MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-14831-TDI/1271

BSSDATA – PROTÓTIPO DE UM SISTEMA OTIMIZADO PARA TRATAMENTO E ANÁLISE DE DADOS DO BRAZILIAN SOLAR SPECTROSCOPE

André Ricardo Fazanaro Martinon

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Drs. Stephan Stephany e Hanumant Shankar Sawant, aprovada em 30 de julho de 2003.

INPE
São José dos Campos
2008

523.76:551.46

Martinon, A. R. F.
BSSDATA – Protótipo de um sistema otimizado para tratamento e análise de dados do Brazilian Solar Spectroscope. - São José dos Campos: INPE, 2003.
116 p. ; (INPE-14831-TDI/1271)

1. Espectrografia solar. 2. Processamento de dados. 3. Análise de dados. 4. Biblioteca dinâmica. 5. Vetorização. I. Título.

Aprovado pela Banca Examinadora em cumprimento a requisito exigido para a obtenção do Título de **Mestre** em **Computação Aplicada.**

Dr. Reinaldo Roberto Rosa
Presidente/INPE, SJCampos-SP

Dr. Stephan Stephany
Orientador/INPE, SJCampos-SP

Dr. Hanumant Shankar Sawant
Orientador/INPE, SJCampos-SP

Dr. Francisco Carlos Rocha Fernandes
Membro da Banca/INPE, SJCampos-SP

Dr. José Hiroki Saito
Membro da Banca
Convidado UFSCAR, São Carlos-SP

Candidato: André Ricardo Fazzanaro Martinon

São José dos Campos, 30 de julho de 2003.

Dedico este trabalho aos meus
pais, ao meu irmão e a Ceila.

AGRADECIMENTOS

Primeiramente, quero agradecer ao Guga, por toda sua ajuda desde a Iniciação Científica e por ser o principal responsável por eu chegar onde cheguei. Por ter acreditado na minha capacidade e no meu trabalho. E por ter me ensinado muito do que eu sei.

Meus sinceros agradecimentos aos meus orientadores Dr. Stephan Stephany e Dr. Hanumant S. Sawant por toda ajuda durante o curso, pelos conselhos e por toda dedicação e paciência que foram muito importantes para a realização deste trabalho

Agradeço a todos do grupo de Radioastronomia Solar.

À minha família pelo incentivo recebido durante toda a minha formação.

À Ceila por sempre me incentivar e me apoiar. Sua amizade sempre foi muito importante.

E a todos os meus amigos que, direta ou indiretamente, contribuíram para a realização deste trabalho. Em especial ao Kleber e ao Rogério.

Um muito obrigado a todas as pessoas que de alguma forma me ajudaram e contribuíram para a realização deste trabalho.

RESUMO

Este trabalho propõe um sistema otimizado para tratamento e análise de dados em radioastronomia solar, específico para o Espectrógrafo Solar BSS – Brazilian Solar Spectroscope, INPE. Este sistema constitui uma extensão do software BSSData, desenvolvido anteriormente. No presente trabalho foi implementada a BSSLibrary, uma biblioteca otimizada de rotinas para manipulação dos dados, ainda em desenvolvimento, tendo sido completadas as rotinas de filtragem de imagens. A otimização foi feita através do uso de instruções de linguagem de máquina vetoriais, típicas dos processadores IA32 atuais. Essa biblioteca foi implementada na forma de uma Dynamic Link Library (DLL) de forma a compor um módulo independente de outros, tais como a interface com o usuário. Testes realizados demonstraram um expressivo ganho de desempenho com a vetorização. O trabalho proposto inclui a configuração de software do sistema de tratamento e análise de dados, bem como explora alternativas de visualização.

BSSDATA – AN OPTIMIZED SYSTEM PROTOTYPE FOR BSS DATA PROCESSING AND ANALYSIS

ABSTRACT

This work proposes an optimized system for data processing and analysis in Solar Radioastronomy, being specifically designed for the BSS - Brazilian Solar Spectroscope, at INPE. This system is an extension of the formerly developed BSSData software. In the current work it was implemented the BSSLibrary, an optimized library that includes routines for data processing. This library is being continuously developed and the set of image filtering routines has been completed. Optimization made use of vector instructions in assembly language, typical in current IA32 processors. This library was implemented as a Dynamic Link Library (DLL), in order to form a self contained module, not dependent from other modules, such as the user interface. Tests show that a good performance is attained through vectorization. The proposed work includes the software configuration of the data processing and analysis system and discusses some alternatives for data visualization.

SUMÁRIO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

CAPÍTULO 1

INTRODUÇÃO

O principal objetivo deste trabalho é apresentar o protótipo de um sistema otimizado para tratamento e análise de dados em radioastronomia solar, sendo estes dados provenientes do Espectrógrafo Solar BSS (Brazilian Solar Spectroscope). O BSS é um radiotelescópio varredor de freqüência de alta sensibilidade e alta resolução temporal e espectral, que é utilizado em observações de radiações solares na faixa de freqüência decimétrica realizadas no Instituto Nacional de Pesquisas Espaciais (INPE).

Este protótipo será uma proposta para uma nova versão do BSSData (Martinon e Fernandes, 2000; Martinon et al., 2002a; Martinon et al., 2002b), que além de agregar maior flexibilidade atenderá de forma eficiente os requisitos dos Pesquisadores do grupo FMI (Física do Meio Interplanetário), que é a necessidade do desenvolvimento constante de novas ferramentas que auxiliem no processo de análise de dados.

Neste capítulo serão apresentados a motivação, os objetivos e a contribuição deste trabalho no contexto da análise dos dados provenientes das observações solares realizadas no INPE, bem como, a organização do texto.

1.1 - Observações Solares e o BSS

As pesquisas da física do meio interplanetário destinam-se a estudar os fenômenos que se iniciam no Sol e se propagam através do espaço heliosférico, particularmente em direção à Terra. Estas pesquisas contribuem para a compreensão dos fenômenos físicos relacionados à atividade solar e à atividade interplanetária subseqüente. No INPE, a linha FMI vem investigando alguns problemas fundamentais da fenomenologia de explosões solares na faixa de freqüência decimétrica, geradas próximas às regiões da atmosfera solar, onde se observam os mecanismos de armazenamento e liberação dos flares correspondentes as explosões solares (Sawant et al., 2001). Os flares solares são fenômenos de natureza explosiva que liberam energia em forma de radiação em uma

grande faixa de comprimentos de onda, que vai desde raio γ (da ordem de Å), até ondas de rádio (da ordem de km), Este fenômenos são transitórios e possuem uma duração que vai desde milissegundos até horas. Para realizar tais investigações, foi desenvolvido um radiotelescópio varredor de freqüência de banda larga (1000-2500) MHz com alta resolução temporal e espectral, e alta sensibilidade utilizando uma antena parabólica de 9 metros de diâmetro, denominado Brazilian Solar Spectroscope (BSS).

O BSS está em operação regular no INPE, em São José dos Campos, desde 1996. Porém, apenas em abril de 1998, foram iniciadas as observações solares sistemáticas, com o sistema completo de aquisição e visualização dos dados digitalizados. O BSS é o único espectrógrafo solar com tais características no Hemisfério Sul (Fernandes, 1997). De modo que as observações solares realizadas com este instrumento aproximadamente entre 11 e 19 UT são de grande importância. Devido a sua localização (longitude ~45^{o} Oeste), entre 16 e 19 UT, o BSS é o único espectrógrafo em operação para observação solar e portanto preenche uma lacuna no monitoramento da emissão solar em rádio através de uma rede observatórios localizados em diferentes longitudes.

1.2 - Motivação e Objetivos

Desde abril de 1998, quando iniciaram-se as observações solares sistemáticas, muitas horas de observações foram registradas digitalmente. Foram observados muitos eventos relacionados com as explosões solares levando ao desenvolvimento de catálogos de explosões solares (Fernandes, 2003 a, b, c). Estes catálogos possuem aproximadamente 340 grupos explosões solares, que podem ser classificados de acordo com as suas características morfológicas. E para cada grupo de explosões solares necessitamos de ferramentas específicas para auxiliar em sua análise.

Para visualizar e analisar esses grupos de explosões contamos com dois softwares: BSSView (Faria, 1999) e BSSData (Martinon e Fernandes, 2000; Martinon et al., 2002a; Martinon et al., 2002b). Sendo o BSSView voltado para visualização e geração de imagens. E o BSSData para tratamento e análise dos dados. Mas, ambos os softwares, não oferecem todas as ferramentas necessárias, principalmente, para análise dos dados referentes às explosões solares.

Dentre estas ferramentas podemos citar a necessidade de extrair informações referentes às explosões de forma automatizada. Mas antes, precisamos realizar um tratamento nos dados brutos para que os algoritmos de extração de dados obtenham bons resultados.

Para solucionar esta demanda por novas ferramentas de análise de dados, o software BSSData deverá ser reestruturado. Portanto, o objetivo deste trabalho é apresentar um protótipo de um sistema otimizado que atenda essa demanda.

1.3 - O Protótipo do Sistema Otimizado

A versão atual do BSSData já é totalmente modular, tendo as rotinas relativas à interface com o usuário desvinculadas das rotinas que manipulam os dados. Além de possuir uma biblioteca com rotinas otimizadas por vetorização, a BSSLibrary (Martinon, 2002b).

Mas como foi exposto anteriormente, existe uma demanda pelo desenvolvimento de novas ferramentas. Atualmente para se acrescentar uma nova ferramenta ao BSSData algumas rotinas devem ser codificadas em diferentes módulos (classes) que compõem todo o BSSData, necessitando também compilar todo o seu código fonte. Além, é claro, de exigir um conhecimento de como está estruturado todo o sistema.

Para sanar estas dificuldades será apresentado um protótipo do que deverá ser um sistema ideal para tratamento e análise de dados do BSS. Este sistema será composto por um framework, que fará a interface com o usuário, e por plug-ins que permitirão acrescentar funcionalidades de forma progressiva, moldando, desta forma, o sistema de acordo com as necessidades.

Resumindo, alguns benefícios advindos deste modelo de desenvolvimento seriam:

- Abertura de qualquer formato de arquivo de dados, desde que possua as informações básicas necessárias para a geração de um espectro dinâmico. Tais como quantidade de canais e varreduras, freqüência e tempos inicial e final de observação, etc.

- A modularização via plug-ins, facilita o desenvolvimento de novas rotinas, permitindo a expansão gradual do software de acordo com as necessidades.

- Pode-se criar templates para cada tipo de plug-in que o software suporta, assim o programador que venha desenvolver novos plug-ins deverá apenas se preocupar com o código que dá a funcionalidade ao plug-in, não necessitando conhecer toda a estrutura do sistema.

1.4 - Descrição dos Capítulos

Sendo o objetivo deste trabalho apresentar um sistema otimizado que supra a demanda por novas ferramentas para análise de dados de explosões solares, no Capítulo 2 serão apresentados aspectos observacionais sobre as explosões solares, suas características morfológicas e, também, um levantamento de algumas ferramentas para processamento e análise de dados.

O Capítulo 3 apresenta um breve histórico e a descrição do Espectrógrafo Solar BSS, bem como, o formato dos dados por ele digitalizados.

No Capítulo 4, nos concentraremos no sistema para tratamento e análise de dados. Será apresentado um histórico do desenvolvimento do software BSSData, além de uma descrição da versão atual. Serão abordadas as técnicas de otimização e visualização adotadas. E principalmente, será apresentado um protótipo para o desenvolvimento de uma nova versão do BSSData.

Finalmente, o Capítulo 5 apresenta as conclusões e os comentários finais.

CAPÍTULO 2

OBSERVAÇÕES RADIOASTRONÔMICAS EM FÍSICA SOLAR

A seguir definiremos brevemente algumas terminologias usadas em radioastronomia solar. Na Seção 2.2 descreveremos as características morfológicas dos grupos de explosões registrados pelo BSS. E finalmente na Seção 2.3 mostraremos um levantamento das ferramentas necessárias ao processo de análise de dados, necessidade esta que levou ao desenvolvimento do software BSSData e agora a propor um sistema otimizado para tratamento e análise dos dados do BSS.

2.1 Radioastronomia Solar

2.1.1 Flares Solares

No dia 1º de setembro de 1859, dois pesquisadores, Carrigton e Hudgson, observaram, simultânea, mas independentemente, um grupo de aspectos brilhantes na superfície do sol, tratava-se do primeiro flare solar observado em luz branca.

Segundo a definição de Zirin (1988, p. 347) "um flare solar é um crescimento transitório na brilhância em H-α, com intensidade, no mínimo, duas vezes maior que a intensidade cromosférica, geralmente impulsivo e acompanhado de algum crescimento em raios-X e fluxo rádio".

Os flares solares são fenômenos explosivos que liberam energia total da ordem de ($10^{26} - 10^{32}$ ergs), na forma de radiação eletromagnética, num grande intervalo de comprimentos de onda, desde raios γ (da ordem de Å), até ondas de rádio (chegando a km), liberando ainda partículas carregadas (elétrons e prótons). Produzem violentos fenômenos magnetohidrodinâmicos, tais como ondas de choque e ejeção de matéria e

englobam desde efeitos fotosféricos até aqueles causados na atmosfera terrestre e em escalas de tempo que vão desde sub-segundos à horas (Brown et al., 1981).

2.1.2- Explosões Solares Decimétricas

Na faixa de freqüências decimétricas (1000-3000 MHz) são possivelmente observados basicamente dois tipos de emissões: emissões com ou sem estruturas finas.

As explosões decimétricas podem apresentar vários tipos de estruturas finas em tempo e/ou em freqüência, pode-se ver exemplos em Allaart et al. (1990), mas as mais importantes são as explosões tipo III decimétricas e suas variantes (Benz et al., 1981; Benz et al. 1983; Sawant et al., 1987; Sawant et al., 1990 e Allaart et al., 1990).

Neste trabalho iremos nos concentrar nas estruturas finas registradas pelo BSS (dots, fiber, lace, patch, spikes, tipo III e zebra).

2.2 Tipos de Explosões

Os dados registrados até o momento pelo BSS, cerca de 340 grupos de explosões solares entre 1999 e 2002, foram classificados e catalogados (Fernandes, 2003 a, b, c) em 8 grupos distintos de acordo com suas características morfológicas. Desta forma as explosões podem ser do tipo dots, fiber, lace, patch, spikes, tipo III, zebra e emissão contínua ou não classificada/complexa. Mas neste trabalho apenas 7 tipos serão abordados, pois como o interesse prioritário das investigações são as explosões que contenham estruturas finas não iremos abordar as emissões contínuas, já que além de não apresentarem estruturas finas não necessitam de ferramentas específicas para a sua análise.

A seguir discutiremos cada tipo morfológico de explosão solar e espectros dinâmicos serão mostrados.

2.2.1 – Dots

As emissões tipo dot, são estruturas finas no espectro rádio, caracterizadas pela curta duração e largura de banda em freqüência bastante estreita. Por serem de curta duração, muitas vezes, são registradas nos limites de resolução instrumental. São emissões solares que apresentam variação da intensidade em função da freqüência. Além de aparecerem isoladas podem aparecer na forma de correntes, mostrando variações de intensidade em função da freqüência com acentuada taxa de deriva em freqüência. As emissões tipo dot, na faixa de freqüências decamétricas, foram revisadas por Bhonsle et al. (1979).

As emissões tipo dot foram observadas em ondas decimétricas por Sawant et al. (2002a). Nesta faixa de freqüência, suas características observacionais são comparáveis as de emissões registradas anteriormente em ondas decamétricas (34.0-35.5 MHz) e milimétricas (4000-8000 MHz) (Allaart et al. 1990).

Como características das emissões tipo dot individuais e na forma de correntes de dots (Sawant et al., 2002a), temos:

- Duração: ~ 50-100 ms;
- Banda em freqüência: ~ 5-15 MHz;
- Intensidade: ~ 20-300 u.f.s.;
- Largura de banda da corrente: ~ 80-200 MHz;
- Duração da corrente: ~ 150-500 ms;
- Taxa de deriva da corrente: ~ 180-1200 MHz/s;
- df/f : 5.0x10^{-3} MHz

As Figuras 2.1 e 2.2 apresentam alguns grupos de explosões tipo dot e estruturas finas com variação de intensidade com freqüência que foram observados pelo BSS (Fernandes, 2003, a, b, c).

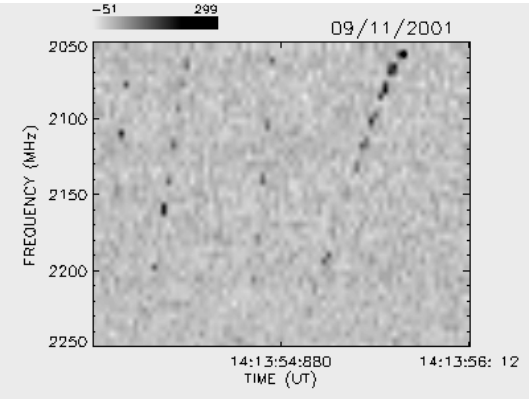

FIGURA 2.1 Correntes de dots registradas no flare solar do dia 11/09/2001.

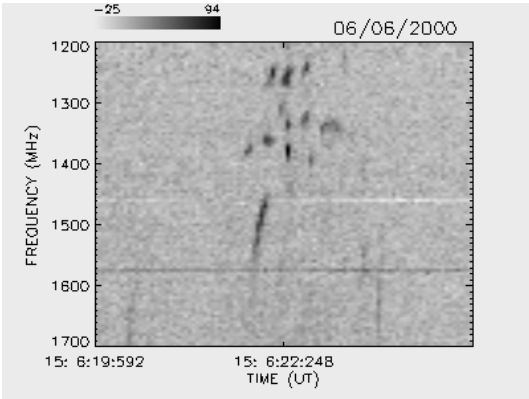

FIGURA 2.2 – Vários dots registrados no flare do dia 06/06/2000.

2.2.2 Fiber

Explosões solares com taxa de deriva intermediária (IMD) foram detectadas pela primeira vez por Young et al. (1961). Esta designação vem do fato que os valores de suas taxas de deriva em freqüência estão entre os das explosões tipo II (abaixo de 1 MHz s^{-1}) e tipo III (acima de 100 MHz s^{-1}). O termo fiber, também utilizado para este tipo de explosões com taxa de deriva intermediária foi introduzido por Slottje (1981). Tipicamente, emissões tipo fiber são caracterizadas por seqüências de "franjas" de emissão (intensas) geralmente sobrepostas à emissão contínuo de variação lenta, normalmente ocorrendo na fase final das explosões do tipo IV. Geralmente, aparecem em grupos de 10 até centenas de explosões individuais, cada uma tendo uma banda total da ordem de poucas centenas de MHz.

Emissões tipo fiber foram relatadas por Thompson e Maxwell (1962), Slotjje (1972), Bernold (1980), Aurass et al. (1987) e Mann et al. (1987), em ondas métricas. Benz e

Mann (1998) investigaram este tipo de emissão na faixa de freqüências decimétricas entre 1 - 3 GHz.

Recentemente, Fernandes et al. (2003d) investigaram emissões tipo fiber registradas pelo BSS com alta resolução temporal (20-50 ms) em ondas decimétricas (950-2650 MHz). De acordo com as investigações de Benz e Mann (1998) e Fernandes et al. (2003d), as principais características de uma única explosão com taxa de deriva intermediária observada acima de 1 GHz são:

- Duração total: 0,2-2,0 segundos;
- Banda em freqüência total: 50-300 MHz;
- Taxa de deriva em freqüência (normalmente negativa): ~ 30-300 MHz/s;
- Separação entre fibras consecutivas: ~ 30-120MHz;
- Banda instantânea: ~ 2-12 %;

As Figuras 2.3 e 2.4 mostram alguns grupos de explosões tipo fiber que foram observados pelo BSS (Fernandes, 2003, a, b, c).

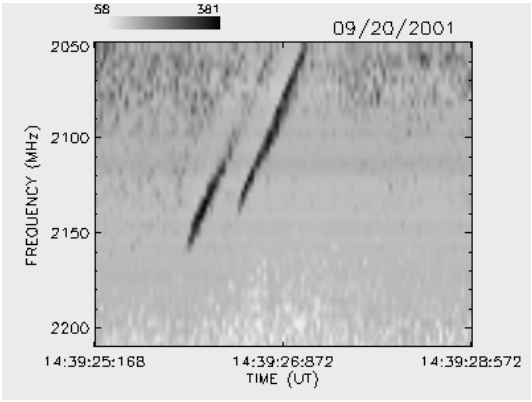

FIGURA 2.3 – Estruturas tipo fiber registradas no flare solar do dia 20/09/2001.

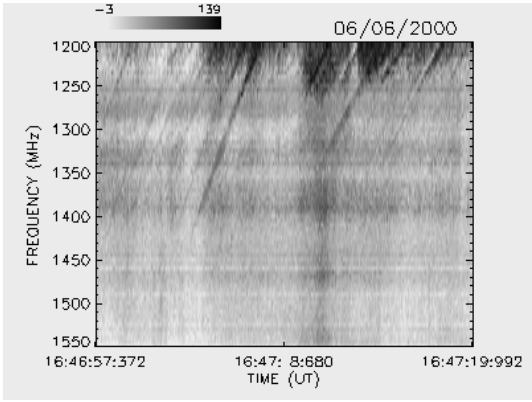

FIGURA 2.4 – Estruturas tipo fiber registradas no flare solar do dia 06/06/2000.

2.2.3 - Lace

Explosões tipo lace (assim batizadas por Jiricka et al., 2001) são caracterizadas por emissões de longa duração (minutos a horas) apresentando estruturas com intensidade oscilando rapidamente em função da freqüência, geralmente sobrepostas a variações lentas (emissão contínua) da ordem de minutos.

São explosões bastante raras, geralmente associadas com emissões tipo II (com presença de ondas de choque) presentes em flares intensos ou ejeções de massa coronais (CME).

Emissões tipo lace foram recentemente registradas na faixa de ondas decimétricas simultaneamente pelo BSS e pelos espectrógrafos do Observatório Ondrejov (Karlický et al., 2001)

As Figuras 2.5 e 2.6 mostram algumas partes de grupos de explosões tipo lace de longa duração que foram observados pelo BSS (Fernandes, 2003, a, b, c).

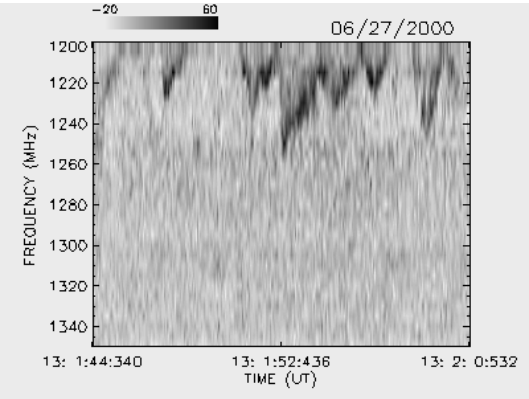

FIGURA 2.5 – Estruturas tipo lace registradas no flare solar do dia 27/06/2000.

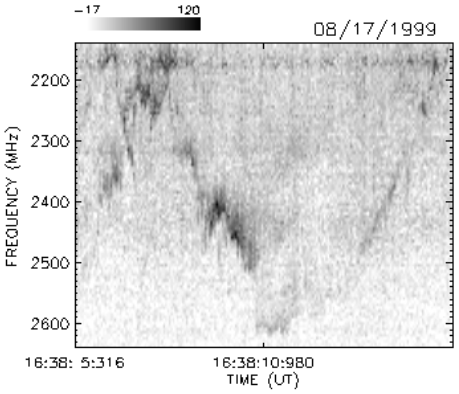

FIGURA 2.6 – Estruturas tipo lace registradas no flare solar do dia 17/08/1999.

2.2.4 Patch

Emissões designadas como tipo patch mostram banda estreita em freqüência (50 - 400 MHz) e ausência de deriva em freqüência, semelhantes a manchas ou borrões (patch) no espectro dinâmico em rádio. Geralmente aparecem em grupos. A duração de cada elemento individual varia de cerca de um a dezenas de segundos, porém as emissões tipo patch apresentam largura de banda e duração bastante variáveis, cobrindo extensas faixas de valores.

Bruggmann et al. (1990) e Smith e Benz (1991) registraram emissões tipo patch, respectivamente nas bandas de freqüência de 7,4-8,2 GHz e 3,40-3,55 GHz. Dados do Espectrógrafo Phoenix também evidenciaram emissões tipo patch na faixa de 100-3000 MHz (Isliker e Benz, 1994). Uma emissão peculiar, também classificada como emissão tipo patch, foi observada pelo BSS, em torno de 1600 MHz (Fernandes et al., 1996a, 1996b). Além disso, pela primeira vez foram registradas pelo BSS explosões tipo patch apresentando emissão central mais intensa, circundada por um envoltório difuso.

As Figuras 2.7 e 2.8 mostram grupos de explosões tipo patch que foram observados pelo BSS e catalogados (Fernandes, 2003, a, b, c).

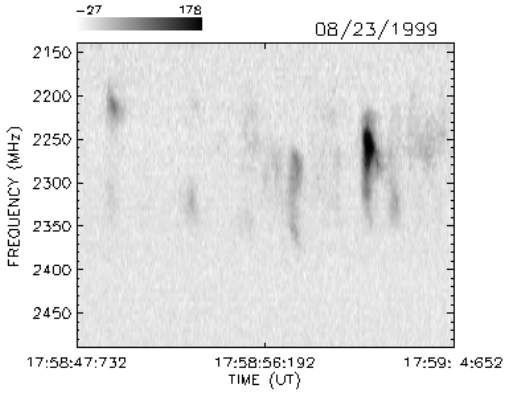

FIGURA 2.7 – Estruturas tipo patch registradas no flare solar do dia 23/08/1999.

