Project Report
ATC-274

Machine Intelligent Gust Front Algorithm for the WSP

S.W. Troxel
W.L. Pughe

10 June 2002

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS

**Prepared for the Federal Aviation Administration,
Washington, D.C. 20591**

**This document is available to the public through
the National Technical Information Service,
Springfield, Virginia 22161.**

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	10 June 2002	Project Report

4. TITLE AND SUBTITLE

Machine Intelligent Gust Front Algorithm for the WSP

5. FUNDING NUMBERS

C — F19628-00-C-0002

6. AUTHOR(S)

S.W. Troxel and W.L. Pughe

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Lincoln Laboratory, MIT
244 Wood Street
Lexington, MA 02420-9108

8. PERFORMING ORGANIZATION REPORT NUMBER

ATC-274

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of Transportation
Federal Aviation Administration
800 Independence Ave., S.W.
Washington, DC 20591

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

This report is based upon studies performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology, under Air Force contract F19628-00-C-0002.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

This document is available to the public through the National Technical Information Service, Springfield, VA 22161

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

The Machine Intelligent Gust Front Algorithm (MIGFA) utilizes multi-dimensional image processing and fuzzy logic techniques to identify gust fronts in Doppler radar data generated by the ASR-9 Weather Systems Processor (WSP). The algorithm generates products that support both safety and planning functions for ATC. Outputs include current and predicted locations of gust fronts, as well as estimates of the wind shear and wind shift associated with each gust front.

This document provides both high level and detailed functional descriptions of FAA build 2.0 of the WSP MIGFA. The document was written with many explicit references to data structures and routines in the actual software in order that it may serve as a useful algorithm development and programmers reference guide.

14. SUBJECT TERMS

15. NUMBER OF PAGES
220

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unclassified

ABSTRACT

The Machine Intelligent Gust Front Algorithm (MIGFA) utilizes multi-dimensional image processing and fuzzy logic techniques to identify gust fronts in Doppler radar data generated by the ASR-9 Weather Systems Processor (WSP). The algorithm generates products that support both safety and planning functions for ATC. Outputs include current and predicted locations of gust fronts, as well as estimates of the wind shear and wind shift associated with each gust front.

This document provides both high level and detailed functional descriptions of FAA build 2.0 of the WSP MIGFA. The document was written with many explicit references to data structures and routines in the actual software in order that it may serve as a useful algorithm development and programmers reference guide.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

LIST OF ILLUSTRATIONS

Figure No. **Page**

LIST OF ILLUSTRATIONS (CONTINUED)

LIST OF TABLES

1. ALGORITHM DESCRIPTION

1.1 Product Description

This document describes FAA build 2.0 (Lincoln Laboratory version wspmigfa-1-4) of the WSP MIGFA algorithm. The WSP MIGFA algorithm detects gust fronts and other wind shifts in ASR-9 WSP radar data and provides extrapolated position forecasts that can be used to predict when these events will impact an airport. In addition to identifying the locations of wind shift boundaries, MIGFA estimates the wind shift and the velocity change (wind shear) associated with each gust front detected and reports these data along with the current and extrapolated gust front locations. Although gust front detection is based on processing of WSP data, MIGFA also processes surface wind sensor data from an anemometer located near the airport in order to improve the accuracy of the wind shift reports.

1.2 Algorithm Overview

Gust fronts present "signatures" in Doppler radar imagery that can be recognized by sufficiently sophisticated automated processing algorithms. MIGFA uses knowledge-based image processing and data fusion techniques to recognize the three principal gust front signatures observed in the radar data:

Velocity convergence: Colliding air masses with different velocities produce a velocity convergence signature at the leading edge of a gust front. Velocity convergence is identified by a sharp decrease in Doppler velocity at increasing ranges along a single radar radial. A variety of situations can produce incomplete, distorted, or fragmented convergence signatures. Since gust fronts often propagate into clear air ahead of the generating storms, there may often be insufficient radar returns to produce this signature.

Thin lines: Gust fronts can be observed in radar reflectivity data as thin lines of increased reflectivity relative to background levels. The thin line is produced by dust, insects, rain droplets, or refractive index changes along the frontal boundary. Thin lines of coherent velocity values surrounded by random (noisy) velocities in low SNR clear air regions are another thin line signature that is often observed in ASR-9 WSP data.

Motion: Gust fronts move conspicuously in a direction perpendicular to the orientation of the convergence boundary and/or thin line.

Figure 1 is a block diagram illustrating general MIGFA processing and data flow. MIGFA is an iterative process that produces a new set of gust front detections, position forecasts, and associated wind shift and wind shear estimates for each scan of WSP radar data received. There are six principal steps involved:

1. Initialization and image preparation
2. Detection (interest generation)
3. Extraction
4. Analysis
5. Prediction
6. Output

An iteration of MIGFA processing begins when a complete scan of WSP base data product "radials" have been received (approximately once every 2 minutes). During the initialization stage, MIGFA takes the input WSP base data and computes a set of images (2-D arrays that represent visually interpretable data) and other auxiliary information. These data are collected into a GFBaseImages data object called "*gfBaseImages*" (descriptions of the various data classes used in MIGFA can be found in Appendix C).

Before proceeding with processing, the anemometer sensor data (e.g. from LLWAS centerfield or ASOS) that have accumulated during the preceding 2 minutes are retrieved from the input buffer and are appended to an *anemometer_history* which itself is contained in a GFRefData data object called *"gfRefData"*. The *gfBaseImages* and *gfRefData* objects serve as global data reference libraries that are globally accessible during various stages of processing.

After initialization, MIGFA executes a set of feature detectors, which are computational modules designed to detect specific physical signatures in radar data. Feature detectors in MIGFA are based on two techniques of knowledge-based image processing: Interest images and functional template correlation (FTC). Knowledge-based image processing is a mechanism of incorporating physical parameters of a search problem into image processing operations. An interest image is a data representation device, serving as a map of evidence for the presence at each pixel location of some feature that is selectively indicative of an object being sought. Higher pixel values reflect greater confidence that the intended feature is present at that location. Using interest as a "common denominator", data fusion is accomplished by combining the pixel-registered interest images produced by the various feature detectors into a single combined interest image. In MIGFA, interest values are mapped across an integer range, with 0 indicating strong evidence against a feature being present, 128 representing the ambiguity point, and 255 indicating maximum confirming evidence for the presence of a feature.

FTC is a generalized matched filter incorporating aspects of fuzzy set theory and producing interest images as output. Specific inputs to feature detectors include the newly received base data images contained in the *gfBaseImages* object as well as the *image_history* and *event_history*, which contain data from prior volume scans. These histories are maintained and stored in the global *gfRefData* object for convenient access by the algorithm. The interest images generated by the various feature detectors are then averaged to form a single combined interest image. Figure 2 shows an example MIGFA analysis display. In addition to displaying the input reflectivity and velocity base data images in the upper left, several derived data images and interest images are shown.

The extraction step takes the combined interest image produced by the detection stage and extracts a set of chains, each chain containing the set of points representing a single gust front. A chain is 1 pixel wide and can be any length. Points (pixels) within a chain are arranged in consecutive order along the length of the gust front. Once chain points have been extracted, an attempt is made to establish correspondence for each extracted point with some point from the collection of chains extracted in the prior scan. If correspondence is established, attributes of speed, direction of motion, and distance moved are computed and assigned to the point from the current volume scan. The extracted chains of points are assembled in the data structure called "*gfEvent*" (a GFEvent object).

The analysis stage performs a number of tasks on each extracted chain, adding additional information to *gfEvent*. These tasks include:

1. Editing of the chains of points encoded in *gfEvent*, attempting to enforce consistency along each chain by either changing the attributes of single points or by breaking chains into sub-chains. Attributes of speed, direction, and distance are then smoothed along the length of each chain.
2. Computing estimates of the various wind products associated with each detected gust front. These estimates are based on a consensus of several different estimation techniques.
3. Applying heuristics that adjust gust front interest scores (sum of interest values of all points in a gust front chain). Some of these heuristics look for behavior that is uncharacteristic of gust fronts and decrease interest scores accordingly. Others use wind shear estimates as a basis for modifying interest scores.

4. Making a final decision as to whether a gust front should be included in the algorithm output. The finished *gfEvent* is then added to the beginning of the *eventHistory* of the *gfRefData* object for use during the processing of subsequent volume scans. If the number of GFEvents in the *eventHistory* exceeds a parameter (nominally 12), then old excess GFEvent objects are removed from the end.

Predictions are generated by taking the information encoded in *gfEvent* and extrapolating the location and appearance of each gust front at incremental time steps in the future. These extrapolations are encoded in the list of predictions (a list of GFPredChain objects) and are stored in the *gfEvent* object.

Finally, the output handler outputs the contents of the *gfEvent* to the output stream. Among other formatting operations, image coordinates in pixels for each gust front curve point are translated into radar-centric coordinates in kilometers.

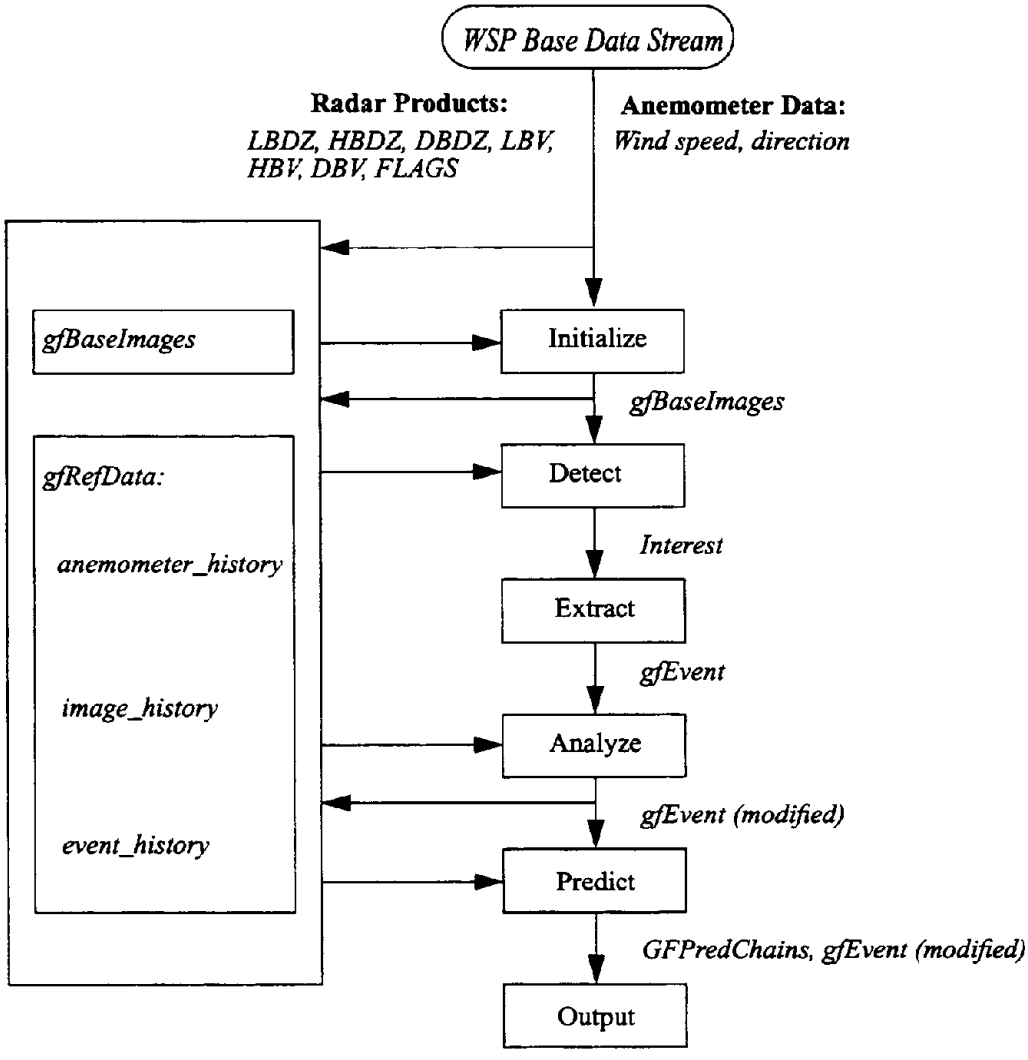


Figure 1. General process and data flow for the WSP MIGFA.

Figure 2. WSP MIGFA Analysis Display

1.3 Algorithm Inputs

Figure 3 shows the data interfaces to the WSP MIGFA. MIGFA processes input from four sources. Algorithm parameters are obtained from two sources: parameter disk files, and VSPs sent from the WSP System Control. Initial algorithm parameters are loaded from disk files using disk I/O. At any time during algorithm processing, a subset of the initial set of algorithm parameters can be received as VSPs from the system control and used to dynamically override the settings for those algorithm parameters.

There are two sensor data inputs: WSP base data (the primary data source used for gust front detection) and anemometer data (used to refine gust front wind shift and wind shear estimates).

1.3.1 Algorithm Parameter Specification

There are a number of algorithm configuration and parameter files that control MIGFA run-time behavior. When MIGFA executes, these files are found via UNIX environment variables that point to the location of each parameter file (this allows flexibility in installation of the parameter files). Table 1 lists the environment variables together with a brief description of the contents of the files. All except the "LOCAL" files are required in order for the algorithm to run. Appendix A contains printouts of the default parameter files.

The local parameter files provide a means for overriding default parameters to reflect site-specific configurations. Site adaptation could be performed by modifying any of the parameters in the default files. However, in most cases, only a few parameters are likely to change from site to site and it is preferable to not alter the default files. MIGFA reads the local parameter files after the default parameter files are read, so any parameter or configuration settings contained in the local files will override the ones in the default file. As opposed to the complete default files, the local parameter files need only contain entries for those parameters that need to be overridden. This helps to keep the local parameter files short, so it is easy to see which parameters have been adjusted for that particular site. Local parameter file settings are specified in the same Tcl language syntax used for setting the algorithm default values.

For example, the location of the airport with respect to the radar (arpOffsetXKM and arpOffsetYKM) needs to be specified for each site (since most ASR-9's are on the airport, the default offsets of 0,0 are often sufficient). These offsets are needed in order for MIGFA to know which segment of a detected gust front to analyze for generating the wind shift and wind shear hazard reports appropriate for the airport region that will be affected. If the airport is an appreciable distance away from the ASR-9, then these values need to be changed to indicate the number of kilometers east and north of the airport reference point (ARP) with respect to the radar. Use negative values to indicate positions west of south of the radar. This can be done by placing the following lines in the local configuration file pointed to by the GF_WSP_LOCAL_CONFIG_FILE variable (lines starting with a "#" in the first column are treated as comments):

```
#
# Airport offsets:
#
set arpOffsetXKM  -1.57
set arpOffsetYKM  -0.89
```

In the WSP MIGFA software, configuration parameters defined in GF_WSP_CONFIG_FILE and GF_WSP_LOCAL_CONFIG_FILE are loaded into a globally accessible GFIntGenConfig object variable called *gfConfig*. General algorithm parameters defined in GF_WSP_PARAM_FILE and GF_WSP_LOCAL_PARAM_FILE are loaded into a global GFParams object variable called *gfParams*.

1.3.2 VSP Stream

A subset of MIGFA parameters called Variable Site Parameters (VSPs) can be changed during program execution via a dedicated TCP/IP stream connection from the WSP System Control. This stream is checked during program initialization and is continuously polled after MIGFA enters its continuous processing loop. Whenever a new set of VSPs are detected on the stream, MIGFA retrieves the new settings and resets the appropriate parameters in its *gfConfig* and *gfParams* data structures, and processing proceeds with the new settings.

Table 1. WSP MIGFA Environment Variables

Variable Name	Description
GF_WSP_CONFIG_FILE	Default data resolution and airport geometry parameters used to configure data arrays and airport position-dependent heuristics.
GF_WSP_LOCAL_CONFIG_FILE	Optional "local" settings used to override settings in GF_WSP_CONFIG_FILE.
GF_WSP_PARAM_FILE	Default general algorithm parameters (over 150 of them).
GF_WSP_LOCAL_PARAM_FILE	Optional "local" parameter file used to override settings in GF_WSP_PARAM_FILE.
GF_DET_PARM_FILE_WSP	Parameters specifying which feature detectors are to be run and, for each detector, values for functional template kernels, scoring functions, interest weights, etc.
GF_EST_PARM_FILE	Parameters specifying which wind estimation methods are available and, for each wind analysis region, which estimators are to be used.
GF_HEU_PARM_FILE	Parameters specifying which heuristic sub-algorithms are invoked to reject improbable gust front features.
GF_LOG_CONFIG_FILE	Message logging facility configuration file used to direct text messages generated by MIGFA.

1.3.3 WSP Base Data

MIGFA receives a new set of base data products from the WSP base data generator once every 24 scans (approximately 2 minutes). Angular resolution of the base data products is 1.41 degrees (256 radials covering 360 degrees), while range extent of each base data product is 111.120 km (ranges 1 through 240 have a resolution of 115.75 meters/gate, while range gates 241 through 420 have a resolution of 463 meters/gate). Table 2 lists the WSP base data products sent to and processed by MIGFA.

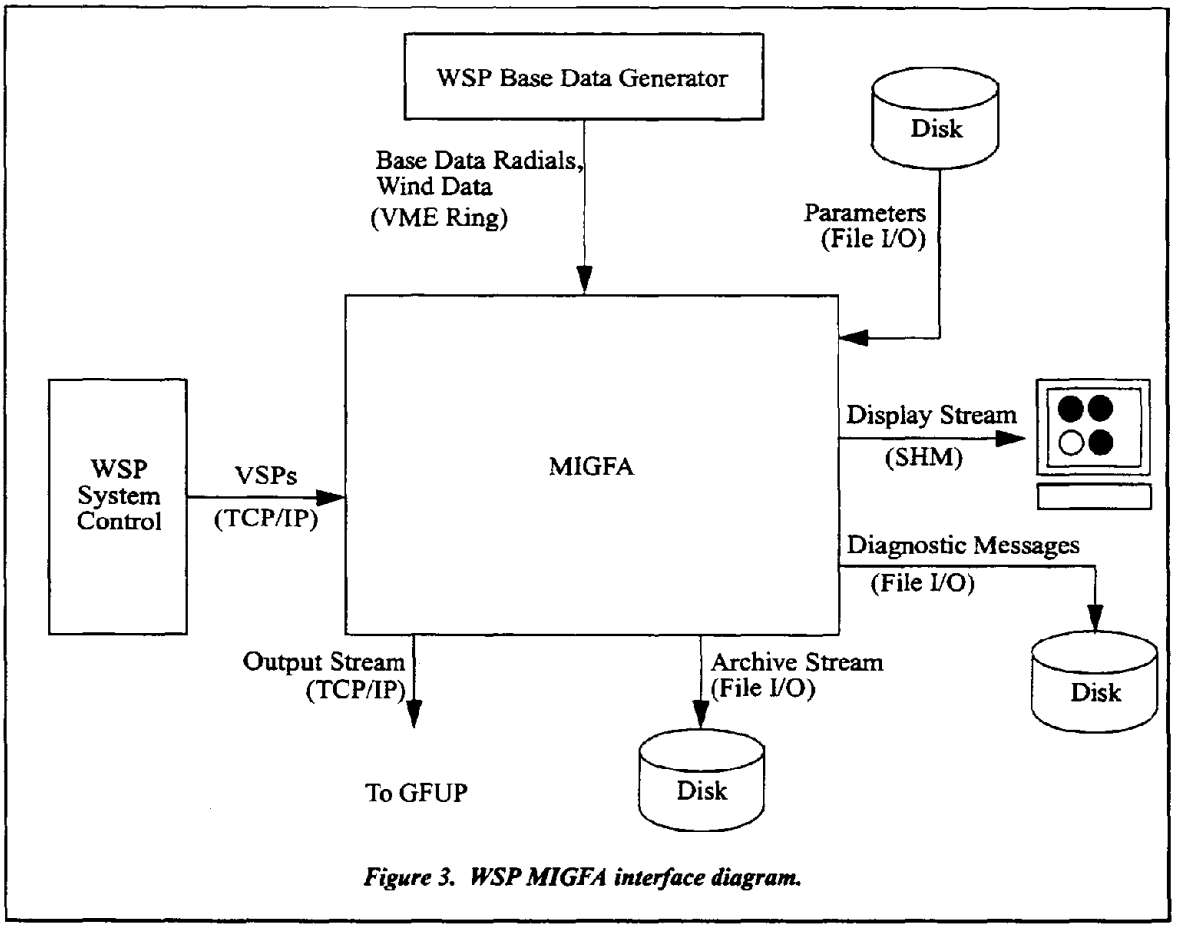


Figure 3. WSP MIGFA interface diagram.

1.3.4 Anemometer Data

If available, MIGFA can use airport wind data to improve the accuracy of its gust front wind estimates. The airport wind data are from a single anemometer (e.g., LLWAS centerfield, ASOS). If wind data is unavailable, MIGFA can continue processing without it (albeit with some likely degradation of wind shift and wind shear report accuracy). There are no time limits as to how old the data can be; data are internally weighted as a function of age and are consequently ignored if too old. Wind data is provided to MIGFA by the WSP Wind Data Server, which accesses whatever wind sensor is available and packs and transmits the time (GMT), wind speed (knots), and wind direction (degrees). When wind data are from the LLWAS centerfield, they are typically available once every 10 seconds. However, MIGFA makes no assumptions about the frequency of wind data input. It processes whatever it receives during its 2-minute processing cycle.

Although MIGFA does provide an option for obtaining airport wind data via a separate TCP/IP socket connection, the WSP Base Data Generator serves the wind data to MIGFA on the same VME ring buffer used to transmit the WSP radar base data products. The MIGFA driver is able to identify these wind data records when they appear on the input base data ring and handles them appropriately.

Table 2. WSP Base Data Products Processed by MIGFA

Product Name	Description
LBDZ	Low Beam Reflectivity (dBZ)
HBDZ	High Beam Reflectivity (dBZ)
DBDZ	Dual Beam Reflectivity (m/s)
LBV	Low Beam Velocity (m/s)
HBV	High Beam Velocity (m/s)
DBV	Dual Beam Velocity (m/s)
FLAGS	Data Quality Flags

1.4 Algorithm Outputs

As shown in Figure 3, MIGFA produces output on up to four output streams: the primary product output stream, an optional archival stream for storing intermediate and final product data, a display stream for **imgsh** graphics commands that generate the MIGFA diagnostic display, and diagnostic message output via the WSP message logging facility. Note that unlike some of the other WSP meteorological algorithms, MIGFA does not have a dedicated status stream. This is because MIGFA's output is actually an intermediate product that is further processed by the **GFUP** algorithm to produce the final gust front product. The **GFUP** algorithm recognizes and handles interrupts in MIGFA processing and generates appropriate status messages for the gust front product. Appendix B describes the MIGFA output record formats.

1.4.1 Output Product Stream

At the end of each processing iteration, MIGFA transmits gust front detection and forecast data together with their associated wind shift estimates to the product output stream via TCP/IP. This product data stream serves as the input to the **GFUP** algorithm which updates the gust front positions and performs final curve smoothing at 1-minute intervals and sends the final gust front product to the SD. The following information are transmitted:

1. WSP base data date/time (GMT).
2. Output data/time (GMT).
3. Unique gust front ID number for each gust front.
4. The u and v components of the propagation velocity of each gust front in m/s.
5. The u and v components of the wind behind each gust front.
6. The position (x,y) in kilometers relative to the radar, of the wind shift arrow for each gust front.
7. The delta-V (approximation of wind shear) value in m/s.
8. A list of (x,y) coordinates in km relative to the radar for each point of each gust front curve.

9. The number of prediction curves associated with the gust front detection as defined by the *gfParams->numPredictionTimes* parameter.

10. A set of gust front predictions for each front, each describing the predicted appearance of the gust front at each of the prediction time intervals defined in *gfParams->numPredictionTimes* (nominally one minute intervals out to 35 minutes in the future). Each prediction is a list of (x,y) coordinates in km relative to the radar for each point in each gust front. Each prediction is tagged with its time in the future (i.e., one of the times encoded in the *gfParams->predictionTimes* parameter) and the associated gust front ID number.

1.4.2 Archive Stream

An archival stream allows for separate archival of final and intermediate algorithm products. In addition to the final product data output to the standard product stream, the archival stream (through command-line arguments) can be configured to archive the interest images that were generated during algorithm processing. The archival stream is usually configured to send its output to a disk file, but it can be configured as an alternative TCP/IP stream, making its contents available to other real-time processes.

1.4.3 Display Stream

MIGFA has an optional diagnostic display that utilizes the Tcl/Tk-based **imgsh** image display package. If the **imgsh** display daemon is running, and the MIGFA diagnostic display option is enabled (via the MIGFA command-line), then MIGFA will send **imgsh** graphics commands to the **imgsh** display via a shared memory channel.

1.4.4 Diagnostic Message Logging

MIGFA uses the WSP message logging facility to direct the output of diagnostic messages [4]. A logging configuration file referenced by the GF_LOG_CONFIG_FILE environment variable is used to direct the output of various classes of diagnostic message output (informational, warning, error, and debug). Output of each class of message can be independently configured to be routed to a terminal, window, file, or discarded. In an operational run-time environment, the messages are directed to log files. A "dailyBackup" setting in the configuration file allows for automatic daily creation of new log files while the algorithm runs uninterrupted. The number of these files maintained on disk is controlled by the "numBackups" entry in the configuration file. See Appendix A.6 for an example configuration file for operational use.

2. FUNCTIONAL DESCRIPTION

Figure 4 is an abbreviated call hierarchy showing the major function calls in the current algorithm implementation. MIGFA draws heavily on routines from the CSKETCH library. The CSKETCH library contains image processing functions (such as methods for performing functional template correlation) and methods for manipulating arrays of data via the *SKArray* class. *SKArrays* are used to store and manipulate much of the radar data and interest images processed and generated by MIGFA.

2.1 Initialization

GFLoadParams loads default and local algorithm parameters from files specified by UNIX environment variables as described in section **1.3.1**. Next, reusable temporary data storage for buffering the input WSP base data radials is allocated by **AllocTempArrays** and the array values are initialized to *nil* by calling function **InitTempArrays**. *Nil* is a special value recognized by CSKETCH array routines that is used to indicate missing or neutral data that should not be processed.

GFAllocObjects performs a variety of initialization functions. First, it allocates memory for the various SKArrays that are part of the *gfBaseImages* object. Additionally, some SKArrays that are exclusively used by the WSP version of MIGFA are allocated and stored in a separate *gfBaseImagesWSP* object. The various SKArrays hold image representations of the input WSP base data as well as derived image data, such as Cartesian mappings of the base data, and various mask arrays used during FTC. Two different *GFIntGenConfig* objects (*gfConfig* and *gfConfigLR*) are used as parameters in constructing the various SKArrays. The global *gfConfig* object is initialized when the GF_WSP_CONFIG_FILE and GF_WSP_LOCAL_CONFIG_FILE are loaded at start-up, and contains the geometry specifications that dictate the allocated sizes for the standard 30 km range polar and Cartesian image arrays used for the bulk of the gust front detection process. Long range (111 km) arrays are allocated after calling **GFFillIntGenConfigLR** to fully initialize the *gfConfigLR* object. The granularity for the long range Cartesian images is set to 4 times that of the standard granularity given by *gfConfig->metersPerPixel*. Typically, this means that the long range images will have a resolution of 1920 meters per pixel, compared to 480 meters per pixel for the standard 30 km images.

A resampler object (*gfResamp*) that is used to map polar data to Cartesian image representations is created and initialized using the geometry parameters contained in the *gfConfig* object.

Next, the GF_EST_PARM_FILE is parsed and the appropriate wind estimator objects are created and put on globally accessible lists called *windAheadEstimators, windInsideEstimators, windBehindEstimators, deltaVEstimators,* and *maxDeltaVEstimators*. Finally, the GF_HEU_PARM_FILE is parsed and the specified heuristic functions are put onto the global list variable *heuristics*.

GFFillStaticObjects generates various static mask arrays that are used to mask regions of input images from interest generation during FTC. The mask arrays are added to the *gfBaseImages* object for later reference by the feature detectors. Figure 5 shows examples of each of the static mask images described below.

The *dataMask* defines a circular region that has a radius equal to the processing range (defined by *gfConfig->cartRadiusKM*). Pixels outside of this area are set to *nil*.

The *detMask* is identical to the *dataMask* except that the radius of the non-*nil* (white) area is 0.5 km smaller. The *detMask* is the more appropriate mask to use for FTC-based interest generators, since it offers more "padding" to prevent edge effects. The *dataMask* is used to provide a neutral border for other non-FTC interest images, such as the anticipation interest image.

The *siteMask* defines a small circular "bulls-eye" of "1"s located at the center of the image and having a radius defined by *gfConfig->siteMaskRadiusKM* (nominally 2 km). All pixels outside of the *siteMaskRadiusKM* are set to *nil*.

The *detSiteMask* is the same as *detMask* except that the *siteMask* has been used to create a "bulls-eye" of *nil* values in the center of the image. The *detSiteMask* or an equivalent explicit combination of *detMask* and *siteMask* is used by most of the feature detectors to prevent opinions being expressed in the immediate vicinity of the radar.

The *eightKmMask* defines an 8-km processing radius. It is used by the Anticipation detector to boost interest within 8 km of the radar and by the HIDZ high-altitude weather detector to prevent opinions at short range where discrimination is poor.

Next, **OpenOutputs** is called to open the output product and archive streams and **OpenInputs** is called to connect to the input WSP base data shared memory ring and to establish the input TCP/IP connection to the anemometer data stream. **OpenInputsWSP** is a WSP-specific function that opens the TCP/IP connection for receiving MIGFA VSPs from the WSP System Control. Before going into the main processing loop, MIGFA repeatedly calls **PollControlStream**, waiting for an initial set of algorithm VSPs to arrive (WSP system control is designed to send the current set of VSP's to any newly connecting client or whenever the VSPs are changed).

2.1.1 Handling of Input Data in the Main Processing Loop

Next, the main algorithm processing loop is entered. MIGFA reads base data records (*BDRecords)* from the shared memory input data ring buffer via calls to **RingBufReadDMA** (for real-time processing on the WSP system) or **RingBufGet** (for offline processing using a base data archive translator process). A *BDRecord* appearing on the MIGFA base data ring is either a radial of WSP radar base data (type = *BD_MSG_RADIAL*) or a wind data record (type = *BD_MSG_WIND*). If the data record is a wind record, then **ProcessWindRecord** is called to retrieve the data. The anemometer data is appended onto the anemometer history (*gfRefData->anemometerHistory*). The size of the anemometer data history is limited by *ANEMOMETER_HISTORY_SIZE*, which is currently set to 1440 data records. For the LLWAS centerfield wind data rate of 10 seconds, this translates to a history size of 4 hours. Once the maximum history size has been reached, old data records are dropped off the end of the history as new data records are appended.

If the *BDRecord* received is of type *BD_MSG_RADIAL*, then MIGFA calls function **CopyRadialProdData** to copy the radar data into the 3-D data buffer *inProdData*. The data for each *BD_MSG_RADIAL* are contained in a WSP *Radial* data structure (see Figure 6). The C-language definition for the *Radial* data structure can be found in the file *baseData.h* in the *share/inc* directory (see also Newell, 2000). A *Radial* is simply a contiguous block of bytes consisting of some basic header information (time stamp, volume scan number, tilt number, azimuth angle, elevation angle, etc.) followed by a data block containing all of the radar base data products for that radial. This data block is logically partitioned as a sequence of base data products (e.g., reflectivity, velocity), which themselves are logically partitioned into a product header (*ProdHdr*) followed by the product data.

The range gate spacings for the product data are defined in the *ProdHdr* block of each product. For the WSP, each base data product has two range gate spacings (segments). The first 240 gates have a gate spacing of 115.75 meters (this is the fundamental gate spacing), while the following 180 gates have a larger gate spacing equal to 4 times the fundamental gate spacing or 463 meters per gate. In order to copy the data into MIGFA's *inProdData* array, product data values from the second range segment are replicated in a 1-to-4 fashion in order to simulate the 115.75 meter gate spacing of the first range segment. The *inProdData* buffer is dimensioned as a short integer array of size 7 products x 256 radials x 960 gates in order to accommodate all of the radar base data from a single scan.

As each radial is read, **CheckPPIStatus** is called to check for the end-of-tilt flag indicating that the last radial for the current scan has been received. If the status return is *SCAN_COMPLETED*, then processing of the scan commences.

```
BEGIN
        GFLoadParams
        AllocTempArrays
        InitTempArrays
        GFAllocObjects
        GFFillStaticObjects
        OpenOutputs
        OpenInputs
        OpenInputsWSP
        PollControlStream

        While ( continue processing )
                RingBufReadDMA (or RingBufGet) to obtain next radial of base data
                CopyRadialProdData
                CheckPPIStatus

                If ( complete scan received ) then
                        GFFillDataObjects
                        InitTempArrays
                        SetSensitivityMode

                        If ( first scan ) then
                                GFAllocDetectors
                        end if

                        PollControlStream
                        ProcessAnemStream

                        GFPreProcessImages
                        GFFillSingleScanObjects

                        RunDetectors
                                SKAverageInterestImages
                        smoothThinLine->RunDetector

                        GFExtract

                        GFAnalyze
                                EditGustFronts
                                ProcessWindAnalysis
                                GFApplyHeuristics
                                MakeReportDecision
                                MakeDisplayPredictions

                        GFBuildImageHistory

                        gfConfig->outHandler

                End if ( complete scan received )
        End while ( continue processing )

        CloseInputs
        CloseInputsWSP
        CloseOutputs
END
```

Figure 4. Abbreviated WSP MIGFA call tree (see text for description)

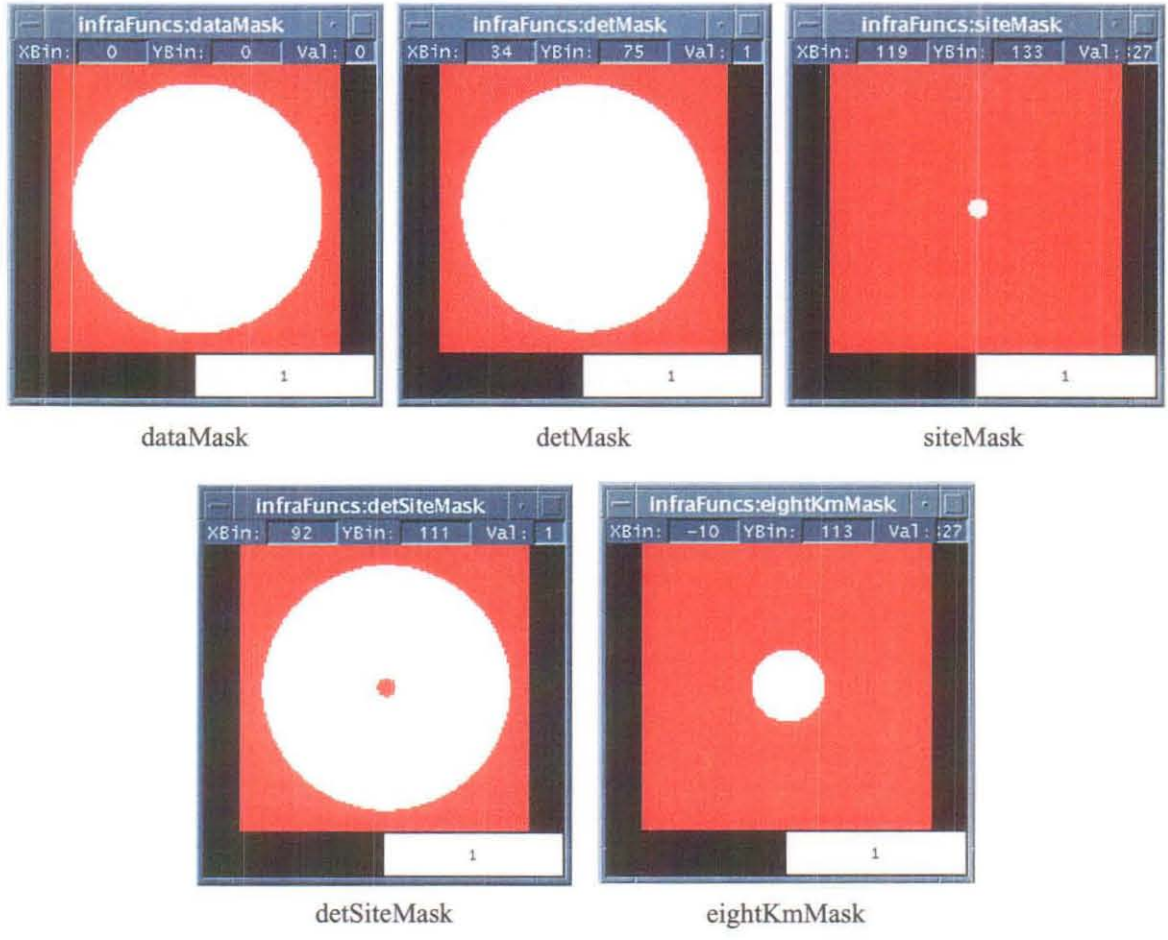


Figure 5. Static binary mask images used during interest generation. Red areas are "NIL" values. White areas have a pixel value of "1".

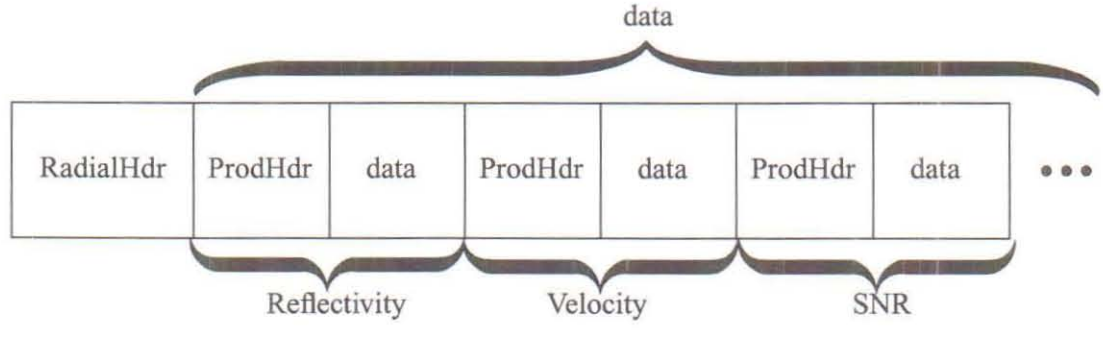


Figure 6. Schematic illustration of WSP radial data record organization

2.2 Image Preparation

After receiving a complete scan's worth of radials, the data are copied from the input data buffer (*inProdData*) into the image SKArray objects that MIGFA will process. Some preprocessing of the data is also done. While this is the primary activity in the image preparation phase of algorithm processing, this is also the stage where the algorithm control stream is checked in order to acquire any VSP changes that may have been issued from the WSP Maintenance Display Terminal (MDT).

2.2.1 Filling the SKArray Base Data Objects

Once all of the radials for a complete gust front scan have been received, image preparation begins. First, function **GFFillDataObjects** is called to copy the WSP base data products that were buffered in the *inProdData* array into the SKArray objects whose reusable storage was pre-allocated by function **GFAllocObjects** during algorithm initialization. The product data values that are received into the *inProdData* buffer and stored in the SKArrays are integer-scaled representations of their true floating point values. The following equation expresses the relationship between the actual and integer-scaled representations of the data:

$$\text{scaledValue} = (\text{actualValue} + \text{offset}) * \text{scaleFactor}$$

The scale factor and offset values used by the WSP base data generation algorithm to encode each base data product are included as part of the base data information that MIGFA receives. For later decoding, these quantities are stored along with the image array data in each SKArray object.

All of MIGFA's functional template computations are performed on integer-scaled representations of the physical image data. The FTC scoring functions (whose nodes are defined in the algorithm detector parameter file) are expressed in terms of integer-scaled values having assumed scalings and offsets, not the actual physical values, and not necessarily the data scalings that were transmitted to MIGFA. **GFFillDataObjects** ensures that the SKArray data that will be subsequently processed are represented with the expected scalings by rescaling the received polar reflectivity and velocity data to the expected scalings via the **SKArrayRescale** function. The expected scalings are given in Table 3:

Table 3. Base Data Scalings Expected by MIGFA Scoring Functions

Product Type	Scale Factor	Offset
Reflectivity (*lbdzPol, hbdzPol, dbdzPol, dzLongPol, lbvLongPol*)	2.0	20
Velocity (*lbvPol, dbvPol, lbvLongPol*)	100.0	100

In addition to storing the scaling information in each SKArray, **GFFillDataObjects** stores the time stamp associated with this interval of processing. This time stamp is the time associated with the last radial received in the current scan. Finally, **GFFillDataObjects** copies the *dbdzPol* data to *gfBaseImages->dzPol* and *lbvPol* to *gfBaseImages->vPolRaw*. This is done so reflectivity and velocity-based feature detectors can be written to utilize common, expected image names for different radar systems.

2.2.2 Setting the Seasonal Sensitivity Mode

Several of the WSP MIGFA feature detectors employ season-specific logic that is automatically invoked depending on the date of the scan being processed. Currently, only two seasons are recognized: "summer" and "non-summer". Function **SetSensitivityMode** sets the *gfRefData->summerMode* boolean variable to

"true" or "false" by comparing the current scan date against the start and end dates of the summer season as defined by the "summerMonths" parameter in the default *paramsWSP* file or the site-specific local *paramsWSP.local* file. The default values for the summer season are 4/1 - 9/30.

2.2.3 Allocation of Feature Detectors

Creation (allocation) of the feature detector objects is accomplished by calling function **GFAllocDetectors** upon receipt of the first scan. **GFAllocDetectors** reads a Tcl-language detector parameter file pointed to by the GF_DET_PARM_FILE_WSP environment variable. Near the top of the parameter file, is a setting for a parameter named *"Detectors"*. *Detectors* is a Tcl list variable that contains the names of all possible feature detectors that can be run during the course of MIGFA processing. For each detector named in the *Detectors* list, **GFAllocDetectors** dynamically creates the detector object by calling its constructor.

A subset of these detectors will eventually be run by the **RunDetectors** function; their interest images will be averaged together to form the combined gust front interest image. **GFAllocDetectors** stores pointers to these specific detector objects in one or both of two linked list variables: *detectorListStatic* and *detectorListMotion*. **RunDetectors** (called by the MIGFA driver that is common to all radar system variants) will iterate over all of the detectors that it finds in *detectorList* -- a local variable that is set to *detectorListStatic* for the first scan (when motion-based feature detection is not yet feasible) and set to *detectorListMotion* for all subsequent scans. Because the WSP motion-based feature detectors were designed to perform their own internal checks regarding availability of prior images (and to modify their output accordingly), the WSP version of **GFAllocDetectors** actually puts the same detectors onto both the *detectorListStatic* and *detectorListMotion* lists. The detectors that are currently part of these lists are (listed by their Tcl parameter name): Anticipation, Converge, ConvMotion, DZMotion, DZSDMotion, HIDZ, SDMotion, TLDZ, TLSD, TLSDDZ, and PrecipEdges. Details of these detectors are described in the next section.

It is important to note that the order of some of these detector names in the Tcl parameter file is significant. For the subset of detectors that will comprise the *detectorList*, execution follows the order in which they are listed in the parameter file. Some detectors require the results of other detectors to function properly. For example, the high altitude weather detector (HIDZ) is prohibited from generating interest in areas where there are line storms or where there is short or long-term anticipation of previously detected gust fronts (as indicated in the Anticipation interest image). Table 4 lists the feature detector dependencies.

Table 4. Feature Detector Dependencies

Feature Detector	Requires Output Of:
Anticipation	LineStorm
HIDZ	LineStorm, Anticipation
TLDZ	DZMotion
TLSD	DZMotion
SDMotion	DZMotion
Converge	ZeroCross
ConvMotion	ZeroCross
DZSDMotion	DZMotion
PrecipEdges	Anticipation, LineStorm, Converge, ConvMotion

2.2.4 Processing the VSP Control Stream

Before gust front feature detection begins, all pending data on the input VSP control stream are processed. Function **PollControlStream** processes the VSP control stream by reading any parameter data records that may have been transmitted from the WSP MDT since the last MIGFA processing iteration. Values of parameters retrieved from this stream override the values that were obtained from the MIGFA parameter files loaded at process startup. Table 5 lists the MIGFA parameters whose values can be overridden by VSPs.

Table 5. WSP MIGFA Variable Site Parameters (VSPs)

Parameter Name	Description	Default Value
siteMaskRadiusKm	Radius in km of mask used to exclude pixels over the radar	2.0 km
arpControlRadiusKM	Distance from airport inside which airport-localized wind analyses are performed	15.0 km
snThreshold	Reflectivity threshold below which velocity data is rejected	-5 dBZ
gfMinScore	Minimum total chain interest score for a candidate gust front	3000
minRawShapeLength	Minimum chain length in pixels needed to be considered as a candidate gust front detection	12.5
minIndexedChainLength	Minimum # of indexed points for a chain to be reported	11

2.2.5 Configuring Images for the Analysis Display

In order for an SKArray image to be displayed using the imgsh-based MIGFA analysis display, the image has to be added to the global *imageListAll* list variable in the MIGFA code. A WSP-specific implementation of function **GFSetDisplayList** is called from the MIGFA driver to create an initial list of images that may be displayed. This initial list does not include the individual feature detector interest images; these are added shortly thereafter by calling **GFCreateDetectorDispList**, which loops over the list of detectors in the *detectorList* variable, and adds the pointer to each detector's interest image to the *imageListAll* list.

If it is desired to add additional non-feature detector images to *imageListAll*, then **GFSetDisplayList** has to be modified to add the specific image to the list. Note that for every image that will be displayed, a corresponding entry in the imgsh MIGFA display script (*migfaDisplay*) must also be made. Specifically, configuration information for the image must be added to the *pinfo* array variable in the *migfaDisplay* script, and the name of the image must be added to the *defaultDispList* variable of the *migfaDisplay* script.

2.2.6 Preprocessing of the Reflectivity Data

A WSP-specific implementation of **GFPreProcessImages** is called to reset all missing data values in *gfBaseImages->dzPol* to an integer-encoded value corresponding to -20 dBZ. This allows FTC operations for thin line detection to treat missing reflectivity pixels as "clear air" returns of -20 dBZ (rather than ignoring the pixels).

2.2.7 Generation of "Single Scan" Objects

Function **GFFillSingleScanObjects** performs preliminary computations of numerous derived quantities and data arrays that change for each new scan of data and are needed for subsequent gust front feature detection and wind analyses.

2.2.7.1 Resampling of polar images to Cartesian images

The majority of FTC-based feature detection in MIGFA is conducted on Cartesian representations of the data. In function **GFFillSingleScanObjects**, SKArrays containing the polar representations of reflectivity, velocity, and flags are resampled to Cartesian SKArrays using the global *gfResamp* resampler created and configured earlier in **GFAllocObjects**. This polar-to-Cartesian resampler class is a "last-in" resampler. That is, if multiple range-azimuth cells of a polar array map to a single Cartesian resolution cell in the output image, the last polar array value retrieved is the one that is stored in the Cartesian output cell.

Both standard range (30 km) and long range (111 km) polar images are resampled to Cartesian images. The *gfResamp* object is used for resampling the standard range polar images into Cartesian images using the parameters that are defined in the global *gfConfig* object (*radialsPerScan, gateSizeMtrs, metersPerPixel, cartRadiusKM).* For resampling the long range images, **GFFillSingleScanObjects** creates and configures a local *SKResamp* object called *gfResampLR* using the parameters of the *gfConfigLR* object that was defined in function **GFAllocObjects.** Note that the *metersPerPixel* element of the *gfConfigLR* object is set to four times that of the one defined for *gfConfig* (nominally 480 m), so the resulting long range Cartesian image resolution is nominally 1920 m.

2.2.7.2 Retrieval of prior images

Images from prior processing iterations are needed for discerning the motion of gust front features in the radar imagery. Retrieval of base images from the prior scan is accomplished by a call to function **GFGetBaseScan**, which examines the image histories (*dzHistory, vHistory, convHistory, sdHistory*) in the global *gfRefData* object, and searches for a prior image whose time stamp is *gfParams->priorInterval* seconds prior to the time of current scan within an acceptable time window given by plus or minus *gfParams->priorIntervalTolerance*. If a matching image from the image history is found, then the prior image is stored in the global *gfBaseImages* structure for later access by feature detectors that require previous scan images in order to detect motion.

2.2.7.3 Estimation of airport, site, and regional winds

The "airport" wind estimate (*gfEvent->airportWind*) is computed from the airport anemometer data history by calling function **GFComputeAirportWind**. The wind estimate is computed as a weighted average of the prior and current anemometer measurements, with the weight of each measurement being proportional to the age of the measurement with respect to the current time (a relatively lower weight is assigned for older wind measurements). The airport wind estimate is used later during wind analysis to help estimate the wind shift (for outbound gust fronts) and the wind shear hazard when a front is within the vicinity of the airport.

The "site" wind (*gfEvent->siteWind*) is intended to be a radar-derived estimate of the winds in the vicinity of the radar site. For radar systems other than the WSP, MIGFA computes the site wind using a simplified optimal estimation (OE) technique that is essentially a VAD analysis of a ring of Doppler values surrounding the airport. However, outside of precipitation regions, the WSP often lacks sufficient sensitivity to provide enough velocity data to support reliable estimation of the site wind in this fashion. Because of the difficulty in estimating the winds from the radar data, and given the fact that most (if not all) ASR-9's are located on the airport, the WSP MIGFA simply sets the site wind (*gfEvent->siteWind*) to be the same as the airport wind (*gfEvent->airportWind*) that was derived from anemometer measurements.

The "regional" wind is a radar-derived estimate of the mid-altitude steering winds within the radar coverage area, and is used to adjust sensitivity either globally or within specific feature detectors. It is computed in function **GFComputeVAD** using a least-squares optimal estimation technique operating on a "cleaned" version of the polar long range velocity image. First, a cleaned velocity image is obtained by calling **ComputeSDImage** to compute a long range velocity standard deviation image

(*gfBaseImagesWSP->sdLongPol*). This image is accessed by function **GFComputeVAD**, which immediately calls function **GoodSnrLBV** to edit those pixels in *gfBaseImages->lbvLongPol* that have:

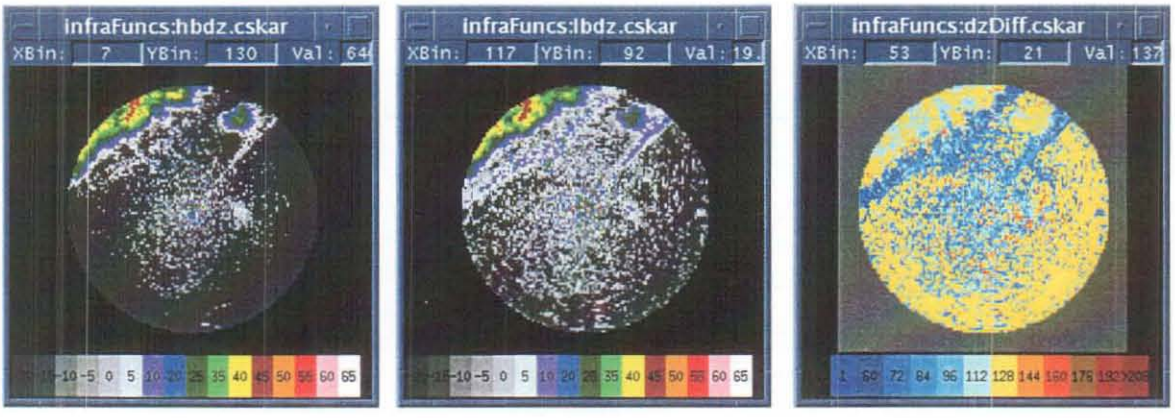
1. Low SNR as inferred by a high standard deviation value in *gfBaseImagesWSP->sdLongPol*
2. Low SNR as inferred by low reflectivity (*gfParams->snThreshold*)
3. Clutter, AP, or out-of-trip weather as flagged in *gfBaseImagesWSP->flagsLongPol*

Next, **GFComputeVAD** creates a local resampler object and uses it to resample the cleaned long range polar velocity image to a 1920 meter resolution long range Cartesian image. In order to arrive at a wind estimate that is representative of the mid-altitude steering winds, an initial preferred annulus ranging from 20 km to 80 km is chosen as the analysis region. A list of velocity samples is assembled by subsampling the Cartesian image at an interval given by the *gfParams->regionalWindSampleSpacing* parameter, and ignoring any pixels that have an invalid velocity value (*nil*) or that fall outside of the annular analysis region. As the list of velocity samples is accumulated, a tally is kept of the number of valid velocity samples extracted from each of four azimuth quadrants: NE, NW, SW, and SE. If any one of the quadrants has no valid velocity samples within the preferred annulus, then the annulus is fattened to the range 0 km to 90 km, and a new attempt is made to find valid velocity points in all quadrants (the initial list of velocity samples is deleted, and a new list is created from the subsampling of the points in the fattened annulus).

