorporado em qualquer sistema de janelas ou pode ser usado sem um sistema de janelas. Ele foi projetado para aproveitar a grande variedade de capacidades de hardware gráfico, de um *framebuffer* básico ao mais sofisticado subsistema gráfico.

O OpenGL foi desenvolvido pela *Silicon Graphics Inc. (SGI)*, que já havia desenvolvido um padrão semelhante chamado IRIS GL. Atualmente é controlado por um consórcio de indústrias conhecido como OpenGL Architectural Review Board (ARB) composto por Digital Equipament, IBM, Intel, Microsoft e SGI. A interface está licenciada para um grande número de softwares de computador e vendedores de hardware.

O OpenGL representa uma camada de abstração entre o hardware gráfico e os programas de aplicação. Para o programador é visível como um conjunto de cerca de 120 rotinas. OpenGL pode ser apresentado como uma máquina de estado que controla um bem definido conjunto de operações. As rotinas OpenGL representam um modo do programador manipular a máquina de estado para gerar a saída gráfica desejada. Na figura (5.1) apresentamos um diagrama representando essa máquina de estado:

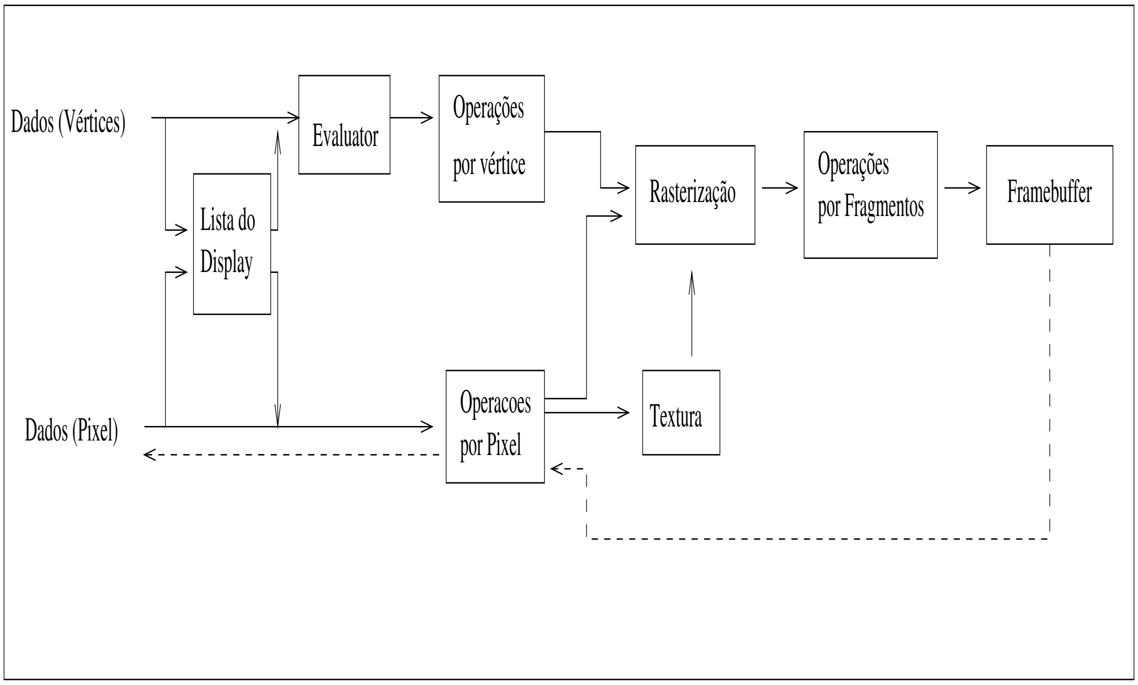


Figura 5.1: Diagrama da máquina de Estado do OpenGL

O modelo para interpretação dos comandos OpenGL é o modelo cliente-servidor, isto é, o programa (cliente) envia os comandos que são interpretados e processados pelo OpenGL (servidor). O Servidor pode ou não operar no mesmo computador que o cliente.

Os comandos de entrada do OpenGL estão representados no lado esquerdo. Os comandos podem ser acumulados em uma lista (*display list*) para processamento posterior ou podem ser enviados diretamente através de um processamento do tipo *pipeline*.

O primeiro estágio corresponde a um mecanismo eficiente para aproximação de curvas e superfícies geométricas avaliando funções polinomiais de valores de entrada. O próximo estágio opera em primitivas geométricas descritas por vértices: pontos, segmentos de retas e polígonos. Nesse estágio os vértices são transformados e as primitivas são " recortadas " (*clipped*) para o volume de visualização. A rasterização produz uma série de endereços do framebuffer e valores usando uma descrição 2D de pontos, segmentos de reta ou polígonos. Cada fragmento é enviado para o próximo estágio onde são realizadas operações individuais antes de serem enviados para o framebuffer (por exemplo operações lógicas nos fragmentos).

Mesa é uma biblioteca gráfica 3D com uma interface de programação (API Application Program Interface) semelhante a do OpenGL. Foi desenvolvida por Brian Paul (Centro de Ciências Espaciais e Engenharia da Universidade de Wisconsin - Madison) [12]. O principal objetivo dessa biblioteca tem sido a correteza. A velocidade não tem sido negligenciada, mas é secundária. Mesa pode ser instalada na maioria dos sistemas Unix que possuem instalados o ANSI C e X11. Também existem versões para Amiga, Microsoft Windows, Macintosh e sistemas NeXT.

5.4 Xwd,Xpr

Xwd é um utilitário do Sistema de Janelas X (*X Window System*) que permite armazenar janelas X em um formato especial (*formatted dump*). Esse formato pode ser lido por vários utilitários X.

Xpr também é um utilitário do Sistema de Janelas X que lê o formato dump e converte para um arquivo postscript que pode ser impresso.

Utilizamos esses dois utilitários para permitir as opções de impressão.

Capítulo 6

Resultados Obtidos

O protótipo de visualização e manipulação de mapas de Poincaré foi implementado e estamos disponibilizando o seu uso no IFGW. O objetivo de portabilidade foi atingido, necessitando apenas instalar as ferramentas citadas no capítulo 5 e alterar a localização das bibliotecas no *Makefile*, de acordo com o sistema instalado. Atualmente o código pode ser executado no IFGW em Solaris ou FreeBSD.

Mostraremos detalhadamente o funcionamento do protótipo através da análise de um conjunto de mapas de Poincaré. Em primeiro lugar, analisaremos um um mapa de Poincaré correspondendo à solução de um problema da mesa de bilhar. Esse mapa é composto por 10 curvas de 5000 pontos cada uma. Na figura 6.1 podemos verificar o mapa carregado a partir de seus arquivos originais. A seguir as curvas redundantes são eliminadas pelo usuário como mostra a figura (6.2) (curvas que estão contidas na região normal, conforme descrito no seção (4.1.2)). Podemos visualmente observar na figura (6.2) a presença de regiões normais e caóticas. Interativamente, as regiões normais podem ser marcadas, através do algoritmo de preenchimento conforme a figura (6.3). Os contornos das regiões normais marcadas são apresentado na figura (6.4).

Além das operações em um mapa de Poincaré, podemos realizar operações entre diferentes mapas de Poincaré. Uma das possibilidades de análise é realizar a interpolação entre os contornos dos mapas de Poincaré sucessivos. Apresentaremos a seguir dois exemplos de interpolação de mapas de Poincaré sucessivos.

Nas figuras (6.5.a), (6.5.b), (6.5.c) e (6.5.d) apresentamos quatro mapas de Poincaré relativos à solução do problema da mesa de bilhar com os seguintes valores de campo magnético: 1/9, 1/10, 1/11 and 1/12 . Nas figuras (6.5.e) e (6.5.f) apresentamos a interpolação obtida a partir das três regiões selecionadas.

Como segundo exemplo consideraremos um mapa de Poincaré gerado a partir do movimento da partícula em um plano x-y e sujeito a uma força conservativa $\mathbf{F} = -\nabla V$ onde a energia potencial é $V(x, y) = \lambda(x^4 + y^4) + x^2Y^2$. Os mapas de Poincaré são obtidos plotando $x(t)$ e $\dot{x}(t)$ toda vez que $y(t) = 0$ com $\dot{y}(t) \geq 0$.

Nas figuras (6.6.a), (6.6.b), (6.6.c) e (6.6.d) apresentamos quatro mapas de Poincaré correspondendo a $\lambda = 0.09$, 0.10, 0.11 and 0.12. Nas figuras (6.6.e) e (6.6.f) apresentamos a interpolação obtida a partir desses quatro mapas.

Para auxiliar na análise do objeto interpolado é permitido ao usuário realizar um corte paralelo ao plano xy. Na figura (6.7) apresentamos o objeto gerado a partir da interpolação dos contornos dos mapas de Poincaré da fig (6.5) e um plano de corte escolhido . Na figura (6.8) apresentamos o resultado do corte.

Um aspecto muito importante para o correto funcionamento do protótipo é a distinção das regiões normais e caóticas. No nosso caso essa distinção está diretamente relacionada com a escolha do tamanho correto da semente. Essa escolha é baseada na análise visual do próprio pesquisador. Numa tentativa de determinarmos o erro que essa análise visual poderia gerar para o cálculo da percentagem da região normal, realizamos o seguinte teste: escolhemos uma determinada região normal para os testes. Determinamos a menor semente que gera um resultado visualmente correto para a determinação da região escolhida. A partir desse valor de semente, aumentamos os valores da coordenadas x,y e verificamos qual a variação no número de pixels que compõem a região normal. Aumentamos o valor da semente até obtermos um valor visualmente errado para a região.