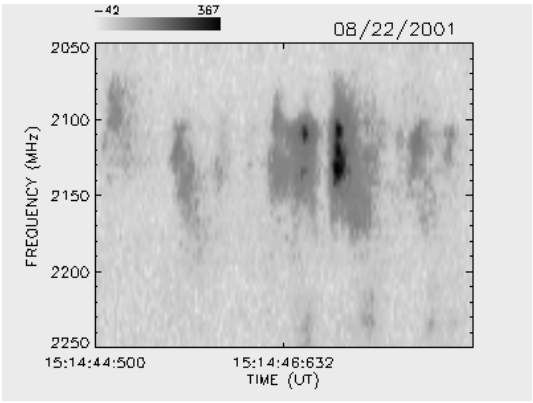

FIGURA 2.8 – Estruturas tipo patch registradas no flare solar do dia 22/08/2001.

2.2.5 Spikes

Emissões tipo spikes são caracterizadas por emissões muito rápidas (algumas dezenas de milisegundos) e de banda muito estreita no espectro em rádio, sendo que cada spike individual apresenta apenas algumas dezenas de MHz (Benz, 1986). As emissões tipo spikes ocorrem em grupos de centenas ou até milhares de eventos individuais, com duração total de segundos ou até vários minutos. Estruturas semi-harmônicas (razão 1:1.2) dentro de grupos de spikes foram registrados pelo BSS.

As Figuras 2.9 e 2.10 apresentam exemplos de grupos de explosões tipo spikes observados pelo BSS (Fernandes, 2003, a, b, c).

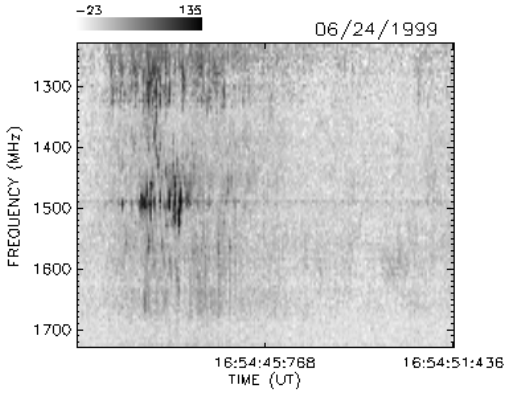

FIGURA 2.9 – Estruturas tipo spikes registradas no flare solar do dia 24/06/1999.

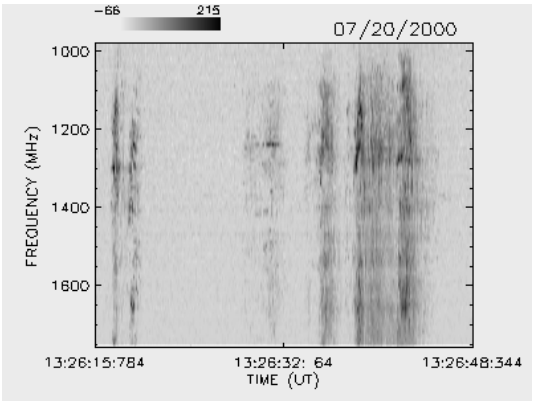

FIGURA 2.10 – Estruturas tipo spikes registradas no flare solar do dia 20/07/2000.

2.2.6 Tipo III

As explosões solares tipo III são uma das primeiras formas de rádio emissão solar descobertas e um dos aspectos mais característicos da atividade solar. São talvez o tipo de rádio emissão solar mais estudado na FMI (Sawant et al., 1994; Fernandes, 1997; Melendez et al., 1999). Recentemente, emissões tipo III decimétricas (acima de 2000 MHz) foram registradas pelo BSS com alta resolução temporal de 20 ms (Cecatto et al., 2003).

As principais características das explosões solares tipo III decimétricas foram resumidas em Fernandes (1997) e Melendez et al. (1999):

a) largura de banda instantânea dependente da freqüência;

b) a taxa de deriva em freqüência da ordem de centenas de MHz/s decresce com o decréscimo da freqüência. A velocidade do feixe de elétrons energéticos, estimada a partir da taxa de deriva, varia entre 15% e 30% da velocidade da luz;

c) raramente ocorrem explosões tipo III isoladas, mas em grupos de 10 ou mais explosões. Freqüentemente ocorrem na fase impulsiva dos flares;

d) a duração é da ordem de 100-600 ms;

e) os tempos de excitação (~ 20-120 ms) são mais curtos que os tempos de decaimento (~ 20-200 ms) e o decaimento é exponencial;

f) as alturas típicas onde são observadas variam de 0,2 a 2 raios solares.

g) podem apresentar banda estreita, intermediária e larga variando entre alguns MHz (próximo da resolução instrumental) até dezenas de MHz (~ 500 MHz).

As Figuras 2.11 e 2.12 de grupos de explosões tipo III observados pelo BSS (Fernandes et al., 2003, a, b, c).

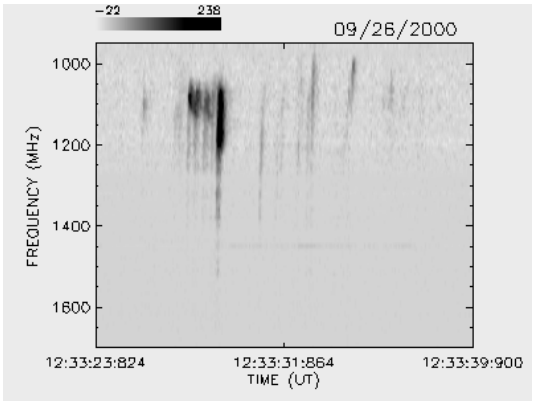

FIGURA 2.11 – Estruturas tipo III registradas no flare solar do dia 26/09/2000.

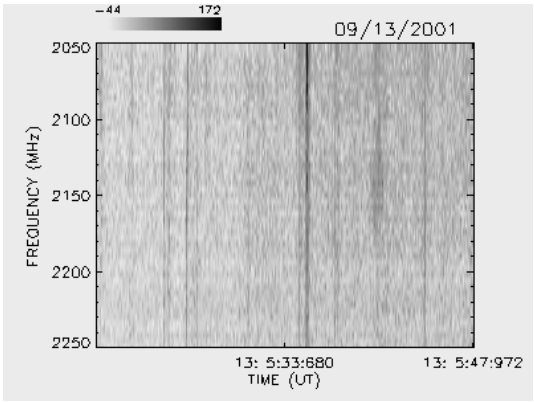

FIGURA 2.12 – Estruturas tipo III registradas no flare solar do dia 13/09/2001.

2.2.7 Zebra

As explosões tipo zebra são caracterizadas por seqüências de emissões quasi-paralelas de banda muito estreita, que ocorrem ao longo de minutos, com baixa e variável taxa de deriva em freqüência. Originalmente, estas emissões foram registradas em freqüências métricas (Tarnstrom e Philip, 1971; Slottje 1972b; Aurass & Chernov 1983), mas recentemente foram observadas em freqüências decimétricas pelo BSS (Sawant et al.,

2002b). Além disso, pela primeira vez emissões apresentando padrões tipo zebra foram observadas exibindo estrutura harmônica nas freqüências de 1650 MHz e 3500 MHz, registradas pelo BSS e pelo Observatório Ondrejov, respectivamente (Sawant et al., 2002 c).

As principais características das emissões tipo zebra na faixa de ondas decimétricas são:

- Banda instantânea em freqüência: ~ 10 MHz;
- Separação média entre duas emissões consecutivas: ~ 70-90 MHz;
- Taxa de deriva em freqüência: ~ 90-150 MHz/s;
- Duração global: segundos a minutos.

A Figuras 2.13 e 2.14 mostram alguns grupos de explosões tipo zebra que foram observados pelo BSS (Fernandes, 2003, a, b, c).

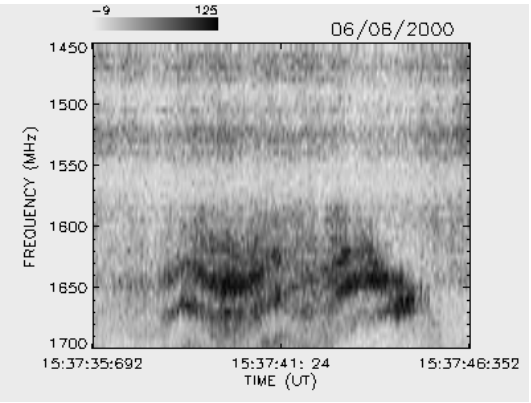

FIGURA 2.13 – Estruturas tipo zebra registradas no flare solar do dia 06/06/2000.

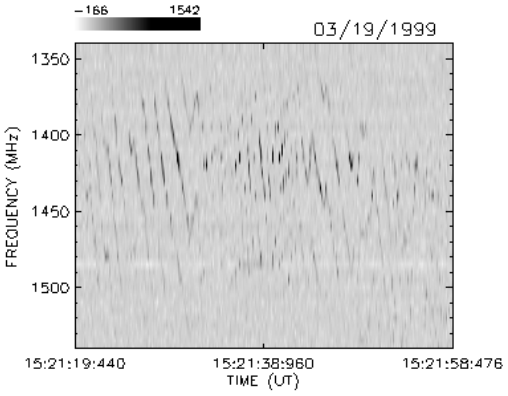

FIGURA 2.14 – Estruturas tipo zebra registradas no flare solar do dia 03/19/1999.

2.3 Processamento e Análise de Dados Radioastronômicos em Física Solar

Originalmente os dados brutos adquiridos pelo BSS, são armazenados em arquivos que contém apenas um cabeçalho, com informações básicas sobre a aquisição, e a matriz de dados (Mais detalhes em 30). Desta forma, antes de serem exibidos ao usuário, estes dados precisam passar por uma série de procedimentos de pré-processamento de dados. Estes procedimentos devem ser feitos pelo software que é usado na visualização e análise dos dados, no caso o BSSData. Tais procedimentos são por exemplo:

1) Transpor a matriz de dados: Uma vez que a matriz de dados é armazenada, no arquivo de dados, tendo nas abscissas associados os canais de freqüência e nas ordenadas as varreduras e como a imagem que devemos exibir ao usuário é a transposta desta matriz, este procedimento torna-se imprescindível.

2) Encontrar o menor e o maior elemento da matriz de dados: Valores necessários para que a interface com o usuário possa se moldar aos dados. Além de várias outras rotinas internas que necessitam de tais valores.

3) Calcular a resolução temporal (DT): O cabeçalho do arquivo de dados já possui o DT que é informado antes do início da digitalização como configuração da observação. Mas este valor não representa necessariamente a taxa com a qual os dados são adquiridos, existe sempre uma pequena diferença. Então, para que a escala de tempo seja a mais exata possível, deve-se calcular este novo DT. De outra forma, teríamos uma defasagem na escala de tempo.

4) Calcular o valor médio dos dados: O valor da média aritmética dos elementos da matriz é importante no desenvolvimento de alguns filtros, principalmente na etapa de validação do código.

5) Calibração: Existe a necessidade de realizar uma calibração nos dados, pois estão em uma unidade arbitrária (como a placa digitalizadora tem uma precisão de 12-bit os dados são representados por uma faixa de valores que vai de 0 à

4095). O ideal seria que estes valores estivessem sendo representados em unidades de fluxo solar (u.f.s).

6) Mapeamento de cores: Os dados precisam ser mapeados em uma escala de valores de 8-bit de precisão (valores de 0 à 255), pois as paletas de cores, utilizadas pelo software, são compostas de 256 cores.

Depois destes procedimentos (pré-processamento), os dados estariam prontos para serem visualizados. Assim um espectro dinâmico (ver Seção 3.3) é montado e exibido ao usuário. A partir do espectro dinâmico, o usuário pode dar início à análise de dados. Mas antes de discutirmos a análise de dados, vamos nos deter ao tratamento de dados, que é uma etapa muito importante, para o processo de análise e extração de atributos.

2.3.1 Tratamento de Dados

Na análise detalhada de cada grupo de explosões solares observado deve-se considerar a variação do fluxo do background solar (fluxo do sol calmo) em função da freqüência e a variação temporal, além da complexidade das explosões e estruturas finas registradas superpostas ao background variável.

O sinal de background registrado pelo BSS não é homogêneo em freqüência, em virtude da resposta do sistema não ser exatamente idêntica para cada freqüência. Tornando esse sinal mais homogêneo as explosões mais fracas, que antes estavam misturadas ao próprio background, ficam mais realçadas. Isso é conseguido através de técnicas de filtragem digital de dados.

Os filtros digitais são usados geralmente para duas finalidades: (1) separação de sinais e (2) restauração de sinais. A separação de sinais é necessária quando um sinal tenha sido contaminado com interferência, ruído, ou outros sinais como no caso deste trabalho. Já a restauração de sinais é usada quando um sinal tenha sido distorcido de alguma forma.

Para realçar as explosões, principalmente no caso das emissões mais fracas, optou-se pelo uso de algoritmos de PDI (Processamento Digital de Imagens) com o objetivo de obter um melhor contraste entre as áreas da imagem que apresentam informações sobre

explosões e o background, para posterior aplicação de algoritmos para detecção de bordas das áreas com explosões, classificando a imagem em duas áreas distintas: com e sem explosão.

De posse desta imagem seriam extraídos e quantificados os atributos de interesse, tais como obtenção dos tempos de subida e descida do sinal de uma explosão, da duração total do fenômeno, da taxa de deriva em freqüência, da banda espectral, da localização do máximo, etc.

2.3.2 Análise de Dados

Atualmente, o software utilizado para o tratamento e análise de dados do BSS, o BSSData, conta apenas com ferramentas que permitem analisar os dados e obter informações apenas visualmente. Seria interessante criar mecanismos que automatizassem parte do processo de análise, principalmente as tarefas de extração de atributos, onde necessitamos de ferramentas que possibilitem quantificar de forma mais precisa as informações. Outro tipo de ferramenta imprescindível para a análise de dados seria uma que permitisse uma análise conjunta com outros dados advindos de outros observatórios.

Em Melendez et al. (1999) foi elaborada uma análise estatística detalhada das emissões tipo III, onde podemos notar a complexidade e a necessidade de um sistema integrado, que permita visualizar, extrair e analisar os diversos tipos de emissões de explosões solares.

As informações necessárias para realizar a análise de dados registrados pelo BSS seriam, em termos gerais, obter as características temporais, espectrais e da intensidade do sinal. Mas como vimos na Seção 2.2, existem vários tipos de explosões solares e cada qual com características morfológicas distintas, havendo, portanto, a necessidade de ferramentas específicas para auxiliar na análise de cada tipo. Tendo o trabalho de Melendez et al. (1999) como ponto de partida, elaborou-se um levantamento preliminar de quais informações precisam ser obtidas para cada tipo de explosão, que serão descritas com mais detalhes a seguir.

2.3.2.1 - Dots

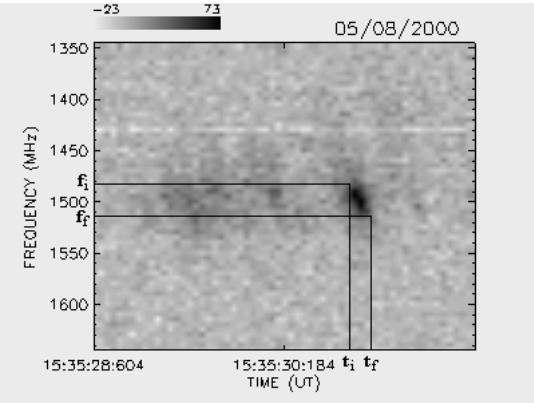

FIGURA 2.15 – Extração do tempo inicial (t_i) e final (t_f) e da freqüência inicial (f_i) e final (f_f) de um dot isolado.

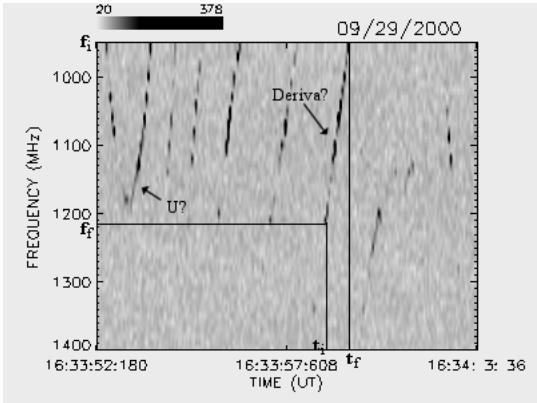

FIGURA 2.16 – Extração de informações de uma corrente de dots.

As emissões tipo dots podem ser analisadas isoladamente ou em conjunto, quando formam uma corrente de dots. Quando analisadas isoladamente, como mostrado na FIGURA 2.15, podemos extrair a freqüência inicial e final, o tempo inicial e final e a intensidade do sinal. Quando temos vários dots formando uma corrente, podemos determinar a taxa de deriva em freqüência da corrente, se forma uma estrutura em forma de U ou de U invertido. A FIGURA 2.16 exemplifica os atributos que podem ser extraídos quando temos uma corrente de dots.

A taxa de deriva em freqüência pode ser calculada de duas formas: direta, usando os valores de Δf e Δt, ou por regressão (linear ou não-linear) dos picos de cada dot isolado, que formam uma corrente.

Ferramentas que façam uma análise estatística também seriam interessantes, por exemplo para achar a duração média de todos os dots de um arquivo de dados, bem como a banda de freqüência média etc.

2.3.2.2 - Fiber

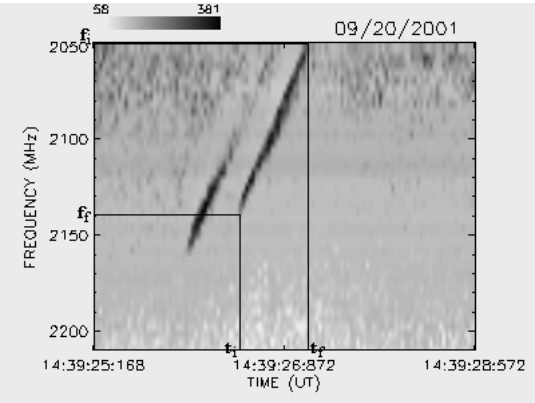

FIGURA 2.17 Extração de informações, de uma emissão fiber, de uma estrutura individual.

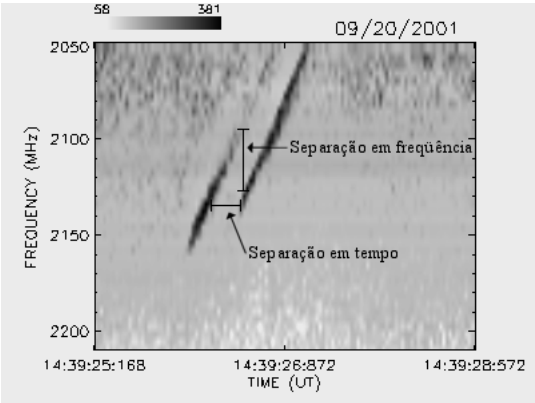

FIGURA 2.18 Determinação da separação em freqüência e tempo de um conjunto de estruturas em uma emissão tipo fiber.

Assim como as emissões tipo dots as emissões tipo fiber também podem ser analisadas em conjunto ou isoladas. Quando analisadas em conjunto (FIGURA 2.18) queremos determinar a separação entre cada estrutura tanto em tempo quanto em freqüência.

Na FIGURA 2.17 exemplificamos o processo de extração de informações de uma única estrutura de uma emissão tipo fiber. Nota-se que podemos extrair a freqüência inicial e final, o tempo inicial e final e a intensidade do sinal ao longo de cada canal de freqüência. A partir destas informações podemos determinar a taxa de deriva em freqüência tanto a direta quanto a por regressão (não-linear) dos picos em cada canal de freqüência.

Pode-se determinar também a banda instantânea, que é a banda em freqüência de uma estrutura individual em um dado instante.

Ferramentas que informem dados estatísticos da emissão também são muito importantes. Por exemplo se quisermos saber qual a separação média em freqüência de um grupo da emissão tipo fiber em questão.

2.3.2.3 Lace

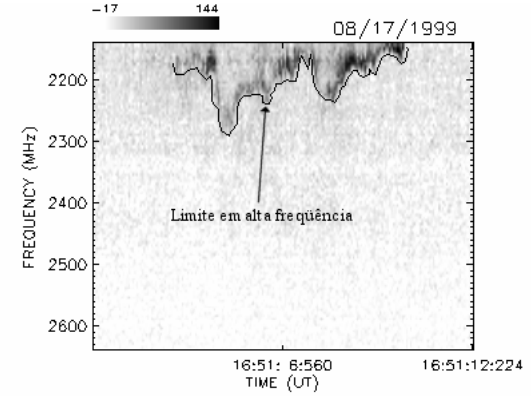

FIGURA 2.19 – Determinar o limite (borda) em alta freqüência de uma explosão tipo Lace.

A ferramenta mais importante para a análise de emissões tipo lace seria uma que permitisse determinar o limite, ou borda, em alta freqüência (FIGURA 2.19) e a partir da curva obtida realizar uma análise de série temporal.

2.3.2.4 Patch

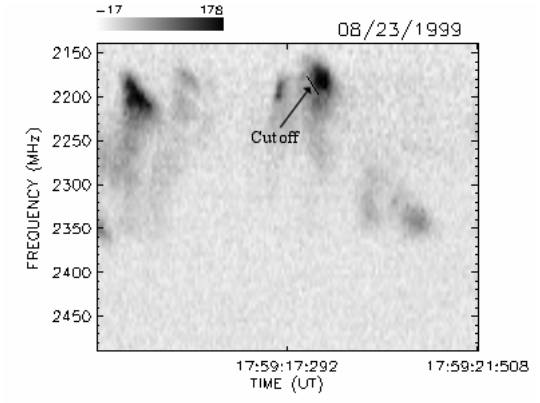

FIGURA 2.20 – Determinar se existe cutoff em uma explosão tipo patch.

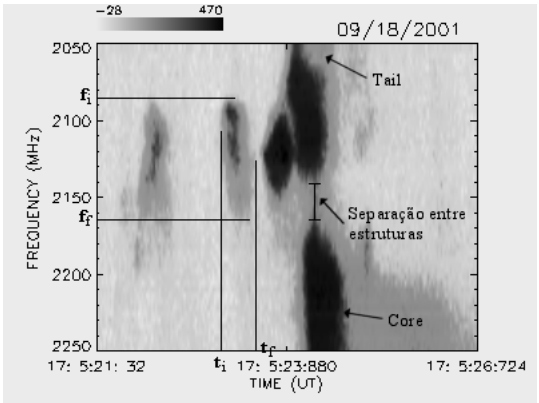

FIGURA 2.21 – Extração de informações de uma explosão tipo patch.

As emissões tipo patch nos permitem extrair diversas informações acerca de seu comportamento, como podemos notar nas FIGURA 2.20 e FIGURA 2.21. Assim como na maioria das emissões podemos determinar a banda em freqüência e duração de cada estrutura que compõe a emissão, através da freqüência inicial e final, e do tempo inicial e final. Quando temos um conjunto de estruturas podemos determinar separação entre cada estrutura tanto em freqüência quanto em tempo. Essa separação pode ser medida de centro a centro (o centro é o pico mais forte de uma estrutura), ou pelas bordas (FIGURA 2.21).

Determinar a razão entre as freqüências centrais, possibilitando constatar se são "harmônicos" ou não.

Determinar a razão entre intensidade do core e do tail. Como também os limites entre eles.

Podemos determinar também se houve um cutoff na estrutura (FIGURA 2.20), e calcular qual a taxa de deriva em freqüência do cutoff.

2.3.2.5 Spikes

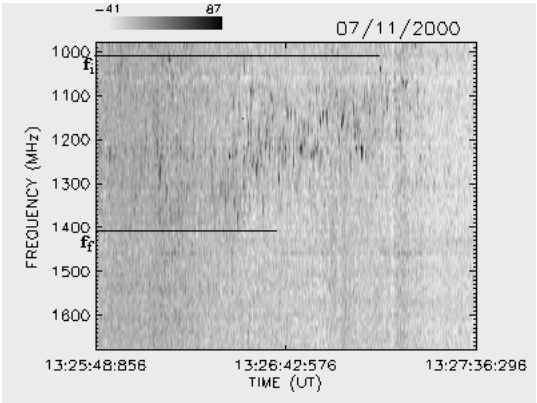

FIGURA 2.22 – Extração do intervalo em freqüência de uma emissão tipo spikes.

Pela complexidade apresentada pela emissão tipo spikes só podemos realizar uma análise global. Por exemplo:

- Usar Fourier para determinar intermitência ou periodicidade.
- "Correlação" para determinar harmônicos (ou semi-harmônicos).

Outra informação necessária seria o intervalo, banda em freqüência (FIGURA 2.22).

2.3.2.6 Tipo III

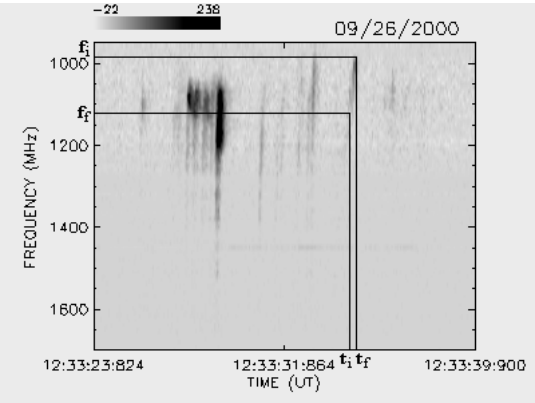

FIGURA 2.23 – Extração de informações, de uma emissão tipo III, de uma estrutura individual.