The final list of retrieved velocity samples will be supplied to an optimal estimation procedure that is computationally intensive. The list of retrieved velocity samples is first checked to see if the number of samples is less than *gfParams->regionalWindMaxNumSamples* (nominally, 64). If there are too many samples, then the list of retrieved velocity samples is further subsampled as required to bring the number of samples under the limit. This final *prunedRadarList* is a list of GFRadarOB data structures, each of which stores location (x, y, azimuth) and Doppler velocity information for the radar observation. If there are at least two observations, then the *prunedRadarList* is passed to function **OptimalEstimation** to estimate the regional wind speed and direction utilizing a least-squares fit to a uniform wind model assumption. **OptimalEstimation** returns a GFWind object that contains the estimated wind velocity along with weight factors that reflect the confidence in the fit. The new regional wind estimate (*newRegionalWind*) that is finally returned by **GFComputeVAD** is obtained by averaging the current wind estimate as provided by **OptimalEstimation** together with the prior regional wind estimate stored in the event history (*gfRefData.eventHistory*). Note that *newRegionalWind* may be *NULL* if there are fewer than two velocity samples or if optimal estimation fails due to noisy velocity data.

2.2.7.4 Computation of high/low beam reflectivity difference image

Low reflectivity echoes from light rain showers often mimic gust front thin line echoes and are a leading cause of false detections. By comparing the reflectivity returned from the high beam of the ASR-9 against that seen in the low beam, it is often possible to discriminate high-altitude weather features from the lower altitude thin line echoes associated with actual gust fronts. This discrimination capability is provided by the high-altitude weather feature detector (HIDZ), which operates on an image representing the difference between the high-beam reflectivity and the low-beam reflectivity. The reflectivity difference image is computed by function **ComputeDZDiff**, which first performs a pixel-wise subtraction of the low-beam reflectivity polar image array values (*gfBaseImages->lbdzPol*) from those of the high beam (*gfBaseImages->hbdzPol*). The resulting integer-scaled dB difference values are truncated to the interval [-128, 127], and then shifted by +128 to the interval [0, 255], so that a value of 128 represents a reflectivity difference of zero. The scoring functions for the HIDZ detector are designed to operate on this shifted integer-scaled representation of the reflectivity difference. The polar dB difference image is then resampled to a Cartesian difference image using the *gfResamp* object, and the result is stored in *gfBaseImages->dzDiff* for later reference. See Figure 7.



(a) (b) (c)

Figure 7. Example computation of high/low beam reflectivity difference image: (a) High beam reflectivity (dBZ). (b) Corresponding low beam reflectivity (dBZ). (c) Integer-scaled reflectivity difference after shifting by +128. Blue colors indicate areas where low beam reflectivity exceeds the high beam (values < 128), and yellow/red colors indicate areas where high beam reflectivity exceeds the low beam (values > 128). Data are from Austin, TX at 17:57:46 GMT

2.2.7.5 Creation of clutter and AP images

Images denoting regions of clutter residue and AP are used for masking purposes in several of the WSP MIGFA feature detectors. The WSP sets a clutter flag in the FLAGS product wherever weather-to-clutter residue ratio is too low to guarantee an unbiased weather estimate [2]. Function **ExtractFlagImage** is called to create an initial binary image indicating the presence of clutter residue from the *gfBaseImages->flags* array. Because the weather-to-clutter residue ratio test often fails in areas of low SNR, the WSP often sets the clutter flag in these areas even though there is not a substantial amount of clutter residue. In order to retain these areas for gust front processing, the initial binary clutter map image is thresholded so that only clutter regions associated with reflectivity greater than 10 dBZ are retained in the clutter map image. Finally, the reflectivity-thresholded binary clutter image is subjected to a 3x3 binary "close" operation which fills small holes in the binary clutter regions and slightly erodes the filled regions. The clutter map image is stored in *gfBaseImages->clutterMap* for later reference. An example binary clutter map image from Albuquerque, NM is shown in Figure 8.

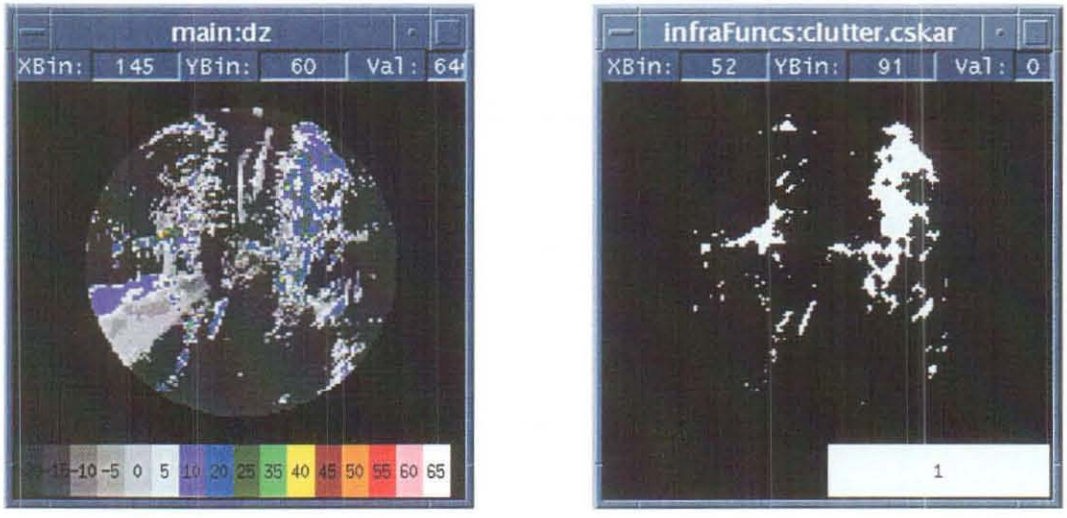


Figure 8. Example of binary clutter map image computed from the WSP FLAGS data (right). The corresponding reflectivity image is shown on the left. Images are from data collected on 7/7/98 at Albuquerque, NM

Areas of anomalous propagation (AP) are also excluded from some detector processing. Again, **ExtractFlagImage** is called to identify areas in the *gfBaseImages->flags* array where the AP flag is set and to create an initial binary map of AP. The initial AP map is dilated using a 3x3 binary "open" operation which has the effect of removing isolated pixels of AP and then slightly dilating the remaining regions. The result is stored in *gfBaseImages->apMap* for later reference.

2.2.7.6 Computation of velocity standard deviation images

An image representing the local standard deviation of the velocity is used as input for feature detectors that can identify a thin line of coherent velocity values against a field of noisy clear air velocity values. The standard deviation is computed over a 3x3 window for each pixel in the polar *gfBaseImages->vPolRaw* image (low beam velocity) by calling function **ComputeSDImage**. Before returning the velocity standard deviation image, **ComputeSDImage** truncates the integer-scaled standard deviation values to the range [0, 2735] (0 to 27.35 m/s), and then linearly remaps the values to the interval [0,255]. Remapping the data in this fashion allows for more precise definitions of scoring functions over a smaller range of values. The scoring functions for the various feature detectors that process the velocity SD image are therefore matched to this range of velocity SD values. The polar velocity SD image is stored in *gfBaseImages->sdPol*. A Cartesian image of the polar velocity SD data is then computed using the *gfResamp* object and the result is stored in *gfBaseImages->sd*.

The velocity SD feature detectors are very sensitive. They utilize high amplitude scoring functions that can generate high scores at the data edges where the *nils* returned during FTC from the outer detector masking region are not sufficient to offset extremely high scores when optimal matches are found at pixels just inside the detector mask. This can lead to bands of moderately positive interest values on the edge of the interest image. To mitigate this "edge-effect", the *nils* in the outer detector mask region of the *gfBaseImages->sd* image are replaced with a value of "32". This serves to return offsetting negative scores where the template kernel extends beyond the edges of the actual data. The trade-off for this is that there is a moderately reduced sensitivity to velocity thin lines at the very edge of the detection range.

2.2.7.7 Computation of "cleaned" velocity image

An image of "good" quality velocity data is needed for derivation of a good quality shear map whose values can be directly used as estimates of the delta-V associated with a detected gust front. Function **FillDBVPol** is called by **GFFillSingleScanObjects** to compute a "cleaned" velocity image from the input DBV data. Originally, this function filled missing values in the DBV field with corresponding LBV product data (hence the function name), but this was found to occasionally cause sharp radial velocity discontinuities that sometimes resulted in false convergence detections. Velocity values having insufficient SNR or associated with clutter, AP, or out-of-trip contamination are rejected in this step.

SNR criteria are indirectly applied through two tests. The first test checks the local velocity standard deviation as contained in the previously computed *gfBaseImages->sdPol* image, and if the standard deviation exceeds a threshold of 125 (approximately 13.4 m/s), the velocity datum is rejected. The second test checks the associated reflectivity value in the *gfBaseImages->dzPol* image, and if the reflectivity is less than *gfParams->snThreshold*, the velocity datum is rejected.

If the *gfConfig->doVMasking* parameter is enabled, then velocity data are rejected if they are associated with areas of excessive clutter residue, AP, or out-of-trip contamination as indicated in the *gfBaseImages->flagsPol* image. Note that in the case where the cleaned velocity is being computed from the *gfBaseImages->dbvPol* data (as is the case at the time of this writing), this editing is largely redundant with what has already been applied during generation of the DBV data by the WSP signal processing algorithms. The current WSP data quality algorithm tends to be over-zealous in flagging clutter residue in areas of low signal-to-noise. To avoid throwing out potentially good velocity measurements, the clutter flags are ignored if the associated reflectivity exceeds 10 dBZ.

The velocity data originally stored in *gfBaseImages->dbvPol* are overwritten with the resulting edited velocity data computed and returned by **FillDBVPol** (before being overwritten, the *dbvPol* data are first resampled by the *gfResamp* object to a Cartesian image and stored in *gfBaseImages->dbv* for future reference). The *gfResamp* object is then used to create a Cartesian representation of the cleaned velocity from *gfBaseImages->dbvPol* which is stored in *gfBaseImages->vClean*. The convergence detectors operate on the *gfBaseImages->vClean* image.

2.2.7.8 Computation of out-of-trip weather image

Images denoting regions of range-ambiguous, or out-of-trip echoes are used for masking in several of the WSP MIGFA feature detectors. Owing to the WSP pulse-pair velocity estimation technique, velocities in areas of out-of-trip echoes exhibit a random +/- Nyquist velocity incoherency that is also seen in regions of insufficient SNR. Thus, out-of-trip can easily be recognized in the WSP base data imagery as relatively narrow, radially aligned wedges of low reflectivity that correspond to areas of high local velocity variance.

The out-of-trip image is computed by invoking the **outOfTripDetector->RunDetector** function directly. This tandem functional template simultaneously probes the *gfBaseImages->dzPol* and *gfBaseImages->sdPol* images to generate a polar interest image of out-of-trip. This polar interest image is then resampled to a Cartesian interest image using the *gfResamp* object, and a gray scale dilation with a 3x3 kernel is applied before returning the final out-of-trip interest image. The interest image is stored in *gfBaseImages->outOfTrip* for later reference. Figure 9 shows an out-of-trip image computed from corresponding reflectivity and velocity images. Figure 10 shows the scoring functions and template kernels for this detector. Note that the kernels only need to be applied at a single orientation (90 degrees) since they are operating on polar images where each row is a radial of data, and the out-of-trip features are aligned with radials.

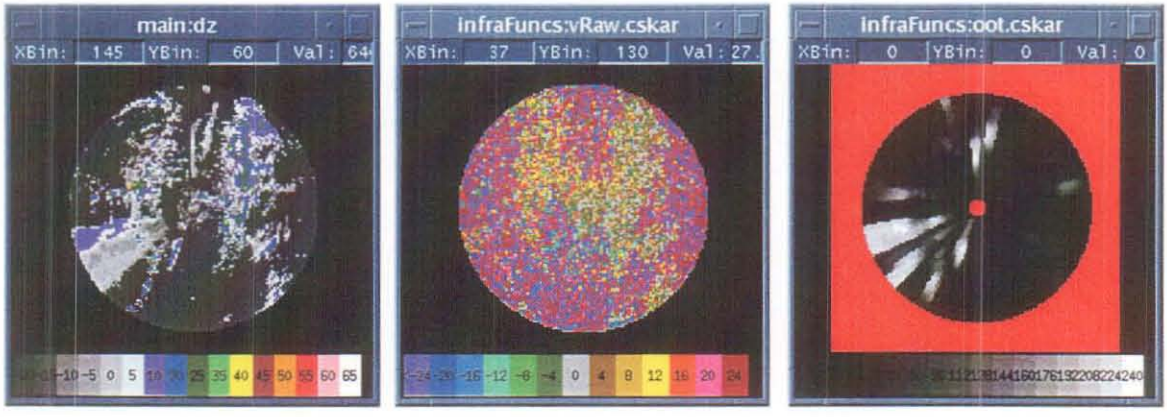
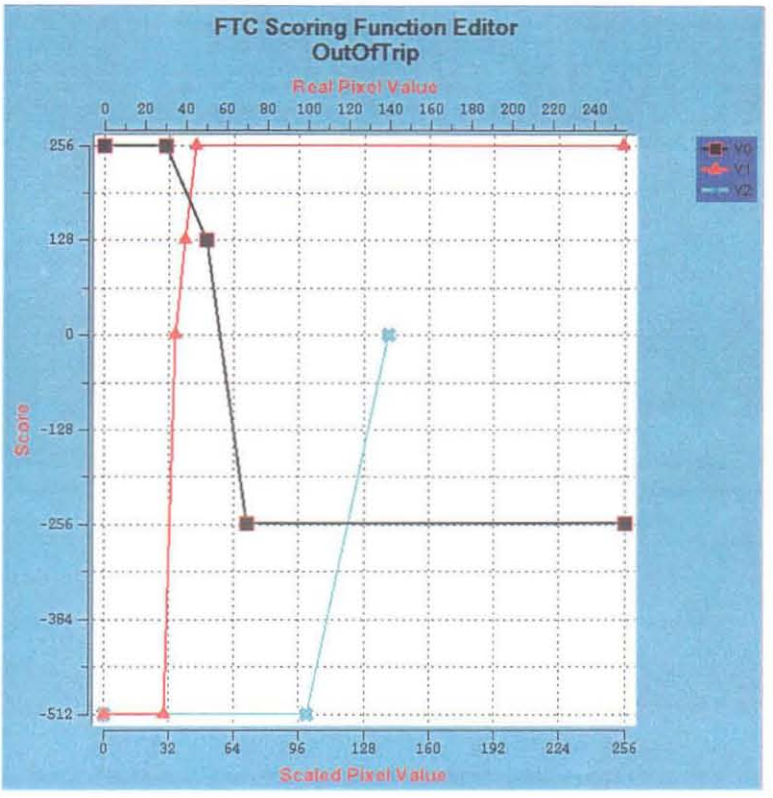


Figure 9. Example of out-of-trip (OOT) interest image (right) computed from corresponding reflectivity (left) and velocity (middle) images. Images are from data collected on 7/7/98 at Albuquerque, NM



Kernel #1: Reflectivity

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

Kernel #2: Velocity

					2				
					2				
					2				
					2				
					2				
					2				
					2				
					2				
					2				
					2				
					2				
					2				
					2				
					2				
					2				
					2				
					2				
					2				
					2				
					2				
					2				
					2				

Kernel rotation angle: 90

Figure 10. Scoring functions and templates for the OutOfTrip weather detector. Lower X-axis of scoring function plot is the integer-scaled pixel value. Upper X-axis is reflectivity in dBZ

2.2.7.9 Computation of radial velocity shear maps (convergence images)

In order to detect the wind convergence associated with gust fronts, two separate "shear" maps representing the local change in radial velocity over two different distance scales are computed from the velocity data contained in *gfBaseImages->dbvPol* and *gfBaseImages->vPolRaw* (recall that *dbvPol* contains "cleaned" DBV data, and *vPolRaw* contains LBV data). The first shear map (*gfBaseImages->conv*) represents the change in velocity over a relatively short distance (*gfParams->convergeWindowSize*) along each radial of the *dbvPol* image, while the second shear map (*gfBaseImages->lowResConv*) represents the change in velocity over a larger distance (*gfParams->lowResConvWindowSize*) as seen in the *vPolRaw* image. Lines of high convergent "shear" values in these images will be identified by the velocity convergence feature detectors.

Function **ComputeConvergenceImage** generates the convergence images. This function uses the CSKETCH library routine **SKLsqDerivFilter** to perform the computation. **SKLsqDerivFilter** computes the change of velocity over a window distance by computing the slope of the least-squares regression line through the data points within the sample window. The derivative is computed only if there are a sufficient fraction of non-missing data points in the window and if the correlation coefficient does not exceed a threshold. When calling **SKLsqDerivFilter**, **ComputeConvergenceImage** requires that at least 50% of the values in the window be not missing, but does not currently impose any correlation coefficient requirement (i.e., the correlation coefficient threshold is zero). After resampling the polar-coordinate shear map image to Cartesian coordinates, all positive derivative values associated with areas of divergent radial velocity change are set to zero, and the sign of the remaining convergent derivative values is reversed to make them positive.

In **GFFillSingleScanObjects**, a 1-km scale shear map (*gfBaseImages->conv*) is computed first by calling **ComputeConvergenceImage** with the *gfBaseImages->dbvPol* image and the appropriate half-window size in range gates as derived from the *gfParams->convergeWindowSize* parameter (nominally, 1000 meters). Isolated "speckles" of convergence are removed by performing a gray-scale erosion on the convergence image with a 3x3 elliptical kernel. It is important to note that the *gfBaseImages->conv* image serves more than one purpose. In addition to its use as input to the Converge and ConvMotion feature detectors, its values contribute to the consensus estimate of the delta-V once a front has been detected. Traditionally, the delta-V, or "wind shear hazard", that is reported represents the velocity change over 1 km. Because of this additional use of the *gfBaseImages->conv* image, the default setting of *gfParams->convergeWindowSize* should probably not be modified from its nominal 1000 meter value without consideration of the impact on reported gust front wind shear estimates.

The low resolution convergence image (*gfBaseImages->lowResConv*) is computed next by calling function **ComputeLowResConv**. **ComputeLowResConv** calls **ComputeConvergeImage** with the *gfBaseImages->vPolRaw image* and the half-window size derived from the *gfParams->lowResConvWindowSize* parameter (nominally, 3000 meters). Gray-scale erosion with a 3x3 elliptical kernel is also performed on this image to remove convergence speckle. Because the only use of the low resolution convergence image is for input to the Converge and ConvMotion feature detectors, **ComputeLowResConv** further processes the convergence image to prepare it for input to FTC. FTC requires that the integer image data lie in the range [0, 255]. CSKETCH library function **SKScaleArrayToBounds** is used to linearly rescale the image data such that the integer representations of those delta-V values falling within the physical lower and upper bounds given by *gfParams->scaleBoundsLowResConv* (nominally 0 and 8 m/s), map to the interval [0, 255]. Integer encodings of delta-V values that exceed the maximum bound are set to 255, and values less than the minimum bound are set to 0 (i.e., the data are clipped). The Converge and ConvMotion detector scoring functions are tuned to these re-scaled integer data representations, so if the scale bounds are changed, then the scoring functions would likely need to be adjusted as well.

When MIGFA computes the wind shear hazard associated with a detected gust front, it obtains shear estimates from the 1-km resolution shear map in a restricted search region immediately behind a relatively short segment of the gust front (the so-called wind analysis segment). To increase the likelihood that a representative shear value will be present at the analysis point, **GFFillSingleScanObjects** computes an additional 7x7 gray-scale dilated representation of the *gfBaseImages->conv* image and stores it in *gfBaseImages->convD77* for later use by the wind estimation routines. Figure 11 shows the normal and low-resolution shear maps computed from the dual-beam and low-beam velocity data.

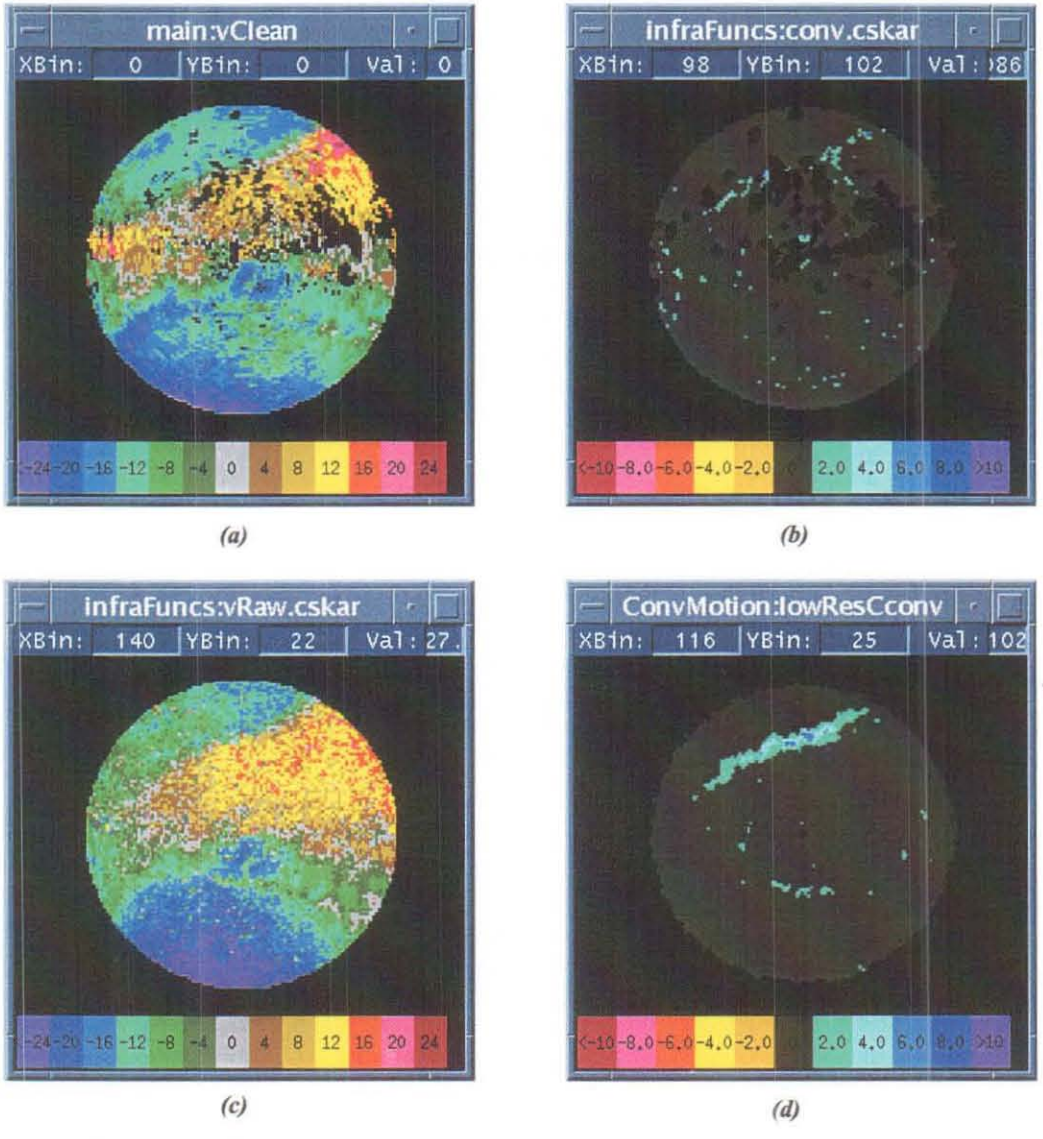


Figure 11. Example shear maps computed from WSP velocity data. (a) Cleaned dual-beam velocity. (b) 1-km convergent radial shear map computed from (a). (c) Low-beam velocity data (vRaw). (d) Low resolution convergent radial shear map computed from (c). Data are from 5/1/00 06:55:22 GMT at Austin, TX.

2.2.7.10 Computation of stratiform rain and storm cell images

Images denoting areas of stratiform (low-intensity) and storm cell (high intensity) storm echoes are used as masks in many of the feature detectors. A functional template approach is used to create interest images of these areas. Copies of these interest images are typically binarized within the feature detectors by applying an interest threshold of 128 in order to generate an appropriate mask image to be utilized in the functional template correlation. Some feature detectors require more aggressive masking of these types of precipitation regions, so several variants of stratiform rain and storm cell images are computed in **GFFillSingleScanImages** and stored in the *gfBaseImages* object for future reference. These are: *stratRain, stratRainExpanded, stratRainD77, stormCells, stormCellsSuppressClutter,* and *stormCellsD77.* Note that the "D77" variants are not currently used by the WSP MIGFA (these are binary images created by first thresholding copies of the *stratRain* and *stormCells* interest images by *gfParams->interestThreshold* and then performing a binary dilation using a 7x7 elliptical kernel).

The *stratRainExpanded* image is generated utilizing a scoring function that returns positive scores for lower values of reflectivity than the standard *stratRain* detector, and is used for masking in feature detectors that look for thin lines in very clear air (TLDZSD, and DZSDMotion). The *stormCellsSuppressClutter* image is the same as *stormCells* except that values in a copy of the *gfBaseImages->dz* image are first set to 0 (-20 dBZ) wherever clutter is indicated in the *gfBaseImages->clutterMap* image. The HIDZ detector is currently the only one that uses the clutter-suppressed storm cells image.

To generate the stratiform rain and storm cell interest images, the **RunDetector** method of each of these detector objects is directly invoked by **GFFillSingleScanImages**. Each of these detectors calls a generic reflectivity-based feature detector named **simpleDZFilterDetector**. **SimpleDZFilterDetector** is an unusual detector in that it does not have its own "built-in" template object. Rather, the calling functions are required to supply a *SKFuncTemplate* object containing the kernel and scoring functions. Figures 12, 13, and 14 show example computations of *stratRainExpanded, stratRain,* and *stormCells* interest images (the *stormCellsSuppressClutter* image is not shown since for the example chosen, its image is identical to the *stormCells* image).

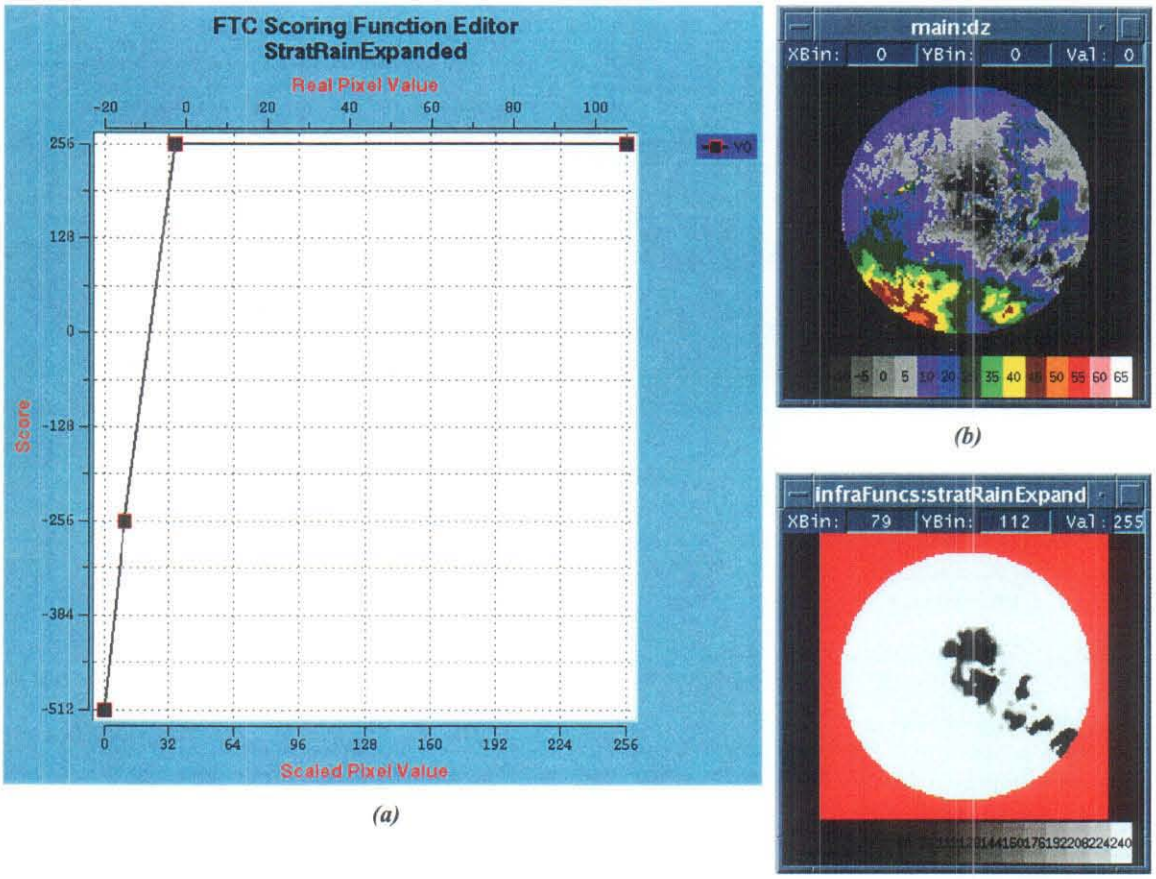


Figure 12. Example computation of "expanded" stratiform rain interest image. The scoring function (a) is applied to the reflectivity image (b) using a 3x3 ellipsoidal kernel. The resulting interest image is shown in (c). Lower X-axis of scoring function is the integer-scaled pixel value. Upper X-axis is reflectivity in dBZ. Data are from 5/24/99 at Albuquerque, NM.

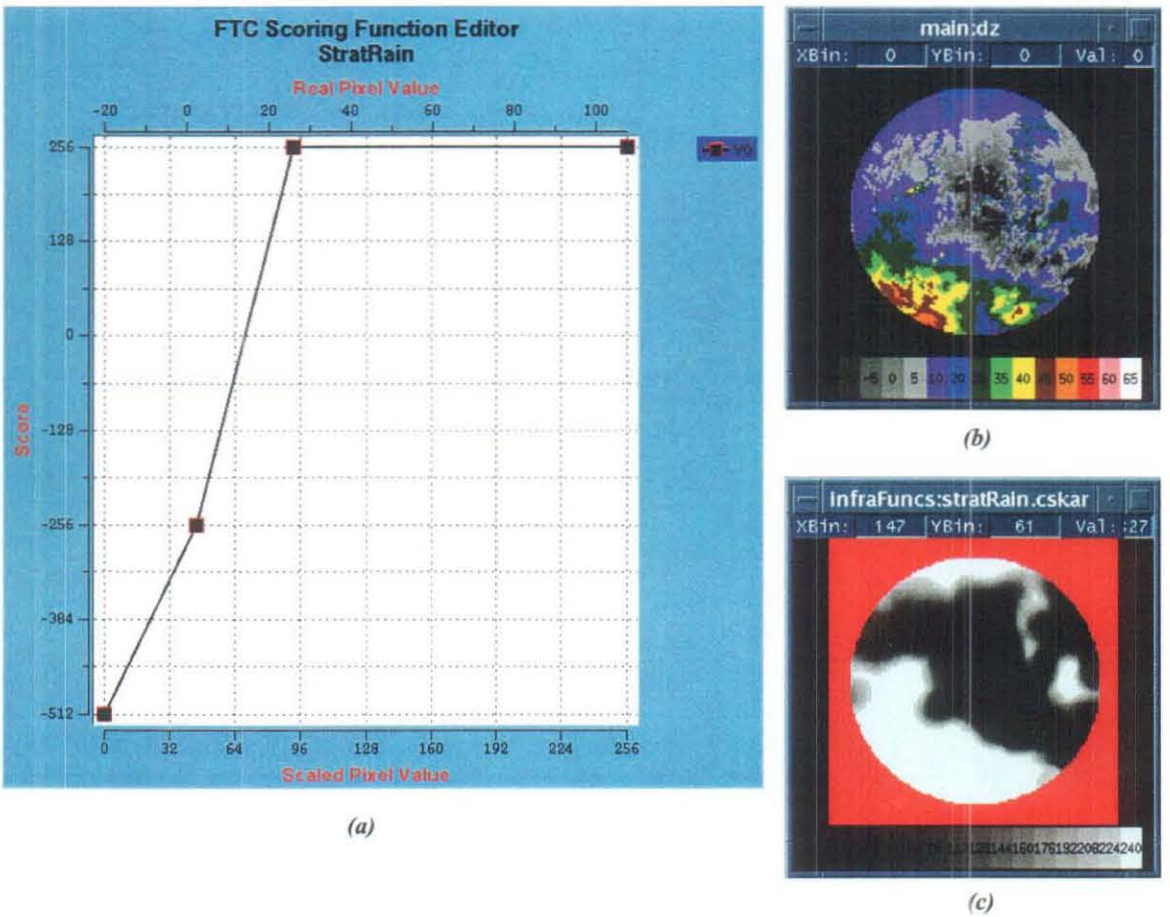


Figure 13. Example computation of stratiform rain interest image. The scoring function (a) is applied to the reflectivity image (b) using a 13x13 ellipsoidal kernel. The resulting interest image is shown in (c). Lower X-axis of scoring function is the integer-scaled pixel value. Upper X-axis is reflectivity in dBZ. Data are from 5/24/99 at Albuquerque, NM.

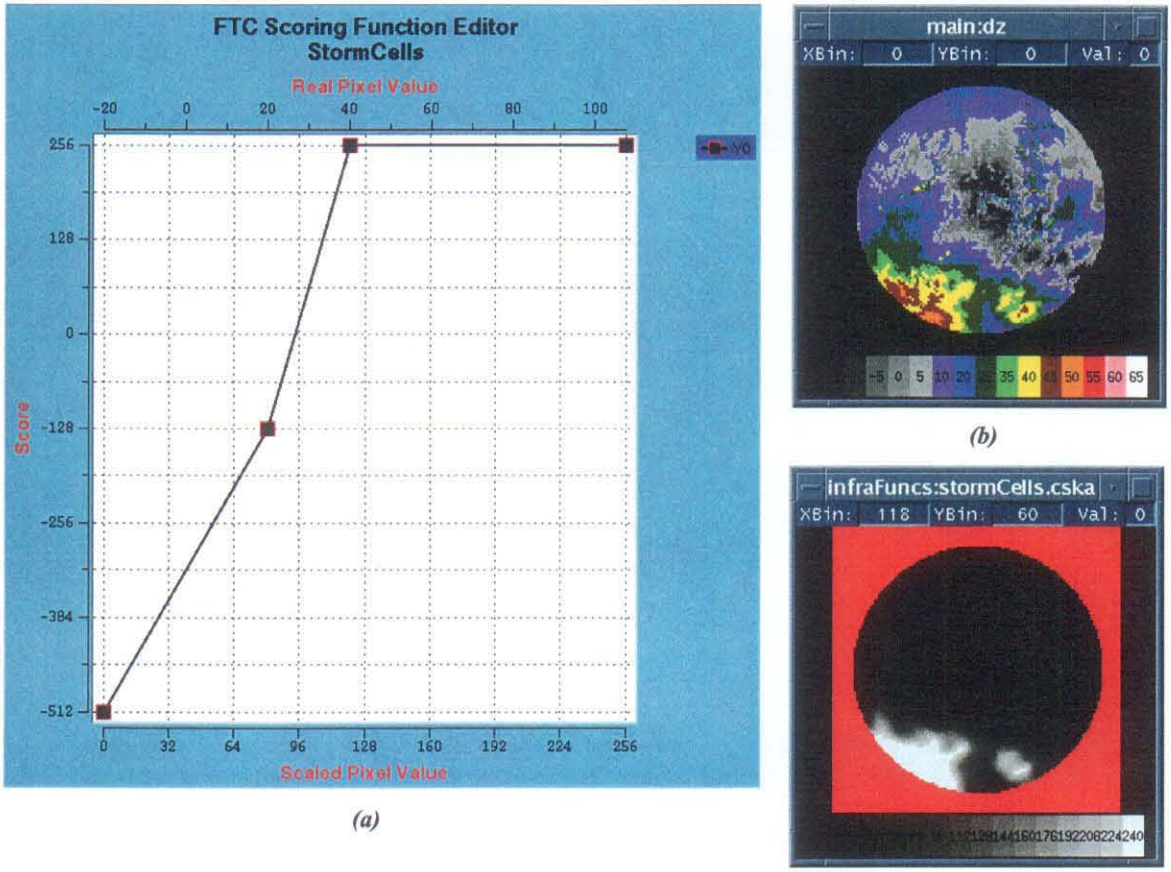


Figure 14. Example computation of storm cells interest image. The scoring function (a) is applied to the reflectivity image (b) using a 9x9 ellipsoidal kernel. The resulting interest image is shown in (c). Lower X-axis of scoring function is the integer-scaled pixel value. Upper X-axis is reflectivity in dBZ. Data are from 5/24/99 at Albuquerque, NM.

2.2.7.11 Computation of zero-Doppler line image

During episodes of vertical wind shear where the wind direction is veering or backing with altitude, the zero Doppler line as seen in the velocity data imagery will be observed to curve with increasing range from the radar (sometimes sharply) as the broadening radar beam increasingly samples winds at higher altitudes that may have a direction that is markedly different from that near the surface. If the vertical wind shear is strong enough, the radial velocity shear can cause high interest scores to be generated by the velocity convergence detectors (Converge, and ConvMotion), potentially leading to false gust front detections. To combat this, a binary zero-Doppler line image is computed for later use as a detection mask in the Converge and ConvMotion feature detectors.

In order to detect zero lines, a two-stage process is used whereby radially oriented zero lines are found first by processing velocity data in the polar domain. This is followed by a second, Cartesian domain step that finds zero line features of any orientation. The two interest images are conservatively combined by taking the pixel-wise minimum of each interest image. Figure 15 shows the scoring functions and templates used for the polar and Cartesian FTC operations. Since zero Doppler lines are radially aligned over at least some fraction of the range gates, the polar coordinate template is applied only over a very limited 40 degree wedge centered on the radial axis.

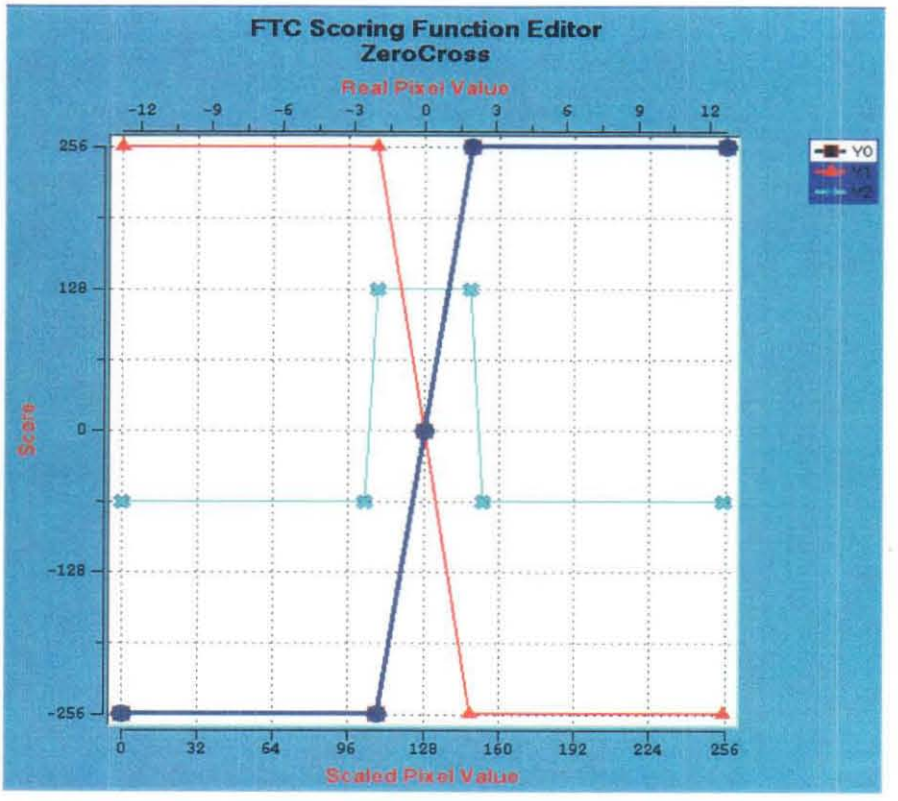
Figure 16 illustrates the output of the various processing stages of the zero line detector. Processing begins by first creating a weather mask image (*wxMask*) by copying the *gfBaseImages->vClean* image and setting all missing (*nil*) values to "1" and everything else to "0" (recall that *gfBaseImages->vClean* has had all velocity pixels not associated with weather removed, and so it is a convenient image for masking). Processing then proceeds by rescaling the velocity data contained in a copy of the *gfBaseImages->vPolRaw* image. The velocity data are rescaled such that each integral increment corresponds to a change of 0.1 m/s, and a value of 128 corresponds to a Doppler value of 0 m/s. The resulting values are clipped to the interval [0 255] (-12.8 to 12.7 m/s). This allows the data to be directly input to FTC later. Figures 16a and 16b show the polar velocity image before and after the rescaling. In order to prevent high interest scores from being generated on isolated lines of zero Doppler values surrounded by missing velocity data, all missing velocity values in the scaled and clipped image are set to 128 (0 m/s). Figure 16c shows the resulting interest image after performing FTC on the rescaled polar velocity. This polar interest image is then resampled to a Cartesian representation, and the *wxMask* image is used to mask out (set to *nil*) any pixels in the interest image that may have resulted from clutter, second trip, etc. (Figure 16d shows the *polScore* image that results from these operations.)

Next, the Cartesian-based zero Doppler line detection stage begins. First the polar velocity data in the *gfBaseImages->vPolRaw* image are resampled to a Cartesian representation using the *gfResamp* object. Masking for the Cartesian FTC operation consists of utilizing the same *wxMask* image that was previously computed and used to mask the Cartesian representation of the interest image from the polar FTC stage described above. Figure 16f shows the *carScore* interest image produced by the Cartesian domain FTC operation.

Earlier experimental versions of this detector performed a chain extension operation in an attempt to achieve better extent of zero line detections. Chain extension extends a single pixel wide chain of high interest along a line of moderate, but slightly below threshold interest values. Artificially high interest values are sometimes produced where missing velocity values in the copies of *gfBaseImages->vClean* image used in this detector were temporarily replaced with zeroes. To prevent inappropriate chain extension along these lines of artificially high interest, locations in the *carScore* interest image where the original *gfBaseImages->vClean* image has missing values are reset to *nil*. The current version of this detector no longer performs chain extraction and extension, so this additional masking step is no longer critical, and it may be possible to eliminate it in the future.

Next, the two interest images, *polScore* and *carScore* are conservatively combined (fused) by taking the pixel-wise minimum of the two images. If only one of the two pixels being compared is non-nil, than the non-nil value is returned as the minimum value. The resulting combined interest image (*score*) is shown in Figure 16g.

The combined *score* interest image may contain "spurs" of zero-line interest that are isolated or branch off of the main zero line that is targeted. These are interest spurs pruned by calling the CSKETCH **EliminatePoorShapes** function. **EliminatePoorShapes** takes the input array and applies a threshold for extracting regions whose areas will be analyzed. A length threshold is applied to eliminate those areas whose major axis does not meet the length threshold criteria. For pruning the zero line interest spurs, **EliminatePoorShapes** is called with a threshold of "96" and length threshold of 16 pixels (7.68 km at 480 meters/pixel). To better ensure complete zero line detection, the returned pruned interest array (Figure 16h) is binary dilated with a 3x3 elliptical kernel to form the final binary image of zero Doppler lines (Figure 16i).



Kernel #1: Polar Template

1		1		1		1		1		1		1			1
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
0		0		0		0		0		0		0			0

Rotation angles:
340, 0, 20, 160, 180, 200

Kernel #2: Cartesian Template

0				2					1
				2					
0				2					1
				2					
0				2					1
				2					
0				2					1
				2					
0				2					1
				2					
0				2					1
				2					
0				2					1
				2					
0				2					1

Rotation angles:
0, 30, 60, 90, 120, 150, 180,
210, 240, 270, 300, 330

Figure 15. Scoring functions and templates for the ZeroCross detector. Lower X-axis of scoring function is the integer-scaled pixel value. Upper X-axis is velocity in m/s.

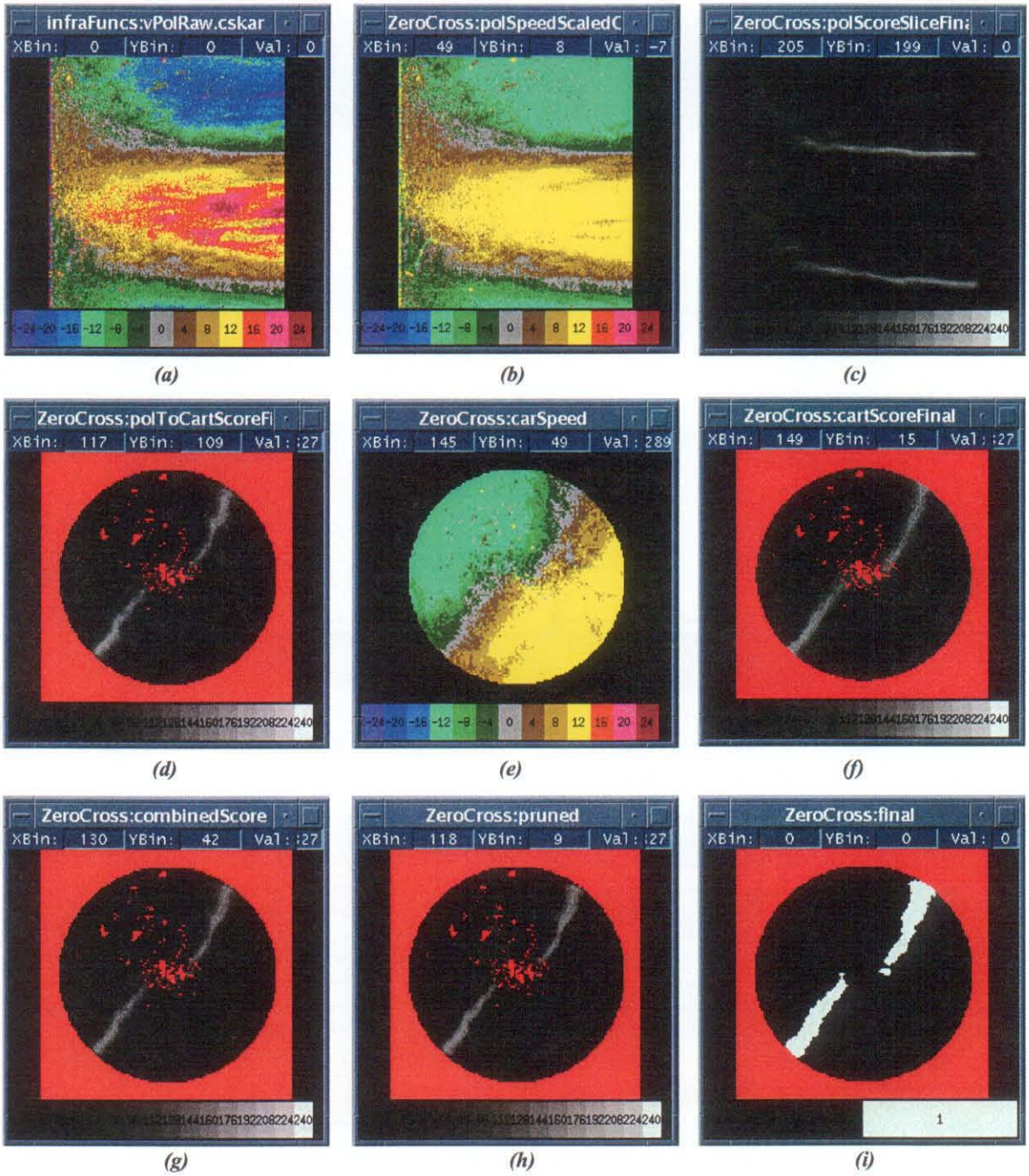


Figure 16. Summary of processing steps in detection of zero-Doppler lines. (a) Input polar velocity image. Each row of image pixels is a single radial (azimuth), with range increasing from left to right. (b) Polar velocity data after rescaling and truncation. (c) Polar zero-line interest image after FTC. (d) Interest image after resampling to Cartesian coordinates and application of non-weather mask. (e) Cartesian representation of rescaled velocity shown in (b). This is input to a Cartesian-based FTC operation which generates a second interest image shown in (f). (g) Combined interest image representing the pixel-wise minimum of (d) and (f). (h) Interest image after pruning to eliminate short interest regions. (i) Final binary image.

2.2.7.12 Generation of line storm interest and orientation images

Line storms - organized lines of thunderstorm cells -- are favored locations for gust fronts. An interest image indicating locations of line storms and an associated line storm orientation image are used in two ways:

1. The line storm interest image and associated orientation image are consulted by the individual gust front feature detectors. Feature detector interest values that are co-located with line storm regions and which have similar orientations to the line storms (as indicated by the line storm orientation image) are boosted by a multiplicative parameter.
2. The line storm interest image is used as a mask in the Anticipation detector to perform localized boosting of anticipation for fronts that are currently being tracked in the vicinity of a line storm. This helps to reduce the likelihood of losing an established gust front detection due to momentary loss of radar signatures.

Line storms are best detected in the long range (111 km) reflectivity data, where the large spatial aspect ratio of these echo regions can be more clearly discerned. Figure 17 shows the functional template and scoring functions that are applied to the long range reflectivity image (*gfBaseImagesWSP->dzLong*) to generate an initial line storm interest image.

Figure 18 illustrates the processing sequence for generation of the line storm interest and orientation images. The orientation image returned from FTC gives the orientation angles at which the best pattern match occurred at each image location, and is used to guide selective boosting of interest from the gust front feature detectors. Because orientation values in areas of low match scores tend to be randomly distributed, values in the initial orientation image are set to *nil* if their corresponding interest value is less than the ambiguity threshold of 128.

Since one of the objectives is to boost gust front feature detector interest in the vicinity of a line storm, the initial line storm interest image undergoes multiple gray scale dilation operations in order to substantially expand the initial areas of supporting interest. Multiple dilations with a small kernel yield a less "blocky" result than a single dilation with a very large kernel. So, four consecutive gray scale dilation operations with a 5x5 elliptical kernel are performed. Given the nominal 1920 km resolution of the input long range reflectivity image, this results in a substantial increase in the amount of area encompassed by supporting interest values in the vicinity of a line storm.

The initial associated orientation image is also dilated so that orientation values can be associated with values in the dilated interest image. The standard gray scale image dilation operation is inappropriate for an orientation image, since it would bias the angle values. Function **MedianDilate** is called four times with a 5x5 elliptical dilation kernel to perform an unbiased dilation of the orientation image. The median dilation operation finds for each location in the input array, the median of all image values superimposed by the non-nil regions of the dilation kernel.

The resulting long-range dilated line storm interest and orientation images have a coarser resolution and extend further in range than the standard 30 km range interest images that they will be later fused with. Therefore, these two images are next resampled to the standard 30 km image geometry. The *SKStmResamp* class in the CSKETCH library is a Cartesian-to-Cartesian resampler class that is used to do this. Once the two images have been resampled to the standard 30-km, 480 meter resolution grid, a final gray scale and median dilation with a 15x15 kernel is performed on the resampled interest and orientation images respectively.

2.2.7.12.1 Use of line storm interest by feature detectors

The line storm interest image and orientation image are used within the individual gust front feature detectors to guide selective boosting of interest values. Within the equipped feature detectors, function **BoostLineStormAlignedInterest** is called to perform this operation. For each pixel in a feature detector interest image $I_d(x,y)$ having orientation $O_d(x,y)$ and co-located within a line storm region having orientation $O_{ls}(x,y)$!= *nil*, a boost factor, B, is computed as:

$$B = 1.0 + ((B_{max} - 1.0) * W)$$

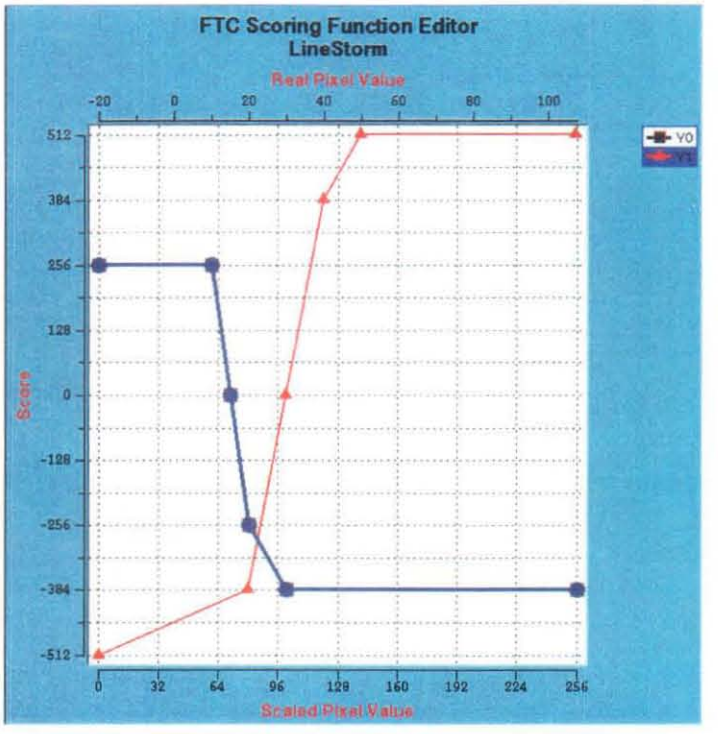
where B_{max} is the maximum boost factor defined by the detector's *LineStormBoostFactor* parameter. W is a weight factor that varies from 0.0 to 1.0 and is a falling linear ramp function of the angular difference between O_d and O_{ls}. $W = 1.0$ for an angular difference of 0 degrees, and $W = 0.0$ for a maximum angular difference of *LineStormAngleTolerance* (a parameter nominally 45 degrees). The adjusted detector interest $I_d(adjusted)$ is then computed using:

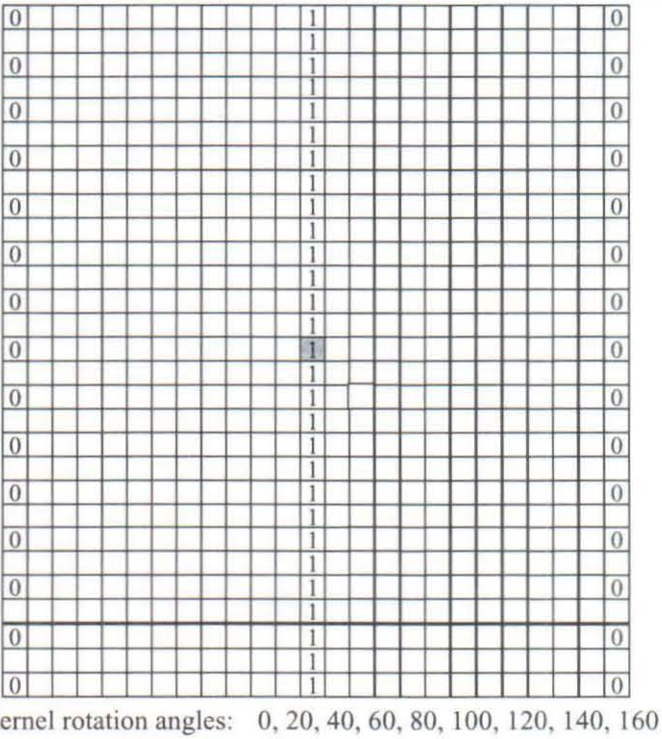
$$I_d(adjusted) = I_d * B$$

Figure 19 illustrates the selective boosting of interest from the reflectivity thin line detector (TLDZ) for a case where a gust front thin line was produced ahead of an approaching line storm. The following detectors call **BoostLineStormAlignedInterest** to adjust interest values based on line storm evidence: *Converge, ConvMotion, DZMotion, DZSDMotion, SDMotion, TLDZ, TLSD, TLSDDZ.*

2.2.7.12.2 Use of line storm interest to boost anticipation

The other use of line storm evidence is to perform localized boosting of anticipation of fronts being currently tracked. In the Anticipation detector, if the expected location of a gust front is within an area of line storm interest > 128, then the value used for the localized anticipation interest (normally set to 127 for newly tracked fronts and 255 for long-term tracked fronts) is obtained from the *Anticipation(LineStormShortTermAntVal)* or *Anticipation(LineStormLongTermAntVal)* parameters, whichever is appropriate. Settings for these two parameters at the time of this writing are 127 (unchanged) and 512 (twice the normal value) for short and long-term anticipation values respectively.