A seguir apresentaremos três tabelas com os resultados obtidos para 3 diferentes regiões do mapa de Poincaré marcadas na figura (6.9). Em cada tabela apresentamos os valores da semente (em coordenadas x e y), o número de pixels preenchidos como região normal, a diferença dos valores em relação ao primeiro valor obtido, a percentagem correspondente a diferença e a relação do número de pixels da região marcada e o número total de pixels que compõem a imagem (no nosso caso a imagem tem dimensão 512x512, resultando em 262144 pixels).

x	y	número de pixels	diferença	percentagem	número de pixels/total
8	5	35039	0	0	0.1336
9	5	34907	132	0.37	0.1332
9	6	34395	644	1.83	0.1312
10	6	34290	749	2.14	0.1300
10	7	33778	1261	3.60	0.1289
11	7	33683	1356	3.87	0.1285
11	8	33171	1868	5.33	0.1265
12	8	33090	1949	5.56	0.1262

Tabela 1: Número de pixels preenchidos para região 1.

x	y	número de pixels	diferença	percentagem	número de pixels/total
4	4	2432	0	0	0.0093
5	4	2358	74	3.14	0.0090
5	5	2257	175	7.75	0.0086
6	5	2195	237	16.0	0.0084
6	6	2095	337	16.0	0.0080
7	6	2043	389	19.0	0.0078

Tabela 2: Número de pixels preenchidos para região 2.

x	y	número de pixels	diferença	percentagem	número de pixels/total
4	4	2304	0	0	0.0088
5	4	2151	153	7.11	0.0082
5	5	2091	213	10.1	0.0080
6	5	1956	348	17.7	0.0075
6	6	1913	391	20.0	0.0073

Tabela 3: Número de pixels preenchidos para região 3.

Podemos observar que as variações obtidas na tabela 1 são pequenas (5.56 %) e que essas variações são maiores para os casos da tabela 2 e 3. Esse aumento deve-se ao fato das regiões 2 e 3 serem regiões pequenas. Porém, quando se leva em conta a imagem como um todo, podemos observar através da coluna (número de pixels/total) que as variações na área total são pequenas com os diferentes tamanhos de semente escolhidos.

