PNNL-11383

Development of Automated Image Co-Registration Techniques: Part II - Multisensor Imagery

T. F. Lundeen
A. K. Andrews*
E. M. Perry
M. V. Whyatt
K. L. Steinmaus

October 1996

Prepared for the U.S. Department of Energy under Contract DE-AC06-76RLO 1830

Pacific Northwest National Laboratory
Richland, Washington 99352

*Post-Doctoral appointment through Association of Western Universities

Development of Automated Image Co-Registration Techniques:
Part II - Multisensor Imagery

Executive Summary

This report summarizes results from the second phase of research on multisensor image registration conducted at the US DOE Pacific Northwest National Laboratory in conjunction with a US DOE office of nonproliferation and national security task titled Multispectral Imagery Analysis (ST474D). The major changes implemented in this phase include modifying the code to run independently of the Advanced Visualization Software (AVS) required in the first version, and adding an image intersection module to determine new ground control points (GCPs) outside the convex hull of the initial GCP dataset. Registration results are presented for cases with images of differing pixel resolutions and spectral bands. Instructions for running the programs and hardcopies of the source code are also included in the report.

Contents

Development of Automatic Image Co-Registration Techniques:
Part II - Multisensor Imagery

1.0 Background

This is the second in a series of PNNL Multispectral Imagery (ST474D) reports on automated co-registration and rectification of multisensor imagery. In the first report (Risch et al. 1992), a semi-automated registration procedure was introduced based on methods proposed by Chen and Lee (1992) which emphasized registration of same sensor imagery. The Chen and Lee approach is outlined in Figure 1, and is described in detail in the first report. PNNL made several enhancements to the Chen and Lee approach; these modifications are outlined in Figure 2 and are also described in detail in the first report. The PNNL enhancements to the Chen and Lee approach introduced in the first phase have been named Multisensor Image Registration Automation (MIRA). These improvements increased computational efficiency and offered additional algorithms for coarse matching of disparate image types. In the MIRA approach, one set of optimum GCP locations are determined based on a Delaunay triangulation technique using an initial set of GCPs provided by the user, rather than repeating this step for each added control point as is proposed by Chen and Lee. The Chen and Lee approach uses an adjacent pixel difference algorithm for coarse matching patches of the reference image with the source image, while the MIRA approach adds other algorithms. Also the MIRA approach checks to determine if the a newly determined GCP fits the existing warping equation.

In the first phase of the research, several improvements were identified for future implementation such as the ability to determine new GCPs outside the convex hull formed by the initial GCP set. In this phase of the research, emphasis was placed on implementing improvements identified in the first phase, and testing the MIRA approach on images with different ground resolutions and spectral ranges characteristic of multisensor imagery.

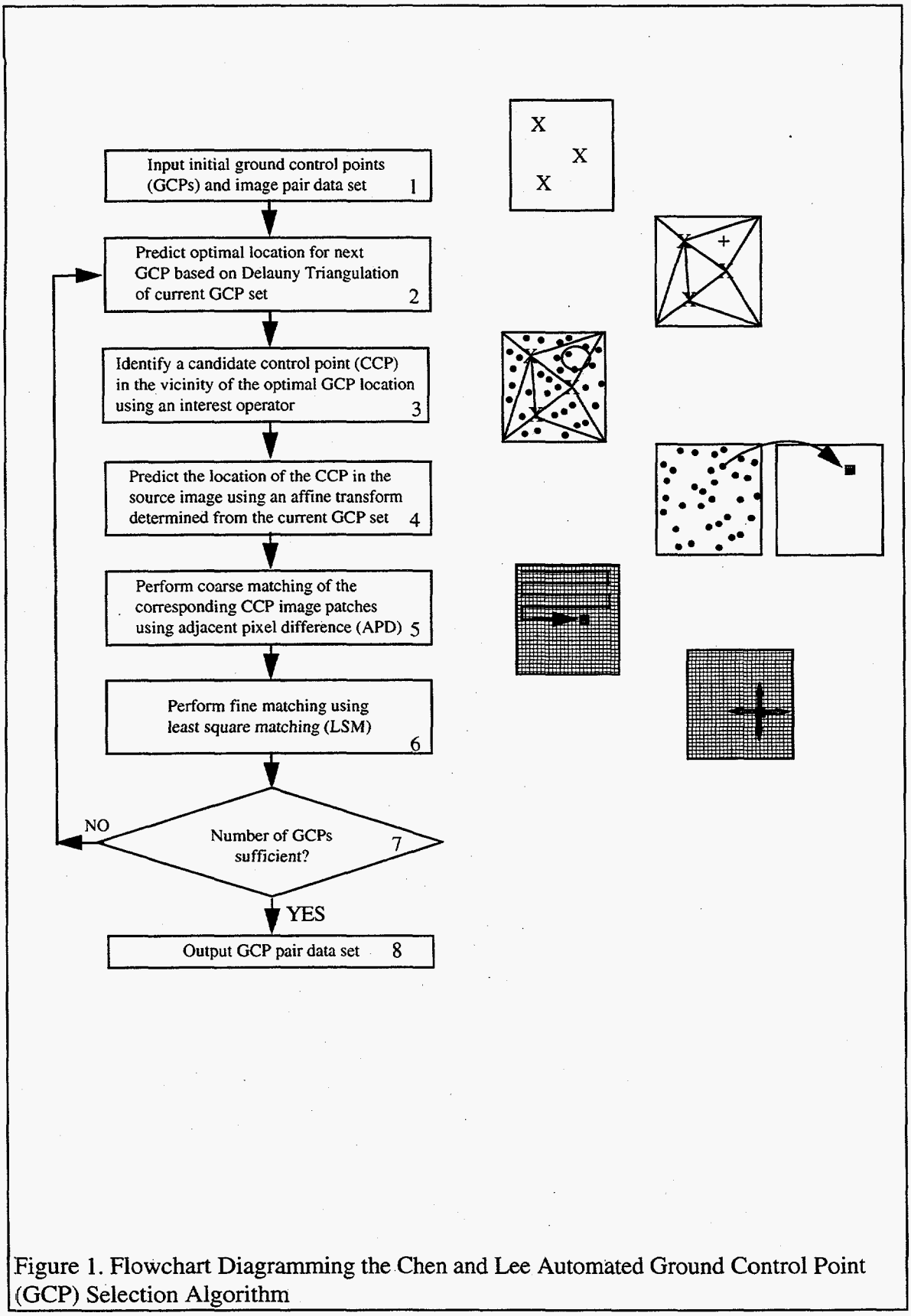


Figure 1. Flowchart Diagramming the Chen and Lee Automated Ground Control Point (GCP) Selection Algorithm

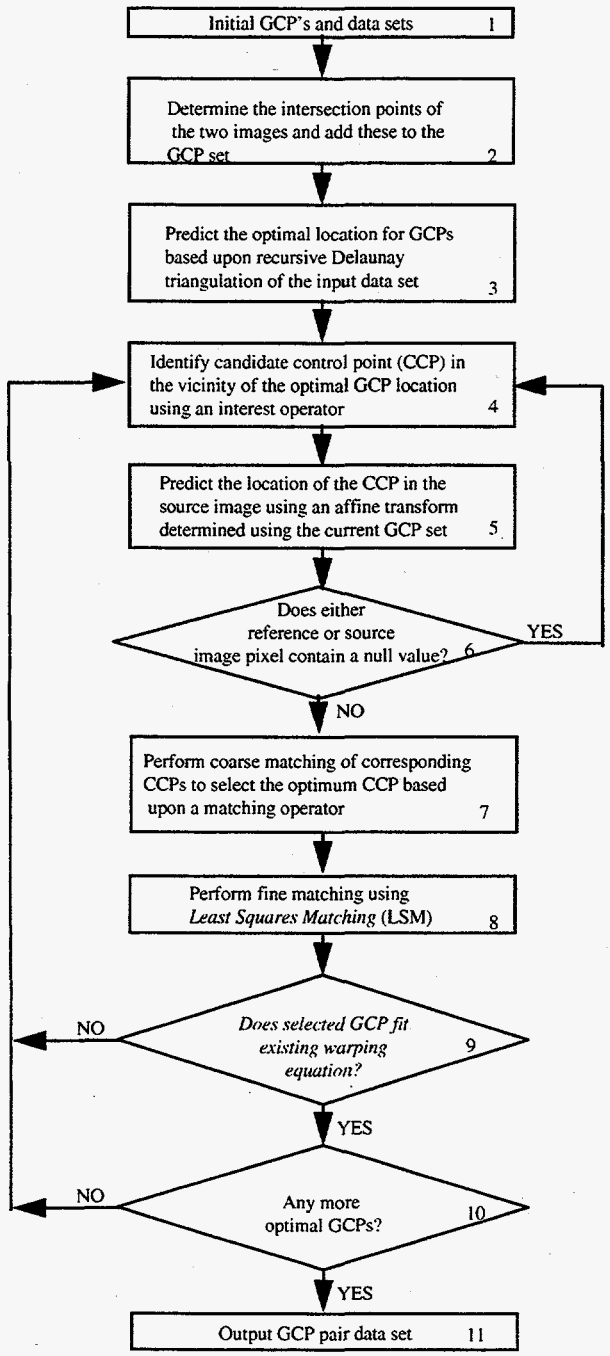


Figure 2. Flowchart Diagramming the PNNL Multisensor Image Registration Automation (MIRA) Algorithm.

In addition to the co-registration difficulties introduced in same-sensor imagery (such as differences in image geometry), multisensor imagery adds additional challenges. Features in multisensor images may have different spatial and statistical characteristic due to:

- specific sensor properties such as field of view and pixel aspect ratio
- different sources of characteristic noise (such as 'striping' or 'line-dropping')
- differences in apparent reflectivity or emissivity properties of the target due to differences in the spectral bands used
- differences in atmospheric transmittance, absorption and scattering in different waveband intervals

In a recent survey by Fonseca and Manjunath (1996) of multisensor registration techniques, seven citations were noted of registration techniques tested on multisensor imagery; however, none of the cited work incorporated imagery with major differences in the waveband regions, such as optical versus thermal imagery or optical and SAR imagery. The registration techniques mentioned in their survey included both area-based (correlating image patches based on a metric such as correlation coefficients or normalized cross correlation) and feature-based (extracting features based on detection of edges, lines, region centroids, etc.) techniques. Because of the complexities of merging optical, thermal and radar data, the PNNL MIRA approach uses both area-based statistics and feature extraction.

2.0 Approach

In this section we discuss the improvements to the enhanced Chen and Lee approach made during phase two of this research . These improvements include the conversion to stand-alone C code (eliminating the use of licensed software), an image intersection module, a user-specified null value check, and additional coarse matching algorithms.

2.1 Conversion to Independent (ANSI C) Code

The original MIRA algorithms introduced in the first phase utilized the Application Visualization System (AVS), which is an object-based visual programming system. Although AVS provided an excellent platform to develop and test the initial software, especially the Delaunay triangulation module, the need to develop hardware independent programs was recognized. In this second phase of the MIRA work, the Delaunay triangulation module was modified to operate outside AVS. The programs modified in this phase should run under any machine platform supporting ANSI C.

2.2 Locating GCPs Outside Convex Hull

One problem identified in the first phase of this research was the limitation of finding control points only within the convex hull, or space defined by the set of initial GCPs. This problem is illustrated in Figure 3. Coarse matching of the source and reference images is constrained to the triangulated areas bounded by the initial GCP set. To solve this problem, an image intersection module has been added (see Figure 2), which determines the corner points of the source file with respect to the reference file coordinates. This is accomplished in one of two ways:

- In the first case, where one image is completely contained within the other, the four corner points of the contained image are returned.
- In the alternate case, where there is overlap between the two images, the source image is 'clipped' against the reference image, using the Cohen-Sutherland line clipping algorithm (Foley et al., 1994).

To initialize this procedure, the user identifies 3-4 GCPs and the dimensions of the two images (minimum and maximum x and y values for the two images). The four corners representing the area of intersection of the two images are then calculated, and these four points are then added to the user specified GCPs to form the initial GCP dataset. As the GCP dataset grows, all of the area bounded by the intersection of the two images is included.

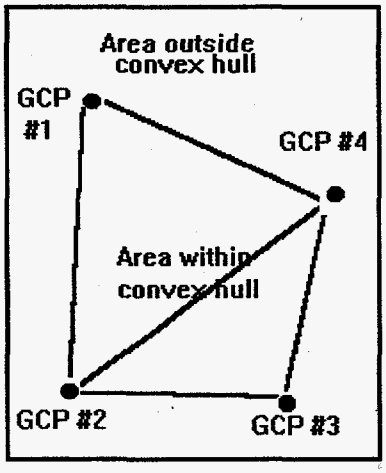


Figure 3. Convex Hull Problem.

2.3 User Specified Null Value

Offsite or null value pixels may be included in an image file from one or more sources, such as:

- image registration causing the image to be skewed or rotated relative to the file coordinate system
- when roll correction has been performed, leaving an irregular or wavering edge of the image
- when there are image 'drop-outs' or other sources of image noise, artifacts, or unwanted features (such as clouds and cloud shadows)

To account for these possibilities, an option was added to allow the user to specify a value for offsite or null pixels. Optimum control points are generated for the entire reference image file (including offsite areas). However, when performing the coarse matching of the reference to source files, the image patches are checked to make sure that null values are not included in the patch. Patches which include null values are discarded.

2.4 Coarse Image Matching Algorithms

In preparation for the coarse matching step of the autoregistration process, both optimum control points (based on Delaunay triangulation) and candidate control points (based on feature extraction) are generated from the reference image. For each optimum control point, the closest candidate control point is selected for processing, and is referred to as the local control point (LCP). Given the location of the LCP in the reference image, the approximate location in the source image is determined from an affine transformation of the existing GCPs. This location is used as the center for the image patch in the reference image, and as the center for a search area in the source image. A series of patches in the source image are extracted. For each patch, one of the coarse matching metrics (adjacent pixel difference, correlation, or patch coding) is used to determine the matching factor between the reference and source patch. After comparing the reference patch with all of the source patches within the search area, the source patch with the best match is used to determine the new GCP (as the center of both the reference and source patches). Each of the coarse matching algorithms is described below.