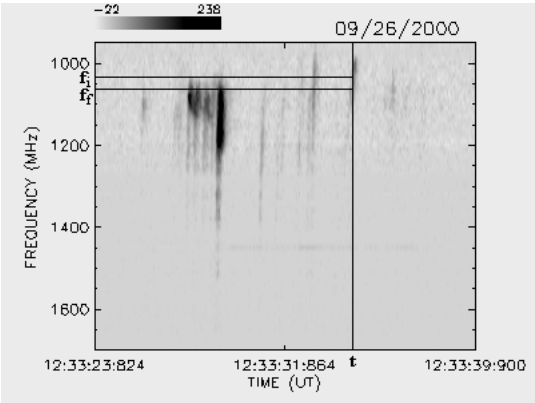

FIGURA 2.24 – Extração da banda instantânea dado um tempo qualquer, de uma emissão tipo III.

Basicamente precisaríamos extrair as mesmas informações que extraímos da emissão tipo fiber. Aqui cabe também uma análise individual ou global das estruturas que compõem a emissão. Nas figuras (FIGURA 2.23 e FIGURA 2.24) são mostradas as informações que precisam ser extraídas.

Outra ferramenta importante seria uma para plotar um perfil temporal em escala logarítmica que permitiria determinar o tempo de decaimento do sinal.

Ferramentas que façam uma análise estatística também seriam interessantes, por exemplo achar a duração média de todas as estruturas etc.

2.3.2.7 Zebra

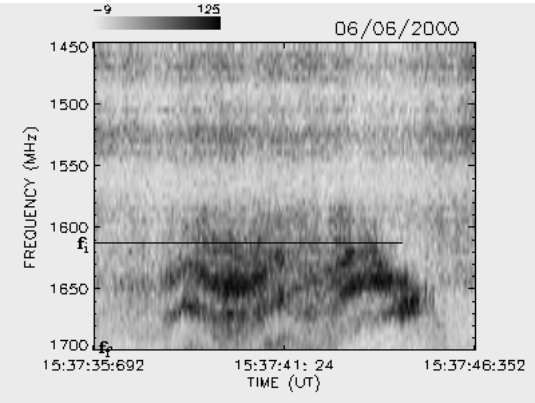

FIGURA 2.25 – Determinação do intervalo em freqüência de uma emissão tipo zebra.

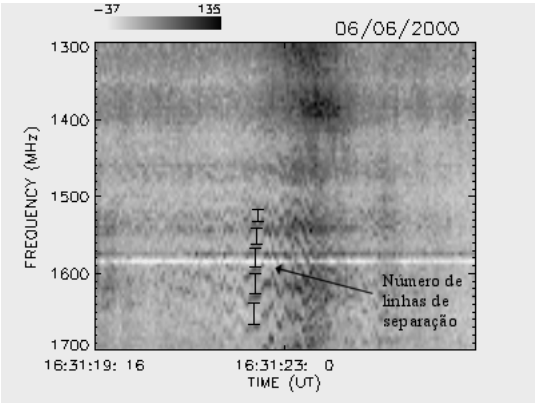

FIGURA 2.26 – Determinação do número de linhas de separação das estruturas de uma emissão tipo zebra.

Como esta sendo mostrado nas figuras, podemos determinar o intervalo, banda, em freqüência (FIGURA 2.25) de uma emissão tipo zebra. Determinar o número de linhas de separação das várias estruturas (FIGURA 2.26). Determinar a separação em freqüência entre cada estrutura.

CAPÍTULO 3

BRAZILIAN SOLAR SPECTROSCOPE (BSS)

O Capítulo anterior apresentou uma breve introdução à radioastronomia solar e uma descrição das características morfológicas dos grupos de explosões solares registrados pelo BSS, além de uma discussão sobre as ferramentas necessárias para o processamento e análise destes grupos de explosões.

A seguir serão apresentados um breve histórico do Espectrógrafo Solar BSS, uma descrição do instrumento e do formato dos dados por ele digitalizados.

3.1 Histórico

Em 1990, teve início a primeira etapa do projeto de desenvolvimento do espectrógrafo solar do INPE, com a instalação do refletor parabólico de 9 metros de diâmetro com um alimentador de banda estreita, operando na faixa de freqüência de (1600 ± 100) MHz (Sawant e Rosa, 1990; Sawant et al., 1991; Sawant et al., 1992). Foram então desenvolvidos e testados todos os elementos do sistema e da aquisição de dados. A partir de 1992, foi introduzido o sistema de aquisição digital de dados. Este instrumento esteve em operação regular até o final de 1994, tendo registrado cerca de 350 grupos de explosões solares (Fernandes, 1992; Sawant et al., 1993; Sawant et al., 1996).

A partir de 1995, teve início a nova etapa do projeto, com a instalação de um alimentador log-periódico de banda larga e modificação de todo o equipamento para operação na faixa de freqüência de 1000 a 2500 MHz. O espectrógrafo de banda larga entrou em operação em 1996, com observações iniciais principalmente para testes e aprimoramento do funcionamento do sistema, incluindo a aquisição digital dos dados em até 100 canais de freqüência e aquisição de tempo do GPS.

Em abril de 1998, o espectrógrafo batizado de Brazilian Solar Spectroscope (BSS), entrou definitivamente em operação regular, totalizando até o momento mais de 1500

horas de observação (Sawant et al., 1996; Fernandes, 1997; Sawant et al., 2000; Sawant et al., 2001).

3.2 Instrumento

TABELA 3.1 - Principais características do bss.

Antena Parabólica	9 metros de diâmetro
Montagem	Polar
Alimentador	log-periódico cruzado
Faixa de Freqüência	1000-2500 MHz
Resolução temporal	10, 20, 50, 100 ou 1000 ms
Resolução espectral	1-3 MHz
Precisão de tempo absoluta	3 ms
Sensibilidade	~3 u.f.s.
Número de canais	25, 50 ou 100
Visualização	tempo quase-real (1-20 min)
Campo de visada	todo o disco
Observação	~ 11:00-19:00 UT

FONTE: Adaptada de Fernandes et al. (2000, p. 35).

O BSS funciona em conjunto com uma antena parabólica de 9 metros de diâmetro, com montagem polar, em cujo foco opera um alimentador de banda larga composto de duas antenas log-periódicas cruzadas, que permite a aquisição das componentes horizontal e vertical do sinal. Atualmente, o sinal destas componentes é somado e então introduzido em um analisador de espectros (HP8559A), onde é feita a varredura de acordo com os parâmetros desejados. Na saída do analisador, os sinais de variação de tensão seguem para os sistemas de aquisição e monitoramento dos dados. A Tabela 3.1 apresenta as principais características do BSS. A FIGURA 3.1 mostra o diagrama representando o sistema do BSS. A FIGURA 3.2 mostra a antena de 9 metros com o alimentador de banda de larga.

A sensibilidade do BSS é cerca de 3 u.f.s., considerando uma resolução espectral de 3 MHz e uma combinação entre resolução temporal (10, 20, 50, 100 ou 1000 ms) e banda de freqüência de observação (100-1000 MHz)(Fernandes, 1997; Sawant et al., 2001).

Krüger e Voight (1995) apresentaram um levantamento dos espectrógrafos solares operando na Europa e recentemente, Bastian et al. (1999) listaram vários espectrógrafos solares em operação no mundo. Como mostra a Tabela 3.2, são poucos os espectrógrafos dedicados às observações solares em operação. Com exceção do BSS, todos os demais instrumentos estão localizados no hemisfério norte oriental (em latitudes acima de 40°). Deste fato resulta que o único espectrógrafo digital capaz de monitorar o Sol na faixa de ondas decimétricas, entre 16 e 19 UT, é o BSS. Essa cobertura exclusiva é ainda maior durante o inverno no hemisfério norte.

Além disso, as resoluções temporal e espectral disponíveis para o BSS são comparáveis ou melhores que as da maioria dos instrumentos em operação (Tabela 3.2). O BSS apresenta ainda a vantagem adicional da flexibilidade na escolha dos parâmetros observacionais, sendo dotado de um recurso de visualização dos dados em tempo quase-real (1 a 5 minutos após a aquisição de cada arquivo de dados, dependendo da duração escolhida para os arquivos).

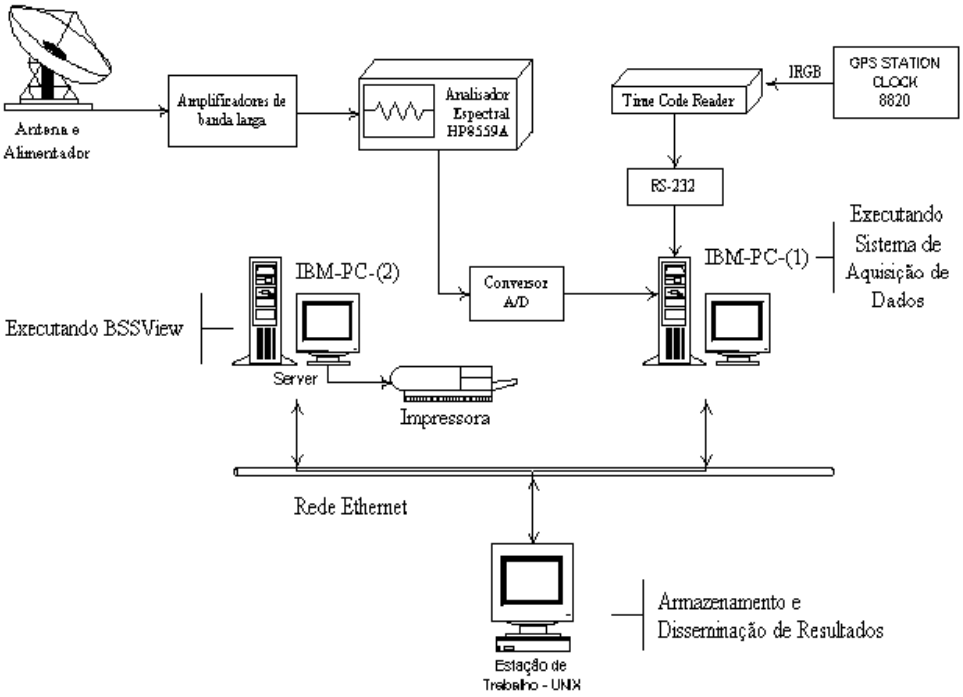

FIGURA 3.1 – Diagrama de blocos do Brazilian Solar Spectroscope – BSS.

FONTE: Adaptada de Fernandes et al. (2000, p. 36).

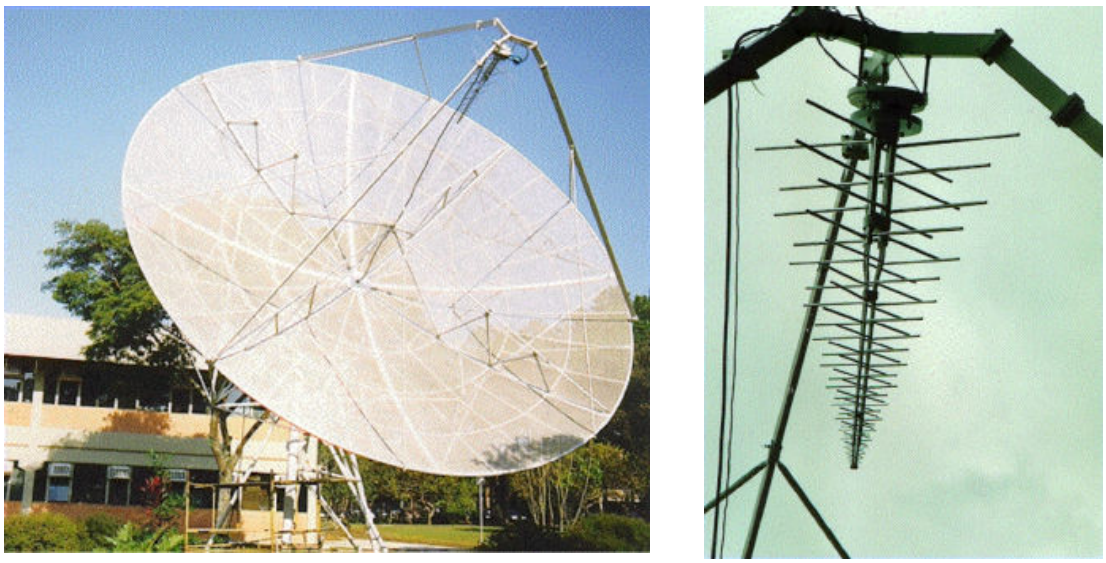

FIGURA 3.2 - Antena parabólica de 9 metros de diâmetro do BSS. No foco nota-se o alimentador log-periódico de banda larga (em destaque na imagem da direita).
FONTE: Adaptada de Fernandes et al. (2000, p. 36).

TABELA 3.2 - Espectrógrafos solares digitais em operação na faixa de ondas decimétricas.

Local	Faixa de Freqüência (MHz)	Resolução Temporal (ms)	Resolução Espectral (MHz)	Referência
Zurich, Suiça	100-4000	0,5-1000	1-3-10	Messmer et al., 1999
Ondrejov, Rep. Checa	800-2000	100	5	Jiricka et al., 1993
Ondrejov, Rep. Checa	2000-4500	100	5	Tlamicha, 1990
Beijing, China	1000-2000	50	20	Fu et al., 1995
S.J. dos Campos, BR	1000-2500	10-1000	1-3	Sawant et al., 2000b

FONTE: Adaptada de Fernandes et al. (2000, p. 36).

3.2.1 O Sistema para Aquisição e Tratamento dos Dados Digitalizados

Como mostrado na FIGURA 3.1, o sinal de saída do analisador de espectros é introduzido, junto com o pulso de sincronismo, em uma placa conversora analógico/digital de 12 bits (ADDA 12). O sinal digitalizado é armazenado localmente na memória RAM do IBM-PC(1). Dependendo da combinação de resolução temporal e banda de freqüência, podem ser digitalizados 25, 50 ou 100 canais de freqüência. A duração de cada arquivo de dados pode ser escolhida (1 a 20 minutos). Cada arquivo é então enviado a um segundo computador IBM-PC(2) para a visualização em tempo quase-real e análise preliminar, o que permite alterar os parâmetros das observações, tais como resoluções temporal/espectral de acordo com a necessidade (Sawant et al., 2001).

Junto com os dados digitalizados, o programa armazena também o tempo absoluto das observações através da aquisição de um código de tempo gerado pela GPS "Station Clock", do Centro de Controle de Satélites (CCS) do INPE. Um tradutor faz a

decodificação do sinal de tempo, que é modulado em 1kHz e transmitido, por meio de linha telefônica, ao tradutor de tempo do observatório solar, sendo gravado juntamente com os dados digitalizados (Faria, 1999). A precisão absoluta de tempo é de ~3 ms.

3.3 Formato dos Dados Digitalizados

Os dados adquiridos pelo BSS são armazenados em arquivos binários, em um formato próprio com a denominação ESP. Este arquivo é constituído de duas partes:

- Cabeçalho: onde são armazenadas informações referentes à aquisição, como por exemplo: tempos inicial e final, quantidade de canais, quantidade de varreduras etc.

- Matriz de dados: onde são armazenados os dados digitalizados.

O cabeçalho tem um tamanho fixo de 56 bytes, já a matriz de dados varia de acordo com as configurações da observação corrente. Na FIGURA 3.3 apresentamos a estrutura de dados escrita em linguagem C que define o cabeçalho de um arquivo ESP. Já a matriz de dados tem cada elemento armazenado em um inteiro de 16 bits.

Em regime normal de operação, o BSS digitaliza uma matriz de 6000 varreduras e 100 canais a cada 5 minutos, gerando um arquivo de 1.14 Mbytes. Desta forma temos um fluxo de dados de 13.7 Mbytes/hora, podendo chegar a um fluxo de dados máximo de 35 Mbytes/hora.

```
typedef unsigned short word;
typedef struct{word hour, min, sec, mile;} wTime;
typedef struct{word day, month, year;} wDate;
//---------------------------------------------------------------------------
typedef struct {
      word  status;
      word  df;            // Resolução espectral
      word  dt;              // Resolução temporal
      wTime time_obs;      // Horário do início da aquisição de dados
      wTime time_end;      // Horário do término da aquisição de dados
      wDate date;          // Data de aquisição
      word  freq_obs;      // Frequência inicial de observação
      word  freq_end;      // Frequência final de observação
      int   scans;         // Número de varreduras usados na digitalização
      word  channels;      // Número de canais usados na digitalização
      word  duration;      // Duração da digitalização em segundos
      word  data_min;      // Mínimo elemento do conjunto de dados (Valor típico = 0)
      word  data_max;      // Máximo elemento do conjunto de dados (Valor típico = 4095)
      char  file_name[12]; // Nome do arquivo de dados
```

```
} sHeader;
```

FIGURA 3.3 – Estrutura de dados do cabeçalho de um arquivo ESP.

Esses arquivos são exibidos ao usuário na forma de espectros dinâmicos, onde todos os canais de freqüência e varreduras são plotados em uma imagem bidimensional e tendo a intensidade do sinal sendo representada por cores de uma paleta de cores pré-definida. Na FIGURA 3.4 podemos ver um exemplo típico de um espectro dinâmico. Nas ordenadas estão associados os canais de freqüência enquanto nas abscissas as varreduras, ou seja, cada linha horizontal representa a variação da intensidade do sinal para uma certa freqüência em relação ao tempo.

Quando ocorre um evento (explosão solar), a intensidade do sinal aumenta em um ou mais canais de freqüência para um dado intervalo de tempo. Um exemplo de espectro dinâmico onde ocorre o registro de uma explosão solar pode ser visto na FIGURA 3.5.

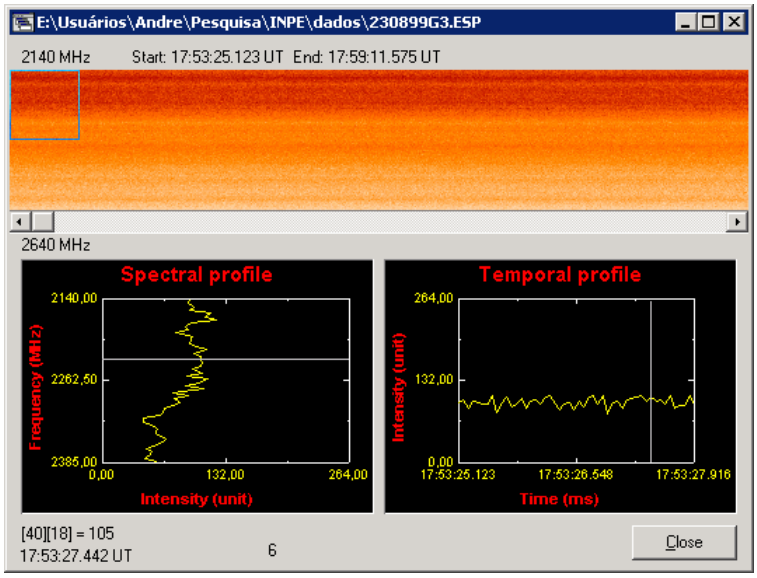

FIGURA 3.4 – Janela de visualização do software BSSData onde, na parte superior temos um exemplo de espectro dinâmico.

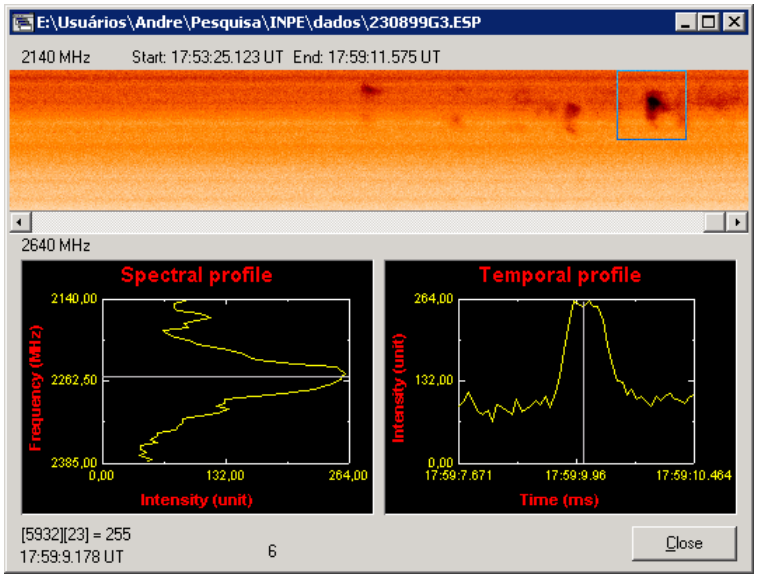

FIGURA 3.5 - Janela de visualização do software BSSData onde, na parte superior, temos um exemplo de espectro dinâmico com a ocorrência de uma explosão solar.

CAPÍTULO 4

O SOFTWARE BSSDATA

A necessidade de ferramentas específicas para análise dos dados registrados pelo BSS, levou ao desenvolvimento, em 1999, do software BSSData (FIGURA 4.1). No início como um projeto PIBIC/INPE (Martinon e Fernandes, 2000) de iniciação científica e sendo aperfeiçoado posteriormente no projeto de mestrado (Martinon et al., 2002b).

A seguir será apresentado o histórico do desenvolvimento do software BSSData. Em seguida mostraremos uma breve descrição da sua interface e das ferramentas que compõem a versão atual. Depois entraremos em mais detalhes sobre a implementação do software, sendo a Seção 4.3 destinada a informar os detalhes da vetorização das rotinas da BSSLibrary, a Seção 4.4 descreverá as rotinas e a estrutura de classes do software e na Seção 0 serão abordados os recursos de visualização presentes no software. Finalmente na Seção 0 será apresentado o protótipo do sistema otimizado para tratamento e análise de dados do BSS, objetivo principal deste trabalho.

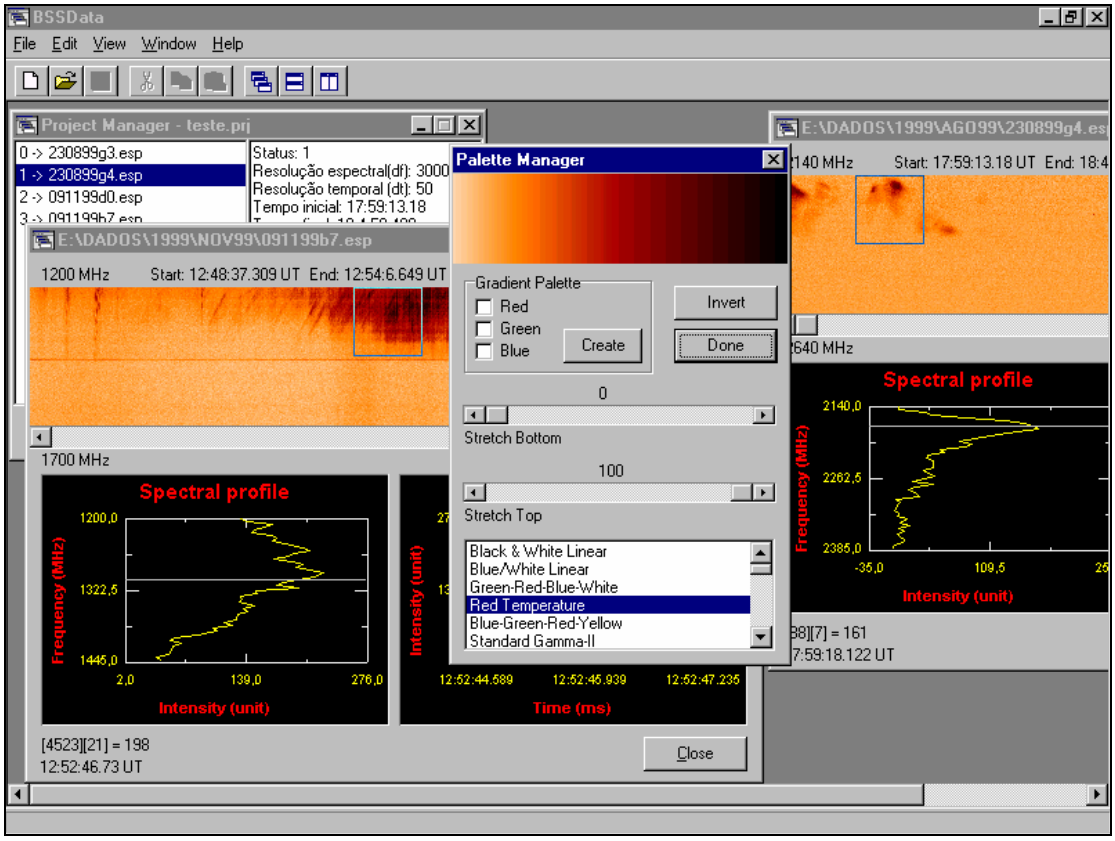

FIGURA 4.1 – Visão completa do funcionamento das ferramentas do software BSSData.

4.1 Histórico

Em meados de 1999, iniciou-se o desenvolvimento do núcleo do BSSData, este primeiro núcleo não possuía sequer uma interface gráfica, muito menos uma hierarquia de classes. As rotinas eram validadas apenas com aplicações console executadas em modo texto. Depois de se ter um conjunto significativo de rotinas e uma idéia básica de como deveria ser o software iniciou-se o desenvolvimento da interface gráfica assim como uma estrutura de classes. Foram definidas classes para:

- Manipular os datasets (datasets é um conjunto de dataset, e um dataset nada mais é do que um arquivo de dados);

- manipular a paleta de cores dos espectros dinâmicos;

- tratar as coordenadas da tela (em pixels) em qualquer outro tipo de coordenada (p. ex. coordenadas cartesianas);

- desenhar molduras com eixos graduados, título, subtítulos, linhas de grade etc, em uma janela gráfica.