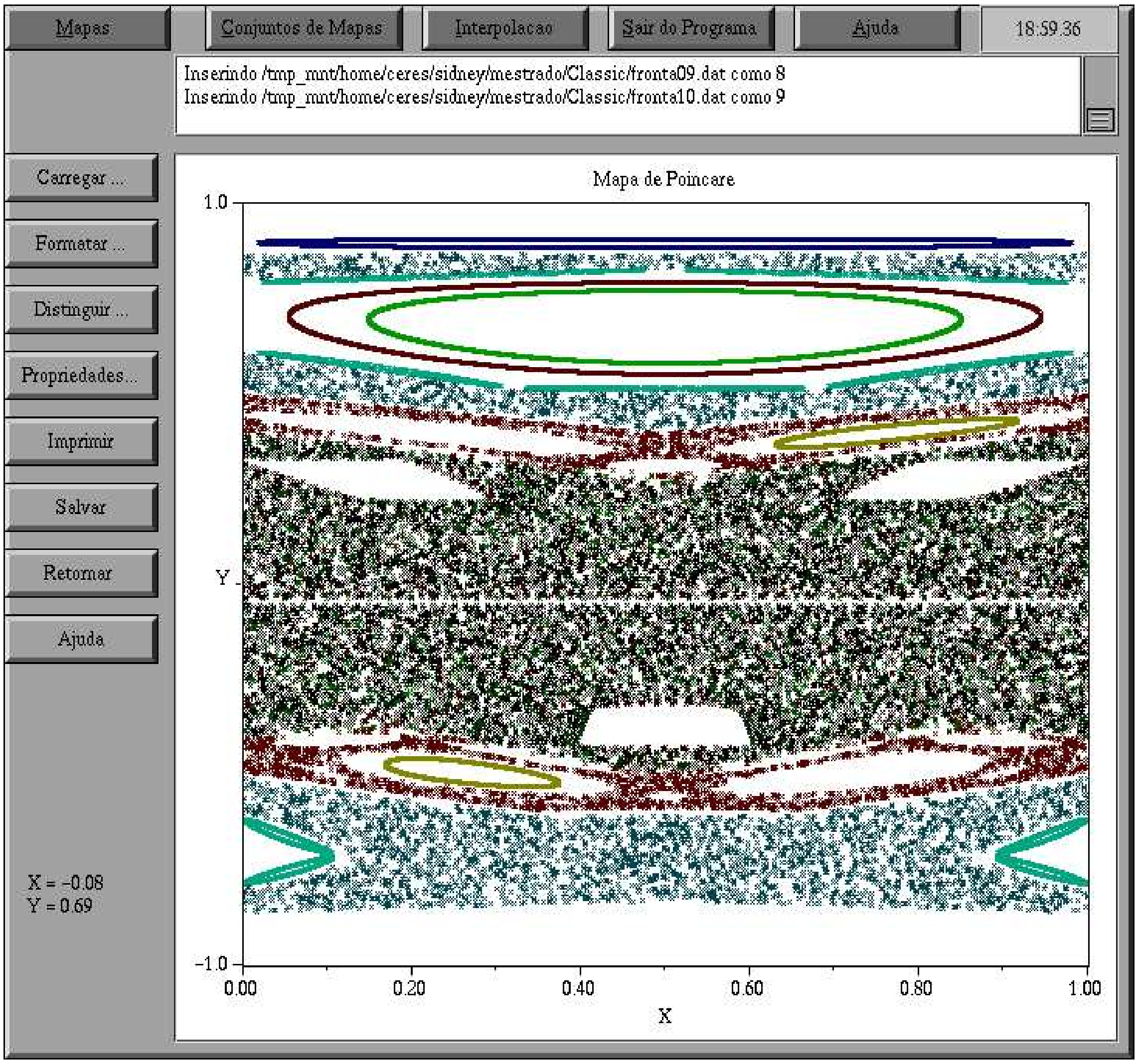


Figura 6.1: Mapa de Poincaré composto por 10 curvas de 5000 pontos cada

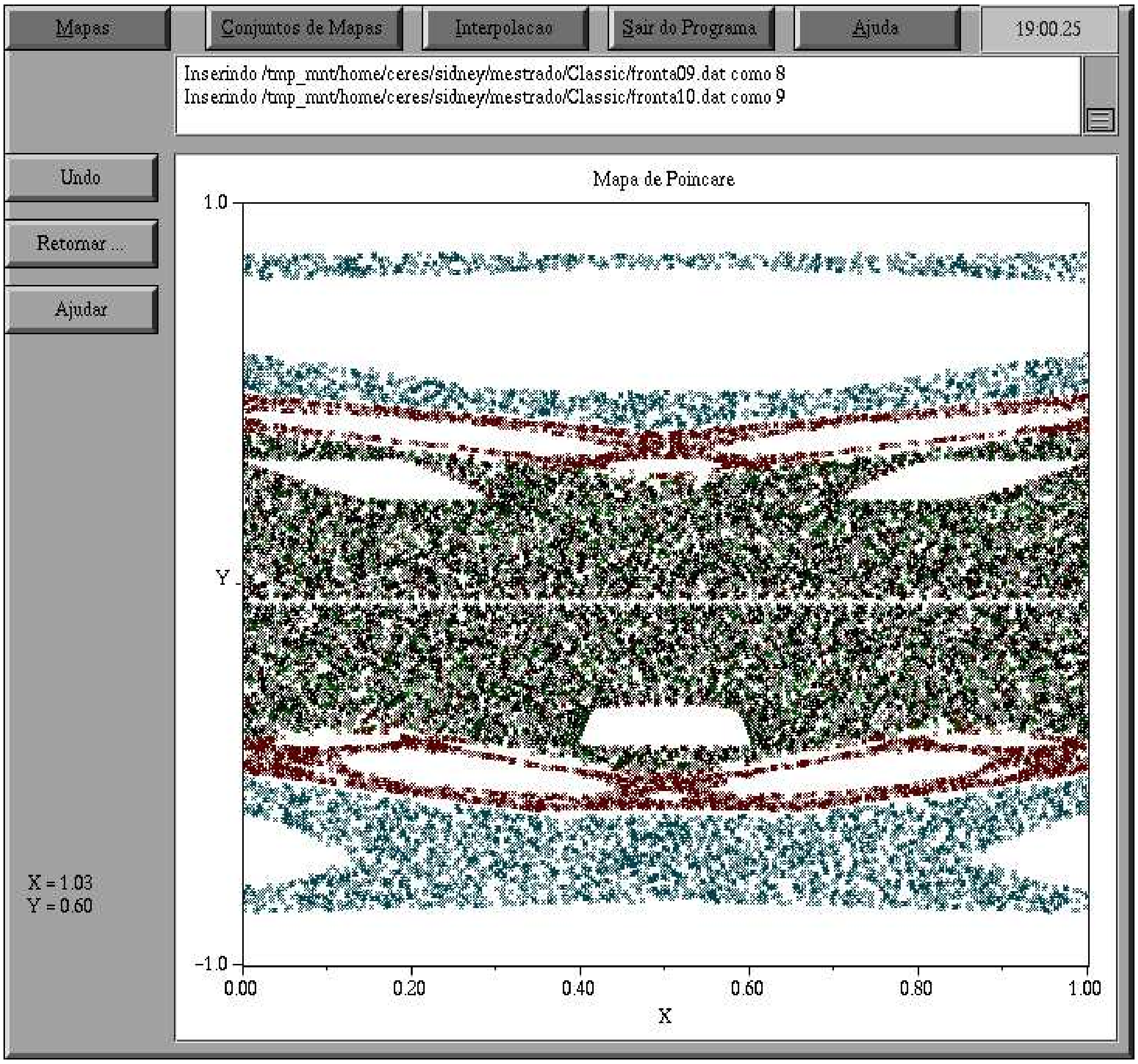


Figura 6.2: Mapa de Poincaré com as curvas redundantes eliminadas

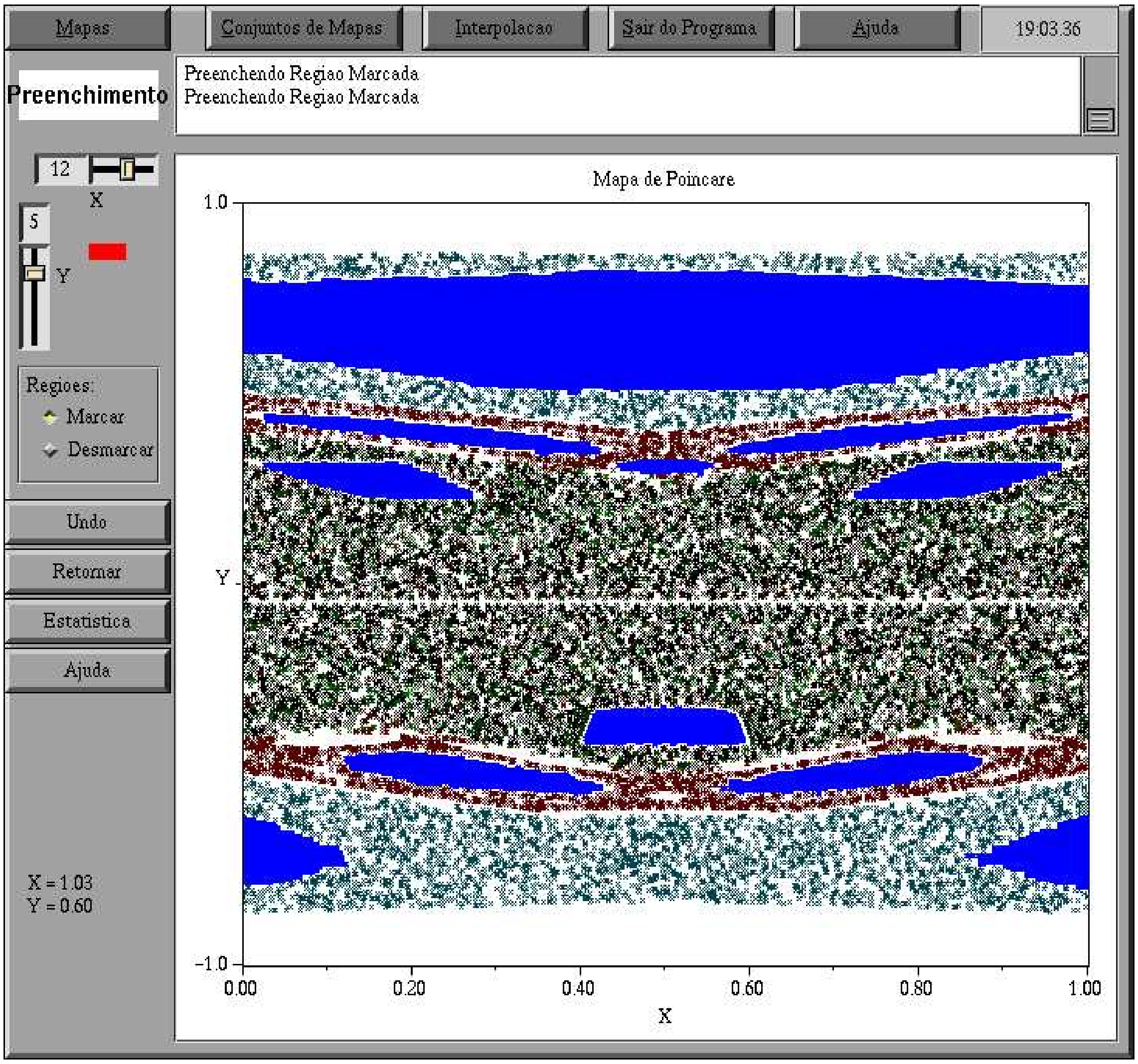


Figura 6.3: Regiões normais do mapa de Poincaré marcadas

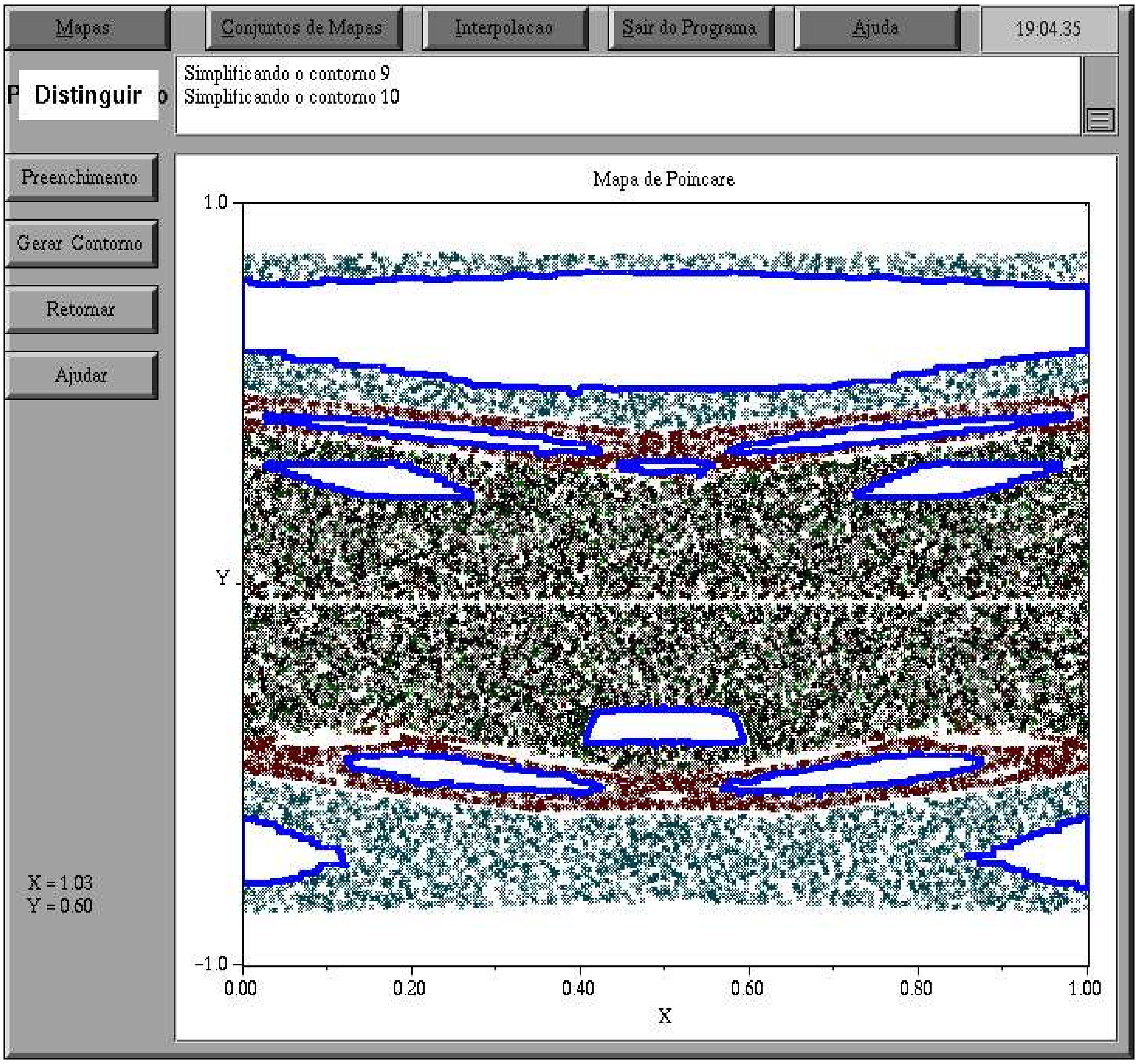
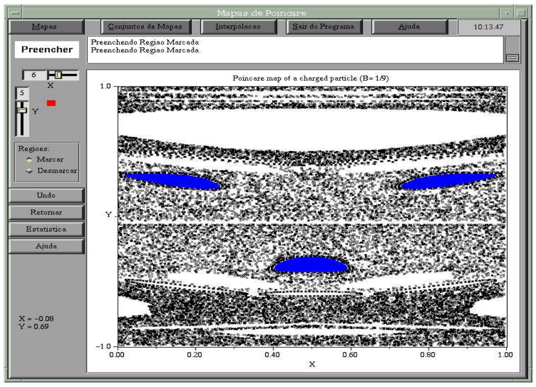
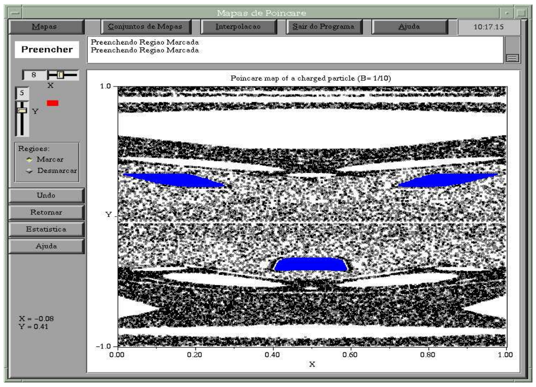


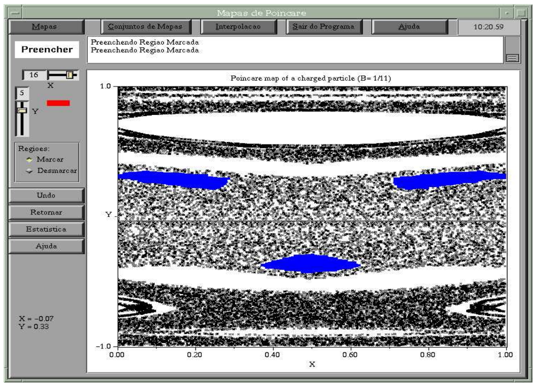
Figura 6.4: Contorno das regiões normais do Mapa de Poincaré



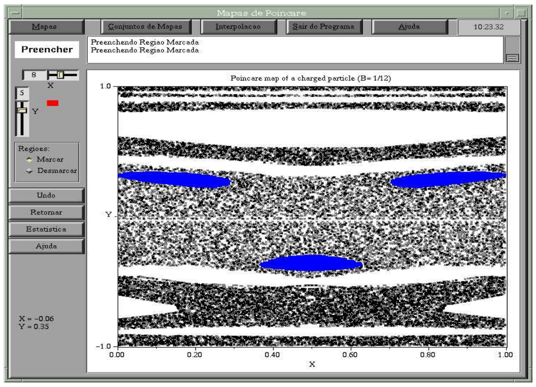
(a)