The adjacent pixel difference (APD) metric evaluates the similarity of two image patches based on feature vectors determined from row and column statistics. The feature vector of an M by M image patch in the reference image is defined as (after Chen and Lee, 1992):

$$f^T_m = [R_1, R_2, \ldots R_m, C_1, C_2, \ldots C_m]$$

where the row and column features are determined as:

$$R_l = \frac{\sum_{y=1}^{m-1} |DN(l,y) - DN(l,y+1)|}{\sum_{y=1}^{m} DN(l,y)}$$

for l = 1,2,...m

$$R_s = \frac{\sum_{x=1}^{m-1} |DN(x,s) - DN(x+1,s)|}{\sum_{x=1}^{m} DN(x,s)}$$

for s = 1,2 ...m

The feature vector for the source image, $f_{i,j}$ is determined in the same way, using the DN values from an M by M patch from the source image. The degree of matching is evaluated by the correlation coefficient between image patches in the source and reference images, as defined by:

$$C_{i,j} = \frac{\mathbf{f}_m^T \bullet \mathbf{f}_{i,j}}{\|\mathbf{f}_m^T\| \|\mathbf{f}_{i,j}\|}$$

For the image correlation, no processing is performed on either the reference or source patches. The degree of matching is based on the normalized cross correlation, defined as:

$$C = \frac{\sum_{i=1}^{N} (\text{ref})(\text{source})}{\sqrt{(\sum_{i=1}^{N} \text{ref}^2)(\sum_{i=1}^{N} \text{source}^2)}})$$

where "ref" is a pixel in the reference patch, "source" is the corresponding pixel in the source patch, and "N" is the number of pixels in the patch.

The image coding metric evaluates all of the interior pixels in the patch (i.e., not the edge rows or columns) to compute the difference between a given pixel and its eight surrounding pixels. Then the locations of the four pixels which differ most from this center pixel are used to compute a location code, defined as:

$$C = \Sigma(2^{L(kk)}),\ kk=1,...4$$

Where L(kk) is the location (1 through 8) of pixel kk. After the code values are determined for both the reference and source images, the degree of matching between the reference and image patches is evaluated as:

$$\Sigma(\text{\# of pixels with same code})/(\text{\# pixels} * 4)$$

Note that the denominator represents a 100 percent match.

3.0 Results

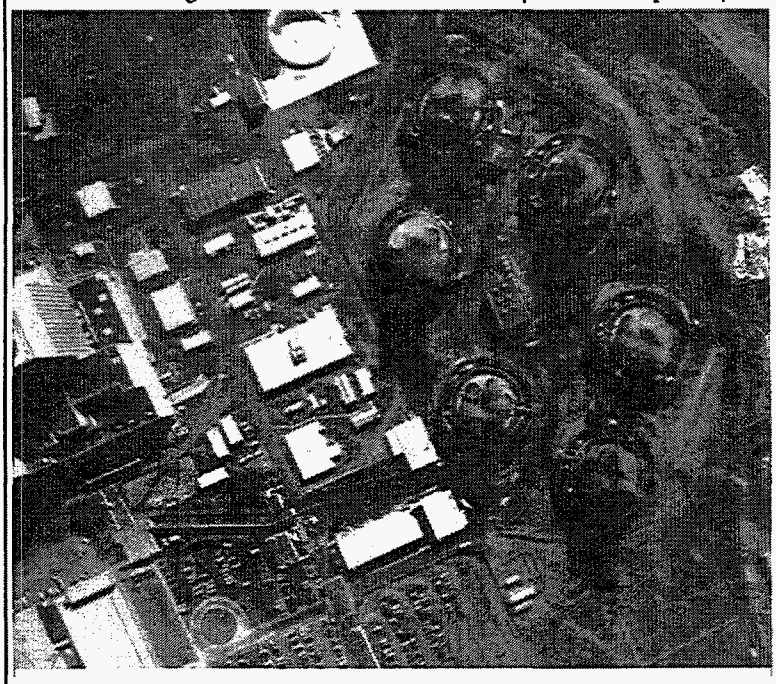
During the second phase of the PNNL MIRA development, tests were performed using multisensor imagery in the visible, near-, mid-, and thermal infrared. Results from these test will be presented to demonstrate the performance of the PNNL MIRA system for various types of multisensor imagery.

The first two examples draw on data collected during Mission 4 of the Department of Energy AMPS data collection over the Washington Public Power Supply System (WPPSS) nuclear power plant on the Hanford site in Richland, WA. In the first example, AMPS high resolution photography is used as the reference image (Figure 4, upper left image), and Daedalus multispectral imagery (Channel 8, daytime) was used as the source image to be warped (see Figure 4, upper right). The coded patch algorithm was used with the search area and patch size set to 25 pixels. The canidate contol point (CCP) threshold, which specifies the contrast required to consider a pixel as part of a CCP feature, was set to 20 digital numbers (DNs). The resulting warped image is shown in the lower left of Figure 4. Note that despite the loss of contrast and spatial resolution in the Daedalus imagery (relative to the photography), the MIRA software was able to find sufficient control points to successfully warp the source image.

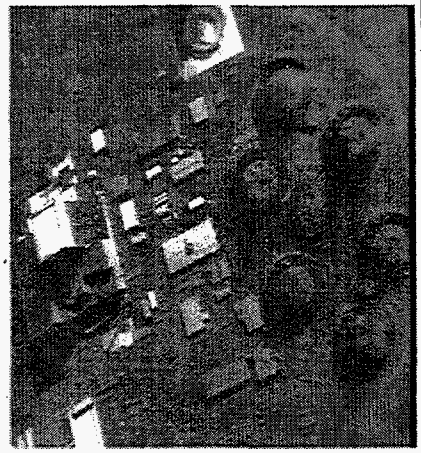
The second example demonstrates the registration of AMPS Daedalus pre-dawn thermal-IR imagery (upper right of Figure 5) to Daedalus daytime mid-IR (upper left of Figure 5). In this case, the search length was specified as 20 pixels, while the patch size remained 25 pixels, and the coded patch matching algorithm was used. The CCP

Semiautomatic Image Registration Example

Reference Image: AMPS RC-30 Aerial IR Photo (Near IR component)



Source Image: AMPS Daedalus Daytime Channel 8 (0.91 - 1.04 microns)



Warped Source Image

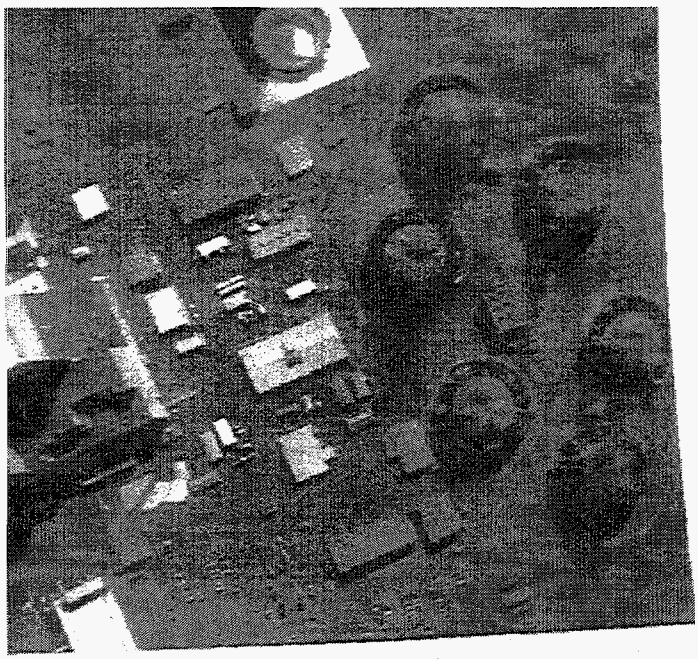


Figure 4. Registration of AMPS Daedalus Channel 8 to RC-30 Photography.

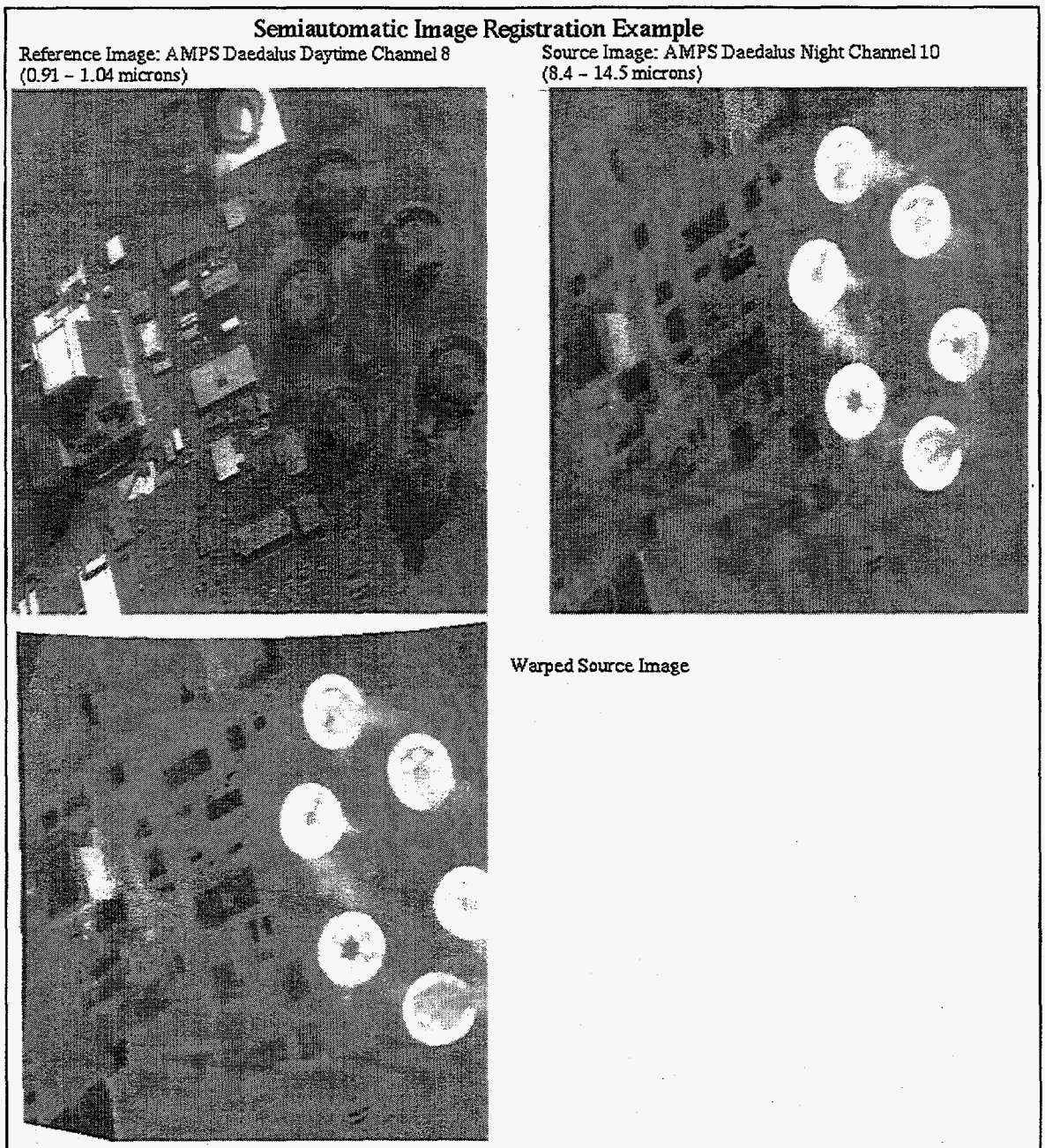


Figure 5. Registration of AMPS Daedalus Channel 10 Nighttime to Daedalus Channel 8 Daytime Imagery.

threshold was set at 10 DNs. The resulting warped image is shown in the lower left of Figure 5. Again, the source image (nighttime thermal infrared) has very different characteristics than the reference image, but the MIRA approach was able to successfully warp the image.

Additional tests were performed to compare results using the different coarse matching algorithms for imagery at different spatial resolutions, and to quantify the results of progressively increasing the spectral difference between the reference and source imagery. The first test utilized Daedalus Channel 6 imagery (.69 - .75 microns), collected in conjuction with AMPS Mission 9 over Camp Pendleton, CA. Table 1 lists these results. Column 5 indicates the root mean square error (RMSE) determined with that set of GCPs, based on a first order fit through the entire set. The RMSE values are much lower than what would be expected from manual selection of GCPs; the RMSE of manually selected GCPs may approach the number of GCPs. Note that the coded patch metric produced the fewest GCPs (while yielding the lowest RMSE), and the image correlation metric generated the greatest number of GCPs. Also, the different coarse matching algorithms selected different features for GCPs (see Figure 6).

Table 1. Comparison of three coarse matching algorithms.