Kernel rotation angles: 0, 20, 40, 60, 80, 100, 120, 140, 160

Figure 17. Scoring functions and template for the LineStorm detector. Lower X-axis of scoring function is the integer-scaled pixel value. Upper X-axis is reflectivity in dBZ. Resolution of 23x29 kernel is 1920 meters/pixel.

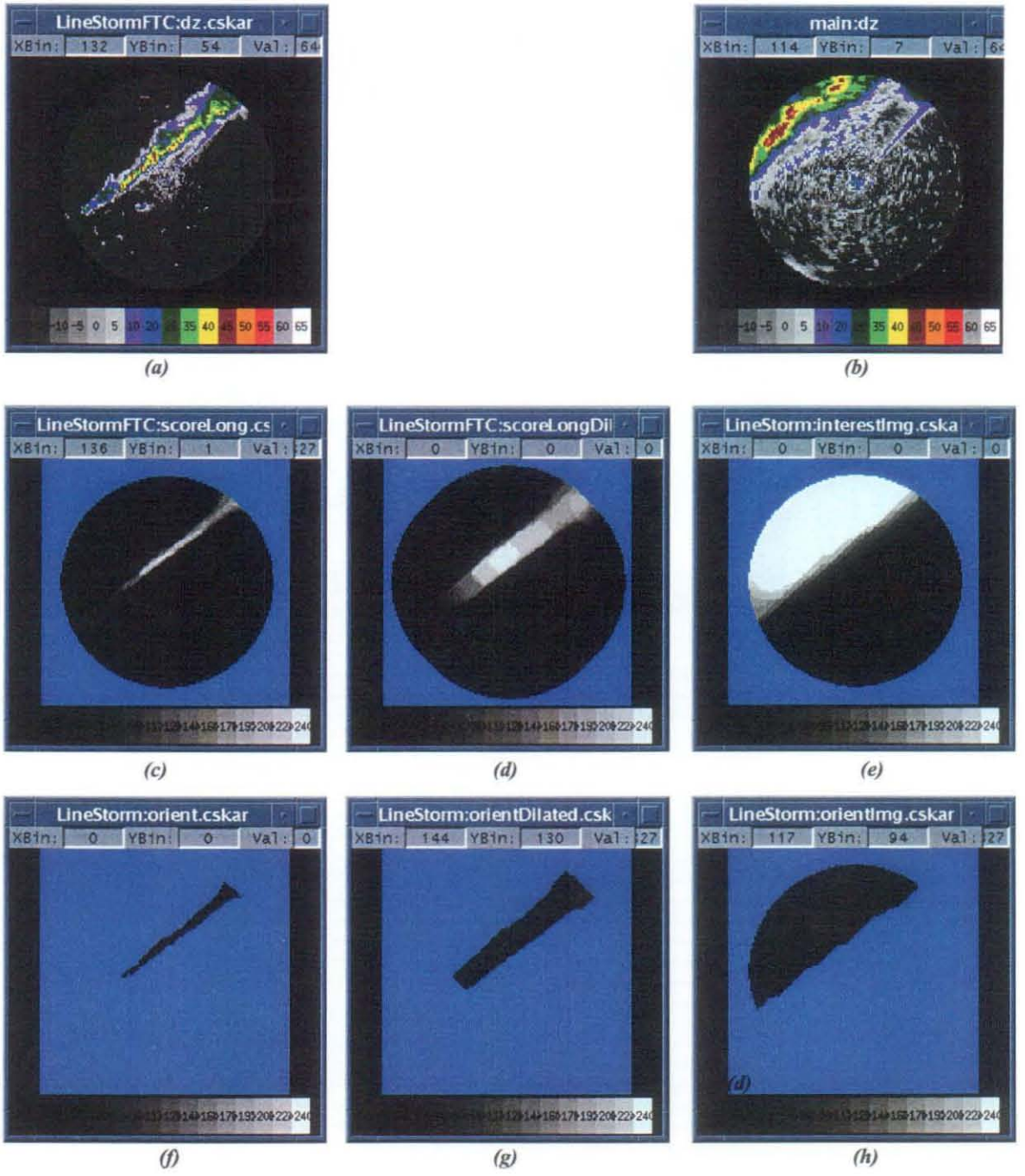


Figure 18. Summary of line storm detector processing. (a) Current long-range (111 km) reflectivity image. (b) Corresponding short-range (30 km) reflectivity image. (c) Initial long-range interest image. (d) Long-range interest image after multiple gray scale dilations. (e) Final interest image after resampling to 30 km grid and final dilation by 15x15 kernel. (f) Initial long-range orientation image after removing values associated with low interest. (g) Long-range orientation image after multiple median dilations. (h) Final orientation image after final dilation with 15x15 median filter.

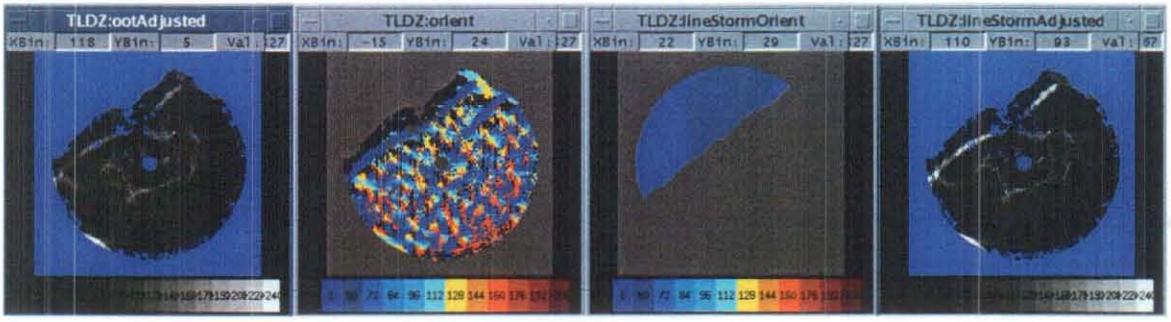


Figure 19. Example of boosting of feature detector interest in the presence of a line storm. (a) Original interest image from the reflectivity thin line feature detector (TLDZ). The relatively bright interest band extending from NE to SW in the upper left quadrant is associated with an approaching gust front ahead of a line storm (see reflectivity image in Figure 18b). (b) Corresponding TLDZ orientation image. Note coherent band of similar orientation values associated with the bright interest band seen in (a). (c) Dilated line storm orientation image. (d) Modified TLDZ interest image showing enhancement of thin line interest ahead of the approaching line storm. Other interest pixels beyond the vicinity of the line storm are unchanged.

2.2.7.13 Dynamic sensitivity adjustment

MIGFA's sensitivity is dynamically adjusted based on the type and extent of weather within its processing range. If there is a predominance of convective weather (high reflectivity areas), then conditions are favorable for gust fronts and MIGFA's overall sensitivity is heightened. Conversely, during episodes of widespread low-reflectivity light rain (stratiform rain), gust fronts are not expected. Moreover, the potential for false alarms actually increases during stratiform rain episodes owing to occasional rain echoes that may organize in moving bands that mimic gust front thin line signatures. Therefore, it is advantageous to reduce MIGFA's sensitivity during these episodes. Additionally, false alarm mitigation logic designed to suppress interest features moving in the same direction as the ambient wind is automatically disabled if convective weather is detected within the processing range.

Precipitation coverages are estimated using both long and short range reflectivity images. Since gust fronts can propagate many kilometers away from generating thunderstorms, a long-range (111 km) look for convective weather is needed to recognize conditions favorable for gust fronts. Conversely, stratiform rain echoes cause false interest features only if they are inside 30 km where the gust front feature detectors operate. So, for general sensitization purposes, it is more important to ascertain the amount of stratiform rain at short range.

A pair of "precip numbers" (numbers ranging from 0 to 10 that represent the relative coverage of stratiform rain and convective weather) are computed by calling function **ComputePrecipNumbersWSP** with a reflectivity image that has all non-weather pixels removed. The computation is performed twice; once with a copy of the standard Cartesian 30-km range reflectivity image, and once with a high resolution (500 m) Cartesian long-range reflectivity image. Function **editNonWxDZ** is called to remove out-of-trip and clutter cells from the reflectivity images before calling **ComputePrecipNumbersWSP**. Stratiform rain and storm cell coverage estimates used to adjust MIGFA's overall sensitivity come from the short-range stratiform rain and the long-range storm cell counts respectively. The two quantities are stored in the global *gfRefData->stratRainCoverage* and *gfRefData->stormCellCoverage* variables for later reference.

For purposes of disabling steering-wind based suppression of interest features, the stratiform rain count from the short range computation is used, while the storm cell coverage is taken as the maximum of the short and long-range storm cell counts. These two quantities are stored in the global *auxRefData->stratRainCoverage* and *auxRefData->stormCellCoverage* variables for later reference.

2.2.7.13.1 Computing the stratiform rain and storm cell coverage

The coverage computation performed by **ComputePrecipNumbersWSP** basically tallies the pixels that fall between reflectivity limits given by parameters *gfParams->stratRainDZ* and *gfParams->stormCellDZ* (each is a pair of dBZ values). Default values for these parameters are located in the paramsWSP file and are set to 5 to 25 dBZ for *gfParams->stratRainDZ* and 30 dBZ and higher for *gfParams->stormCellDZ*. Because these thresholds will likely be different depending on the general climate of each site, the default values are typically overridden in the site-specific *paramsWSP.local* file.

The tallying approach allows for assigning larger weights to pixels at short range in order to give the short range weather more import. The parameters that control the range-dependent differential weighting of stratiform rain and storm cell pixels are *cellScalar, cellBeta, stratScalar,* and *stratBeta*. Figure 20 illustrates the use of *stratScalar* and *stratBeta*; the use of *cellScalar* and *cellBeta* is analogous. All of these values, as well as the piecewise-linear/parabolic shape of the weighting function, have been determined empirically.

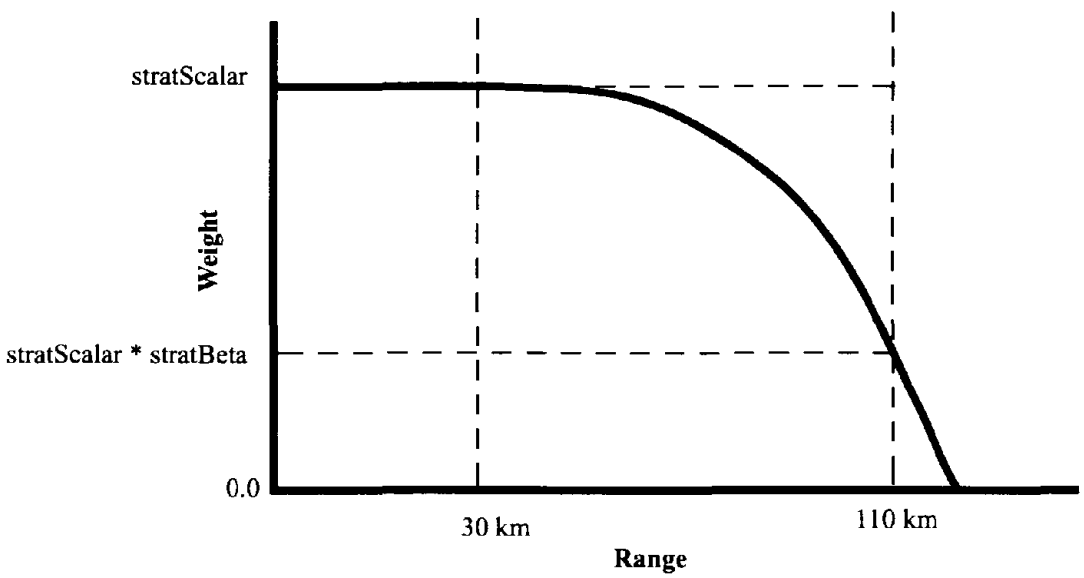


Figure 20. Illustration of stratiform rain and storm cell pixel weighting for precipitation coverage computation.

The precip numbers stored in *gfRefData->stratRainCoverage* and *gfRefData->stormCellCoverage* are used by the Anticipation detector (discussed in the next section) to index into the *gfParams->precipBiasTable*. The *gfParams->precipBiasTable* is an 11x11 SKArray that is constructed and loaded with the data from the *precipBiasTableData* parameter of the *paramsWSP* (or

paramsWSP.local) file. Figure 21 shows an example of *precipBiasTableData* suitable for the Austin, TX environment. The table contains background anticipation interest values as a function of stratiform rain (x-axis) and convective weather (y-axis) coverage, with (0,0) at the lower left of the table. As can be seen from the figure, decreasing values of background interest are returned for increasing amounts of stratiform rain in the absence of convective weather. When the background interest is averaged against the precombined interest from the other detectors, the overall interest levels will be suppressed or enhanced depending on the type and extent of the precipitation. A value of "96" in the *precipBiasTableData* represents the highest background interest that can be used without causing an undesirably high background interest that could result in a confusing (overly bright) interest map with numerous chaotic detections.

```
set precipBiasTableData {
 "96 96 96 96 96 96 96 96 96 96 96"
 "96 96 96 96 96 96 96 96 96 96 96"
 "96 96 96 96 96 96 96 96 96 96 96"
 "96 96 96 96 96 96 96 96 96 96 96"
 "96 96 96 96 96 96 96 96 96 80 80"
 "96 96 96 96 96 96 96 80 80 64 64"
 "96 96 96 96 96 80 80 64 64 48 48"
 "96 96 96 80 80 48 48 48 48 32 32"
 "96 96 80 48 48 32 32 32 32 16 16"
 "96 80 64 48 32 32 16 16 16  0  0"
 "80 80  0  0  0  0  0  0  0  0  0"
}
```

Figure 21. Example precipBiasTableData from paramsWSP.local file for Austin, TX

2.3 Feature Detection

Detection in MIGFA is based on a consensus approach in which the results of multiple feature detectors (computational modules that detect particular signatures or sets of signatures) contribute to a combined interest image. Each individual feature detector computes its own interest image, in most cases through the use of functional template correlation [1], and assigns weights to the interest image as a whole. In addition, a feature detector may assign to a particular pixel in the interest image a *nil* value, meaning that the feature detector has no opinion at that location. Each feature detector constructs an appropriate "mask" image of *nil* values depending on that detector's areas of skill. For example, because observability of most gust front signatures is reduced in the immediate vicinity of the radar, the masks in the majority of the WSP MIGFA feature detectors include a 2-km "bulls-eye" of *nil* values to help ensure that possible negative evidence in this region doesn't cause a previously detected gust front to be dropped when it is over the radar - and in the case the WSP - over the airport.

The feature detectors in *detectorList* are invoked during the detection stage, with each detector producing an interest image. As described previously, *detectorList* is equal to *detectorListStatic* for the first scan processed, and *detectorListMotion* for all subsequent scans. After each of the feature detectors has been run, CSKETCH library function **SKAverageInterestImages** is called to compute a weighted average of the individual interest images. The resulting combined interest image is stored in *gfBaseImages->rawInterest* where it is subsequently used during the extraction stage as the primary basis for extraction.

2.3.1 Suppression of false thin line interest features

Bands of low-reflectivity precipitation moving with mid-altitude steering winds are a frequent source of false detections. These bands often produce nearly identical radar signature characteristics as gust front thin lines (both in reflectivity and in velocity). As a result, the various reflectivity and velocity thin-line feature detectors (DZMotion, TLDZ, TLSD, SDMotion, and DZSDMotion) often generate inappropriately high interest values on these echoes, increasing the likelihood of false alarms. To combat misinterpretation of these weather features, these feature detectors are equipped to utilize one of two approaches in order to reduce interest values that are likely to be associated with moving precipitation features rather than gust fronts:

1. Wintertime fixed azimuth sector suppression of interest features moving from the south.
2. Dynamic sector suppression of interest features moving in a direction consistent with mid-altitude steering winds.

Given sufficient WSP velocity data quality, the dynamic steering-wind sector based suppression is the preferable approach for adjusting the interest images, since the direction from which moving interest features are suppressed is computed based on the actual wind direction observed by the radar. However, during the winter (and transitional) seasons, suppression of interest features moving from a southerly sector provides a suitable fall-back. The fixed sector suppression approach is a statistically reasonable once since during these months, many areas of the country are visited by organized low pressure systems, where banded stratiform rain echoes are often observed moving with the southerly flow in the warm sector ahead of an approaching cold front.

These suppression functions are used conservatively. If any convective weather or clear conditions are present (as determined from the storm cell and stratiform rain reflectivity pixel counts), then direction-based interest suppression by either approach is automatically disabled. Additionally, steering-wind based suppression is performed only if there is sufficient confidence in the steering wind computation (given the quantity and quality of Doppler velocity data).

The selected feature detectors call function **SuppressSector** to perform the adjustment of interest values by one of these methods, depending on season and precipitation regime. Its parameters are:

1. Initial feature detector interest image to be (possibly) adjusted.
2. DZMotion detector orientation image.
3. Start, end azimuth bounds to be used for fixed sector suppression (*BiasSector* detector parameter).
4. Multiplicative interest bias factor for fixed sector suppression (*SectorBias* detector parameter).
5. Suggested bias factor to for adjustment of interest values by dynamic sector suppression (*SteeringWindSectorBias* detector parameter).

Interest feature suppression by either of the two methods requires knowledge of the direction of motion for each interest pixel under consideration. The direction of motion is inferred from the orientation image produced by the DZMotion thin-line motion detector. Because of this dependency on the DZMotion orientation image, the DZMotion detector is the first feature detector that is run after the Anticipation detector. The DZMotion detector, by virtue of its 360 degree rotating template, provides a directionally unambiguous thin line orientation angle at each pixel. With this information, the direction of motion for any given feature detector interest pixel is taken to be orthogonal to the DZMotion orientation angle at the corresponding pixel location.

2.3.1.1 Procedure for suppression of false thin line features

Shortly after creation of an initial interest image, the feature detector calls **SuppressSector** to possibly modify the image. **SuppressSector** first calls **FindWindBias** to compute the steering wind based bias. It computes the starting and ending azimuth bounds for the dynamic suppression sector centered on the steering wind direction, and determines an appropriate bias factor that may be different than the suggested dynamic bias factor depending on the confidence of the associated steering wind analysis.

Three parameters contained in the site-specific *detectorParamsWSP* parameter file are utilized by **FindWindBias**:

1. *steeringWindSectorWidth*
2. *vadBias*
3. *vadConfidenceRamp*

The *steeringWindSectorWidth* parameter defines the angular width of the azimuthal suppression sector that will be centered on the direction of the steering wind.

The *vadBias* parameter controls how much to bias the steering wind bias computation in favor of the radar-derived estimate of the steering winds versus an optional additional estimate of the steering winds provided by a storm-motion based estimate. Storm motion based estimation of steering winds is not currently enabled in the WSP MIGFA, so *vadBias* should be set to 1.0 in the parameter file.

The *vadConfidenceRamp* parameter is a value between 0.0 and 1.0 that controls the degree to which the lack of confidence in the regional wind calculation reduces the initial amount of suggested bias. For example, a *vadConfidenceRamp* value of zero means that the regional wind estimate will be used as the basis for direction-based interest suppression regardless of the confidence in the regional wind computation. As of this writing, a value of 0.4 is used for "wet" sites, and 0.9 is used for "dry" sites.

Function **FindWindBias** begins by first retrieving the regional wind estimate that was previously computed and stored in *gfEvent->regionalWind* as described in Section 2.2.7.3. If the regional wind estimate indicates nearly calm conditions (both of the wind components (u or v) are 0.1 m/s or less), then the interest bias factor is simply set to 1.0 (no interest biasing), and the function returns.

Otherwise, processing proceeds to compute the steering wind confidence factor, *rz*, that is simply the minimum of the u and v component weight factors that were computed previously during the regional wind computation. If the steering wind confidence factor is less than or equal to 0.3, then there is insufficient confidence in the steering wind computation, and the **FindWindBias** returns a bias factor of 1.0 (no biasing).

Otherwise, the steering wind direction (computed from the u and v components of the *gfEvent->regionalWind*) is used as the center of the azimuth sector whose limits are computed by adding +/- one half of the *steeringWindSectorWidth* to the steering wind direction.

Next, the dynamic suppression factor, *s*, is computed. The smallest that it can be is the ideal value suggested by the detector's *SteeringWindSectorBias* parameter, and the largest it will be is 1.0 (no suppression). The first adjustment to *SteeringWindSectorBias* is made based on the steering wind confidence factor, *rz*, that was computed as described above. The steering wind confidence factor is input to a rising linear ramp function that returns a scalar between 0 and 1 depending on *rz*'s location in the interval [*vadConfidenceRamp*, 1]. The resulting scalar, *zed*, that is output from the linear ramp weighting function is used to compute a dampened suppression factor, *s*, from the suggested suppression factor *SteeringWindSectorBias*, using:

$$s = zed * SteeringWindSectorBias + (1.0 - zed)$$

A second adjustment to *s* is made based on the precipitation coverage that is evident on the current scan. Dynamic sector suppression is disallowed in clear or stormy conditions by setting *s* to 1.0 if *auxRefData->stratRainCoverage* < 1 or *auxRefData->stormCellCoverage* > 1. If there is only minimal convective weather present (as indicated by *auxRefData->stormCellCoverage* = 1), then the adjusted suppression factor, *s*, is further adjusted by one half using:

$$s = (s + 1.0) / 2.0;$$

Function **FindWindBias** returns the computed steering wind sector limits and the adjusted steering wind sector bias factor, *s*, to function **SuppressSector**.

SuppressSector continues by deciding, based on the season, whether to adjust the feature detector interest image by applying the dynamic steering wind bias factor computed by **FindWindBias**, or whether to consider using fixed sector suppression. If the system is operating in summer mode (*gfRefData->summerMode* = TRUE), then the only candidate method is the dynamic steering wind sector suppression. Function **AdjustForDirection** is called to adjust all interest pixels whose corresponding locations in the DZMotion orientation image indicate a direction of movement from within the dynamic steering wind sector.

If MIGFA is operating outside of the summer months (*gfRefData->summerMode* = FALSE), then fixed sector suppression is considered. First, the fixed sector suppression factor whose value is given by *SectorBias* is conditioned according to the precipitation regime. If there are clear or stormy conditions, then fixed sector suppression is effectively disabled by setting the fixed sector bias factor to 1.0. If there is only minimal convective weather present, then the fixed sector bias factor is halved using the same equation used for adjusting the dynamic steering wind suppression factor above.

With this adjusted fixed sector suppression factor in hand, **SuppressSector** decides whether to apply the fixed sector or the dynamic steering wind sector adjustment by first checking the value of the adjusted steering wind bias factor, *s*. If *s* is greater than 0.95 (indicating little confidence the steering wind computation), then **SuppressSector** calls **AdjustForDirection** with the precipitation-adjusted fixed sector suppression factor and the fixed sector limits (*BiasSector)*. Otherwise, **AdjustForDirection** is called with the dynamic steering wind sector limits and the adjusted steering wind bias factor.

2.3.2 Running the feature detectors

Function **RunDetectors** is the driver subroutine for executing the feature detectors on the *detectorList* whose interest images will be subsequently averaged to produce the combined interest image that is the basis for gust front extraction. Following are brief descriptions of each feature detector in the order in which they are executed. The order of execution is dictated by the order in which the feature detector names are listed in the detector parameter file. The current list of feature detectors consists of the following:

1. Anticipation - Expected front location and general (background) interest
2. DZMotion - Reflectivity thin line motion detector
3. HIDZ - High altitude weather detector (a negative interest generator)
4. TLDZ - Static reflectivity thin line detector
5. TLSD - Static velocity thin line detector
6. SDMotion - Velocity thin line motion detector
7. TLSDDZ - Tandem velocity thin with no appreciable reflectivity detector
8. Converge - Velocity convergence detector
9. ConvMotion - Velocity convergence motion detector
10. DZSDMotion - Tandem reflectivity/velocity thin line detector
11. PrecipEdges - Storm edge detector (a negative interest generator)

2.3.3 Anticipation

2.3.3.1 Parameters

Table 6 lists the default site-adaptable parameter settings for the Anticipation detector.

Table 6. Default Parameters for the Anticipation Detector

Parameter Name	Nominal Value	Units
LineStormShortTermAntVal	127	interest
LineStormLongTermAntVal	512	interest
WinterBiasFactor	0.67	unitless
WinterBiasThresh	80	interest
WeightFunc	See Table 7	unitless

Table 7. Default Averaging Weight Function for the Anticipation Detector

Interest Value	Weight
0 - 15	1.00
16 - 95	0.75
>= 96	0.33

2.3.3.2 Mask images

The following pre-computed mask images are used by this detector:

gfBaseImages->eightKmMask
gfBaseImages->siteMask
gfBaseImages->detMask

2.3.3.3 Description

The Anticipation interest image provides a mechanism to spatially adjust the detection sensitivity of MIGFA, based on situational context. High anticipation values get averaged with interest values from other feature detectors to increase the likelihood of detection at specific locations. Low anticipation values suppress the likelihood of detection.

Computation of the anticipation interest image begins by first retrieving the line storm interest image from that was previously computed during data preparation. A binary line storm mask image is computed by thresholding a copy of the line storm interest image at 128. This binary line storm mask image will be used to further boost anticipation at expected locations of gust fronts.

Next, function **ComputeGFForecastMap** is called to generate a map of forecasted gust front locations appropriate for the time of the current scan. Elements in the resulting image array that have values of 1

(indicating short tracking history) and 2 (long tracking history) are reset to interest values of 127 and 255 respectively. Because organized lines of thunderstorms are very likely to have a gust front near the leading edge of the line, anticipation interest values for expected locations of gust fronts are set to *Anticipation(LineStormShortTermAntVal)* or *Anticipation(LineStormLongTermAntVal)* (as the case may be) wherever gust front anticipation pixels spatially coincide with pixels in the line storm mask image computed earlier. This helps to maintain tracking of line storm induced gust fronts that are often embedded in the precipitation along the leading edge of the line. Such strong anticipation interest will very nearly cause coasting of a detection within the vicinity of a line storm even if there are no other directly observable gust front signatures.

To help maintain tracking of gust fronts as they approach and cross over the radar, interest values of forecasted gust front points are further increased as follows. Within 8 km of the radar, anticipation interest for forecasted gust front points is increased by 10 percent. Within the small 2-km radius of the radar site mask (*gfBaseImages->siteMask*), interest values associated with short-term tracking history (127's) are boosted to the long-term tracking value of 255.

The remaining areas of the interest image are set to a background level based on the relative amounts of stratiform rain and storm cells as determined by indexing the precipitation bias lookup table *gfParams->precipBiasTable* using the precipitation coverage values previously computed and stored in *gfRefData->stratRainCoverage* and *gfRefData->stormCellCoverage*. During winter months, when MIGFA is more prone to false alarms, a site-adaptable fractional reduction in background anticipation is made. If the current date falls outside *gfParams->summerMonths*, and if the background anticipation value obtained from the table is less than *Anticipation(WinterBiasThresh)*, then the background anticipation value is scaled by multiplying the original background anticipation value by *Anticipation(WinterBiasFactor)*.

At ranges beyond 20 km, the background anticipation level is limited to a value of 64. This is done so as not to overly sensitize the algorithm at long range where false alarm probabilities increase due to high-altitude weather returns.

The resulting interest image is stored in *gfBaseImages->anticipation* for later retrieval during the interest combination stage. Figure 18d shows an example anticipation interest image.

2.3.4 DZMotion - Reflectivity Thin Line Motion Detector

2.3.4.1 Parameters

Table 8 lists the default parameter settings for the DZMotion reflectivity thin line motion detector. The functional template scoring functions, kernels, and kernel rotation angles are shown in Figure 22

Table 8. Default Parameters for the DZMotion Detector

Parameter Name	Nominal Value	Units
SteeringWindSectorBias	0.25	unitless
LineStormAngleTolerance	45	degrees
LineStormBoostFactor	1.5	unitless
BiasSector	(90, 250)	degrees
SectorBias	0.25	unitless
ConfWeight	1.0	unitless
DisConfWeight	0.5	unitless

2.3.4.2 Mask images

The following pre-computed mask images are used by this detector:

gfBaseImages->detSiteMask
gfBaseImages->stratRainD77

2.3.4.3 Description

The DZMotion detector looks for lines of low reflectivity moving through clear air. The motion of these lines can be detected by differencing the current and prior reflectivity images. A moving gust front thin line will show up in the difference image as a line of positive difference values (corresponding to the current location of the front) paralleling a line of negative difference values (corresponding to the previous location of the front). Figure 23 uses a simulated thin line signature moving at 10 m/s to illustrate the processing stages of the DZMotion detector described as follows.

The DZMotion detector utilizes functional templates that look for a pattern of positive and negative difference lines in close proximity to each other. Because the distance behind which the negative difference line trails the positive difference line depends on the propagation speed of the front, three separate FTC operations are performed, each utilizing template kernels with different widths between the positive and negative difference scoring regions. The three kernels are designed to match propagation speeds of 5 m/s, 10 m/s, and 15 m/s. The same scoring functions are used for all three templates, since the intensity of the difference line does not depend on the speed of movement. Within the typical range of gust front propagation speeds, a reasonably good match score will be obtained with one or more of these templates. The pixel-wise maximum of the three intermediate interest images is then computed to form the final DZMotion interest image. Recall that one of the by-products of FTC is a corresponding orientation image that contains the template rotation angle at which the best pattern match occurred. The

final orientation image is constructed by selecting for each pixel, the orientation value associated with maximum score of the three intermediate interest images.

Because the positive/negative reflectivity difference pattern is not rotationally symmetric, the templates for all three of the FTC operations must be rotated through a full 360 degrees to determine the best match. The resulting FTC orientation images are therefore directionally unambiguous. Consequently, adding 90 degrees to an orientation value yields the direction of motion of the reflectivity feature at that location. The use of the DZMotion orientation image to deduce the motion of reflectivity features is used as an aid in deciding whether to suppress interest values for pixels whose direction of motion is inconsistent with expectations for a gust front.

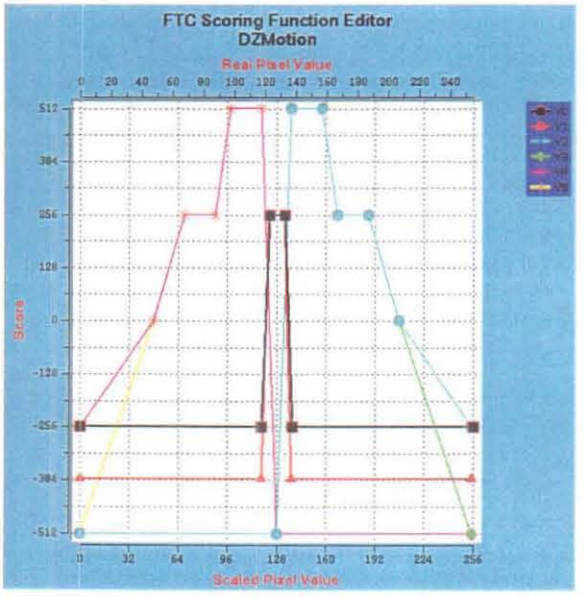
After the three FTC operations and generation of the maximum interest image, the maximum interest image is modified by reducing all radially-aligned interest pixels that are co-located with out-of-trip weather echoes by calling the function **SuppressOOTAlignedInterest**. Interest values whose associated orientation angles are within *gfParams->outOfTripAngleTolerance* (nominally, 20 degrees) of the radar azimuth angle are reduced by an amount equal to the corresponding out-of-trip interest (*gfBaseImages->outOfTrip*) multiplied by the reduction factor parameter *gfParams->ootReductionFactorDZMotion* (nominally, 0.7). That is,

$$interest(x,y) = interest(x,y) - [out\text{-}of\text{-}trip(x,y) * ootReductionFactor]$$

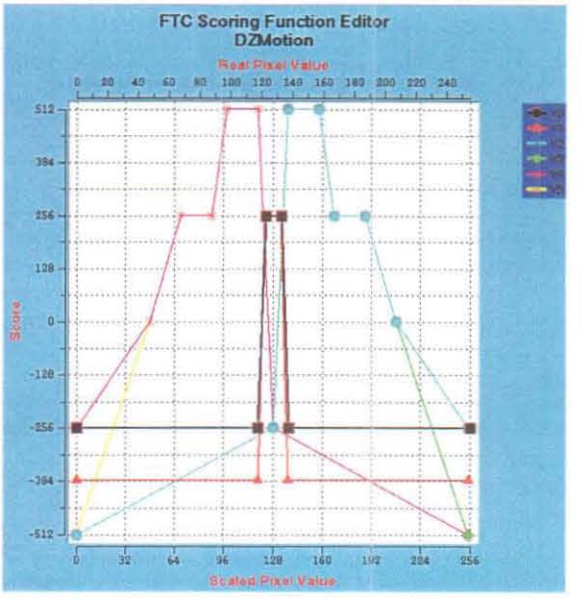
The interest image, together with the *LineStormBoostFactor* and *LineStormAngleTolerance* parameters, is passed to function **BoostLineStormAlignedInterest**, which uses the results from the line storm detector to boost interest pixels that are aligned with and proximate to a line storm (see Section 2.2.7.12.1).

Finally, the interest image is passed to function **SuppressSector** together with the *biasSector, sectorBias,* and *steeringWindSectorBias* parameters to suppress interest pixels whose direction of motion (as inferred from the DZMotion orientation image) are inconsistent with gust fronts (see Section 2.3.1).

Scoring Functions for AUS



Scoring Functions for ABQ



Kernel #1: 5 m/s motion

			2		4
0			2		4
			2		4
0			2		4
			2		4
0			2		4
			2		4
0			2		4
			2		4
1			3		5
			3		5
1			3		5
			2		4
0			2		4
			2		4
0			2		4
			2		4
0			2		4
			2		4
0			2		4
			2		4

Kernel #2: 10 m/s motion

			2		4
0			2		4
			2		4
0			2		4
			2		4
0			2		4
			2		4
0			2		4
			2		4
1			3		5
			3		5
1			3		5
			2		4
0			2		4
			2		4
0			2		4
			2		4
0			2		4
			2		4
0			2		4
			2		4

Kernel #3: 15 m/s motion

			2		4
0			2		4
			2		4
0			2		4
			2		4
0			2		4
			2		4
0			2		4
			2		4
1			3		5
			3		5
1			3		5
			2		4
0			2		4
			2		4
0			2		4
			2		4
0			2		4
			2		4
0			2		4
			2		4

Kernel rotation angles: 0, 20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 220, 240, 260, 280, 300, 320, 340

Figure 22. Scoring functions and templates for the DZMotion reflectivity thin line motion detector. Lower and upper X-axes of scoring functions are the integer-scaled pixel value of the reflectivity time-difference image.

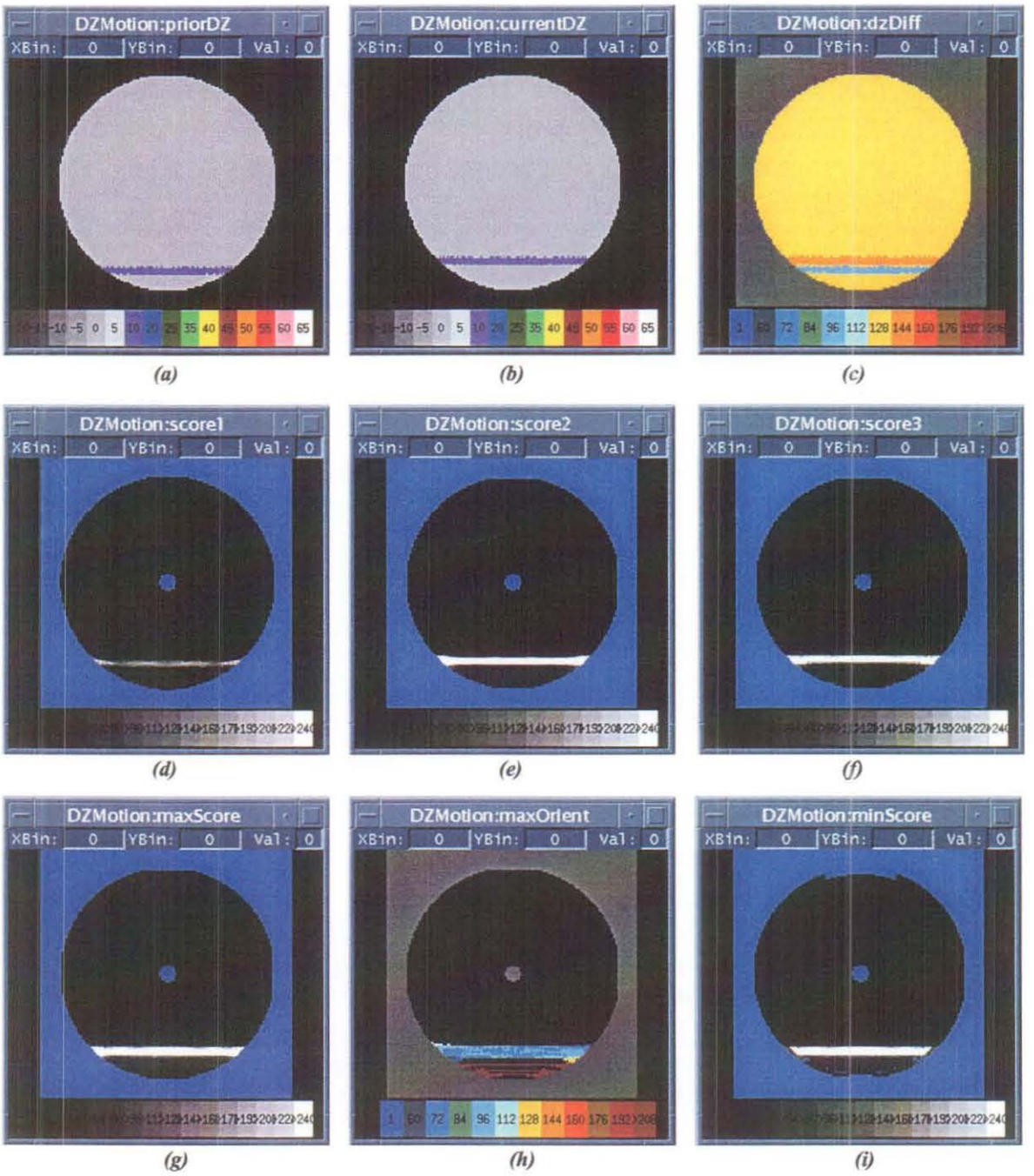


Figure 23. Images illustrating processing steps in reflectivity thin line motion detector (DZMotion) using a simulated thin line signature moving at 10 m/s. (a) Prior reflectivity image (dBZ). (b) Current reflectivity image (dBZ). (c) Scaled and shifted reflectivity difference image. Negative differences are indicated with green/blue colors, while positive differences (> 128) are indicated with yellow/orange colors. (d) Interest image from 5 m/s motion template. (e) Interest image from 10 m/s motion template. (f) Interest image from 15 m/s motion template. (g) Pixel-wise maximum interest. (h) Template match orientation associated with maximum interest scores. (i) Final DZMotion interest image.

2.3.5 TLDZ - Reflectivity Thin Line Detector

2.3.5.1 Parameters

Table 9 lists the default parameter settings for the TLDZ reflectivity thin line detector. The functional template scoring functions, kernels, and kernel rotation angles are shown in Figure 24.

Table 9. Default Parameters for the TLDZ Detector

Parameter Name	Nominal Value	Units
SteeringWindSectorBias	0.25	unitless
LineStormAngleTolerance	45	degrees
LineStormBoostFactor	1.5	unitless
BiasSector	(90, 250)	degrees
SectorBias	0.25	unitless
ConfWeight	1.0	unitless
DisConfWeight	0.3	unitless

2.3.5.2 Mask images

The following pre-computed mask images are used by this detector:

gfBaseImages->detMask
gfBaseImages->stratRain
gfBaseImages->siteMask
gfBaseImages->clutterMap

2.3.5.3 Description

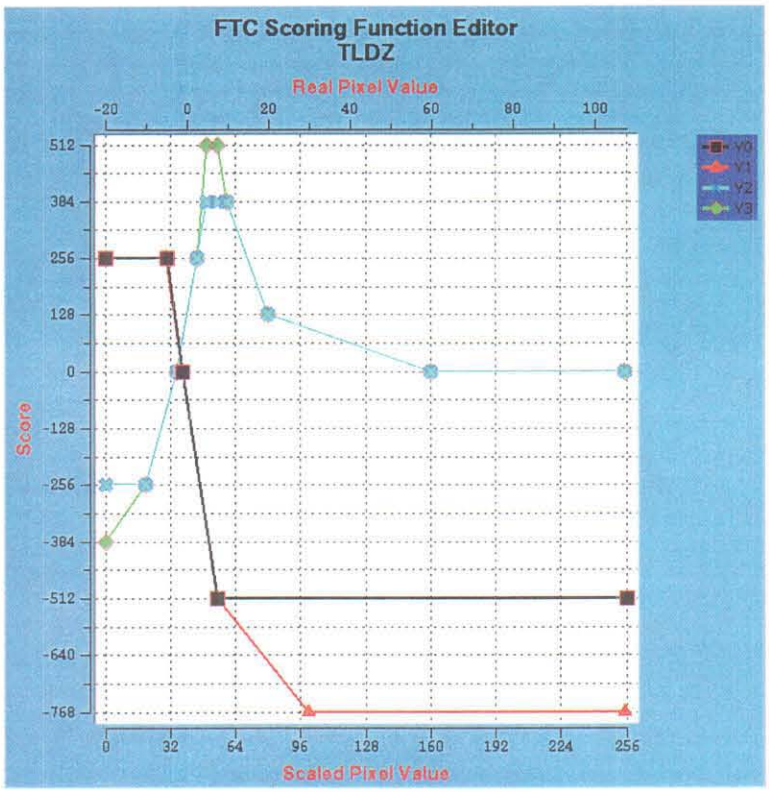
The TLDZ detector identifies low reflectivity thin line echoes in the current reflectivity image that are indicative of gust fronts. The thin line echo is often evident when a gust front moves out ahead of the generating thunderstorm into relatively clear air. Figure 25 illustrates the stages of the TLDZ detector.

Gust front thin lines generally have reflectivities in the range of 5 to 15 dBZ. Consequently, they are not observable if there is any obscuring precipitation. Prior to FTC, a mask image is constructed that corresponds to locations where the *gfBaseImages->stratRain* image has interest values > 128. The mask image is further expanded to include any locations where there is potentially obscuring clutter residue as indicated in the binary *gfBaseImages->clutterMap* image.

The *gfBaseImages->dz* image (dual-beam reflectivity), together with the detection mask described above are supplied as input to FTC operation. Following FTC, **SuppressOOTAlignedInterest** is called to suppress radially aligned thin line interest that is co-located with out-of-trip echoes. TLDZ interest values whose associated orientation angles are within *gfParams->outOfTripAngleTolerance* of the radar azimuth angle are reduced by an amount equal to the corresponding out-of-trip interest (*gfBaseImages->outOfTrip*) multiplied by the reduction factor parameter *gfParams->ootReductionFactorTLDZ* (nominally, 0.75).

The interest image, together with the *LineStormBoostFactor* and *LineStormAngleTolerance* parameters, is passed to function **BoostLineStormAlignedInterest**, which uses the results from the line storm detector to boost interest pixels that are aligned with and proximate to a line storm (see Section 2.2.7.12.1).

Finally, the interest image is passed to function **SuppressSector** together with the *biasSector, sectorBias,* and *steeringWindSectorBias* parameters to suppress interest pixels whose direction of motion (as inferred from the DZMotion orientation image) are inconsistent with gust fronts (see Section 2.3.1).



0				2				0
				2				
0				2				0
				2				
0				2				0
				2				
1				3				1
				3				
1				3				1
				2				
0				2				0
				2				
0				2				0
				2				
0				2				0

Kernel rotation angles: 0, 20, 40, 60, 80, 100, 120, 140, 160

Figure 24. Scoring functions and template for the TLDZ reflectivity thin line detector. Lower X-axis of scoring function is the integer-scaled pixel value. Upper X-axis is reflectivity in dBZ.

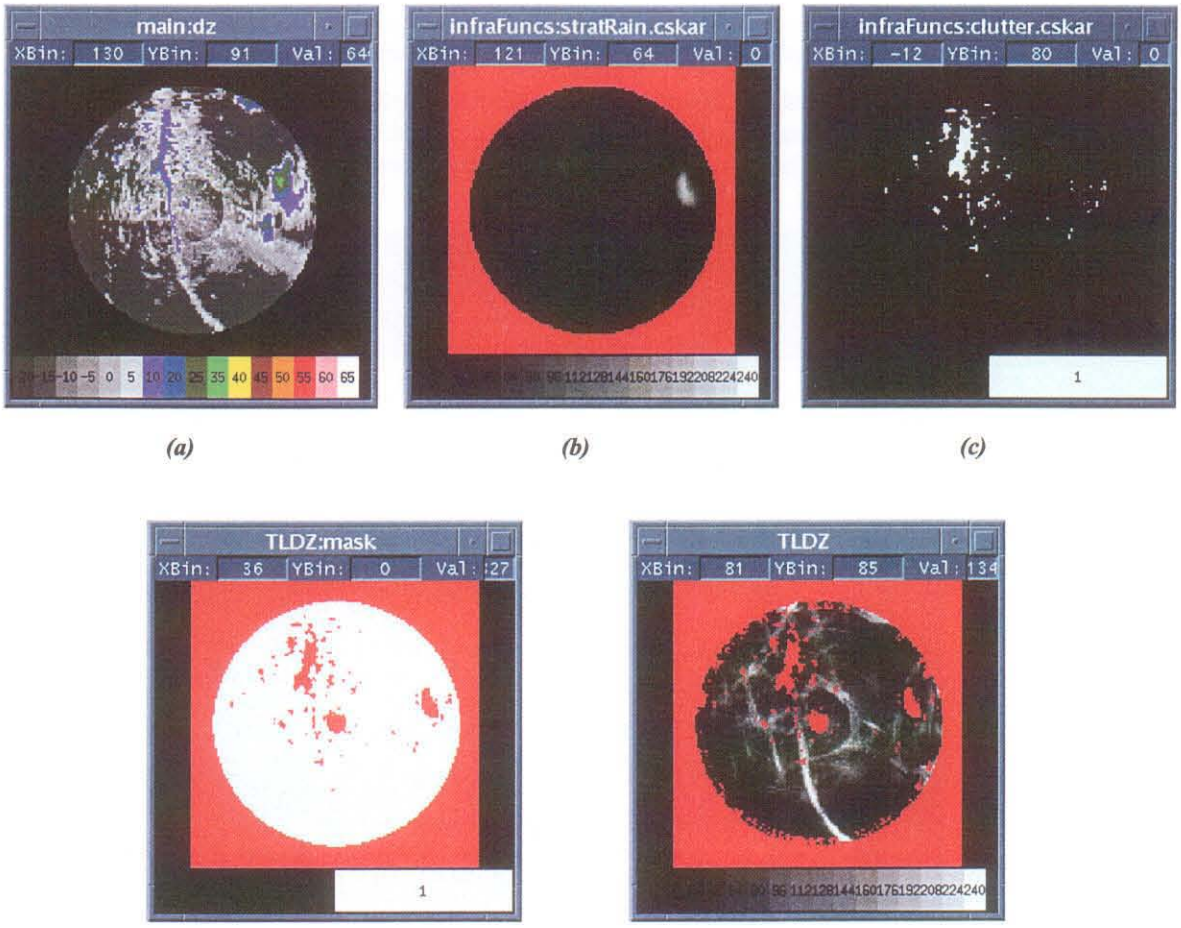


Figure 25. Summary of processing stages of TLDZ reflectivity thin line detector. A gust front thin line echo can be seen as a N-S oriented line of 0-10 dBZ echoes just west of the radar in the input dual-beam reflectivity (a). The corresponding stratiform rain interest image is shown in (b). This stratiform rain image is used together with the corresponding binary clutter mask image (c) and 2-km site mask (not shown) to form the detection mask shown in (d). The resulting TLDZ interest image is shown in (e). Data are from 7/13/99 at Austin, TX

2.3.6 TLSD - Velocity Thin Line Detector

2.3.6.1 Parameters

Table 10 lists the default parameter settings for the TLSD velocity thin line detector. The functional template scoring functions, kernel, and kernel rotation angles are shown in Figure 27.

Table 10. Default Parameters for the TLSD Detector

Parameter Name	Nominal Value	Units
SteeringWindSectorBias	1.00	unitless
LineStormAngleTolerance	45	degrees
LineStormBoostFactor	1.5	unitless
BiasSector	(90, 250)	degrees
SectorBias	0.25	unitless
ConfWeight	1.2	unitless
DisConfWeight	0.0	unitless

2.3.6.2 Mask images

The following images are used as masks by this detector:

gfBaseImages->detSiteMask,
gfBaseImages->stratRain,
gfBaseImages->clutterMap

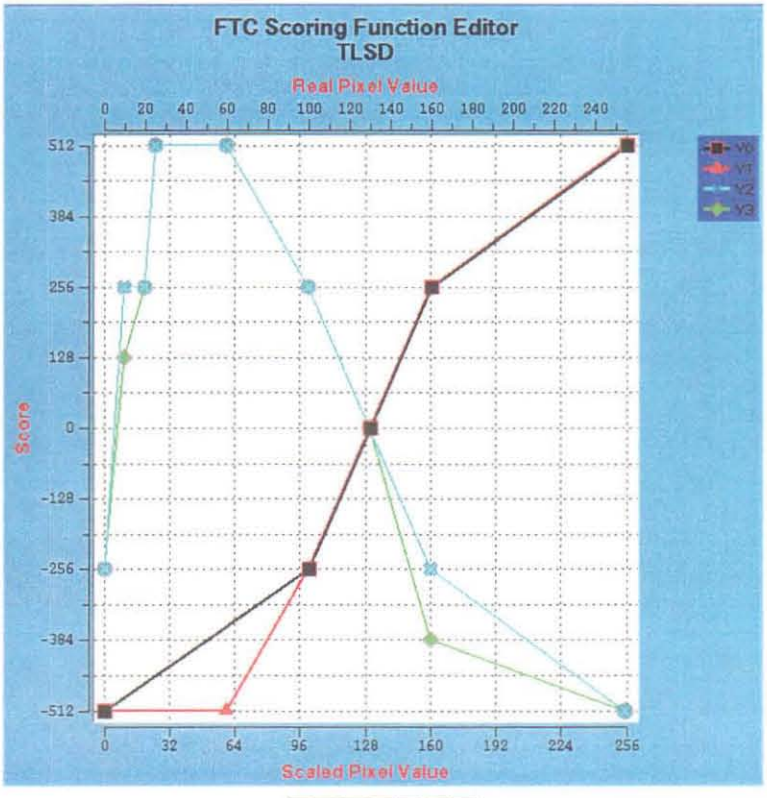
2.3.6.3 Description

The TLSD detector identifies thin line echoes in the low-beam velocity (LBV) image. These appear in LBV images as lines of spatially coherent velocity estimates, embedded in a background of noisy velocity estimates associated with low SNR clear-air regions. The velocity thin line signature cannot be seen if there is obscuring precipitation or if the air is moist enough to produce spatially coherent velocity estimates. Figure 27 illustrates the processing stages of the TLSD detector.