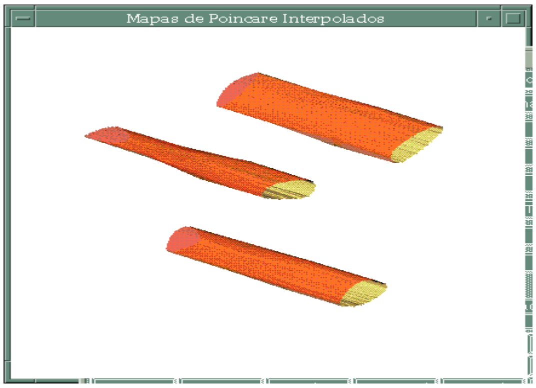
(b)



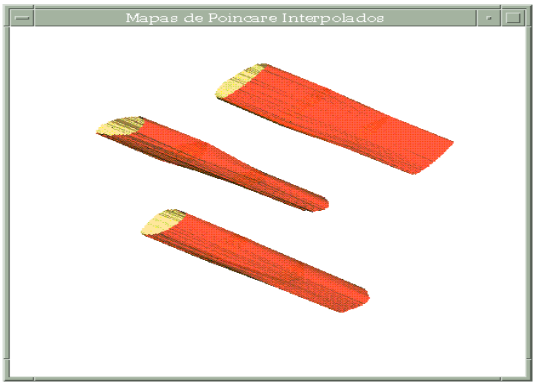
(c)



(d)



(e)



(f)

Figura 6.5: Mapas de Poincaré para uma partícula carregada

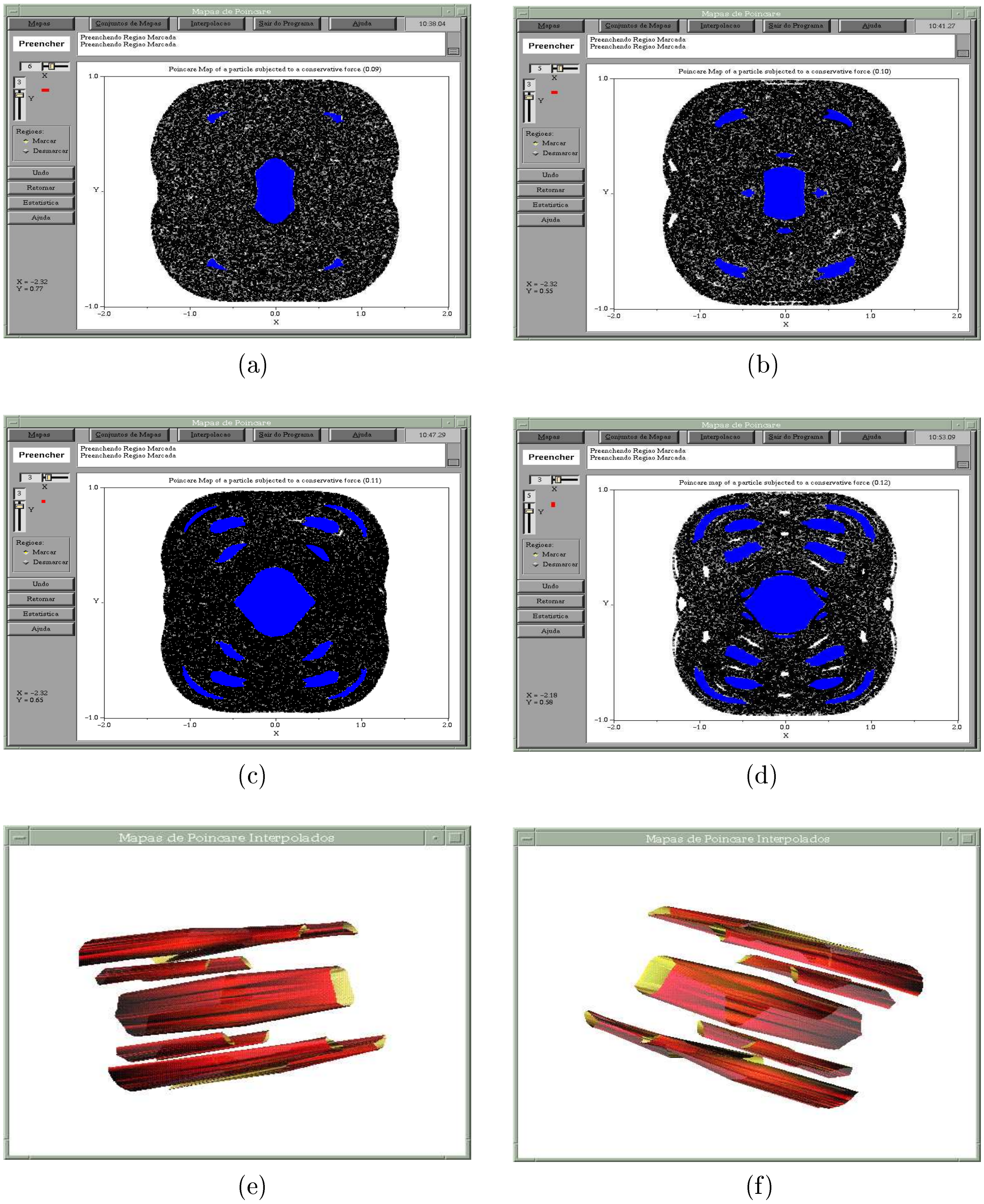


Figura 6.6: Mapas de Poincaré de uma partícula sujeita a uma força conservativa

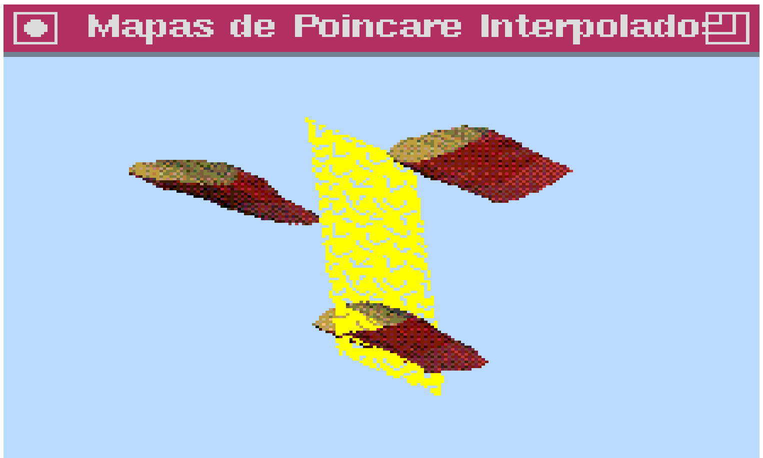


Figura 6.7: Objeto gerado a partir da interpolação dos contornos e o plano de corte

Figura 6.8: Corte no objeto gerado a partir da interpolação dos contornos

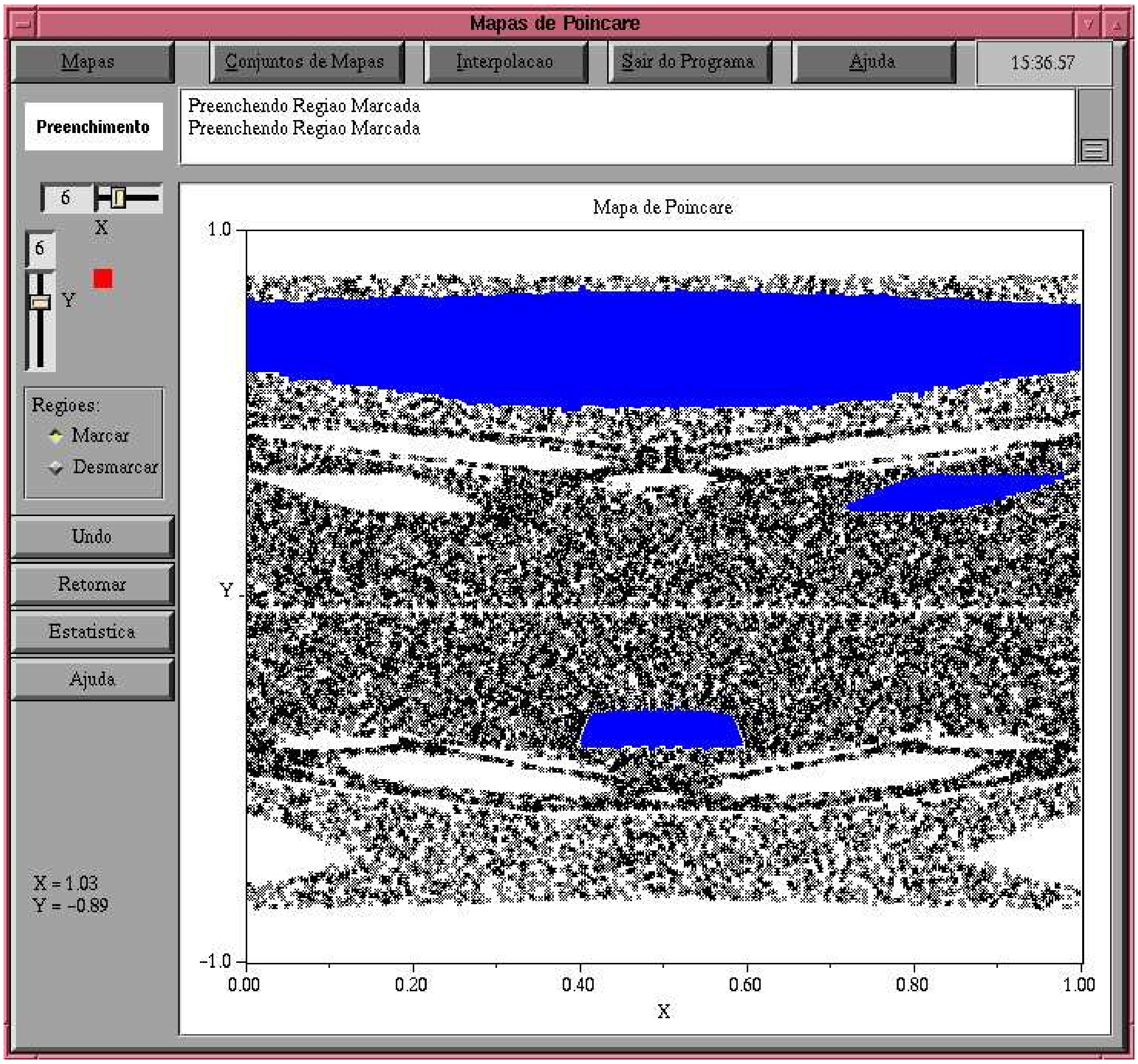


Figura 6.9: 3 Regiões normais marcadas para a análise

Capítulo 7

Conclusão e Trabalhos Futuros

Nesse trabalho desenvolvemos um protótipo para a Visualização e Manipulação de mapas de Poincaré. Com esse trabalho pretendemos apresentar uma nova opção para análise de dados, iniciando uma nova área de interesse para os pesquisadores do IFGW.