Source Image	Reference Image	Matching Algorithm	Number of GCPs	RMSE (pixels)
2250' AGL Daedalus B6	4500' AGL Daedalus B6	Image Correlation	155	5.3
"	"	Adj. Pixel Diff.	96	4.1
"	"	coded patch	31	3.99

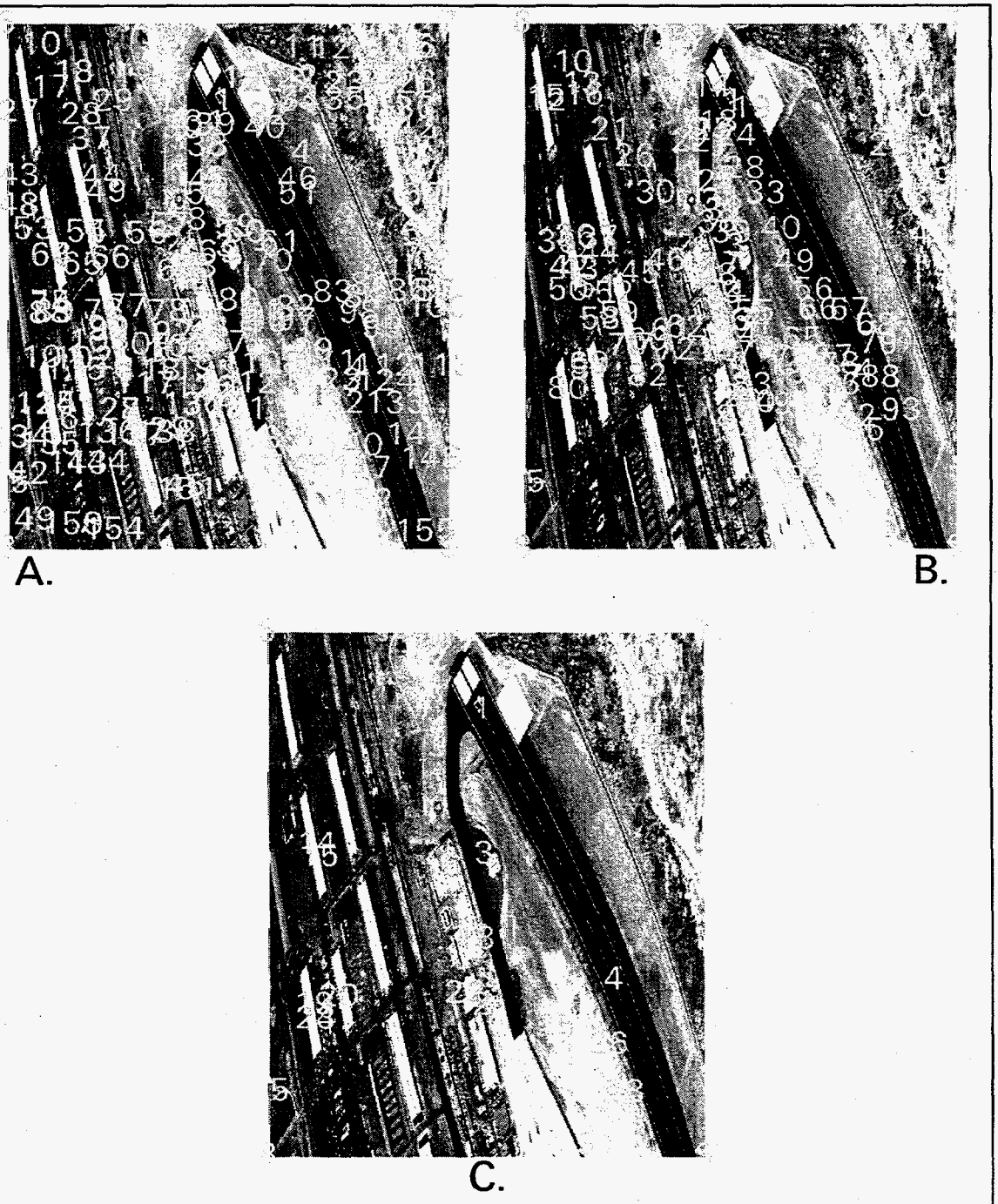


Figure 6. Network of Calculated GCPs on the Reference Image Using Correlation (A), APD (B) and Coded Patch Algorithms (C).

In the second test, Daedalus imagery from the same Camp Pendleton, CA dataset was used. For the reference image, Daedalus Channel 6 (0.69 - 0.75 microns), collected at 2250 ft. AGL, was selected. For the source images, Daedalus Channels 8 (.91 - 1.05

microns), 9 (3.0-5.5 microns) and 10 (8.4 to 14.5 microns) were used. The intent was to evaluate the GCP dataset generated, and the resulting RMSE for multisensor images as the separation between the bands increased. Each of the three coarse matching algorithms were also evaluated. Table 2 summarizes the results of these analyses, for each of the three coarse matching algorithms and each of the three Daedalus Channels used as source images.

Table 2. Evaluation of band separation on different coarse matching algorithms.

Source Image	Reference Image	Matching Algorithm	Number of GCPs	RMSE (pixels)
2250' AGL Daedalus B6	2250' AGL Daedalus B8	Image Correlation	126	5.8
"	2250' AGL Daedalus B9	"	125	8.0
"	2250' AGL Daedalus B10	"	123	7.6

Source Image	Reference Image	Matching Algorithm	Number of GCPs	RMSE (pixels)
2250' AGL Daedalus B6	2250' AGL Daedalus B8	Adjacent Pixel Diff.	88	6.1
"	2250' AGL Daedalus B9	"	47	5.7
"	2250' AGL Daedalus B10	"	47	5.6

Source Image	Reference Image	Matching Algorithm	Number of GCPs	RMSE (pixels)
2250' AGL Daedalus B6	2250' AGL Daedalus B8	Coded Patch	49	5.7
"	2250' AGL Daedalus B9	"	12	8.1
"	2250' AGL Daedalus B10	"	12	7.2

The results show that in general, as the band waveband difference increases between reference and source images, the number of GCPs decreased, however GCPs were found for all of the source images. As with the previous test, the image correlation metric found the number of GCPs followed by the APD and the coded patch metric. Also, the RMSE values were again smaller than what would be expected from manual selection of GCPs.

4.0 Conclusions

Several modifications have been implemented in phase two of the multisensor image co-registration research. The MIRA software developed at PNNL now includes three image patch matching algorithms and an image intersection module to allow GCP determination outside of the convex hull of the original GCPs. The MIRA software no longer requires AVS software. Tests in phase two have demonstrated successful registration of multisensor imagery. RMSE values (based on a first order fit through the GCPs calculated with MIRA) indicate accuracies much better than normally obtained with manual registration.

Future enhancements to the MIRA software will include improved algorithms for coarse matching of image patches, co-registration of radar and optical imagery, and testing on other types of reference images, such as digital maps and digital terrain model data. Fonesca and Manjunath (1996) suggest that it is unlikely that a single registration technique will work satisfactorily for all applications, and they suggest integrating the various techniques into a rule based artificial intelligence system. Currently the MIRA software contains options for matching algorithms and search parameters which the user can fine-tune for various applications. Future enhancements may include machine intelligence to best select the correct matching algorithms and search parameters for a given combination of imagery.

References

Chen, L.C. and L.H. Lee. 1992. Progressive generation of control frameworks for image registration. Photogrammetric Eng. and Remote Sensing. 58(9):1321-1328.

Foley, J.D., A. van Dam, S.K. Feiner, J.F. Hughes, R.L. Phillips. 1994. Introduction to Computer Graphics. 2nd ed. Addison Wesley, Reading MA.

Fonseca, M.G. and B.S. Manjunath. 1996. Registration techniques for multisensor remotely sensed imagery. Photogrammetric Eng. and Remote Sensing. 62(9):1049-1056.

Risch, J.S., T.F. Lundeen, K.L. Steinmaus, G.E. Wukelic, and G.M. Petrie. 1994. Development of automated image co-registration techniques: Part 1 - same sensor imagery. Report to DOE under contract DE-AC06-76RLO.

APPENDIX A. HOW TO RUN THE AUTOREG SOFTWARE

Running the PNNL MIRA (Multisensor Image Registration Automation) software requires four steps:

- Select an initial GCP set
- Run ClipImages to determined the intersection points of the two images
- Run DelTriang to determine the optimum control point locations
- Run AutoReg to generate the new GCP dataset

These steps are described below.

1. Manual selection of the minimum number of ground control points.

- Select a minimum of 4 ground control points between the source and reference image.
- Export them into a plain ASCII file. There should be one set of control points per line, space delimitted, with source file x and y coordinates preceding the reference file x and y coordinates. The coordinates may be either integer or real numbers. There should be no blank lines or comments.

Here is an example exported GCP file:

167.1560 699.0009 25.8842 833.1438
192.2160 331.5397 62.1073 76.5127
533.9680 349.9777 666.5731 18.4717
398.6695 640.3110 438.6048 670.6290
378.9331 412.0436 402.9410 195.1122

2. Find the intersection of the two images using ClipImages.
ClipImages assumes the origin is at the images' upper left corner. If the software used to select the GCP's assumes the origin is at the lower left (such as ERDAS IMAGINE does) then the y-coordinates must be reversed. You can do this with the "yflip" utility which knows how to reverse source or reference y-coordinates in GCP files (format above). For example, to reverse a source image with 804 rows of samples and also a corresponding reference image with 866 rows:

yflip.pl -w src -l 804 -f gcpfile > gcp_srcflipped
yflip.pl -w ref -l 866 -f gcp_srcflipped > gcp_bothflipped

gcp_bothflipped would look like:

167.1560 105.4990 25.8842 33.3561
192.2160 472.9602 62.1073 789.9872
533.9680 454.5222 666.5731 848.0282

398.6695 164.1889 438.6048 195.8709
378.9331 392.4563 402.9410 671.3877

Next, run ClipImages. There are 9 required command line arguments that must be given in the following order:

- the name of the GCP file, with relative or absolute path name if appropriate
- the minimum x pixel value for the source file (the first column - usually 0 or 1)
- the minimum y pixel value for the source file (the first row - usually 0 or 1)
- the maximum x pixel value for the source file
- the maximum y pixel value for the source file
- the minimum x pixel value for the reference file (the first column - usually 0 or 1)
- the minimum y pixel value for the reference file (the first row - usually 0 or 1)
- the maximum x pixel value for the reference file
- the maximum y pixel value for the reference file

The pixel values can be given as integer or real numbers, for example:

```
ClipImages gcp_bothflipped 0.5 0.5 720.5 804.5 \
0.5 0.5 720.5 866.5 > clip.out
```

clip.out might look like:

```
151.611632   95.276201    0.500000    0.500000
556.273869   46.592385  720.500000    0.500000
562.156006  463.521309  720.500000  866.500000
157.493769  512.205125    0.500000  866.500000
```

3. Find the Optimum Control Points using DelTriang.

First, parse the coordinates defining the intersection for the reference image into a separate file:

```
awk '{printf("%f %f\n", $3, $4);}' clip.out > clip.refcoords
```

clip.refcoords would look like:

```
0.500000 0.500000
720.500000 0.500000
720.500000 866.500000
0.500000 866.500000
```

Then, parse the reference image coordinates from the (possibly flipped) GCP file and concatonate it, with the file of intersection points, into a new file.

```
awk '{printf("%f %f\n", $3, $4);}' gcp_bothflipped > gcp.refcoords
```

cat gcp.refcoords clip.refcoords > both.refcoords

Then run DelTriang:

DelTriang both.refcoords > DelTriang.out

DelTriang.out might look like:

```
 41.000000  541.000000
156.000000  721.000000
117.000000  106.000000
553.000000  167.000000
603.000000  414.000000
647.000000   96.000000
450.000000  783.000000
  0.000000  757.000000
501.000000  593.000000
334.000000  123.000000
416.000000  363.000000
369.000000  531.000000
622.000000  541.000000
316.000000  279.000000
140.000000  480.000000
270.000000  747.000000
  0.000000  108.000000
539.000000  315.000000
564.000000  703.000000
134.000000  324.000000
540.000000    0.000000
540.000000   41.000000
180.000000  866.000000
630.000000  866.000000
203.000000  615.000000
180.000000    0.000000
720.000000  649.000000
217.000000  170.000000
720.000000  216.000000
661.000000  216.000000
497.000000  467.000000
277.000000  422.000000
  0.000000  649.000000
  0.000000  216.000000
360.000000    0.000000
720.000000  433.000000
```

```
540.000000   866.000000
  0.000000   433.000000
360.000000   866.000000
 25.884205    33.356106
 62.107395   789.987244
666.573181   848.028259
438.604828   195.870956
402.941040   671.387756
  0.500000     0.500000
720.500000     0.500000
720.500000   866.500000
  0.500000   866.500000
```

4. Run AutoReg to Generate the New GCP Dataset.

Concatonate the ground control points file that was an input to ClipImages and the output intersection points file into a new file.

cat gcp_bothflipped clip.out > gcp_and_clip

gcp_and_clip would look like:

```
167.1560 105.4990 25.8842 33.3561
192.2160 472.9602 62.1073 789.9872
533.9680 454.5222 666.5731 848.0282
398.6695 164.1889 438.6048 195.8709
378.9331 392.4563 402.9410  671.3877
151.611632    95.276201     0.500000     0.500000
556.273869    46.592385   720.500000     0.500000
562.156006   463.521309   720.500000   866.500000
157.493769   512.205125     0.500000   866.500000
```

AutoReg currently deals only with integer values. Parse both gcp_and_clip and DelTriang.out into new files containing only integer values:

awk '{printf("%d %d %d %d\n",$1,$2,$3,$4);}' gcp_and_clip > gcp_and_clip.integer

awk '{printf("%d %d\n", $1, $2);}' DelTriang.out > DelTriang.out.int

AutoReg has several required command line arguments:

AutoReg -i gcp_and_clip.int -o AutoReg.out -c DelTriang.out.int srcfile reffile

-i is the input GCP file
-o is the output (new) GCP datafile
-c is the OCP data file

-srcfile is the name of the directory containing the source data file and its header
-reffile is the name of the directory containing the reference data file and its header

In addition, you may want to specify the following parameters:

-M matching mode; 0 for image correlation, 1 for adjacent pixel difference, and 2 for coded patch
-t for the DN threshold used for finding CCPs. <1 is the fraction of the std. deviation; >1 is the value in DN.