An image representing the local velocity standard deviation was previously computed and stored in *gfBaseImages->sd* as described in section 2.2.7.6. Because the velocity thin line can only be observed when embedded in a region of relatively low SNR, a mask is constructed in order to exclude regions of precipitation or clutter residue from FTC. The mask corresponds to regions where the *gfBaseImages->stratRain* image has interest values > 128 or where clutter residue is indicated in the binary *gfBaseImages->clutterMap* image.

The *gfBaseImages->sd* image, together with the detection mask described above are supplied as input to FTC operation. The resulting interest image, together with the *LineStormBoostFactor* and *LineStormAngleTolerance* parameters, is passed to function **BoostLineStormAlignedInterest**, which uses the results from the line storm detector to boost interest pixels that are aligned with and proximate to a line storm (see Section 2.2.7.12.1).

Finally, the interest image is passed to function **SuppressSector** together with the *biasSector, sectorBias,* and *steeringWindSectorBias* parameters to suppress interest pixels whose direction of motion (as inferred from the DZMotion orientation image) are inconsistent with gust fronts (see Section 2.3.1).



			2			
0			2			0
			2			
0			2			0
			2			
0			2			0
			2			
1			3			1
1			3			1
1			3			1
			2			
0			2			0
			2			
0			2			0
			2			
0			2			0
			2			

Kernel rotation angles: 0, 20, 40, 60, 80, 100, 120, 140, 160

Figure 26. Scoring functions and template for the TLSD velocity thin line detector. Lower and upper X-axes of scoring functions are the integer-scaled pixel value.

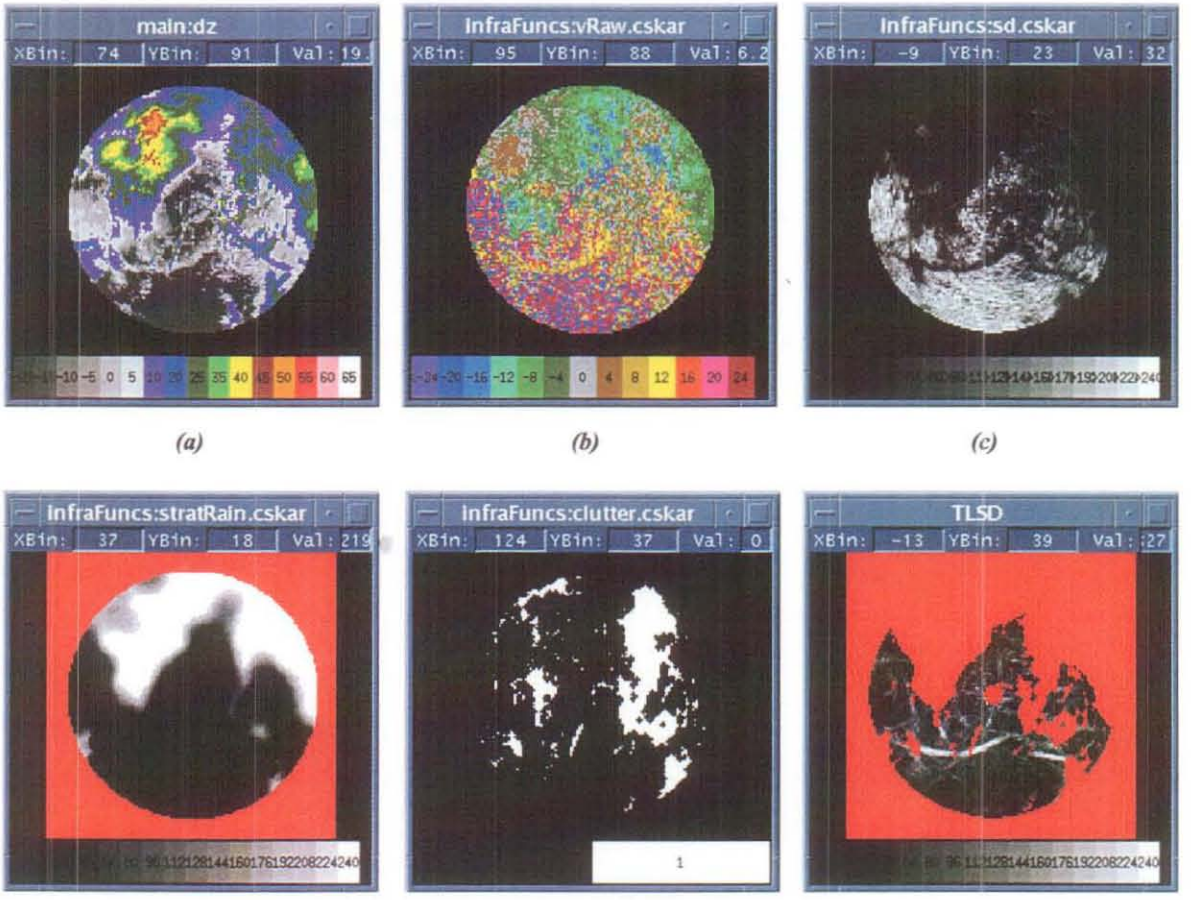


Figure 27. Summary of velocity thin line (TLSD) processing. A gust front located south of the radar is oriented W-E can be seen as a thin line of reflectivity (a) and a corresponding line of spatially coherent positive velocities in (b). The derived velocity standard deviation image that is input to the TLSD detector is shown in (c) where dark areas represent areas of small local standard deviation (coherent velocity). The stratiform rain image (d) and clutter image (e) are used to form a detection mask of nil (no opinion) values that can be seen as red areas in the resulting interest image shown in (f). Data are from 7/28/94 00:09:39 GMT at Albuquerque, NM.

2.3.7 HIDZ - High-Altitude Weather Detector

2.3.7.1 Parameters

Table 11 lists the default parameter settings for the HIDZ high altitude weather detector. The functional template scoring functions, kernel, and kernel rotation angles are shown in Figure 28.

Table 11. Default Parameters for the HIDZ Detector

Parameter Name	Nominal Value	Units
ConfWeight	0.0	unitless
DisConfWeight	4.0	unitless

2.3.7.2 Mask images

The following images are used as masks by this detector:

gfBaseImages->detMask
gfBaseImages->eightKmMask
gfBaseImages->stormCellsSuppressClutter
gfBaseImages->outOfTrip
gfBaseImages->clutterMap
gfBaseImages->dz
gfBaseImages->anticipation
lineStormDetector->interestImg

2.3.7.3 Description

Radar returns from elongated, low reflectivity precipitation regions can produce signatures that closely resemble gust front thin line signatures. However, since gust fronts are a near-surface phenomenon, less power is typically returned from the high receiving beam of the radar than from the low beam. For rain echoes, which typically extend to high altitudes, this is often not the case.

The HIDZ detector generates negative interest wherever reflectivity from the high beam of the radar exceeds that of the low beam. This negative interest serves to counteract positive interest that might be generated from the reflectivity thin line detectors. A difference image representing the subtraction of low-beam reflectivity from high-beam reflectivity (*gfBaseImages->dzDiff*) is pre-computed by function **ComputeDZDiff** during earlier image preparation as described in section 2.2.7.4, and is supplied as input to the FTC operation that generates the returned interest image. Figure 29 shows an example of HIDZ interest image produced for a line storm that is preceded by a gust front thin line.

As can be seen in Table 11, negative interest generated by this detector is given a large amount of disconfirming weight. This is done in order to ensure that the negative evidence is sufficient to overcome the multiple instances of positive interest that may be generated from the various thin-line detectors in association with elongated, low reflectivity rain bands. Because this detector has so much potential to reduce the overall combined interest, it is allowed to express opinions only in areas where it is reasonably certain that no gust front could be present. This is done through careful generation of a mask image from a variety of image sources.

Mask formation starts with a copy of the usual *gfBaseImages->detMask*, which excludes the area outside of the detection radius. The *gfBaseImages->eightKmMask* is then used to expand the mask to exclude the 8 km radius over the radar where altitude discrimination is poor.

Because high reflectivity storm regions are not a typical source of false alarms for the thin line detectors, and because these areas might contain embedded gust fronts detectable by the convergence detectors, these high reflectivity areas are excluded using the *gfBaseImages->stormCellsSuppressClutter* image.

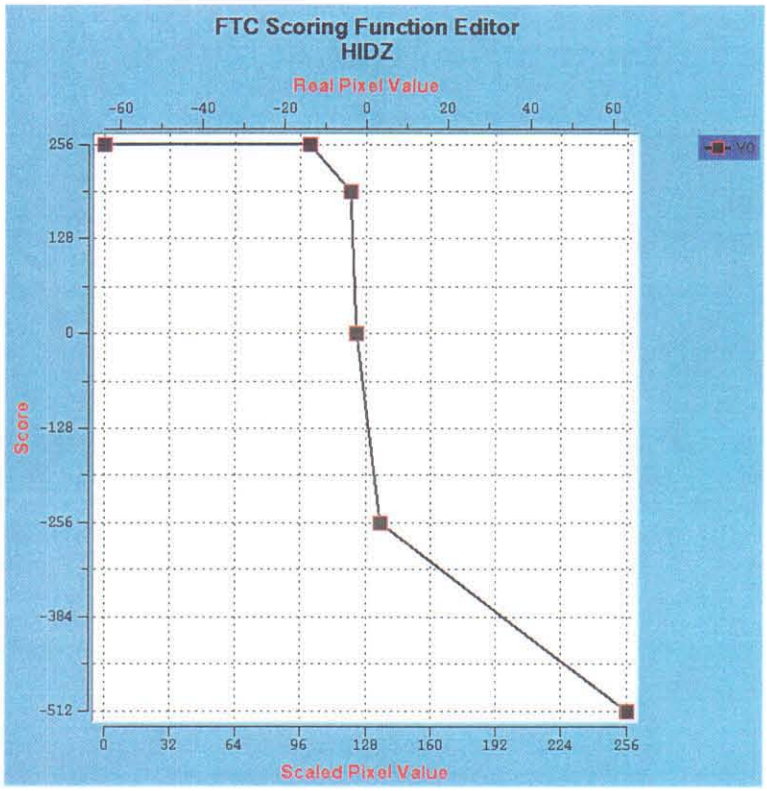
To avoid inadvertently inhibiting detection of a gust front moving through an area of out-of-trip weather contamination, the HIDZ detector is masked from expressing opinions wherever the *gfBaseImages->outOfTrip* image has interest values greater than 48.

Regions of clutter residue (indicated in the *gfBaseImages->clutterMap* image) are also masked since the presence of clutter residue indicates that clutter power is likely biasing the low beam returns; high altitude weather discrimination based on high/low beam reflectivity differencing would not be practical in these areas.

Because the leading edge of a line storm is a favored location for gust fronts and an often difficult area to observe gust front signatures, the *lineStormDetector->interestImg* is used to further expand the mask for excluding HIDZ detection.

Very low SNR regions (as inferred by dual-beam reflectivity values less than -10.0 dBZ) are excluded because the high/low beam power difference is likely to be an unreliable indicator near the system noise level.

Finally, anticipated locations of gust fronts that have been tracked sufficiently long to generate long term anticipation interest (interest values > 255) are masked by binary thresholding the *gfBaseImages->anticipation image* at 255 and performing a 5x5 dilation of the remaining long-term anticipation pixels.



	1	
1	1	1
	1	

Figure 28. Scoring functions and template for the HIDZ high altitude weather detector. Lower X-axis of scoring function is the integer-scaled pixel value. Upper X-axis is the high/low beam dBZ difference.

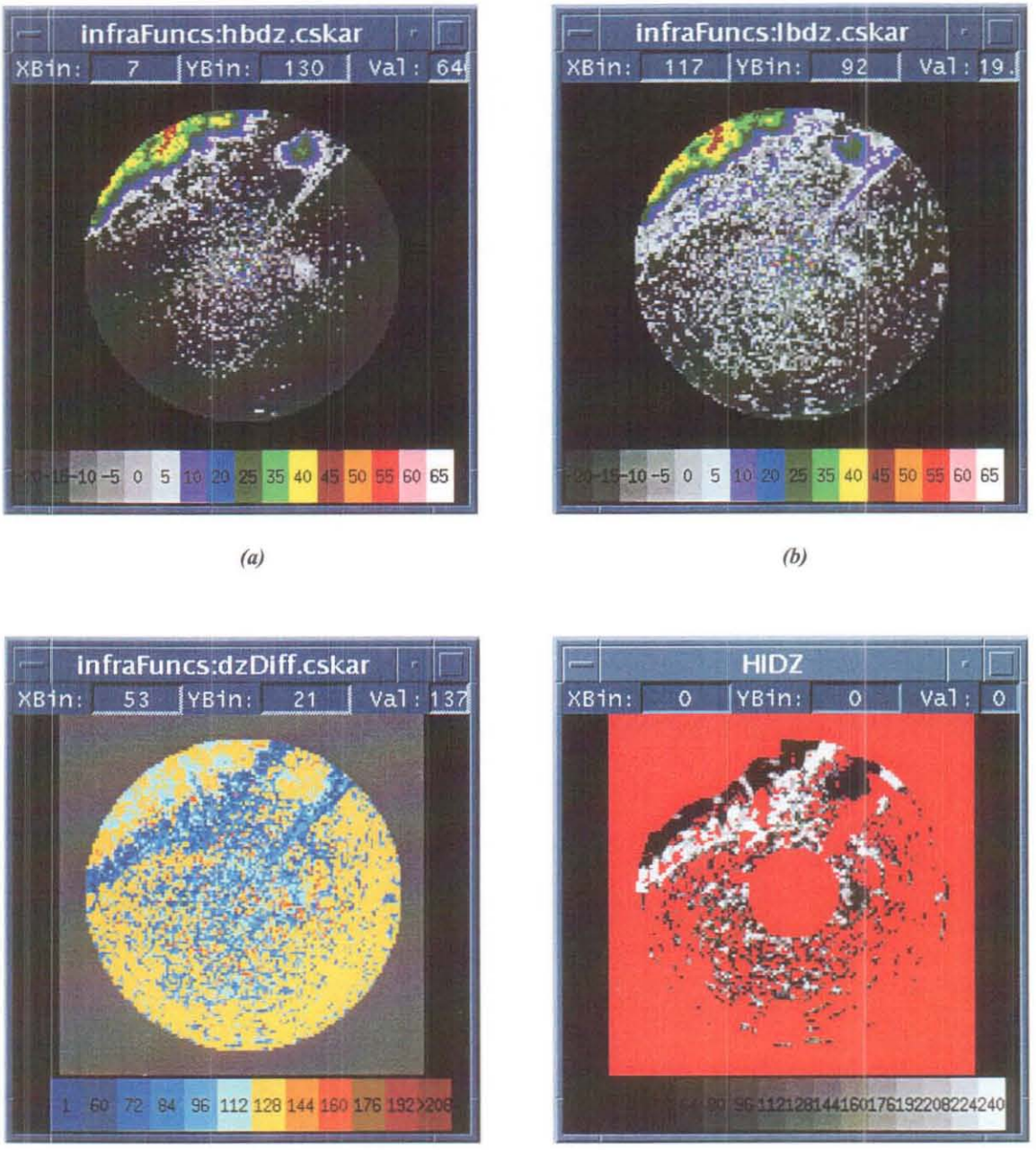


Figure 29. Example output of high altitude weather detector (HIDZ). (a) and (b) are the corresponding low and high beam reflectivity images, respectively. (c) Integer-scaled reflectivity difference after shifting by +128. Blue colors indicate areas where low beam reflectivity exceeds the high beam (values < 128), and yellow/red colors indicate areas where high beam reflectivity exceeds the low beam (values > 128). (d) Final HIDZ interest image. Areas where high altitude weather is present are indicated with negative interest values (black areas). See text for explanation of interest masking (red pixels). Data are from 03/08/99 17:57:46 GMT at Austin, TX.

2.3.8 SDMotion - Velocity Thin Line Motion Detector

2.3.8.1 Parameters

Table 12 lists the default parameter settings for the SDMotion velocity thin line motion detector. The functional template scoring functions, kernel, and kernel rotation angles are shown in Figure 30.

Table 12. Default Parameters for the SDMotion Detector

Parameter Name	Nominal Value	Units
SteeringWindSectorBias	1.00	unitless
LineStormAngleTolerance	45	degrees
LineStormBoostFactor	1.5	unitless
BiasSector	(90, 250)	degrees
SectorBias	0.25	unitless
ConfWeight	1.2	unitless
DisConfWeight	0.3	unitless

2.3.8.2 Mask images

The following images are used as masks by this detector:

gfBaseImages->detSiteMask,
gfBaseImages->stratRain

2.3.8.3 Description

Gust front boundaries that have propagated into clear air may appear as thin lines of spatially coherent, low standard deviation velocity values surrounded by areas of high velocity standard deviation. The movement of these coherent velocity lines is indicative of a gust front. The SDMotion detector identifies moving thin lines of low velocity standard deviation as computed from the current and prior-scan Doppler velocity fields (*gfBaseImages->sd, gfBaseImags->sdPrior).* Figure 31 summarizes the processing stages of the SDMotion detector.

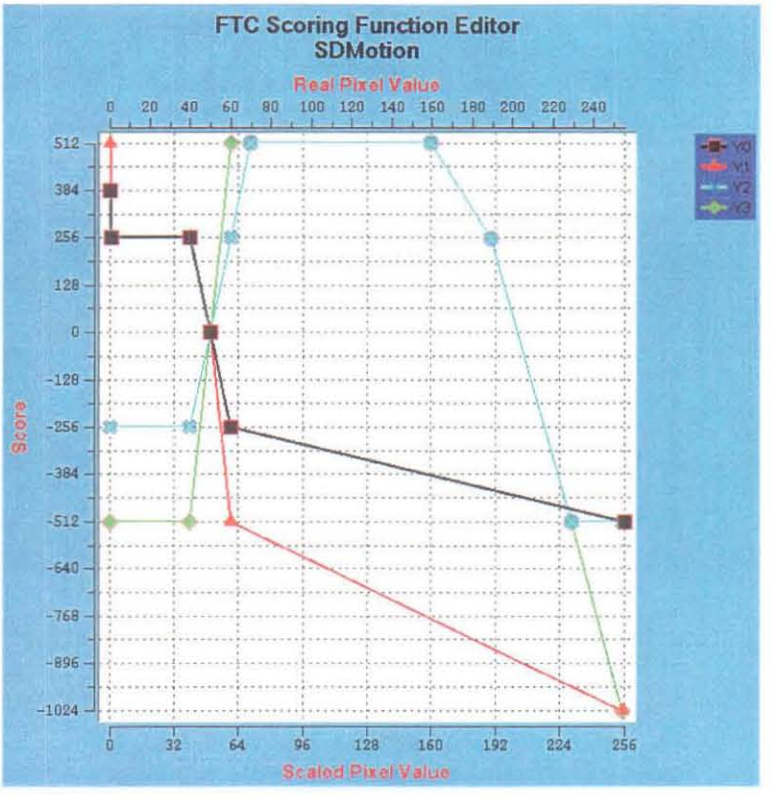
Because the velocity thin line can only be observed when embedded in a region of relatively low SNR, a mask is constructed in order to exclude regions of precipitation and clutter residue from FTC processing. Specifically, the mask corresponds to regions where the *gfBaseImages->stratRain* interest image has interest values greater than 128 (the *gfBaseImages->stratRain* image includes the clutter residue locations). The precipitation mask is dilated by calling **GrayScaleDilate** with a 3x3 dilation kernel. Following mask computation, the current and prior velocity standard deviation images are pixel-wise subtracted to yield a motion difference image named *sdMotion* that serves as the input to FTC together with the mask image (the velocity standard deviation images were previously computed as described in section 2.2.7.6 and stored in variables *gfBaseImages->sd* and *gfBaseImages->sdPrior*).

The *sdMotion* image, together with the detection mask described above are supplied as input to the FTC operation. The resulting interest image, together with the *LineStormBoostFactor* and

LineStormAngleTolerance parameters, is passed to function **BoostLineStormAlignedInterest**, which uses the results from the line storm detector to boost interest pixels that are aligned with and proximate to a line storm (see Section 2.2.7.12.1).

The interest image is then passed to function **SuppressSector** together with the *biasSector, sectorBias,* and *steeringWindSectorBias* parameters to suppress interest pixels whose direction of motion (as inferred from the DZMotion orientation image) are inconsistent with gust fronts (see Section 2.3.1).

Finally, high interest values that may arise from partial pattern matches at the border between the radar data and the *nil* pixels in the outer image mask are reduced by limiting all interest values to a maximum of 128 within a 1.5 km wide annulus whose maximum radius coincides with edge of the data being processed (as given by the *gfConfig->cartRadiusKM* parameter).



		2	2	2		
0			2			0
		2	2	2		
0			2			0
		2	2	2		
0			2			0
		2	2	2		
1			3			1
1		3	3	3		1
1			3			1
		2	2	2		
0			2			0
		2	2	2		
0			2			0
		2	2	2		
0			2			0
		2	2	2		

Kernel rotation angles: 0, 20, 40, 60, 80, 100, 120, 140, 160

Figure 30. Scoring functions and template for the SDMotion velocity standard deviation line motion detector. Lower and upper X-axes of scoring functions are the integer-scaled pixel value.

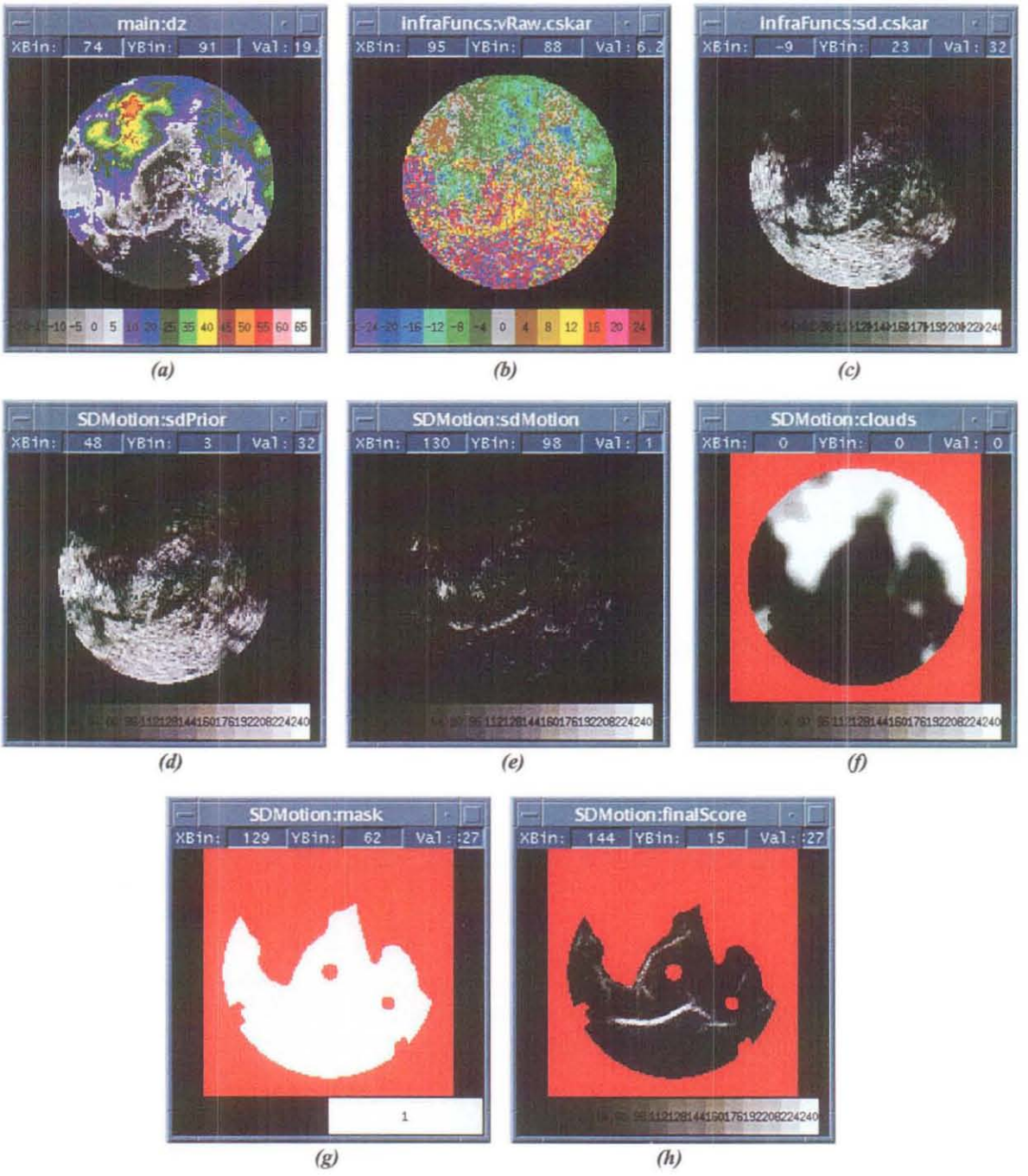


Figure 31. Summary of velocity thin line motion (SDMotion) processing. A gust front located south of the radar is oriented W-E can be seen as a thin line of reflectivity (a) and a corresponding line of spatially coherent positive velocities in (b). (c) Current velocity standard deviation image computed from (b). (d) Prior velocity standard deviation image. (e) Difference image produced by subtracting (c) from (d). The stratiform rain image (d) and clutter image (e) are used to form a detection mask of nil (no opinion) values that can be seen as red areas in the resulting interest image shown in (f). Data are from 7/28/94 00:09:39 GMT at Albuquerque, NM.

2.3.9 TLSDDZ - Tandem Velocity Thin Line / No Reflectivity Detector

2.3.9.1 Parameters

Table 13 lists the default parameter settings for the tandem TLSDDZ Velocity Thin Line / No Reflectivity detector. The functional template scoring functions, kernel, and kernel rotation angles are shown in Figure 32.

Table 13. Default Parameters for the TLSDDZ Detector

Parameter Name	Nominal Value	Units
LineStormAngleTolerance	45	degrees
LineStormBoostFactor	1.5	unitless
ConfWeight	1.0	unitless
DisConfWeight	1.0	unitless

2.3.9.2 Mask images

The following images are used as masks by this detector:

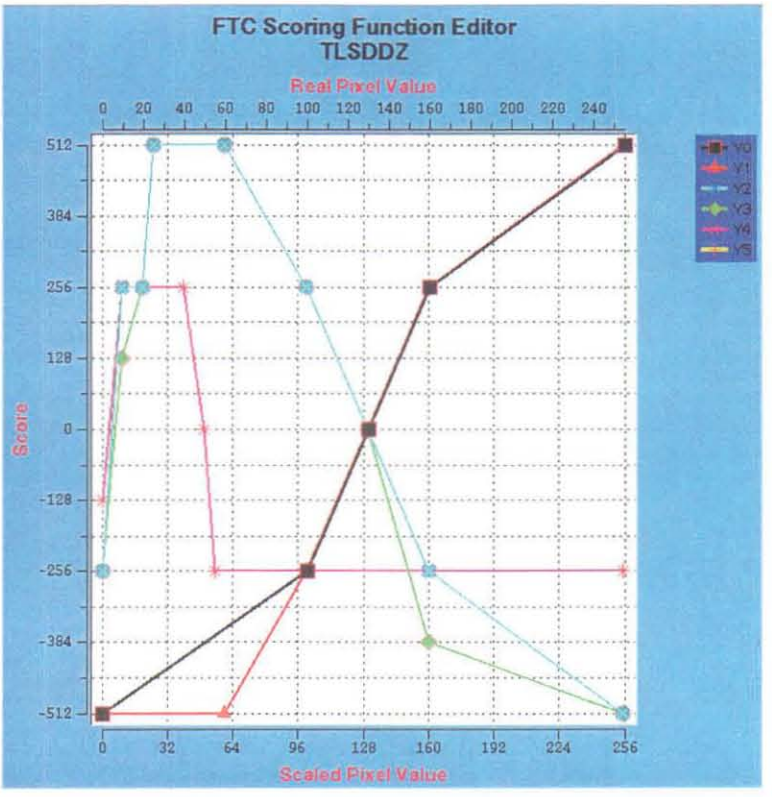
gfBaseImages->detSiteMask,
gfBaseImages->stratRainExpanded

2.3.9.3 Description

The TLSDDZ detector is a tandem detector that looks for velocity thin lines that do not have any appreciable corresponding reflectivity. This exploits a feature of the WSP base data processing whereby radar returns from very weak gust front boundaries may be just strong enough to produce a line of coherent velocity estimates, but not strong enough to generate a reflectivity thin line echo. Such disassociated velocity thin lines have nearly always been found to be indicative of gust fronts.

The detectTLSDDZ functional template is jointly applied to both the *gfBaseImages->sd* and *gfBaseImages->dz* images. This detector has no opinion where expanded stratiform rain is present (note that the stratiform rain image used as a mask includes clutter breakthrough flagged by the WSP), or at locations within short range of the radar. Following FTC, the interest image, together with the *LineStormBoostFactor* and *LineStormAngleTolerance* parameters, is passed to function **BoostLineStormAlignedInterest**, which uses the results from the line storm detector to boost interest pixels that are aligned with and proximate to a line storm (see Section 2.2.7.12.1). Finally, confirming interest score values (values > 127) are further enhanced by a factor of 1.25.

Figure 33 illustrates the processing of the TLSDDZ detector. In the example shown, the gust front is visible only as a thin line of coherent inbound velocities oriented N-S and approaching the radar from the west in the *gfBaseImages->vRaw* image shown in Figure 33b. The signal is too weak to produce a corresponding thin line in the reflectivity image shown in Figure 33a. The velocity thin line is seen as a line of relatively low velocity standard deviation as indicated by the dark pixels in the *gfBaseImages->sd* image shown in Figure 33c. The resulting interest image after application of FTC is shown in Figure 33d.



Kernel #1: SD

			2			
0			2			0
			2			
0			2			0
			2			
0			2			0
			2			
1			3			1
1			3			1
1			3			1
			2			
0			2			0
			2			
0			2			0
			2			
0			2			0
			2			

Kernel #2: DZ

			4			
			4			
			4			
			4			
			4			
			4			
			4			
			5			
			5			
			5			
			4			
			4			
			4			
			4			
			4			
			4			

Kernel rotation angles: 0, 20, 40, 60, 80, 100, 120, 140, 160

Figure 32. Scoring functions and templates for the TLSDDZ velocity thin line with no reflectivity detector. Kernel #1 utilizes scoring functions 0, 1, 2, and 3, and is applied to the velocity standard deviation image. Kernel #2 utilizes scoring functions 4 and 5 and is simultaneously applied to the reflectivity image. Lower and upper X-axes of scoring functions are the integer-scaled pixel value.

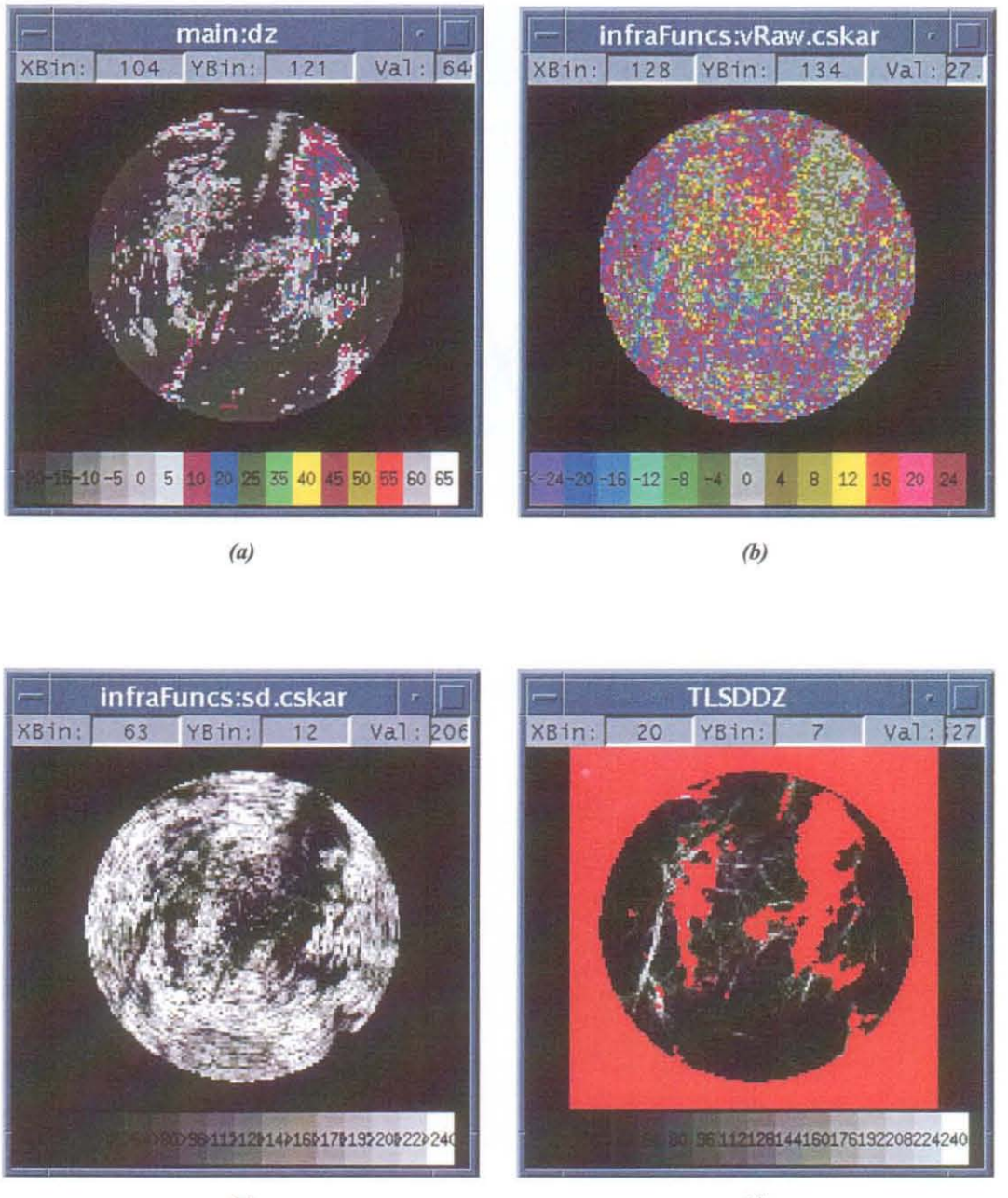


Figure 33. Example output of velocity thin line with no reflectivity detector (TLSDDZ). (a) Reflectivity image. (b) Low beam velocity image. The gust front can be seen as a thin N-S oriented line of inbound velocities (blue) approaching from the west. (c) Velocity standard deviation image computed from (b). (d) Final TLSDDZ interest image. See text for explanation of interest masking (red pixels). Data are from 07/08/00 01:13:03 GMT at Albuquerque, NM.

2.3.10 Converge - Velocity Convergence Detector

2.3.10.1 Parameters

Table 14 lists the default parameter settings for the Converge velocity convergence detector. The functional template scoring functions, kernel, and kernel rotation angles are shown in Figure 34.

Table 14. Default Parameters for the Converge Detector

Parameter Name	Nominal Value	Units
LineStormAngleTolerance	45	degrees
LineStormBoostFactor	1.5	unitless
ConfWeight	1.0	unitless
DisConfWeight	0.0	unitless

2.3.10.2 Mask images

The following images are used as masks by this detector:

gfBaseImages->detMask
gfBaseImages->apMap
gfBaseImages->clutterMap

2.3.10.3 Description

Gust fronts are accompanied by sharp changes in wind speed and/or direction over a relatively short distance. Where there is sufficient SNR and good Doppler velocity measurements, these convergent boundaries can readily be observed in radar Doppler velocity data. However, when gust fronts propagate away from generating thunderstorms into clear air, there may be insufficient sensitivity to produce good Doppler measurements, and the velocity convergence signature may not be observed. Even given sufficient signal returns, the velocity convergence signature tends to diminish whenever a gust front passes over the radar and becomes radially aligned owing to a lack of a radial velocity component (Doppler blindness). For these reasons, the disconfirming weight for this detector is currently set to zero, reflecting the lack of skill in being able to able to state that a gust front is not present at all locations in the imagery.

The WSP Converge detector is a tandem detector that applies simultaneous templates to the *gfBaseImages->conv* and *gfBaseImages->lowResConv* images previously computed during image preparation as described in section 2.2.7.9. The *gfBaseImages->lowResConv* image is used in conjunction with the 1-km scale *gfBaseImages->conv* image in order to help support recognition of broader-scale convergence patterns that may not have a sharply focussed shear boundary.

Figure 35 shows example images summarizing the processing steps in the Converge detector. Before performing the tandem FTC, a copy of the 1-km scale *gfBaseImages->conv* image is rescaled so that the integer-scaled values fall within the required FTC input range of [0,255] (a similar rescaling operation was immediately performed on the *gfBaseImages->lowResConv* image at the time it was created). CSKETCH library function **SKScaleArrayToBounds** is used to linearly rescale the *gfBaseImages->conv* data such that the integer representations of those delta-V values falling within the physical lower and upper bounds given by *gfParams->scaleBoundsConvCellConv* (nominally 0 and 1.5 m/s), map to the interval [0, 255]. Integer encodings of delta-V values that exceed the maximum bound are set to 255, and values less than

the minimum bound are set to 0 (i.e., the data are clipped). Figures 35d and 35e show rescaled convergent shear maps plotted using gray scale.

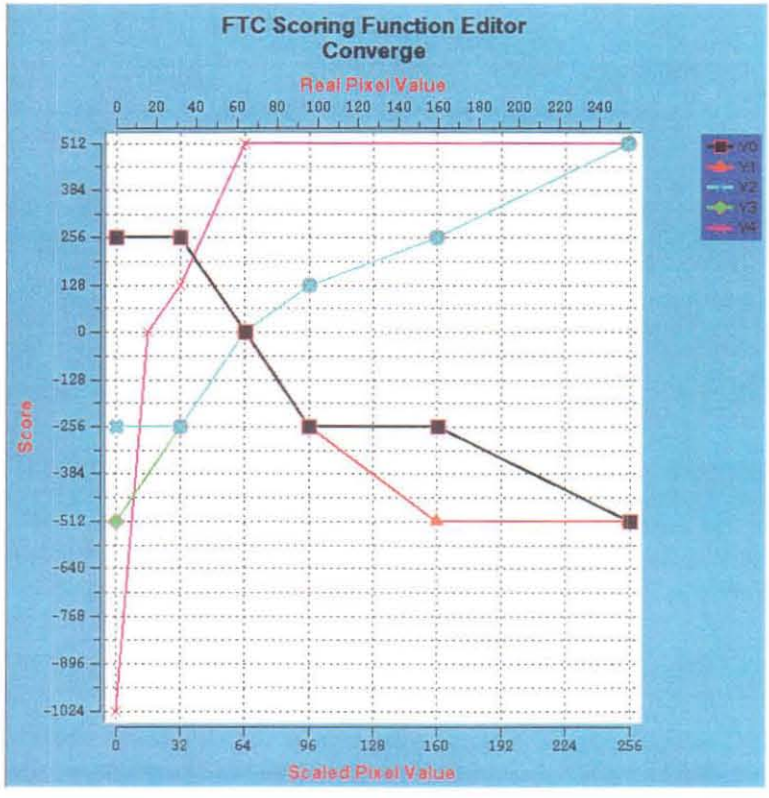
In addition to the standard *gfBaseImages->detMask* masking, areas of missing velocity data as indicated by *nil* pixels in the *gfBaseImages->lowResConv*, are also masked. Areas of AP breakthrough or clutter residue contamination are optionally masked from convergence detection depending on the setting of the *gfConfig->doVMasking* configuration parameter in the *configWSP* parameter file. If *gfConfig->doVMasking* is "0", then no detector masking is performed. Otherwise, a mask is constructed from the logical "OR" of the binary AP and clutter map images (*gfBaseImages->apMap* and *gfBaseImages->clutterMap*) and supplied to the FTC operation to prevent opinions from being generated in those areas. Figures 35h and 35i show the difference in the final interest images with the different settings of *gfConfig->doVMasking*.

Following FTC, the resulting interest image is modified by calling function **AdjustInterestByOrientation** to boost interest values whose associated best-match orientation angle are within 60 degrees of the radar's azimuth angle through the given point. For orientation angle differences of 20 degrees or less, the interest values are boosted (multiplied) by 2.0. For angle differences between 20 and 60 degrees, the boosting factor is a decreasing linear ramp from 2.0 to 1.0. Interest values whose orientation angle difference is greater than 60 degrees are left unchanged. This boosting helps to maintain convergence interest as fronts become radially aligned and the resulting Doppler measurements tend toward 0 m/s. The boosted interest values are clipped to the maximum value given by *gfParams->ftcImageRange* (nominally, 256).

In a vertically sheared environment, the zero-Doppler line will curve with range, reflecting the change in wind direction as the radar beam samples higher altitude winds. Thus, where the zero-Doppler line is oriented across azimuths, there may be an apparent velocity convergence signature. To reduce the probability of false alarms from vertical shear boundaries, a value of 128 is subtracted from the Converge interest wherever the interest values coincide with areas of zero-Doppler lines as indicated in the binary *gfBaseImages->zeroCross* image. The modified interest image is then clipped to a minimum allowable interest value of 0.

Next, the interest image, together with the *LineStormBoostFactor* and *LineStormAngleTolerance* parameters, is passed to function **BoostLineStormAlignedInterest** in order to boost interest pixels that are aligned with and proximate to a line storm (see Section 2.2.7.12.1).

Finally, because convergence signatures are not reliably present in areas of low SNR, the detector is prevented from expressing disconfirming opinions by calling **GFSetElementsInIntervals** to set to *nil* all pixels in the interest image that fall within the disconfirming interest interval of [0, 127] and which fall outside any precipitation areas (i.e., "non-weather" areas where *gfBaseImages->stratRain* has interest values less than 128). Figure 35h shows the final interest image after the selective masking of disconfirming interest values.



Kernel #1: conv

0					2				
					2				0
0					2				
					2				0
0					2				
					2				0
1					3				1
					3				
1					3				1
					2				0
0					2				
					2				0
0					2				
					2				0
0					2				

Kernel #2: lowResConv

					4				
					4				
					4				
					4				
					4				
					4				
					4				
					4				
					4				
					4				
					4				
					4				
					4				
					4				
					4				

Kernel rotation angles: 0, 20, 40, 60, 80, 100, 120, 140, 160

Figure 34. Scoring functions and templates for the Converge velocity convergence detector. Kernel #1 utilizes scoring functions 0, 1, 2, and 3, and is applied to the scaled 1-km convergence image. Kernel #2 utilizes scoring function 4 and is simultaneously applied to the large-scale lowResConv convergence image. Lower and upper X-axes of scoring functions are the integer-scaled pixel value.

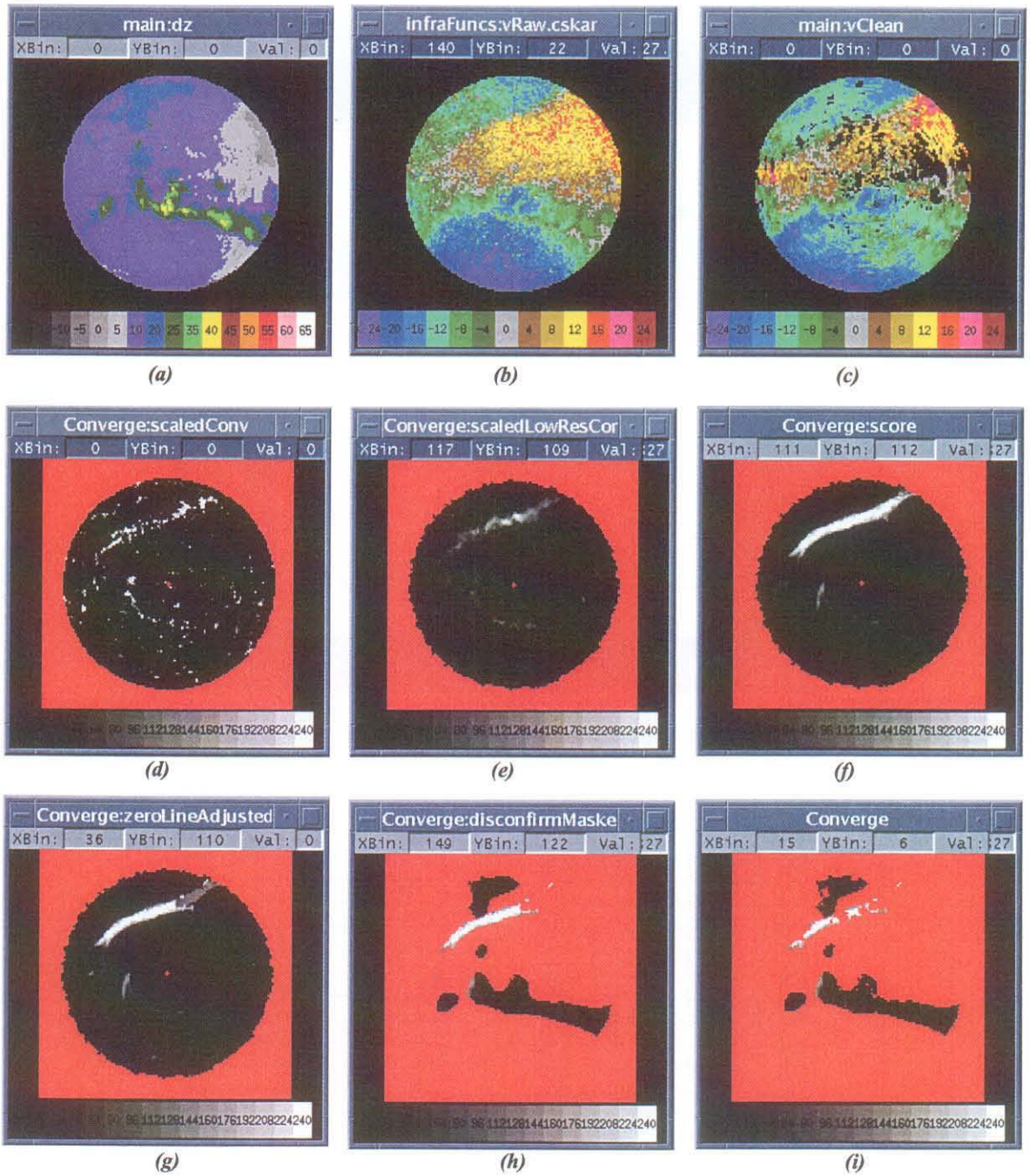


Figure 35. Images illustrating processing steps in velocity convergence detector (Converge). (a) Reflectivity image. (b) Low beam velocity. (c) Cleaned dual-beam velocity. (d) 1-km convergent shear map computed from (b) after rescaling to interval [0, 255]. (e) Low resolution convergent shear map computed from (c) after similar rescaling. (f) Initial interest image after tandem FTC. (g) Interest image after boosting areas of radially aligned interest and suppression in areas of zero-Doppler lines. (h) Final interest image after "non-weather" masking of disconfirming interest. (i) Final interest image as in (h), except with "VMasking" enabled to mask areas of AP and clutter residue contamination. Data are from 5/1/00, 06:55:22 GMT at Austin, TX

2.3.11 ConvMotion - Velocity Convergence Motion Detector

2.3.11.1 Parameters

Table 14 lists the default parameter settings for the ConvMotion velocity convergence motion detector. The functional template scoring functions, kernel, and kernel rotation angles are shown in Figure 36.

Table 15. Default Parameters for the ConvMotion Detector

Parameter Name	Nominal Value	Units
LineStormAngleTolerance	45	degrees
LineStormBoostFactor	1.5	unitless
ConfWeight	1.0	unitless
DisConfWeight	0.0	unitless

2.3.11.2 Mask images

The following images are used as masks by this detector:

gfBaseImages->detMask
gfBaseImages->apMap
gfBaseImages->clutterMap

2.3.11.3 Description

The ConvMotion detector identifies moving boundaries of convergent radial shear. It is actually a tandem feature detector that simultaneously performs FTC on both a convergence time-differenced image as well as the current low-resolution convergence image (*gfBaseImages->lowResConv*). Probing the current low-resolution convergence image in addition to the 1-km scale convergence difference image helps to ensure that the ConvMotion detector will still generate moderate positive interest in those cases where a convergence zone exists, but is too broad to be seen in the motion of the 1-km scale convergence field alone. For the same reasons discussed previously in the description of the Converge detector, the disconfirming weight of this detector is currently set to zero.

Figure 37 summarizes the processing steps of the ConvMotion detector. First, the value of *gfBaseImages->convPrior>DataReady* is checked to see if a prior convergence image exists for which to construct the difference image. If there is no available prior image (as is the case for the first couple of scans after the algorithm has started), then FTC is not performed and the returned interest image is simply set to a moderately disconfirming interest value of 64. Otherwise, processing proceeds as follows.

The prior convergence image (*gfBaseImages->convPrior*) is pixelwise subtracted from the current convergence image (*gfBaseImages->conv*), and the resulting difference values are stored in a local SKArray (*workConv*). Since the ConvMotion detector only looks for lines of positive convergent shear difference, all negative difference values in *workConv* are set to zero.

Before performing the tandem FTC, the *workConv* difference array is rescaled so that the integer-scaled values fall within the required FTC input range of [0,255]. The CSKETCH library function **SKScaleArrayToBounds** is used to linearly remap the *workConv* data such that values falling between the minimum and maximum delta-V difference values given by *gfParams->scaleBoundsConvCellConvMotion* (nominally 0 and 2.0 m/s) correspond to values of 0 and 255, respectively. Delta-V difference values that

exceed the maximum bound are set to the maximum bound, and values less than the minimum bound are set to the minimum bound (i.e., the image data are clipped).

In addition to the standard *gfBaseImages->detMask* masking, areas of AP breakthrough or clutter residue contamination are optionally masked from convergence detection depending on the setting of the *gfConfig->doVMasking* configuration parameter in the *configWSP* parameter file. If *gfConfig->doVMasking* is "0", then no detector masking is performed. Otherwise, a mask is constructed from the logical "OR" of the binary AP and clutter map images (*gfBaseImages->apMap* and *gfBaseImages->clutterMap*) together with a copy of the *gfBaseImages->detMask* and supplied with the *workConv* image to the FTC operation to prevent opinions from being generated in those areas.

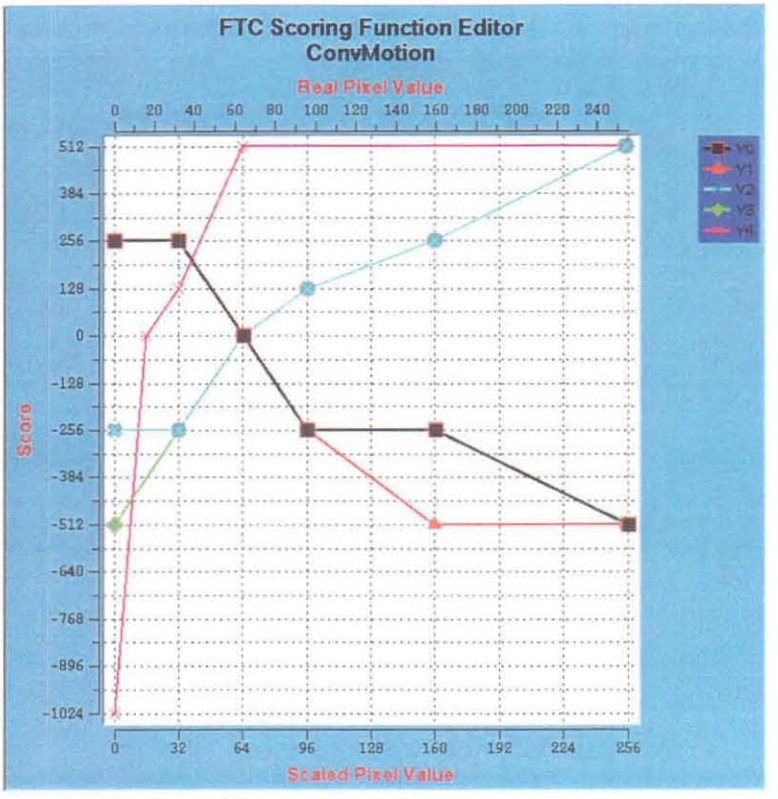
In similar fashion to the Converge detector, the resulting interest image is then modified by calling function **AdjustInterestByOrientation** to boost radially aligned interest. Interest values whose associated orientation angle differ from the radar azimuth angle by 20 degrees or less are boosted (multiplied) by 3.0. For angle differences between 20 and 60 degrees, the boosting factor is computed as a decreasing linear ramp from 3.0 to 1.5. Interest values whose orientation angle difference is greater than 60 degrees are left unchanged. Again, this boosting helps to maintain convergence interest as fronts become radially aligned and the resulting Doppler measurements tend toward 0 m/s. The boosted interest values are clipped to the maximum value given by *gfParams->ftcImageRange* (nominally, 256).

Finally, interest values that coincide with zero-Doppler lines are reduced by subtracting 128 from the ConvMotion interest wherever these occur. The modified interest image is then clipped to a minimum allowable interest value of 0.

Next, the interest image, together with the *LineStormBoostFactor* and *LineStormAngleTolerance* parameters, is passed to function **BoostLineStormAlignedInterest** in order to boost interest pixels that are aligned with and proximate to a line storm (see Section 2.2.7.12.1).