Pretendemos com esse trabalho disponibilizar uma ferramenta simples e funcional, utilizando recursos simples de computação gráfica e processamento de imagens.

O desenvolvimento do protótipo envolveu várias áreas básicas de computação tais como, desenvolvimento de interfaces gráficas, operações de filtragem de imagens, morfologia matemática, reconstrução 3D e visualização de imagens 3D.

O programa foi desenvolvido com uma série de ferramentas de domínio público que podem ser utilizadas no desenvolvimento de outros programas. O conhecimento adquirido no uso de tais ferramentas é importante, visto que muitas das ferramentas são recentes e ainda estão em desenvolvimento (tais como *Xforms* e *Mesa*)

O texto apresentado pretende servir como ponto inicial para a continuação do desenvolvimento do protótipo. Por esse motivo, tentamos, mesmo de maneira abreviada, apresentar os principais conceitos sobre Mapas de Poincaré, Visualização Científica, as técnicas e ferramentas utilizadas no desenvolvimento do protótipo.

Sob o ponto de vista computacional, foi apresentado o desafio de trabalhar com um conjunto de dados incompletos gerados a partir de simulações. Conseguimos através de técnicas simples obter uma solução elegante e interativa com o usuário.

Outro aspecto que vale destacar é a importância da interação entre as diversas áreas de conhecimento, trabalhando-se com equipes multidisciplanares. No nosso caso, verificamos a importância da participação de um especialista na análise dos mapas de Poincaré, eliminando dúvidas a respeito dos dados a serem analisados e propondo mecanismos de análise.

Esse trabalho não pretende esgotar o assunto de análise de mapas de Poincaré. A nossa expectativa é que esse protótipo evolua com a substituição gradual de seus módulos.

Um ponto que ainda não pode ser desenvolvido foi uma análise detalhada da interface do programa com o usuário. No futuro seria importante realizar " testes " com os usuários para verificar qual o grau de usabilidade da interface.

Alguns pontos que podem ser melhorados:

- Uso de estruturas mais sofisticadas para armazenamento de dados. Até o presente momento não sentimos a necessidade de uma estrutura de dados mais inteligente, mas com o aumento do volume de dados seria importante o uso mais eficiente dos recursos de memória.
- No módulo de formatar o mapa, utilizar rotinas da biblioteca Mesa para apresentação dos dados bidimensionais relativos a um único mapa. Na implementação atual os dados são apresentados utilizando-se rotinas implementadas pelo toolkit *Xforms (xy-plot)*
- Aperfeiçoar o módulo de interpolação, refinando o código do programa de reconstrução 3D ou utilizar técnicas de morfologia matemática para a interpolação entre os mapas.
- O módulo de Filtragem poderia explorar um pouco as técnicas de reconhecimento de padrões para um processamento automático (ou semi-automático) de regiões
- modificações na interface a fim de aumentar o grau de usabilidade

Apêndice A

Documentação da Ferramenta

Apresentaremos nesse apêndice a documentação geral do protótipo. Detalhes específicos do código podem ser obtidos nos comentários realizados no próprio código fonte.

O código da implementação do protótipo está dividido em vários arquivos apresentados no diretório base e em 4 subdiretórios. Cada diretório possue um arquivo *makefile* que permite a compilação do código correspondente.

O diretório base possue um conjunto de funções básicas do protótipo, tais como:

- Leitura dos arquivos que compõem o mapa de Poincaré (*leitura.c*).
- Crescimento por semente (*seed01.c*).
- Operações de morfologia (*morfologia.c*).
- Simplificação dos contornos (*aproxima_retas.c*).
- Conjunto de operações do tipo salvar e imprimir mapas (*principal.c*).

Outro arquivo a se destacar é o *estru.h* que possue definições das estruturas de dados globais utilizadas no protótipo. No diretório principal existem arquivos de comandos (*shell-scripts*) utilizando comandos *c-shell* que auxiliam operações de manipulação dos arquivos de dados:

- *converte_fortran*: realiza um pré-processamento nos arquivos de entrada de dados, convertendo o símbolo D em E (notação de ponto flutuante em Fortran para C).

- *renumera_contorno*: utilizado para associar corretamente a ordem de um conjunto de contornos que representa um mapa de Poincaré
- *gerar_pedrini*: utilizado para converter o conjunto de contornos que representa um mapa no formato utilizado pelo módulo de reconstrução.

No diretório principal ainda existem os subdiretórios *interface, reconstrução, mesa* e *help* que serão apresentados nas seções seguintes.

A.1 Interface

Nesse diretório estão os arquivos que geram a interface gráfica do protótipo. A interface foi deesenvolvida com o auxílio da ferramenta gráfica *fdesign*, exceto o *form* correspondente à apresentação da matriz de mapeamento para interpolação dos mapas (figura ()) que foi escrita diretamente em linguagem C com rotinas da biblioteca *xforms*. O arquivo correspondente a esse código está apresentado no arquivo *matriz.c*.

O arquivo que compõem a descrição da interface utilizado pelo programa *fdesign* é o arquivo *poincare.fd* . A interface gráfica pode ser dividida nos seguintes *forms*:

- principal: janela principal do protótipo.
- seta_labels: janela para ajuste dos parâmetros dos gráficos bi-dimensionais.
- dir: janela utilizada para selecionar os arquivos que compõem um determinado mapa de Poincaré.
- ajuda: janela para apresentar as mensagens de ajuda ao usuário.
- propriedades: janela para apresentar as propriedades de determinado mapa de Poincaré.
- impressao: janela que permite ao usuário escolher o dispositivo de impressão (nome da impressora ou nome do arquivo).

O *forms principal* está dividido em vários grupos, que podem ser associados aos módulos descritos na figura (4.1):

- grupo_principal: conjunto de botões dos módulos principais do programa.
- grupo_carregar: corresponde a interface para *Carregar Arquivos*.
- grupo_formatar: corresponde a interface para *Formatar o Mapa*.
- grupo_eliminar_curvas: corresponde a interface para eliminar curvas.
- grupo_distinguir: corresponde a interface para *Distinguir Regiões*.
- grupo_mapas: conjunto de botões dos módulo para processamento e análise de um mapa de Poincaré.
- grupo_cor_curvas: corresponde a interface para associar cores ao conjunto de pontos que compõem um mapa.
- grupo_pedrini: corresponde a interface para *Interpolação 3D*
- grupo_preenchimento: corresponde a interface para realizar o crescimento por semente.
- grupo_mesa: corresponde a interface para *Visualização 3D*
- grupo_conjunto_mapas: corresponde a interface para *Operações entre Mapas*

Para correto funcionamento com os demais módulos do programa, o *fdesign* deve ser utilizado com as opções *Track Geometry, Emit UI code* e *Alt Format*. Com essas opções, o arquivo poincare.c e poincare.h serão gerados automaticamente após escolhido a opção *save* (*File, Save*) do *fdesign*.

As funções de *callback* estão apresentadas nos demais arquivos do diretório: *apresenta.c, distinguir.c, carregar.c, formatar.c, poincare_cb.c, conjunto.c, iluminacao.c, poincare_main.c, mapas.c, reconstruir.c, cor_curvas.c, matriz.c, diretorio.c* e *mesa.c*.

A.2 Reconstrução

As rotinas de interpolação entre os mapas estão baseadas no código das rotinas de reconstrução 3D desenvolvidas por Pedrini. A estrutura geral do programa foi mantida, apenas convertendo os arquivos em linguagem Pascal para C (*medidas.pas, triangula.pas e separa_cont.pas*) e resolvendo alguns problemas na implementação original citados na apêndice C. Incluímos também um arquivo *makefile* facilitando o processo de compilação. Os detalhes sobre a implementação desse módulo pode ser verificado em [13].

A.3 Mesa

Nesse diretório estão as rotinas para visualização 3D do objeto gerado a partir da interpolação. Nesse diretório temos dois arquivos fonte:

- normais.c: calcula as normais dos vértices do objeto interpolado.
- visual3d.c: utiliza as rotinas de OpenGL/Mesa para a visualização.

A comunicação entre o módulo de interpolação e visualização é realizada através de um conjunto de arquivos (*/tmp/mp/FACES*.DAT*), resultantes do processo de interpolação.

A.4 Help

Esse diretório contém os arquivos com as mensagens de ajuda ao usuário. Esses arquivos são nomeados *helpnumero*, onde "numero " corresponde ao número que será passado como argumento para a chamada da função help.