For more information on the application of this algorithm, contact

Dr. Thomas F. Lundeen
Pacific Northwest National Laboratories
Phone: (509) 372-6055 FAX: (509) 372-6397
E-mail: tf_lundeen@pnl.gov

APPENDIX B. AUTOREG SOURCE CODE

AutoReg.c

```
/*+*/
/*
************************************************************
*******
**
**  0) NAME:      AutoReg
**
**  1) PURPOSE:   PRISMS program for autoregistering two
PRISMS
**                data sets, using a set of initial user
define
**                Ground Control Points (GCPs) and the
location
**                for the optimal location for new control
points.
**
**  2) USAGE:
**           AutoReg [-hDv][-m mode] -p pntfile Source
Reference
**                    Output
**
**             -h          : Print out the help message.
**             -D          : Print out the version
information.
**
**  3) ALGORITHM/METHOD:
**         1.
**
**  4) LIMITATIONS:
**
**  5) Create: 8/5/95
**
**  6) By:       Thomas Lundeen
**               Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
************************************************************
*******
*/
/*-*/
#ifndef lint
static char
```

```
rcs_info[] = "$Id$";
#endif

/*  System Includes */

#include <stdio.h>
#include <string.h>

/*  User Includes   */

#include "PRISMSMain.h"
#include "AutoReg.h"

/*  Globals       */

int
Verbose,
Mode,
PatchSize,
SearchLength,
PRISMSErrValue = 0;

float
MatchThreshold,
CCPThreshold;
char
Source[512],
Reference[512],
InputFile[512],
OutputFile[512],
OCPLFile[512],
*ProgramName = "AutoReg",
PRISMSErrMsg[512],
*Description = "To semi-automaticaly register two data
sets.";

Arguments   Args[] = {
    {
        'v',    BooleanArg, OPTIONAL,   0,  1,
            (char *) &Verbose,  "0",
            "\0",   "Display the program status
information."
    },
    {
        ' ',    StringArg,  REQUIRED,   1,  1,
            Source, "\0",
            "source",   "The name of the source data set."
    },
    {
        ' ',    StringArg,  REQUIRED,   1,  1,
```

```
            Reference,  "\0",
            "reference",    "The name of the reference data
set."
    },
    {
        'i',    StringArg,  REQUIRED,   1,  1,
            InputFile,  "\0",
            "input",    "The name of the input ground
control points (GCP) file."
    },
    {
        'o',    StringArg,  REQUIRED,   1,  1,
            OutputFile, "\0",
            "output",   "The name of the output control
points file."
    },
    {
        'c',    StringArg,  REQUIRED,   1,  1,
            OCPLFile,   "\0",
            "ocpl", "The name of the file containing the
optimal control\n\
                        point locations."
    },
    {
        's',    IntArg, OPTIONAL,   1,  1,
            (char *) &PatchSize,    "16",
            "PatchSize",    "The size of the image patches
used."
    },
    {
        'l',    IntArg, OPTIONAL,   1,  1,
            (char *) &SearchLength, "8",
            "length",   "The length of the grid search
used."
    },
    {
        'm',    FloatArg,   OPTIONAL,   1,  1,
            (char *) &MatchThreshold,   "0.8",
            "MatchThreshold",   "The threshold level used to
when comparing \n\
                        image patches."
    },
    {
        't',    FloatArg,   OPTIONAL,   1,  1,
            (char *) &CCPThreshold, "-0.5",
            "CCPThreshold", "The DN threshold used when
finding CCPs.\n\
                        For values less than one, the
threshold is set\n\
```

```
                        to (-CCPThreshold) * Standard
diavation."
    },
    {
        'M', IntArg, OPTIONAL, 1, 1,
            (char *) &Mode, "0",
            "mode", "The comparison mode to be used.\n\
                    0 : Correlation.\n\
                    1 : Apd.\n\
                    2 : Coded Patch."
    },
    {
        ' ', EndArg, 0, 0, 0, "", "", "", ""    }
};


/******************* CopyDS ******************/

int main (argc, argv)
int
argc;
char
*argv[];
{
    CPInfo
        *OCPData = NULL;

    GCPInfo
        *GCPData = NULL;

    DataSetInfo
        source,
        reference;

    int
    ReadArgs();

    /*  Read in the calling arguments
*/

    if (ReadArgs(Args, argc, argv, Description) != 0)
    {
        exit(-1);
    }

    /*  Open the source data set for reading.
*/
    if (!OpenDataSet(Source, &source, ReadData))
    {   /*  Open the reference data set.
*/
```

```
        if (!OpenDataSet(Reference, &reference, ReadData))
        {   /*  Read in the Ground Control Points
*/
            if ((GCPData = RdGCPFile(InputFile)) != NULL)
            {   /* Read in the optimum control point
locations  */
                if ((OCPData = RdOCPFile(OCPLFile)) != NULL)
                {
                    if (!AutoRegProc (&source, &reference,
                        GCPData, OCPData, MatchThreshold,
                        CCPThreshold, PatchSize,
SearchLength,
                        Mode, Verbose))
                    {
                        (void)WrGCPFile (OutputFile,
GCPData);
                    }
                }
            }
            (void) CloseDataSet(&source);
        }
        (void) CloseDataSet(&reference);
    }

    if (GCPData != NULL)
        (void) FreeGCPInfo (GCPData);

    if (OCPData != NULL)
        (void) FreeCPInfo (OCPData);

    if (PRISMSErrValue != 0)
    {
        PRISMSError(PRISMSErrValue, PRISMSErrMsg, NULL);
    }
    exit(PRISMSErrValue);
}
```

```
/*+*/
/*
************************************************************
*******
**
**  0) NAME:     AutoRegProc
**
**  1) PURPOSE:  The main subroutine which processes the
data and
**               finds the new Ground Control Points.
**
**  2) USAGE:
**       int   AutoRegProc(DataSetInfo *source, DataSetInfo
*reference,
**               GCPInfo *GCPData, CPInfo *OCPData,
**               float MatchThreshold, float CCPThreshold,
**               int PatchSize, int MatchLength,  int Mode,
**               int Verbose)
**
**  3) ALGORITHM/METHOD:
**       1.
**
**  4) LIMITATIONS:
**
**  5) Create: 8/5/95
**
**  6) By:     Thomas Lundeen
**             Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif

/*  System Includes */

#include <stdio.h>
#include <malloc.h>
#include <sys/types.h>
```

```
#include <sys/stat.h>
#include <fcntl.h>

/*  User Includes   */

#include "AutoReg.h"
#include "PRISMSProgDefines.h"
#include "PRISMSErrors.h"

/*  Externals       */

extern int
PRISMSErrValue;

extern char
PRISMSErrMsg[];


/****************** CopyProc ******************/

int AutoRegProc(DataSetInfo *source, DataSetInfo *reference,
GCPInfo *GCPData, CPInfo *OCPData, float MatchThreshold,
float CCPThreshold, int PatchSize, int MatchLength, int
Mode,
int Verbose)
{
    float
    Threshold;

    int
    out,
        yy,
        MinY,
        MaxY,
        xx,
        MinX,
        MaxX,
        HalfLength,
        XMin,
        YMin,
        XMax,
        YMax,
        ii,
        Total,
        GCPCount,
        OCPCount,
        CCPCount,
        LCPCount,
        LCPCntr,
        Found,
```

```
        NewPoint,
        index[4],
        RXSize,
        RYSize,
        RImageSize,
        SXSize,
        SYSize,
        SImageSize,
        SearchRadius,
        SearchRadiusSquared,
        Distance,
        VectorCount,
        PercentCount,
        CCPRefX,
        CCPRefY,
        MaxXRefLoc,
        MaxYRefLoc,
        MaxXSrcLoc,
        MaxYSrcLoc;

    double
    CenterX,
        CenterY;

    CPInfo
        *ocpptr,            /* Pointer to Optimum Control
Point     */
    *ccpptr,            /* Pointer to a Candidate Control
Point */
    *CCPData,           /* List of Candidata Control Points
*/
    LCPData[100];       /* List of CCP's around the current
OCP */

    TriangleInfo
        *Triangles = NULL,  /* Current List of Triangles
*/
    *triptr;            /* Pointer to the current Triangle
*/

    GCPInfo
        *LastGCP = NULL,
        *gcpptr;            /* Pointer to the current GCP
*/

    TransCoef
        ForwardTrans,       /* Forward transformation
coefficents   */
    ReverseTrans;       /* Reverse transformation
coefficents   */
```

```
    float
    Range[2],
        Mean,
        StdDeviation,
        MaxValue,
        MatchValue,
        *SImage,
        *RImage,
        *SPatch,
        *RPatch;

    void
    *SProcessed = NULL,      /* Pointer to the processed
patches     */
    *RProcessed = NULL;

    /*  Setup the I/O buffer
*/

    HalfLength = MatchLength/2;
    SXSize = source->SubsetLengths[X];
    SYSize = source->SubsetLengths[Y];
    SImageSize = SXSize * SYSize;

    if ((SImage = (float *) malloc(SImageSize *
sizeof(float))) == NULL)
    {
        PRISMSErrValue  = ErrAllocating;
        PRISMSErrMsg[0] = '\0';
        return (TRUE);
    }


    RXSize = reference->SubsetLengths[X];
    RYSize = reference->SubsetLengths[Y];
    RImageSize = RXSize * RYSize;

    if ((RImage = (float *) malloc(RImageSize *
sizeof(float))) == NULL)
    {
        PRISMSErrValue  = ErrAllocating;
        PRISMSErrMsg[0] = '\0';
        return (TRUE);
    }

    if ((RPatch = (float *) malloc(PatchSize* PatchSize *
sizeof(float))) == NULL)
    {
        PRISMSErrValue  = ErrAllocating;
```

```
        PRISMSErrMsg[0] = '\0';
        return (TRUE);
    }

    if ((SPatch = (float *) malloc(PatchSize* PatchSize *
sizeof(float))) == NULL)
    {
        PRISMSErrValue  = ErrAllocating;
        PRISMSErrMsg[0] = '\0';
        return (TRUE);
    }

    /*  Count the number of GCPs and OCPs
*/

    for (GCPCount= 0, gcpptr = GCPData; gcpptr->Next !=
NULL; GCPCount++,
        gcpptr = gcpptr->Next);

    for (OCPCount= 0, ocpptr = OCPData; ocpptr != NULL;
OCPCount++,
        ocpptr = ocpptr->Next);

    /*  Read in the source image
*/

    for (ii = 0; ii < 4; ii++)
        index[ii] = 0;

    if (Verbose)
        printf ("Reading the source image.\n");

    if (ReadPlane (source, SImage, index, XY))
        return(TRUE);

    if (Verbose)
        printf ("Reading the reference image.\n");

    if (ReadPlane (reference, RImage, index, XY))
        return(TRUE);

    /*  Find the bounding box for the reference image
*/

    FindBBox(GCPData, &XMin, &XMax, &YMin, &YMax, FALSE);

    if (Verbose)
        printf ("Bounding box size of: Xmin = %d  Ymin = %d
Xmax = %d  Ymax = %d \n",
            XMin, YMin, XMax, YMax);
```

```
    /*  Find the data threshold
*/

    FindRange(reference, RImage, &Range, &Mean,
&StdDeviation,
        RImageSize);

    printf ("Std Deviation =  %g\n", StdDeviation);

    if (CCPThreshold > 1.0)
        Threshold = CCPThreshold;
        else
        Threshold = CCPThreshold * StdDeviation;

    if (Verbose)
        printf ("Setting the threshold value to %g\n",
Threshold);

    /*  Find Candidate Control Points within the Ref. Image
*/

    if (Verbose)
        printf ("Finding a set of canidate control
points.\n");

    if ((CCPData = FindCCPS(RImage, RXSize, RYSize, XMin,
XMax,
        YMin, YMax, &CCPCount, Threshold)) == NULL)
    {
        PRISMSErrValue = -1;
        (void) strcpy (PRISMSErrMsg,
            "No canidate control points found in the
image.");
        return (TRUE);
    }


    if (Verbose)
        printf ("A total of %d canidate control points were
found.\n",
            CCPCount);
    /*
        for (ii = 1, ccpptr = CCPData; ccpptr != NULL;
            ccpptr = ccpptr->Next, ii++)
            {
            printf ("%4d:  %5d   %5d   %5g\n", ii,
                ccpptr->XLoc, ccpptr->YLoc, ccpptr-
>InterestValue);
            }
```

```
    */

    /*  Find the Search Radius
*/

    FindSearchRadius (OCPData, OCPCount, &SearchRadius,
        &SearchRadiusSquared);

    if (Verbose)
        printf ("Node Search Radius = %d\n", SearchRadius);

    /*  Searching for a GCP around each OCP
*/

    if (Verbose)
        printf ("Seaching for GCP around each OCP
locations.\n");


    NewPoint    = TRUE;

    for (ocpptr = OCPData; ocpptr != NULL; ocpptr = ocpptr-
>Next)
    {
        MaxValue    = MatchThreshold;
        MaxXRefLoc  = -1;
        MaxYRefLoc  = -1;
        MaxXSrcLoc  = -1;
        MaxYSrcLoc  = -1;

        LCPCount = FindLCPs(GCPData, ocpptr, CCPData,
LCPData,
            SearchRadiusSquared);

        if (Verbose)
        {
            printf ("%3d CCPs found within radius %d of the
",
                LCPCount, SearchRadius);
            printf ("OCP at location %5d, %5d\n",
                ocpptr->XLoc, ocpptr->YLoc);
        }

        fflush(stdout);

        /*  Triangulate the Current Set of Ground Control
Points (if needed)    */

        if (NewPoint)
        {
```

```
            if (Triangles != NULL)
                FreeTriInfo(Triangles);

            Triangles = Triangulate2D(GCPData);

            /*  Calculate the transforms for each of the
triangles      */

            for (triptr = Triangles; triptr != NULL;
                triptr = triptr->Next)
            {
                CalcTransform (triptr);
            }
        }

        /*  Search the candidate control points
*/

        for (LCPCntr = 0; LCPCntr < LCPCount; LCPCntr++)
        {
            CCPRefX = LCPData[LCPCntr].XLoc;
            CCPRefY = LCPData[LCPCntr].YLoc;

            /*  Find out which triangle its in
*/

            for (triptr = Triangles; (triptr != NULL) &&
                (!PointInTriangle(triptr, CCPRefX,
CCPRefY));
                triptr = triptr->Next);

            if (triptr != NULL)
            {
                CenterX = (triptr->Forward.aX * (double)
CCPRefX)
                    + (triptr->Forward.bX * (double)
CCPRefY)
                    + triptr->Forward.cX;

                CenterY = (triptr->Forward.aY * (double)
CCPRefX)
                    + (triptr->Forward.bY * (double)
CCPRefY)
                    + triptr->Forward.cY;

                if (!ReadRefPatch(reference, RImage, RPatch,
PatchSize,
                    CenterX, CenterY, triptr))
                {
```

```
                    if ((RProcessed = ProcessPatch
(RProcessed, RPatch,
                        PatchSize, Mode)) == NULL)
                    {
                        strcpy (PRISMSErrMsg,
                            "Processing Reference Patch");
                        PRISMSErrValue = -1;
                        return (TRUE);
                    }

                    /* Grid Search      */

                    MinX = ((int) CenterX + 0.5) -
HalfLength;
                    MaxX = ((int) CenterX + 0.5) +
HalfLength;
                    MinY = ((int) CenterY + 0.5) -
HalfLength;
                    MaxY = ((int) CenterY + 0.5) +
HalfLength;
                    /*
                                        MinX = (int) CenterX
+ 0.5;
                                        MinY = (int) CenterY
+ 0.5;
                                        MaxX = MinX + 1;
                                        MaxY = MinY + 1;
                    */


                    for (yy = MinY; yy < MaxY; yy++)
                    {
                        for (xx = MinX; xx < MaxX; xx++)
                        {
                            if (!ReadSrcPatch(source,
SImage, SPatch,
                                PatchSize, xx, yy))
                            {
                                if ((SProcessed =
ProcessPatch (SProcessed, SPatch,
                                    PatchSize, Mode)) ==
NULL)
                                {
/* code folded from here */
    strcpy (PRISMSErrMsg,
        "Processing Source Patch");
    PRISMSErrValue = -1;
    return (TRUE);
/* unfolding */
                                }
```

```
                                    MatchValue = ComparePatch
(RProcessed,
                                        SProcessed, Mode,
PatchSize);

                                    /*fprintf ( stderr,
"MatchValue: %f\n", MatchValue );*/
                                    if (MatchValue > MaxValue)
                                    {
/* code folded from here */
    MaxValue    = MatchValue;
    MaxXSrcLoc  = xx;
    MaxYSrcLoc  = yy;
    MaxXRefLoc  = CCPRefX;
    MaxYRefLoc  = CCPRefY;
/* unfolding */
                                    }
                                }
                            }
                        }
                    }
                }
            }

            if (MaxXSrcLoc >= 0)
            {       /* Add a new GCP        */
                if (Verbose)
                {
                    printf ("Adding point at (%4d, %4d) - (%4d,
%4d)\n",
                        MaxXRefLoc, MaxYRefLoc, MaxXSrcLoc,
MaxYSrcLoc);
                }
                NewPoint = TRUE;
                if (LastGCP == NULL)
                {
                    for (gcpptr = GCPData; gcpptr != NULL;
                        gcpptr = gcpptr->Next)
                        LastGCP = gcpptr;
                }

                LastGCP->Next = (GCPInfo *)
malloc(sizeof(GCPInfo));
                LastGCP       = LastGCP->Next;
                LastGCP->SrcX = MaxXSrcLoc;
                LastGCP->SrcY = MaxYSrcLoc;
                LastGCP->RefX = MaxXRefLoc;
                LastGCP->RefY = MaxYRefLoc;
                LastGCP->Next = NULL;
```

```
        }
        else
            NewPoint = FALSE;
    }

    /* need to remove CCP from CCP list? May or may not be
worth it. */
    (void) free(SPatch);
    (void) free(RPatch);
    return (FALSE);
}
```

```
/*+*/
/*
*************************************************************
*******
**
**  0) NAME:      AutoRegUtil
**
**  1) PURPOSE:
**
**
**  2) USAGE:
**          AutoRegUtil();
**
**
**  3) ALGORITHM/METHOD:
**        1.
**
**  4) LIMITATIONS:
**
**  5) Create: 24/5/95
**
**  6) By:       Thomas Lundeen
**              Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
*************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif

/*  System Includes */

#include <stdio.h>
#include <stdlib.h>

/*  User Includes   */

/*  Externals       */

extern int
PRISMSErrValue;
```

```
extern char
PRISMSErrMsg[];


int *IntVect(int ncols)
{
    int *vectptr;
    if ((vectptr = (int *) memalign(32, ncols *
sizeof(int))) == NULL)
    {
        fprintf(stderr,"\nIntVect: Unable to allocate
storage for ivector");
        exit(1);
    }
    return vectptr;
}


int **IntMatrix(int nrows, int ncols)
{
    int i0;
    int **matptr;
    if (nrows<2) nrows = 2;
    if (ncols<2) ncols = 2;
    if ((matptr = (int **) memalign(32,nrows * sizeof(int
*))) == NULL)
    {
        fprintf(stderr,"\nIntMatrix: Unable to allocate
storage for **imatrix");
        exit(1);
    }
    if ((matptr[0] = (int *) memalign(32,nrows * ncols *
sizeof(int))) == NULL)
    {
        fprintf(stderr,"\nIntMatrix: Unable to allocate
storage for imatrix[]");
        exit(1);
    }
    for (i0=1; i0<nrows; i0++) matptr[i0] = matptr[0] + i0 *
ncols;
    return matptr;
}


double **DoubleMatrix(int nrows, int ncols)
{
    int i0;
    double **matptr;
    if (nrows<2) nrows = 2;
```

```
    if (ncols<2) ncols = 2;
    if ((matptr = (double **) memalign(32,nrows *
sizeof(double *))) == NULL)
    {
        fprintf(stderr,"\nDoubleMatrix: Unable to allocate
storage for **dmatrix");
        exit(1);
    }
    if ((matptr[0] = (double *) memalign(32,nrows * ncols *
sizeof(double))) == NULL)
    {
        fprintf(stderr,"\nDoubleMatrix: Unable to allocate
storage for dmatrix[]");
        exit(1);
    }
    for (i0=1; i0<nrows; i0++) matptr[i0] = matptr[0] + i0 *
ncols;
    return matptr;
}

void FreeVecti(int *vectptr)
{
    free(vectptr);
}


void FreeMatrixi(int **matptr)
{
    free(matptr[0]);
    free(matptr);
}


void FreeMatrixd(double **matptr)
{
    free(matptr[0]);
    free(matptr);
}
```