Assim, no final de 1999, tínhamos um protótipo do BSSData e a partir de então se deu início ao desenvolvimento de novas rotinas e a interface gráfica foi sendo adaptada de acordo com as necessidades dos usuários.

Ao final de 2000 chegamos a uma versão estável (Martinon e Fernandes, 2000), com importantes ferramentas para auxiliar na análise dos dados do BSS. Com uma hierarquia de classes bem definida que facilita o desenvolvimento contínuo de novas ferramentas.

Em 2001, começo a cursar as disciplinas do mestrado e com os conhecimentos adquiridos e trabalhos realizados no decorrer do curso, novas rotinas foram desenvolvidas. Sendo integradas ao BSSData ao longo de 2002, quando o software passou por uma reestruturação de seu núcleo passando a ser executado através de uma DLL (Dynamic Link Library), batizada de BSSLibrary (Martinon et al, 2002b).

A BSSLibrary é uma biblioteca de rotinas vetorizadas (ver 0 para mais detalhes da vetorização) que se integra ao BSSData formando assim um software para análise de dados do BSS.

E agora queremos alcançar um novo grau de flexibilidade no desenvolvimento do BSSData. Essa nova proposta de desenvolvimento será discutida com maiores detalhes em 0.

Com o protótipo do sistema que será discutido em 0 almejamos obter uma maior flexibilidade no desenvolvimento do BSSData. Este sistema será baseado em plug-ins permitindo, assim, que o usuário molde o software BSSData de acordo com as suas necessidades. Esses plug-ins poderão ser habilitados ou desabilitados a qualquer momento, pelo usuário, mesmo com o software em execução. Assim, quando houver a necessidade de uma nova ferramenta, ou módulo, a mesma poderá ser codificada na

forma de um plug-in e posteriormente adicionada ao BSSData, sem que haja a necessidade de recompilar todo o seu código-fonte (do BSSData).

4.2 Descrição

O software BSSData (Martinon e Fernandes, 2000; Martinon et al., 2002a, b) (FIGURA 4.1) auxilia na análise dos dados provenientes do BSS e para tal possui ferramentas para destacar e determinar visualmente os parâmetros das explosões, manipular as cores do espectro dinâmico e organizar os dados em conjuntos distintos.

O núcleo da versão preliminar do BSSData foi desenvolvido em linguagem C++ mantendo portabilidade com qualquer compilador C++. Para construir todas as janelas que fazem a interface com o usuário foi utilizada a ferramenta de programação Borland C++ Builder 6.0. Como características principais dessa versão, desenvolvida no escopo de um projeto PIBIC (Martinon e Fernandes, 2000), podemos destacar:

- O número de arquivos abertos simultaneamente é teoricamente ilimitado, sendo limitado apenas pelo limite máximo de memória virtual (Memória principal mais área de swap, menos espaço dedicado ao sistema operacional).

- Foi desenvolvida uma estrutura de classes de rotinas e dados de modo a permitir fácil inclusão de novas rotinas de tratamento de dados.

- A maioria das rotinas gráficas do Borland C++ Builder foi reescrita em C++ de forma a otimizar seu desempenho computacional.

Essa versão original do BSSData foi estendida, com a inclusão de novas rotinas e também aperfeiçoada com técnicas de vetorização que serão descritas adiante.

A seguir, serão descritas as ferramentas que estão implementadas na versão atual do software BSSData.

4.2.1 Janela Principal

Ao iniciar o software é apresentada uma janela gráfica (FIGURA 4.2) através da qual o usuário terá acesso às ferramentas que permitirão manipular os conjuntos de dados provenientes do BSS. Esta janela gráfica contém alguns componentes que são comuns a qualquer aplicação Windows, são eles:

1) Barra de comandos - (FIGURA 4.2a) Localizada na parte superior da janela, possui vários menus do tipo pull-down divididos nas categorias: File; Edit; View; Window e Help. Em cada menu existem vários comandos que permitem o acesso às várias ferramentas que compõem o software.

2) Barra de ferramentas - (FIGURA 4.2b) Localizada logo abaixo da barra de comandos, permite a seleção de atividades tais como: criar um novo projeto vazio; adicionar um arquivo de dados ao projeto corrente; salvar o arquivo de dados atual; copiar, recortar e colar texto; organizar as janelas abertas na área de trabalho.

3) Barra de status - (FIGURA 4.2c) Localizada na parte inferior da janela, exibe várias informações ao usuário ao longo da utilização do software. Tais informações podem ser desde mensagens de ajuda até alertas de problemas que porventura possam ocorrer.

4) Área de trabalho - (toda a área cinza escuro na FIGURA 4.2) Por área de trabalho define-se toda a região da janela disponível à abertura de novas janelas e/ou diálogos pertencentes ao software, tais como: "Project Manager", "View", "Profile View", "Zoom" etc. Sendo o software uma aplicação MDI (Multiple Document Interface) temos uma janela principal que contém várias outras janelas (Child Windows), possibilitando desta forma que vários conjuntos de dados possam ser visualizados ao mesmo tempo.

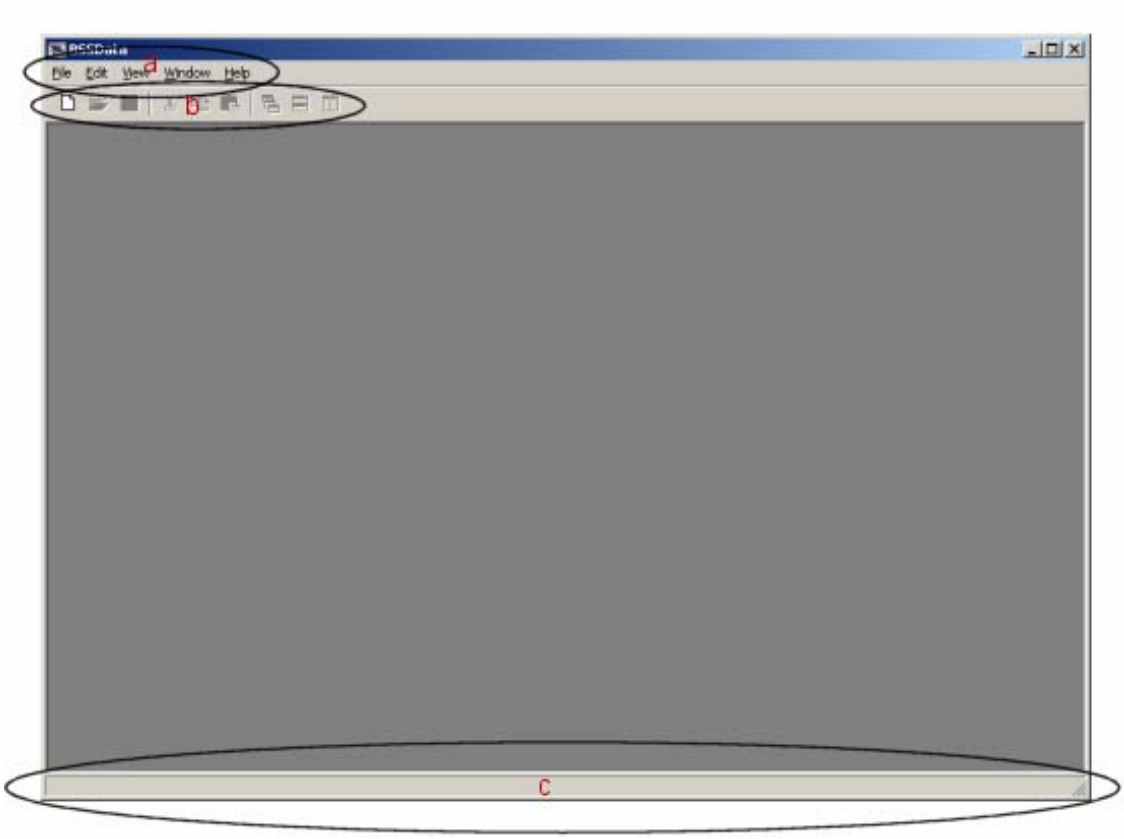

FIGURA 4.2 – Janela Principal do software BSSData

Além destes componentes o software apresenta outras características herdadas do Windows:

- As janelas podem ser minimizadas, maximizadas ou fechadas ao acionar os botões apropriados no canto superior direito ou através do menu de sistema localizado no canto superior esquerdo, acessível ao clicar sobre o ícone da aplicação.

- Os tamanhos das janelas podem ser alterados.

- Caixas de diálogo não podem ser minimizadas, maximizadas ou redimensionadas.

- Descrições sobre os ícones podem ser visualizadas ao passar o mouse sobre eles.

4.2.2 Manipulação de Conjunto de Dados

Através do diálogo "Project Manager" (FIGURA 4.3) os arquivos de dados registrados pelo BSS podem ser manipulados. Este diálogo, acessível pelo menu File comando New Project ou pelo ícone na barra de ferramentas, oferece uma lista de todos os conjuntos de dados abertos na memória. Ao selecionar um conjunto de dados desta lista as informações contidas no cabeçalho do arquivo são apresentadas na caixa de texto situada ao lado direito. Finalmente, este diálogo, possui três botões Add, Remove e View que, respectivamente, adiciona um novo conjunto de dados à lista, remove o conjunto de dados selecionado da lista e visualiza o conjunto de dados selecionado.

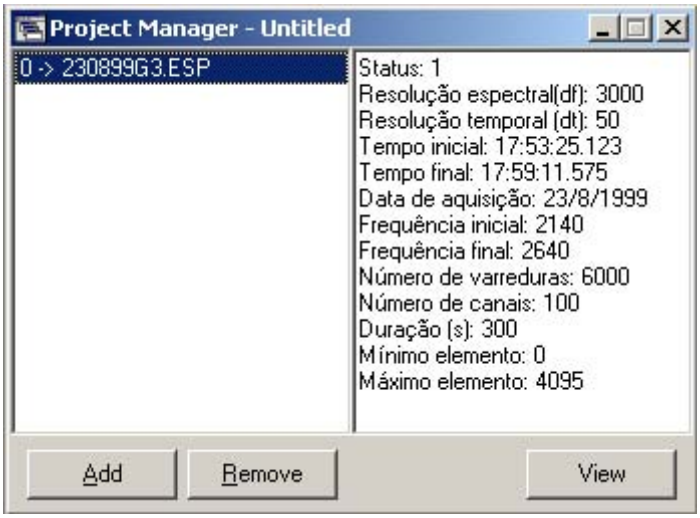

FIGURA 4.3 – Diálogo "Project Manager" usada no gerenciamento dos conjuntos de dados abertos.

4.2.3 Visualização

Após selecionar um arquivo da lista do diálogo "Project Manager" e clicar sobre o botão View será exibida uma janela para a visualização do espectro dinâmico (FIGURA 4.4) referente a este arquivo. Nesta janela podem ser realizados, tanto procedimentos de pós-processamento de dados como análise e determinação visual de informações.

4.2.3.1 Manipulação da Caixa de Seleção

A seleção de uma área do espectro dinâmico é feita através de uma caixa de seleção. Os dados contidos nesta área serão usados em várias ferramentas, tais como remoção de background, plotagem estática e dinâmica de perfis temporal e espectral, zoom, geração de imagens, extração de conjuntos de dados, etc.

Ao clicar sobre qualquer ponto dentro do espectro dinâmico a caixa de seleção tem suas posições alteradas, usando-se o canto superior esquerdo da caixa como base. Seu tamanho pode ser alterado através do menu popup (acessível ao clicar com o botão direito dentro da caixa) clicando-se no item "Resize", um diálogo será, então, aberto.

4.2.3.2 Visualização Dinâmica de Perfis Temporal e Espectral

Esta rotina desenha dinamicamente, através do movimento do mouse, os perfis temporal e espectral (FIGURA 4.4) do conjunto de dados contido em uma área previamente selecionada pelo usuário, auxiliando na determinação visual dos parâmetros das explosões (obtenção dos tempos de subida e descida, da duração total, da taxa de deriva em freqüência, etc.).

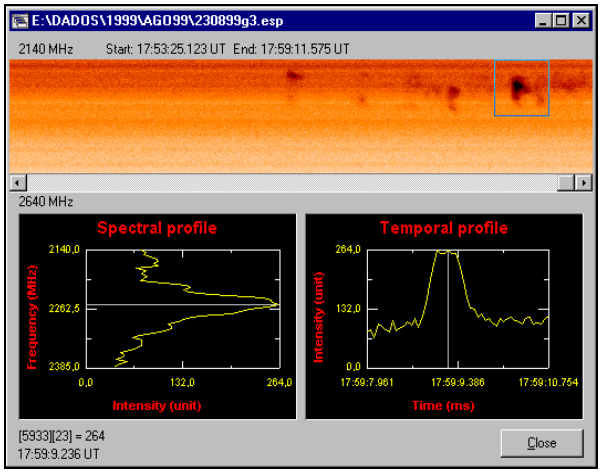

FIGURA 4.4 - Janela para visualização do espectro dinâmico e seus perfis temporais e espectrais.

4.2.3.3 Ferramentas para Filtragem de Dados

O objetivo de realizar uma filtragem nos dados do BSS é minimizar o ruído de fundo e realçar as explosões solares. A versão atual do módulo de filtragem do BSSData possui os seguintes filtros implementados:

- **Rotina para subtrair o background**: Esse background refere-se ao fluxo do sol calmo. Aplicando essa rotina temos como resultado um espectro dinâmico com um fundo mais homogêneo, onde as explosões ficam mais realçadas.

- **Filtro por derivadas**: Esse filtro realça as explosões dando um aspecto de relevo à imagem (pseudo 3D). A principal finalidade desse filtro é ajudar na identificação das fases de subida e descida do sinal (início e final do fenômeno de interesse).

- **Convolução**: possibilita a utilização de filtros lineares passa-baixas e passa-altas, visando a eliminação de ruídos nos dados, através da modificação de sua máscara.

- **Filtro da Mediana**: permite identificar a intensidade média do sinal, podendo-se escolher uma máscara 3x3 ou 5x5.

Filtros Morfológicos: foram implementadas a dilação e a erosão, usando técnicas de morfologia matemática, podendo-se escolher os elementos estruturantes.

Estes filtros estão disponíveis pelo menu popup (acessível ao clicar com o botão direito do mouse em qualquer área do espectro dinâmico exceto dentro da caixa de seleção).

Os filtros implentados foram testados em dados de registros de explosões solares apresentando diferentes tipos de estruturas finas (Martinon et al., 2003). Antes de aplicar os filtros de média, mediana e dilatação os dados foram submetidos a um algoritmo para "remoção" de background. Este algoritmo funciona determinando uma amostra dos dados que possa ser representativa do sinal de background, segue-se então o cálculo da média dos valores de cada canal da amostra. Por fim os valores médios de cada canal são subtraídos dos dados originais. Com isso, obtemos um background

homogêneo em todas as freqüências. Com o background homogêneo aplicamos os outros filtros com o intuito de melhorar ainda mais o contraste entre as explosões e o background. Nas FIGURA 4.5, 4.6 e FIGURA 4.6 apresentamos o resultado da aplicação dos filtros de média, mediana e dilação em 6 tipos explosões solares.

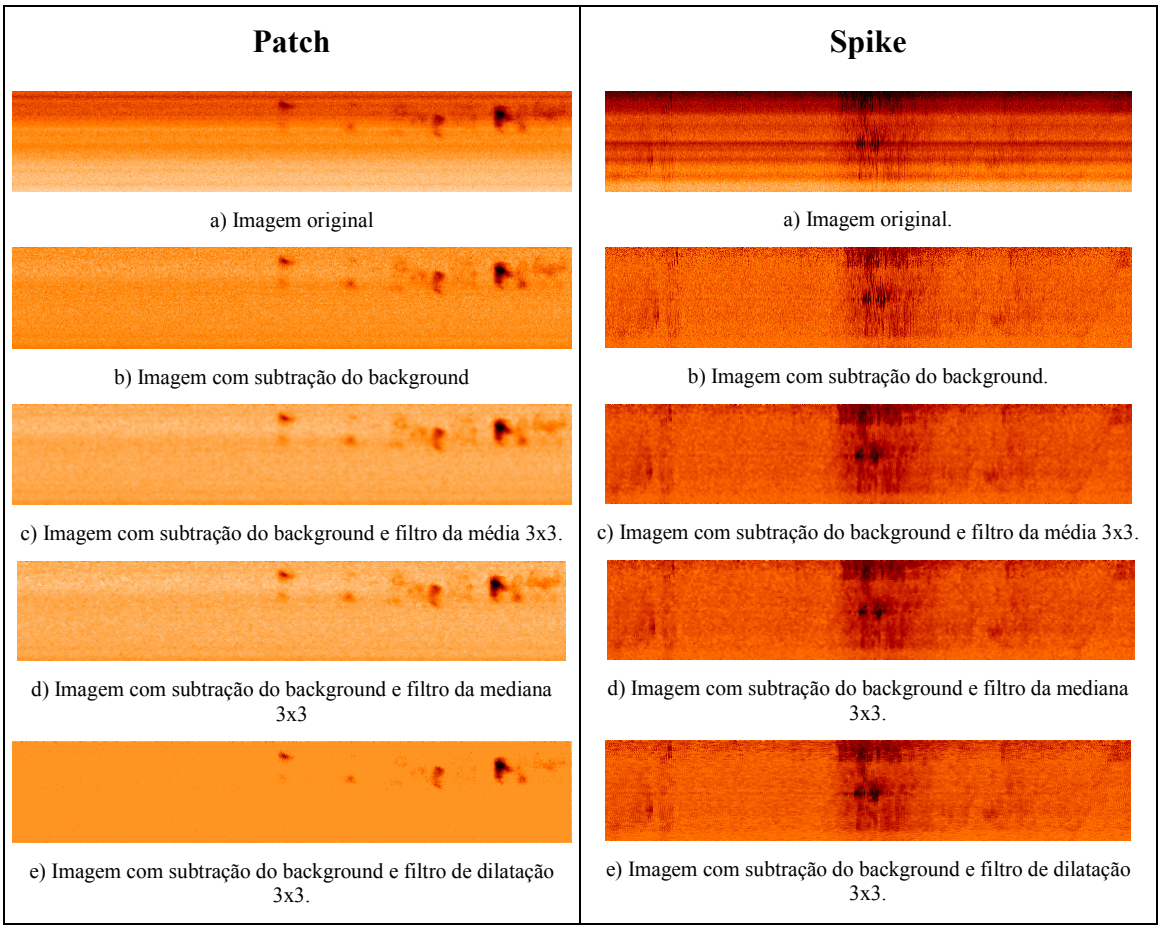

FIGURA 4.5 – Aplicação de rotinas para filtragem de ruído nos tipos de explosões patch e spike.

O algoritmo de subtração de background se mostrou bastente eficiente para homogeneização do ruído de fundo em toda a banda de freqüência, sendo apenas bastante dependente da escolha da amostra. Para a maioria das estruturas testadas os resultados obtidos com os filtros da Média, Mediana e Dilatação são comparáveis, quanto ao destaque das estruturas, no entanto o filtro da Dilatação foi o que apresentou melhor resultado na homogeneização do background. Para as explosões apresentando estruturas finas com duração muito curta (tipo III) e banda muito estreita em freqüência

(dots, zebra), nestes casos, a Dilatação descaracteriza as estruturas. Nos casos testados obteve-se uma diminuição e homogeneização significativa do ruído de fundo, resultando num aumento do contraste e definição das regiões do espectro caracterizando explosões solares, possibilitando a determinação mais precisa dos parâmetros espectro-temporais de cada explosão

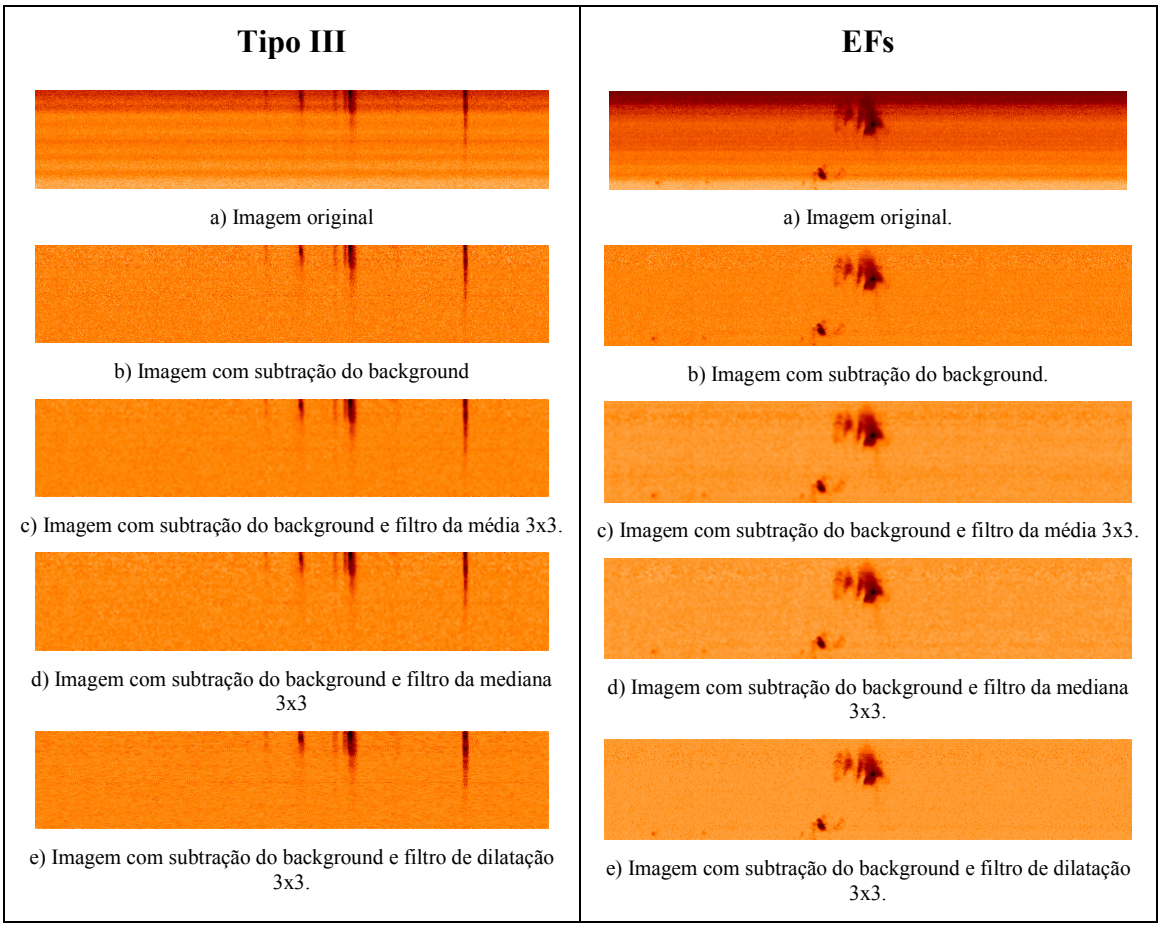

FIGURA 4.6- Aplicação de rotinas para filtragem de ruído nos tipos de explosões tipo III e EFs.

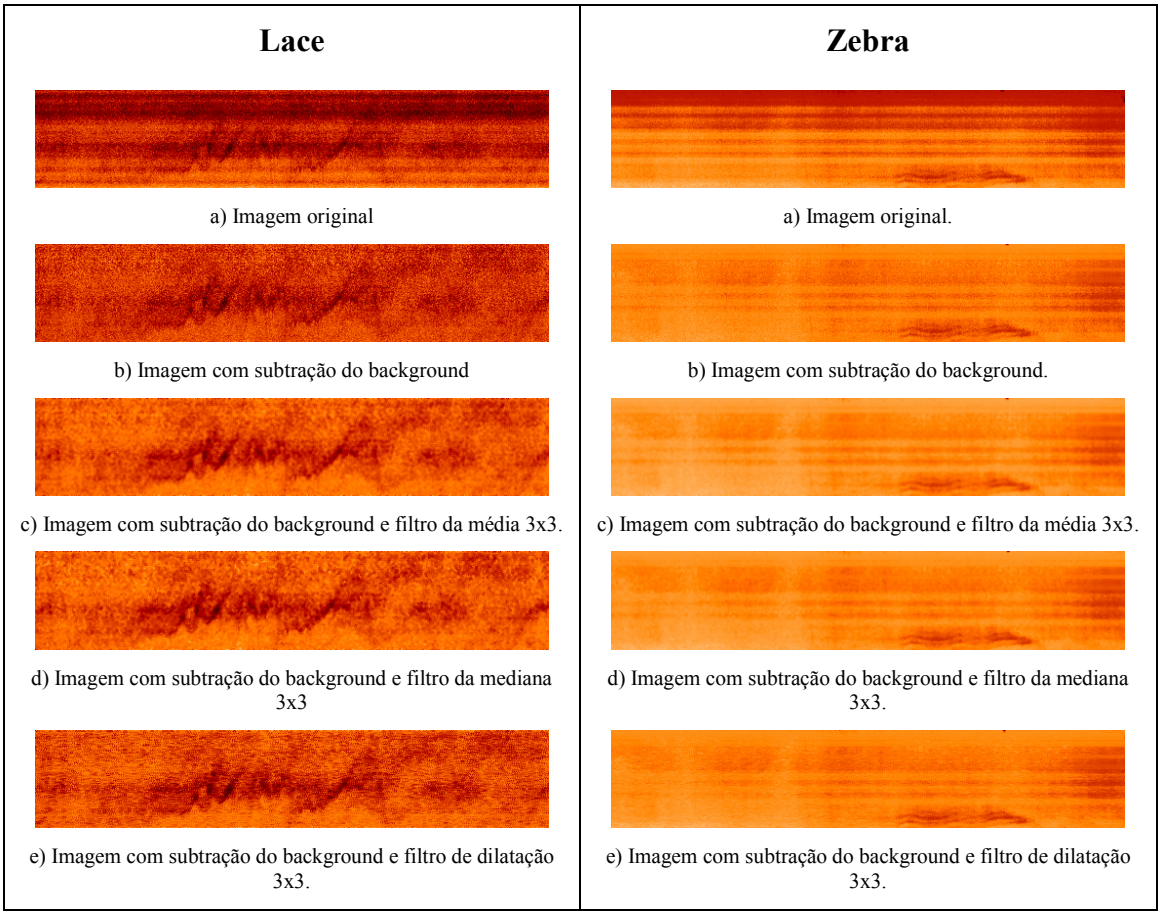

FIGURA 4.6 - Aplicação de rotinas para filtragem de ruído nos tipos de explosões lace e zebra.