Apêndice B

Telas do Protótipo

Nesse apêndice apresentaremos as várias janelas que compõem o protótipo:

- Principal (Figura B.1);
- Operações com mapas de Poincaré (Figura B.2);
- Operações de carregar com mapas de Poincaré (Figura B.3);
- Operações para formatação dos mapas de Poincaré (Figura B.4);
- Opções de *label* para os mapas de Poincaré (Figura B.5);
- Opções de *cores* para os mapas de Poincaré (Figura B.6);
- Operação de eliminar curvas dos mapas de Poincaré (Figura B.7);
- Operação de distinguir regiões dos mapas de Poincaré (Figura B.8);
- Operação de preenchimento de regiões dos mapas de Poincaré (Figura B.9);
- Operações entre conjuntos de mapas de Poincaré (Figura B.10);
- Operação de interpolação de mapas de Poincaré (Figura B.11);
- Operação de visualização dos mapas interpolados (Figura B.12);

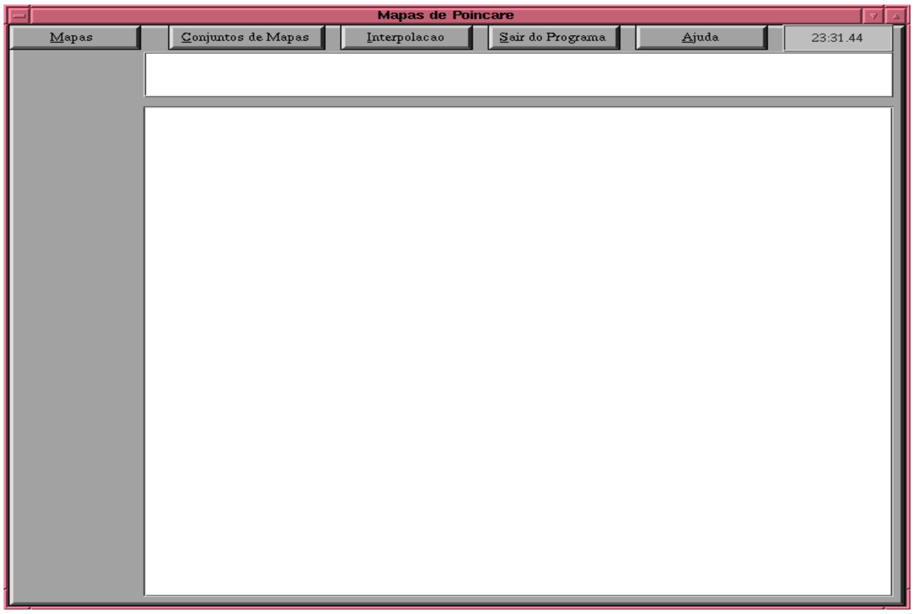


Figura B.1: Janela principal do protótipo

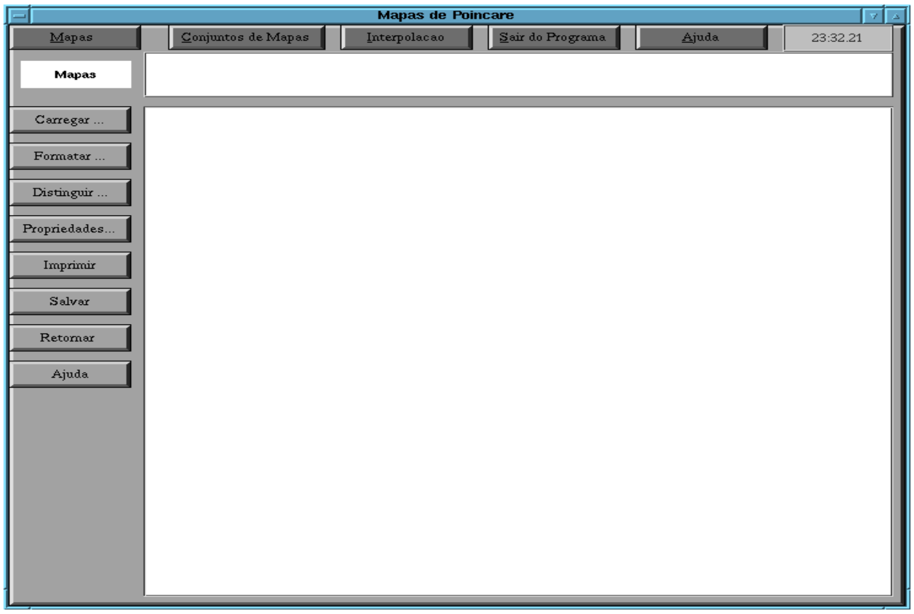


Figura B.2: Janela apresentando as operações com mapas de Poincaré

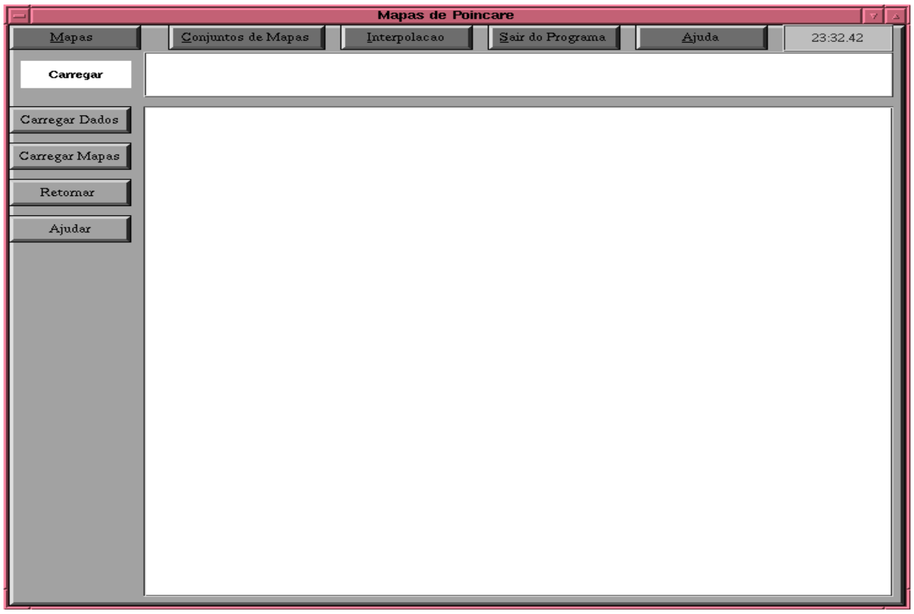


Figura B.3: Janela apresentando a operações de carregar com mapas de Poincaré

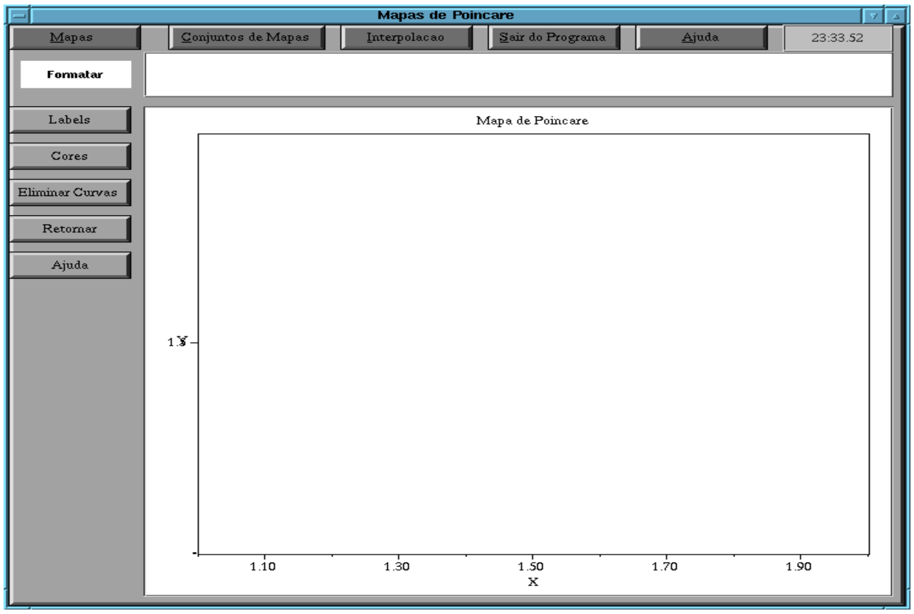


Figura B.4: Janela apresentando as operações para formatação dos mapas de Poincaré