CalcTransform.c

```
/*+*/
/*
************************************************************
*******
**
**  0) NAME:      CalcTransform
**
**  1) PURPOSE:
**
**
**  2) USAGE:
**          CalcTransform();
**
**
**  3) ALGORITHM/METHOD:
**        1.
**
**  4) LIMITATIONS:
**
**  5) Create: 31/5/95
**
**  6) By:       Thomas Lundeen
**              Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif


/*  System Includes */
#include <stdio.h>
#include <math.h>

/*  User Includes   */

#include "AutoReg.h"

/****************** CalcTransform ******************/
```

```
void    CalcTransform(TriangleInfo *TriData)
{
    double
    RefX[3],
        RefY[3],
        SrcX[3],
        SrcY[3],
        denom,
        ax,
        ay,
        bx,
        by,
        cx,
        cy;

    int
    ii;

    /*  Load the point array for a foward Transform
*/

    for (ii = 0; ii < 3; ii++)
    {
        RefX[ii]  = TriData->Vertices[ii]->RefX;
        RefY[ii]  = TriData->Vertices[ii]->RefY;
        SrcX[ii]  = TriData->Vertices[ii]->SrcX;
        SrcY[ii]  = TriData->Vertices[ii]->SrcY;
    }

    /*  Calculate the Coefficents
*/

    denom = ((RefX[2] - RefX[0]) * (RefY[1] - RefY[0])) -
        ((RefX[1] - RefX[0]) * (RefY[2] - RefY[0]));


    ax = (((SrcX[2] - SrcX[0]) * (RefY[1] - RefY[0])) -
        ((RefY[2] - RefY[0]) * (SrcX[1] - SrcX[0])))/denom;

    ay = (((SrcY[2] - SrcY[0]) * (RefY[1] - RefY[0])) -
        ((RefY[2] - RefY[0]) * (SrcY[1] - SrcY[0])))/denom;

    bx = (((SrcX[1] - SrcX[0]) * (RefX[2] - RefX[0])) -
        ((SrcX[2] - SrcX[0]) * (RefX[1] - RefX[0])))/denom;

    by = (((SrcY[1] - SrcY[0]) * (RefX[2] - RefX[0])) -
        ((SrcY[2] - SrcY[0]) * (RefX[1] - RefX[0])))/denom;

    cx= SrcX[0] - bx*RefY[0] - ax*RefX[0];
```

```
    cy= SrcY[0] - by*RefY[0] - ay*RefX[0];

    /*  Tranfer the coordinate back
*/

    TriData->Forward.aX = ax;
    TriData->Forward.bX = bx;
    TriData->Forward.cX = cx;
    TriData->Forward.aY = ay;
    TriData->Forward.bY = by;
    TriData->Forward.cY = cy;

    /*  Load the point array for a Reverse Transform
*/

    for (ii = 0; ii < 3; ii++)
    {
        RefX[ii]  = TriData->Vertices[ii]->SrcX;
        RefY[ii]  = TriData->Vertices[ii]->SrcY;
        SrcX[ii]  = TriData->Vertices[ii]->RefX;
        SrcY[ii]  = TriData->Vertices[ii]->RefY;
    }

    /*  Calculate the Coefficents
*/

    denom = ((RefX[2] - RefX[0]) * (RefY[1] - RefY[0])) -
        ((RefX[1] - RefX[0]) * (RefY[2] - RefY[0]));


    ax = (((SrcX[2] - SrcX[0]) * (RefY[1] - RefY[0])) -
        ((RefY[2] - RefY[0]) * (SrcX[1] - SrcX[0])))/denom;

    ay = (((SrcY[2] - SrcY[0]) * (RefY[1] - RefY[0])) -
        ((RefY[2] - RefY[0]) * (SrcY[1] - SrcY[0])))/denom;

    bx = (((SrcX[1] - SrcX[0]) * (RefX[2] - RefX[0])) -
        ((SrcX[2] - SrcX[0]) * (RefX[1] - RefX[0])))/denom;

    by = (((SrcY[1] - SrcY[0]) * (RefX[2] - RefX[0])) -
        ((SrcY[2] - SrcY[0]) * (RefX[1] - RefX[0])))/denom;

    cx= SrcX[0] - bx*RefY[0] - ax*RefX[0];

    cy= SrcY[0] - by*RefY[0] - ay*RefX[0];

    /*  Tranfer the coordinate back
*/

    TriData->Reverse.aX = ax;
```

```
    TriData->Reverse.bX = bx;
    TriData->Reverse.cX = cx;
    TriData->Reverse.aY = ay;
    TriData->Reverse.bY = by;
    TriData->Reverse.cY = cy;

}
```

```
/*+*/
/*
*************************************************************
*******
**
**  0) NAME:      CodeMatchProc
**
**  1) PURPOSE:
**
**
**  2) USAGE:
**          CodeMatchProc();
**
**
**  3) ALGORITHM/METHOD:
**      1.
**
**  4) LIMITATIONS:
**
**  5) Create: 15/6/95
**
**  6) By:       Aaron Andrews
**              Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
*************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif


/*  System Includes */

/*  User Includes   */

/*  Globals       */

/*  Externals       */

extern int
PRISMSErrValue;
```

```
extern char
PRISMSErrMsg[];

/******************* CodeMatchProc ******************/
float CodeMatchProc(unsigned char *RefPatch, unsigned char
*SrcPatch, int PatchSize)
{
    unsigned
    ii,
        CodeSize,
        Sum = 0;

    float
    PerfectMatch;

    unsigned char
    MatchBits;



    CodeSize = PatchSize - 2;                    /* these
should be */
    PerfectMatch = CodeSize * CodeSize * 4.0;   /* made
global     */

    for (ii = 0; ii < CodeSize * CodeSize; ii++)
    {
        MatchBits = (*RefPatch) & (*SrcPatch);
        Sum += BitCount(MatchBits);
        RefPatch++;
        SrcPatch++;
    }
    return ((double)Sum/PerfectMatch);
}
```

```
/*+*/
/*
************************************************************
*******
**
**  0) NAME:      CodeUtil
**
**  1) PURPOSE:
**
**  4) LIMITATIONS:
**
**  5) Create: 27/6/95
**
**  6) By:       Aaron Andrews
**              Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif


/*  System Includes */

/*  User Includes   */

/*  Globals       */

/*  Externals       */

extern int
PRISMSErrValue;

extern char
PRISMSErrMsg[];


/****************** CodeUtil ******************/

int BitCount(unsigned char x)
```

```
{
    int b;

    for (b=0; x != 0; x >>=1) if (x & 01) b++;
    return (b);
}

void ShellSort2(float arr1[], unsigned arr2[])
{
    unsigned n=8,i,j,inc;
    unsigned v, w;

    inc=13;
    do {
        inc /= 3;
        for (i=inc+1;i<=n;i++) {
            v = arr1[i];
            w = arr2[i];
            j = i;
            while (arr1[j-inc]>v) {
                arr1[j]=arr1[j-inc];
                arr2[j]=arr2[j-inc];
                j -= inc;
                if (j <= inc) break;
            }
            arr1[j] = v;
            arr2[j] = w;
        }
    } while (inc >1);
}
```