4.2.4 Cores

No sistema operacional Windows, dependendo do adaptador de vídeo em uso, pode-se exibir as seguintes quantidades de cores:

TABELA 4.1 - Número de cores x tamanho lógico do pixel.

Número de cores	Tamanho de cada pixel
16 cores	4 bits
256 cores	8 bits
High Color (65.536 cores)	16 bits
True Color (16.777.216 cores)	24 bits

No software BSSData adotou-se como padrão uma imagem de 8 bits por pixel, sendo representada por uma paleta de 236 cores, pois no modo de 256 cores, 20 cores são reservadas para o sistema operacional. Assim, podemos executá-lo em todos os modos de vídeo maiores ou iguais a 8 bits por pixel.

As cores, em qualquer modo de vídeo, são formadas por uma combinação das cores vermelho, verde e azul. E cada combinação dessas três cores é representada por um índice entre 0 e 255 na paleta (sendo que de 0 a 9 e de 246 a 255 são reservados ao sistema operacional).

A janela "Palette Manager" (FIGURA 4.7) nos oferece opções para gerar paletas com gradientes de cores (combinação das cores vermelha, verde e azul), esticar o início ou o final da paleta, inverter a paleta ou escolher uma paleta já definida.

Existem cerca de 41 paletas de cores pré-definidas, mas este número pode ser estendido criando um arquivo de definição de cores. Este arquivo é composto de 256 linhas, sendo cada uma composta de 3 colunas que representam respectivamente a quantidade das cores vermelho, verde e azul que irão compor a cor desejada.

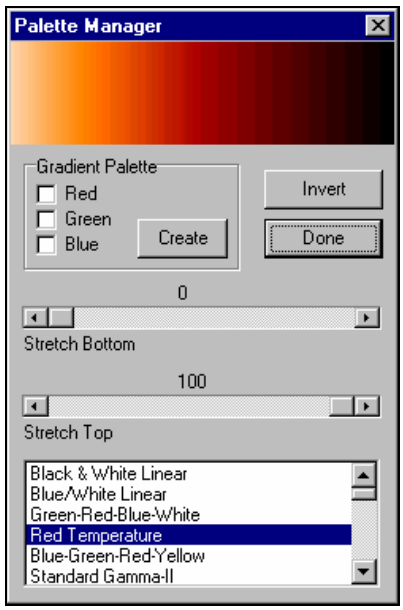

FIGURA 4.7 - Janela para manipulação da paleta de cores.

4.3 Otimização de Desempenho por Vetorização

Para otimizar as rotinas já implementadas, e na implementação de novas rotinas, no software BSSData, utilizaremos dois conjuntos de instruções presentes na maioria dos processadores atuais. Estes conjuntos de instruções utilizam técnicas SIMD (Single Instruction Multiple Data), ou seja, uma única instrução é usada para executar a mesma operação em mais de uma variável.

Para otimizar as rotinas que envolvam operações somente com números inteiros, utilizaremos o conjunto de instruções MMX (Intel, 2002, a, b, c). Já as que envolvem operações em ponto flutuante, serão otimizadas através do conjunto de instruções 3DNow! (AMD, 1999; AMD 2000, a, b; AMD 2002) (presente nos processadores Athlon e Duron da AMD).

Algumas rotinas do software BSSData já foram otimizadas utilizando as instruções MMX (Martinon et al., 2002a). A seguir mostraremos os resultados dos testes realizados em relação ao tempo de processamento das rotinas.

Nestes testes, confrontamos rotinas escritas em C com rotinas similares em linguagem assembly utilizando instruções MMX. Esperava-se uma diminuição substancial no tempo de processamento, devido à utilização destas instruções.

Os testes foram conduzidos em um microcomputador padrão IBM/PC com processador AMD Athlon 1.2 Ghz, com 256 Mb de memória RAM, e tendo como sistema operacional o Linux (distribuição Conectiva 7.0, kernel 2.4.5).

Foram utilizados os compiladores GCC 2.95.3 e NASM 0.98 (compilador assembly). O programa foi executado em prioridade máxima, de forma a obter-se um valor preciso do tempo de processamento, valor este obtido através da instrução RDTSC (linguagem assembly).

Foi utilizado um vetor aleatório de 600.000 bytes (6000 x 100). Este tamanho foi escolhido pois reflete a realidade dos dados com os quais o software BSSData deverá

trabalhar em regime normal. Exceto para a rotina de transposição foi utilizado um vetor aleatório de 640.000 bytes (800x800) por causa das limitações da rotina.

Comparou-se o desempenho das rotinas em C, contra rotinas utilizando MMX com o vetor alinhado (MMXa) e não alinhado (MMX). O alinhamento do vetor é uma tarefa realizada automaticamente nos programas em linguagem C, mas foi necessário realizar este procedimento manualmente no caso do MMX, pois caso contrário existe uma perda de desempenho, como pode ser visto na Tabela 4.2. Os valores absolutos dos testes são exibidos na FIGURA 4.8. Os números em parênteses foram normalizados utilizando os valores das rotinas em C como base.

TABELA 4.2 - Comparação de desempenho (valores em quilociclos do processador).

Rotina	**C**	**MMX**	**MMXa**
Contraste	8.357 (1 x)	2.716 (3.08 x)	2.584 (3.23 x)
Brilho	6.439 (1 x)	2.663 (2.42 x)	2.439 (2.64 x)
MinMax (byte)	5.942 (1 x)	1.340 (4.43 x)	1.205 (4.93 x)
MinMax (word)	7.100 (1 x)	2.665 (2.66 x)	2.420 (2.93 x)
Zoom 2x	7.804 (1 x)	2.778 (2.81 x)	2.544 (3.07 x)
Zoom 4x	7.267 (1 x)	2.485 (2.92 x)	2.300 (3.16 x)
Transposta	21.397 (1 x)	2.553 (8.38 x)	2.288 (9.35 x)

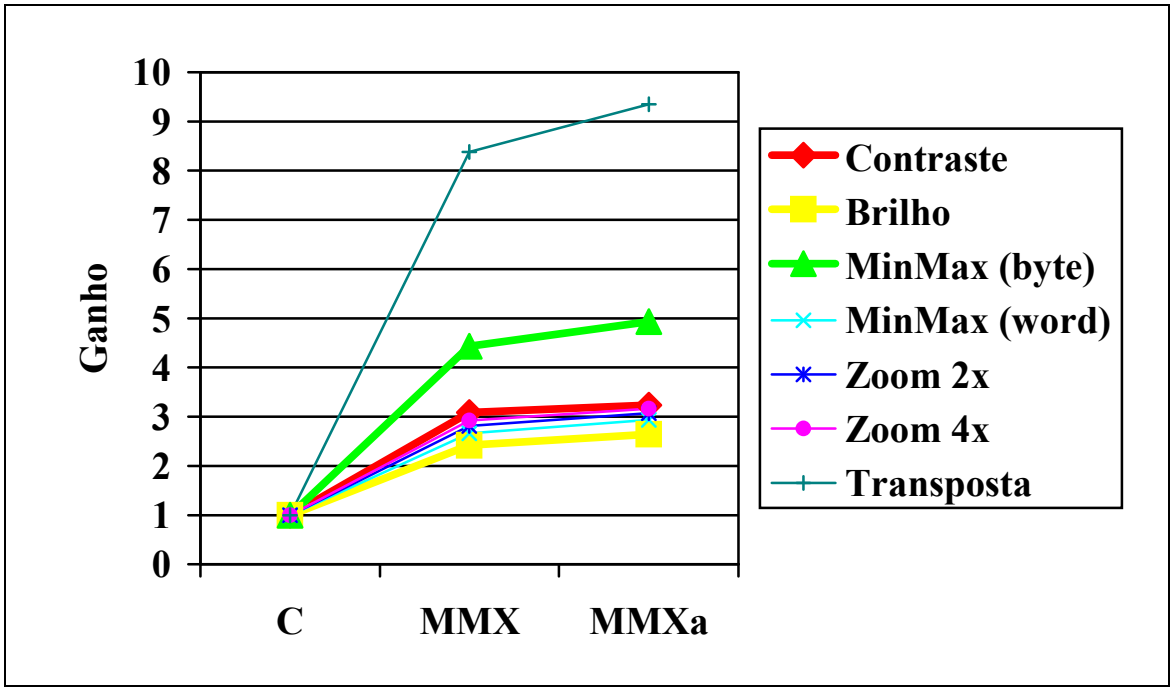

FIGURA 4.8 - Gráfico comparativo de desempenho.

A seguir esses dois conjuntos de instruções, MMX e 3DNow!, serão detalhados.

4.3.1 Instruções MMX

A tecnologia MMX foi projetada para acelerar aplicações multimídia e de comunicações através de novas instruções e tipos de dados que permitem um novo nível de desempenho. Ela explora o paralelismo inerente de muitos algoritmos de multimídia e comunicações, mantendo a compatibilidade com as aplicações e sistemas operacionais existentes. Foram introduzidas pela fabricante de processadores Intel, em sua linha de processadores, pela primeira vez no Pentium MMX (família x86).

As instruções MMX atendem algoritmos com as seguintes características:

- Tipos de dados inteiros pequenos (8-bit e 16-bit).
- Loops muito repetitivos e pequenos.
- Freqüentes multiplicações e adições.

- Algoritmos de processamento intensivo.
- Operações altamente paralelas.

Podemos dizer então que o MMX é um conjunto de instruções para tipo de dados inteiros de propósito geral que podem ser aplicadas a uma grande variedade de aplicações multimídia e de comunicações. Os destaques da tecnologia são:

- Técnica SIMD (Single Instruction, Multiple Data).
- 57 novas instruções.
- 8 registradores MMX de 64-bit.
- 4 novos tipos de dados.

Os 4 novos tipos de dados são:

- Packed byte – 8 elementos de 8-bit.
- Packed word – 4 elementos de 16-bit.
- Packed doubleword – 2 elementos de 32-bit.
- Quadword – 1 elemento de 64-bit.

Por exemplo, pixels são geralmente representados por inteiros de 8-bit. Com a tecnologia MMX, oito desses pixels podem ser agrupados em uma única quantidade de 64-bit e movidos para um registrador MMX, quando uma instrução MMX é executada os oito valores dos pixels são pegos imediatamente do registrador MMX, as operações aritméticas ou lógicas são realizadas em paralelo sobre todos os oito elementos e o resultado é escrito em um registrador MMX. O grau de paralelismo que pode ser alcançado com a tecnologia MMX depende do tamanho dos dados, assim provê um grau de paralelismo 8, usando dados de 8-bit, e 1, ou seja sem paralelismo, usando dados de 64-bit.

A tecnologia MMX é integrada à arquitetura x86 de modo que mantenha completa compatibilidade com os sistemas operacionais existentes. Isto é obtido utilizando os registradores de ponto flutuante, existentes, como registradores MMX. Por essa razão, não foram criados novos registradores para conceder a tecnologia MMX, desta forma os sistemas operacionais usam os mecanismos padrões para interagir com o estado de ponto flutuante para salvar e restaurar código MMX: instruções de ponto flutuante que salvam/restauram o estado de ponto flutuante também manipulam o estado MMX. Aplicações podem executar, ambas, rotinas MMX e rotinas de ponto flutuante, mas não podem ser intercaladas. A tecnologia MMX não introduz nenhuma exceção nova ou informações de estado, assim os sistemas operacionais correntes podem rodar aplicações que utilizam instruções MMX.

As instruções MMX cobrem várias áreas funcionais incluindo:

- Operações aritméticas básicas, tais como soma, subtração, multiplicação, deslocamento aritmético e multiplicação-soma.
- Operações de comparação.
- Instruções de conversão, para converter entre os novos tipos de dados: agrupar e desagrupar dados de tamanhos menores para maiores.
- Operações lógicas, tais como AND, AND NOT, OR e XOR.
- Operações de deslocamento.
- Instruções de transferência de dados, para transferência de registrador-registrador MMX ou load/store de 64-bit e 32-bit para a memória.
- Instrução para gerenciamento de estado (EMMS), para manipular transições de MMX para ponto flutuante.

As instruções aritméticas, de comparação e deslocamento são projetadas para suportar os diferentes tipos de dados da tecnologia MMX: essas instruções têm um opcode diferente para cada tipo de dado suportado.

Como resultado, as instruções da tecnologia MMX são implementadas com 57 opcodes.

Todas as instruções MMX, exceto a instrução EMMS, referem-se e operam sobre dois operandos: o operando de origem e o de destino. O primeiro operando é o destino e o segundo é a origem. A instrução sobrescreve o operando de destino com o resultado da instrução.

Por exemplo:

OPERAÇÃO DESTINO, ORIGEM

Será decodificado como:

DESTINO = DESTINO OPERAÇÃO ORIGEM

Uma instrução MMX, típica, tem a seguinte sintaxe:

- Prefixo: **P** para packed.
- Operação da instrução: por exemplo: **ADD**, **CMP** ou **XOR**.
- Sufixo:
  - **US** para saturação sem sinal.
  - **S** para saturação com sinal.
- **B, W, D ou Q** para o tipo de dado: packed byte, packed word, packed doubleword ou packed quadword.

Por exemplo, **PADDSB** é uma instrução MMX (**P**) que soma (**ADD**) os 8 bytes (**B**) dos operandos de origem e de destino e satura o resultado levando em consideração o sinal (**S**).

A seguir mostraremos o código-fonte de um exemplo simples, somar dois vetores de 8 elementos através de uma única instrução MMX.

Abaixo temos o código em C que faz a chamada da função, escrita em assembly, que utiliza as instruções MMX.

```
#include <stdio.h>
#include <stdlib.h>
extern void soma (unsigned char *source, unsigned char *dest);
int main(void)
{
   unsigned char x[8] = { 100,  50, 200, 110,  20,  30,  60,  10};
   unsigned char y[8] = {  50,  50,  50,  50,  50,  50,  50,  50};
   soma(y, x);

   printf("Resultado: {%d, %d, %d, %d, %d, %d, %d, %d}\n", x[0], x[1], x[2], x[3], x[4],
          x[5], x[6], x[7]);
   return 1;
}
```

Agora, temos o código, em linguagem assembly, da função que soma os dois vetores.

```
bits 32                    ; Modo protegido de 32 bit.
      section .text              ; Inicia o segmento de codigo.
      global soma                ; Exporta o rotulo soma.
            align  16
      soma:
            push   ebp           ; Deve-se presevar o conteudo original do reg EBP.
            mov    ebp, esp      ; Inicializa EBP para acessar os parametros.
            mov    eax, [ebp+8]  ; EAX = ponteiro para o vetor de origem.
            movq   mm0, [eax]    ; Carrega em MM0 o conteudo do vetor de origem.
            mov    edx, [ebp+12] ; EDX = ponteiro para o vetor de destino.
            movq   mm1, [edx]    ; Carrega em MM1 o conteudo do vetor de destino.
            paddusb mm1, mm0     ; Soma em MM1 8 bytes com unsigned saturation.
            movq   [edx], mm1    ; Copia de volta para a memoria o resulto.
      emms                       ; Encerra o estado MMX.
      pop    ebp                 ; Restaura o registrador EBP original.
      ret
```

Para compilar esse código utilizaram-se os compiladores GCC e NASM (compilador assembly) ambos presentes em todas as distribuições atuais do sistema operacional Linux. Para facilitar o processo de compilação foi criado um arquivo Makefile, mostrado a seguir:

```
simple : simple.o mmx_simple.o
	gcc -o simple simple.o mmx_simple.o
simple.o : simple.c
	gcc -c simple.c
mmx_simple.o : mmx_simple.asm
	nasm -f elf mmx_simple.asm
```

Após compilar o programa, através do comando make, podemos executá-lo tendo como resultado a soma dos dois vetores. Veja o resultado abaixo:

```
[martinon@master mmx]$ ./simple
Resultado: {150, 100, 250, 160, 70, 80, 110, 60}
[martinon@master mmx]$
```

4.3.2 Instruções 3DNow!

O conjunto de instruções 3DNow! foi desenvolvido pela Advanced Micro Devices (AMD) para seu processador K6-2. Em parte, essas instruções foram projetadas em substituição a unidade de ponto-flutuante que o K6-2 possuía. Já que esta unidade não utilizava arquitetura em pipeline.

O 3DNow! pode vir implementado em dois conjuntos de instruções distintos (somente nos processadores Athlon e Duron). O Enhanced 3DNow! e o 3DNow! original que vinha nos processadores K6-2. O conjunto de instruções Enhanced 3DNow! possui mais cinco instruções, e um conjunto de instruções MMX mais otimizado (Uma expansão que a Intel Corporation incluiu no processador Pentium III). O 3DNow! trabalha somente com ponto-flutuante de precisão simples.

Além das vantagens da técnica SIMD, o 3DNow! ainda oferece várias vantagens sobre o uso de instruções de ponto-flutuante normais, x87. São elas:

- Modo de acesso aos registradores que previne a necessidade de instruções FXCH para manipular a pilha de operandos.
- Vantagem na performance, na família de processadores Athlon, Duron e K6-2, sobre x87.
- Operações de divisão extremamente rápidas com precisão razoável.
- Operações de raiz quadrada extremamente rápidas com precisão razoável.

Uma desvantagem seria não ter funções equivalentes em 3DNow! das instruções matemáticas, científicas, presentes no x87.

As instruções 3DNow! da AMD quase poderiam ser chamadas de uma extensão das instruções MMX. Já que elas usam os registradores MMX, só que ao invés de considerar os registradores MMX como valores inteiros, consideram-se como valores de ponto-flutuante. Desta forma, um registrador MMX pode armazenar dois valores de ponto-flutuante de 32 bits.

A seguir mostraremos o código-fonte de um exemplo simples, somar dois vetores de 2 elementos através de uma única instrução 3DNow.

Abaixo temos o código em C que faz a chamada da função, escrita em assembly, que utiliza as instruções 3DNow.

```
#include <stdio.h>
#include <stdlib.h>
extern void soma (float *source, float *dest);
int main(void)
{
   float x[2] = { 3.5,  5.2};
   float y[2] = { 0.5,  4.8};
   soma(y, x);
   printf("Resultado: {%f, %f}\n", x[0], x[1]);
   return 1;
}
```

Agora, temos o código, em linguagem assembly, da função que soma os dois vetores.

```
    bits 32                    ; Modo protegido de 32 bit.
    section .text              ; Inicia o segmento de codigo.
    global soma                ; Exporta o rotulo soma.
        align  16
    soma:
        push   ebp             ; Deve-se presevar o conteudo original do reg EBP.
        mov    ebp, esp        ; Inicializa EBP para acessar os parametros.
        mov    eax, [ebp+8]    ; EAX = ponteiro para o vetor de origem.
        movq   mm0, [eax]      ; Carrega em MM0 o conteudo do vetor de origem.
        mov    edx, [ebp+12]   ; EDX = ponteiro para o vetor de destino.
        movq   mm1, [edx]      ; Carrega em MM1 o conteudo do vetor de destino.
        pfadd  mm1, mm0        ; Soma em MM1 os valores de mm0.
        movq   [edx], mm1      ; Copia de volta para a memoria o resulto.

    femms                      ; Encerra o estado 3DNow!.
    pop    ebp                 ; Restaura o registrador EBP original.
    ret                        ; Fim da rotina.
```

Para compilar esse código foram utilizados os compiladores GCC e NASM (compilador assembly) ambos presentes em todas as distribuições atuais do sistema operacional

Linux. Para facilitar o processo de compilação foi criado um arquivo Makefile, mostrado a seguir:

```
simple : simple.o 3dnow_simple.o
	gcc -o simple simple.o 3dnow_simple.o

simple.o : simple.c
	gcc -c simple.c

3dnow_simple.o : 3dnow_simple.asm
	nasm -f elf 3dnow_simple.asm
```

Após compilar o programa, através do comando make, podemos executá-lo tendo como resultado a soma dos dois vetores. Veja o resultado abaixo:

```
[martinon@master 3dnow]$ ./simple
Resultado: {4.000000, 10.000000}
[martinon@master 3dnow]$
```

4.4 Descrição das Classes

A versão atual do BSSData foi desenvolvida utilizando o conceito de programação orientada à objetos, deste modo algumas classes foram desenvolvidas para compor sua estrutura lógica. Tais classes cobrem tarefas que vão desde gerenciar o espaço em memória reservado aos datasets em uso até a exibição de eixos graduados em saídas gráficas. A seguir serão descritas as funcionalidades destas classes. E uma descrição detalhada de suas propriedades e métodos privados, protegidos e públicos pode ser vista no Apêndice A.

4.4.1 cDataset

Desenvolvida para possibilitar o manuseio de um dataset. A classe tem como membros de dados (propriedades) o cabeçalho e a matriz de dados constantes no arquivo de dados associado a um objeto instanciado da classe cDataset.

As várias funções-membro (métodos) permitem realizar tarefas de filtragem, de cálculos estatísticos e de manipulação da matriz de dados.

Qualquer ferramenta que necessite utilizar e alterar as informações de um dataset devem ser desenvolvidas nesta classe.

4.4.2 cDatasets

Auxilia no gerenciamento dos vários datasets abertos na memória principal pelo usuário através do BSSData. Possui métodos para adicionar ou remover um dataset da lista de datasets, gravar as modificações em disco e recuperar informações do dataset correntemente em uso.

Qualquer nova ferramenta que venha a ser necessária ao gerenciamento de um conjunto de datasets deverá ser implementada nesta classe.

4.4.3 cPalette

Classe desenvolvida com o intuito de facilitar o uso de paletas de cores pelo software. Com ela podemos abrir arquivos de paletas de cores pré-definidas que serão usadas para representar visualmente o valor de intensidade dos elementos da matriz de dados do dataset.

4.4.4 cScaledCoord

Essa classe foi desenvolvida para mapear as coordenadas da tela (em pixels) em um sistema de coordenadas cartesianas, possibilitando a visualização de gráficos com um campo de visão limitado à uma área (no sistema de coordenadas cartesianas) definida através do método cScaledCoord::SetViewPort.

Assim para se desenhar um ponto na tela, por exemplo, passa-se o valor de suas coordenadas cartesianas x e y para os métodos cScaledCoord::RealX e cScaledCoord::RealY, respectivamente, e estes devolvem as coordenadas correspondentes em pixels.

4.4.5 cViewGraph

Essa classe foi desenvolvida com o intuito de se desenhar uma moldura com eixos graduados, título, subtítulos, linhas de grade etc, em uma janela gráfica. Através dessa classe pode-se, alterar as cores de todos os elementos da moldura bem como a fonte (tipo, tamanho, cor etc) de todos os textos exibidos na mesma. Além de ajustar a graduação dos eixos quando se deseja corrigir o aspecto da imagem.

Outra característica importante é que quando a janela, que contém a moldura, é redimensionada pelo usuário, a moldura se ajusta automaticamente ao novo tamanho.

4.4.6 cVGBoxedInside

Essa é uma classe derivada de cViewGraph. Portanto, herda todas as suas propriedades, exceto as privadas, e métodos, exceto os privados. Ela desenha uma moldura em forma de caixa com as marcações dentro da área de plotagem.

4.4.7 cVGBoxedOutside

Essa é uma classe derivada de cViewGraph. Portanto, herda todas as suas propriedades, exceto as privadas, e métodos, exceto os privados. Ela desenha uma moldura em forma de caixa com as marcações fora da área de plotagem, ou seja, na margem da moldura.

4.4.8 cVGCartesian

Essa é uma classe derivada de cViewGraph. Portanto, herda todas as suas propriedades, exceto as privadas, e métodos, exceto os privados. Ela desenha uma moldura com eixos coordenados centrados na área de plotagem.

4.4.9 cVGTimeBoxedInside

Essa é uma classe derivada de cViewGraph. Portanto, herda todas as suas propriedades, exceto as privadas, e métodos, exceto os privados. Ela desenha uma moldura em forma de caixa com as marcações dentro da área de plotagem sendo o eixo das abscissas

graduado com valores temporais. Essa escala é definida através de um tempo inicial e a resolução temporal do dataset.

4.5 Rotinas de Visualização do BSSData

Nesta Seção iremos tratar especificamente das rotinas de visualização incorporadas ao BSSData. Tais rotinas foram aperfeiçoadas ao longo do desenvolvimento do BSSData e futuramente deverá receber mais melhorias com a adoção da biblioteca OpenGL.

A seguir introduziremos alguns conceitos de visualização científica, logo após serão abordadas as estratégias de implementação usadas no desenvolvimento do BSSData. Nas seções seguintes teremos uma breve descrição da biblioteca OpenGL e do software de visualização OpenDX, que futuramente deverão fazer parte do sistema otimizado para tratamento e análise de dados do BSS. Esta biblioteca (OpenGL) e este software de visualização (OpenDX) trazem uma ótima vantagem com sua adoção, são open-source, ou seja não possuem o código fechado, proprietário. Eles podem substituir perfeitamente o IDL (Interactive Data Language), um software proprietário, caro, com um desempenho computacional inferior, e que impõe muitas limitações para o desenvolvimento de uma interface com o usuário mais amigável.