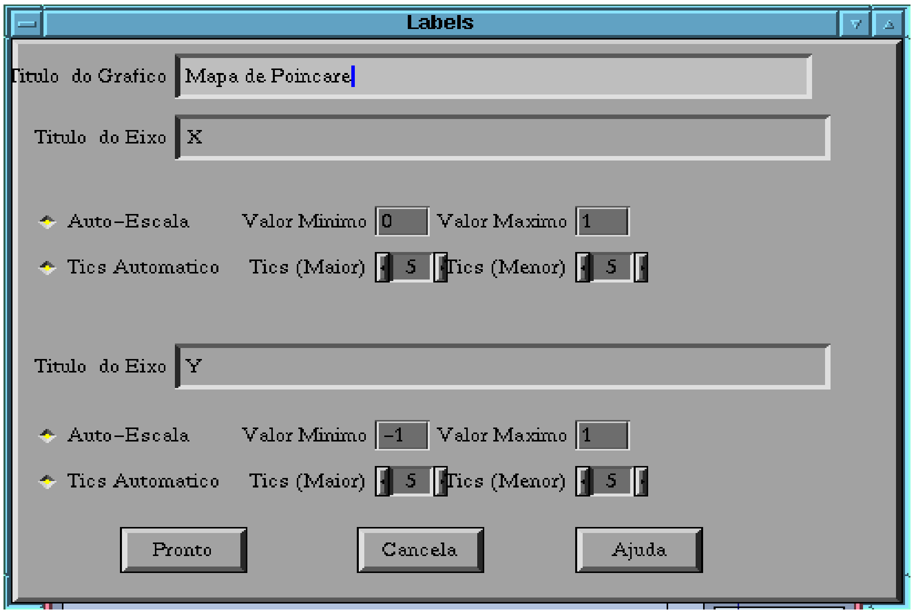


Figura B.5: Janela apresentando as opções de *label* para os mapas de Poincaré

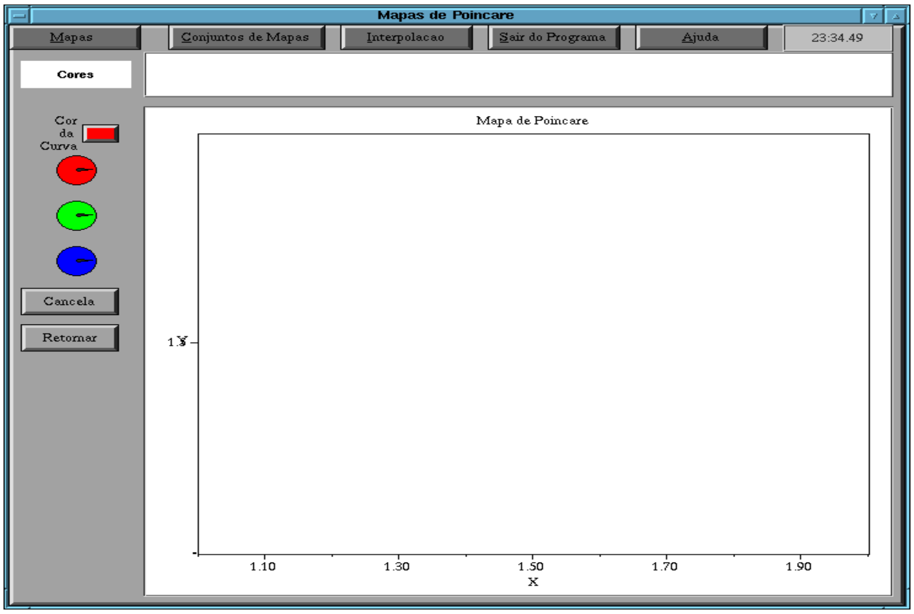


Figura B.6: Janela apresentando as opções de *cores* para os mapas de Poincaré

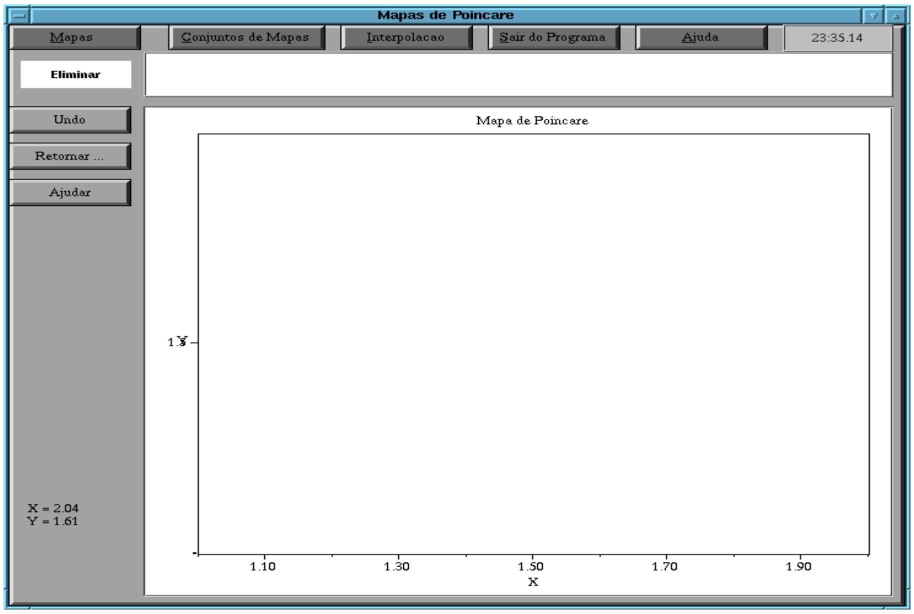


Figura B.7: Janela apresentando a operação de eliminar curvas dos mapas de Poincaré

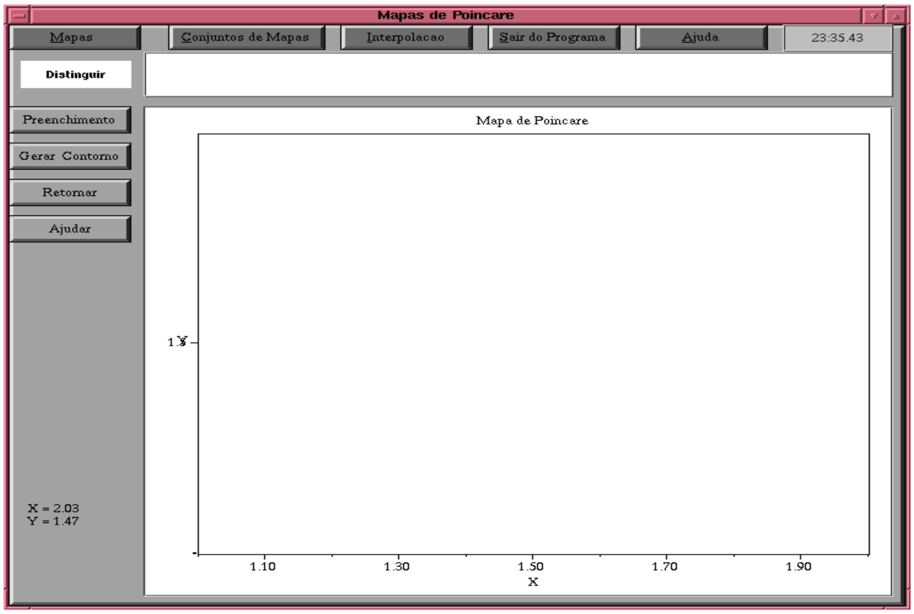


Figura B.8: Janela apresentando a operação de distinguir regiões dos mapas de Poincaré

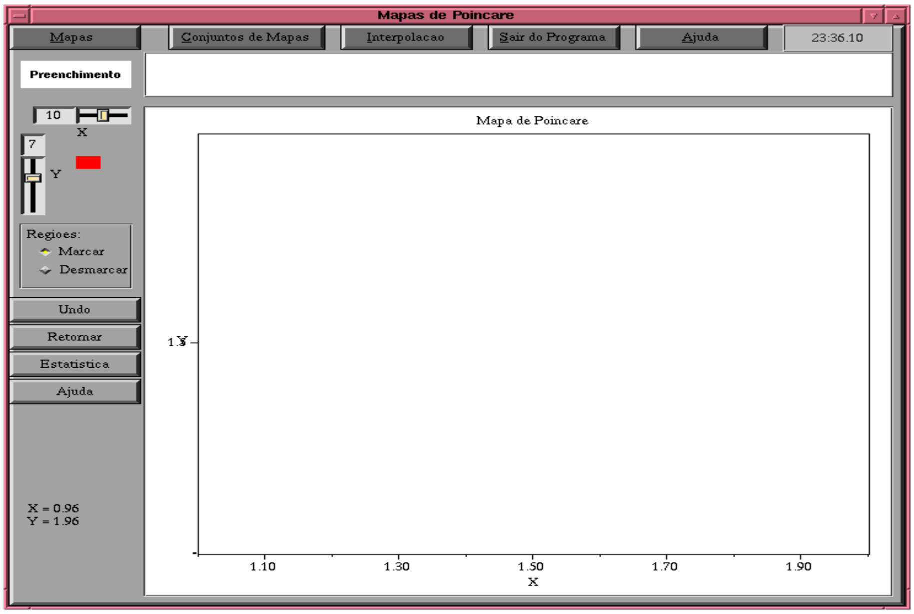


Figura B.9: Janela apresentando a operação de preenchimento de regiões dos mapas de Poincaré

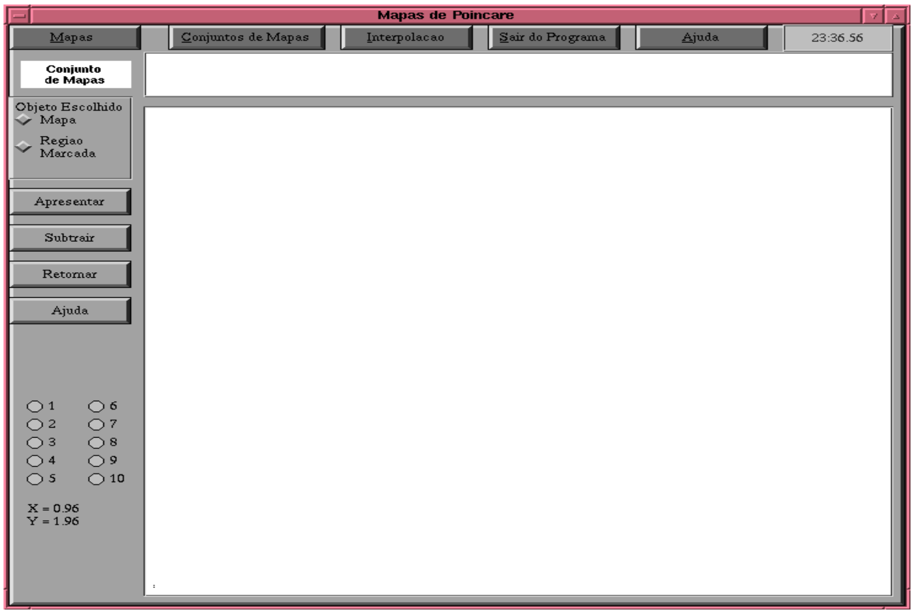


Figura B.10: Janela apresentando as operações entre conjuntos de mapas de Poincaré