As with the Converge detector, because convergence signatures are not reliably present in areas of low SNR, the ConvMotion detector is prevented from expressing disconfirming opinions by calling **GFSetElementsInIntervals** to set to *nil* all pixels in the interest image that fall within the disconfirming interest interval of [0, 127] and which fall outside any precipitation areas (i.e., "non-weather" areas where *gfBaseImages->stratRain* has interest values less than 128).

Finally, high interest values that may arise from partial pattern matches at the border between the radar data and the *nil* pixels in the outer image mask are reduced by limiting all interest values to a maximum of 128 within a 1.5 km wide annulus whose maximum radius coincides with edge of the data being processed (as given by the *gfConfig->cartRadiusKM* parameter). Figure 37i shows the final interest image after the selective masking of disconfirming interest values.



Kernel #1: conv

0				2					
				2					0
0				2					
				2					0
0				2					
				2					0
	1			3				1	
				3					
	1			3				1	
				2					0
0				2					
				2					0
0				2					
				2					0
0				2					

Kernel #2: lowResConv

				4					
				4					
				4					
				4					
				4					
				4					
				4					
				4					
				4					
				4					
				4					
				4					
				4					
				4					
				4					

Kernel rotation angles: 0, 20, 40, 60, 80, 100, 120, 140, 160

Figure 36. Scoring functions and templates for the ConvMotion velocity convergence motion detector. Kernel #1 utilizes scoring functions 0, 1, 2, and 3, and is applied to the scaled 1-km convergence image. Kernel #2 utilizes scoring function 4 and is simultaneously applied to the large-scale lowResConv convergence image. Lower and upper X-axes of scoring functions are the integer-scaled pixel value.

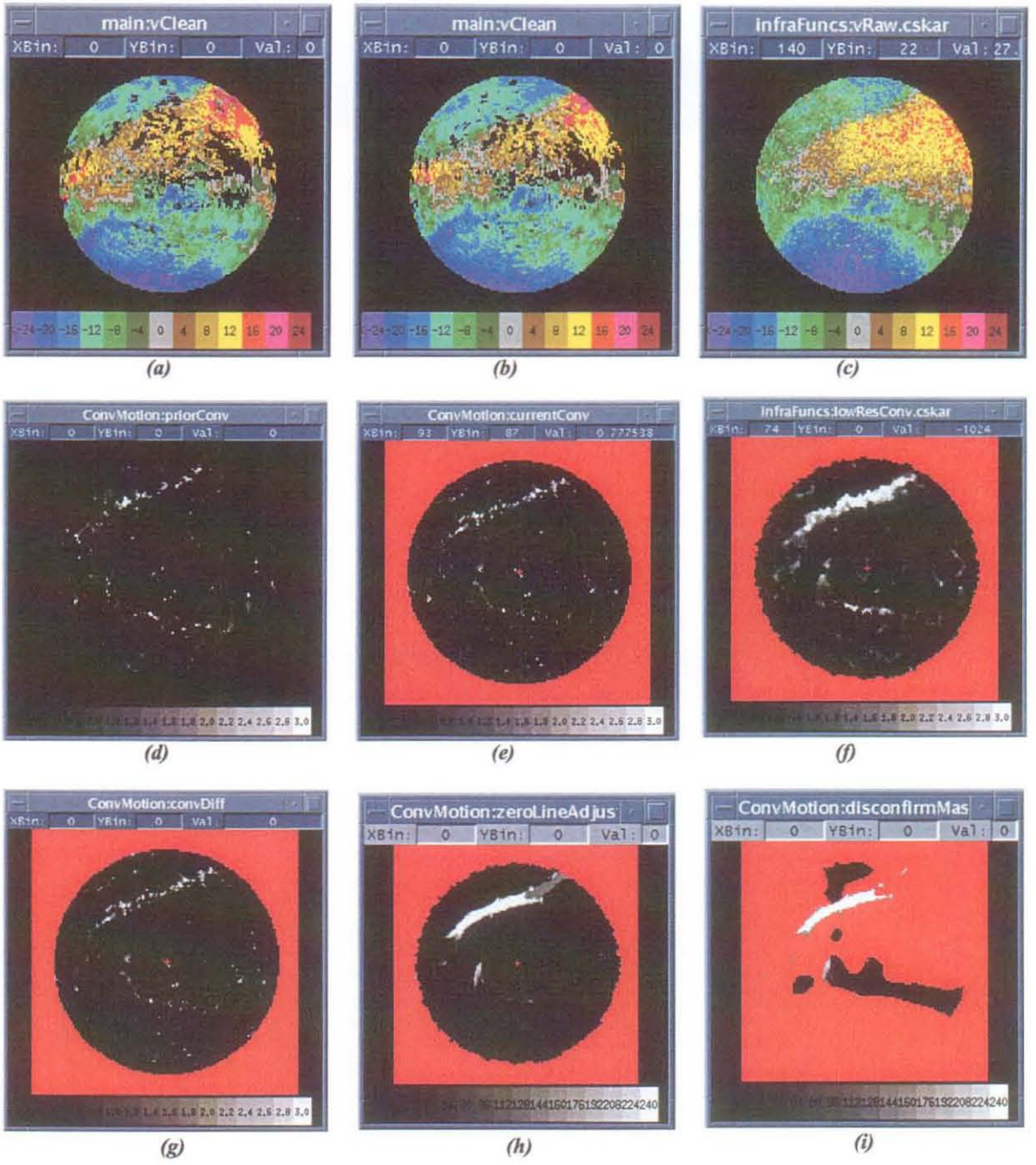


Figure 37. Images illustrating processing steps in velocity convergence motion (ConvMotion). (a) Prior cleaned dual-beam velocity. (b) Current cleaned dual-beam velocity. (c) Current low-beam velocity. (d) Prior 1-km convergent shear map computed from (a). The gray scale ranges from 0 to 3 m/s. (e) Current 1-km convergent shear map computed from (b). (f) Current low-resolution convergent shear map computed from (c). (g) Convergence difference image produced by subtracting (e) from (d). (h) Interest image after adjustments for radial orientation, zero Doppler lines, and line storms. (i) Final interest image after "non-weather" masking of disconfirming interest (see reflectivity image in Figure 35a). Data are from 5/1/00, 06:55:22 GMT at Austin, TX

2.3.12 DZSDMotion - Tandem Reflectivity/Velocity Thin Line Motion Detector

2.3.12.1 Parameters

Table 16 lists the default parameter settings for the DZSDMotion detector. The functional template scoring functions, kernel, and kernel rotation angles are shown in Figure 38.

Table 16. Default Parameters for the DZSDMotion Detector

Parameter Name	Nominal Value	Units
SteeringWindSectorBias	1.00	unitless
LineStormAngleTolerance	45	degrees
LineStormBoostFactor	1.5	unitless
BiasSector	(90, 250)	degrees
SectorBias	0.25	unitless
ConfWeight	1.0	unitless
DisConfWeight	0.0	unitless

2.3.12.2 Mask images

The following images are used as masks by this detector:

gfBaseImages->detSiteMask,
gfBaseImages->stratRainExpanded,

2.3.12.3 Description

The DZSDMotion detector identifies reflectivity thin lines that are coincident with moving velocity thin lines. Figure 39 illustrates the processing steps. Velocity thin lines are recognized as lines of spatially coherent velocity estimates (low standard deviation), embedded in a background of noisy velocity estimates associated with low SNR clear-air regions. Because the velocity thin line can only be observed when embedded in a region of relatively low SNR, a mask is constructed in order to exclude moving regions of precipitation from FTC processing. Specifically, the mask corresponds to regions where the *gfBaseImages->stratRainExpanded* interest image has interest values greater than 128, and where the associated Doppler value obtained from the low-beam velocity image (*gfBaseImages->vRaw*) is essentially non-zero ($|vRaw| > 0.5$ m/s).

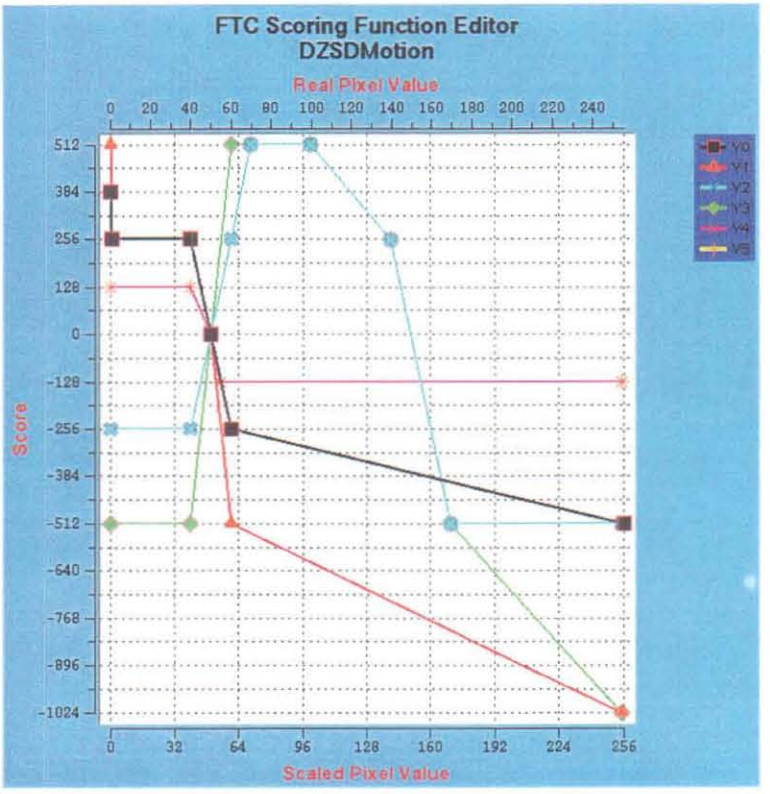
Once the mask has been prepared, the current and prior velocity standard deviation images are pixel-wise subtracted to yield a motion difference image named *sdMotion* that serves as the input to FTC together with the mask image (the velocity standard deviation images were previously computed as described in section 2.2.7.6 and stored in variables *gfBaseImages->sd* and *gfBaseImages->sdPrior*).

Following FTC, interest features that are in close proximity and aligned with line storms (if any) are boosted by calling function **BoostLineStormAlignedInterest** together with the *LineStormBoostFactor* and *LineStormAngleTolerance* parameters. Next, **SuprressSector** is called together with the *biasSector, sectorBias, and steeringWindSectorBias* parameters in order to possibly suppress interest values whose

motion as inferred from the DZMotion orientation image is unlikely to be associated with a gust front. See Section 2.3.1 for a complete description of the suppression logic.

Following the dynamic interest adjustments, a further adjustment is made to highlight confirming interest values (values greater than 127), by multiplying only the confirming values by a constant boost factor of 1.25.

Finally, high interest values that may arise from partial pattern matches at the border between the radar data and the *nil* pixels in the outer image mask are reduced by limiting all interest values to a maximum of 128 within a 1.5 km wide annulus whose maximum radius coincides with edge of the data being processed (as given by the *gfConfig->cartRadiusKM* parameter).



Kernel #1: SDMotion

		2			
0		2			0
		2			
0		2			0
		2			
0		2			0
		2			
1		3			1
1		3			1
1		3			1
		2			
0		2			0
		2			
0		2			0
		2			
0		2			0
		2			

Kernel #2: DZ

		4			
		4			
		4			
		4			
		4			
		4			
		4			
		5			
		5			
		5			
		4			
		4			
		4			
		4			
		4			
		4			

Kernel rotation angles: 0, 20, 40, 60, 80, 100, 120, 140, 160

Figure 38. Scoring functions and templates for the DZSDMotion (reflectivity thin line with velocity thin line motion) detector. Kernel #1 utilizes scoring functions 0, 1, 2, and 3, and is applied to the velocity standard deviation motion difference image. Kernel #2 utilizes scoring functions 4 and 5 and is simultaneously applied to the reflectivity image. Lower and upper X-axes of scoring functions are the integer-scaled pixel value

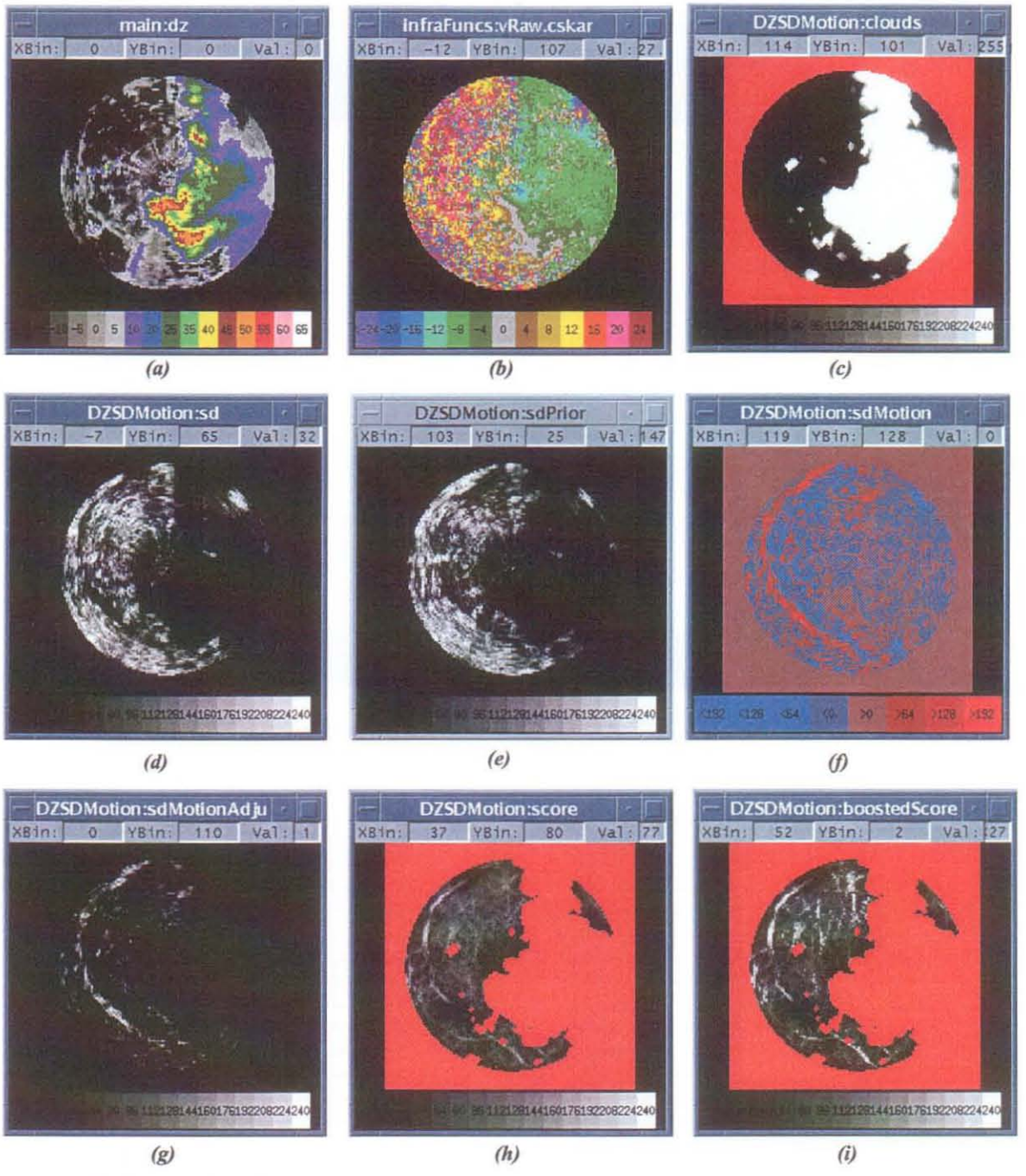


Figure 39. Images illustrating processing steps in tandem reflectivity thin line/velocity thin line motion detector (DZSDMotion). A weak gust front is present near the left edge of the images. (a) Current reflectivity. (b) Current low-beam velocity. (c) StratRainExpanded image used for mask. (d) Current velocity standard deviation image (sd) computed from (b). (e) Prior velocity standard deviation image. (f) SDMotion difference image formed by subtracting (e) from (d). (g) SDMotion image after setting negative differences to zero. (h) Interest image after FTC. (i) Final interest image after boosting confirming interest scores by 1.5. Data are from 7/5/99, 22:06:47 GMT at Albuquerque, NM

2.3.13 PrecipEdges - Precipitation Edge Detector

2.3.13.1 Parameters

Table 17 lists the default parameter settings for the PrecipEdges detector. The functional template scoring functions, kernel, and kernel rotation angles are shown in Figure 40.

Table 17. Default Parameters for the PrecipEdges Detector

Parameter Name	Nominal Value	Units
ConfWeight	0.0	unitless
DisConfWeight	4.0	unitless

2.3.13.2 Mask images

The following images are used as masks by this detector:

gfBaseImages->detMask,
lineStormDetector->interestImg
DetectConverge->convergeImg
DetectConvMotion->convMotionImg

2.3.13.3 Description

The edges of low-reflectivity (stratiform) rain echo regions tend to generate false alarms owing to the fact that reflectivity levels of such rain echoes overlap those found in gust front thin lines. When the region outside of the stratiform rain is clear, then both the center line and one of the flanking strips of indices from the reflectivity thin line functional template are contributing positive scores. This can also be a problem for thin line motion detection, since the leading edge of a moving stratiform rain echo produces a positive difference line in the difference image that is used as the basis of thin line motion detection. Consequently, a functional template designed to look for edges is used to highlight the boundaries between stratiform rain and clear air. The interest values are inverted, producing a map of low interest wherever precipitation edges exist.

The PrecipEdges computation begins by setting up the mask that will be used to prevent interest generation at inappropriate locations in the image. The mask is built in stages, starting with a copy of the basic *gfBaseImages->detMask* that defines the outer border area of *nil* pixels beyond the radius of the actual data. Additional pixels in the mask image are then set to *nil* in the following image regions:

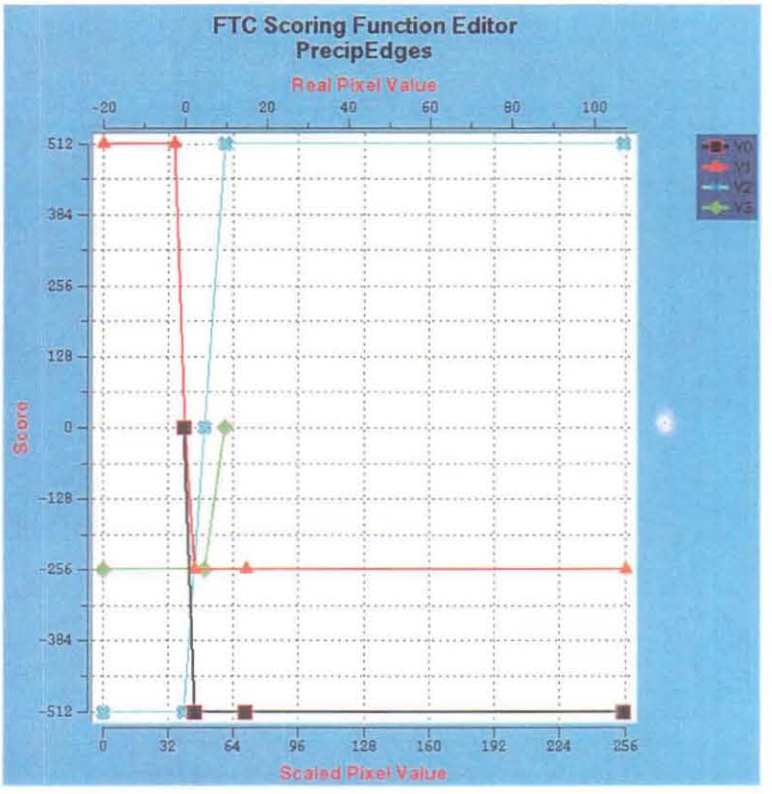
1. Expected locations of long-term tracked gust fronts as indicated in the current Anticipation interest image.
2. Areas of line storms as indicated by the current line storm interest image.
3. Areas where there is velocity convergence as indicated by the *Converge* or *ConvMotion* interest images.

The relatively large amplitude scoring functions in the PrecipEdges detector can generate high scores at the image edges from just a partial pattern match when the template extends partially into the area of *nil* values in the outer mask region beyond the edge of the actual data. To counteract this edge effect, the input to FTC is a copy of the *gfBaseImages->dz* reflectivity image with the *nil* values in the outer region beyond the actual data replaced by a value of 45 (2.5 dBZ). With the scoring functions shown in Figure 40, this

means that negative (disconfirming) scores will be returned from the portions of the template that extend beyond the actual weather data, and will result in reduction of the average interest score that is returned by FTC near the edges of the data.

The modified reflectivity image is supplied along with the mask that was constructed as described above to FTC. Note that the center of rotation for the template is not located at the geometric center of the kernel. Therefore, the kernel must be rotated through a full 360 degrees at each image location in order to evaluate all candidate match orientations. The output interest image will have confirming scores at the edges of the precipitation regions. The regions of confirming scores are then expanded by performing a grayscale dilation operation with a 3x3 kernel.

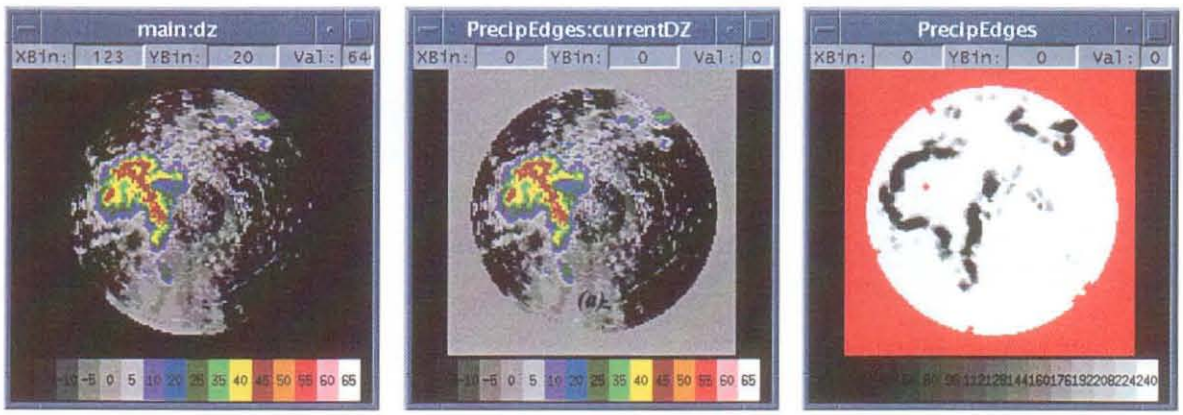
Finally, in order to make this into an interest image that can be combined in a negative fashion with the other interest images, the interest values are flipped about the ambiguity value of 128 by multiplying each interest value by -1 and then adding 255. Figure 41 shows example results of the PrecipEdges detector.



0		0		1			2		3			3	
0		0		1			2		3			3	
0		0		1			2		3			3	
0		0		1			2		3			3	
0		0		1			2		3			3	

Kernel rotation angles: 0, 20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 220, 240, 260, 280, 300, 320, 340

Figure 40. Scoring functions and template for the PrecipEdges detector. Lower X-axis of scoring function is the integer-scaled pixel value. Upper X-axis is reflectivity in dBZ.



(a) *(b)* *(c)*

Figure 41. Example computation of the precipitation edges interest image (PrecipEdges). (a) Reflectivity image. (b) Modified copy of reflectivity image after setting outer edge values to 45 (2.5 dBZ). (c) Final PrecipEdges interest image after FTC and inverting the interest scores to produce the "negative" interest image. Data are from 07/21/99 21:16:23 GMT at Austin, TX

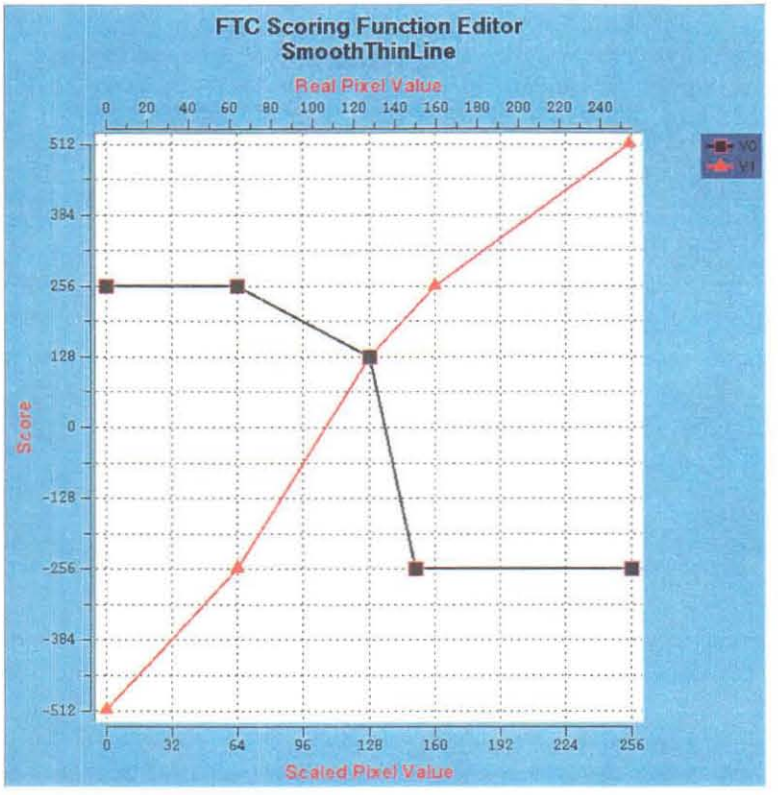
2.4 Interest combination

The various interest images generated by the feature detectors are combined in two stages. First **SKAverageInterestImages** is called to compute a weighted average for each pixel across all interest images except the *gfBaseImages->anticipation* image. The weight assigned each interest pixel during the averaging is determined from the intensity of the pixel. If the interest value falls within the disconfirming interval of [0, 127], then the pixel value is multiplied by the disconfirming weight parameter for that detector. If the pixel falls within the confirming interval of [128, 255], it gets the confirming weight. *Nil* pixels are not included in the average.

For the second stage of interest combination, the pre-combined interest image computed above is assigned confirming and disconfirming weights of 1.0, and **SKFuzzyWeightedAverage** is called to average the pre-combined interest with the anticipation interest image. Anticipation interest values are assigned averaging weights based on the weighting function nodes defined in the detector parameter file (see Table 7 for the default weight settings). The resulting weighted average representing the final combined interest image is stored in the *gfBaseImages->rawInterest* variable.

2.4.1 Thin line smoothing

Thin lines in the combined interest image can be fragmented where gust fronts intersect with out-of-trip weather, are obscured by storm cells, or have signals that are near the minimum detectable signal level. The **smoothThinLine** function uses a "bow-tie" functional template (Figure 42) to smooth thin lines, bridging the gaps between co-linear fragments and suppressing random, unaligned points of high interest values. By placing more kernel elements at the ends than at the center, the bow-tie weights the influence of the end regions over that of the center. Consequently, the bow tie generates high output interest scores for an image point between two co-linear high interest segments, even if that point itself has a low input interest value. Also, because of how the scoring functions are designed, the bow-tie filter suppresses and amplifies co-linear interest values that are below and above the level of ambiguity (*gfParams->interestThreshold, nominally 127).* An example input and output of thin line smoothing is shown in Figure 43.)



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

Kernel rotation angles:

0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170

Figure 42. Scoring functions and template for the "bow-tie" thin line smoother. Lower and upper X axes of scoring functions are the is the pixel interest value.

Figure 43. Combined interest image before (left) and after thin line smoothing (middle). The rightmost image is the corresponding orientation image. Data are from 12/11/00 20:10:53 GMT at Austin, TX

2.5 Extraction

The extraction phase begins after thin line smoothing of the combined interest image is completed. Gust front chain candidates are extracted by applying a threshold to the smoothed, combined interest image, eliminating resulting shapes that are too short, thinning the remaining interest bands to single-pixel wide chains, extending the chains to recover valuable detection length lost during thresholding, establishing correspondence with chains from prior detections, and finally, collecting the disjointed network of chains into reasonable combinations of non-looping chains that represent individual gust fronts. Figure 44 illustrates the function call hierarchy for the extraction phase of MIGFA. The hierarchy is not exhaustive; in particular, low-level functions from the CSKETCH library are omitted

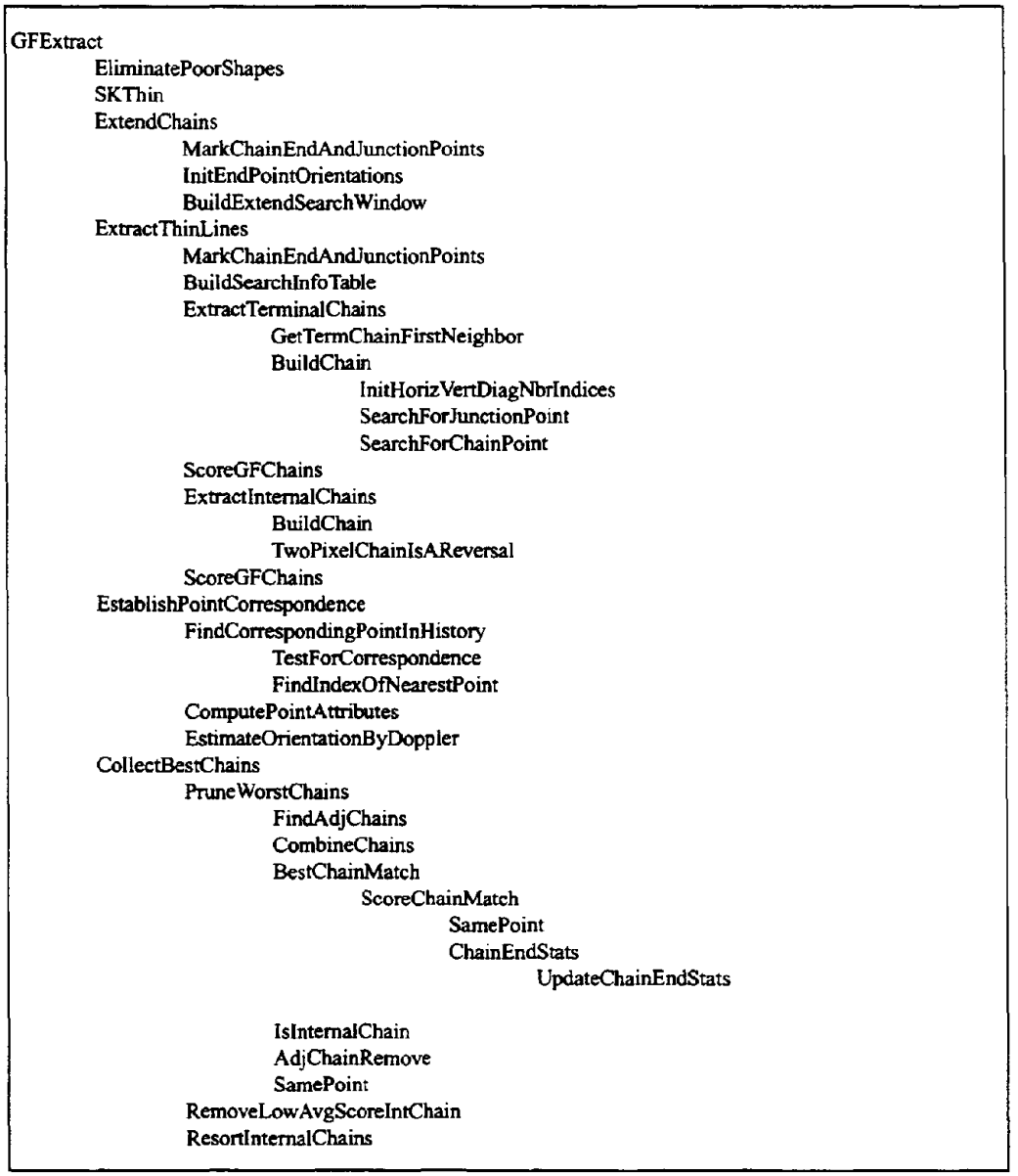


Figure 44. Function call tree for the gust front extraction phase of MIGFA (see text for description).

2.5.1 Eliminating Poor Interest Shapes

Function **EliminatePoorShapes** applies a threshold of *gfParams->interestThreshold* (nominally, 127) to the smoothed combined interest image to create a binary image of candidate gust fronts. The lengths of the resulting elongated shapes are then computed. Contiguous regions of 1's that are shorter than *gfParams->minRawShapeLength* are reset to 0. The result of thresholding and pruning a smoothed, combined interest image (Figure 45a) is shown in Figure 45b.

2.5.2 Thinning

Binary elongated shapes are thinned using an FTC implementation of homotopic thinning [5] implemented by function **SKThin** in the CSKETCH library. Briefly, homotopic thinning is a binary pattern matching technique that iteratively looks for pixel patterns that are characteristic of a shape that is not fully thinned (i.e., one pixel in width), and removes a single later of pixels from the outside. When no more pixels can be removed, the function stops. The result of this operation is shown in Figure 45c.

2.5.3 Chain Extension

The thinned chains of points are then extended along ridges of relatively high interest, using what is essentially a road-following technique in function **ExtendChains**. At each marked endpoint, the 8-connected neighborhood (neighborhood of 8 surrounding pixels) is examined, looking for the pixel with the maximum interest and whose orientation (found in the *gfBaseImages->baseOrient* image) is less than the maximum deviation (defined by *gfParams->extend1Params[2]* - nominally, 41 degrees) from that of the initial end point. If the maximum interest score is greater than or equal to the minimum interest score given by *gfParams->extend1Params[1]* (nominally, 80), then the point is added to the chain, and the process continues until a qualifying point cannot be found or the maximum number of extended points given by *gfParams->extend1Params[0]* (nominally, 10) has been reached. The whole chain extension process is repeated a second time by another call to **ExtendChains** with the *gfParams->extend2Params array*, this time allowing points with lower interest values to be considered (nominally 48), but with a much more restrictive angular deviation (nominally 11 degrees).

With normal chain extension, there are two "nearest" pixels within the 8-connected region that are considered for addition to the chain. The two pixels chosen depend on the orientation of the chain at its current end, and the extension direction given by the *gfBaseImages->baseOrient* image. If aggressive chain extending has been enabled via the *gfConfig->doAggressiveExtend* parameter (this is enabled for the WSP), then an additional third pixel on the "other" side of the initial pair is considered. This allows the chain extension to "wander" a little more freely in search of a higher interest path to follow. The results of the two rounds of chain extension can be seen in panels (d) and (e) of Figure 45.

After the "extend" operation, the image is re-thinned to eliminate any small clusters of pixels that may have been introduced by the extension operations (Figure 45f).

2.5.4 Extracting Thin Line Chains

After chains have been extended outward through high interest areas, the resulting chains may be highly branched and may even contain loops. Each network of chains is described as a graph -- a set of nodes (junction points) that are linked together by edges (chains of points). Chain end points that are not found at junctions are called terminal points. A chain having one or two terminal points is called a terminal chain; otherwise it has two nodes as end points and is called an internal chain. Function **ExtractThinLines** parses the chains into sets of internal and terminal chains that are stored and returned in linked lists of GFChain data objects (Figures 45g and 45h). Function **ScoreGFChains** is called to compute chain score values (total, minimum, average) for every internal and external chain based on the corresponding interest values in the smoothed combined interest image (*gfBaseImages->smoothedInterest*).

2.5.5 Establishing Point Correspondence

Correspondence can be difficult to establish when two or more gust fronts collide; in such cases, a point in the prior event that is nearest to a point in the current extracted chain is not necessarily the correct corresponding point. MIGFA attempts to find the detected gust front point from the prior scan that has a projected location (given propagation speed and direction estimates for that point) that is nearest to a particular point in the current scan. If no such point in the prior event is found, then a point is assumed to correspond to the closest point in the prior event. Once a corresponding point is found, the current point is assigned the index of the corresponding point in the previous GFEvent (obtained from the GFEvent history list *gfRefData->eventHistory*).

If the distance between the two corresponding points is too large, or if the distance moved for the two points is inconsistent with prior history, then the point is unindexed again (the link is broken). Through the index links, a point can be tracked backwards in time. The number of prior events through which a point can be tracked is called its depth. A depth of "0" means that the point is unindexed. Once indexed, each point is assigned various attributes including: coordinates, distance moved, direction moved, depth, Doppler velocity value, interest value, and propagation speed. Whenever correspondence is established for a chain point, the tracking depth is set to one more than the depth assigned to the corresponding prior point.

Function **EstablishPointCorrespondence** is called twice: once with the list of terminal chains, and once with the list of internal chains. Table 18 lists the parameters that are used by function **EstablishPointCorrespondence**. See Appendix A.2 for brief descriptions of how each of the parameters is used.

Table 18. Parameters Used By Function EstablishChainCorrespondence

Parameter Name	Nominal Value	Units
maxEventOneDimDistanceMovedKM	5.0	km
minCorrespondenceDepth	3	unitless
minDistanceMovedKM	3.0	km
maxOrientationChange	21	degrees
maxChangeInDistanceMovedKM	1.5	km
maxPctChangeInDistanceMoved	1.0	unitless
maxEventDistanceMovedKM	5.0	km

2.5.6 Collecting "Best Chains"

Function **CollectBestChains** is called to find, for each disjoint network of terminal and internal chains, the combination of non-looping chains with the highest summed interest score (the highest scoring combination is usually, but not always, the longest combination). An additional constraint is imposed to favor the combining of segments that are moving consistently in terms of direction and propagation speed. Consider the simple example shown in Figure 46. The most interesting combination of chains is discovered by iteratively pruning away terminal chains. The basic operation is to find an internal node having two or more terminal chains (e.g., node E in Figure 46). The terminal chain with the lowest score (chain EG) is removed, leaving chain EF at node E. (If chain DE were found to be moving with a direction and speed that were more consistent with EG than with EF, the EF would have been removed from the search graph instead.) If the resulting pruned graph (Step 2) has an internal node with only one internal chain (e.g., node E), then the internal node is removed and the internal and single remaining terminal chains are combined (Step 3). This process continues until no more internal nodes with two or more terminal chains are found. If at this point a network still contains more than one chain, then a loop exists somewhere in the network (as in BCD in Step 3). A loop is broken by removing the internal chain with the lowest summed interest score (CD). Pruning of terminal chains resumes, resulting in the removal of chain BC in Step 5, the fusing of chains BD and DF in Step 6, and the fusing of chains AB and BF in Step 7. The process is repeated until only one chain remains (chain AF in Step 7). If the resulting combined chain has a score greater than a minimum threshold (*gfParams->gfMinScore*), then that chain is included in the GFEvent for the current scan.

Once the highest scoring combined chain is discovered, its constituent edges are removed from the original graph. The remaining graph structure is examined again (Step 8), looking for another maximally scoring combined chain. Processing continues as long as combined chains with interest scores not less than *gfParams->gfMinScore* (assume a value of 20 in this example) are extracted (edge EG with an interest score of 20 is retained, the result of fusing chains BC and BD is rejected). Figure 45i shows the final above-threshold combined chains. This completes the extraction process.

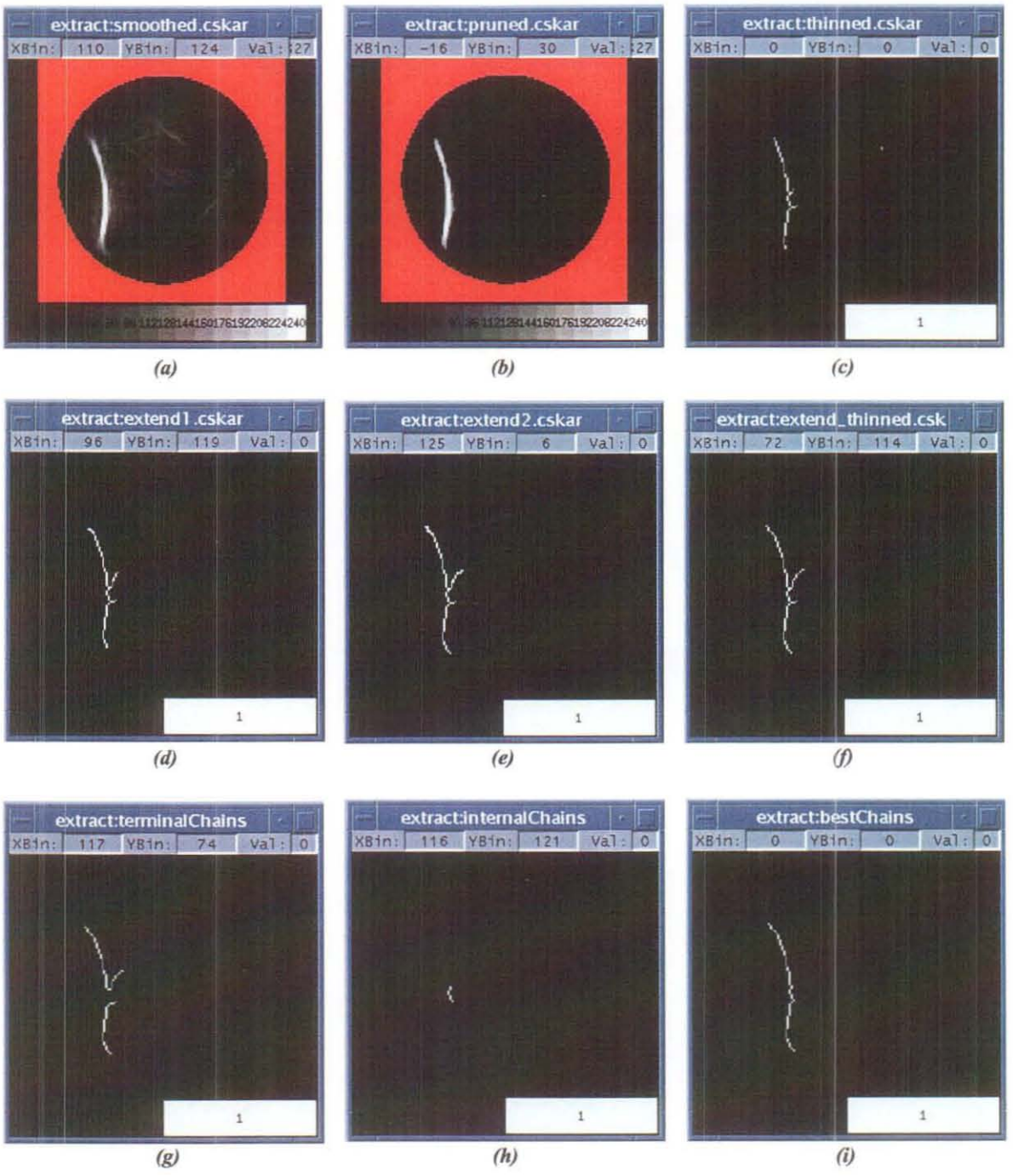


Figure 45. Chain extraction summary. (a) Smoothed combined interest image. (b) "Pruned" image after thresholding and elimination of small interest shapes. (c) Thinned (single-pixel wide) chain image. (d) Chain image after first round of extension. (e) Chain image after second round of extension. (f) Chain image after re-thinning to single-pixel wide chain. (g) Terminal chains. (h) Internal chains. (i) Final extracted chains. Data are from 7/08/98 21:08:17 GMT at Albuquerque, NM

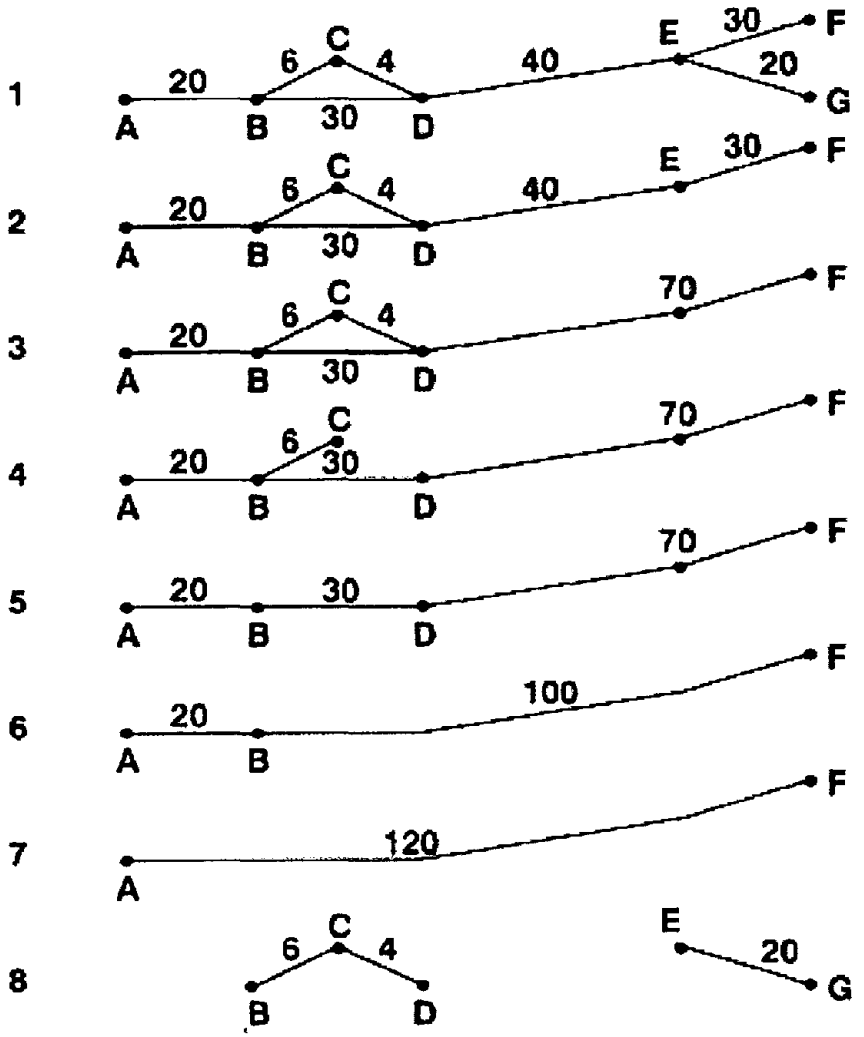


Figure 46. Example extraction of most interesting chains from a simple graph structure. (See text for explanation.)

2.6 Gust Front Analysis

GFAnalyze edits and smooths the gust front chains, subdivides chains that have segments exhibiting different behaviors, computes the various wind estimates, applies heuristics to reject gust fronts exhibiting inconsistent behaviors, and makes a final decision as to whether or not to report the gust front in the final product output. The function call hierarchy for the gust front analysis stage (**GFAnalyze**) is shown in Figure 47. The hierarchy is not exhaustive; in particular, low-level functions from the CSKETCH library are omitted, as well as the function call hierarchy for the wind analysis (described later).

Before chain analysis begins, function **GFBuildPointArray** is called in order build a GFPointArray object to store all of the chain points from the list of gust front chains in a single data structure. This data representation is more efficient for many of the subsequent chain processing operations. The point array is stored in *gfEvent->gfPointArray* for later reference.

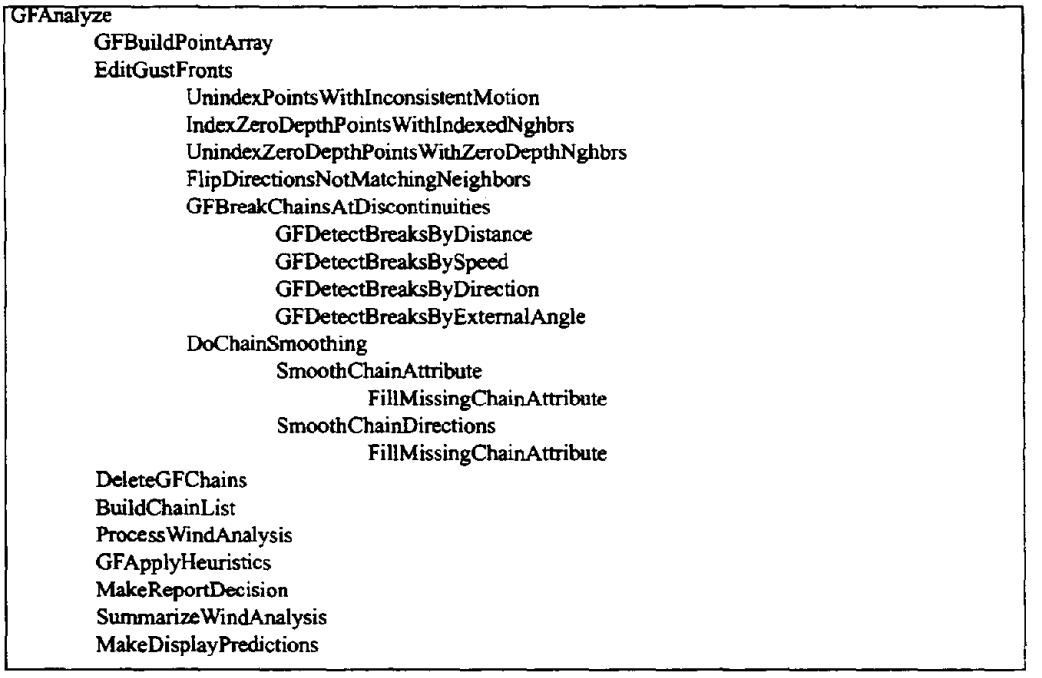


Figure 47. Function call tree for the gust front analysis phase of MIGFA (see text for description).

2.6.1 Editing Gust Front Chains

Table 19 lists the default settings for parameters that are used in the chain editing process. Function **EditGustFronts** uses several techniques to make point behavior more consistent over a local neighborhood of chain points, and then breaks up single chains that have portions behaving in different ways. The following functions are utilized to perform consistency editing of chain points:

1. Function **UnindexPointsWithInconsistentMotion** is called to unindex points that exhibit motion characteristics (specifically, distance moved) that are not in line with the average motion for the chain. If the motion of the point is more than *gfParams->maxStdDevsMoved* than the average motion of all of the chain points, then the point is unindexed. Unindexing consists of setting the tracking depth to 0 and the prior index to *nil* for the point in question.
2. Function **IndexZeroDepthPointsWithIndexedNghbrs** loops over chain points, looking for points with tracking depth = 0. If a depth 0 point is bracketed by neighbors from the same chain that have non-zero depth, then one of the non-zero depth points is arbitrarily chosen and its attributes (with the exception of the location and interest score) are copied to the depth 0 point.
3. Function **UnindexIndexedPointsWithZeroDepthNghbrs** is called to loop over chain points, unindexing points that are bracketed by zero depth points.
4. Function **FlipDirectionsNotMatchingNeighbors** flips by 180 degrees the current direction of gust front points whose direction of motion is likely 180 degrees out of phase with neighboring pixels. This is necessary because in some cases (especially for newly-detected or slow-moving fronts), the direction of motion for some of the points may be 180 degrees ambiguous.

Following consistency editing, **EditGustFronts** attempts to break up chains that exhibit discontinuities in various attributes. For example, differences in gust front propagation speed or direction can indicate that two different fronts have been inappropriately merged into a single one and should be separated. This operation is accomplished in function **GFBreakChainsAtDiscontinuities** by means of a consensus approach using four different criteria for detecting discontinuities. A collection of 1-D feature detectors is used, each detector looking for some indication of discontinuity and generating an interest line (a 1-D interest image). The interest lines are averaged, and where the average interest values are large enough, the chain is subdivided. Four functions are called to apply the chain breaking criteria:

1. Function **GFDetectBreaksByDistance** identifies locations where gust front chains should likely be broken due to discrepancies in the distance moved by neighboring gust front points in the prior scan. A 1-D array of distances moved by corresponding points in the previous scan is passed to the CSKETCH library least squares derivative filter function **SKLsqDerivFilter** with a filter window size of *gfParams->breaksByWSize*. The resulting derivative array is then converted to a 1-D interest image by linearly mapping the derivative values within the range given by *gfParams->scaleBoundsBreaksByDist* (a pair of numbers) to the interest interval of [0, 255]. The interest image is assigned confirming/disconfirming averaging weights given by the *gfParams->weightsBreaksByDistance* parameter.
2. Function **GFDetectBreaksBySpeed** determines locations where gust front chains should likely be broken due to discrepancies in the speed of motion of neighboring gust fronts points in the prior scan. A 1-D array of propagation speeds associated with corresponding points in the previous scan is passed to **SKLsqDerivFilter** with a filter window size of *gfParams->breaksByWSize*. The resulting derivative array is then converted to a 1-D interest image by linearly mapping the derivative values within the range given by *gfParams->scaleBoundsBreaksBySpeed* (a pair of numbers) to the interest interval of [0, 255]. The interest image is assigned confirming/ disconfirming averaging weights given by the *gfParams->weightsBreaksBySpeed* parameter.

3. Function **GFDetectBreaksByDirection** determines locations where gust front chains should likely be broken due to discrepancies in the direction of motion of neighboring gust front points in the prior scan. Two passes are made over the chain. The first looks at the direction of motion of 2 points nominally 6 nodes apart in the chain. If the points meet certain tracking depth criteria (1 or greater) and have sufficiently different directions of motion (i.e., greater than *gfParams->breaksByDirCriterionAngle*), a score is computed by multiplying the direction difference by *gfParams->breaksByDirFactor*. Next, the index is found of the points within the comparison window where the greatest jump in orientation between two consecutive pixels occurred. The computed score is assigned to that index location in the interest image.