4.5.1 Visualização Científica

A visualização desempenha um papel importante no entendimento de grandes quantidades de dados que são produzidos nas simulações computacionais de sistemas reais. É extremamente difícil para ler e entender um arquivo mesmo que tenha apenas alguns Mbytes de tamanho. Como o volume de dados aumenta, uma representação visual dos dados torna-se rapidamente o método mais prático para se entendê-los.

A idéia de representar dados visualmente remonta a tempos bem anteriores à visualização baseada em computadores. A ligação entre a propagação do cólera e uma bomba de água fornece um exemplo do uso da visualização para a análise de um problema. Entre 1853 e 1854 houve uma epidemia de cólera em Londres, e ao marcar

num mapa todos os casos de cólera que já haviam ocorrido o Dr. John Snow pôde constatar que todas as vítimas moravam em torno de uma bomba de água em particular. Assim, com esta simples representação dos dados, pôde-se descobrir que aquela bomba de água estava contaminada.

Com o crescente desenvolvimento da área de visualização científica (VC), o processo de visualizar deixou de ser uma simples ferramenta para apresentar resultados para se tornar uma importante ferramenta que proporciona o entendimento dos dados. Com isso, surgiram diversos modelos de VC. Nesta seção nos deteremos a dois modelos mais comumente usados: o modelo cíclico de visualização proposto por Upson, et al. (1989) e o modelo pipeline proposto por Habber e McNabb (1990).

No **modelo cíclico** de visualização o processo de visualização dos dados é composto de uma seqüência cíclica de passos (FIGURA 4.9) que serão repetidos até que o resultado obtido esteja de acordo com o que o usuário deseja. A simulação numérica de um processo natural qualquer produz um conjunto de dados que devem ser analisados, podendo assim, revelar falhas tanto no software quanto na teoria a partir da qual este foi derivado. A tarefa de análise dos dados também pode ser vista como um ciclo de passos, repetidos até que todas as questões estejam resolvidas, neste processo a visualização dos dados torna-se uma importante ferramenta. Assim, a tarefa de análise pode ser quebrada nos seguintes passos:

- Os dados brutos ou primários são filtrados e tratados. Diminuindo o volume de dados.
- Estes dados filtrados e/ou tratados são mapeados para uma estrutura abstrata.
- A estrutura do passo anterior é sintetizada em imagens e exibidas ao usuário.

E estes passos podem ser repetidos até que a imagem corresponda ao esperado. A FIGURA 4.9 ilustra o modelo cíclico de visualização onde os retângulos representam objetos e as elipses, operações sobre estes.

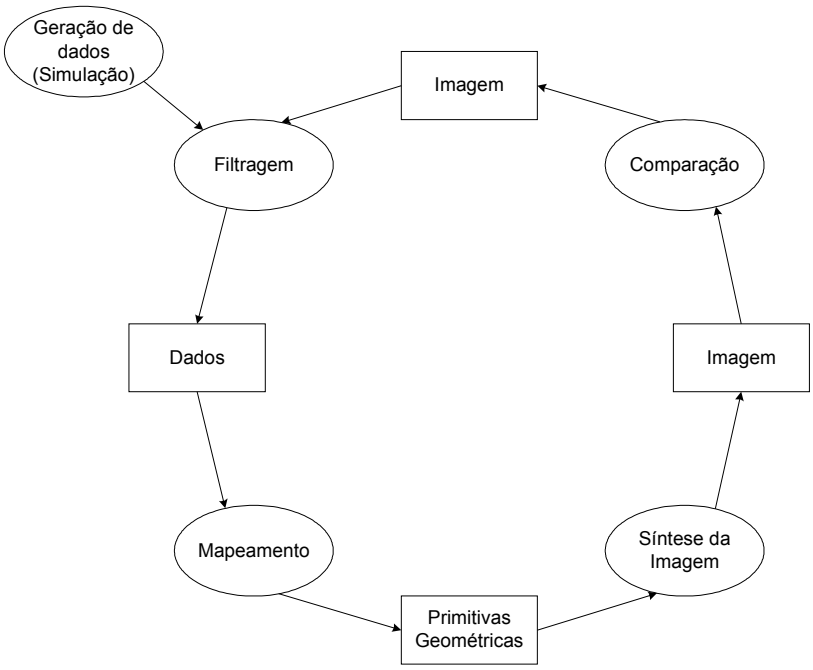

FIGURA 4.9 – O modelo cíclico de visualização.

O **modelo pipeline** de visualização, proposto por Habber e McNabb, fornece uma abordagem mais detalhada do processo de visualização, sendo também aplicado na geração de visualizações de dados científicos, obtidos por simulações ou observações científicas (Habber e McNabb, 1990). O processo de visualização é dividido em uma série de transformações, que quando executadas mapeiam os dados originais em uma imagem, formando assim um pipeline de visualização. Este processo é dividido em três etapas básicas, tratamento dos dados, geração do objeto abstrato de visualização (OAV) e síntese da imagem (FIGURA 4.10).

- Tratamento dos dados: abrange a transposição dos dados brutos para um formato conveniente, a filtragem desses dados formatados e subseqüentes operações de transformação de dados, tais como normalizações, calibrações e interpolações.

- Geração do OAV: esta etapa é responsável pelo mapeamento dos dados já tratados para um objeto chamado "Objeto Abstrato de Visualização", o qual servirá de base para a síntese da imagem. As características dos dados são

mapeadas para atributos que definem o OAV, tais como a geometria, cor, textura, etc.

- Síntese da imagem (Rendering): os atributos do OAV servem de base para a síntese da imagem a ser visualizada.

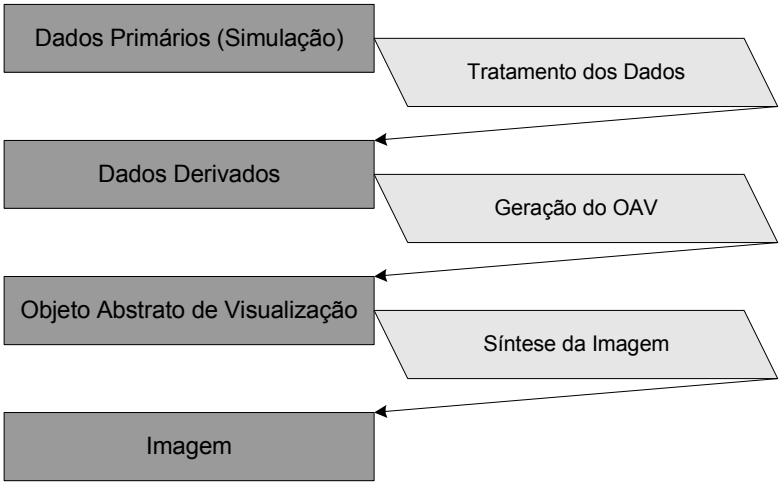

FIGURA 4.10 – O modelo pipeline de visualização.

4.5.2 Estratégias de Implementação

A estratégia adotada, para a implementação das rotinas de visualização do BSSData, foi a de desenvolver rotinas próprias aproveitando os recursos da GDI (Graphics Device Interface) do Windows e das rotinas da VCL (Visual Components Library) desenvolvida pela Borland e integrada ao Borland C++ Builder, além do uso de vetorização (ver mais detalhes em 0) em partes críticas do código. Futuramente deverá ser utilizada a biblioteca gráfica OpenGL tanto na geração de gráficos 2D como 3D, pois atualmente praticamente todas as placas de vídeo possuem a OpenGL incorporada ao seu chipset tornado, assim, o desempenho computacional bem superior.

Em todas saídas gráficas foi adotada a técnica de double buffering, que consiste em montar a imagem primeiramente na memória e depois exibí-la ao usuário. Esta técnica elimina o efeito de flickering quando o usuário usa uma barra de rolagem para "rolar" a imagem e quando temos gráficos dinâmicos, que vão sendo desenhados com a movimentação do mouse sobre o espectro dinâmico.

As imagens são montadas por partes, primeiro uma moldura gráfica é desenhada, contendo eixos graduados, legendas, títulos etc. Depois, as coordenadas são mapeadas para que o gráfico seja exibido dentro da área útil da moldura. Logo após, as rotinas que efetuam os cálculos e geram os gráficos são chamadas. Por fim, com imagem pronta na memória, copiamos para um device context (DC) com uma função específica da GDI (BitBlt).

Para montarmos o espectro dinâmico o processo é bem parecido, a única diferença é que temos que adequar a matriz de dados às 236 cores disponíveis na paleta de cores. Assim é feito um mapeamento, que transforma os valores de uma escala de 12-bit de precisão (0 à 4095) para uma escala de 8-bit de precisão (256 cores, menos as 20 cores que são reservadas ao sistema). A matriz resultante é utilizada na montagem de um bitmap na memória, juntamente com a paleta de cores escolhida pelo usuário. Feito isto a imagem está pronta para ser exibida ao usuário.

Como já foi dito anteriormente, mudaremos esta estratégia. Iremos adotar uma biblioteca específica para visualização gráfica (OpenGL) e utilizar o software OpenDX para testar visualização antes de implementá-las no BSSData. Objetivando um desenvolvimento mais rápido e eficiente em termos computacionais e de recuros (pessoas, máquinas etc).

A seguir descreveremos brevemente a biblioteca gráfica OpenGL e o software de visulização OpenDX.

4.5.3 Biblioteca Gráfica OpenGL

A OpenGL é uma biblioteca para modelagem e gráficos 2D e 3D que além de ser muito rápida é extremamente portável. Através dela podemos criar gráficos em 3D com uma qualidade visual próxima de um software de ray-tracing, mas com a vantagem de os gráficos serem gerados, em termos de tempo computacional, muito mais rápidos com a OpenGL. Ela usa algoritmos desenvolvidos e otimizados pela Silicon Graphics, Inc. (SGI).

A OpenGL foi projetada para ser utilizada em conjunto com hardware especialmente desenvolvido e otimizado para a exibição e manipulação de gráficos 3D. Desta forma, atualmente, praticamente todas as placas de vídeo possuem compatibilidade com a OpenGL, assim o processamento de gráficos 3D passa a ser efetuado pela placa de vídeo e não mais pela CPU.

OpenGL é o mais importante ambiente para o desenvolvimento de aplicações gráficas 2D e 3D portáveis e interativas. Desde sua introdução em 1992, tornou-se a API (application programming interface), para gráficos 2D e 3D, mais utilizada por desenvolvedores de software (Animação em 3D e modelagem; CAD/CAM; Simulação visual e realidade virtual; VRML; Jogos).

A biblioteca OpenGL auxilia no desenvolvimento rápido de aplicações gráficas pois tem incorporado a sua biblioteca, um conjunto de funções para sintetizar imagens (render), mapear texturas, efeitos especiais, e outras funções para visualização.

Para controlar a especificação da OpenGL foi criada uma sociedade independente, a OpenGL Architecture Review Board. Tornando assim a OpenGL um padrão multiplataforma, para aplicações gráficas, verdadeiramente aberto.

As implementações da OpenGL estão disponíveis ao longo de vários anos numa grande variedade de plataformas. Sendo os acréscimos para a especificação bem controlados, bem como a divulgação de tais mudanças feitas em tempo hábil aos desenvolvedores, a OpenGL torna-se uma biblioteca bastante estável.

Outra característica importante é a consistência da apresentação dos resultados visuais, em qualquer hardware em conformidade com a API OpenGL, não importando o sistema operacional nem o sistema de janelas em uso.

A OpenGL é estruturada de forma intuitiva e lógica. Tendo como resultado aplicações com menos linhas de código em comparação com outros pacotes e bibliotecas gráficas. Principalmente, pelo encapsulamento de informações específicas do hardware, que

permite ao desenvolvedor não se preocupar com o projeto de aplicações específicas para o hardware em uso.

Além de todos estas vantagens, ela ainda é uma API muito bem documentada. Existe uma grande quantidade de livros e sites sobre a OpenGL.

As rotinas da OpenGL facilitam o desenvolvimento de aplicações gráficas pois oferecem acesso a primitivas geométricas e de imagens, listas de visualização, modelagem de transformações, iluminação e textura, anti-aliasing, blending, e muitos outros recursos.

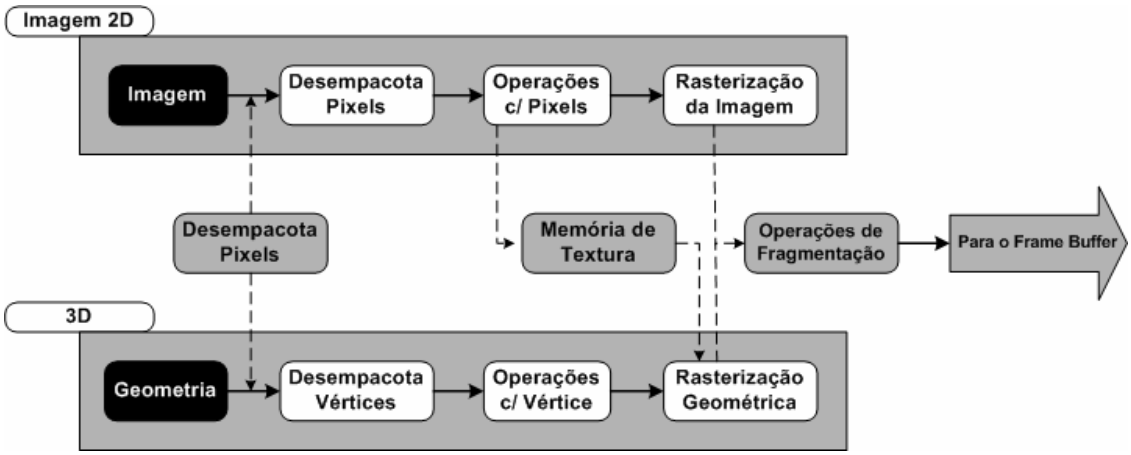

FIGURA 4.11 – Diagrama do Pipeline de Visualização da OpenGL.

FONTE: Adaptada da URL: http://www.opengl.org/developers/about/overview.html.

Todos os elementos do "estado" OpenGL - mesmo o conteúdo da memória de textura e o frame buffer - podem ser acessados por uma aplicação OpenGL. A OpenGL também suporta aplicações de visualização 2D com as imagens sendo tratadas como se fossem primitivas geométricas, que podem ser manipuladas como objetos geométricos 3D. Como pode ser visto no diagrama de pipeline da FIGURA 4.11, imagens e vértices definindo primitivas geométricas são passados do pipeline OpenGL para o frame buffer.

4.5.4 Software de Visualização OpenDX

O OpenDX (Open Data Explorer) é a versão open source do "Visualization Data Explorer" desenvolvido pela IBM. A última versão do "Visualization Data Explorer" foi a 3.1.4B, sendo o OpenDX baseado nela. Essa versão, open source, permite gerar visualizações para examinar desde simples conjuntos de dados até analisar dados complexos, dependentes do tempo, provenientes de fontes distintas.

Desde sua introdução em 1991 como Visualization Data Explorer, vem sendo continuamente aperfeiçoado e expandido, e além disso tem seu código-fonte disponível permitindo expandi-lo de acordo com as necessidades do projeto.

A seguir descreveremos algumas de suas características:

- A GUI (Graphical User Interface) foi construída para o ambiente de interface padrão: OSF/Motif(tm) e X Window System(tm). As imagens podem ser sintetizadas em janelas com pixels de precisão de 8-, 12-, 16-, 24-, e 32-bit. A GUI possui uma grande variedade de interadores, diretos e indiretos. Os diretos permitem manipular diretamente as imagens (p.ex. rotação ou zoom). Os indiretos (botões, sliders, dials) possibilitam o controle de vários aspectos da visualização. Esses interadores se ajustam aos dados, isto é, examinam os dados para determinar os valores mínimo e máximo.

- Fornece centenas de funções agrupadas em módulos. Por exemplo, "Compute" faz o trabalho de dezenas de funções individuais. Ele resolve equações algébricas e trigonométricas arbitrárias, assim como atualiza os dados.

- Possui um modelo de dados aperfeiçoado, orientado por objetos. Por manipular todos os dados de entrada, não importando a origem, de maneira uniforme, permite que cada módulo atue de forma apropriada. Além disso, mantém o sistema de coordenadas durante os passos de realização e visualização. Com isso torna fácil a correlação de dados oriundos de várias fontes distintas.

- O OpenDX foi projetado para trabalhar em ambiente cliente/servidor. Ele também pode executar em paralelo em uma máquina multiprocessada de memória compartilhada. Com isso, as visualizações podem ser distribuídas através de múltiplas estações de trabalho em um ambiente heterogêneo.

Maiores detalhes sobre o OpenDX podem ser obtidos em http://www.opendx.org/.

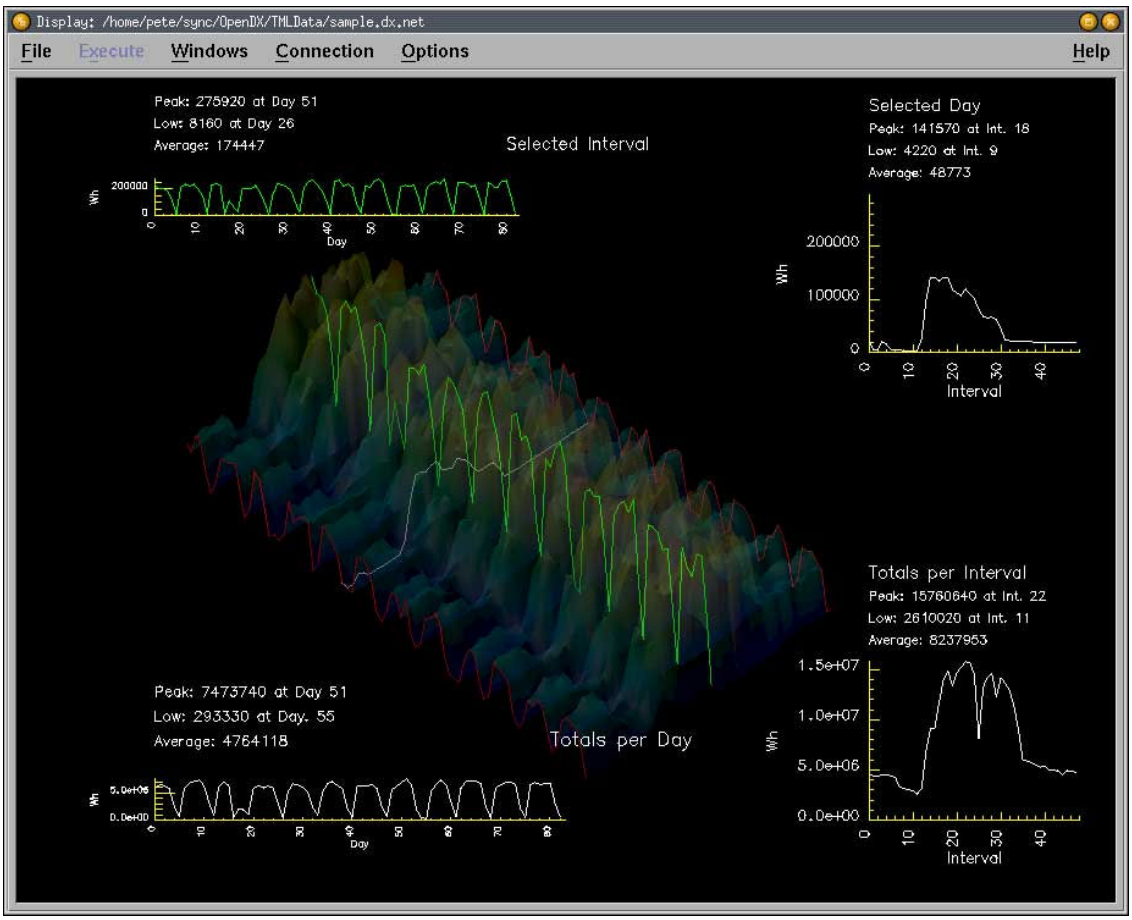

FIGURA 4.12 – Exemplo de visualização gerada pelo OpenDX.

FONTE: Retirada da URL: http://www.opendx.org/.

4.6 Estrutura Proposta

No diagrama da FIGURA 4.13 é apresentada uma visão macro do relacionamento entre a interface com o usuário (BSSData framework), a biblioteca otimizada BSSLibrary e

os plug-ins que venham a ser desenvolvidos. Neste fica evidente que teremos 4 tipos de plug-ins distintos, são eles:

- Manipulação de arquivos de dados: este tipo de plug-in será responsável pelas rotinas que manipulam arquivos de dados, basicamente ler e gravar um arquivo de dados. Para cada tipo de arquivo de dados teríamos um plug-in, ou seja, um plug-in responsável pela abertura de arquivos de dados tipo ESP, outro para arquivos de dados tipo DAT (arquivo de dados em modo ASCII, primeiro formato utilizado na digitalização de dados do BSS), outro para FITS (formato amplamente adotado pela comunidade científica) e quantos mais forem necessários. Com isso atenderíamos um dos requisitos levantados em 2.3.2 que era permitir a análise conjunta com dados advindos de outros observatórios.

- Visualização: estes plug-ins serão utilizados para gerar as diversas visualizações gráficas necessárias ao processo de análise de dados. No diagrama da FIGURA 4.13 exemplificamos alguns plug-ins visualização que deverão ser desenvolvidos para compor o sistema, estes plug-ins seriam por exemplo para gerar um histrograma, gerar perfil temporal e espectral estáticos, obter um zoom de uma área do espectro dinâmico, transformar o espectro dinâmico em uma superfície 3D e outras visualizações que se descubram necessárias. Para ser possível a geração de imagens e exibí-las ao usuário o framework do BSSData deverá criar uma janela padrão à todas as visualizações e enviar um handle de um contexto OpenGL aos plug-ins de visualização permitindo que estes desenhem nesta área.

- Filtros: estes plug-ins serão utilizados para implementar os filtros utilizados no tratamento de dados. Alguns destes filtros foram discutidos em 2.3.1 e como obtivemos bons resultados e como são uns dos requisitos leventados para o sistema devemos criar mecanismos que facitem o desenvolvimento dos mesmos. Basicamente o framework enviaria os dados ao plug-in e estes aplicariam os filtros nos dados retornando a matriz de dados modificada.

Um outro mecanismo importante para os plug-ins de filtragem de dados seria a possibilidade de abrir um diálogo de configuração do filtro, este diálogo deverá ser codificado dentro do plug-in permitindo assim que cada plug-in tenha um diálogo próprio.

- Extração de informações: como foi mostrado na Seção 2.3.2 existe a necessidade de criar ferramentas que facilitem a extração de informações do espectro dinâmico. Portanto, teremos um tipo plug-in responsável somente para extrair informações. Estes plug-ins permitiriam por exemplo determinar a taxa de deriva em freqüência de forma direta ou através de regressão linear ou não-linear, determinar a separação em freqüência e em tempo entre estruturas, determinar limites ou bordas e por serem codificados em forma de plug-ins poderemos criar novas ferramentas de acordo com os problemas que forem surgindo.

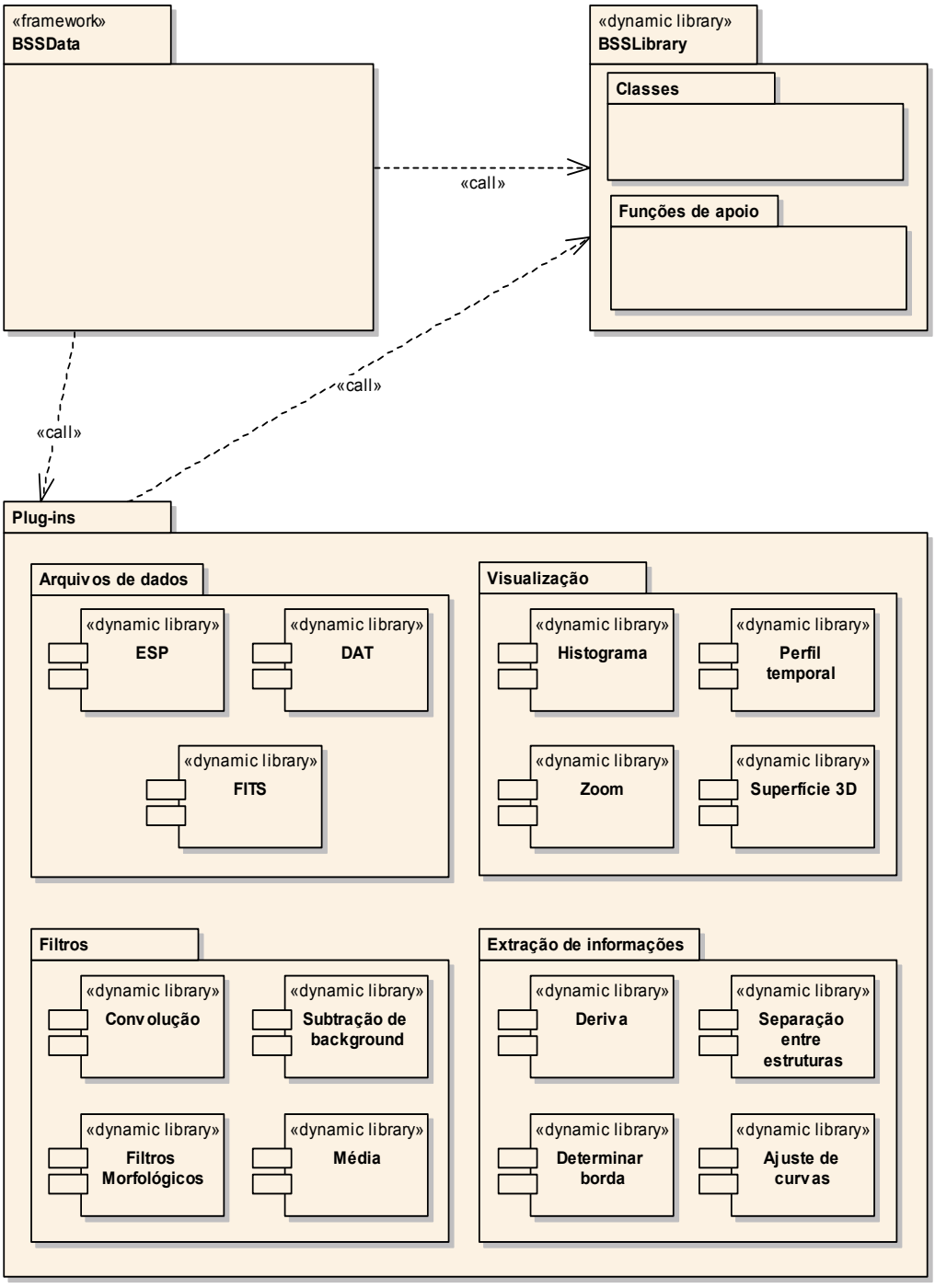

FIGURA 4.13 – Visão macro do protótipo do sistema otimizado para tratamento e análise de dados do BSS.