ComparePatch.c

```
/*+*/
/*
*************************************************************
*******
**
**  0) NAME:     ComparePatch
**
**  1) PURPOSE:  To compare to image patches to find the
degree
**               of simularity between the two.
**
**  2) USAGE:
**          float   ComparePatch (float *, float *, int,
int);
**
**          factor = ComparePatch (RefPatch, SrcPatch, Mode,
**                      PatchSize);
**
**  3) ALGORITHM/METHOD:
**        1.
**
**  4) LIMITATIONS:
**
**  5) Create: 6/6/95
**
**  6) By:      Thomas Lundeen
**             Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
*************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif


/*  System Includes */

#include <stdio.h>
#include <math.h>
#include <malloc.h>
```

```
/*  User Includes    */

#include "AutoReg.h"

float CorMatchProc(float *, float *, int);

/****************** ComparePatch ******************/

float   ComparePatch (void *RefPatch, void * SrcPatch, int
Mode,
int PatchSize)
{
    float
    MatchValue = 0.0;

    switch (Mode)
    {
    case APD: /* Adjacent Pixel Difference */
        MatchValue = ApdMatchProc(RefPatch, SrcPatch,
PatchSize);
        break;
    case CODE: /* coded patch method */
        MatchValue = CodeMatchProc(RefPatch, SrcPatch,
PatchSize);
        break;
    case CORRELATE:
    default:            /* Correlation              */
        MatchValue = CorMatchProc(RefPatch, SrcPatch,
PatchSize);
        break;
    }

    return (MatchValue);
}


/****************** CorMatchProc ******************/

float CorMatchProc(float *RefPatch, float *SrcPatch, int
PatchSize)
{
    float
    NumPixels,
        *RefPtr,
        *RefEnd,
        *SrcPtr,
        Value,
        Value2,
        SumSqRef = 0.0,
        SumSqSrc = 0.0,
```

```
        SumCross = 0.0;


    RefPtr    = RefPatch;
    SrcPtr    = SrcPatch;
    RefEnd    = RefPatch + (PatchSize * PatchSize);
    NumPixels = PatchSize * PatchSize;

    while (RefPtr < RefEnd)
    {
        Value     = *SrcPtr;
        SumSqSrc += Value * Value;

        Value2     = *RefPtr;
        SumSqRef += Value2 * Value2;


        SumCross += Value * Value2;

        RefPtr++;
        SrcPtr++;
    }

    if ((SumSqRef > 0.0) && (SumSqSrc > 0.0))
        Value = SumCross/((float) sqrt((double) SumSqRef *
SumSqSrc));
        else
        Value = 0.0;

    return (Value);
}
```

CPCompare.c

```
/*+*/
/*
*************************************************************
*******
**
**  0) NAME:      CPCompare
**
**  1) PURPOSE:  Subroutine for comparing the interest
values for
**               sorting the CP data.
**
**  2) USAGE:
**          CPCompare();
**
**
**  3) ALGORITHM/METHOD:
**        1.
**
**  4) LIMITATIONS:
**
**  5) Create: 23/5/95
**
**  6) By:      Thomas Lundeen
**              Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
*************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif


/*  System Includes */

#include <stdio.h>

/*  User Includes   */

#include "AutoReg.h"

/****************** CPCompare ******************/
```

```
int CPCompare (CPInfo *CPPtr2, CPInfo *CPPtr1)
{
    return ((int) CPPtr1->InterestValue -
        (int) CPPtr2->InterestValue);
}
```

```
/*+*/
/*
************************************************************
*******
**
**  0) NAME:      FindBBox
**
**  1) PURPOSE:  To find the bounding box for a set of
ground
**               control points.
**
**  2) USAGE:
**          void FindBBox(GCPInfo *, int *, int *, int *,
int *);
**
**          FindBBox (GCPData, XMin, XMax, YMin, YMax);
**
**  3) ALGORITHM/METHOD:
**        1.
**
**  4) LIMITATIONS:
**
**  5) Create: 10/5/95
**
**  6) By:       Thomas Lundeen
**              Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif


/*  System Includes */

#include <stdio.h>

/*  User Includes   */
```

```
#include "AutoReg.h"


void FindBBox(GCPData, XMin, XMax, YMin, YMax, FLAG)
GCPInfo *GCPData;
int     *XMin,
*XMax,
*YMin,
*YMax,
FLAG;
{
    GCPInfo
        *gcpptr;
    int
    ii,
        xmin,
        xmax,
        ymin,
        ymax;

    if (FLAG)
    {
        gcpptr = GCPData;

        xmin = gcpptr->SrcX;
        xmax = gcpptr->SrcX;
        ymin = gcpptr->SrcY;
        ymax = gcpptr->SrcY;

        for (gcpptr = gcpptr->Next; gcpptr != NULL; gcpptr =
gcpptr->Next)
        {
            if (gcpptr->SrcX > xmax)
                xmax = gcpptr->SrcX;

            if (gcpptr->SrcX < xmin)
                xmin = gcpptr->SrcX;

            if (gcpptr->SrcY > ymax)
                ymax = gcpptr->SrcY;

            if (gcpptr->SrcY < ymin)
                ymin = gcpptr->SrcY;

        }
    }
    else
    {
        gcpptr = GCPData;
```

```
        xmin = gcpptr->RefX;
        xmax = gcpptr->RefX;
        ymin = gcpptr->RefY;
        ymax = gcpptr->RefY;

        for (gcpptr = gcpptr->Next; gcpptr != NULL; gcpptr =
gcpptr->Next)
        {
            if (gcpptr->RefX > xmax)
                xmax = gcpptr->RefX;

            if (gcpptr->RefX < xmin)
                xmin = gcpptr->RefX;

            if (gcpptr->RefY > ymax)
                ymax = gcpptr->RefY;

            if (gcpptr->RefY < ymin)
                ymin = gcpptr->RefY;
        }
    }

    *XMin = xmin;
    *XMax = xmax;
    *YMin = ymin;
    *YMax = ymax;
}
```

FindCCPS.c

```
/*+*/
/*
*************************************************************
*******
**
**  0) NAME:      FindCCPS
**
**  1) PURPOSE:  To sieve through the image and find all of
the
**               canidate control points.
**
**  2) USAGE:
**          CCPINfo *FindCCPS(image, thres, xmin, xmax,
ymin,
**                  ymax, &count);
**
**  3) ALGORITHM/METHOD:
**       1.
**
**  4) LIMITATIONS:
**
**  5) Create: 11/5/95
**
**  6) By:       Thomas Lundeen
**              Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
*************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif


/*  System Includes */

#include <stdio.h>
#include <malloc.h>
#include <math.h>

/*  User Includes   */
```

```
#include "PRISMSErrors.h"
#include "AutoReg.h"

/*  Globals       */

/*  Externals       */

extern int
PRISMSErrValue;

extern char
PRISMSErrMsg[];


/****************** FindCCPS ******************/

CPInfo *FindCCPS(float *image, int width, int hieght, int
xmin,
int xmax, int ymin, int ymax, int *count, float Threshold)
{
    CPInfo
        *First   = NULL,
        *Last    = NULL,
        *CCPList = NULL;

    int
    CCPCount = 0,
    match_flag,
    ii,
    jj,
    kk,
    nn,
    win_offset[8];

    unsigned char
    BV;

    float
    Gzero,
    DN,
    IV,
    *pixel_ptr,
    *pixel_ptr2;

    int
    win3x3[8][2]= {
        1,-1,1,0,1,1,0,1,-1,1,-1,0,-1,-1,0,-1   };

    unsigned char
```

```
    patterns[24]= {
        10,40,160,130,26,104,161,134,11,44,176,
                    196,131,
14,56,224,15,60,240,195,30,120,225,135 };

    /* precompute window offset */

    for (kk=0; kk<8; kk++)
        win_offset[kk] = win3x3[kk][1]*width+win3x3[kk][0];

    for (jj=ymin;jj<= ymax;jj++)
    {
        pixel_ptr = image+jj*width;

        for (ii=xmin;ii<=xmax;ii++)
        {
            BV= 0;
            pixel_ptr2 = pixel_ptr+ii;
            Gzero      = *pixel_ptr2;

            /* examine DN values in a 3x3 subwindow about
central pixel */

            for (kk=0;kk<8;kk++)
            {
                DN= *(pixel_ptr2+win_offset[kk]);
                /*
                *   if DN diff btw center pix & adj pix abv
specifd thresh,
                *   sum position code
                */
                if (fabs(Gzero-DN) > Threshold)
                    BV+= (unsigned char)(1<<kk);
            }
            /*
            *   check to see if calculated bit value matches
an acceptable
            *   pattern
            */
            match_flag= 0;

            for (kk=0;(kk<24) && (match_flag == 0);kk++)
            {
                if (BV==patterns[kk])
                {   /* match found                          */
                    if ((CCPList = (CPInfo *)
malloc(sizeof(CPInfo))) == NULL)
                    {
                        PRISMSErrValue = NULL;
                        (void) strcpy (PRISMSErrMsg,
```

```
                        "for CCP data.");
                    return(NULL);
                }

                IV= 0.0;
                for (nn=0; nn<8; nn++)
                    IV+= fabs(Gzero-
*(pixel_ptr2+win_offset[nn]));
                CCPList->XLoc          = ii;
                CCPList->YLoc          = jj;
                CCPList->InterestValue = IV;
                CCPList->Next          = NULL;

                if (Last != NULL)
                    Last->Next = CCPList;
                    else
                    First = CCPList;

                Last = CCPList;
                CCPCount++;
            }
        }
    }
}

*count = CCPCount;

return (First);
}
```

```
/*+*/
/*
************************************************************
*******
**
**   0) NAME:      FindLCPs
**
**   1) PURPOSE:   To find all of the CCPs around the selected
**                 GCP within the specified radius squared.
**
**   2) USAGE:
**           FindLCPs();
**
**
**   3) ALGORITHM/METHOD:
**         1.
**
**   4) LIMITATIONS:
**
**   5) Create: 22/5/95
**
**   6) By:       Thomas Lundeen
**               Pacific Northwest Laboratory
**
**   7) $Revision$
**
**   8) $Log$
**
************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif

/*  System Includes */

#include <stdio.h>

/*  User Includes   */

#include "AutoReg.h"

/******************* FindLCPs ******************/
```

```
int FindLCPs(GCPInfo *GCPData, CPInfo *ocpptr, CPInfo
*CCPData,
CPInfo *LCPList, int SearchRadiusSquared)
{
    int
    LCPCount = 0,
        Found,
        CCPDist,
        OCPXLoc,
        OCPYLoc,
        XDist,
        YDist;

    CPInfo
        *ccpptr;

    GCPInfo
        *gcpptr;

    Found = 0;

    OCPXLoc = ocpptr->XLoc;
    OCPYLoc = ocpptr->YLoc;

    for (gcpptr = GCPData; gcpptr != NULL; gcpptr = gcpptr-
>Next)
    {
        if ((OCPXLoc == gcpptr->RefX) && (OCPYLoc == gcpptr-
>RefY))
        {
            Found = 1;
            break;
        }
    }

    if (!Found)
    {   /* Find all the CCPs around the OCP     */
        /* within the desired search radius     */

        for (ccpptr = CCPData; ccpptr != NULL; ccpptr =
ccpptr->Next)
        {
            XDist   = (OCPXLoc - ccpptr->XLoc);
            YDist   = (OCPYLoc - ccpptr->YLoc);
            CCPDist = (XDist * XDist) + (YDist * YDist);

            if ((CCPDist < SearchRadiusSquared) && (LCPCount
< 100))
            {
                LCPList[LCPCount].XLoc = ccpptr->XLoc;
```

```
                    LCPList[LCPCount].YLoc = ccpptr->YLoc;
                    LCPList[LCPCount].InterestValue = ccpptr-
>InterestValue;
                    LCPCount++;
                }
            }
        }

        /*  Sort the List relative to the interest value
*/

        qsort (LCPList, LCPCount, sizeof(CPInfo), CPCompare);

        return (LCPCount);
}
```

FindRange.c

```
/*+*/
/*
************************************************************
*******
**
**   0) NAME:      FindRange
**
**   1) PURPOSE:
**
**
**   2) USAGE:
**           FindRange();
**
**
**   3) ALGORITHM/METHOD:
**         1.
**
**   4) LIMITATIONS:
**
**   5) Create: 30/2/94
**
**   6) By:      Thomas F. Lundeen
**               Battelle Laboratories
**
**   7) $Revision$
**
**   8) $Log$
**
************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif

/*  System Includes */

#include <stdio.h>
#include <math.h>

/*  User Includes   */

#include "DataSetInfo.h"

/******************* FindRange ******************/
```

```
FindRange(dsinfo, data, Range, Mean, StdDeviation, Points)
DataSetInfo
*dsinfo;
float
*data,
*Range,
*Mean,
*StdDeviation;
int
Points;
{
    register float
    *ptr,
        *end,
        mean,
        min,
        max,
        dif,
        value;

    float
    dmin,
        dmax;

    register int
    PCount;

    ptr = data;
    end = data + Points;

    min    =  dsinfo->Maximum;
    max    =  dsinfo->Minimum;
    mean   = 0.0;
    dif    = 0.0;
    PCount = 0;
    dmin   = dsinfo->Minimum;
    dmax   = dsinfo->Maximum;

    while (ptr < end)
    {
        value = *ptr;
        ptr++;

        if ((value > dmin) && (value < dmax))
        {
            if (value < min)
                min = value;

            if (value > max)
```

```
                max = value;

            mean += value;
            PCount++;
        }
    }

    if (PCount == 1)
    {
        PRISMSError(-1, "No valid data found.\n", NULL);
        exit(-1);
    }

    mean = mean/((float) PCount);
    ptr  = data;

    while (ptr < end)
    {
        value = *ptr;
        ptr++;

        if ((value > dmin) && (value < dmax))
        {
            value = value - mean;
            dif  += value * value;
        }
    }

    *StdDeviation = (float) sqrt(((double) dif)/
        ((double) PCount - 1));

    *Mean    = mean;

    Range[0] = min;
    Range[1] = max;
}
```