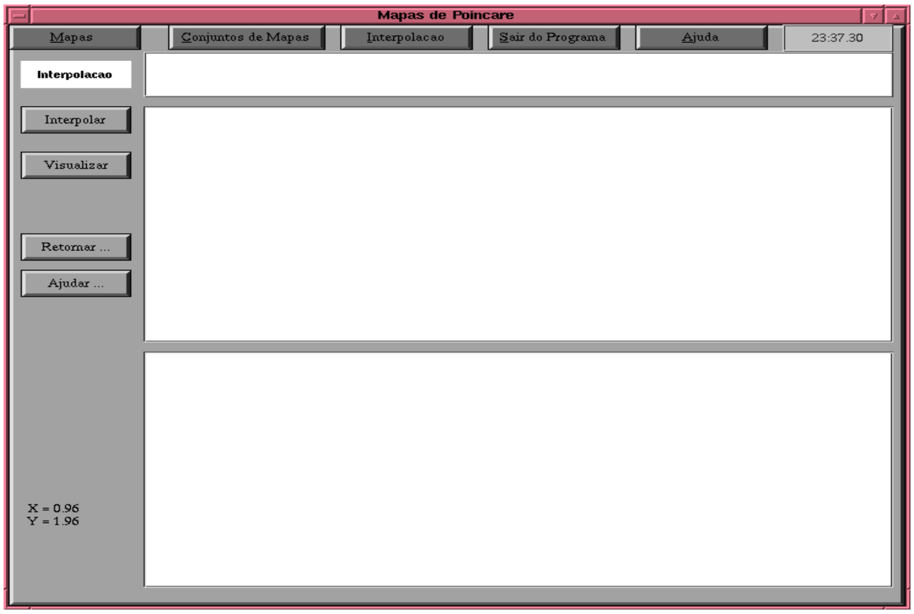


Figura B.11: Janela apresentando a operação de interpolação de mapas de Poincaré

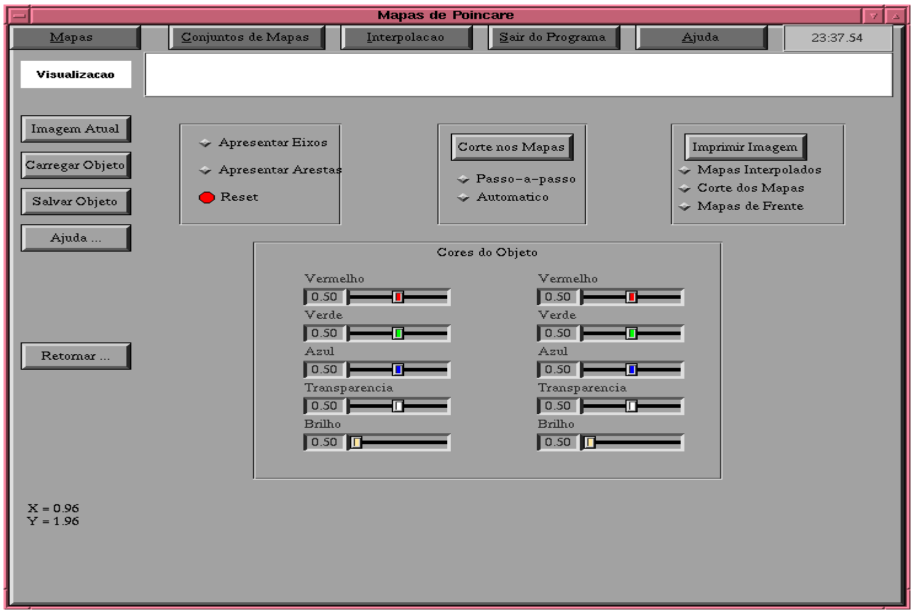


Figura B.12: Janela apresentando a operação de visualização dos mapas interpolados

Apêndice C

Correções na Implementação do Algoritmo de Reconstrução

Ao realizarmos a integração do módulo de reconstrução 3D ao protótipo observamos algumas etapas do algoritmo de reconstrução que não foram corretamente implementados. Apresentaremos nesse apêndice os pontos observados e a solução realizada em cada um deles. Como realizamos a conversão do código em Pascal para C, iremos apresentar todas as soluções em C.

C.1 Alteração em *renumerar_contornos*

Na função *renumerar_contornos* do arquivo *triangula.c* a atualização da variável responsável pelo conjunto de vértices não era realizada corretamente. A lista de vértices apontada por *t1* estava corretamente renumerada, mas o conjunto C não havia sido atualizado. A solução foi incluir em um looping, onde os vértices eram percorridos, a instrução para atualização do conjunto C, bem como a atualização das variáveis M, $n2$ e $prim2$.

Apresentamos a seguir o código da implementação original:

```
  t1 = prim1;
  do {
    (*n1)++;
/* printf("%3.0f  %3.0f\n", t1->x, t1->y); */
```

```
    t1 = t1->dir;
  } while (t1 != prim1);
```

O código com a solução ficou da seguinte forma:

```
  t1 = prim1;
  P_expset(C,0L);
  do {
    (*n1)++;
    t1->v=*n1;
    P_addset(C,(int)t1->v);
 /* printf("%3.0f  %3.0f\n", t1->x, t1->y); */
    t1 = t1->dir;
  } while (t1 != prim1);
  M = *n1;
  *n2 = *n1;
  prim2 = prim1;
```

C.2 Chamada da função transformação

Na função *caso1_1n* do arquivo *triangula.c* a chamada para a função transformação é realizada originalmente como

```
transformacao(C, E, prim2, prim, 1L, n2, num)
```

A solução implementada faz a troca por

```
transformacao(C, E, prim2, prim, 1L, 1L, num)
```

O mesmo se aplica para na função *caso2_1n* do arquivo *triangula.c* que originalmente é chamada como

```
transformacao(C, E, prim2, prim, 1L, n2, num)
```

e na solução implementada utilizamos

```
transformacao(C, E, prim2, prim, 1L, 1L, num);
```

C.3 Variável correspondente ao contorno associado ao *slice* desejado

Na *caso2_1n* do arquivo *triangula.c* a variável correta para realizar o mapeamento em *slice* corresponde a variável j e não i. A correção é realizada trocando-se a linha

```
str_(st, i);
```

pela linha

```
str_(st, j);
```

C.4 Problemas com análise da matriz de mapeamento

Um outro problema verificado foi a correta execução de mapeamento de acordo com a matriz de mapeamento. Nas funções de análise da matriz foram utilizadas variáveis para indicar quantos elementos estavam na mesma linha ou coluna da matriz. A partir dessas variáveis é invocado o programa apropriado para triangulação.

O problema é que essas variáveis foram erroneamente utilizadas para representar também quais contornos deveriam ser triangulados. Essa simplificação funciona para o caso dos contornos que participam do processo de triangulação possuírem números imediatamente sucessivos. Por exemplo, vamos supor que a matriz de mapeamento seja do tipo:

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	1	1	1	0
0	0	0	0	0

Essa matriz indica que os contornos 2,3 e 4 do primeiro slice devem ser triangulados com o contorno 4 do segundo slice. A chamada da função para triangulação original enviava o valor do contorno inicial para a triangulação do primeiro slice e o valor 3 representando que são 3 contornos para a triangulação (correspondendo aos valores 2, 3 e 4). Nesse caso o programa funcionará corretamente.

Supondo uma nova matriz do tipo:

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	1	0	1	0
0	0	0	0	0

O valor enviado seria o valor inicial 2 e o número de contornos nesse slice, que corresponde a 2. No algoritmo original a triangulação seria realizada entre os contornos 2 e 3 do primeiro slice, enquanto na realidade a triangulação deveria ser com os contornos 2 e 4 do primeiro slice.

Para solucionar esse problema, foi alterado o formato do arquivo de dados que faz a comunicação (*MAPA_**) entre o programa de análise da matriz e a triangulação propriamente dita. Incluímos nesse arquivo os números dos contornos que devem ser triangulados, utilizando o número de contornos como contador para o número de elementos a serem lidos no arquivo.

Essa modificação foi realizada em todos os módulos que utilizam a análise da matriz de mapeamento:

- Alterações nas funções *gera_mapa* e *analisar_matriz* do arquivo *reconstr3d.c*, gravando o número do contorno no arquivo de comunicação.
- Função *caso1_1n* em *triangula.c*. Alteradas as linhas:

```
for (l = j; l < acum1 + j; l++){

} /* do looping do for */
```

por

```
for (contsid =0; contsid < acum1; contsid++){

if (contsid < acum1 -1) fscanf(arq1,"%d",&l);
} /* do looping do for */
```

- Função *caso2_1n* em *triangula.c*. Alteradas as linhas:

```
for (l = i; l < acum2 + i; i++){

} /* do looping do for */
```

por

```
for (contsid =0; contsid < acum2; contsid++){

if (contsid < acum2 -1) fscanf(arq1,"%d",&l);
} /* do looping do for */
```

Referências Bibliográficas

[1] Visualization in scientific computing — a synopsis. *IEEE Computer Graphics and Applications*, 7(7):61–70, July 1987.

[2] James D. Foley et al. *Computer Graphics: Principles and Practice.* Addison Wesley Publishing Company, Reading, Mass., 1996.

[3] R.A. Earnshaw and D. Watson. A classification scheme for scientific visualization. *Animation and Scientific Visualization - Tools & Applications*, pages 125–139, 1993.

[4] B. Eckhardt and D.M.Yao. *Physica D*, (65):100, 1993.

[5] F. C. Peyrin e C. L. Odet Ekoule, A.B. A triangulation algorithm from arbitrary shaped multiple planar contours. *ACM Transactions on Graphics*, 10(2):182–199, Abril 1991.

[6] Rafael C. Gonzalez and Paul Wintz. *Digital Image Processing.* Addison Wesley Publishing Company, Reading, Mass., 1992.

[7] Richard Alan Peters II. ftp://image.vuse.vanderbilt.edu/pub/morph.tar.z.

[8] Amil K. Jain. *Fundamentals of Digital Image Processing.* Prentice-Hall, Inc, Englewood Cliffs, 1989.

[9] Thomas A. DeFanti McCormiek, Bruce H. and Maxime D. Brown. Visualization in scientific computing (visc):definition, domain and recommendations. *Computer Graphics*, 21(6), November 1987.

[10] Tom Davis Neider, Jackie and Mason Woo. *OpenGL Programming Guide - The Official Guide to Learning OpenGL, Release 1.* New York, Addison-Wesley Publishing Company, 1993.

[11] Wayne Niblack. *An Introduction to Digital Image Processing.* Prentice-Hall International, 1986.

[12] Brian Paul. Mesa library. *http://www.ssec.wisc.edu/ brianp/Mesa.html.*

[13] H. Pedrini. Recontrução a partir de seções transversais de objetos. *Tese de Mestrado, FEE/UNICAMP*, 1994.

[14] David Rogelberg. *OpenGL Reference Manual - The Official Reference Document for OpenGL, Release 1.* New York, Addison-Wesley Publishing Company, 1993.

[15] David F. Rogers. *Procedural Elements for Computers Graphics.* New York, McGraw-Hill Book Company, New York, 1985.

[16] S.L. e J. N. Mosier Smith. Guidelines for designing user interface software. *ESD-TR-86-278/MTR 10090, BedFord,MITRE Corporation*, 1986.

[17] T.C.Zhao and Mark Overmars. Forms library for x. *http://www.cpt.etu.ru/ gerd/xforms/zhao/.*

[18] Craig Upson, Thomas A. Faulhaber, Jr., David Kamins, David Laidlaw, David Schlegel, Jeffrey Vroom, Robert Gurwitz, and Andries van Dam. The Application Visualization System: a computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):30–42, July 1989.

[19] Jonas Gomes e Luiz Velho. *Computação Gráfica: Imagem.* IMAP/SBM/Série Computação e Matemática.

[20] J. Weber. Visualization: Seeing is believing. *Byte*, pages 121–128, April 1993.