A BSSLibrary e os plug-ins serão escritos em linguagem C/C++ e linguagem assembly no caso das rotinas que forem otimizadas por vetorização, e deverão ser compilados

como DLL (Dynamic Link Library). Uma DLL é uma biblioteca de ligação dinâmica, ou seja, podem ser carregadas dinamicamente pelo software host. Os benefícios que teremos ao utilizar uma DLL seriam por exemplo, reaproveitá-las nos softwares que venham a ser desenvolvidos e que necessitem de funcionalidades já desenvolvidas, obtendo um ganho considerável de tempo de desenvolvimento, e como vários softwares podem acessar uma mesma DLL teríamos uma redução na quantidade de memória principal utilizada.

O framework, denominado BSSData, continuará sendo desenvolvido na ferramenta Borland C++ Builder, já que o framework será responsável pela interface com o usuário. Através do BSSData o usuário poderá habilitar ou desabilitar os plug-ins de acordo com suas necessidades. O software, então, deverá se moldar automaticamente com as novas configurações, criando todas as entradas em menus de comandos, barra de ferramentas, menus popup, ou seja, ele deverá ser auto-ajustável.

Para compor o sistema continuaríamos tendo a BSSLibrary (Martinon et al., 2002b), que além de ser uma biblioteca otimizada por vetorização irá fornecer todas as classes e funções de apoio para o desenvolvimento do próprio framework e dos plug-ins.

Para facilitar o desenvolvimento dos plug-ins criaremos templates de código para cada tipo de plug-in, ou seja, será criada toda a estrutura do plug-in com a declaração das funções (que obrigatoriamente deverão ter nomes pré-definidos), includes, etc. Desta forma, o programador ficaria preocupado somente com a implementação do código que dará funcionalidade ao plug-in não sendo necessário um conhecimento prévio de todo o projeto do sistema.

Com tudo o que foi exposto podemos enumerar os diversos benefícios que teremos ao adotar este modelo de desenvolvimento:

1) Abertura de qualquer formato de arquivo de dados, desde que possua as informações básicas necessárias para a geração de um espectro dinâmico. Tais como quantidade de canais e varreduras, freqüência e tempo inicial e final de observação, etc.

2) A modularização via plug-ins, facilita o desenvolvimento de novas rotinas, permitindo a expansão gradual do software de acordo com as necessidades.

3) Pode-se criar templates para cada tipo de plug-in que o software suporta, assim o programador que venha desenvolver novos plug-ins deverá apenas se preocupar com o código que dá a funcionalidade ao plug-in, não necessitando conhecer toda a estrutura do sistema.

4) Incorporar otimizações por vetorização aos plug-ins.

5) Facilidade para testar novas ferramentas sem que a estrutura do software tenha que ser modificada.

6) Possibilita o desenvolvimento de um sistema integrado de hardware e software, pois este modelo visa a reutilização dos componentes.

7) Finalmente, todas as classes, rotinas, diálogos etc, que foram desenvolvidos para a versão atual do BSSData serão reutilizados.

CAPÍTULO 5

CONCLUSÕES E COMENTÁRIOS FINAIS

Neste trabalho diversos aspectos relacionados ao desenvolvimento de um sistema otimizado para tratamento e análise de dados radioastronômicos (em física solar) foram abordados. Esse sistema foi baseado no software BSSData, inicialmente desenvolvido no escopo de um projeto de iniciação científica PIBIC/INPE. No presente trabalho, diversas melhorias foram feitas nesse software, tais como a otimização de rotinas através de vetorização utilizando instruções MMX que fazem parte da linguagem de máquina dos processadores utilizados, de arquitetura IA32. Esta otimização melhorou significativamente o desempenho computacional das mesmas, chegando à um ganho de até 9 vezes em relação ao código original escrito na linguagem C. Com estes resultados fica evidente o benefício trazido pela vetorização, bem como a aplicabilidade do uso de tais técnicas para o desenvolvimento do sistema proposto.

Alguns filtros de imagens foram implementados obtendo-se um bom resultado visual, isto é, a filtragem atingiu o seu objetivo de apresentar um melhor contraste entre as áreas da imagem que apresentam informações sobre explosões e o fundo (background).

Isto confirma que a filtragem é um passo importante para a realização de qualquer análise dos dados. Com a finalidade de implementar esses filtros, foi desenvolvida uma biblioteca otimizada, a qual foi implementada na forma de uma DLL (Dynamic Link Library), denominada BSSLibrary, a qual desvincula as rotinas de manipulação de dados da interface com o usuário.

O conjunto de rotinas implementadas foi selecionado a partir de um levantamento preliminar das ferramentas necessárias para o tratamento e análise de dados do BSS, sendo que apenas algumas rotinas selecionadas foram implementadas para mostrar a viabilidade do sistema proposto. Isso incluiu testar alguns aspectos técnicos de implementação, como por exemplo, a utilização de DLL's, integração com a biblioteca de visualização OpenGL, etc.

Atualmente a maioria dos sistemas para visualização, tratamento e análise de dados espectrais de radioastronomia solar são desenvolvidos com a utilização do software IDL

(Interactive Data Language), como por exemplo o BSSView (Faria, 1999), o RagView (Csillaghy, 1998), o SSW (SolarSoftWare - http://www.lmsal.com/solarsoft/). Entretanto, o problema em se adotar o IDL é que se trata de um software proprietário, exigindo o pagamento de licenças de uso que aparentemente não apresentam boa relação custo-benefício, uma vez que este software limita o desenvolvimento de interfaces amigáveis com o usuário e, sobretudo, tem um desempenho computacional bem inferior a um software equivalente desenvolvido em linguagem C.

Neste contexto, foi desenvolvido o BSSData e proposto o novo modelo de desenvolvimento baseado em plug-in's. A experiência prévia do autor no desenvolvimento da versão atual permitiu verificar a necessidade de um sistema que permita um desenvolvimento gradual o qual não afete a sua estrutura lógica, tal como o sistema proposto neste trabalho.

Os requisitos desejados são:

- Flexibilidade;
- adaptabilidade às necessidades do usuário;
- manutenção simples de forma a facilitar a inclusão de novas ferramentas;
- facilidade de acesso a dados provenientes de outros observatórios (outros formatos).

Como sugestões para trabalhos futuros podem-se citar, além da implementação do sistema otimizado proposto neste trabalho, a implementação de um sistema modular integrado de hardware e software, com facilidades tais como um banco de dados para armazenamento dos dados primários e tratados, interface "web" para acesso e manipulação dos dados via Internet e a automatização e melhoria do sistema de

aquisição, de tal forma que permita a visualização em tempo real dos dados que estão sendo adquiridos.

REFERÊNCIAS BIBLIOGRÁFICAS

Allaart, M. A. F.; van Nieuwkoop, J.; Slottje, C.; Sondaar, L. H. Atlas of fine structure in solar microwave bursts. **Solar Physics**, Oct. 1990.

Aurass, H.; Chernov G. P. Zebra pattern flux density observation during the type IV bursts on October 12, 1981. **Solar Physics,** v.84, p.339-345, 1983.

Aurass, H.; Chernov G. P.; Karlický M.; et al., On a sequence of remarkable fine structures in type IV burst of 24 April, 1985. **Solar Physics**, v.112, p.347-357, 1987.

AMD. **3DNow! Instruction porting guide application note**. Aug. 1999. Disponível em: <http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs /22621.pdf>. Acesso em: 23 mar. 2002.

AMD. **3Dnow! Technology manual**. Mar. 2000. Disponível em: <http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/21928. pdf>. Acesso em: 23 mar. 2002.

AMD. **AMD Athlon processor x86 code optimization guide**. Feb. 2002. Disponível em:<http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs /22007.pdf>. Acesso em: 23 mar. 2002.

AMD. **AMD Extensions to the 3DNow! and MMX instruction sets manual**. Mar. 2000. Disponível em: <http://www.amd.com/us-en/assets/content_type/white_papers_ and_tech_docs/22466.pdf>. Acesso em: 23 mar. 2002.

Bastian, T. S.; Gary, D. E.; White, S. M., Report to the Astronomy and Astrophysics Survey Committee. 1999. Disponível em: <http://solar.njit.edu/fasrt/Panel99r.html>. Acesso em: 22 mar. 2002.

Benz, A. O.; Furst, E.; Hirth, W.; Perrenoud, M. R. Solar radio blips and X-ray kernels. **Nature**, v.291, p.210-211, May 1981.

Benz, A. O.; Bernould, T. E. X.; Dennis, B. R. Radio blips and hard X-rays in solar flares. **The Astronomical Journal**, v.271, n.1, p.355-366, Aug. 1983.

Benz, A. O. Millisecond radio spikes. **Solar Physics**, v.104, p.99-110, 1986.

Benz, A. O.; Güdel, M. Harmonic emission and polarization of millisecond radio spikes. **Solar Physics**, v.111, n.1, p.175-180, 1987.

Benz, A. O.; Su, H.; Magun, A.; Stehling, W. Millisecond microwave spikes at 8 GHz during solar flares. **Astronomy and Astrophysics Supplement Series**. In press, 1991.

Benz, A. O.; Mann G. Intermediate drift bursts and the coronal magnetic field. **Astronomy and Astrophysics**, v.333, p.1034-1042, 1998.

Bernold, T. E. X. **A catalogue of fine structures in type IV solar radio bursts. Astronomy and Astrophysics**, v.42, p.43-58, 1980.

Bhonsle, R.V.; Sawant, H.S., Degaonkar, S.S. **Space Sciences Reviews**, v.24, p.259, 1979.

Brown, J. C.; Smith, D. F.; Spicer, D. S. Solar flare observations and their interpretation. In: Jordan, S. (ed). **The Sun as a star**: monograph series on thermal phenomena in stellar atmospheres. Washington: NASA, 1981. (NASA SP-450).

Bruggmann, G.; Benz, A.O.; Magun, A.; Stehling, W. **Astronomy and Astrophysics**, v.240, p.506, 1990.

Cecatto, J. R., Fernandes, F. C. R., Neri, J. A. C. F., Krishan, V., Sawant, H. S. High resolution time profile of decimetric type-III bursts. **Advances in Space Research**, 2003.

Csillaghy, A. **The image processing software Ragview.** Institute of Astronomy, ETH-Zentrum, Zurich, Switzerland. June 1998. Disponível em: <http://www.astro.phys. ethz.ch/rapp/software/ragview-manual.ps.>. Acesso em: 22 mar. 2002.

Dulk, G. A. Radio emission from the sun and stars. **Annual Review of Astronomy and Astrophysics**, v.23, n.1, p.169-224, 1985.

Faria, C. **Um sistema para tratamento e visualização de dados astronômicos espectrais**. 1999. (INPE-8036-TDI/752). Dissertação (Mestrado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 1999.

Fernandes, F.C.R. **Espectrógrafo decimétrico de alta sensibilidade e resolução:** análise preliminar das explosões solares. 1992. (INPE-5537-TDI/525). Dissertação (Mestrado em Astrofísica) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 1992.

Fernandes, F.C.R.; Sawant, H. S.; Zheleznyakov, V. V. Decimetric slow drift spliting pair. **Advances in Space Research**, v.17, n.4/5, p.143-146, 1996a.

Fernandes, F.C.R.; Zheleznyakov, V. V.; Sawant, H. S. Narrow band split frequency decimeter solar bursts. **Solar Physics**, v.168, n.1, p.125 - 135, 1996b.

Fernandes, F.C.R., **Espectrógrafo digital decimétrico de banda larga e investigações de flares solares em ondas decimétricas e raio-X.** 1997. (INPE-6396 – TDI/612). Tese (Doutorado em Astrofísica) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 1997.

Fernandes, F. C. R.; Cecatto, J. R.; Neri, J. A. C. F.; Faria, C.; Martinon, A. R. F.; Rosa, R. R.; Mesquita, F. P. V.; Portezani, V. A.; Andrade, M. C.; Alonso, E. M. B.; Vats, H. O.; Sawant, H. S. O Brazilian Solar Spectroscope (BSS) e os problamas atuais da física solar. **Boletim da Sociedade Astronômica Brasileira**, v.20, n.2, p.33-43, 2000.

Fernandes, F.C.R. **Catálogo de espectros dinâmicos de explosões solares decimétricas registradas pelo Brazilian Solar Spectroscope (BSS):** 1999. São José dos Campos, SP: INPE, 2003a. (INPE-965-RPQ/740). 82p.

Fernandes, F.C.R. **Catálogo de espectros dinâmicos de explosões solares decimétricas registradas pelo Brazilian Solar Spectroscope (BSS):** 2000. São José dos Campos, SP: INPE, 2003b. (INPE-9653-RPQ/739). 147p.

Fernandes, F.C.R. **Catálogo de espectros dinâmicos de explosões solares decimétricas registradas pelo Brazilian Solar Spectroscope (BSS):** 2001. São José dos Campos, SP: INPE, 2003c. (INPE-9652-RPQ/738). 126p.

Fernandes, F.C.R.; Krishan, V.; Cecatto, J. R.; Freitas, D. C.; Sawant, H. S. High resolution studies of intermediate drift bursts. **Advances in Space Research**, 2003d.

Fu, Q; Qin, Z.; Ji, H.; Pei, L. A broadband spectrometer for decimeter and microwave radio bursts. **Solar Physics,** v.160, n.1. p.97-103, 1995.

Güdel, D.A.; Benz, A.O. Time profiles of solar radio spikes. **Astronomy and Astrophysics**, v.231, n.1, p.202-212, May 1990.

Haber, R. B.; McNabb, D. A. Visualization idioms: a conceptual model for scientific visualization systems. In: Nielson, G.M.; Shriver, B. (eds). **Visualization in scientific computing.** [S.l.]: Rosenblum, 1990. p. 103-112.

Intel. **IA-32 Intel architecture software developer's manual**. Disponível em: <http://developer.intel.com>. Acesso em: 22 maio 2002a.

Intel. **Programming with the Intel MMX technology**. Disponível em: <http://developer.intel.com>. Acesso em 22 maio 2002b.

Intel. **Software development for SIMD technology**. Disponível em: <http://developer.intel.com>. Acesso em: 22 maio 2002c.

Isliker, H.; Benz, A.O. **Astronomy and Astrophysics Supplement Series**, v.104, p.145, 1994.

Jiricka, K.; Karlický, M.; Kepta, O.; Tlamicha, A. Fast drift burst observations with the new Ondrejov radiospectograph. **Solar Physics**, v.147, n.1 p.203-206, 1993.

Jiricka, K.; Karlický, M.; Mészárosová, H., Snízek, V. **Astronomy and Astrophysics,** v.375, p.243, 2001.

Karlicky, M., Barta, M., Jiricka, K., Meszarosova, H., Sawant, H. S., Fernandes, F. C. R., Cecatto, J. R., **Radio bursts with rapid frequency variations - Lace bursts**, Astronomy & Astrophysics, v.356, p.683 - 687, 2001.

Krüger, A.; Voight, W. Solar radio spectrography in Europe. **Solar Physics**, v.161, n.2, p.393-405, 1995.

Mann, G.; Karlický, M.; Motschmann, U. On the intermediate drift burst model. **Solar Physics**, v.110, p.381-389, 1987.

Martinon, A. R. F.; Fernandes, F. C. R., **Desenvolvimento/adaptação de software com aplicação na análise de dados do Brazilian Solar Spectroscope (BSS)**. Iniciação Científica, PIBIC/INPE. São José dos Campos: Instituto Nacional de Pesquisas Espaciais, 2000.

Martinon, A.R.F; Dobrowolski, M.; Jônatas Preto, A.; Stephany, S. Otimização de rotinas do software BSSData utilizando MMX. In: Encontro Regional de Matemática Aplicada e Computacional (ERMAC), 3, 2002, São José dos Campos. **Anais...** São José dos Campos: [S.n.], 2002a.

Martinon, A.R.F; Sawant, H. S.; Stephany, S.; Jônatas Preto, A.; Dobrowolski, M. BSSLibrary - Uma biblioteca de rotinas vetorizadas para filtragem de dados em radioastronomia solar. In: Workshop dos Cursos de Computação Aplicada do INPE, 2., 2002, São José dos Campos, SP. **Anais...** São José dos Campos: INPE, 2002b. p. 167-172.

Martinon, A. R. F.; Sawant, H. S.; Fernandes, F. C. R.; Stephany, S.; Preto, A. J.; Dobrowolski, K. M. BSSData - um programa otimizado para filtragem de dados em radioastronomia solar. In: Reunião Anual da SAB, 29., 2003, São Pedro, SP. **Boletim da SAB,** v. 23, p. 201-201, 2003.

Melendez, J.L., Sawant H.S.; Fernandes, F.C.R.; Benz, A.O. Statistical analysis of high-frequency decimetric type III bursts. **Solar Phys**ics, v.187, n.1, p.77-88, 1999.

Messmer, P.; Benz, A. O.; Monstein, C. PHOENIX-2: a new broadband spectrometer for decimetric and microwave radio bursts: first results. **Solar Physics**, v.187, n.2, p.335-345, 1999.

Sawant, H. S.; Costa, J. E. R.; Trevisan, R. H.; Lattari, C. J. B.; Kaufmann, P. Low-level decimetric (1.6 GHz) solar burst activity. **Solar Physics**, v.111, n.1, p.189-199, 1987.

Sawant, H.S.; Rosa, R.R. **Revista Mexicana de Astronomia y Astrofisica**, v.21, p.651, 1990.

Sawant, H.S.; Sobral, J.H.A.; Neri, J.A.C.F.; Fernandes, F.C.R.; Rosa, R.R.; Cecatto, J.R., Martinazzo, D. Reunião Anual da SBPC, 43.,1991. **Anais...** São Paulo: SBPC, 1991. p. 689.

Sawant, H.S.; Sobral, J.H.A.; Neri, J.A.C.F.; Fernandes, F.C.R.; Rosa, R.R.; Cecatto, J.R., Martinazzo, D. **Lecture Notes in Physics**, v.399, p.318, 1992.

Sawant, H.S.; Sobral, J.H.A.; Neri, J.A.C.F.; Fernandes, F.C.R.; Cecatto, J.R.; Rosa, R.R. High sensitivity digital decimetric spectroscope. **Advances in Space Research**, v.13, p.199, 1993.

Sawant, H.S.; Fernandes, F.C.R., Neri, J.A.C.F. **Astrophysical Journal Supplement Series**, v.90, p.689, 1994.

Sawant, H.S.; Sobral, J.H.A.; Fernandes, F.C.R.; Cecatto, J.R.; Day, W.R.G.; Neri, J.A.C.F.; Alonso, E.M.B.; Moraes, A**.** High sensitivity wide band digital solar polarimetric spectroscope. **Advances in Space Research**, n. 17, p. 385, 1996.

Sawant, H.S.; Subramanian, K.R.; Faria, C.; Stephany, S.; Fernandes, F.C.R.; Cecatto, J.R.; Rosa, R.R.; Alonso, E.M.B.; Mesquita, F.P.V.; Portezani, V.A. **ASP Conference Series**, v.206, p.341-346, 2000.

Sawant, H.S.; Subramanian, K.R.; Sobral, J.H.A.; Faria, C.; Fernandes, F.C.R.; Cecatto, J.R.; Rosa, R.R.; Vats, H.O.; Neri, J.A.C.F.; Alonso, E.M.B.; Mesquita, F.P.V.; Portezani, V.A.; Martinon, A. R. F. Brazilian solar spectroscope (bss). **Solar Physics**, v.200, n.1/2, p.167-176, 2001.

Sawant, H. S.; Fernandes, F. C. R.; Cecatto, J. R.; Vats, H. O.; Neri, J. A. C. F.; Portezani, V. A.; Martinon, A. R. F.; Karlický, M.; Jiricka, K.; Mészárosová, H., Decimetric dot-like structures. **Advances in Space Research,** v.29. n.3, p.349-354, 2002a.

Sawant, H. S., Karlicky, M., Fernandes, F. C. R., Cecatto, J. R. The 1.0-4.5 GHz zebra patterns in the June 6, 2000 flare. In: COSPAR Colloquia Series 13: multi-wavelength observations of coronal structure and dynamics. 2002b p.315– 316.

Sawant, H. S., Karlicky, M., Fernandes, F. C. R., Cecatto, J. R., Observation of harmonically related solar radio zebra patterns in the 1-4 GHz frequency range. **Astronomy and Astrophysics**, v.396, p.1015-1018, 2002c.

Slottje, C. Peculiar absoption and emission microstructures in the type IV solar outburst of March 2, 1970. **Solar Physics**, v.25, p.210, 1972a.

Slottje, C. Meeting of the CESRA, 2., 1971, Trieste. **Proceedins...** Trieste: [S.n.], 1972b. v.88.

Slottje, C. **Atlas of fine structuresof dynamic spectra.** [S.l.]: Dwingeloo, 1981.

Smith, D.F.; Benz, A.O. A mechanism for producing plasma radiation in the gigahertz range by precipitating high-energy protons. **Solar Physics,** v.131, n.2, p.351-361, 1991.

Stähli, M.; Benz, A. O. Microwave emission of solar electron beams. **Astronomy and Astrophysics**, v.175, n.1/2, p.271-276, Mar. 1987.

Sturrock, P. A. Type III solar radio bursts. In: Hess, W.N. (ed). **Proceedins of AAS-NASA Symposium on the Physics of Solar Flares**. Washington: NASA, 1964. p.357 (NASA SP-50).

Tarnstrom, G.L.; Philip, K.W. Solar radio spikes bursts. **Astronomy and Astrophysics**, v.16, n.1, p.21-27, 1972a.

Tarnstrom G. L.; Philip, K. W. **Scientifi report.** University of Alaska, UAG R-217, 1971.

Tarnstrom G.L.; Philip, K.W. Spike burst-typeIII burst associations. **Astronomy and Astrophysics**, v.17, n.1, p.267-275, 1972b.

Thompson, A.R.; Maxwell, A. Spectral observations of solar radio bursts: III. Continuum bursts. **Astrophysics Journal**, v.136, p.546, 1962.

Tlamicha, A. **Report.** Astronomical Institute of the Czechoslovak Academy of Science, 1990.

Upson, C.; Faulhaber, T.; Kamins, D.; Laidlaw, D.; Schlegel, D.; Vroom, J.; Gurwitz, R.; van Dam A. The Application Visualization System: A Computational Environment for Scientific Visualization. **IEEE Computer Graphics and Applications**, v. 9, n.4,p.30-42, July 1989.

Young, C. W.; Spencer C. L.; Moreton G. E. A preliminary study of the dynamic spectra of solar radio bursts in the frequency range 500-950 Mc/s. **Astrophysics Journal**, v.133, p.243, 1961.

Zirin, H. Solar flares. In: _____ (ed). **Astrophysic of the sun**. Cambridge, Cambridge University, 1988. Cap. 11, p.343-404.

APÊNDICE A

Classe cDataset

Nas tabelas a seguir serão apresentadas as **propriedades públicas**, os **métodos privados** e os **métodos públicos** da cDataset.

PROPRIEDADES PÚBLICAS DA CLASSE CDATASET

Propriedade	Descrição	Declaração
cDataset::header	Cabeçalho do arquivo ESP.	**sHeader** *header;
cDataset::matrix	Matriz de dados contida no arquivo ESP.	**short** *matrix;
cDataset::media	Média aritmética de todos os elementos da matriz.	**short** media;
cDataset::max	Valor máximo de matrix.	**short** max;
cDataset::min	Valor mínimo de matrix.	**short** min;
cDataset::dt	Resolução temporal, calculada pelo método DTCalculate.	**double** dt;
cDataset::index	Índice do conjunto de dados, usado para achar seu endereço na memória.	**int** index;

MÉTODOS PRIVADOS DA CLASSE CDATASET

Propriedade	Descrição	Declaração
cDataset::DTCalculate	Fornece o DT (resolução temporal) real, diferente do DT que consta no cabeçalho. Usado para termos uma escala de tempo mais exata.	**void** DTCalculate(**void**);

MÉTODOS PÚBLICOS DA CLASSE CDATASET

Método	Descrição	Declaração
cDataset::cDataset	Construtor da classe. Responsável pela abertura do arquivo, alocação de espaço em memória para os dados, execução de rotinas de pré-processamento.	cDataset(**char** *Filename);
cDataset::~cDataset	Destrutor da classe. Desaloca a memória alocada pelo construtor.	~cDataset(**void**);
cDataset::GetMinMax	Retorna os elementos máximo e mínimo da matriz.	**void** GetMinMax(**void**);

cDataset::Inverse	Inverte todos os elementos da matriz. A inversão é feita subtraindo-se de 4095 cada elemento da matriz original e o resultado desta operação será a matriz invertida.	**void** Inverse(**void**);
cDataset::BackSubtract	Subtrai o sinal de background da matriz.	**void** BackSubtract(**int** x1, **int** x2);
cDataset::Derivate	Filtro da derivada.	**void** Derivate(**void**);
cDataset::Convolution	Filtro de convolução.	**void** Convolution(**double** *mask, **int** order, **double** divisor, **short** bias);
cDataset::Media	Filtro da média.	**void** Media(**void**);
cDataset::Median	Filtro da mediana.	**void** Median(**const int** order);
cDataset::Dilate	Filtro morfológico de dialação.	**void** Dilate(**short** mask[][3], **const int** order);
cDataset::Erode	Filtro morfológico de erosão.	**void** Erode(**short** mask[][3], **const int** order);
cDataset::Transpose	Transpõe a matriz de dados.	**void** Transpose(**void**);

Classe cDatasets

Nas tabelas a seguir serão apresentadas todas as **propriedades privadas** e os **métodos públicos** da classe cDatasets.