FindSearchRadius.c

```
/*+*/
/*
************************************************************
*******
**
**  0) NAME:      FindSearchRadius
**
**  1) PURPOSE:
**
**
**  2) USAGE:
**          FindSearchRadius();
**
**
**  3) ALGORITHM/METHOD:
**        1.
**
**  4) LIMITATIONS:
**
**  5) Create: 11/5/95
**
**  6) By:       Thomas Lundeen
**              Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif

/*  System Includes */

#include <math.h>

/*  User Includes   */

#include "AutoReg.h"

/****************** FindSearchRadius ******************/

void    FindSearchRadius(CPInfo *OCPData, int Count,
```

```
int *SearchRadius, int *SearchRadiusSquared)
{
    CPInfo
        *iiptr,
        *jjptr;

    float
    avg_sep;

    int
    min[5],
        sep,
        kk;


    /* Find average node-node distance for 5 points closest
to each node */
    avg_sep=0.0;

    for (iiptr=OCPData; iiptr != NULL; iiptr = iiptr->Next)
    {
        for (kk=0;kk<5;kk++)
            min[kk] = 1e5;

        for (jjptr=OCPData; jjptr != NULL; jjptr = jjptr-
>Next)
        {
            if (jjptr != iiptr)
            {
                sep = ((iiptr->XLoc - jjptr->XLoc)
                    * (iiptr->XLoc - jjptr->XLoc))
                    + ((iiptr->YLoc - jjptr->YLoc)
                    * (iiptr->YLoc - jjptr->YLoc));

                if (sep<=min[0])
                {
                    min[4]=min[3];
                    min[3]=min[2];
                    min[2]=min[1];
                    min[1]=min[0];
                    min[0]=sep;
                }
                else if (sep <=min[1])
                {
                    min[4]=min[3];
                    min[3]=min[2];
                    min[2]=min[1];
                    min[1]=sep;
                }
                else if (sep<=min[2])
```

```
                    {
                        min[4]=min[3];
                        min[3]=min[2];
                        min[2]=sep;
                    }
                    else if (sep<=min[3])
                    {
                        min[4]=min[3];
                        min[3]=sep;
                    }
                    else if (sep<=min[4])
                    {
                        min[4]=sep;
                    }
                }
            }
            for (kk=0;kk<5;kk++)
                avg_sep += sqrt(min[kk]);
        }

        avg_sep /= (5.0 * (float) Count);

        kk = (int)(1.2 * 0.5 * avg_sep);

        *SearchRadius = kk;
        *SearchRadiusSquared = kk * kk;
    }
```

FreeCPInfo.c

```
/*+*/
/*
************************************************************
*******
**
**  0) NAME:     FreeCPInfo
**
**  1) PURPOSE:  To step throung and delete the allocated
memory
**               for the control data.
**
**  2) USAGE:
**          void FreeGCPInfo(GCPInfo *);
**
**
**  3) ALGORITHM/METHOD:
**        1.
**
**  4) LIMITATIONS:
**        None
**
**  5) Create: 23/5/95
**
**  6) By:     Thomas Lundeen
**             Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif


/*  System Includes */

#include <stdio.h>
#include <malloc.h>

/*  User Includes   */

#include "AutoReg.h"
```

```
/******************** FreeGCPInfo ********************/

void FreeCPInfo(CPInfo *CPData)
{
    CPInfo
        *Next;

    while (CPData != NULL)
    {
        Next = CPData->Next;
        (void) free(CPData);
        CPData = Next;
    }
}
```

```
/*+*/
/*
**********************************************************
*******
**
**  0) NAME:      FreeGCPInfo
**
**  1) PURPOSE:  To step throung and delete the allocated
memory
**               for the ground control data.
**
**  2) USAGE:
**          void FreeGCPInfo(GCPInfo *);
**
**
**  3) ALGORITHM/METHOD:
**        1.
**
**  4) LIMITATIONS:
**        None
**
**  5) Create: 23/5/95
**
**  6) By:       Thomas Lundeen
**              Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
**********************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif


/*  System Includes */

#include <stdio.h>
#include <malloc.h>

/*  User Includes   */

#include "AutoReg.h"
```

```
/******************** FreeGCPInfo ********************/

void FreeGCPInfo(GCPInfo *GCPData)
{
    GCPInfo
        *Next;

    while (GCPData != NULL)
    {
        Next = GCPData->Next;
        (void) free(GCPData);
        GCPData = Next;
    }
}
```

FreeTriInfo.c

```
/*+*/
/*
*************************************************************
*******
**
**  0) NAME:     FreeTriInfo
**
**  1) PURPOSE:  To step throung and delete the allocated
memory
**               for the ground control data.
**
**  2) USAGE:
**          void FreeTriInfo(TriInfo *);
**
**
**  3) ALGORITHM/METHOD:
**        1.
**
**  4) LIMITATIONS:
**        None
**
**  5) Create: 23/5/95
**
**  6) By:      Thomas Lundeen
**             Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
*************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif


/*  System Includes */

#include <stdio.h>
#include <malloc.h>

/*  User Includes   */

#include "AutoReg.h"
```

```
/******************* FreeTriInfo ******************/

void FreeTriInfo(TriangleInfo *TriData)
{
    TriangleInfo
        *Next;


    while (TriData != NULL)
    {
        Next = TriData->Next;
        (void) free(TriData);
        TriData = Next;
    }
}
```

PointInTriangle.c

```
/*+*/
/*
************************************************************
*******
**
**  0) NAME:      PointInTriangle
**
**  1) PURPOSE:  To determine if the specified point is
contained
**               withing the triangle as defined by the 4
next
**               GCP entries.
**
**  2) USAGE:
**          PointInTriangle();
**
**
**  3) ALGORITHM/METHOD:
**        1.
**
**  4) LIMITATIONS:
**
**  5) Create: 25/5/95
**
**  6) By:       Thomas Lundeen
**              Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif

/*  System Includes */
/*#include <stdio.h>

/*  User Includes   */

#include "AutoReg.h"
```

```
/******************* PointInTriangle *******************/

int PointInTriangle (TriangleInfo *Triptr, int XLoc, int
YLoc)
{
    int
    ret = FALSE,
        DeltaX1,
        DeltaX2,
        DeltaY1,
        DeltaY2,
        DeltaAx,
        DeltaAy;

    float
    Diff1,
        Diff2,
        Diff3;

    /*  Calculate the distances between the first vertex
*/
    /*  and the other vertices in the triangle
*/

    /*fprintf ( stderr, "     0: RefX, RefY = ( %d, %d )\n",
Triptr->Vertices[0]->RefX, Triptr->Vertices[0]->RefY );

    fprintf ( stderr, "     1: RefX, RefY = ( %d, %d )\n",
Triptr->Vertices[1]->RefX, Triptr->Vertices[1]->RefY );
    fprintf ( stderr, "     2: RefX, RefY = ( %d, %d )\n",
Triptr->Vertices[2]->RefX, Triptr->Vertices[2]->RefY );
    */
    DeltaX1 = Triptr->Vertices[1]->RefX - Triptr-
>Vertices[0]->RefX;
    DeltaY1 = Triptr->Vertices[1]->RefY - Triptr-
>Vertices[0]->RefY;

    DeltaX2 = Triptr->Vertices[2]->RefX - Triptr-
>Vertices[0]->RefX;
    DeltaY2 = Triptr->Vertices[2]->RefY - Triptr-
>Vertices[0]->RefY;

    /*  Calculate the distance between the point and the
*/
    /*  first vertex
*/

    DeltaAx = XLoc - Triptr->Vertices[0]->RefX;
    DeltaAy = YLoc - Triptr->Vertices[0]->RefY;
```

```
    /*  Determine the
*/

    /*fprintf ( stderr, "Ax %d, Ay %d, X1 %d, Y1 %d, X2 %d,
Y2 %d\n", DeltaAx, DeltaAy, DeltaX1, DeltaY1, DeltaX2, De
        ltaY2 );
    */

    Diff1 = (float) (DeltaX1 * DeltaY2) - (DeltaX2 *
DeltaY1);
    Diff2 = (float) ((DeltaAx * DeltaY2) - (DeltaAy *
DeltaX2))/Diff1;
    Diff3 = (float) ((DeltaAx * DeltaY1) - (DeltaAy *
DeltaX1))/Diff1;

    /*fprintf ( stderr, "Diff1: %f, %f, %f\n", Diff1, Diff2,
Diff3 );
    */

    if ((Diff2 >= 0) && (Diff2 <= 1) && (Diff3 >= 0) &&
(Diff3 <= 1))
        ret = TRUE;

    return (ret);
}
```

```
/*+*/
/*
************************************************************
*******
**
**  0) NAME:      ProcessApdPatch
**
**  1) PURPOSE:
**          Creates a feature vector for the image patch
Patch
**          using the adjacent pixel difference method.
**
**  2) USAGE:
**          ProcessApdPatch();
**
**
**  3) ALGORITHM/METHOD:
**        1.
**
**  4) LIMITATIONS:
**
**  5) Create: 15/6/95
**
**  6) By:       Aaron Andrews
**              Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif


/*  System Includes */
#include <stdio.h>
#include <malloc.h>

/*  User Includes   */
```

```
#include "AutoReg.h"

/*  Globals       */

/*  Externals       */

extern int
PRISMSErrValue;

extern char
PRISMSErrMsg[];


/****************** ProcessApdPatch ******************/

void *ProcessApdPatch(float *Processed, float *Patch, int
PatchSize)
{
    void
    *RPatch = NULL;

    int
    ix, /* counter in x direction */
    iy; /* counter in y direction */

    float
    *PixelPtr,
        DiffSum,  /* sum of adjacent pixel differences for a
row/column */
    DNSum,    /* sum of DNs across a row/column */
    *FVecPtr; /* Pointer to APD feature vector */


    if (Processed != NULL) /* See if FVecPtr has already
been alloc'ed */
        FVecPtr = Processed;
        else
    {
        if ((FVecPtr = (float *)malloc(2 * PatchSize *
sizeof(float))) == NULL)
        {
            /* print error message */
            return (RPatch);
        }
    }
    RPatch = FVecPtr;
    /* First compute APD component for each row in the patch
*/
    for (iy = 0; iy < PatchSize; iy++)
    {
```

```
        PixelPtr = Patch + iy * PatchSize;
        DiffSum = 0.0;
        DNSum = 0.0;
        for (ix = 0; ix < PatchSize - 1; ix++)
        {
            DiffSum += fabs(*PixelPtr - *(PixelPtr + 1));
            DNSum += *PixelPtr;
            PixelPtr++;
        }

        DNSum += *PixelPtr;

        if (DNSum > 0.0)
            *FVecPtr = DiffSum / DNSum;
            else
            *FVecPtr = 0.0;

        FVecPtr++;
    }

    /* Now compute APD component for each column in the
patch */
    for (ix = 0; ix < PatchSize; ix++)
    {
        PixelPtr = Patch + ix;
        DiffSum = 0.0;
        DNSum = 0.0;
        for (iy = 0; iy < PatchSize - 1; iy++)
        {
            DiffSum += fabs(*PixelPtr - *(PixelPtr +
PatchSize));
            DNSum += *PixelPtr;
            PixelPtr += PatchSize;
        }

        DNSum += *PixelPtr;

        if (DNSum > 0.0)
            *FVecPtr = DiffSum / DNSum;
            else
            *FVecPtr = 0.0;

        FVecPtr++;
    }
    return (RPatch);
}
```

```
/*+*/
/*
*************************************************************
*******
**
**  0) NAME:     ProcessCodePatch
**
**  1) PURPOSE:
**          Encodes each pixel in the image patch (Patch)
according
**          to the positions of the 4 most different
surrounding
**          pixels.
**
**  2) USAGE:
**          ProcessCodePatch();
**
**
**  3) ALGORITHM/METHOD:
**        1.
**
**  4) LIMITATIONS:
**
**  5) Create: 15/6/95
**
**  6) By:      Aaron Andrews
**             Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
*************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif


/*  System Includes */
#include <stdio.h>
#include <malloc.h>

/*  User Includes   */
```

```
#include "AutoReg.h"
#include "PRISMSErrors.h"

extern int
PRISMSErrValue;

extern char
PRISMSErrMsg[];


/****************** ProcessCodePatch ******************/

void *ProcessCodePatch(unsigned char *Processed, float
*Patch, int PatchSize)
{
    void
    *RPatch = NULL;

    int
    ix,
        iy,
        kk,
        idx,
        count;

    unsigned
    loc[8];

    unsigned char
    *CodedPtr,
        *TmpPtr,
        code;

    float
    *PixelPtr,
        diff[8],
        DN;


    if (Processed != NULL)        /* See if CodedPtr was
already malloc'ed */
        CodedPtr = Processed;
        else
    {
        if ((CodedPtr = (unsigned char *)malloc((PatchSize-
2) * (PatchSize - 2)
            * sizeof(unsigned char))) == NULL)
        {
            PRISMSErrValue = ErrAllocating;
            PRISMSErrMsg[0] = '\0';
```

```
            return (RPatch);
        }
    }

    RPatch = CodedPtr;
    /* Encode all but the perimeter pixels */
    for (iy = 1; iy < PatchSize - 1; iy++)
    {
        for (ix = 1; ix < PatchSize - 1; ix++)
        {
            PixelPtr = Patch + iy * PatchSize + ix;
            DN = *PixelPtr;     /* Central DN value */
            PixelPtr -= PatchSize + 1; /* Upper left pixel
relative to DN */
            count = 0;

            /* Compute differences between central pixel
value and */
            /* the eight surrounding pixel values.
*/
            for (idx = 0; idx < 3; idx++)
            {
                diff[count] = fabs(*PixelPtr - DN);
                loc[count] = count++;
                PixelPtr++;
            }

            PixelPtr += PatchSize - 3;
            diff[count] = fabs(*PixelPtr - DN);
            loc[count] = count++;

            PixelPtr += 2;
            diff[count] = fabs(*PixelPtr - DN);
            loc[count] = count++;

            PixelPtr += PatchSize - 2;
            for (idx = 0; idx < 3; idx++)
            {
                diff[count] = fabs(*PixelPtr - DN);
                loc[count] = count++;
                PixelPtr++;
            }

            /* Sort differences to find the four pixels most
different */
            /* from the central pixel.
*/
            ShellSort2(diff-1, loc-1);

            code = 0;
```

```
            for (kk = 4; kk < 8; kk++)
                code += (unsigned char)(1 << loc[kk]);  /*
Encode according  */
            /* to position.      */
            *CodedPtr++ = code;
        }
    }
    return (RPatch);
}
```