   In the second pass, pairs of non-zero tracking depth point segments that are separated by segments of zero depth points are examined. If the direction of motion of the last point of one segment differs from the direction of motion of the first point of the next segment by more than 90 degrees, and if this direction change is spatially persistent over the next 10 points, a high score is generated by multiplying the large angle difference by *gfParams->breaksByDirFactor* to indicate a potential chain break at a point between the 2 segments (as above, at the index of the point where the greatest two consecutive pixel jump in point orientation occurred). The resulting interest image is assigned confirming/disconfirming averaging weights given by the *gfParams->weightsBreaksByDirection* parameter.

4. Function **GFDetectBreaksByExternalAngle** identifies locations where gust front chains should likely be broken due to sharp "corners" seen in traversing the chain (e.g., rapid changes in the direction of orientation of the chain). At every point in the chain, the external angle formed by the point and its chain neighbors five pixels away on either side is computed. An interest score indicating the likelihood that the chain should be broken at the centerpoint is computed by multiplying the external angle by *gfParams->breaksByExtAngleFactor*. The score is halved if the two "endpoints" of the window have motion attributes that indicate they will eventually intersect (e.g., the front is converging on itself). The resulting interest image is assigned confirming/ disconfirming averaging weights given by the *gfParams->weightsBreaksByExtAngle* parameter.

Finally, after chain editing, function **DoChainSmoothing** is called to smooth the various chain point attributes using spatial window averages over the chain. Function **SmoothChainAttribute** performs the averaging of the distance-moved and speed attributes using parameter *gfParams->smoothAttributeWindowSize* to govern the length of the averaging window. Function **SmoothChainDirections** performs the averaging of the chain point directions using the *gfParams->smoothDirectionWindowSize* parameter. The averagings are performed in two passes. The first pass performs averaging and skips missing (*nil*) attributes; the second pass performs averaging again, but subsequently fills the missing attributes (via function **FillMissingChainAttributes**) with an average of the nearest two neighboring average values.

The initial chain list (whose points where copied to the *gfEvent->gfPointArray* array) is obsolete following chain editing. A new chain list is rebuilt from *gfEvent->gfPointArray* by calling **DeleteGFChains** to delete the obsolete chain list, and then calling **BuildChainList** to construct the updated chain list.

Before **BuildChainList** returns, and after rebuilding the chain list, function **ScoreGFChains** is called to recompute score values (*chain->score, chain->minScore, chain->avgScore*) for each gust front candidate chain based on the corresponding interest values in the *gfBaseImages->smoothedInterest* interest image.

Table 19. Parameters Used by Function EditGustFronts

Parameter Name	Nominal Value	Units
maxStdDevsMoved	1.5	km
breaksByWSize	6	pixels
scaleBoundsBreaksByDist	(0, 0.43)	pixels
weightsBreaksByDistance	(1.0, 1.0)	unitless
scaleBoundsBreaksBySpeed	(0, 0.002)	pixels/sec
weightsBreaksBySpeed	(1.0, 1.0)	unitless
breaksByDirCriterionAngle	45	degrees
breaksByDirFactor	4.266667	unitless
weightsBreaksByDirection	(4.0, 1.0)	unitless
breaksByExtAngleFactor	3.94	unitless
weightsBreaksByExtAngle	(2.0, 2.0)	unitless
smoothAttributeWindowSize	11	pixels
smoothDirectionWindowSize	5	pixels

2.6.2 Wind Analysis

MIGFA uses a consensus approach to estimating wind shift and wind shear values for detected gust fronts. The consensus approach computes a weighted average of several estimates (each estimate based on a different technique) for each windfield product. Along with the combined estimate for each product, a weight or confidence value for the product estimate is also computed.

Figure 48 is the function call hierarchy for the wind analysis phase. Function **ProcessWindAnalysis** is the processing driver for running the various wind estimators on each GFChain in the chain list. For each GFChain, the wind estimation segment is determined by calling function **GFChainCreateWindSegment**. The segment over which the analysis will be performed is chosen based on the distance of the front from the airport reference point (ARP). If the front is more than *gfParams->arpControlTimeSec* seconds (nominally 20 minutes) away from an airport control radius, *gfParams->arpControlRadiusKM* (nominally, 15 km), then MIGFA simply chooses a segment of length 2 * *gfParams->windEstSegmentHalfLength* pixels centered about the front's midpoint as the focal point for the wind analysis. Otherwise, if the nearest point of a gust front is within *gfParams->arpControlTimeSec* seconds of the airport control radius, the wind estimation segment is shifted to be centered on the point on the front that is closest to the ARP. Once the wind analysis segment has been identified, function **GFChainComputeWindSegmentStats** is called to compute the average speed, direction, and tracking depth over the wind estimation segment of the chain

(*GFChain->windSegAvgSpeed, GFChain->windSegAvgDirection,* and *GFChain->windSegAvgDepth,* respectively).

Figure 48. Function call tree for the MIGFA wind analysis stage (see text for description).

Once the appropriate gust front segment has been identified, MIGFA conducts multiple windfield analyses within three defined analysis regions as shown in Figure 49.

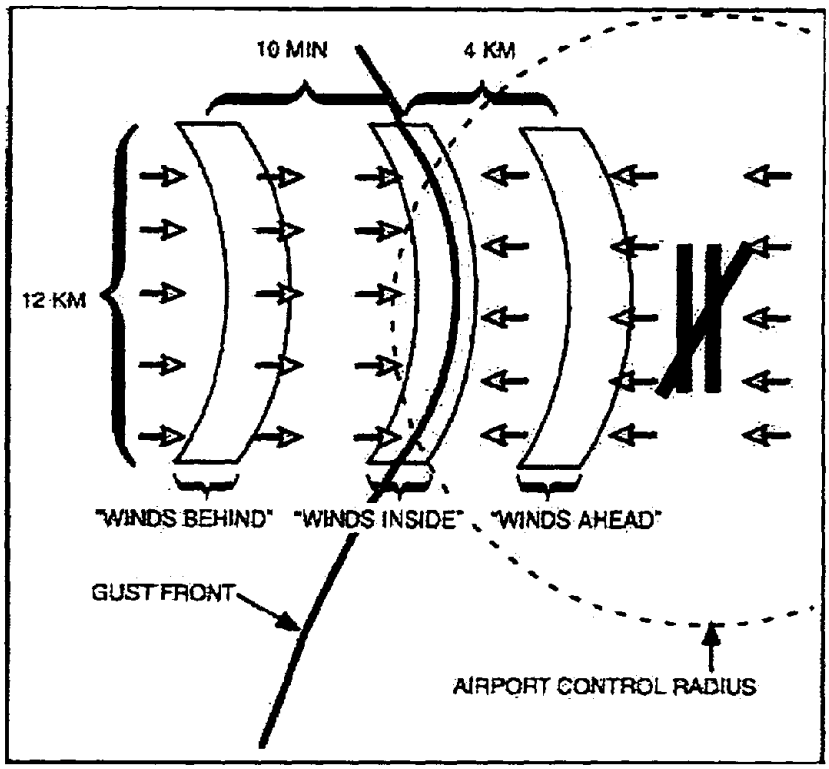


Figure 49. Wind analysis regions for MIGFA gust front wind shift and wind shear estimation. For WSP, the "winds behind" region is defined to be 0 minutes behind the front, so the "winds behind" region for WSP MIGFA is the same as the "winds inside" region.

Wind estimates derived from the "winds ahead" analysis region are used for computing the wind shear hazard (delta-V) across the front. Some wind estimators developed for other MIGFA variants (e.g. TDWR, NEXRAD) analyze the radar Doppler measurements in a region just ahead of the front as defined by the *gfParams->windSamplingDistAheadKM* parameter (nominally, 4 km). However, because the WSP often lacks reliable Doppler velocity measurements in the clear air ahead of many gust fronts, the WSP MIGFA does not utilize any Doppler-based wind estimation methods for that region, and the "wind ahead" region is not strictly defined. Rather, the WSP MIGFA relies on current and historical distance-weighted airport anemometer measurements for determination of the winds ahead (described later).

The "winds inside" analysis region is so named because of the association with being "inside" the thin line echo region of many gust fronts. The winds in the thin line echo region are associated with the "nose" of the outflow immediately behind the front. These can be stronger and more turbulent than the steadier outflow winds further behind the front, and are analyzed primarily for purposes of computing the localized wind shear (delta-V) across the frontal boundary. These winds are also used as part of the consensus for the wind shift behind the front.

For estimation of the "winds behind", it is desirable to sample the winds at a distance behind the front that is presumably representative of the persistent outflow winds. Airport runway planning needs have dictated that a wind shift needs to persist for at least 10 minutes to warrant a runway change consideration. The radar velocity data points analyzed for estimation of the "winds behind" are chosen based on the

propagation speed of each point in the wind estimation segment and the value of the *gfParams->windSamplingTimeBehindSec* parameter. Because many WSP gust front detections are based on thin line echo signatures that lack good quality velocity measurements outside of the thin line echo region, the WSP MIGFA is unable to reliably analyze the ideal region 10 minutes behind the front. Instead, Doppler values inside the frontal boundary itself are analyzed by setting *gfParams->windSamplingTimeBehindSec* to 0 for WSP. Note that for WSP this makes the "winds behind" analysis region essentially the same as the "winds inside" region (described above).

A list of all available wind estimation methods is obtained from the *windEstimationMethods* parameter in the parameter file referenced by the GF_EST_PARM_FILE environment variable (typically, this file is named "estmtrParamsWSP". See Appendix A.4 for an example of this file). Note that the specification of a particular estimator in the parameter file does not necessarily mean that it will be run for all three of the analysis regions. Function **GFAllocObjects** (run at algorithm initialization as described in Section 2.1) contains the logic that loads the appropriate wind estimator methods onto one of the three linked lists: *windAheadEstimators, windInsideEstimators,* and *windBehindEstimators*. Table 20 lists the wind estimators that are loaded by **GFAllocObjects** onto each of the three lists (provided they are also specified GF_EST_PARM_FILE).

Functions **GFRunWindBehindEstmtrs, GFRunWindInsideEstmtrs**, and **GFRunWindAheadEstmtrs** are called by **ProcessWindAnalysis** to run the selected estimators in each of the three analysis regions. Each estimator returns a GFWind data object containing the resulting wind estimate and associated weight value. The GFWind objects from each estimator run for the three analysis regions are packed into lists (*windBehindData, windInsideData,* and, *windAheadData*). The lists of wind estimates are combined to form consensus estimates of each wind quantity for each analysis region by calling **SKCombineWindEstimates**. The resulting consensus wind estimates for each region are stored in the *windBehind, windInside*, and *windAhead* variables of the GFChain being analyzed.

Consensus estimation of the wind shear hazard (delta-V) over the wind estimation segment is performed by calling **GFRunDeltaVEstmtrs** to generate a list of delta-V estimates, and then calling **SKCombineWindEstimates** to form the consensus. Similarly, function **GFRunMaxDeltaVEstmtrs** is called to compute a list of maximum delta-V estimates over the entire front. The list of maximum delta-V estimates is then combined using **SKCombineWindEstimates** to form the consensus maximum delta-V. The consensus delta-V and maximum delta-V estimates are stored in the *deltaV* and *maxDeltaV* variables of the GFChain being analyzed.

For the last stage of wind analysis, quality and consistency checks are conducted on the *windBehind* and *maxDeltaV* consensus quantities. First, function **GFFixCalmWindBehindDirection** is called to enforce a requirement that a weak *windBehind* estimate for a GFChain generally agrees in direction with the estimate generated by the "ByNormal" estimator in the "winds behind" region (described later). If the magnitude of the consensus *windBehind* estimate is less than *gfParams->calmWindSpeed* (nominally 1.5 m/s), and if the difference between the direction of the *windBehind* and the *"behindByNormal"* wind estimates is greater than *gfParams->calmWindAngleDiff* (nominally, 60 degrees), then the consensus *windBehind* estimate is set to the *behindByNormal* estimate. The second test requires that the *deltaV* not exceed the *maxDeltaV*. If it does, then the *maxDeltaV* is reset to *deltaV*.

Table 20. Wind Estimators Used for Wind Analysis Regions

Estimator List Name	Estimators
windAheadEstimators	ByAnemometer, ByHistory
windInsideEstimators	ByNormal, ByAnemometer, ByHistory
windBehindEstimators	ByNormal, ByAnemometer, ByHistory
deltaVEstimators	ByVecDiff, ByShear, ByHistory
maxDeltaVEstimators	BySegment, ByHistory

2.6.2.1 Descriptions of wind estimators

The following sections describe the various wind estimators that are employed to form the consensus estimates of winds in each analysis region as well as the delta-V estimates.

2.6.2.1.1 "ByNormal" wind estimator (perpendicular wind model)

The "ByNormal" estimator is used for the "winds behind" and "winds inside" estimates. It assumes that winds behind the gust front are perpendicular to the orientation of the gust front. Radial Doppler velocity values v_r within the analysis region are converted to horizontal wind speeds | V | using the relationship:

$$| V | = v_r / \cos \theta$$

where θ is the viewing angle difference between the radar azimuth and the angle perpendicular to the orientation of the gust front. This calculation is unstable for large θ which occurs when the gust front is nearly aligned with a radar radial, so θ is constrained to a maximum of 60 degrees.

Function **GetNormalPixelWindSpeedEst** is called repeatedly to obtain a viewing angle adjusted wind velocity (speed and direction) measurement and associated confidence weight at the requested time-distance behind each gust front chain point. Since very slow moving gust front chains are more often associated with false alarms than operationally significant gust fronts, the original Doppler velocity measurement is not adjusted by the cosine of the viewing angle if the propagation speed of the chain point is less than 1.5 m/s. As a sanity check, the returned wind estimate is limited to a maximum value of 40 m/s. Under the perpendicular wind model, an initial value for the retrieved wind direction is simply taken to be the propagation direction of the chain point.

GetNormalPixelWindSpeedEst computes an associated confidence weight for the wind velocity by inputting the point's corresponding propagation speed to a rising-S weight function (**SKRisingS**) with nodes defined by *gfParams->fuzzyNormalEstSpeed*. This results in a weight value between 0 and 1. As an additional sanity check, if the point is slow moving (propagation speed less than 1.5 m/s), and if there is a conflict between the point's propagation direction and the sign of the retrieved Doppler measurement, then a sharp reduction in confidence is expressed by reducing the weight to 10% of its initial value. In addition, the initial wind direction estimate (based on the point's propagation direction) is flipped by 180

degrees to reconcile the direction conflict. A final weight adjustment is made by multiplying the weight value by one-half times the cosine of the viewing angle (θ).

At each point in the analysis region, the initial weight returned by **GetNormalPixelWindSpeedEst** is further adjusted for tracking depth by multiplying the weight value by a scalar returned by indexing the *gfParams->depthWeightTable* by the tracking depth, up to a maximum depth value of *gfParams->maxDepthWeightTableIndex*.

A weighted average of the wind speed and direction values is then computed (*avgSpeed, avgDir*). The individual pre-assigned weights of the wind values are also averaged (*avgWeight)*. If no Doppler measurements were available in the analysis region, then *avgSpeed* and *avgDir* are assigned the propagation speed (*propSpeed*) and direction *(propDirection*) of the wind estimation segment midpoint, and *avgWeight* is assigned a value of *gfParams->windLowWeight* (0.1). An initial weight factor (*weightFactor*) that will be used to adjust *avgWeight* is set to the value of *gfParams->maxWeightBehindByNormal* or *gfParams->maxWeightInsideByNormal*, as the case may be.

A number of limits and adjustments are then imposed on *avgSpeed* and *avgWeight*:

1. *avgSpeed* is limited to a minimum of *gfParams->minNormalSpeed* (nominally 2.57 m/s).
2. If *avgSpeed* is less than *propSpeed,* then *avgSpeed* is set to *propSpeed* and *weightFactor* is halved.
3. If *avgSpeed* exceeds a maximum wind speed limit, V_{max}, given by:

   V_{max} = *gfParams->windBehindLimitFactor* * *propSpeed* + *gfParams->windBehindLimitOffset,*

   then *avgSpeed* is set to V_{max} and *weightFactor* is halved.
4. The standard error of the individual cosine-adjusted wind speed values is input to a ramp plateau weight function (**SKRampPlateau**) whose nodes are defined by *gfParams->fuzzyBehindByNormalError* or *gfParams->fuzzyInsideByNormalError* as the case may be. The resulting value (between 0 and 1) is used to scale *weightFactor*. If there were an insufficient number of wind speed values to compute the standard error, then *weightFactor* is scaled by multiplication by *gfParams->windLowWeight*. *AvgWeight* is then multiplied by the adjusted *weightFactor*.

The final values for *avgSpeed, avgDirection,* and *avgWeight* are packed into a GFWind data object and returned as the result of the "ByNormal" estimator. Table 21 lists the default settings for parameters referenced by the "ByNormal" wind estimator.

Table 21. Parameters Used by the "ByNormal" Wind Estimator

Parameter Name	Nominal Value	Units
fuzzyNormalEstSpeed	(0.0, 1.0, 2.0)	m/s
depthWeightTable	(0.01, 0.2, 0.5, 1.0)	depth (integer)
windLowWeight	0.1	unitless
maxDepthWeightTableIndex	3	unitless
maxWeightBehindByNormal	1.0	unitless
maxWeightInsideByNormal	1.0	unitless
minNormalSpeed	2.57	m/s
windBehindLimitFactor	1.4	unitless
windBehindLimitOffset	7.5	m/s

2.6.2.1.2 "ByAnemometer" wind estimator

For a gust front that has just crossed the airport, an airport anemometer (such as the LLWAS centerfield anemometer or ASOS) provides the most direct measurement of the near-surface winds behind the gust front. Likewise, for a gust front that is approaching and is in close proximity to the airport, the anemometer provides a good measure of the ambient winds ahead of the front and can be used in estimating the wind shear hazard. However, representativeness of this measurement decreases with distance from the location where the winds are to be estimated. Therefore, this estimate is given a large weight when the wind sensor location (assumed to be the airport reference point) is close to the given wind analysis region, and progressively lower weights with increasing distance. This estimator is used in all three of the wind analysis regions ("ahead", "inside", and "behind"). Table 22 lists the default settings for parameters referenced by the "ByAnemometer" wind estimator.

Using the "ByAnemometer" method to estimate winds ahead of the front

If the "ByAnemometer" method is being used to estimate the winds in the region ahead of the front, then function **EstimateAheadByCenter** is called. This function estimates the winds ahead of a front given its location with respect to a "center" location (in this case, the ARP). The function is supplied the current gust front chain, the offset location of the ARP, and a maximum weight value given by *gfParams->maxWeightAheadByAnemometer*.

Given an input gust front, it is first determined whether or not the gust front is approaching the supplied center location. If it is, and if the front is greater than *gfParams->gfImpactRadius* from the center location, then the current anemometer-based airport wind measurement (*gfEvent->airportWind*) is used as

an estimate of the winds ahead of the front. The distance requirement is needed in order to obtain a wind measurement that is presumably unaffected by the approaching front.

If the front is not approaching, or is too close to the center location, then an attempt to retrieve a historical anemometer-based wind estimate (*priorAirportWind*) is made by calling function **GetHistoricalAirportWind**. Function **GetHistoricalAirportWind** searches back the through the event history (*gfRefData->eventHistory*) until it locates a time when the midpoint of the wind estimation segment for the front falls outside of *gfParams->gfImpactRadius*. It then sets *priorAirportWind* to the *airportWind* from the retrieved historical GFEvent.

If the tracking history of a front is exhausted before being able to obtain a suitable historical GFEvent, then function **GetAirportWindViaAnemometer** is called as an attempt to find an estimate of the winds ahead of a gust front using historical anemometer data contained in *gfRefData->anemometerHistory*. Specifically, the oldest available historical gust front point corresponding to the wind estimation segment midpoint of the front is supplied to this routine, along with the time of the GFEvent containing the historical point. Next, the time is computed at which the point would have been approaching the airport but was outside the *gfParams->gfImpactRadius*, via function **EstimateImpactDeltaTime**. The anemometer history is then searched for anemometer data from that time to form an estimate of the winds ahead of the front. If no such data can be found, then a wind speed of zero is returned with a weight of *gfParams->windLowWeight* (0.1).

GetHistoricalAirportWind then computes a weight factor that is a function of the time difference between the time of the retrieved *priorAirportWind* and the current time. Specifically, the weight factor is obtained by calling **SKFallingS** with the computed time difference (in minutes) and a set of falling-S weight function nodes having values of 30, 60, and 120 minutes. This means that a *priorAirportWind* that is less than 30 minutes old will have a full weight factor of 1.0, one that is 60 minutes old will have a weight factor of 0.5, and one that is 2 hours or more old will have a weight factor of 0. The original weight value for the *priorAirportWind* is adjusted (multiplied) by this weight factor before being returned as the historical wind estimate to **EstimateAheadByCenter**.

EstimateAheadByCenter makes a final adjustment to the weight associated with the obtained wind estimate, based on the distance of the front and the tracking depth of the front. If the front is too close or too far away, or if the tracking depth is small, the weight will be reduced. An initial *weightFactor* is first initialized to *gfParams->maxWeightAheadByAnemometer*. The *weightFactor* is then adjusted by multiplying it by the weight returned by calling **SKSPlateau** with the distance of the front from the ARP and the "S-Plateau" weight function nodes given by *gfParams->fuzzyAheadByCenterDist*. With the default nodes, the S-Plateau will return a maximum weight factor of 1.0 for fronts that are between 6 and 10 km of the ARP. The *weightFactor* is further adjusted by multiplying it by the weight value obtained by indexing *gfParams->depthWeightTable* by the depth of the wind segment midpoint (up to a maximum depth of *gfParams->maxDepthWeightTableIndex*). Finally, the weight of the wind estimate to be returned is multiplied by the *weightFactor*, with a limit of *gfParams->windLowWeight*.

Using the "ByAnemometer" method to estimate winds inside or behind the front

If the "ByAnemometer" method is being used to estimate the winds in the region "inside" or "behind" the front, then function **EstimateByCenter** is called. This function takes the current airport wind measurement, and adjusts its weight based on its appropriateness given the tracking depth of the front and the location of the front with respect to a "center" location (in this case, the ARP). The function is supplied the current gust front chain, the offset location of the ARP, the current airport wind (*gfEvent->airportWind*), and a maximum weight value given by *gfParams->maxWeightInsideByAnemometer* or *gfParams->maxWeightBehindByAnemometer* as the case may be.

Given the current gust front chain, it is first determined whether or not the gust front is approaching the supplied center location (i.e., the ARP). If it is approaching, a weight of zero (i.e. an estimate which will be entirely ignored) is returned, as this method of wind estimation is only applicable in cases where the front is moving away from the radar. If the front is not approaching, the winds via anemometer data will reflect conditions behind the front, so a weight factor is computed based on the distance between the wind segment midpoint and the "center" location; the shorter the distance, the greater the weight.

The distance weight factor is computed by calling **SKSPlateau** with the distance of the wind segment midpoint from the ARP and the "S-Plateau" weight function nodes given by *gfParams->fuzzyEstimateByCenterDist*. With the default nodes, the S-Plateau will return a maximum weight factor of 1.0 for fronts that are between 3 and 6 km of the ARP.

A tracking depth-based weight factor is separately obtained by indexing *gfParams->depthWeightTable* by the depth of the wind segment midpoint (up to a maximum depth of *gfParams->maxDepthWeightTableIndex*).

The returned weight value is the product of the distance and tracking depth-based weight factors multiplied by *gfParams->maxWeightInsideByAnemometer* or *gfParams->maxWeightBehindByAnemometer* as the case may be.

Table 22. Parameters Used by the "ByAnemometer" Wind Estimator

Parameter Name	Nominal Value	Units
maxWeightAheadByAnemometer	1.0	unitless
maxWeightInsideByAnemometer	1.0	unitless
maxWeightBehindByAnemometer	1.0	unitless
gfImpactRadius	6.0	km
windLowWeight	0.1	unitless
fuzzyAheadByCenterDist	(1, 3, 6, 10, 15, 40)	km
fuzzyEstimateByCenterDist	(1, 2, 3, 6, 10, 20)	km
depthWeightTable	(0.01, 0.2, 0.5, 1.0)	depth (integer)
maxDepthWeightTableIndex	3	unitless

2.6.2.1.3 "ByHistory" wind and delta-V estimator (persistence model)

The consensus wind estimate from the previous processing interval can be used both as a default value (assuming persistence) and as a dampening mechanism when averaged with other wind estimates. The weight assigned to the persistence estimate is computed as a fraction of the prior consensus weight, depending on how close the attributes (proximity and propagation direction) of the previous gust front chain match the current one. The "ByHistory" estimate is a component of the consensus wind estimate for for all three of the wind analysis regions ("ahead", "inside", and "behind"). It is also used as an estimator for the *deltaV* and *maxDeltaV*. Table 23 lists the default settings for parameters referenced by the "ByHistory" wind estimator.

Function **GetHistoricalChainWind** is called to retrieve the historical wind estimate. For a given gust front chain, the wind estimation segment midpoint for the front is examined. Due to the point correspondence routines executed previously, this point may have an associated point from a prior scan. If it does, then all of the gust front chains from the prior scan are examined by function **FindBestCorrespChain** to determine which one most likely corresponds to the current gust front. The desired type of wind estimate ("ahead", "behind", etc.) of this prior-time gust front chain is taken to be a "historical" estimate of the same type of wind for the current front. If the current front's wind segment midpoint had no prior point (e.g. has not been tracked prior to the current scan), a zero wind is returned (a GFWind object which has all fields set to zero).

Once a historical estimate has been found, a weight factor is computed by adjusting the weight of the historical estimate based on the distance between the wind segment midpoints of the current and historical gust front chains, and the difference in the direction of motion. The distance-based weight factor (*weightFactor1*) is computed by calling **SKFallingRamp** with the distance (in km) and the falling ramp function nodes given by *gfParams->fuzzyHistChainDist*. The value for *weightFactor1* returned by **SKFallingRamp** is limited to a minimum value of 0.1. The second propagation velocity-based weight factor *(weightFactor2)* is computed by calling **SKFallingRamp** with the angle difference between the propagation directions of the current and historical previous wind segments, together with falling ramp function nodes given by *gfParams->fuzzyHistChainAngle*. The value for *weightFactor2* is limited to a minimum value of 0.001. The combined weight factor returned by **GetHistoricalChainWind** for the retrieved historical wind estimate is taken to be the product of the original consensus weight value for the historical wind estimate and the two weight factors, *weightFactor1* and *weightFactor2*.

The final weight value for the wind estimate returned by the "ByHistory" estimator is then computed by multiplying the weight factor from **GetHistoricalChainWind** by the maximum weight parameter for the particular region (or quantity) being analyzed (i.e., *gfParams->maxWeightAheadByHistory*, *gfParams->maxWeightInsideByHistory*, *gfParams->maxWeightBehindByHistory*, *gfParams->maxWeightDeltaVByHistory*, or *gfParams->maxWeightMaxDeltaVByHistory*).

Table 23. Parameters Used by the "ByHistory" Wind and Delta-V Estimator

Parameter Name	Nominal Value	Units
maxWeightAheadByHistory	1.0	unitless
maxWeightInsideByHistory	1.0	unitless
maxWeightBehindByHistory	1.0	unitless
maxWeightDeltaVByHistory	1.0	unitless
maxWeightMaxDeltaVByHistory	1.0	unitless
fuzzyHistChainDist	(10, 20)	km
fuzzyHistChainAngle	(90, 135)	degrees

2.6.2.2 "ByShear" delta-V estimator

The "ByShear" delta-V estimator retrieves values from the dilated shear map (*gfBaseImages->convD77*) that correspond to pixels in the wind estimation segment of the gust front chain, and returns the delta-V value corresponding to a parameter percentile of the delta-V distribution in the wind estimation segment.

As each delta-V value is retrieved from the shear map, a weight is computed as the cosine of the angle difference between the radar azimuth of the point and the propagation direction of the point. This results in low weights being assigned for delta-V values from points that are moving perpendicular to the radar line of sight. The distance from the radar of each retrieved delta-V sample is also computed. After all of the delta-V values corresponding to the wind estimation segment have been retrieved from the shear map, the standard deviation (*stdDev*) is computed.

The delta-V values are then sorted in increasing order, and the sample corresponding to the *gfParams->percentileDeltaVByShear* percentile in the distribution is selected as the delta-V estimate.

If the delta-V value is greater than 5.0 m/s, then a weight of *gfParams->maxWeightDeltaVByShear* is assigned.

Otherwise, the weight that was computed for the retrieved delta-V value based on the direction of movement with respect to the line of site is used as an initial weight value that is adjusted (via multiplication) by two weight factors: one that is based on the standard error of the delta-V values, and one that is based on the distance of the values from the radar. The standard error weight factor (*factor1*) is computed by applying a falling-S weight function to the standard error (*stdDev / sqrt(N)*). The falling-S weight function nodes are defined by parameter *gfParams->fuzzyDeltaVByShearError*. The distance weight factor (*factor2*) is computed by calling **SKSPlateau** to apply an S-Plateau shaped weight function with nodes defined by parameter *gfParams->fuzzyDeltaVByShearDist*.

Table 24 lists the default settings for parameters referenced by the "ByHistory" wind estimator.

Table 24. Parameters Used by the "ByShear" Delta-V Estimator

Parameter Name	Nominal Value	Units
percentileDeltaVByShear	0.75	unitless
maxWeightDeltaVByShear	0.1	unitless
fuzzyDeltaVByShearError	(1, 2, 10)	m/s
fuzzyDeltaVByShearDist	(0, 1, 3, 10, 20, 60)	km

2.6.2.2.1 "ByVecDiff" delta-V estimator (wind vector difference)

With this estimator, the delta-V across the front is computed using the previously computed consensus estimates obtained for the "behind" and "inside" regions of the front. Figure 50 illustrates the vector geometry. The delta-V is computed as the magnitude difference between the winds immediately behind the front (the "inside" winds), V_b, and the vector component of the wind ahead of the front, V_a, that is parallel to V_b (labelled V_a'). Note that depending on the actual amount of wind shear, this estimator is less likely to accurately capture the 1-km interval change in velocity across the front, since it measures only the end-to-end velocity difference across a relatively large distance, and is insensitive to the velocity gradient of the front across the desired 1-km distance.

After calling **GFComputeVectorDeltaV** to compute the vector difference delta-V estimate, an initial weight for the delta-V estimate is computed as the minimum of the *windAhead* and *windInside* weights. The final weight value is the product of this initial weight and the maximum weight given by *gfParams->maxWeightDeltaVByVectorDiff* (nominally, 1.0).

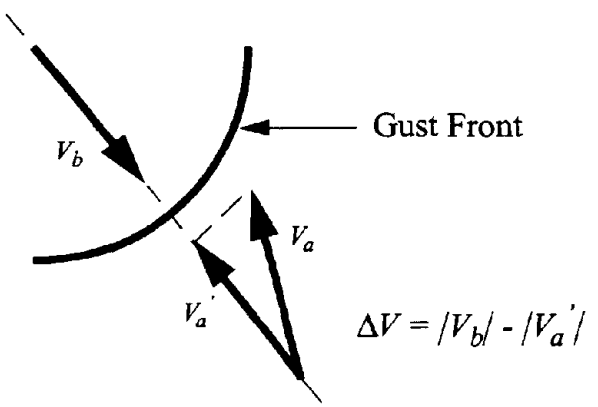


Figure 50. Illustration of vector difference method for computing the delta-V (ΔV) across a gust front.

2.6.2.2.2 "ByHistory" delta-V estimator

See Section 2.6.2.1.3.

2.6.2.2.3 "BySegment" maximum delta-V estimator

This is a very simple method which merely copies the previously-computed combined deltaV estimate (*deltaV*) of the input gust front chain and adjusts the weight of the estimate. That deltaV estimate was computed over the wind estimation segment, hence the name of this method. The weight that is returned is simply the weight of the consensus *deltaV* multiplied by the maximum weight given by *gfParams->maxWeightMaxDeltaVInSegment* (nominally, 1.0).

2.6.3 Heuristics

Heuristics are functions that invoke rules to eliminate or add further support to candidate gust front chains, making use of knowledge of typical gust front behavior. The set of heuristics that can be used (essentially, functions that implement rules) are defined by the TCL parameter *Heuristics* in the parameter file referenced by the GF_HEU_PARM_FILE environment variable (typically named "heuristicParamsWSP"). The methods named by the *Heuristics* parameter are loaded at startup by **GFAllocObjects** (see Section 2.1) into a global list variable named *heuristics*. See Appendix A.5 for an example of this file.

Function **GFApplyHeuristics** is called by **GFAnalyze** to invoke the heuristic methods that have been loaded onto the *heuristics* list. For every chain in the list of candidate chains, four heuristics are currently run: *ChainLengthSmall*, *InterestingDeltaV*, *SideLobeChain*, and *ChainLagsBhdArpWinds*. Each of these methods returns a boolean variable, *unindex*, that indicates whether the entire chain is to be unindexed (some of the heuristics are designed to perform their own conditional unindexing of selected points within a chain, and will return a value of false for *unindex*). If an entire chain is to be unindexed, **GFApplyHeuristics** will loop over every point in the chain, setting the *priorIndex* to *nil* and the *depth* to 0.

Table 25 lists the default parameters for the heuristics.

Table 25. Default Parameters Used by Heuristics

Heuristic	Parameter Name	Nominal Value	Units
ChainLengthSmall	minIndexedChainLength	11	pixels
InterestingDeltaV	minDeltaVWeight	0.0075	unitless
InterestingDeltaV	fuzzyInterestingDVDV	(0.0, 10.0)	m/s

Following is a brief description of each of the heuristics:

2.6.3.1 ChainLengthSmall

If the indexed (non-zero depth) chain length is less than *gfParams->minIndexedChainLength*, the entire chain is unindexed (depth set to 0).

2.6.3.2 InterestingDeltaV

The overall chain interest score of a gust front chain (*chain->score*) is increased as a function of the convergence strength (delta-V) of the front. First, the smallest delta-V estimate of the chain ("deltaVByVectorDiff" or "deltaVByShear") is chosen for consideration. If its associated confidence weight is greater than *gfParams->minDeltaVWeight*, then function **SKRisingRamp** is called to compute a scale factor using the rising ramp function nodes defined by *gfParams->fuzzyInterestingDVDV*. The scale factor is multiplied by 2 and then limited to a minimum value of 1.0. The *chain->score* is then multiplied by the scale factor.

2.6.3.3 SideLobeChain

High reflectivity weather at close range can produce strong signals which show up in sidelobes of the ASR-9 antenna pattern. This produces ghost echoes located identical range intervals, but significantly displaced in azimuth from the primary weather echo. The smearing of the signal by the sidelobes results in an azimuthal arc of low reflectivity echoes and a corresponding arc of coherent Doppler velocity estimates having the same Doppler velocity sign and magnitude as the primary weather echo source. The arc may

subtend more than 180 degrees and its occurrence is generally limited to weather reflectivities greater than 40 dBZ occurring within 20 km of the radar. The sidelobe returns are typically less than 0 DBZ, but have been observed as strong as 5 dBZ. The reflectivity and annular range extent of the sidelobe returns are comparable to those expected for gust front reflectivity thin lines and can therefore be a potential source of false detections. As the high intensity weather approaches, the radius of the associated sidelobe ring decreases. Thus, a gust front chain that has resulted from detection of a sidelobe echo will have propagation directions that reflect this behavior. Figure 51 shows a particularly strong occurrence of sidelobe contamination. In this case, the contamination is apparent in both the reflectivity and low-beam velocity data.

Gust front chains that are associated with sidelobe contamination are identified by examining the chain point propagation directions for the first, middle, and last points on the chain. If the propagation direction difference between the first and last point is greater than 40 degrees (implying large convergent or divergent front propagation), and if the propagation directions of the first, middle, and last points are all within 14 degrees of radial alignment with the ASR-9, then the chain is considered to be a sidelobe artifact, and the entire chain is unindexed.

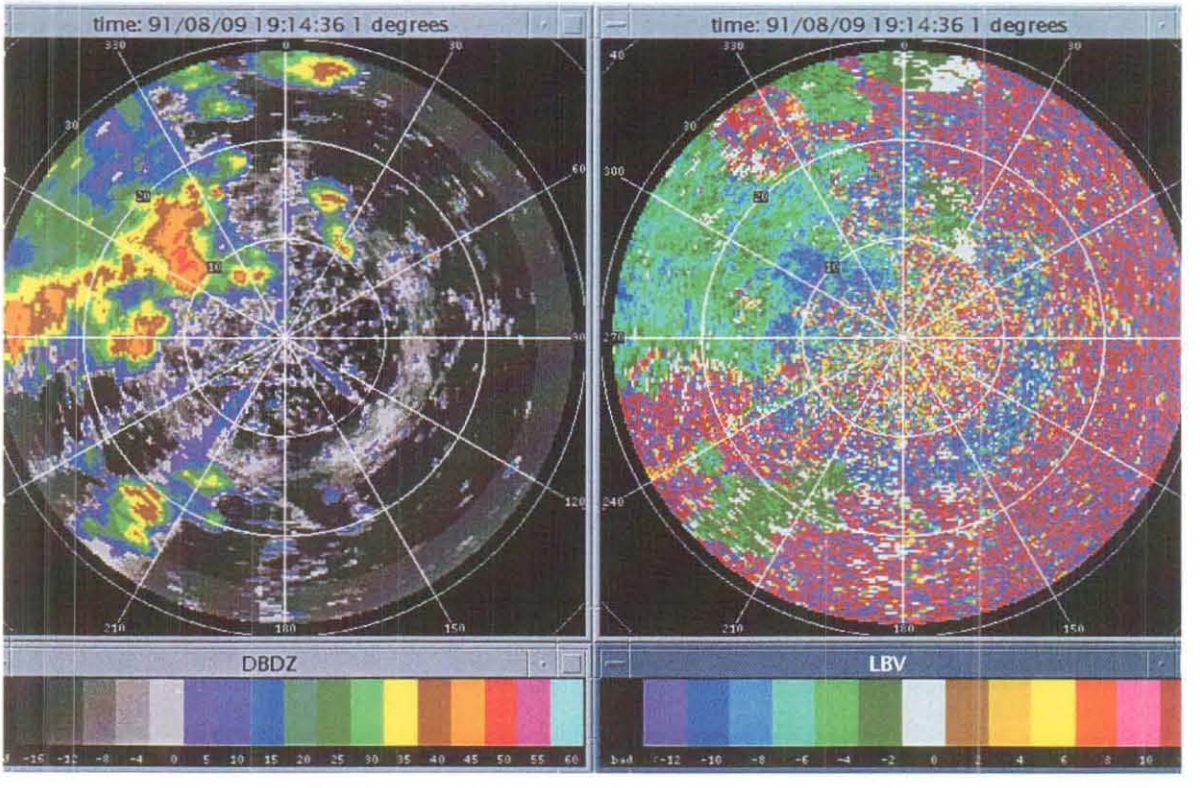


Figure 51. Example of sidelobe contamination in ASR-9 WSP reflectivity (DBDZ) and velocity (LBV) images. Sidelobe contamination appears as a broad ring of slightly elevated reflectivity values and a corresponding ring of lower Doppler variance (compared to clear air noise). In this example, the rings can be seen at a range of 13 km. Range rings are in km, reflectivity in dBZ, and Doppler velocity in m/s.

2.6.3.4 ChainLagsBhdArpWinds

Function **GFChainLagsBhdWinds** unindexes chain points approaching the radar whose point movement does not indicate convergence with respect to the airport wind. For each point in a gust front chain, the propagation speed and direction are obtained. Function **SKPointApproachingLocation** is called to evaluate the propagation direction and location of a gust front chain point to determine if the point is approaching the radar. If the point is approaching, and the distance of the point from the radar is less than 20 km, and the propagation direction of the point is within 60 degrees of the radar azimuth of the point, then the point is to be considered as a candidate for unindexing. Function **GFComputeVectorDeltaV** is then called to compute the vector difference of the chain point propagation velocity and the airport wind (*gfEvent->airportWind*) . If the vector difference (delta-V) is negative, then there is divergence, and the point is unindexed (i.e., the *priorIndex* is set to *nil* and the *depth* is set to 0).

2.6.4 Making the Final Report Decision

Function **MakeReportDecision** is called by **GFAnalyze** with the list of candidate gust front chains in order to subject the chains to a final series of tests that will decide whether each candidate front is to be reported or not. The decision is based on multiple factors, including the overall interest score of the front, how long the front has been tracked (the historical depth), the propagation speed of the gust front, and its strength (delta-V). Briefly, a front is not reported if:

1. The interest score is low and the front has not been tracked long enough (tracking depth test).
2. The number of indexed points with prior tracking history is too small (indexed length test).
3. The front has a low-confidence delta-V estimate and is not near the airport (delta-V distance test).
4. The delta-V as a function of the propagation and wind shift speeds is too low (delta-V speed test).
5. The reportable gust front length after pruning of low-confidence points is too short (reportable length test).

MakeReportDecision starts by initializing a local variable *reportDecision* to zero (thereby assuming initially that the front will not be reported). Each of the five tests executed in the order listed above results in a revision of *reportDecision* to a value of 0 or 1. The final value of *reportDecision* is assigned to *chain->display* before leaving function **MakeReportDecision**. Table 26 lists the default settings for the parameters involved in **MakeReportDecision**. Details of the report decision tests follow.

Table 26. Default Parameters Used by MakeReportDecision

Tests	Parameter Name	Nominal Value	Units
Tracking depth test	depthThresholdWithNoDV	(100000, 6800, 4800, 3900, 3000, 0)	interest score
Tracking depth test, Indexed length test	depthThresholdWithDV	(100000, 7000, 5000, 4000, 35000, 0)	interest score
Tracking depth test, Delta-V speed test	decisionMinDeltaV	(6.0 , 6.0 , 6.0, 6.0 , 6.0 , 6.0 , 5.0 , 5.0 , 5.0 , 5.0 , 5.0 , 4.0 , 4.0 , 4.0 , 4.0 , 4.0 , 3.0 , 3.0 , 3.0 , 3.0 , 3.0 , 3.0 , 3.0 , 3.0 , 3.0 , 3.0 , 2.57, 2.57, 2.57, 2.57, 2.57, 2.57, 2.57, 2.57, 2.57, 2.57)	m/s
Delta-V distance test	gfImpactRadius	6	km
Delta-V distance test	minDeltaVWeight	0.0075	m/s
Reportable length test	minReportLength	16	pixels

2.6.4.1 Tracking depth test

Low confidence detections (i.e., fronts with a low interest score) require a longer tracking history before they are reported. The chain being tested is passed to function **RequiredChainDepth** to compute the minimum required tracking depth for the chain based on the overall chain interest score. The required depth is obtained by comparing the *chain->score* against decreasing score thresholds ordered by increasing depth index in one of two parameter look-up tables: *gfParams->depthThresholdTableWithNoDV*, and *gfParams->depthThresholdTableWithDV*.

The choice of look-up table is determined by the strength and confidence (the weight) of the consensus delta-V estimate. If the delta-V weight is less than the parameter *gfParams->minDeltaVWeight* (indicating very low confidence) or if the delta-V value is less than the minimum reportable delta-V as given by the last element of *gfParams->decisionMinDeltaV* (nominally 2.57 m/s), then the delta-V is weak or has low confidence, and *gfParams->depthThresholdTableWithNoDV* is chosen as the look-up table. This table has lower score thresholds for each depth than the *gfParams->depthThresholdTableWithDV* table that is otherwise chosen. Given the *chain->score*, the required depth is the index of the lowest ordered entry in the depth table (highest score) that is exceeded by the chain score value. The chain is flagged as "reported" (*reportDecision* = 1) if the *chain->windSegAvgDepth* is greater than or equal to the required depth.

2.6.4.2 Indexed length test

The chain under test is passed to function **GetIndexedPoints** which, if the *chain->score* exceeds the minimum score contained in the last element of the *gfParams->depthThresholdTableWithDV,* returns the number of points in the chain that have a tracking depth that is 1 or higher. Otherwise, if the *chain->score* is too low, the number of indexed points is simply set to zero. If the number of indexed points returned by **GetIndexedPoints** is less than or equal to 7, then the indexed length of the chain is too small and *reportDecision* is consequently set to zero.

2.6.4.3 Delta-V distance test

If the nearest point of the candidate gust front is greater than *gfParams->gfImpactRadius* (nominally, 6 km) from the airport, and if the weight of *chain->maxDelta*V is less than *gfParams->minDeltaVWeight,* then the chain is not reported (*reportDecision* = 0).

2.6.4.4 Delta-V speed test

The final delta-V test is based on the minimum of the front's propagation speed and the speed of the winds behind the front. The *propSpeed* is obtained from *chain->windSegAvgSpeed*, and the speed of the wind behind the front (*speedBehind)* is obtained from *chain->windBehind*. The minimum of the *propSpeed* and the *speedBehind* is multiplied by 10 and rounded to the nearest integer to generate an index that is applied to the *gfParams->decisionMinDeltaV* table. The table returns a minimum delta-V threshold (*dvThresh*). (The *gfParams->decisionMinDeltaV* table is a list of delta-V values that decrease with increasing speed index. Therefore, faster moving fronts have a lower minimum delta-V threshold.)

If the *chain->deltaV* is less than *dvThresh*, then *reportDecision* is set to 0. In addition, the more drastic step is taken of unindexing the front in order to avoid generating a tracking history for the front. The unindexing consists of setting the *priorIndex* to *nil* and *depth* to 0 for the chain, and setting the *chain->windSegAvgDepth* to 0.

2.6.4.5 Reportable length test

The final test requires that the reportable length of the front (i.e., the length of all points for which tracking history is sufficient to allow position predictions to be made), exceeds a minimum length given by *gfParams->minSDPredictionDepth*. Function **PredictFrontAtTime** (described in more detail in the next section) is called with a requested prediction time interval of 0 minutes. With a zero-minute prediction interval, no point extrapolations are made, but logic that removes low confidence, zero-depth chain fragments near the ends of a gust front chain is executed. If the number of remaining chain points after calling **PredictFrontAtTime** is less than *gfParams->minReportLength*, then the chain is not reported (*reportDecision* = 0).

2.7 Predictions (position forecasts)

The extracted *event*, containing all gust fronts detected for the volume scan, is used to make predictions of where the points having sufficient depth and interest are likely to be at some time in the future. Given the direction moved, the propagation speed, and the current coordinates of a point, a new coordinate is computed for some time in the future.

Function **MakeDisplayPredictions** is called by **GFAnalyze** to generate a complete set of 1-minute interval predictions for each chain in the list of chains for the current *event*. The number of predictions generated is governed by the *gfParams->numPredictionTimes* parameter (nominally, 36). For each prediction interval, a prediction chain of type GFPredChain is computed by function **PredictFrontAtTime**.

For every point in the chain, **PredictFrontAtTime** checks to make sure that the point's tracking depth is greater than *gfParams->minSDPredictionDepth*. If the depth is sufficient, then the point is added to a list of valid depth points. The current default setting for the minimum depth is zero, so initially all points on the chain are put on the list (extrapolations can still be made for depth zero points because previous chain attribute smoothing operations fill in the missing propagation information with averages of nearby chain values).

Function **PruneBadTerminalPoints** is called by **PredictFrontAtTime** to iteratively prune away short segments of zero-depth points near the end of the gust front chain where tracking is often less reliable (e.g., there may be many zero-depth points interspersed with positive-depth points in cases where gust front length increases from scan to scan). The routine is called twice - once for each end of the chain.

Function **BuildPredictionChain** is then called to actually fill in a gust front prediction for the particular prediction time interval. The input list of valid depth detection points are translated forward according to their individual propagation velocities. Then, duplicate points are removed and small gaps filled in via straight lines (larger gaps will remain unfilled). Finally, duplicate prediction chain points that may have been introduced by the filling are again purged from the list, before returning the fill gust front prediction chain.

2.8 Updating the image history

The last operation prior to output is to update the image history lists that are used to supply prior images for operations such as motion differencing. Function **GFBuildImageHistory** is called by the main MIGFA driver to update the image histories. The updated histories include: *gfRefData->dzHistory* (reflectivity image history), *gfRefData->vHistory* (velocity image history), *gfRefData->convHistory* (convergence image history), and *gfRefData->sdHistory* (velocity standard deviation image history). Each of the image histories is hard-limited to a maximum of 3 images. Each time a new image is added, the oldest one is deleted from the history list.

2.9 Output

As summarized in Section 1.4, MIGFA has a primary product output stream, an optional archival output stream, and a diagnostic analysis display stream.

2.9.1 Analysis Display Output

If the optional MIGFA analysis display is running (*migfaDisplay)*, then selected image arrays computed during processing are displayed at the end of the processing cycle just prior to product output (see Figure 2).

Given a list of all available images (contained in the global *imageListAll* variable), and the list of image names that the display daemon knows about (global *displayList*), function **BuildImageDisplayList** is called to create a sublist of matching image names (*matchedDisplayList*) that is passed to function **DisplayImages** for display. Overlay graphics are then added to selected image windows. The candidate chains are displayed by function **GFDrawDetections** in the image window labelled "CANDIDATES". The final reported current locations, and associated 10- and 20-minute predictions are displayed by function **GFDrawPredictions** in the image window labelled "PREDICTIONS".

The latest airport anemometer measurement stored at the top of the *gfRefData->anemometerHistory* is displayed on the PREDICTIONS window by function **DisplayAirportWind.** An single informational string is constructed and at the top of the analysis display by function **SetDisplayVar.** The informational string contains:

1. Scan time
2. Seasonal sensitivity mode setting (summer vs. winter)
3. Stratiform rain and storm cell precipitation coverage numbers
4. Regional wind speed and direction (*gfEvent->regionalWind*) as computed from the Doppler velocity data by **GFComputeVAD** (Section 2.2.7.3)

2.9.2 Product and Archive Output

Handler routines to perform output formatting and transmission for the product and archival streams are assigned by the **OpenOutputs** function during algorithm initialization. Function **OutputGFFeatures** is the output handler assigned to *gfConfig->outHandler* (a function pointer variable) for output of end-user product records only. It extracts the gust front detection and forecast information from the current GFEvent data object, and packs and sends the various data records as described in Appendix B.

If diagnostic images are to be output along with the product records, then function **ArchiveGFObjects** is the output handler that is invoked. It packs the SKArray image data for the requested images into separate SKArray WxObject records that are transmitted prior to calling **OutputGFFeatures** to pack and send the standard set of product data records.

3. SOFTWARE DESCRIPTION

3.1 Overview

MIGFA was originally prototyped using LISP and then re-written using the C++ programming language. Both of these object-oriented languages were chosen because of the need to manage a complex processing task that utilizes a large number of dynamically allocated data objects. Using an object oriented language like C++ greatly simplifies algorithm code by freeing the programmer from having to explicitly manage the overhead of data object instantiation, initialization, and destruction. Furthermore, object oriented methods are tightly coupled to the data that they operate on (encapsulation), thereby providing additional safety against illegal software instructions that can cause improper behavior or crashes.

3.2 Organization of Software Modules

Approximately 60% of the code that makes up the WSP version of MIGFA is common to all versions of MIGFA. The common MIGFA code modules are located in subdirectories under the "*migfa/*" directory. WSP-specific algorithm software modules are located under the "*wspmigfa/*" directory. The main driver routine for the WSP MIGFA is function main() in file migfaDriver.C in *wspmigfa/realtime/src*. Table 27 provides a brief description of the contents of each source file subdirectory under the *migfa* and *wspmigfa* directories.

MIGFA makes use of a number of software libraries developed at Lincoln Laboratory and included with the WSP software distribution. The source code for these packages is located in separate directories outside of the MIGFA software. Table 28 lists the required Lincoln software libraries.

Table 27. MIGFA Software Directory Contents

Directory	Contents
migfa/inc	Class definitions, data structures, constants, function prototypes
migfa/analyze	Routines to edit chains, apply heuristic tests, wind analysis driver
migfa/colormaps	Colormap files for imgsh displays
migfa/common	Common utility functions used by all MIGFAs
migfa/debug	GFDebug class (used for diagnostic SKArray file output)
migfa/extract	Chain extraction routines
migfa/intgen	Feature detector driver, thin line interest smoother
migfa/io	Input/output routines
migfa/predict	Routines for generating gust front prediction curves
wspmigfa/incWSP	WSP-specific data classes, structures and constants (including feature detector classes)
wspmigfa/detectors	WSP-specific feature detectors
wspmigfa/disptcl	Tcl/Tk imgsh diplay scripts
wspmigfa/params	General and local site-specific parameter files
wspmigfa/realtime	Main driver routine
wspmigfa/tools	Utilities to support performance analysis
wspmigfa/windest	WSP-specific gust front wind estimators
wspmigfa/wsp	WSP-specific I/O, data preparation routines

Table 28. Additional Lincoln Software Libraries Used by WSP MIGFA

Library Name	Contents
csketch	C++ image processing library
sclite	C-language server-client interprocess communication library supporting TCP and UDP protocols
share	Shared memory ring buffer routines, WSP base data record parsing and packing routines, Tcl parameter parsing routines
WSP	Miscellaneous WSP-specific utility functions
WxObj	C++ library for packing/unpacking algorithm product data records and interface for sending and receiving data via server-client routines in *sclite* library
imgsh	Tcl/Tk extension to generate displays of image data
llutil	Utility classes for managing linked lists, stacks, and queues; LLTime class for manipulating time stamps
mem	Custom memory manager
log	Message logging
paramTcl	Tcl parameter parsing

3.3 Required External Software

Several additional public domain software packages are required to build and run the MIGFA software:

Tcl/Tk
Perl
GNU g++/gcc (C++ compiler)
GNU make utility (gmake)
Zlib compression library

3.4 Navigating the Software

With some 19,000 lines of common and WSP-specific MIGFA software (not including supporting libraries like CSKETCH), and functions divided among many separate files and directories, navigating the software can be a daunting task. A freeware C++-to-HTML conversion program called *ctoohtml* (http://www-anw.cs.umass.edu/~fagg/ctoohtml/ctoohtml.html) can be used to automatically generate HTML pages for each of the MIGFA software source files. The *MakeHTML* script in the wspmigfa/scripts

directory runs the *ctoohtml* utility with the proper options and lists of software files to create the HTML files in the directory in which it is executed (the script assumes that *ctoohtml* is installed and is in the user's executable search path).