PROPRIEDADES PRIVADAS DA CLASSE CDATASETS

Propriedade	Descrição	Declaração
cDatasets::iCurrent	Índice do conjunto de dados corrente.	**int** iCurrent;
cDatasets::iTotal	Número total de conjuntos de dados abertos.	**int** iTotal;
cDatasets::path**	Lista dos caminhos de todos os conjuntos de dados abertos.	**char** **path;
cDatasets::list**	Lista de ponteiros para todos os conjuntos de dados abertos.	**cDataset** **list;
cDatasets::current	Ponteiro para o conjunto de dados corrente.	**cDataset** *current;

MÉTODOS PÚBLICOS DA CLASSE CDATASETS

Método	Descrição	Declaração
cDatasets::cDatasets	Construtor da classe. Inicializa as propriedades relativas ao controle da lista de objetos, bem como as listas em si.	cDatasets(**void**);
cDatasets::~cDatasets	Destrutor da classe. Desaloca toda a memória alocada pelos métodos classe.	~cDatasets(**void**);
cDatasets::AddDataset	Adiciona um novo conjunto de dados à lista de objetos.	**void** AddDataset(**char** *FileName);
cDatasets::SaveDataset	Grava as modificações do conjunto de dados selecionado em disco.	**void** SaveDataset(**char** *FileName);
cDatasets::RemoveDataset	Remove um conjunto de dados da lista de objetos.	**void** RemoveDataset(**int** index);
cDatasets::GetTotal	Retorna a quantidade de conjuntos de dados abertos no momento.	**int** GetTotal(**void**) {return iTotal;}
cDatasets::GetPath	Retorna o caminho, na árvore de diretórios, do arquivo de dados selecionado.	**char** *GetPath(**int** i) {return path[i];}
cDatasets::SetCurrent	Muda o ponteiro do conjunto de dados corrente.	**void** SetCurrent(**int** i);
cDatasets::GetCurrent	Retorna um ponteiro para o conjunto de dados atual.	**cDataset** *GetCurrent(**void**) {return current;}

Classe cPalette

Nas tabelas a seguir serão apresentadas todas as **propriedades privadas**, os **métodos privados** e os **métodos públicos** da classe cPalette.

PROPRIEDADES PRIVADAS DA CLASSE CPALETTE

Propriedade	Descrição	Declaração
cPalette::hDC	Armazena um handle para o DC (Device Context) da aplicação.	**HDC** hDC;
cPalette::LogicalPalette	Ponteiro para uma paleta de cores lógica alocada pelo construtor da classe.	**LOGPALETTE** *LogicalPalette;
cPalette::LogicalPalette1	Ponteiro para uma paleta de cores lógica, secundária, alocada pelo construtor da classe.	**LOGPALETTE** *LogicalPalette1;

cPalette::hPalette	Armazena um handle para a paleta de cores criada.	**HPALETTE** hPalette;

MÉTODOS PRIVADOS DA CLASSE CPALETTE

Método	Descrição	Declaração
cPalette:: ScaleValue	Realiza o mapeamento 256 cores em 236 cores, pois 20 cores são reservadas ao sistema.	**int** ScaleValue(**const bool** flag, **const int** i);
cPalette:: FillLP1	Preenche a paleta de cores lógica, secundária.	**void** FillLP1(**void**);

MÉTODOS PÚBLICOS DA CLASSE CPALETTE

Método	Descrição	Declaração
cPalette::cPalette	Construtor da classe. Aloca as paletas lógicas e cria uma paleta padrão de tons de cinza.	cPalette(**void**);
cPalette::~cPalette	Destrutor da classe. Desaloca toda a memória utilizada pela classe.	~cPalette(**void**);
cPalette::GetColorDepth	Retorna a quantidade de cores (256 cores, high color, true color) do dispositivo de exibição.	**int** GetColorDepth(**void**);
cPalette::GetPixelDepth	Retorna o tamanho de cada pixel em bits (8-bit, 16-bit, 24-bit, 32-bit).	**int** GetPixelDepth(**void**);
cPalette::CreateGradientPalette	Cria um gradiente de cores.	**void** CreateGradientPalette (**const bool** red, **const bool** green, **const bool** blue);
cPalette::InvertPalette	Inverte a paleta.	**void** InvertPalette(**void**);
cPalette::ReadPaletteFile	Responsável pela leitura dos dados de um arquivo pré-definido.	**void** ReadPaletteFile(**char** *filename);
cPalette::Stretch	Estica a paleta de cores.	**void** Stretch(**int** percentbottom, **int** percenttop);
cPalette::GetPalette	Retorna o handle da paleta de cores.	**HPALETTE** GetPalette(**void**) {return hPalette;}
cPalette::GetIndependentPalette	Retorna um handle para uma nova paleta de cores, independente da criada pela classe.	**HPALETTE** GetIndependentPalette(**void**);

Classe cScaledCoord

Nas tabelas a seguir serão apresentadas todas as **propriedades privadas**, as **propriedades públicas** e os **métodos públicos** da classe cScaledCoord.

PROPRIEDADES PRIVADAS DA CLASSE CSCALEDCOORD

Propriedade	Descrição	Declaração
cScaledCoord::MaxX	Largura da janela gráfica na qual deve ser feita a conversão de coordenadas.	**double** MaxX;
cScaledCoord::MaxY	Altura da janela gráfica na qual deve ser feita a conversão de coordenadas.	**double** MaxY;
cScaledCoord::Xi	Define a coordenada lógica x, inicial	**double** Xi;
cScaledCoord::Yi	Define a coordenada lógica y, inicial.	**double** Yi;
cScaledCoord::Xf	Define a coordenada lógica x, final.	**double** Xf;
cScaledCoord::Yf	Define a coordenada lógica y, final.	**double** Yf;
cScaledCoord::tmpXi	Armazena um valor temporário da coordenada lógica x (inicial), usado no cálculo para correção do aspecto da imagem (proporção).	**double** tmpXi;
cScaledCoord::tmpYi	Armazena um valor temporário da coordenada lógica y (inicial), usado no cálculo para correção do aspecto da imagem (proporção).	**double** tmpYi;
cScaledCoord::tmpXf	Armazena um valor temporário da coordenada lógica x (final), usado no cálculo para correção do aspecto da imagem (proporção).	**double** tmpXf;
cScaledCoord::tmpYf	Armazena um valor temporário da coordenada lógica y (final), usado no cálculo para correção do aspecto da imagem (proporção).	**double** tmpYf;
cScaledCoord::PlotArea	Define o retângulo, dentro da janela gráfica, que deve sofrer a conversão de coordenadas.	**TRect** PlotArea;

PROPRIEDADES PÚBLICAS DA CLASSE CSCALEDCOORD

Propriedade	Descrição	Declaração
cScaledCoord::AspectRatio	Deve receber um valor booleano, verdadeiro (true) ou falso (false). Se for verdadeiro corrige o aspecto da imagem caso contrário não faz a correção.	**bool** AspectRatio;

MÉTODOS PÚBLICOS DA CLASSE CSCALEDCOORD

Método	Descrição	Declaração
cScaledCoord::cScaledCoord	Construtor da classe. Essa classe possui dois construtores. Então, se for passado um parâmetro do tipo TRect a classe é inicializada definindo um plano contido em uma área física de x pixels de largura por y pixels de altura. Caso contrário a classe irá ser inicializada com valores padrões e deve-se, posteriormente, chamar o método cScaledCoord::Resize para ajustar o tamanho da janela gráfica.	cScaledCoord(**void**); cScaledCoord(**TRect** a);
cScaledCoord::~cScaledCoord	Destrutor da classe. Esse método não possui nenhum código, pois não se está alocando memória internamente na classe.	~cScaledCoord(**void**) {;}
cScaledCoord::RealX	Retorna o resultado da conversão de uma coordenada x lógica em uma coordenada x física.	**int** RealX(**double** x);
cScaledCoord::RealY	Retorna o resultado da conversão de uma coordenada y lógica em uma coordenada y física.	**int** RealY(**double** y);
cScaledCoord::ScaledX	Retorna o resultado da conversão de um x físico da janela gráfica em um x lógico.	**double** ScaledX(**int** x);
cScaledCoord::ScaledY	Retorna o resultado da conversão de um y físico da janela gráfica em um y lógico.	**double** ScaledY(**int** y);
cScaledCoord::SetViewPort	Define as posições, das coordenadas lógicas, da janela gráfica.	**void** SetViewPort(**double** x1, **double** y1, **double** x2, **double** y2);
cScaledCoord::GetViewPort	Retorna as posições, das coordenadas lógicas, da janela gráfica	**sFRect** GetViewPort(**void**);
cScaledCoord::Resize	Ajusta as posições da janela gráfica, em coordenadas físicas.	**void** Resize(**TRect** a);
cScaledCoord::GetScaledCoord	Retorna o resultado da conversão das coordenadas x e y físicas em coordenadas x e y lógicos.	**sFSize** GetScaledCoord(**int** x, **int** y);

Classe cViewGraph

Nas tabelas a seguir serão apresentadas todas as **propriedades protegidas**, as **propriedades públicas**, os **métodos privados**, os **métodos protegidos** e os **métodos públicos** da classe cViewGraph.

PROPRIEDADES PROTEGIDAS DA CLASSE CVIEWGRAPH

Propriedade	Descrição	Declaração
cViewGraph::Paper	Endereço de memória (ponteiro) da área na qual deve ser desenhada a moldura.	**TCanvas** *Paper;
cViewGraph::Margin	Armazena o tamanho das margens da moldura.	**TFRect** Margin;
cViewGraph::bWidth	Armazena a largura da moldura.	**int** bWidth
cViewGraph::bHeight	Armazena a altura da moldura.	**int** bHeight;
cViewGraph::TicksPosX	Vetor que armazena as posições das marcas principais do eixo x.	**int** *TicksPosX;
cViewGraph::TicksPosY	Vetor que armazena as posições das marcas principais do eixo y.	**int** *TicksPosY;
cViewGraph::SubTicksPosX	Vetor que armazena as posições das marcas secundárias do eixo x.	**int** *SubTicksPosX;
cViewGraph::SubTicksPosY	Vetor que armazena as posições das marcas principais do eixo y.	**int** *SubTicksPosY;
cViewGraph::lfxstr	Armazena o tamanho máximo (largura e altura) do texto usado nos rótulos do eixo x.	**TFSize** lfxstr;
cViewGraph::lfystr	Armazena o tamanho máximo (largura e altura) do texto usado nos rótulos do eixo y.	**TFSize** lfystr;
cViewGraph::stfxstr	Armazena o tamanho (largura e altura) do texto usado no subtítulo do eixo x.	**TFSize** stfxstr;
cViewGraph::stfystr	Armazena o tamanho (largura e altura) do texto usado no subtítulo do eixo y.	**TFSize** stfystr;
cViewGraph::tfstr	Armazena o tamanho (largura e altura) do texto usado no título da moldura.	**TFSize** tfstr;
cViewGraph::PlotAreaSize	Armazena o tamanho (largura e altura) da área que pode ser usada para plotar as curvas.	**TFSize** PlotAreaSize;

PROPRIEDADES PÚBLICAS DA CLASSE CVIEWGRAPH

Propriedade	Descrição	Declaração
cViewGraph::ClassType	Quando uma classe derivada de cViewGraph é inicializada o seu construtor preenche essa propriedade. Que é usada ao longo do programa para se fazer "typecasts"	**AnsiString** ClassType;
cViewGraph::AxisX	Define as informações referentes ao eixo x através de uma estrutura sAxis.	**sAxis** AxisX;
cViewGraph::AxisY	Define as informações referentes ao eixo y através de uma estrutura sAxis.	**sAxis** AxisY;
cViewGraph::TitleFont	Define a fonte usada para exibir o título da moldura.	**TFont** *TitleFont;
cViewGraph::SubTitleXFont	Define a fonte usada para exibir o subtítulo do eixo x.	**TFont** *SubTitleXFont;
cViewGraph::SubTitleYFont	Define a fonte usada para exibir o subtítulo do eixo y.	**TFont** *SubTitleYFont;
cViewGraph::LabelFont	Define a fonte usada para exibir os rótulos desenhados nos eixo graduados.	**TFont** *LabelFont;
cViewGraph::Title	Define o texto a ser exibido como título da moldura.	**AnsiString** Title;
cViewGraph::SubTitleX	Define o texto a ser exibido como subtítulo do eixo x.	**AnsiString** SubTitleX;
cViewGraph::SubTitleY	Define o texto à ser exibido como subtítulo do eixo y.	**AnsiString** SubTitleY;
cViewGraph::ShowTitle	Define se será exibido um título na moldura.	**bool** ShowTitle;
cViewGraph::ShowSubTitleX	Define se será exibido um subtítulo no eixo x da moldura.	**bool** ShowSubTitleX;
cViewGraph::ShowSubTitleY	Define se será exibido um subtítulo no eixo y da moldura.	**bool** ShowSubTitleY;
cViewGraph::ShowGridLines	Define se serão exibidas linhas de grade na moldura.	**bool** ShowGridLines;
cViewGraph::ShowLabel	Define de serão exibidos rótulos nos eixos graduados da moldura.	**bool** ShowLabel;
cViewGraph::PlotAreaColor	Define a cor de fundo da área onde será plotada as curvas.	**TColor** PlotAreaColor;
cViewGraph::MarginAreaColor	Define a cor da margem da moldura.	**TColor** MarginAreaColor;
cViewGraph::AxisColor	Define a cor dos eixos graduados.	**TColor** AxisColor;
cViewGraph::TicksColor	Define a cor das marcações principais dos eixos graduados. Essa cor é, também, usada para desenhar as linhas de grade principais.	**TColor** TicksColor;

Propriedade	Descrição	Declaração
cViewGraph::SubTicksColor	Define a cor das marcações secundárias dos eixos graduados. Essa cor é, também, usada para desenhar as linhas de grade secundárias.	**TColor** SubTicksColor;
cViewGraph::TicklenX	Define a tamanho das marcações principais do eixo x. Esse tamanho é uma porcentagem da altura da janela gráfica.	**double** TicklenX;
cViewGraph::TicklenY	Define a tamanho das marcações principais do eixo y. Esse tamanho é uma porcentagem da largura da janela gráfica.	**double** TicklenY;
cViewGraph::AspectRatio	Define se deve ou não ser feita a correção do aspecto da imagem.	**bool** AspectRatio;
cViewGraph::xdigits	Define o número de casas decimais que deverão ser exibidas no texto dos rótulos do eixo x.	**int** xdigits;
cViewGraph::ydigits	Define o número de casas decimais que deverão ser exibidas no texto dos rótulos do eixo y.	**int** ydigits;

MÉTODOS PRIVADOS DA CLASSE CVIEWGRAPH

Método	Descrição	Declaração
cViewGraph::InitialSetting	Define os valores padrões para todas as propriedades da classe.	**void** InitialSetting(**void**);

MÉTODOS PROTEGIDOS DA CLASSE CVIEWGRAPH

Método	Descrição	Declaração
cViewGraph::CalculateMargin	Esse método faz o cálculo do tamanho das margens, necessário, para exibir títulos, subtítulos, rótulos etc, de acordo com os valores definidos nas propriedades da classe. Como é um método virtual ele pode ser sobrecarregado por classes que derivem da classe cViewGraph.	**virtual void** CalculateMargin(**void**);

Método	Descrição	Declaração
cViewGraph::CalculateTickPos	Esse método cálculo a posição das marcações principais e secundárias e as armazena nos vetores TicksPosX, TicksPosY, SubTicksPosX e SubTicksPosY, que propriedades privadas da classe.	**void** CalculateTickPos(**void**);
cViewGraph::GraduateLine	Desenha um eixo graduado na moldura, com suas respectivas marcações.	**void** GraduateLine(**double** pos, **int** begin, **int** end, **int** orientation, **int** edge, **sAxis** axis);
cViewGraph::GraduateText	Desenha os textos dos rótulos nos eixos graduados. Como é um método virtual ele pode ser sobrecarregado por classes que derivem da classe cViewGraph.	**virtual void** GraduateText(**double** posx, **double** posy, **int** orientation);
cViewGraph::DrawGridLines	Desenha as linhas de grade na moldura.	**void** DrawGridLines(**int** begin, **int** end, **int** orientation);

MÉTODOS PÚBLICOS DA CLASSE CVIEWGRAPH

Método	Descrição	Declaração
cViewGraph::cViewGraph	Construtor da classe. Essa classe possui dois construtores. Qualquer um dos dois inicializa a classe, mas se não for passado os dois parâmetros do tipo int deve-se, posteriormente, chamar o método cViewGraph::Resize.	cViewGraph(**TCanvas** *p); cViewGraph(**TCanvas** *p, **int** Width, **int** Height);
cViewGraph::~cViewGraph	Destrutor da classe. Desaloca todas as propriedades que foram alocadas dinamicamente na memória.	~cViewGraph(**void**);
cViewGraph::Redraw	Redesenha a moldura na área de memória definida pela propriedade cViewGraph::Paper. Como é um método virtual ele pode ser sobrecarregado por classes que derivem da classe cViewGraph.	**virtual void** Redraw**(void);**
cViewGraph::Resize	Redimensiona a moldura. Há a necessidade de se chamar o método cViewGraph::Redraw para que a moldura seja redesenhada com sua nova dimensão.	**void** Resize**(int** Width**, int** Height**);**
cViewGraph::GetPlotAreaSize	Retorna a área gráfica disponível para que uma curva seja plotada.	**TFRect** GetPlotAreaSize(**void**);

Classe cVGBoxedInside

Nas tabelas a seguir serão apresentadas todos os **métodos privados** e **públicos** da classe cVGBoxedInside.

MÉTODOS PRIVADOS DA CLASSE CVGBOXEDINSIDE

Método	Descrição	Declaração
cVGBoxedInside::CalculateMargin	Sobrescreve o método herdado de cViewGraph, acrescentando rotinas de cálculo específicos para a classe cVGBoxedInside.	**void** CalculateMargin(**void**);
cVGBoxedInside::GraduateText	Sobrescreve o método herdado de cViewGraph, acrescentando rotinas de plotagem de texto específicas para a classe cVGBoxedInside.	**void** GraduateText(double posx, **double** posy, **int** orientation);

MÉTODOS PÚBLICOS DA CLASSE CVGBOXEDINSIDE

Método	Descrição	Declaração
cVGBoxedInside::Redraw	Redesenha a moldura. Sobrescrevendo a o método herdado de cViewGraph, acrescentando rotinas específicas para a classe cVGBoxedInside.	**void** Redraw(**void**);

cVGBoxedInside::cVGBoxedInside	Construtor da classe. Essa classe possui dois construtores. Eles definem a propriedade cViewGraph::ClassType para "VGBoxedInside" e passam os valores que recebem como parâmetros para os construtores da classe, da qual deriva, cViewGraph.	cVGBoxedInside(**TCanvas** *p);
		
		cVGBoxedInside(**TCanvas** *p, **int** Width, **int** Height);

cVGBoxedInside::~cVGBoxedInside	Destrutor da classe. Não contém nenhum código, pois depois dele é chamado o destrutor da classe cViewGraph.	~cVGBoxedInside(**void**);

Classe cVGBoxedOutside

Nas tabelas a seguir serão apresentados todos os **métodos privados** e **públicos** da classe cVGBoxedOutside.

MÉTODOS PRIVADOS DA CLASSE CVGBOXEDOUTSIDE

Método	Descrição	Declaração
cVGBoxedOutside::CalculateMargin	Sobrescreve o método herdado de cViewGraph, acrescentando rotinas de cálculo específicos para a classe cVGBoxedOutside.	**void** CalculateMargin(**void**);
cVGBoxedOutside::GraduateText	Sobrescreve o método herdado de cViewGraph, acrescentando rotinas de plotagem de texto específicas para a classe cVGBoxedOutside.	**void** GraduateText(double posx, **double** posy, **int** orientation);

MÉTODOS PÚBLICOS DA CLASSE CVGBOXEDOUTSIDE

Método	Descrição	Declaração
cVGBoxedOutside::Redraw	Redesenha a moldura. Sobrescrevendo a o método herdado de cViewGraph, acrescentando rotinas específicas para a classe cVGBoxedOutside.	**void** Redraw(**void**);

cVGBoxedOutside::cVGBoxedOutside	Construtor da classe. Essa classe possui dois construtores. Eles definem a propriedade cViewGraph::ClassType para "VGBoxedOutside" e passam os valores que recebem como parâmetros para os construtores da classe, da qual deriva, cViewGraph.	cVGBoxedOutside(**TCanvas** *p);
		
		cVGBoxedOutside(**TCanvas** *p, **int** Width, **int** Height);

cVGBoxedOutside::~cVGBoxedOutside	Destrutor da classe. Não contém nenhum código, pois depois dele é chamado o destrutor da classe cViewGraph.	~cVGBoxedOutside(**void**);

Classe cVGCartesian

Nas tabelas a seguir serão apresentados todos os métodos privados e públicos da classe cVGCartesian.

MÉTODOS PRIVADOS DA CLASSE CVGCARTESIAN

Método	Descrição	Declaração
cVGCartesian::CalculateMargin	Sobrescreve o método herdado de cViewGraph, acrescentando rotinas de cálculo específicos para a classe cVGCartesian.	**void** CalculateMargin(**void**);
cVGCartesian::GraduateText	Sobrescreve o método herdado de cViewGraph, acrescentando rotinas de plotagem de texto específicas para a classe cVGCartesian.	**void** GraduateText(double posx, **double** posy, **int** orientation);

MÉTODOS PÚBLICOS DA CLASSE CVGCARTESIAN

Método	Descrição	Declaração
cVGCartesian::Redraw	Redesenha a moldura. Sobrescrevendo o método herdado de cViewGraph, acrescentando rotinas específicas para a classe cVGCartesian.	**void** Redraw(**void**);
cVGCartesian::cVGCartesian	Construtor da classe. Essa classe possui dois construtores. Eles definem a propriedade cViewGraph::ClassType para "VGCartesian" e passam os valores que recebem como parâmetros para os construtores da classe, da qual deriva, cViewGraph.	cVGCartesian(**TCanvas** *p); cVGCartesian (**TCanvas** *p, **int** Width, **int** Height);
cVGCartesian::~cVGCartesian	Destrutor da classe. Não contém nenhum código, pois depois dele é chamado o destrutor da classe cViewGraph.	~cVGCartesian (**void**);

Classe cVGTimeBoxedInside

Nas tabelas a seguir serão apresentados todas as **propriedades públicas**, os **métodos privados** e os **métodos públicos** da classe cVGTimeBoxedInside.

PROPRIEDADES PÚBLICAS DA CLASSE CVGTIMEBOXEDINSIDE

Propriedade	Descrição	Declaração
cVGTimeBoxedInside::InitialTime	Tempo inicial, usado para definir a escala de tempo que deverá ser plotada no eixo das abscissas.	**TDateTime** InitialTime**;**
cVGTimeBoxedInside::dt	Resolução temporal do arquivo de dados.	**int** dt**;**

MÉTODOS PRIVADOS DA CLASSE CVGTIMEBOXEDINSIDE

Método	Descrição	Declaração
cVGTimeBoxedInside::CalculateMargin	Sobrescreve o método herdado de cViewGraph, acrescentando rotinas de cálculo específicos para a classe cVGTimeBoxedInside.	**void** CalculateMargin(**void**);
cVGTimeBoxedInside::GraduateText	Sobrescreve o método herdado de cViewGraph, acrescentando rotinas de plotagem de texto específicas para a classe cVGTimeBoxedInside.	**void** GraduateText(double posx, **double** posy, **int** orientation);

MÉTODOS PÚBLICOS DA CLASSE CVGTIMEBOXEDINSIDE

Método	Descrição	Declaração
cVGTimeBoxedInside::Redraw	Redesenha a moldura. Sobrescrevendo o método herdado de cViewGraph, acrescentando rotinas específicas para a classe cVGTimeBoxedInside.	**void** Redraw(**void**);

Método	Descrição	Declaração
cVGTimeBoxedInside::cVGTimeBoxedInside	Construtor da classe. Essa classe possui dois construtores. Eles definem a propriedade cViewGraph::ClassType para "VGTimeBoxedInside" e passam os valores que recebem como parâmetros para os construtores da classe, da qual deriva, cViewGraph.	cVGTimeBoxedInside (**TCanvas** *p); cVGTimeBoxedInside (**TCanvas** *p, **int** Width, **int** Height);
cVGTimeBoxedInside::~cVGTimeBoxedInside	Destrutor da classe. Não contém nenhum código, pois depois dele é chamado o destrutor da classe cViewGraph.	~cVGTimeBoxedInside (**void**);