Once the HTML files have been generated, a web browser can be used to follow hypertext links to function calls, data object definitions, or constants (these are referred to as "tags"). In addition to the individual source code HTML files, *ctoohtml* also produces a number of useful index files that can be useful starting points:

1. files.html - Alphabetically ordered list of HTML source files
2. tags.all.html - Alphabetically ordered list of tags
3. tags.file.html - Tags sorted by source file
4. migfaDriver.C.html - HTML version of the main driver routine

Alternatively, for those familiar with the emacs editor, an emacs tags file can be generated using the *maket-ags* script in the wspmigfa/scripts directory. This tags file can then be loaded with the emacs *visit-tags-table* command and used in the editor to quickly jump to functions, automatically loading the files.

3.5 Running the WSP MIGFA

Typing "**wspmigfa -h**" will print the list of command line options:

```
Usage: wspmigfa [options]

 Options:

  -r vmeRingAddr  Specify that base data input is from VME
                  ring buffer (default).  Address is in hex
                  The keyword 'default' can be used to specify
                  the default address

  -m shmKey       Specify that base data input is from shared
                  memory ring buffer. Address (key) is in hex
                  The keyword 'default' can be used to specify
                  the default address

  -c ctrlStream   Name of VSP input control stream.
                  'default' results in /SC/vsp

  -f              Use full range (240 gates) of WSP DBV product.
                  Default value (for old data) is 160 gates.

  -w streamName   Stream name for input airport wind data.
                  'default' results in /SC/llwas_6

  -o streamName   Output stream name.  A streamName of 'default'
                  results in /SC/MIGFA_GF_DETECTIONS

  -a archiveName  Optional MIGFA output archive filename
```

```
-i arcNames      Quoted list of image names to output

-p               Enable output of images (listed with -i option)
                 to output stream in addition to archive file

-d debugDir      Turn on debugging with output to debugDir

-n displayName   Set Tk display window to displayName

-D               Turn on debugging for detectors.

-E               Turn on debugging for extract routines.

-H               Turn on debugging for heuristics.

-I               Turn on debugging for I/O.

-W               Turn on debugging for wind estimators.

-s nScans        Quit after processing nScans (default: loop forever

-N               No diagnostic display (default is display enabled)

-O               Offline mode (base data ring is not flushed on start-up)
-h               Print (this) help message
```

3.5.1 Input and Output Options

For real-time operations on the WSP system, base data are received via the VME ring buffer, so the **-r** option should be used. Alternatively, the **-m** option sets up input on a shared memory ring. This is the option used when using the WSP ***bdplay*** utility to transmit base data from files or tape to the algorithm. The **-r** and **-m** options take address keys as arguments. If "default" is specified, then the address defined by GF_RING_ADDR (0x21800000) in *$SW_HOME/share/inc/wsp_comm.h* is used. The address specified must match the address being utilized by the base data sender.

The **-o** option is used to specify the product output stream. If the name given as the argument to the **-o** option is prefixed with **/SC/** or **/ServerClient/** then output will be on a stream via TCP. The name that follows the prefix must be associated with a unique port ID in the configuration file pointed to by the SC_CONFIG_FILE or ALG_SERVICES environment variables (the setting of SC_CONFIG_FILE, if present, will override that of ALG_SERVICES). If the argument to the **-o** option is not prefixed, then output will be to a disk file with the name provided.

A small set of VSPs may be transmitted by the WSP remote monitoring function at any time, and will immediately affect algorithm processing without having to restart the process. The WSP MIGFA receives the VSPs on a TCP stream specified with the **-c** option.

Older WSP data recorded prior to 3/6/00 contain only 160 gates (10 n mi) of dual-beam velocity (DBV) data. Specifying the **-f** option tells the WSP MIGFA to process the full 240 gates of DBV data now available.

Airport wind data (typically from the LLWAS centerfield anemometer) are currently transmitted as a type of WSP base data record on the WSP base data stream. Previously, wind data were received on a separate TCP/IP stream. If an external LLWAS stream connection is desired (with data provided by a suitable LLWAS data server process), then the **-w** option can be used to specify the stream name. This name must be registered in the SC_CONFIG_FILE or ALG_SERVICES configuration file.

For offline archival of algorithm products, the **-a** option can be used to specify the archive filename. This option may also be used to specify a second output TCP/IP product stream by using the **/SC/** prefix and registering the stream name in the SC_CONFIG_FILE or ALG_SERVICES configuration file. Diagnostic SKArray images may also be archived to the stream specified by the **-a** option by using the **-i** option followed by a quoted list of image names to be output. The image names must correspond to the image names that are used to index the *pinfo* array in the **migfaDisplay** imgsh script. Similarly, the **-p** option can be used to output images on the standard product output stream specified by the **-o** option.

3.5.2 Debugging Output Options

Several command-line options are used to control output of information for debugging. Specifying the **-d** option turns on a basic level of debugging with output to the directory specified. A large number of individual SKArray image files will be output to the directory with names indicative of their purpose in the processing. The *gfDebug.writeSKArray* command is used to write all debug images in MIGFA. This command can be searched for in the code (using the UNIX **grep** command for example) to identify which stage in the processing the image corresponds. Additional diagnostic SKArray image files for the feature detectors will be output if the **-D** option is specified. Specifying the **-E** option will output diagnostic text and SKArray image files that indicate the various stages of chain extraction. Specifying the **-H** option will cause the names of the heuristics to be printed as they are being executed. The **-I** option is reserved for future implementation of diagnostic message output related to I/O. It currently does not produce any output.

The **-W** option turns on diagnostic text output showing the results of the various wind estimators. Figure 52 shows an example of the output that is produced. Consensus (combined) values for the "WIND BEHIND" are printed first in u(east), v(north) vector component form along with their corresponding confidence weights (uweight and vweight). The constituent estimates that were combined to form the consensus (WIND BEHIND DATA) follow and are printed in similar fashion. The diagnostic output continues with the printing of the combined and constituent wind estimates for the "WINDS INSIDE" and "WINDS AHEAD" regions. Finally, the combined and constituent estimates for the "DELTA-V" and the "MAX DELTA-V" are printed.

```
WIND BEHIND:
 combined u: -4.53548 (uweight = 0.0942025)
 combined v: -5.8599 (vweight = 0.0942025)
 WIND BEHIND DATA:
  behindByNormal u: -4.58752 (uweight = 0.0968485)
  behindByNormal v: -5.96565 (vweight = 0.0968485)
  behindByHistory u: -4.8935 (uweight = 4.68303e-06)
  behindByHistory v: -5.67885 (vweight = 4.68303e-06)
  behindByAnemometer u: -2.73405 (uweight = 0.00279874)
  behindByAnemometer v: -2.201 (vweight = 0.00279874)
WIND INSIDE:
 combined u: -4.53548 (uweight = 0.0942025)
 combined v: -5.8599 (vweight = 0.0942025)
 WIND INSIDE DATA:
  insideByNormal u: -4.58752 (uweight = 0.0968485)
  insideByNormal v: -5.96565 (vweight = 0.0968485)
  insideByHistory u: -4.8935 (uweight = 4.68303e-06)
  insideByHistory v: -5.67885 (vweight = 4.68303e-06)
  insideByAnemometer u: -2.73405 (uweight = 0.00279874)
  insideByAnemometer v: -2.201 (vweight = 0.00279874)
WIND AHEAD:
 combined u: 0 (uweight = 0.0999002)
 combined v: 0 (vweight = 0.0999002)
 WIND AHEAD DATA:
  aheadByHistory u: 0 (uweight = 0.0001)
  aheadByHistory v: 0 (vweight = 0.0001)
  AheadByAnemometer u: 0 (uweight = 0.1)
  AheadByAnemometer v: 0 (vweight = 0.1)
DELTA-V:
 combined u: 4.99778 (uweight = 0.0797058)
 DELTA-V DATA:
  deltaVByVectorDiff u: 7.41007 (uweight = 0.0942025)
  deltaVByShear u: 0.863931 (uweight = 0.0549205)
  deltaVByHistory u: 1.18872 (uweight = 5.53671e-05)
MAX DELTA-V:
 combined u: 4.99778 (uweight = 0.0797058)
 MAX DELTA-V DATA:
  maxDeltaVBySegment u: 4.99778 (uweight = 0.0797058)
  maxDeltaVByHistory u: 1.18872 (uweight = 5.53671e-05)
```

Figure 52. Example wind estimation diagnostic output generated with the -W MIGFA debugging option.

3.5.3 Options Related to Offline Processing

The remaining options are provided for additional control during offline processing using recorded base data. The **-s** option can be used to specify a maximum number of scans to process before termination (typically the algorithm will cycle forever). Specifying the **-O** (offline mode) option prevents the usual flushing of the input base data ring buffer on start-up. Depending on the timing of base data arrival to the ring buffer during offline processing, the flushing may cause some of the data for the first scan to be inadvertently discarded before being processed. Finally, the **-N** option disables the usual output to the MIGFA diagnostic display (if the **-N** option is not specified, then the **migfaDisplay** imgsh script must be running, or MIGFA will hang when it tries to send display commands).

3.5.4 MIGFA Diagnostic Display

The optional MIGFA display is a separate imgsh process that MIGFA communicates with in order to display the many images that are generated in the course of processing (See Newell[3] for a description of the Tcl/Tk-based imgsh graphics package). As shown in Figure 2, the display is divided up into 16 image sub-windows. The image names are shown at the upper left of each image. If the left mouse button is held down on any of the image labels, a pull-down menu of all displayable images is presented (there are more images available for display than the 16 sub-windows allow). To restore the original sequence of images on the display, select "Restore Default Display" from the "Options" menu in the upper left corner of the display.

The mouse can be used to interact with the display in a number of useful ways. By moving the cursor over any of the image sub-windows, a continuous readout of the underlying image values is displayed in the upper right corner of each image sub-window (the actual arrow-shaped cursor is echoed as a plus-shaped cursor at the corresponding image location in all of the other image sub-windows). The radar range and azimuth of the pointer location are continuously displayed in the upper left corner of the MIGFA display.

Another useful feature is the zoom feature. To zoom in on a particular region, hold the left mouse button while dragging the cursor to create a zoom box. After stretching the zoom box to the desired size, release the left mouse button, then hold down the right mouse button to obtain a zoom menu. From the zoom menu, you can select "Zoom" to zoom only the image sub-window that the cursor is currently in. Alternatively, you can select "ZoomAll" to simultaneously zoom all of the image sub-windows to the same zoom region selected in the current sub-window. To return to the original full-size view, hold down the right mouse button and select "UnZoom" (for the current window) or "UnZoomAll" (to unzoom all of the windows).

Appendix A: MIGFA PARAMETER FILES

This appendix contains printouts of the default parameter settings for the five required MIGFA parameter files.

A.1. GF_WSP_CONFIG_FILE

```
##########################################################################
#
#       File:  configWSP
#
#
#       Basic fields of the GFIntGenConfig structure which must be known
#       before image sizes, etc. can be computed.  This file specifies
#       default configuration variables for WSP MIGFA.
#
#       IMPORTANT NOTE regarding "cartRadiusKM".  This variable must be chosen
#       so that the resulting number of needed gates of data is a multiple of
#       4.  (This in turn is needed since the MIGFA algorithm needs to shrink
#       polar arrays down by a factor of 4 for some computations).  For example,
#       suppose a coverage radius of 65.0 km is desired.  With a "gateSizeMtrs"
#       setting of 150.0 meters, this would require 433.33 gates of data to
#       cover the desired radius ((150.0 meters / gate) * 433.33 gates = 65,000
#       meters).  This number of gates should be bumped up to 436 (436 is the
#       smallest integer which is a multiple of 4 and is greater than 433.33.
#       The required coverage radius is thus 436 * 150 = 65,400 meters.
#
#
##########################################################################
#

# Location of airport reference point (ARP) with respect to the radar.
# X is positive for locations east of the radar, Y is positive for locations
# to the north.
set arpOffsetXKM      0.0
set arpOffsetYKM      0.0

# Max input processing range from the radar in kilometers
set cartRadiusKM    27.78


# Radius in km of a mask used to identify pixels near the radar site
set siteMaskRadiusKM  2.0

# Internal processing resolution of Cartesian images.
set metersPerPixel  480.0
set kmsPerPixel       0.48

# Number of radials in a scan
set radialsPerScan    256

# Number of meters per range gate in input base data
set gateSizeMtrs    115.75

# Azimuth resolution of input base data
```

```
set azPerRadial    1.40625

# Names of input base data images recognized by the display daemon
set inImages       "LBDZ LBV HBDZ DBDZ DBV FLAGS"

# used to control radar-dependent algorithm logic.
# Choices are "TDWR", "ASR9", or "NEXRAD"
set radarSystem    "ASR9"

# SNR base data product available? 0=FALSE, 1=TRUE
set snProductAvailable     0

# Enable(1)/disable(0) SNR thresholding within feature dectetors
set doSnDetectorProcessing  0

# Enable(1)/disable(0) use of out-of-trip image within feature detectors
set doOotDetectorProcessing  0

# Enable(1)/disable(0) airport-centered processing
set doAirportCenteredProcessing  0

# Enable(1)/disable(0) masking of interest images wherever velocity data
# is missing
set doVMasking             0

# Enable(1)/disable(0) "aggressive" chain extension.  Conservative for
# ITWS, aggressive for G&D & WSP.
set doAggressiveExtend  1

# Max detection coverage range in km if airport-centered processing is
# enabled
set airportCoverageRadiusKM   -1.0

# Max detection coverage range in pixels if airport-centered processing is
# enabled
set airportCoverageRadiusInPixels  -1.0
```

A.2. GF_WSP_PARAM_FILE

```
##############################################################################
#
#       File: paramsWSP
#
#       Description:
#
#     Default algorithm parameters for MIGFA.  This file should be read in
#     first, and any local parameters should be set in another parameter
#       file.  To set a multiline parameter, the open brace must come on the
#       same line as the set keyword.
#
#       cvs id is: "@(#)$Id: paramsWSP,v 1.5 2001/01/05 16:36:28 setht Exp $ MIT/LL"
#
##############################################################################

# The time intervals in seconds between current volume scan and prior
# volume scan for computing predictions for short-term and long-term
# anticipation.
set anticipationIntervals  { 120  240 }

# The distance from the airport in kilometers at which the
# rules for selecting the gust front wind estimation segment change. Used
# in Function gfCreateWindEstSegment.
set arpControlRadiusKM              15.0

# The maximum forecast time in seconds of a gust front used with the
# airport-control-radius* to determine the method of selecting a
# gust front segment.  Used in Function gfCreateWindEstSegment.
set arpControlTimeSec               1200

# The minimum angular deviation in degrees between points before a non-nil
# interest value is assigned. Used in Function
# detect-breaks-by-direction.
set breaksByDirCriterionAngle       45

# Factor used to convert angle differences into interest values.
# Used in Function detect-breaks-by-direction.
set breaksByDirFactor         4.266667

# Factor used to convert computed external angles into interest
# values. Used in Function detect-breaks-by-external-angle.
set breaksByExtAngleFactor          3.94

# The size of the window over which the first derivative
# is computed. Used in Function detect-breaks-by-distance
# and detect-breaks-by-speed.
set breaksByWSize                   6
```

```
# For low wind speeds, the maximum allowed difference in degrees
# between the directions of the combined wind estimate and the
# estimate generated using the ``by-normal" technique. Used in
# Function GFFixCalmWindDirection.
set calmWindAngleDiff           60

# The wind speed in m/s below which the wind direction may be
# unreliable. Used in Function GFFixCalmWindDirection.
set calmWindSpeed     1.5

# The number of points at the beginning end of the chain to
# be included in statistics. Used in Function ChainEndStatistics.
set chainEndSublength           10

# The necessary ratio of disagreeing to agreeing neighbors for a
# direction to be reversed. Used in Function FlipDirectionsToConsensus().
set consensusFlipFactor         4.0

# The (one-sided) distance defining the local neighborhood when looking for
# direction consensus. Used in Function FlipDirectionsToConsensus().
set consensusSearchLimit        32

# A table of minimum delta V values in m/s indexed by the gust front
# propagation speed rounded to the nearest decimeter/sec. Used in function
# MakeReportDecision and RequiredChainDepth.
set decisionMinDeltaV {
 6.0  6.0  6.0  6.0
 6.0  6.0  5.0  5.0
 5.0  5.0  5.0  4.0
 4.0  4.0  4.0  4.0
 3.0  3.0  3.0  3.0
 3.0  3.0  3.0  3.0
 3.0  3.0  2.57
 2.57 2.57 2.57
 2.57 2.57 2.57
 2.57 2.57 2.57
}

# The number of elements in the decisionMinDeltaV table
set decisionMinDeltaVSize       36

# A table of minimum depth necessary for depths from 0 to 8 for
# gust fronts with reliable delta V estimates.
set depthThresholdTableWithDV {
 100000  7000  5000  4000  3500  0
}

# The of elements in the depthThresholdTableWithDV table
set depthThresholdTableWithDVSize     5
```

```
# A table of minimum depth necessary for depths from 0 to 8 for
# gust fronts without reliable delta V estimates.
set depthThresholdTableWithNoDV {
 100000  6800  4800  3900  3000  0
}

# The of elements in the depthThresholdTableWithNoDV table
set depthThresholdTableWithNoDVSize   5

# A table of numeric factors used to adjust weights as a function of
# depth in wind estimation functions.
set depthWeightTable            { 0.01  0.2  0.5  1.0 }

# Minimum interest value, above which interest scores for points with
# good viewing angles are recomputed.
set detectorMinInt              64

# The number of volume scans of computed results maintained in
# refData.eventHistory.
set eventHistoryLength          12

# The maximum difference in time between a requested time and events that
# correspond to it.
set eventTimeDifference         60

# The maximum number of pixels, minimum interest score, and the maximum
# angular deviation in degrees, that can be added for the first round of
# chain extensions in Subalgorithm migfaExtract.
set extend1Params               { 10  80  41 }

# The maximum number of pixels, minimum interest score, and the maximum
# angular deviation in degrees, that can be added for the second round of
# chain extensions in Subalgorithm migfaExtract.
set extend2Params               { 20  48  11 }

# The number of integer values used in scaling input images for
# functional template correlation.
set ftcImageRange               256

# A list of 6 numbers defining an s-plateau computation of weight
# as a function of the distance in km from a gust front to a sensor.
# Used in Function estimate-ahead-by-anemometer.
set fuzzyAheadByCenterDist      { 1.0  3.0  6.0  10.0  15.0  40.0 }

# A list of 3 numbers defining a falling-s computation of weight
# as a function of the difference in time in seconds between the
# volume scan time and a historical time. Used in Function
# estimate-ahead-by-anemometer.
set fuzzyAheadByAnemometerTime  { 1800.0  3600.0  7200.0 }
```

```
# A list of 3 numbers defining a falling-s computation of weight
# as a function of the difference in time in seconds between the
# volume scan time and a historical time. Used in Function
# estimate-ahead-by-site-wind.
set fuzzyAheadBySiteWindTime     { 1800.0   3600.0   7200.0 }

# A list of 2 numbers defining an falling-ramp computation of weight
# as a function of the time difference in seconds between the volume
# scan time and the TWIND analysis time. Used in Function
# estimate-ahead-by-oe.
set fuzzyAheadByTwindTime        { 300.0   900.0 }

# These 2 numbers (in seconds) define a falling-ramp computation of airport
# winds as a functions of time difference between the scan and the closest
# (in time) anemometer measurement in function ComputeAirportWind().
set fuzzyAirportWindTime         { 300.0   1800.0 }

# A list of 4 numbers defining a ramp-plateau computation of weight
# as a function of the standard error of velocity samples in m/s.
# Used in Function gfEstimateBehindByNormal.
set fuzzyBehindByNormalError     { 0.0   0.0   0.4   10.0 }

# A list of 3 numbers defining a rising-s computation of weight as
# a function of the propagation speed of a gust front in m/s. Used
# in Function estimate-behind-by-oe.
set fuzzyBehindByOESpeed         { 0.0   2.0   5.0 }

# A list of 2 numbers defining a falling-ramp computation of weight
# as a function of the elapsed time in seconds between the TDWR volume
# scan and the TWIND data time. Used in Function
# estimate-behind-by-twind.
set fuzzyBehindByTwindTime       { 0.0   0.0 }

# These 3 numbers (in degrees) define a falling-s computation of chain
# match score as a function of angle difference (at 2 chain ends) in
# function ScoreChainMatch().
set fuzzyChainMatchAngle         { 5.0   15.0   25.0 }

# These 2 numbers define a falling-ramp computation of
# chain  match score as a function of avg score difference (at 2 chain
# ends) in function ScoreChainMatch().
set fuzzyChainMatchAvgScore      { 20.0   160.0 }

# These 3 numbers (in meters/ sec) define a falling-s computation of chain
# match score as a function of speed in function ScoreChainMatch().
set fuzzyChainMatchSpeed         { 2.0   5.0   8.0 }

# These 3 numbers (unitless) define a rising-s computation of chain
# match score as a function of the sum of the interest scores of the
# 2 chains in function ScoreChainMatch().
```

```
set fuzzyChainMatchIntSum       { 1000.0  2000.0  3000.0 }

# These 3 numbers (unitless) define a rising-s computation of chain
# match score as a function of the minimum of the interest scores of the
# 2 chains in function ScoreChainMatch().
set fuzzyChainMatchIntMin       { 500.0  1000.0  1500.0 }

# A list of 2 numbers defining a falling-ramp computation of weight
# as a function of the difference in orientation in degrees between
# two gust fronts. Used in Function chasing-other-chains.
set fuzzyChasingChainsAngle     { 0.0  40.0 }

# A list of 2 numbers defining a falling-ramp computation of weight
# as a function of the difference in speed in m/s between two gust
# fronts. Used in Function chasing-other-chains.
set fuzzyChasingChainsSpeed     { -1.0  1.0 }

# A list of 2 numbers defining a rising-ramp computation of weight
# as a function of the stratiform rain number in *precip-numbers*.
# Used in Function chasing-other-chains.
set fuzzyChasingChainsStrat     { 2.0  4.0 }

# A list of 2 numbers defining a falling-ramp computation of weight
# as a function of the storm cell number in *precip-numbers*.
# Used in Function chasing-other-chains.
set fuzzyChasingChainsStorm     { 0.0  1.0 }

# A list of 6 numbers defining an s-plateau computation of weight
# as a function of the distance between the radar site and the
# gust front segment. Used in Function estimate-delta-v-by-shear.
set fuzzyDeltaVByShearDist      { 0.0  1.0  3.0  10.0  20.0  60.0 }

# A list of 3 numbers defining a falling-s computation of weight as
# a function of the standard error of velocity values in m/s.
# Used in Function estimate-delta-v-by-shear.
set fuzzyDeltaVByShearError     { 1.0  2.0  10.0 }

# A list of 6 numbers defining an s-plateau computation of weight
# as a function of the distance in km between a gust front and some
# location. Used in Function estimate-by-center.
set fuzzyEstimateByCenterDist   { 1.0  2.0  3.0  6.0  10.0  20.0 }

# A list of 2 numbers defining a rising-ramp computation of an
# adjustment factor as a function of high interest values. Used in
# Function compute-gf-score.
set fuzzyGFScoreInterest        { 240.0  255.0 }

# A list of 2 numbers defining a falling-ramp computation of weight
# as a function of distance in km. Used in Function
# get-historical-chain-attribute.
```

```
set fuzzyHistChainDist          { 10.0   20.0 }

# A list of 2 numbers defining a falling-ramp computation of weight
# as a function of angular difference in degrees. Used in Function
# get-historical-chain-attribute.
set fuzzyHistChainAngle         { 90.0   135.0 }

# A list of 4 numbers defining a ramp-plateau computation of weight
# as a function of the standard error of velocity samples in m/s.
# Used in Function estimate-inside-by-normal.
set fuzzyInsideByNormalError    { 0.0   0.0   0.4   10.0 }

# A list of 3 numbers defining a rising-s computation of weight as
# a function of the propagation speed of a gust front in m/s. Used
# in Function estimate-inside-by-oe.
set fuzzyInsideByOESpeed        { 0.0   2.0   5.0 }

# A list of 2 numbers defining a rising-ramp computation of weight
# as a function of the gust front delta v in m/s. Used in Function
# interesting-delta-v.
set fuzzyInterestingDVDV        { 0.0   10.0 }

# A list of 6 numbers defining an s-plateau computation of weight
# as a function of the distance in km between the radar site and the gust
# front segment. Used in Function estimate-max-delta-v-by-shear.
set fuzzyMaxDeltaVByShearDist   { 0.0   1.0   3.0  10.0  20.0  60.0 }

# A list of 3 numbers defining a falling-s computation of weight
# as a function of the standard error of velocity values in m/s.
# Used in Function estimate-max-delta-v-by-shear.
set fuzzyMaxDeltaVByShearError  { 1.0  2.0  10.0 }

# A list of 3 numbers defining a rising-s computation of weight as
# a function of the propagation speed of a front in m/s. Used in
# Function get-normal-pixel-wind-speed-est.
set fuzzyNormalEstSpeed         { 0.0  1.0  2.0 }

# A list of 2 numbers defining a rising-ramp computation of weight as
# a function of the ambient wind magnitude in m/s.  Used to desensitize
# certain detectors in the event of strong ambient winds.
set fuzzyVelWeightScaling       { 7.5  15.0 }

# A list of 3 numbers defining a rising-s computation of weight as
# a function of the gust front delta v in m/s. Used in function
# GFUninterestingDeltaV.
set fuzzyUninterestingDVDV      { 1.0  2.0  4.0 }

# A list of 2 numbers defining a falling-ramp computation of weight
# as a function of the estimated radar site wind velocity in m/s.
# Used in Function GFUninterestingDeltaV.
```

```
set fuzzyUninterestingDVSiteV    { 2.0  10.0 }

# A list of 2 numbers defining a rising-ramp computation of weight
# as a function of the propagation speed of a gust front in m/s.
# Used in Function GFUninterestingDeltaV.
set fuzzyUninterestingDVSpeed    { 0.0  3.0 }

# The minimum interest score allowed for a gust front to be included
# in MIGFA-gust-fronts. Used in functions ChasingOtherChains and
# CollectBestChains.
set gfMinScore              3000

# The distance in km around a wind sensor at which a gust front
# begins to affect measurements.
set gfImpactRadius          6.0

# The (one-sided) distance defining the local neighborhood when looking for
# direction consensus. Used in Function FlipDirectionsToNearestIndexed().
set indexedSearchLimit      32

# The interest value threshold applied to interest images to
# distinguish features from background.
set interestThreshold       127

# The window size in meters used to compute convergence values in
# GFBaseImages.conv.
set convergeWindowSize      1000
#set convergeWindowSize      2000

# The window size in meters used to compute convergence values in
# GFBaseImages.lowResConv.
set lowResConvWindowSize    3000
#set lowResConvWindowSize    2000

# The maximum change in distance moved in km allowed for current point
# relative to its corresponding point in the prior volume scan. Used
# in Function ComputePointAttributes().
set maxChangeInDistanceMovedKM    1.5

#  Largest index into depthWeightTable array used in wind estimation
#  routines
set maxDepthWeightTableIndex    3

# The maximum distance in kms a gust front point can move from volume scan
# to volume scan and still be indexed. Used in Function
# ComputePointAttributes()
set maxEventDistanceMovedKM     5.0

# The maximum one-dimensional distance in kms between a current gust front
# point and a prior point.  Used in Function
```

```
# FindCorrespondingPointInHistory().
set maxEventOneDimDistanceMovedKM    5.0

# The maximum change in orientation in degrees between a current point and
# corresponding point in the prior volume scan. Used in functions
# ComputePointAttributes() and FindCorrespondingPointInHistory().
set maxOrientationChange          21

# The maximum percent change in distance moved allowed for current
# point relative to its corresponding point in the prior volume scan.
# Used in Function ComputePointAttributes().
set maxPctChangeInDistanceMoved     1.0

# The maximum gap in a gust front prediction chain which may be filled
# in by a straight line of pixels. These gaps can occur, for example,
# in outwardly-expanding gust fronts from microbursts.
set maxPointGap                10

# The minimum deviation in kms of a gust front point distance moved
# relative to the other points in the gust front. Used in Function
# editGustFronts.
set maxStdDevsMoved              1.5

# The maximum weight assigned to wind estimates generated by wind
# estimation functions.
set maxWeightAheadByOE           1.0
set maxWeightAheadByHistory      1.0
set maxWeightAheadByAnemometer     1.0
set maxWeightAheadBySiteWind     1.0
set maxWeightAheadByTwind        0.0
set maxWeightBehindByHistory     1.0
set maxWeightBehindByAnemometer    1.0
set maxWeightBehindByNormal      1.0
set maxWeightBehindByOE          1.0
set maxWeightBehindBySiteWind     1.0
set maxWeightBehindByTwind       0.0
set maxWeightDeltaVByHistory     1.0
set maxWeightDeltaVByShear       0.1
set maxWeightDeltaVByVectorDiff    1.0
set maxWeightInsideByHistory     1.0
set maxWeightInsideByNormal      1.0
set maxWeightInsideByOE          1.0
set maxWeightInsideByAnemometer    1.0
set maxWeightInsideBySiteWind     1.0
set maxWeightInsideByTwind       0.0
set maxWeightMaxDeltaVByHistory    1.0
set maxWeightMaxDeltaVByNormal     1.0
set maxWeightMaxDeltaVInSegment    1.0
set maxWeightMaxDeltaVByShear      0.1
```

```
# The minimum depth needed in the absence of sufficient distance moved to
# compute propagation speed for a gust front point. Used in function
# ComputePointAttributes.
set minCorrespondenceDepth        3

# The minimum confidence factor required to trust delta v estimates.
# Used in Functions InterestingDeltaV, UninterestingDeltaV, Required
# ChainDepth and makeReportDecision
set minDeltaVWeight               0.0075

# The minimum distance moved in km in the absence of sufficient depth needed
# to compute propagation speed for a gust front point. Used in Function
# ComputePointAttributes.
set minDistanceMovedKM            3.0

# The minimum number of indexed points required for a chain to be
# reported. Used in the heuristics Function IndexedChainLengthTooSmall.
set minIndexedChainLength         11

# The minimum average wind speed estimates using the ``by-normal"
# technique (front direction of motion assumed to be normal to thin
# line orientation). Used in Functions EstimateBehindByNormal and
# and EstimateInsideByNormal
set minNormalSpeed                2.57

# The minimum length in pixels of an extracted raw binary region
# necessary to be considered as a gust front candidate. Used in
# Function EliminatePoorShapes.
set minRawShapeLength             12.5

# Minimum depth used in MakeReport Decision
set minReportDepth                1

# Minimum chain length in pixels used in MakeReportDecision
set minReportLength     16

# Minimum depth allowed for points to be included in forecast generation.
set minSDPredictionDepth          0

# For now, parameter to indicate how many gust front chains to display.
# This parameter will be removed after initial testing of prediction
# routines.
set numChainsToDisplay            5

# Number of distinct times for predictions.  Note that this value must
# be in agrrement with the array size (and number of valid values) for
# the parameter "predictionTimes" shown later in this file.
set numPredictionTimes            36

# Interest reduction factors for detectors which will suppress
```

```
# radially aligned interest which coincides with out-of-trip weather
# echoes.  Interest scores at such pixels are multiplied by the
# corresponding reduction factor.
set ootReductionFactorDZMotion      0.70
set ootReductionFactorTLDZ          0.75
set ootReductionFactorTLDZV         0.0

# The tolerance in degrees for determining if an interest value
# orientation is aligned with an out-of-trip signal. Used in
# Function suppress-oot-aligned-interest.
set outOfTripAngleTolerance         20

# Percentile value (expressed as a decimal fraction from 0.0 to 1.0)
# of sorted wind shear values used as an estimate of delta V. Used
# in Function GFDeltaVEstByShear::RunEstmtr()
set percentileDeltaVByShear         0.75

# Percentile value (expressed as a decimal fraction from 0.0 to 1.0)
# of sorted wind shear values used as an estimate of delta V. Used
# in Function estimateMaxDeltaVByNormal.
set percentileMaxDeltaVByNormal     0.8

# Percentile value (expressed as a decimal fraction from 0.0 to 1.0)
# of sorted wind shear values used as an estimate of delta V. Used
# in Function GFMaxDeltaVEstByShear::RunEstmtr()
set percentileMaxDeltaVByShear      0.8

# Boolean indicating whether predictions should be generated for all
# predicted chains.
set predictAllChains                0

# A list of prediction times for displayed fronts.
set predictionTimes {
 0  1  2  3  4  5  6  7  8  9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35
}

# A scalar value that specifies how many seconds ago to look in the image
# history for the prior image.
set priorInterval                   240

# A scalar value that specifies the window half width used in determining
# whether an image from the history list is to be used as a prior image.
set priorIntervalTolerance          60

# A scalar value that specifies the maximum number of velocity
# samples used in computeRegionalWind.
set regionalWindMaxNumSamples       64
```

```
# A scalar value that specifies the spacing(in pixels) between
# velocity samples used in computeRegionalWind.
set regionalWindSampleSpacing      4

# A list of two numbers defining the mapping of distance moved values
# (in pixels) to the integer range [0,255] in the Function
# DetectAzShear::RunDetector(). Not used in WSP
set scaleBoundsAzShear             { 0.0  25.0 }

# A list of two numbers defining the mapping of distance moved values
# (in pixels) to the integer range [0,255] in the Function
# GFDetectBreaksByDistance.
set scaleBoundsBreaksByDist        { 0  0.43 }

# A list of two numbers defining the mapping of speed values (in
# pixels/sec) to the integer range [0,255] in the Function
# GFDetectBreaksBySpeed.
set scaleBoundsBreaksBySpeed       { 0  0.002 }

# Lists of two numbers defining the mapping of convergence values to the
# integer range [0,255] used in various detectors.
set scaleBoundsConvCellConv        { 0  1.5 }
set scaleBoundsConvCellConvMotion  { 0  2 }
set scaleBoundsConvDZConv0         { 0  0.5 };# Not used in WSP
set scaleBoundsConvDZConv1         { 0  1.0 };# Not used in WSP
set scaleBoundsConvDZConv2         { 0  2.0 };# Not used in WSP
set scaleBoundsConvDZConvMotion0   { 0  0.5 };# Not used in WSP
set scaleBoundsConvDZConvMotion1   { 0  1.0 };# Not used in WSP
set scaleBoundsConvDZConvMotion2   { 0  2.0 };# Not used in WSP
set scaleBoundsConvHighConv        { 0  15.0 };# Not used in WSP
set scaleBoundsConvNonCellConv     { 0  4 };# Not used in WSP
set scaleBoundsConvNonCellConvMotion  { 0  2 };# Not used in WSP
set scaleBoundsDZ                  { -20  107 };# Not used in WSP
#set scaleBoundsLowResConv         { 0  12 }
set scaleBoundsLowResConv          { 0  8 }
set scaleBoundsVTLDZV              { -0.5  0.5 };# Not used in WSP

# The time in seconds, used as processing interval to go backward in
# time in such increments. Used in  Function EstimateImpactDeltaTime
# which is called by EstimateAheadByCenter which is called from
# gfEstimateAheadByAnemometer
set scanInterval                   120

# The smallest number of contiguous non-zero depth points at the
# end of a chain to be included in the chain.  Used in Function
# PredictFrontAtTime.
set smallestTerminalSubchainLength    4

# The window size over which gust front point attributes "distanceMoved"
```

```
# and "speed" are smoothed in Function DoChainSmoothing.  Must be an odd
# integer.
set smoothAttributeWindowSize        11

# The window size over which gust front point directions are smoothed in
# Function DoChainSmoothing.  Must be an odd integer.
set smoothDirectionWindowSize        5

# The SNR (reflectivity for WSP) threshold below which velocity data is rejected.
set snThreshold                      -5.0

# Months for summer operations  (WSP only)
set summerMonths                     { 4  1  9  30 }

# Confirming and disconfirming weights assigned to the
# interest images generated by the break detectors.
set weightsBreaksByDirection         { 4.0  1.0 }
set weightsBreaksByDistance          { 1.0  1.0 }
set weightsBreaksBySpeed             { 1.0  1.0 }
set weightsBreaksByExtAngle          { 2.0  2.0 }

# A scaling factor used to adjust the wind shift estimate just prior
# to output.
set windBehindBias                   1.0

# A scale factor used in the calculation of the largest wind shift
# value allowed for a given gust front as a function of propagation
# speed. Used by function GFWindEstByNormal::RunEstmtrNormalBI
set windBehindLimitFactor            1.4

# An offset used in the calculation of the largest wind shift value
# allowed for a given gust front as a function of propagation speed.
# Used by function GFWindEstByNormal::RunEstmtrNormalBI
set windBehindLimitOffset            7.5

# The half lenght of the segment used for estimating the winds.  Used in
# function GFChainCreateWindSegment.
set windEstSegmentHalfLength         12

# A low weight used in calculations of wind estimate weights. Used
# in various wind estimation functions.
set windLowWeight                    0.1

# The distance in km ahead of gust front points where velocity values are
# sampled. Used in Function GFMaxDeltaVNormal::Run.
set windSamplingDistAheadKM          4.0

# The time in seconds, used together with the estimated propagation speed
# of a gust front, to determine how far behind a gust front sample Doppler
# values are to be collected for wind analysis.  Used in Function
```

```
# GFWindEstByNormal::RunBehind
set windSamplingTimeBehindSec       0

# The time in seconds used to compute the distance behind a front where
# Terminal Winds velocity estimates are retrieved. Used by class
# GFWindEstByTWIND.
set windSamplingTimeTwind       1200

#
# Following is a lookup table that returns a background interest value for
# given precip numbers. Used to intialize the SKArray<short> precipBiasTable
# member variable of GFIntGenConfig structure.
#
# A "WET" environment table:
#
set precipBiasTableData {
 "80  80  80  80  80  80  80  80  80  80  80"
 "80  80  80  80  80  80  80  80  80  80  80"
 "80  80  80  80  80  80  80  80  80  80  64"
 "80  80  80  80  80  80  80  80  80  64  48"
 "80  80  80  80  80  80  80  80  80  64  48"
 "80  80  80  80  80  80  80  80  64  48  48"
 "80  80  80  80  80  80  80  64  64  48  48"
 "80  80  80  80  80  80  64  64  48  48  32"
 "80  80  80  80  80  80  64  48  48  48  32"
 "80  80  80  80  64  64  48  48  48  32  32"
 "64  64  64  64  48  48  48  32  16   0   0"
}

#
# Integer reflectivity bounds in dBZ used to tally pixels of stratiform
# rain and storm cells.
#
set stratRainDZ         { 5  25 }
set stormCellDZ         { 30 255 }

##############################################################################
#
# End
#
##############################################################################
```

A.3. GF_DET_PARM_FILE_WSP

```
##############################################################################
#
#       File: detectorParamsWSP
#
#       Description:
#
#       Parameter settings for the various feature detectors employed
#     by the WSP MIGFA algorithm
#
#       Note: Detector weight settings are grouped together at the end of
#     this file for ease of tuning.
#
#       cvs id is: "@(#)$Id: detectorParamsWSP,v 1.12 2001/03/10 16:00:29 setht Exp $ MIT/LL"
#
##############################################################################
#

#
# Global steering wind-related parameters used by multiple detectors
#
set steeringWindSectorWidth     80
set vadBias                 1.0
set vadConfidenceRamp       0.4

set DZMotion(SteeringWindSectorBias)     0.25
set TLDZ(SteeringWindSectorBias)        0.25
set TLSD(SteeringWindSectorBias)        1.00
set SDMotion(SteeringWindSectorBias)     1.00
set DZSDMotion(SteeringWindSectorBias)    1.00

#
# Line storm boosting parameters:
#
set LineStormAngleTolerance          45

set Converge(LineStormBoostFactor)     1.5
set ConvMotion(LineStormBoostFactor)    1.5
set DZMotion(LineStormBoostFactor)     1.5
set DZSDMotion(LineStormBoostFactor)    1.5
set SDMotion(LineStormBoostFactor)     1.5
set TLDZ(LineStormBoostFactor)         1.5
set TLSD(LineStormBoostFactor)         1.5
set TLSDDZ(LineStormBoostFactor)       1.5


#
# List of all "detectors" that can be run, and the order in which they will
# be run. The order is important because some detectors require the output of
```

```
# other detectors.
#
set Detectors { LineStorm Anticipation SimpleDZFilter StormCells StratRain \
StratRainExpanded DZMotion HIDZ TLDZ TLSD SDMotion TLSDDZ \
Converge ConvMotion DZSDMotion OutOfTrip PrecipEdges ZeroCross SmoothedInt }


##########################################################################
#
# Line Storm Detector Parameters
#
##########################################################################

set LineStorm(CMAP)          INTEREST
set LineStorm(DebugImages)    0
set LineStorm(HistorySize)    0
set LineStorm(HistoryWeights)  {0.0}
set LineStorm(WeightFunc)     "(0 0.5) (127 0.5) (128 1.0) (32767 1.0)"
set LineStorm(Angles)         "0 20 40 60 80 100 120 140 160"

set LineStorm(FuncTable) {
"(0    255) (60   255) (70     0) (80  -255) (100  -384) (255  -384) "
"(0   -512) (80  -384) (100    0) (120  384) (140   512) (255   512) "
}
set LineStorm(KernelCenter) " 11 14 "
set LineStorm(KernelData) {
"0 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 0"
"9 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 9"
"0 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 0"
"9 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 9"
"0 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 0"
"9 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 9"
"0 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 0"
"9 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 9"
"0 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 0"
"9 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 9"
"0 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 0"
"9 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 9"
"0 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 0"
"9 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 9"
"0 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 0"
"9 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 9"
"0 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 0"
"9 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 9"
"0 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 0"
"9 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 9"
"0 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 0"
"9 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 9"
```

```
"0 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 0"
"9 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 9"
"0 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 0"
"9 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 9"
"0 9 9 9 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 0"
}


#########################################################################
#
# Anticipation Detector Parameters
#
#########################################################################

set Anticipation(CMAP)          INTEREST
set Anticipation(DebugImages)    0
set Anticipation(HistorySize)    0
set Anticipation(HistoryWeights)  {0.0}
set Anticipation(LineStormShortTermAntVal)     127
set Anticipation(LineStormLongTermAntVal)      512
set Anticipation(WinterBiasFactor)          0.67
set Anticipation(WinterBiasThresh)           80

#########################################################################
#
# New positive/negative difference DZ Motion Detector Parameters
#
#########################################################################

set DZMotion(CMAP)          INTEREST
set DZMotion(DebugImages)    0
set DZMotion(HistorySize)    0
set DZMotion(HistoryWeights)  {0.0}
set DZMotion(BiasSector)     {90 250}
set DZMotion(SectorBias)     0.25
set DZMotion(WeightFunc)     "(0 0.0) (127 0.0) (128 1.0) (32767 1.0)"
set DZMotion(Angles) "0 20 40 60 80 100 120 140 160 180 200 220 240 260 280 300 320 340"

#
# DZMotion scoring functions:
#
#  Scoring functions 0 and 1 = region ahead of front.
#  Scoring functions 2 and 3 = leading (positive difference) edge of front.
#  Scoring functions 4 and 5 = trailing (negative difference) edge of front.
#

set DZMotion(FuncTable) {
"(0 -255) (118 -255) (123 255) (133 255) (138 -255) (255 -255)"
"(0 -382) (118 -382) (123 255) (133 255) (138 -382) (255 -382)"
```

```
"(0 -512) (128 -512) (138 512) (158 512) (168 255) (188 255) (208 0) (255 -255)"
"(0 -512) (128 -512) (138 512) (158 512) (168 255) (188 255) (208 0) (255 -512)"
"(0 -255) (48 0) (68 255) (88 255) (98 512) (118 512) (128 -512) (255 -512)"
"(0 -512) (48 0) (68 255) (88 255) (98 512) (118 512) (128 -512) (255 -512)"
}

#
# DZMotion kernels. Note that the kernel center of rotation is not in the
# middle of the kernel. Instead, it is offset so that the resulting interest
# value is assigned to the location corresponding to the leading (positive difference)
# edge of the front.
#
# Kernel 1: 5 m/s motion
#
set DZMotion(KernelCenter1) { 3 10 }
set DZMotion(KernelData1) {
"9 9 9 2 9 4"
"0 9 9 2 9 4"
"9 9 9 2 9 4"
"0 9 9 2 9 4"
"9 9 9 2 9 4"
"0 9 9 2 9 4"
"9 9 9 2 9 4"
"0 9 9 2 9 4"
"9 9 9 2 9 4"
"1 9 9 3 9 5"
"9 9 9 3 9 5"
"1 9 9 3 9 5"
"9 9 9 2 9 4"
"0 9 9 2 9 4"
"9 9 9 2 9 4"
"0 9 9 2 9 4"
"9 9 9 2 9 4"
"0 9 9 2 9 4"
"9 9 9 2 9 4"
"0 9 9 2 9 4"
"9 9 9 2 9 4"
}

#
# Kernel 2: 10 m/s motion
#
set DZMotion(KernelCenter2) { 3 10 }
set DZMotion(KernelData2) {
"9 9 9 2 9 9 9 9 4"
"0 9 9 2 9 9 9 9 4"
"9 9 9 2 9 9 9 9 4"
"0 9 9 2 9 9 9 9 4"
"9 9 9 2 9 9 9 9 4"
"0 9 9 2 9 9 9 9 4"
```

```
"9 9 9 2 9 9 9 9 4"
"0 9 9 2 9 9 9 9 4"
"9 9 9 2 9 9 9 9 4"
"1 9 9 3 9 9 9 9 5"
"9 9 9 3 9 9 9 9 5"
"1 9 9 3 9 9 9 9 5"
"9 9 9 2 9 9 9 9 4"
"0 9 9 2 9 9 9 9 4"
"9 9 9 2 9 9 9 9 4"
"0 9 9 2 9 9 9 9 4"
"9 9 9 2 9 9 9 9 4"
"0 9 9 2 9 9 9 9 4"
"9 9 9 2 9 9 9 9 4"
"0 9 9 2 9 9 9 9 4"
"9 9 9 2 9 9 9 9 4"
}

#
# Kernel 3: 15 m/s motion
#
set DZMotion(KernelCenter3) { 3 10 }
set DZMotion(KernelData3) {
"9 9 9 2 9 9 9 9 9 9 4"
"0 9 9 2 9 9 9 9 9 9 4"
"9 9 9 2 9 9 9 9 9 9 4"
"0 9 9 2 9 9 9 9 9 9 4"
"9 9 9 2 9 9 9 9 9 9 4"
"0 9 9 2 9 9 9 9 9 9 4"
"9 9 9 2 9 9 9 9 9 9 4"
"0 9 9 2 9 9 9 9 9 9 4"
"9 9 9 2 9 9 9 9 9 9 4"
"1 9 9 3 9 9 9 9 9 9 5"
"9 9 9 3 9 9 9 9 9 9 5"
"1 9 9 3 9 9 9 9 9 9 5"
"9 9 9 2 9 9 9 9 9 9 4"
"0 9 9 2 9 9 9 9 9 9 4"
"9 9 9 2 9 9 9 9 9 9 4"
"0 9 9 2 9 9 9 9 9 9 4"
"9 9 9 2 9 9 9 9 9 9 4"
"0 9 9 2 9 9 9 9 9 9 4"
"9 9 9 2 9 9 9 9 9 9 4"
"0 9 9 2 9 9 9 9 9 9 4"
"9 9 9 2 9 9 9 9 9 9 4"
}


#########################################################################
#
#  Storm Cell Detector Parameters
#
```

```
#######################################################################

set StormCells(CMAP)          INTEREST
set StormCells(DebugImages)    0
set StormCells(HistorySize)    0
set StormCells(HistoryWeights)  {0.0}
#  Dummy weight function; not used.
set StormCells(WeightFunc)     "(0 1.0) (32767 1.0)"
set StormCells(Angles)         "0"

set StormCells(FuncTable) {
"(0 -512) (80 -128) (120 256) (255 256)"
}


#######################################################################
#
# Stratiform Rain Detector Parameters
#
#######################################################################

set StratRain(CMAP)          INTEREST
set StratRain(DebugImages)    0
set StratRain(HistorySize)    0
set StratRain(HistoryWeights)  {0.0}
#  Dummy weight function; not used.
set StratRain(WeightFunc)     "(0 1.0) (32767 1.0)"
set StratRain(Angles)         "0"

set StratRain(FuncTable) {
"(0   -512) (45   -256) (92   256) (255   256) "
}

#######################################################################
#
# DZ Thin-line (TLDZ) Detector Parameters
#
#######################################################################

set TLDZ(CMAP)          INTEREST
set TLDZ(DebugImages)    0
set TLDZ(HistorySize)    0
set TLDZ(HistoryWeights)  {0.0}
set TLDZ(WeightFunc)     "(0 0.5) (127 0.5) (128 1.0) (32767 1.0)"
set TLDZ(BiasSector)     {90 250}
set TLDZ(SectorBias)  0.25
set TLDZ(Angles)         "0 20 40 60 80 100 120 140 160"

set TLDZ(FuncTable) {
"(0 255) (30 255) (38 0) (55 -511) (255 -511)"
```

```
"(0 255) (30 255) (38 0) (55 -511) (100 -768) (255 -768)"
"(0 -255) (20 -255) (35 0) (45 255) (50 384) (55 384) (60 384) (80 128) (160 0) (255 0)"
"(0 -384) (20 -255) (35 0) (45 255) (50 512) (55 512) (60 384) (80 128) (160 0) (255 0)"
}

set TLDZ(KernelCenter) { 5 7 }
set TLDZ(KernelData) {
"0 9 9 9 9 2 9 9 9 9 0"
"9 9 9 9 9 2 9 9 9 9 9"
"0 9 9 9 9 2 9 9 9 9 0"
"9 9 9 9 9 2 9 9 9 9 9"
"0 9 9 9 9 2 9 9 9 9 0"
"9 9 9 9 9 2 9 9 9 9 9"
"1 9 9 9 9 3 9 9 9 9 1"
"9 9 9 9 9 3 9 9 9 9 9"
"1 9 9 9 9 3 9 9 9 9 1"
"9 9 9 9 9 2 9 9 9 9 9"
"0 9 9 9 9 2 9 9 9 9 0"
"9 9 9 9 9 2 9 9 9 9 9"
"0 9 9 9 9 2 9 9 9 9 0"
"9 9 9 9 9 2 9 9 9 9 9"
"0 9 9 9 9 2 9 9 9 9 0"
}


#######################################################################
#
#  Thin line Standard Deviation Detector Parameters
#
#######################################################################

set TLSD(CMAP)          INTEREST
set TLSD(DebugImages)    0
set TLSD(HistorySize)    0
set TLSD(HistoryWeights)  {0.0}
set TLSD(WeightFunc)     "(0 1.0) (32767 1.0)"
set TLSD(BiasSector)     {90 250}
set TLSD(SectorBias)  0.25
set TLSD(Angles)         "0 20 40 60 80 100 120 140 160"

set TLSD(FuncTable) {
"(0 -512) (100 -256) (130 0) (160 256) (255 512)"
"(0 -512) (60 -512) (100 -256) (130 0) (160 256) (255 512)"
"(0 -256) (10 256) (20 256) (25 512) (60 512) (100 256) (130 0)\
 (160 -256) (255 -512)"
"(0 -256) (10 128) (20 256) (25 512) (60 512) (100 256) (130 0)\
 (160 -384) (255 -512)"
}

set TLSD(KernelCenter) { 3 8 }
```

```
set TLSD(KernelData) {
"9 9 9 2 9 9 9"
"0 9 9 2 9 9 0"
"9 9 9 2 9 9 9"
"0 9 9 2 9 9 0"
"9 9 9 2 9 9 9"
"0 9 9 2 9 9 0"
"9 9 9 2 9 9 9"
"1 9 9 3 9 9 1"
"1 9 9 3 9 9 1"
"1 9 9 3 9 9 1"
"9 9 9 2 9 9 9"
"0 9 9 2 9 9 0"
"9 9 9 2 9 9 9"
"0 9 9 2 9 9 0"
"9 9 9 2 9 9 9"
"0 9 9 2 9 9 0"
"9 9 9 2 9 9 9"
}

#########################################################################
#
# Simple DZ Filter Parameters
#
#########################################################################

set SimpleDZFilter(CMAP)          INTEREST
set SimpleDZFilter(DebugImages)    0
set SimpleDZFilter(HistorySize)    0
set SimpleDZFilter(HistoryWeights)  {0.0}
set SimpleDZFilter(WeightFunc)     "(0 1.0) (32767 1.0)"


#########################################################################
#
# Hi Beam Reflectivity Detector Parameters
#
#########################################################################

set HIDZ(CMAP)          INTEREST
set HIDZ(DebugImages)    0
set HIDZ(HistorySize)    0
set HIDZ(HistoryWeights)  {0.0}
set HIDZ(WeightFunc)     "(0 1.0) (32767 1.0)"
set HIDZ(Angles)         "0"

set HIDZ(FuncTable) {
"(0 256) (100 256) (120 192) (123 0) (135 -256) (255 -512)"
}
```

```
#########################################################################
#
#  Standard Deviation Motion Detector Parameters
#
#########################################################################

set SDMotion(CMAP)           INTEREST
set SDMotion(DebugImages)     0
set SDMotion(HistorySize)     0
set SDMotion(HistoryWeights)  {0.0}
set SDMotion(WeightFunc)      "(0 1.0) (32767 1.0)"
set SDMotion(BiasSector)      {90 250}
set SDMotion(SectorBias)      0.25
set SDMotion(Angles)          "0 20 40 60 80 100 120 140 160"

set SDMotion(FuncTable) {
"(0 384) (1 256) (40 256) (50 0) (60 -256) (255 -512)"
"(0 512) (1 256) (40 256) (50 0) (60 -512) (255 -1024)"
"(0 -256) (40 -256) (50 0) (60 256) (70 512) (160 512) (190 255)\
 (230 -512) (255 -512)"
"(0 -512) (40 -512) (50 0) (60 512) (70 512) (160 512) (190 255)\
 (230 -512) (255 -1024)"
}

set SDMotion(KernelCenter) { 3  8 }
set SDMotion(KernelData) {
"9 9 2 2 2 9 9"
"0 9 9 2 9 9 0"
"9 9 2 2 2 9 9"
"0 9 9 2 9 9 0"
"9 9 2 2 2 9 9"
"0 9 9 2 9 9 0"
"9 9 2 2 2 9 9"
"1 9 9 3 9 9 1"
"1 9 3 3 3 9 1"
"1 9 9 3 9 9 1"
"9 9 2 2 2 9 9"
"0 9 9 2 9 9 0"
"9 9 2 2 2 9 9"
"0 9 9 2 9 9 0"
"9 9 2 2 2 9 9"
"0 9 9 2 9 9 0"
"9 9 2 2 2 9 9"
}

#########################################################################
#
# Simultaneous Thin Line Reflectivity/Standard Deviation Detector Parameters
#
```

```
#############################################################################

set TLSDDZ(CMAP)          INTEREST
set TLSDDZ(DebugImages)    0
set TLSDDZ(HistorySize)    0
set TLSDDZ(HistoryWeights)  {0.0}
set TLSDDZ(WeightFunc)     "(0 1.0) (32767 1.0)"
set TLSDDZ(Angles)         "0 20 40 60 80 100 120 140 160"

set TLSDDZ(FuncTable) {
"(0 -512) (100 -256) (130 0) (160 256) (255 512)"
"(0 -512) (60 -512) (100 -256) (130 0) (160 256) (255 512)"
"(0 -256) (10 256) (20 256) (25 512) (60 512) (100 256) (130 0)\
 (160 -256) (255 -512)"
"(0 -256) (10 128) (20 256) (25 512) (60 512) (100 256) (130 0)\
 (160 -384) (255 -512)"
"(0 -128) (10 256) (40 256) (50 0) (55 -256) (255 -256)"
"(0 -128) (10 256) (40 256) (50 0) (55 -256) (255 -256)"
}

set TLSDDZ(KernelCenter) { 3 8 }
set TLSDDZ(KernelData0) {
"9 9 9 2 9 9 9"
"0 9 9 2 9 9 0"
"9 9 9 2 9 9 9"
"0 9 9 2 9 9 0"
"9 9 9 2 9 9 9"
"0 9 9 2 9 9 0"
"9 9 9 2 9 9 9"
"1 9 9 3 9 9 1"
"1 9 9 3 9 9 1"
"1 9 9 3 9 9 1"
"9 9 9 2 9 9 9"
"0 9 9 2 9 9 0"
"9 9 9 2 9 9 9"
"0 9 9 2 9 9 0"
"9 9 9 2 9 9 9"
"0 9 9 2 9 9 0"
"9 9 9 2 9 9 9"
}

set TLSDDZ(KernelData1) {
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 5 9 9 9"
```

```
"9 9 9 5 9 9 9"
"9 9 9 5 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
}

#########################################################################
#
# Convergence Detector Parameters
#
#########################################################################

set Converge(CMAP)          INTEREST
set Converge(DebugImages)    0
set Converge(HistorySize)    0
set Converge(HistoryWeights)  {0.0}
set Converge(WeightFunc)     "(0 1.0) (32767 1.0)"
set Converge(Angles)         "0 20 40 60 80 100 120 140 160"

set Converge(FuncTable) {
"(0 256) (32 256) (64 0) (96 -256) (160 -256) (255 -512)"
"(0 256) (32 256) (64 0) (96 -256) (160 -512) (255 -512)"
"(0 -256) (32 -256) (64 0) (96 128) (160 256) (255 512)"
"(0 -512) (32 -256) (64 0) (96 128) (160 256) (255 512)"
"(0 -1024) (16 0) (32 128) (64 512) (255 512)"
}

set Converge(KernelCenter) { 5 7 }
set Converge(KernelData0) {
"0 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 0"
"0 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 0"
"0 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 0"
"1 9 9 9 9 3 9 9 9 9 1"
"9 9 9 9 9 3 9 9 9 9 9"
"1 9 9 9 9 3 9 9 9 9 1"
"9 9 9 9 9 2 9 9 9 9 0"
"0 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 0"
"0 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 0"
"0 9 9 9 9 2 9 9 9 9 9"
```

```
}

set Converge(KernelData1) {
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
}


#########################################################################
#
# Convergence Motion Detector Parameters
#
#########################################################################

set ConvMotion(CMAP)          INTEREST
set ConvMotion(DebugImages)    0
set ConvMotion(HistorySize)    0
set ConvMotion(HistoryWeights)  {0.0}
set ConvMotion(WeightFunc)     "(0 1.0) (32767 1.0)"
set ConvMotion(Angles)         "0 20 40 60 80 100 120 140 160"

set ConvMotion(FuncTable) {
"(0 256) (32 256) (64 0) (96 -256) (160 -256) (255 -512)"
"(0 256) (32 256) (64 0) (96 -256) (160 -512) (255 -512)"
"(0 -256) (32 -256) (64 0) (96 128) (160 256) (255 512)"
"(0 -512) (32 -256) (64 0) (96 128) (160 256) (255 512)"
"(0 -1024) (16 0) (32 128) (64 512) (255 512)"
}


set ConvMotion(KernelCenter) { 5 7 }
set ConvMotion(KernelData0) {
"0 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 0"
"0 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 0"
"0 9 9 9 9 2 9 9 9 9 9"
```

```
"9 9 9 9 9 2 9 9 9 9 0"
"9 1 9 9 9 3 9 9 9 1 9"
"9 9 9 9 9 3 9 9 9 9 9"
"9 1 9 9 9 3 9 9 9 1 9"
"9 9 9 9 9 2 9 9 9 9 0"
"0 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 0"
"0 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 0"
"0 9 9 9 9 2 9 9 9 9 9"
}

set ConvMotion(KernelData1) {
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
"9 9 9 9 9 4 9 9 9 9 9"
}


#########################################################################
#
#  Simultaneous Thin Line/Standard Deviation Motion Detector Parameters
#
#########################################################################

set DZSDMotion(CMAP)          INTEREST
set DZSDMotion(DebugImages)    0
set DZSDMotion(HistorySize)    0
set DZSDMotion(HistoryWeights)  {0.0}
set DZSDMotion(WeightFunc)     "(0 1.0) (32767 1.0)"
set DZSDMotion(BiasSector)     {90 250}
set DZSDMotion(SectorBias)     0.25
set DZSDMotion(Angles)         "0 20 40 60 80 100 120 140 160"

set DZSDMotion(FuncTable) {
"(0 384) (1 256) (40 256) (50 0) (60 -256) (255 -512)"
"(0 512) (1 256) (40 256) (50 0) (60 -512) (255 -1024)"
"(0 -256) (40 -256) (50 0) (60 256) (70 512) (100 512) (140 255)\
```

```
 (170 -512) (255 -512)"
"(0 -512) (40 -512) (50 0) (60 512) (70 512) (100 512) (140 255)\
 (170 -512) (255 -1024)"
"(0 128) (40 128) (50 0) (55 -128) (255 -128)"
"(0 128) (40 128) (50 0) (55 -128) (255 -128)"
}

set DZSDMotion(KernelCenter) { 3 8 }
set DZSDMotion(KernelData0) {
"9 9 9 2 9 9 9"
"0 9 9 2 9 9 0"
"9 9 9 2 9 9 9"
"0 9 9 2 9 9 0"
"9 9 9 2 9 9 9"
"0 9 9 2 9 9 0"
"9 9 9 2 9 9 9"
"1 9 9 3 9 9 1"
"1 9 9 3 9 9 1"
"1 9 9 3 9 9 1"
"9 9 9 2 9 9 9"
"0 9 9 2 9 9 0"
"9 9 9 2 9 9 9"
"0 9 9 2 9 9 0"
"9 9 9 2 9 9 9"
"0 9 9 2 9 9 0"
"9 9 9 2 9 9 9"
}

set DZSDMotion(KernelData1) {
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 5 9 9 9"
"9 9 9 5 9 9 9"
"9 9 9 5 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
"9 9 9 4 9 9 9"
}

##########################################################################
```

```
#
# Out Of Trip Detector Parameters
#
#########################################################################

set OutOfTrip(CMAP)           INTEREST
set OutOfTrip(DebugImages)    0
set OutOfTrip(HistorySize)    0
set OutOfTrip(HistoryWeights) {0.0}
set OutOfTrip(WeightFunc)     "(0 1.0) (32767 1.0)"
set OutOfTrip(Angles)         "90"

set OutOfTrip(FuncTable) {
"(0 255) (30 255) (50 128) (70 -255) (255 -255)"
"(0 -512) (30 -512) (35 0) (40 128) (45 255) (255 255)"
"(0 -512) (100 -512) (140 0) (141 nil) (255 nil)"
}

set OutOfTrip(KernelCenter) { 5 10 }
set OutOfTrip(KernelData0) {
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
"0 9 9 9 9 1 9 9 9 9 0"
}

set OutOfTrip(KernelData1) {
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
```

```
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
"9 9 9 9 9 2 9 9 9 9 9"
}


#########################################################################
#
# Expanded Stratiform Rain Detector Parameters
#
#########################################################################

set StratRainExpanded(CMAP)          INTEREST
set StratRainExpanded(DebugImages)    0
set StratRainExpanded(HistorySize)    0
set StratRainExpanded(HistoryWeights)  {0.0}
set StratRainExpanded(WeightFunc)     "(0 1.0) (32767 1.0)"
set StratRainExpanded(Angles)         "0"

set StratRainExpanded(FuncTable) {
"(0 -512) (10 -256) (35 256) (255 256)"
}


#########################################################################
#
# Precip Edges Detector Parameters
#
#########################################################################
#
#
set PrecipEdges(CMAP)          INTEREST
set PrecipEdges(DebugImages)    0
set PrecipEdges(HistorySize)    0
set PrecipEdges(HistoryWeights)  {0.0}
#  Dummy weight function; not used.
```

```
set PrecipEdges(WeightFunc)      "(0 1.0) (32767 1.0)"
set PrecipEdges(Angles)          "0 20 40 60 80 100 120 140 160 180\
                        200 220 240 260 280 300 320 340"
set PrecipEdges(FuncTable) {
"(0 nil) (39 nil) (40 0) (45 -512) (70 -512) (255 -512)"
"(0 512) (35 512) (40 0) (45 -256) (70 -256) (255 -256)"
"(0 -512) (40 -512) (50 0) (60 512) (255 512)"
"(0 -256) (50 -256) (60 0) (61 nil) (255 nil)"
}

set PrecipEdges(KernelCenter) { 5 2 }
set PrecipEdges(KernelData) {
"0 9 0 9 1 9 9 2 9 3 9 9 3 9"
"0 9 0 9 1 9 9 2 9 9 3 9 9 3"
"0 9 0 9 1 9 9 2 9 3 9 9 3 9"
"0 9 0 9 1 9 9 2 9 9 3 9 9 3"
"0 9 0 9 1 9 9 2 9 3 9 9 3 9"
}


##########################################################################
#
#  Velocity Zero-line Detector Parameters
#
##########################################################################
#
set ZeroCross(CMAP)            INTEREST
set ZeroCross(DebugImages)     0
set ZeroCross(HistorySize)     0
set ZeroCross(HistoryWeights)  {0.0}
#  Dummy weight function; not used.
set ZeroCross(WeightFunc)      "(0 1.0) (32767 1.0)"
set ZeroCross(ConfWeight)      0.0
set ZeroCross(DisConfWeight)   0.0

set ZeroCross(PolTemplateAngles) "340 0 20 160 180 200"
set ZeroCross(CarTemplateAngles) "0 30 60 90 120 150 180 210 240 270 300 330"

set ZeroCross(PolFuncTable) {
"(0 -255) (108 -255) (128   0) (148  255) (255  255)"
"(0  255) (108  255) (128   0) (148 -255) (255 -255)"
"(0  -64) (103  -64) (108  128) (148  128) (153  -64) (255  -64)"
}

set ZeroCross(CarFuncTable) {
"(0 -255) (108 -255) (128   0) (148  255) (255  255)"
"(0  255) (108  255) (128   0) (148 -255) (255 -255)"
"(0  -64) (103  -64) (108  128) (148  128) (153  -64) (255  -64)"
}
```

```
set ZeroCross(PolKernelData) {
"1 9 1 9 1 9 1 9 1 9 1 9 1 9 1"
"9 9 9 9 9 9 9 9 9 9 9 9 9 9 9"
"9 9 9 9 9 9 9 9 9 9 9 9 9 9 9"
"9 9 9 9 9 9 9 9 9 9 9 9 9 9 9"
"9 9 9 9 9 9 9 9 9 9 9 9 9 9 9"
"2 2 2 2 2 2 2 2 2 2 2 2 2 2 2"
"9 9 9 9 9 9 9 9 9 9 9 9 9 9 9"
"9 9 9 9 9 9 9 9 9 9 9 9 9 9 9"
"9 9 9 9 9 9 9 9 9 9 9 9 9 9 9"
"9 9 9 9 9 9 9 9 9 9 9 9 9 9 9"
"0 9 0 9 0 9 0 9 0 9 0 9 0 9 0"
}

set ZeroCross(CarKernelData) {
"0 9 9 9 9 2 9 9 9 9 1"
"9 9 9 9 9 2 9 9 9 9 9"
"0 9 9 9 9 2 9 9 9 9 1"
"9 9 9 9 9 2 9 9 9 9 9"
"0 9 9 9 9 2 9 9 9 9 1"
"9 9 9 9 9 2 9 9 9 9 9"
"0 9 9 9 9 2 9 9 9 9 1"
"9 9 9 9 9 2 9 9 9 9 9"
"0 9 9 9 9 2 9 9 9 9 1"
"9 9 9 9 9 2 9 9 9 9 9"
"0 9 9 9 9 2 9 9 9 9 1"
"9 9 9 9 9 2 9 9 9 9 9"
"0 9 9 9 9 2 9 9 9 9 1"
"9 9 9 9 9 2 9 9 9 9 9"
"0 9 9 9 9 2 9 9 9 9 1"
}



#########################################################################
#
# Thin-line Smoother Parameters
#
#########################################################################

set SmoothedInt(CMAP)          INTEREST
set SmoothedInt(DebugImages)    0
set SmoothedInt(HistorySize)    0
set SmoothedInt(HistoryWeights)  {0.0}
#  Dummy weight function; not used.
set SmoothedInt(WeightFunc)     "(0 1.0) (32767 1.0)"


#########################################################################
#
```

```
# Detector Weights
#
################################################################################

set Anticipation(WeightFunc) \
 "(0 1.00) (15 1.00) (16 0.75) (95 0.75) (96 0.33) (32767 0.33)"

set DZMotion(ConfWeight)       1.0
set DZMotion(DisConfWeight)    0.5

set TLDZ(ConfWeight)           1.0
set TLDZ(DisConfWeight)        0.3

set TLSD(ConfWeight)           1.2
set TLSD(DisConfWeight)        0.0

set HIDZ(ConfWeight)           0.0
set HIDZ(DisConfWeight)        4.0

set SDMotion(ConfWeight)       1.2
set SDMotion(DisConfWeight)    0.3

set TLSDDZ(ConfWeight)         1.0
set TLSDDZ(DisConfWeight)      1.0

set Converge(ConfWeight)       1.0
set Converge(DisConfWeight)    1.0

set ConvMotion(ConfWeight)     1.0
set ConvMotion(DisConfWeight)  1.0

set DZSDMotion(ConfWeight)     1.0
set DZSDMotion(DisConfWeight)  0.0

set PrecipEdges(ConfWeight)    0.0
set PrecipEdges(DisConfWeight) 4.0
```

A.4. GF_EST_PARM_FILE

```
#
#  Full set of wind estimators.  Not all methods will be applied at all
#  three relevant locations (ahead / inside / behind gust fronts)
#
set windEstimationMethods { ByNormal ByCenter BySiteWind ByOE ByTwind \
                ByHistory ByAnemometer }

#  Delta V in the wind estimation segment.
set deltaVEstimators { ByVecDiff ByShear ByHistory }

#  Delta V across the entire gust front.
set maxDeltaVEstimators { ByNormal BySegment ByShear ByHistory }

set windAheadEstimators { ByAnemometer BySiteWind ByOE ByTwind ByHistory }

set windInsideEstimators { ByNormal ByAnemometer BySiteWind ByOE ByTwind \
            ByHistory }

set windBehindEstimators { ByNormal ByAnemometer BySiteWind ByOE ByTwind \
                ByHistory }

# Heuristics Functions
set heuristicsMethod { ChainLengthSmall UninterestingDeltaV InterestingDeltaV \
          ChasingOtherChains }
```

A.5. GF_HEU_PARM_FILE

```
#
# List of all possible heuristics.  MIGFA selects from these.
#
set Heuristics { ChainLengthSmall UninterestingDeltaV InterestingDeltaV \
        ChasingOtherChains SidelobeChain ChainLagsBhdArpWinds }
```

A.6. GF_LOG_CONFIG_FILE

```
#=============================================================================
#
# File: log.conf
#
# Logging facility configuration file.
#
#
#=============================================================================

#=============================================================================
#
# Output streams for the main 4 logging classes. Output can be to
# a file and/or stdout/stderr, with different streams seperated by a
# comman. Note that stdout and stderr can have their output streams
# directed to a file also.
# The same file can be specified for multiple logging classes.
# To discard all messages for a given logging class, specify /dev/null
# as the output stream.
#
#=============================================================================

info   /ll/sw/wsp2/logs/%T.migfa.log
warn   /ll/sw/wsp2/logs/%T.migfa.log
err    /ll/sw/wsp2/logs/%T.migfa.log
debug  /ll/sw/wsp2/logs/%T.migfa.log


#=============================================================================
#
# stdout/stderr can be redirected to a logging file (including one of
# the ones specified above). This is handy if libraries external to
# an application make use of stdout/stderr - the output will be trapped
# in the logfile. stdout/stderr may also be redirected to /dev/null
#
#=============================================================================
#

stdout  /ll/sw/wsp2/logs/%T.migfa.log
stderr  /ll/sw/wsp2/logs/%T.migfa.log


#=============================================================================
#
# Debug message control. Allows specification of debug level and scope
#
# Debugging level can range from 1 to 4, with 4 being the most detailed
# (more output). The default level is 1.
#
# If debugging enabled by specifying output stream other than /dev/null,
# debugging messages from all modules in the application will be output
# by default (global scope). The output can be made more selective using
# two commands (in combination or individually)
```

```
#
#   debugEnableFile  <file> [startLine] [stopLine]
#
#   debugEnableFunction  <functionName>
#
# If [startLine] and [stopLine] are not specified for the file version,
# debugging messages are enabled for the entire file.  Up to 20 of each
# 'rule' can be specified at one time.
#
# The matching process is a simple substring match for both the
# <file> and <functionName> fields, so, for example, a function name of
# "Gust" would match any function prototype containing the word "Gust"
# (In the case of C++, even if the "Gust" string is part of one of the
# function arguments)
#
#=============================================================================

debuglevel      2

#debugEnableFile     logtest1

# Enable debugging for all member functions of class 'Dummy'
#debugEnableFunction  Dummy::


#=============================================================================
#
# To save backup copies of logfiles across multiple program runs, specify
# a non-zero value for numBackups
#
#=============================================================================

numBackups      20


#=============================================================================
#
# Daily backup setting.  Setting this value to 'true' or 'yes' allows
# separate logfiles to be maintained for each 24-hour period.  The
# current file becomes a backup file, which is kept around until the
# 'numBackups' value is exceeded.
#
# The default is to do the switchover at 00:00:00 local time. If the
# additional argument 'GMT' is supplied, the switchover will occur at
# 00:00:00 GMT
#
# If no 'dailyBackup' setting is specified (commented out or missing),
# creation of backup files is controlled solely by 'maxHistorySize'.
#
#=============================================================================

dailyBackup true
#dailyBackup true GMT
```

```
#=============================================================================
#
# Max history size in bytes.  If logging output exceeds this amount,
# the current file is made into a backup (number of backups controlled
# by above parameter) and a new file is started.
#
#=============================================================================

maxHistorySize   7500000
```

Appendix B: MIGFA OUTPUT RECORD FORMATS

At the end of each iteration of MIGFA processing, some or all of the following records are transmitted in the following sequence:

```
FRAME_START
        SKARRAY (if image archiving requested))
        GF_TILT_HEADER
                GF_DETECTION (DETECT_ID = 1)
                        GF_FORECAST (DELTA_T = 1 minute)
                        GF_FORECAST (DELTA_T = 2 minutes)
                                .
                                .
                        GF_FORECAST (DELTA_T = 35 minutes)
                GF_DETECTION (DETECT_ID = 2)
                        GF_FORECAST (DELTA_T = 1 minute)
                        GF_FORECAST (DELTA_T = 2 minutes)
                        .
                        .
                .
                .
        GF_TILT_TRAILER
FRAME_END
```

FRAME_START and FRAME_END are always transmitted to the optional MIGFA archival stream ("-a" command-line option), but are only transmitted to the product output stream (the one specified by the "-o" option) if diagnostic image output was specified via the "-i" command-line option. Images are ouput in separate SKARRAY records.

If no gust fronts are detected for an iteration of MIGFA processing, then no GF_DETECTION or GF_FORECAST records will be transmitted. The following tables describe the format of each type of record.

Data are packed in big-endian byte order (most significant byte at lowest address offset).

FRAME_START Record Format	
Word # (2 byte)	**Description**
0	Record ID (1470)
1	Record length in 2-byte words (14)
2-3	Sequence #
4-5	Frame type
6-7	Frame sequence number
8-13	Data start time (month, day, year, hour, minute, second)

SKARRAY Image Record Format	
Word # (2 byte)	**Description**
0	Record ID: SKARRAY_CHAR = 14000, SKARRAY_SHORT = 14001, SKARRAY_INT = 14002, SKARRAY_FLOAT = 14003, SKARRAY_DOUBLE = 14004
1	Record length in 2-byte words (variable)
2-3	Sequence #
4-5	# of dimensions for array (e.g., 1, 2, 3)
6-7	Total number of array elements (nElem)
8-9	Element size (bytes)
10-11	Slice size in X dimension (xsize)
12-13	Slice size in Y dimension (ysize)
14-15	Slice size in Z dimension (zsize)
16-17	Confirming weight factor
18-19	Disconfirmgin weight factor

SKARRAY Image Record Format	
Word # (2 byte)	**Description**
20-51	Image name (e.g., VCLEAN)
52-83	Image data class (e.g., DZ, V, INTEREST)
84	Image ID
85-90	Image time (month, day, year, hour, minute, second)
91-92	Coordinate system (0=SK_CARTESIAN, 1=SK_POLAR)
93-94	Image bin size (resolution) in X dimension (meters)
95-96	Image bin size (resolution) in Y dimension (meters or degrees)
97-98	Image bin size (resolution) in Z dimension (meters or degrees)
99-100	Orientation (degrees from true north)
101-102	Image value scale factor
103-104	Image value offset
105-108	X location of reference point in image coordinates
109-112	Y location of reference point in image coordinates
113-116	Z location of reference point in image coordinates
117-120	X location of reference point in world coordinates
121-124	Y location of reference point in world coordinates
125-128	Z location of reference point in world coordinates
129-131	Reference latitude (degrees)
132-133	Reference longitude (degrees)
134-135	Reference altitude (meters)
136- 136 + (nElements x bytesPerElement / 2)	Image values sequentially ordered from bottom to top of the image with origin (0,0) at lower left corner for a 2-D array). Data are ordered in column-ordered matrix format ('x' varies first, followed by 'y', then 'z')

GF_TILT_HEADER Record Format	
Word # (2 byte)	**Description**
0	Record ID (1001)
1	Record length in 2-byte words (51)
2-3	Sequence #
4	Tilt #
5	Elevation angle (deg x 10)
6-11	Data start time (month, day, year, hour, minute, second)
12	Local time zone offset (minutes behind UT) - (not used)
13-20	Radar name (not used)
21-28	Site name (not used)
29-36	Project name (not used)
37-42	Data write time (month, day, year, hour, minute, second)
43-44	Latitude of radar (deg north x 100000) - (not used)
45-46	Longitude of radar (deg east x 100000) - (not used)
47	Altitude of radar (meters above sea level) - (not used)
48	Nyquist velocity (m/s x 100)
49	Radar PRF (Hz)
50	"Bad value" used to identify missing or bad data

GF_DETECTION Record Format	
Word # (2 byte)	**Description**
0	Record ID (2901)
1	Record length in 2-byte words (variable)
2-3	Sequence #
4-9	Data start time (month, day, year, hour, minute, second)
10	U (west) component of gust front propagation speed (m/s x 100)
11	V (north) component of gust front propagation speed (m/s x 100)
12	U (west) component of winds ahead of gust front (m/s x 100)
13	V (north) component of winds ahead of gust front (m/s x 100)
14-15	Not used
16	U (west) component of winds behind gust front (m/s x 100)
17	V (north) component of winds behind gust front (m/s x 100)
18	X-coordinate of wind shift analysis point (km from radar x 100)
19	Y-coordinate of wind shift analysis point (km from radar x 100)
20	Orientation of gust front (deg x 10)
21-22	Detection ID #
23	Event ID # (same as detection ID)
24-25	Not used
26	# of points in gust front detection curve (npts)
27-(26+npts)	X-coordinates of gust front points (km from radar x 100)
27+npts	# of points in gust front detection curve
28 + npts - (27+2*npts)	Y-coordinates of gust front points (km from radar x 100)
28+2*npts	Delta-V (m/s x 100)

GF_FORECAST Record Format	
Word # (2 byte)	**Description**
0	Record ID (2905)
1	Record length in 2-byte words (variable)
2-3	Sequence #
4-9	Data start time (month, day, year, hour, minute, second)
10	Forecast delta-T (minutes)
11	Not used
12-13	Detection ID #
14-15	Event ID # (same as Detection ID)
16	# of points in gust front forecast curve (npts)
17-(16+npts)	X-coordinates of gust front forecast points (km from radar x 100)
17+npts	# of points in gust front forecast curve
18 + npts - (17+2*npts)	Y-coordinates of gust front forecast points (km from radar x 100)

GF_TILT_TRAILER Record Format	
Word # (2 byte)	**Description**
0	Record ID (2090)
1	Record length in 2-byte words (11)
2-3	Sequence #
4	Radar tilt #
5-10	Output write time (month, day, year, hour, minute, second)

FRAME_END Record Format	
Word # (2 byte)	**Description**
0	Record ID (1417)
1	Record length in 2-byte words (14)
2-3	Sequence #
4-5	Frame type
6-7	Frame sequence #
8-13	Data start time (month, day, year, hour, minute, second)

Appendix C: MIGFA DATA CLASSES

This appendix contains descriptions for the C++ classes utilized by MIGFA. The class descriptions are listed in alphabetical order.

class GFBaseImages

Name

class GFBaseImages

Synopsis

#include <gfBaseImages.h>

Hierarchy

GFBaseImages

Description

Class to hold all of the base images needed for processing. More members are present than are needed, but this allows a single class to be used for multiple radars. All members are set to NULL initially and space is allocated for them as necessary. Only those members which have space allocated for them will be deleted.

Constructors

GFBaseImages();
Default constructor. Initializes all pointers to NULL.

Destructors

~GFBaseImages();
The GFBaseImages destructor checks to see which member pointers are not NULL and deletes those which are not.

Public data members

*SKArray<short> *dz;*
*SKArray<short> *vRa*
*SKArray<short> *vClean;*
*SKArray<short> *dbv;*
*SKArray<short> *sn;*
*SKArray<short> *apMap;*
*SKArray<short> *clutterMap;*
*SKArray<short> *dzPol;*
*SKArray<short> *lbdzPol;*
*SKArray<short> *lbvPol;*
*SKArray<short> *hbdzPol;*
*SKArray<short> *dbdzPol;*
*SKArray<short> *vPolRaw;*
*SKArray<short> *dbvPol;*
*SKArray<short> *snPol;*
*SKArray<short> *vPolClean;*
*SKArray<short> *flagsPol;*
*SKArray<short> *dzPrior;*
*SKArray<short> *vPrior;*
*SKArray<short> *convPrior;*
*SKArray<short> *sdPrior;*
*SKArray<short> *azShear;*
*SKArray<short> *conv;*
*SKArray<short> *convD77;*
*SKArray<short> *lowResConv;*
*SKArray<short> *dzDiff;*

*SKArray<short> *outOfTrip;*
*SKArray<short> *sd;*
*SKArray<short> *flags;*
*SKArray<short> *stormCells;*
*SKArray<short> *stormCellsSuppressClutter;*
*SKArray<short> *stormCellsD77;*
*SKArray<short> *stratRain;*
*SKArray<short> *stratRainD77;*
*SKArray<short> *stratRainExpanded;*
*SKArray<short> *sparseV;*
*SKArray<short> *zeroCross;*
*SKArray<short> *sdPol;*
*SKArray<short> *outOfTripPol;*
*SKArray<short> *dataMask;*
*SKArray<short> *detMask;*
*SKArray<short> *detSiteMask;*
*SKArray<short> *siteMask;*
*SKArray<short> *eightKmMask;*
*SKArray<short> *rawInterest;*
*SKArray<short> *smoothedInterest;*
*SKArray<short> *baseOrient;*
*SKArray<short> *anticipation;*
*SKArray<short> *velAlias;*

See Also Library skarr, class SKArray

Document Revision Date 10 December, 1998

class GFBaseImagesWSP

Name

class GFBaseImagesWSP

Synopsis

#include <gfBaseImages.h>

Hierarchy

GFBaseImages->GFBaseImagesWSP

Description

Class to hold WSP-specific base images needed for processing.

Constructors

GFBaseImagesWSP();
Default constructor. Initializes all pointers to NULL.

Destructors

~GFBaseImagesWSP();
Checks to see which member pointers are not NULL and deletes those which are not.

Public data members

*SKArray<short> *dzLongPol;*
Long range polar reflectivity image

*SKArray<short> *lbvLongPol;*
Long range low-beam velocity image

*SKArray<short> *flagsLongPol;*
Long range WSP data quality flags image

*SKArray<short> *dzLong;*
Long range Cartesian reflectivity image

*SKArray<short> *lbvLong;*
Long range Cartesian low-beam velocity image

*SKArray<short> *flagsLong;*
Long range Cartesian data quality flags image

*SKArray<short> *clutterMapLong;*
Long range Cartesian clutter map image

*SKArray<short> *outOfTripLong;*
Long range Cartesian out-of-trip weather image

*SKArray<short> *sdLongPol;*
Long range polar velocity standard deviation image

*SKArray<short> *lbvGoodSnrLong;*
Cleaned long range Cartesian low-beam velocity image

class GFBaseImagesWSP

See Also Library skarr, class SKArray

Document Revision Date 10 December, 1998

class GFChain

Name

class GFChain

Synopsis

#include <gfChain.h>

Hierarchy

SKChain->GFChain

Description

Class for holding information about a gust front detection

Constructors

GFChain();
Default constructor. Sets *display* and *forecastTimeInMinutes* to zero, and all other member variables of type T to be equal to *SKArray<T>::SK_NIL*. All pointers are set to point to NULL.

Destructors

~GFChain()
Default destructor. Deletes all GFPoints which comprise the gust front detection curve, all non NULL pointers and all wind data LLDLists.

Public data members

short *number;*
An arbitrary number assigned to a GF for bookkeeping.

short *ndxInPointArray;*
Index of the chain start point in the gfPointArray member of the current event object.

short *length;*
Number of points in chain.

short *windSegStartNdx;*
Index of first point of subchain (relative to start of chain) over which wind estimates are made.

short *windSegMidNdx;*
Index of midpoint of subchain (relative to start of chain) over which wind estimates are made.

short *windSegLastNdx;*
Index of last point of subchain (relative to start of chain) over which wind estimates are made.

short *windSegLength;*
Number of points in the wind estimation segment.

float *windSegAvgDepth;*
Average tracking depth of points "in the wind estimation segment". Actually, what is stored is the 75th percentile of depth (e.g. a depth higher than 75% of all depths among points in the wind estimation segment).

float *windSegAvgDirection;*
Average direction of motion for points "in the wind estimation segment".

float *windSegAvgSpeed;*
Average speed for points "in the wind estimation segment".

GFPoint **nearestToArp;*
Pointer to chain point nearest to the Airport Reference Point (ARP).

GFPoint **windSegFirstPoint;*
Pointer to the first GFPoint in the wind estimation segment.

GFPoint **windSegMidPoint;*
Pointer to the middle GFPoint in the wind estimation segment.

GFPoint **windSegLastPoint;*
Pointer to the last GFPoint in the wind estimation segment.

GFWind **windAhead;*
Pointer to GFWind object containing consensus estimate of winds ahead of the front.

GFWind **windInside;*
Pointer to GFWind object containing consensus estimate of winds "inside" the front.

GFWind **windBehind;*
Pointer to GFWind object containing consensus estimate of winds behind the front (wind shift).

LLDList *windAheadData;*
List of GFWind objects containing individual estimates of winds ahead of the front from the different wind estimators.

LLDList *windInsideData;*
List of GFWind objects containing individual estimates of winds inside the front from the different wind estimators.

LLDList windBehindData;
List of GFWind objects containing individual estimates of winds behind the front from the different wind estimators.

*GFWind *deltaV;*
Consensus delta-V estimate for the front. This quantity is the one that is reported for the gust front.

*GFWind *maxDeltaV;*
Consensus maximum delta-v for the front. Used by various heuristics.

LLDList deltaVData;
List of GFWind objects containing the individual estimates of delta-V from the different delta-V estimators.

LLDList maxDeltaVData;
List of GFWind objects containing the individual estimates of maximum delta-V from the different maximum delta-V estimators.

SKCoordI windShiftArrowPos;
Image coordinates of the center of the wind estimation segment that is the basis of the anchor location for the wind shift arrow.

float avgDepth;
Average tracking depth for the entire chain.

float avgScore;
Average score of GFPoints in the chain.

short minScore;
Minimum score of GFPoints in the chain.

short display;
Flag to indicate whether or not to display this chain (and / or any associated predictions) (0 = don't display, 1 = do display).

LLDList predictions;
List of GFPredChain objects containing incremental position forecasts.

short forecastTimeInMinutes;
Valid "forecast time" for the chain. A current detection chain has a forecast time of zero. When a GFChain is used to encode a predection chain, the forecast time will be a 1-minute interval.

Related global functions

void DeleteGFChains(LLDList& chains);
Iterates through LLDList of GFChain objects, deleting each GFChain object.

See Also

class *SKChain* (CSKETCH library), class GFPoint, class GFPredChain

Document Revision Date

15 February, 2002

class GFDeltaVEstBase

Name

class GFDeltaVEstBase

Synopsis

#include <gfDeltaVEstBase.h>

Hierarchy

GFDeltaVEstBase

Description

Base class for the delta v estimators. All delta v estimators are derived off of this class. This is done so that an LLNIDList of pointers to the estimators can be formed by using pointers of type *GFDeltaVEstBase*.

Component Structures

struct GFDeltaVEstParams
{
char name[256];
};
Structure to hold the name of the delta v estimator.

Constructors

*GFDeltaVEstBase(char *, TclObject &);*
Default constructor. Copies the name of the estimator into the GFDeltaVEstParams structure and reads in the parameters.

Destructors

virtual ~GFDeltaVEstBase();
Default destructor. Currently a no-op.

Public virtual functions

virtual GFDeltaV RunEstmtr(GFChain *) = 0;*
Function defined in a derived class used to estimate the delta v across a chain.

Public member functions

GFDeltaV Run(GFChain *);*
Calls *RunEstmtr()* defined in the derived class. Sets the v and vwt components of the GFDeltaV* to 0.0 for all delta v estimators.

See Also

Library gfext, class GFChain.
Library baseobj, class TclObject;

Document Revision Date

10 December, 1998

class GFDetectorBase

Name

class GFDetectorBase

Synopsis

#include <gfDetectorBase.h>

Hierarchy

GFDetectorBase

Description

Base class for the detectors. All detectors are derived off of this class. This is done so that an LLNIDList of pointers to the detectors can be formed by using pointers of type GFDetectorBase. This allows for different lists of detectors to be formed depending on the different conditions that exist.

Enumerations

enum GFDetectorMasking { TDWR_MASKING, NEXRAD_MASKING_WITH_V, NEXRAD_MASKING_WITHOUT_V };
This enumeration is used for dynamic identification of the type of masking. Allows for each detector to be used for multiple radar systems.

Constructors

*GFDetectorBase(char *, TclObject &);*
Default constructor. Initializes GFDetectorParams class and the prior interest image if applicable.

Destructors

virtual ~GFDetectorBase();
Default destructor. Deletes the prior interest image if applicable.

Public virtual functions

*virtual void RunDetector(GFBaseImages &, void * = NULL) = 0;*
Function used to compute the current interest image for the detector.

virtual SKArray<short> GetCurrInterestImg(void) = 0;*
Returns a pointer to the current interest image of the derived detector.

Public member functions

void Run(GFBaseImages &);
Calls the *RunDetector()* of the derived class.

SKArray<short> GetPriorInterestImg(void);*
Returns a pointer to the prior interest image of the derived detector.

See Also

Library skarr, class SKArray.
Library baseobj, class GFDetectorParams.

Document Revision Date

10 December, 1998

class GFDetectorParams

Name

class GFDetectorParams

Synopsis

#include <gfDetectorParams.h>

Hierarchy

GFDetectorParams

Description

Class to contain the variables for the detector parameters.

Constructors

GFDetectorParams();
Default constructor. Sets all history variables to 0.

Destructors

~GFDetectorParams();
Default destructor. Deletes history weights.

Public data members

char name[256];
Parameter name.

char dataClass[256];
Type of data (e.g., DZ, V, INTEREST). Used for display colormap.

int historySize;
Number of current and prior detector images to maintain.

*float *historyWeights;*
Array of temporal weight factors.

Related global functions

friend ostream& operator << (ostream &, GFDetectorParams &);
Insertor for the GFDetectorParams class. Inserts the data members and their meaning into an ostream.

Document Revision Date

10 December, 1998

class GFEvent

Name

class GFEvent

Synopsis

#include <gfevent.h>

Hierarchy

GFEvent

Description

Class to store all final computational results from a single scan of MIGFA processing. A "LLDList" of such objects make up the MIGFA processing "history".

Constants

#define MAX_HEADER_PRODS 25

Constructors

GFEvent();
Default Constructor. Initializes variables, sets *siteWind, airportWind,* and *regionalWind* to *NULL.*

Destructors

~GFRefData();
Default Destructor. Deletes all GFWind objects and GFChains on the *gfChains* list.

Public data members

LLDListLink link;
Link to another GFEvent.

short imageNum;
Arbitrary image number.

unsigned long currentTime;
Time in seconds of the radar data corresponding to this event.

unsigned long referenceTime;
Time in seconds of a prior volume scan. This prior scan is used as a basis for estimating motion and in computing speeds.

GFWind siteWind;*
Site wind estimate.

GFWind airportWind;*
Airport wind estimate.

GFWind regionalWind;*
Regional wind estimate.

LLDList gfChains;
LLDList of "GFChain" structures, each encoding computed results for a single gust front.

GFPointArray *gfPointArray;*
Single array which encodes information about all gust fronts detected for the current scan.

SKArray<short> *prevIndices*
"PrevIndices" array. Each (x,y) location in this array stores the index of the corresponding point into the previous event's "gfPointArray" structure.

See Also

class LLDList, class GFWind, class GFPointArray, Library skarr, class SKArray.

Document Revision Date

19 March, 2002

class GFHeuristicsBase

Name

class GFHeuristicsBase

Synopsis

#include <gfHeuristicBase.h>

Hierarchy

GFHeuristicBase

Description

Base class for the heuristics. All heuristics classes are derived off of this class. This is done so that an LLNIDList of pointers to the heuristics can be formed by using pointers of type GFHeuristicBase.

Component Structures

```
struct GFHehuristicParams
{
        char    name[256];
};
```

Structure to hold the name of the heuristic.

Constructors

*GFHeuristicBase(char *, TclObject &);*
Default constructor. Copies the name of the heuristic into the GFHeurisiticParams structure and reads in the parameters.

Destructors

~GFHeuristicBase();
Default destructor. Currently a no-op.

Public virtual functions

*virtual bool RunHeuristic(GFChain *) = 0;*
Function defined in a derived class to compute the heuristic of a chain.

Public member functions

*bool Run(GFChain *);*
Calls *RunHeuristic()* defined in the derived class.

Access functions

char GetName(void);*
Returns a pointer to the name component of the GFHeuristicParams structure.

See Also

Library gfext, class GFChain

Document Revision Date

10 December, 1998

class GFPointArray

Name

class GFPointArray

Synopsis

#include <gfPointArray.h>

Hierarchy

GFPointArray

Description

Class for arrays of GFPointInfo objects. Since many gust front analysis operations involve splitting up an array of gust front points into two distinct chains, this class was implemented to ease cleanup of various discarded arrays, etc.

Constructors

GFPointArray();
This constructor create a GFPointArray object with default values of zero or NULL as appropriate.

GFPointArray(short);
This constructor takes as its argument the number of points to be encoded in the array. All points are assumed to lie in a single chain. An array of type GFPointInfo is created of size number of points. The number of chains is set to one, and the lenght of the chain is number of points.

GFPointArray(short, short);
Constructor with two arguments. First argument is number of points to be encoded in the array, the second is the number of chains represented in the array. An array of type GFPointInfo is created of size number of points. The number of chains is set to number of chains, and chainLengths is set to number of chains.

GFPointArray(const GFPointArray &);
Copy constructor. Creates GFPointArray object with all fields and data buffers copied from input GFPointArray.

Destructors

~GFPointArray();
Default destructor. Deletes all memory allocated for member arrays of pointArray and chainLengths.

Assignment operators

GFPointArray& GFPointArray::operator = (const GFPointArray &);
Assignment operator. Copies all fields from input GFPointArray and allocates space for GFPointArray returned from this operator.

Public data members

*GFPointInfo *pointArray;*
Array of "GFPointInfo" structures; used for efficiency as points stored in this fashion can be looked up just by indexing into an array rather than having to traverse point lists in the original GFChain objects.

short nPoints;
Total number of encoded gust front points in all chains.

short *nChains;*
Number of chains encoded in the array of points. Note that in some contexts, the GFPointArray class will be used to store information on all current gust front chains; in other contexts just a single chain.

short **chainLengths;*
Lengths of the individual chains contained in the pointArray. Used for indexing into the pointArray when computing attributes for gust front chains as a whole.

See Also

Library baseobj, class GFPointInfo.

Document Revision Date

10 December, 1998

class GFPointInfo

Name

class GFPointInfo

Synopsis

#include <gfPointArray.h>

Hierarchy

GFPointInfo

Description

Class for storing gust front point information. Same basic information as the *GFPoint* class, but useful for speeding up searches for points.

Enumerations

typedef enum SKType { GF_X, GF_Y, GF_DIRECTION, GF_PRIOR_INDEX, GF_DISTANCE_MOVED, GF_SPEED, GF_DEPTH, GF_INTEREST };
Enumerated type for specifying a particular chain attribute.

Constructors

GFPointInfo();
Default constructor. Creates a *GFPointInfo* with all data members set to zero.

Destructors

~GFPointInfo();
Default destructor. Currently a no-op.

Public member functions

void CopyFrom(GFPointInfo &);
Copy all attributes of a *GFPointInfo* object into another *GFPointInfo* object.

void CopyFrom(GFPoint &);
Copy all attributes of a *GFPoint* object into a *GFPointInfo* object.

Access functions

float GetFloatAttribute(GFPointAtt);
Function to return the value of the desired floating point member of a *GFPointInfo* object.

void SetFloatAttribute(GFPointAtt, float);
Function to set the value of the desired floating point member of a *GFPointInfo* object.

Public data members

short x;
short y;
short priorIndex;
short depth;
short interest;
float direction;
float distanceMoved;
float speed;

See Also

Library baseobj, class GFPoint.

class GFPointInfo

Document Revision Date 10 December, 1998

class GFPredChain

Name

class GFPredChain

Synopsis

#include <gfChain.h>

Hierarchy

SKChain->GFPredChain

Description

Class for holding information about a gust front prediction chain

Constructors

GFPredChain();
Default constructor. Sets *forecastTimeInMinutes* and *number* to *SKArray<short>::SK_NIL.*

Destructors

~GFPredChain();
Default destructor. Deletes all GFPoints which comprise the gust front prediction curve.

Public data members

short forecastTimeInMinutes;
Valid "forecast time" for the chain. The forecast time will be a 1-minute interval.

short number;
An arbitrary number assigned to a GF for bookkeeping. The number matches the *number* field of a corresponding *GFChain.*

See Also

class *SKChain* (CSKETCH library), class *GFPoint,* class *GFPredChain*

Document Revision Date

19 February, 2002

class GFRefData

Name

class GFRefData

Synopsis

#include <gfrefData.h>

Hierarchy

GFRefData

Description

Class to contain data used throughout the algorithm. Most of the data contained in this class will change from scan to scan.

Constants

#define MAX_HEADER_PRODS 25

Component Structures

```
typedef struct
{
        short   prf;
        short   scanNum;
        short   tiltNum;
        LLTime time;
        short   prodScale[MAX_HEADER_PRODS];
        short   prodOffset[MAX_HEADER_PRODS];
        char    *siteName;
} ScanHeader;
```

Structure to hold data taken from the scan header of the tilt.

```
typedef struct {
        unsigned long time;
        SKArray<float> *uArray;
        SKArray<float> *vArray;
        SKArray<float> *uVarArray;
        SKArray<float> *vVarArray;
} TWindData;
```

Structure to hold ITWS Terminal Winds data (not used for WSP).

Constructors

GFRefData();
Default Constructor. Initializes all non LLDList elements to zero.

Destructors

~GFRefData();
Default Destructor. Deletes all data on all LLDLists.

Public data members

TWindData twindData;
Terminal gridded winds data for the current scan.

ScanHeader scanHeader;
ScanHeader contains base data time, product scalings, etc.

LLDList baseScan;

class GFRefData

List of prior DZ, V, and CONV images from a prior radar scan; needed by detectors which are motion-based.

short *stratRainCoverage;*
A number used to indicate the degree of coverage of the radar images with stratiform rain.

short *stormCellCoverage;*
A number used to indicate the degree of coverage of the radar images with storm cells.

unsigned long *currentTime;*
Current time in seconds.

LLDList *anemometerHistory;*
List of anemometer data recorded over the last "anemometerHistoryHours" (parameter).

bool *summerMode;*
summerMode is true if the current data is within the summer months defined by *GFParams->summerMonths.*

LLNIDList *dzHistory;*
List of prior reflectivity images for the last "imageHistoryLength" (parameter) scans.

LLNIDList *vHistory;*
List of prior velocity images for the last "imageHistoryLength" (parameter) scans.

LLNIDList *convHistory;*
List of prior velocity convergence images for the last "imageHistoryLength" (parameter) scans.

LLNIDList *sdHistory;*
List of prior velocity standard deviation images for the last "imageHistoryLength" (parameter) scans.

LLDList *eventHistory;*
List of GFEvent objects containing the computational results of the last *gfParams->eventHistoryLength* scans.

See Also Library llutil, class LLDList, class LLNIDList, struct LLTime.

Document Revision Date 10 December, 1998

class GFWindEstBase

Name

class GFWindEstBase

Synopsis

#include <gfWindEstBase.h>

Hierarchy

GFWindEstBase

Description

Base class for the wind estimators. All wind estimators are derived off of this class. This is done so that an LLNIDList of pointers to the estimators can be formed by using pointers of type *GFWindEstBase*.

Component Structures

```
struct GFWindEstParams
{
        char    name[256];
};
```

Structure to hold the name of the wind estimator.

Constructors

*GFWindEstBase(char *, TclObject &);*
Default constructor. Copies the name of the estimator into the GFWindEstParams structure and reads in the parameters.

Destructors

virtual ~GFWindEstBase();
Default destructor. Currently a no-op.

Public virtual functions

virtual GFWind RunEstmtrAhead(GFChain *) = 0;*
Function defined in the derived class to estimate the wind ahead of the front.

virtual GFWind RunEstmtrInside(GFChain *) = 0;*
Function defined in the derived class to estimate the wind inside of the front.

virtual GFWind RunEstmtrBehind(GFChain *) = 0;*
Function defined in the derived class to estimate the wind behind the front.

Public member functions

GFWind RunAhead(GFChain *);*
Calls the *RunEstmtrAhead()* function as defined in the derived class.

GFWind RunInside(GFChain *);*
Calls the *RunEstmtrInside()* function as defined in the derived class.

GFWind RunBehind(GFChain *);*
Calls the *RunEstmtrBehind()* function as defined in the derived class.

Public data members

GFWindEstParams params;
Name of wind estimator.

class GFWindEstBase

See Also

Library gfext, class GFChain.
Library baseobj, class TclObject.

Document Revision Date

10 December, 1998

class TclObject

Name

class TclObject

Synopsis

#include <tclObject.h>

Hierarchy

ParamTcl->TclObject

Description

This class acts as an interface to a Tcl interpreter.

Constructors

*TclObject(char *) : ParamTcl(file);*
Default constructor. Takes a pointer to a file name which is used in the call to the base class constructor.

Destructors

~TclObject();
Default destructor. Currently a no-op.

Public member functions

*void get(char *, short *);*
*void get(char *, int *);*
*void get(char *, float *);*
*void get(char *, char *);*
*void get(char *, short **, int *);*
*void get(char *, int **, int *);*
*void get(char *, float **, int *);*
*void get(char *, char **, int *);*
*void get(char *, short [], int *);*
*void get(char *, int [], int *);*
*void get(char *, float [], int *);*
*void get(char *, LLNIDList &);*
All of the preceding Public Member Functions are wrappers for calls to the appropriate functions in *ParamTcl*.

*void get(char *, GFDetectorParams &);*
Function to read in detector parameters for detector with name as passed in by the character pointer. Parameters read in from file as defined in the constructor.

*void get(char *, GFWindEstParams &);*
Function to read in wind estimator parameters for estimators with name as passed in by the character pointer. Parameters read in from file as defined in the constructor. Currently a no-op.

*void get(char *, GFDeltaVEstParams &);*
Function to read in delta v estimator parameters for delta v estimators with name as passed in by the character pointer. Parameters read in from file as defined in the constructor. Currently a no-op.

*void get(char *, GFHeuristicParams &);*
Function to read in heuristic parameters for heuristics with name as passed in by the character pointer. Parameters read in from file as defined in the constructor. Currently a no-op.

See Also

Libraries paramTcl, llutil, baseobj. Classes ParamTcl, LLNIDList, GFDetectorParams, GFWindEstParams, GFDeltaVEstParams, GFHeuristicsParams.

Document Revision Date

10 April, 2002

GLOSSARY

AP	Anomalous Propagation
ARP	Airport Reference Point
ASOS	Automated Surface Observing System
ASR-9	Airport Surveillance Radar - 9
ATC	Air Traffic Control
DMA	Direct Memory Access
FAA	Federal Aviation Administration
FTC	Functional Template Correlation
GMT	Greenwich Mean Time
GFUP	Gust Front Update
I/O	Input/Output
LLWAS	Low Level Windshear Alert System
MDT	Maintenance Display Terminal
MIGFA	Machine Intelligent Gust Front Algorithm
PPI	Plan Position Indicator
PRF	Pulse Repetition Frequency
RDT	Ribbon Display Terminal
SD	Situation Display
SHM	Shared Memory
SNR	Signal to Noise Ratio
TCP/IP	Transmission Control Protocol/Internet Protocol
VAD	Velocity Azimuth Display
VSP	Variable Site Parameter
WSP	Weather Systems Processor

REFERENCES

1. R.L. Delanoy, S.W. Troxel, "The Machine Intelligent Gust Front Algorithm", MIT Lincoln Laboratory, Project Report ATC-196, 1993
2. Weber, M.E., "ASR Weather Systems Processor Signal Processing Algorithms", MIT Lincoln Laboratory, Project Report ATC-255, Publication Pending
3. Newell, Oliver, "ASR-9 Weather Systems Processor Software Overview", MIT Lincoln Laboratory, Project Report ATC-264, October 20, 2000
4. Newell, Oliver, "WSP Utility Libraries", MIT Lincoln Laboratory Project Report ATC-284, October 23, 2000
5. S. Levialdi, "Parallel pattern processing", IEEE Trans. Syst. Man and Cybern., SMC-1:292-296, 1971