ProcessPatch.c

```
/*+*/
/*
************************************************************
*******
**
**  0) NAME:      ProcessPatch
**
**  1) PURPOSE:
**
**
**  2) USAGE:
**          ProcessPatch();
**
**
**  3) ALGORITHM/METHOD:
**        1.
**
**  4) LIMITATIONS:
**
**  5) Create: 13/6/95
**
**  6) By:       Thomas Lundeen
**              Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif


/*  System Includes */

#include <stdio.h>
#include <malloc.h>

/*  User Includes   */

#include "AutoReg.h"
```

```
/****************** ProcessPatch ******************/

void    *ProcessPatch (void *Processed, float *Patch, int
PatchSize, int Mode)
{
    void
    *RPatch = NULL;

    switch (Mode)
    {
    case CORRELATE:
        RPatch = Patch;
        break;
    case APD:
        RPatch = ProcessApdPatch(Processed, Patch,
PatchSize);
        break;
    case CODE:
        RPatch = ProcessCodePatch(Processed, Patch,
PatchSize);
        break;
    default:
        RPatch = Patch;
    }

    return (RPatch);
}
```

```
/*+*/
/*
************************************************************
*******
**
**  0) NAME:      RdGCPFile
**
**  1) PURPOSE:  To read in the ground control point file.
**
**  2) USAGE:
**          GCPInfo *RdGCPFile();
**
**
**  3) ALGORITHM/METHOD:
**        1.
**
**  4) LIMITATIONS:
**
**  5) Create: 9/5/95
**
**  6) By:       Thomas Lundeen
**              Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif

/*  System Includes */

#include <stdio.h>
#include <malloc.h>

/*  User Includes   */

#include "AutoReg.h"
#include "PRISMSErrors.h"
```

```
/*  Externals       */

extern int
PRISMSErrValue;

extern char
PRISMSErrMsg[];


/******************* RdGCPFile *******************/

GCPInfo *RdGCPFile(char *FileName)
{
    FILE
        *fp;

    char
    Line[512];

    int
    RefX,
        RefY,
        SrcX,
        SrcY,
        count;

    GCPInfo
        *First = NULL,
        *Last  = NULL,
        *GCPData;

    if ((fp = fopen (FileName, "r")) == NULL)
    {
        (void) strcpy(PRISMSErrMsg,
            "Error opening the ground control point file:
");
        (void) strcat(PRISMSErrMsg, FileName);
        PRISMSErrValue = -1;
        return (NULL);
    }

    while (fgets(Line, 512, fp) != NULL)
    {
        if ((Line[0] != '#') && (sscanf(Line, "%d %d %d %d",
            &RefX, &RefY, &SrcX, &SrcY) == 4))
        {
            if ((GCPData = (GCPInfo *) malloc
(sizeof(GCPInfo)))
                == NULL)
            {
```

```
                (void) strcpy(PRISMSErrMsg, "for the GCP
data.");
                PRISMSErrValue = ErrAllocating;
                return (NULL);
            }

            GCPData->RefX = RefX;
            GCPData->RefY = RefY;
            GCPData->SrcX = SrcX;
            GCPData->SrcY = SrcY;
            GCPData->Next = NULL;

            if (Last != NULL)
                Last->Next = GCPData;
                else
                First = GCPData;

            Last = GCPData;
        }
    }

    (void) fclose(fp);
    return (First);
}
```

```
/*+*/
/*
*************************************************************
*******
**
**  0) NAME:      RdOCPFile
**
**  1) PURPOSE:  To read in the optimum control point file.
**
**  2) USAGE:
**          int RdOCPFile();
**
**
**  3) ALGORITHM/METHOD:
**        1.
**
**  4) LIMITATIONS:
**
**  5) Create: 9/5/95
**
**  6) By:       Thomas Lundeen
**              Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
*************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif

/*  System Includes */

#include <stdio.h>
#include <malloc.h>

/*  User Includes   */

#include "AutoReg.h"
#include "PRISMSErrors.h"

/*  Externals       */
```

```
extern int
PRISMSErrValue;

extern char
PRISMSErrMsg[];


/******************* RdOCPFile ******************/

CPInfo  *RdOCPFile(FileName)
char    *FileName;
{
    FILE
        *fp;

    char
    Line[512];

    int
    XLoc,
        YLoc;

    CPInfo
        *Last,
        *First,
        *CPData;

    if ((fp = fopen (FileName, "r")) == NULL)
    {
        (void) strcpy(PRISMSErrMsg,
            "Error opening the optimum control point file:
");
        (void) strcat(PRISMSErrMsg, FileName);
        PRISMSErrValue = -1;
        return (NULL);
    }

    Last   = NULL;
    CPData = NULL;
    First  = NULL;

    while (fgets(Line, 512, fp) != NULL)
    {
        if ((Line[0] != '#') && (sscanf(Line, "%d %d",
            &XLoc, &YLoc) == 2))
        {
            if ((CPData = (CPInfo *) malloc
(sizeof(CPInfo))) == NULL)
            {
```

```
                (void) strcpy(PRISMSErrMsg, "for the OCP
data.");
                PRISMSErrValue = ErrAllocating;
                return (NULL);
            }
            CPData->XLoc          = XLoc;
            CPData->YLoc          = YLoc;
            CPData->InterestValue = 0.0;
            CPData->Next          = NULL;

            if (Last != NULL)
                Last->Next = CPData;
                else
                First = CPData;

            Last = CPData;
        }
    }

    (void) fclose (fp);
    return (First);
}
```

```
/*+*/
/*
*************************************************************
*******
**
**  0) NAME:      ReadRefPatch
**
**  1) PURPOSE:
**
**
**  2) USAGE:
**          ReadRefPatch();
**
**
**  3) ALGORITHM/METHOD:
**        1.
**
**  4) LIMITATIONS:
**
**  5) Create: 5/6/95
**
**  6) By:       Thomas Lundeen
**              Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
*************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif

/*  User Includes   */

#include "AutoReg.h"

/****************** ReadRefPatch ******************/

int ReadRefPatch(DataSetInfo *dsinfo, float *Image, float
*Patch,
int PatchSize, double CenterX, double CenterY,
TriangleInfo *triptr)
```

```
{

    int
    yy,
        xx,
        XSize,
        YSize,
        XCoor,
        YCoor;

    double
    XStart,
        XPos,
        YPos;

    float
    *imageptr,
        *patchptr;

    patchptr = Patch;

    CenterX -= (double) (PatchSize >> 1);
    CenterY -= (double) (PatchSize >> 1);
    XSize    = dsinfo->SubsetLengths[X];
    YSize    = dsinfo->SubsetLengths[Y];

    XStart = CenterX;
    for (yy = 0; yy < PatchSize; yy++)
    {
        CenterX = XStart;
        for (xx = 0; xx < PatchSize; xx++)
        {
            XPos = (triptr->Reverse.aX * CenterX) +
                (triptr->Reverse.bX * CenterY) +
                triptr->Reverse.cX;

            YPos = (triptr->Reverse.aY * CenterX) +
                (triptr->Reverse.bY * CenterY) +
                triptr->Reverse.cY;

            XCoor = (int) XPos + 0.5;
            YCoor = (int) YPos + 0.5;

            if ((XCoor < 0) || (XCoor >= XSize) ||
                (YCoor < 0) || (YCoor >= YSize))
            {
                return (TRUE);
            }

            imageptr = Image + ((YCoor * XSize) + XCoor);
```

```
            *patchptr = *imageptr;
            patchptr++;
            CenterX += 1.0;
        }
        CenterY += 1.0;
    }

    return (FALSE);
}
```

```
/*+*/
/*
*************************************************************
*******
**
**  0) NAME:      ReadSrcPatch
**
**  1) PURPOSE:
**
**
**  2) USAGE:
**          ReadSrcPatch();
**
**
**  3) ALGORITHM/METHOD:
**        1.
**
**  4) LIMITATIONS:
**
**  5) Create: 5/6/95
**
**  6) By:       Thomas Lundeen
**              Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
*************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif

/*  User Includes   */

#include "AutoReg.h"

/****************** ReadSrcPatch ******************/

int ReadSrcPatch(DataSetInfo *dsinfo, float *Image, float
*Patch,
int PatchSize, int CenterX, int CenterY)
{
```

```
    int
    HighX,
        HighY,
        XSize,
        YSize,
        ii,
        jj;

    float
    *imageptr,
        *patchptr;


    /*  Check to make sure all of the data is within the
bounds of  */
    /*  the image.
*/

    CenterX  -= PatchSize >> 1;
    CenterY  -= PatchSize >> 1;
    HighX     = CenterX + PatchSize;
    HighY     = CenterY + PatchSize;

    XSize = dsinfo->SubsetLengths[X];
    YSize = dsinfo->SubsetLengths[Y];

    if ((CenterX < 0) || (CenterY < 0) ||
        (HighX >= XSize) || (HighY >= YSize))
    {
        return (TRUE);
    }

    patchptr = Patch;
    for (ii = CenterY; ii < HighY; ii++)
    {
        imageptr = Image + ((ii*XSize) + CenterX);
        for (jj = CenterX; jj < HighX; jj++)
        {
            *patchptr = *imageptr;
            imageptr++;
            patchptr++;
        }
    }

    return (FALSE);
}
```

```
/*+*/
/*
*************************************************************
*******
**
**  0) NAME:     Triangulate2D
**
**  1) PURPOSE:  To find either the 2D convex hull or the 2D
**               Delaunay triangulation of the list of input
**               points.  Based on:
**                  nosort.c
**                  Watson, D.F., "Computing the n-
dimensional
**                  Delaunay tesellation with application to
Voronoi
**                  prototypes", The Computer J., 24(2), p
167-162
**
**  2) USAGE:
**          Triangulate2D();
**
**
**  3) ALGORITHM/METHOD:
**        1.
**
**  4) LIMITATIONS:
**
**  5) Create: 18/5/95
**
**  6) By:     John S. Risch
**             Pacific Northwest Laboratory
**    Modified:
**             Thomas F. Lundeen
**
**  7) $Revision$
**
**  8) $Log$
**
*************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
```

```
#endif


/*  System Includes */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*  User Includes   */

#include "AutoReg.h"

/*  Globals       */

#define MIN(a,b)    ((((a) < (b)) ? (a) : (b))
#define MAX(a,b)    ((((a) > (b)) ? (a) : (b))
#define SQ(x)       (x) * (x)
#define RANGE       10.0
#define TSIZE       75
#define BIGNUM      1E37
#define EPSILON     0.00001

/*  Externals       */

/****************** Triangulate2D ******************/

TriangleInfo  *Triangulate2D(GCPInfo *GCPData)
{

    double xx, bgs, **mxy, **wrk, **pts, **ccr;
    int i0, i1, i2, i3, i4, i5, i6, i7, i8, i9, i11, ii[3];
    int dim, dm, dim1, nts, tsz, chl, *id, **tmp, **a3s;
    int jj, XPos, YPos;

    GCPInfo
        *gcpptr;

    TriangleInfo
        *First,
        *Last,
        *outptr;

    int numpts;

    dim    = 2;
    numpts = 0;
    chl    = 0;

    /* allocate and initialize memory  */
```

```
mxy = DoubleMatrix(2, dim);

for (i0=0; i0<dim; i0++)
    mxy[0][i0] = - (mxy[1][i0] = BIGNUM);

dim1 = dim + 1;
wrk = DoubleMatrix(dim, dim1);

for (i0=0; i0<dim; i0++)
    for (i1=0; i1<dim1; i1++)
        wrk[i0][i1] = -RANGE;

for (i0=0; i0<dim; i0++)
    wrk[i0][i0] = RANGE * (3 * dim - 1);

/* copy points from input point list to working array */

for (gcpptr = GCPData, numpts = 0; gcpptr != NULL;
    gcpptr = gcpptr->Next, numpts++);

pts = DoubleMatrix(numpts + dim1, dim);

for (gcpptr = GCPData, i0 = 0; gcpptr != NULL;
    gcpptr = gcpptr->Next, i0++)
{
    pts[i0][0]= gcpptr->RefX;
    pts[i0][1]= gcpptr->RefY;

    for (i1=0; i1<dim; i1++)
    {
        if (mxy[0][i1] < pts[i0][i1])
            mxy[0][i1] = pts[i0][i1];
        if (mxy[1][i1] > pts[i0][i1])
            mxy[1][i1] = pts[i0][i1];
    }
}

for (bgs=0, i0=0; i0<dim; i0++)
{
    mxy[0][i0] -= mxy[1][i0];
    if (bgs < mxy[0][i0])
        bgs = mxy[0][i0];
}

bgs *= EPSILON;
srand(367);

for (i0=0; i0<numpts; i0++)
    for (i1=0; i1<dim; i1++)
```

```
            pts[i0][i1] += bgs * (0.5 - (double)rand() /
0x7fffffff);

    for (i0=0; i0<dim1; i0++)
        for (i1=0; i1<dim; i1++)
            pts[numpts+i0][i1] = mxy[1][i1] + wrk[i1][i0] *
mxy[0][i1];

    FreeMatrixd(mxy);

    for (i1=1, i0=2; i0<dim1; i0++)
        i1 *= i0;

    tsz = TSIZE * i1;
    tmp = IntMatrix(tsz + 1, dim);
    i1 *= (numpts + i1);
    id = IntVect(i1);

    for (i0=0; i0<i1; i0++)
        id[i0] = i0;

    a3s = IntMatrix(i1,dim1);
    ccr = DoubleMatrix(i1,dim1);

    for (a3s[0][0]=numpts, i0=1; i0<dim1; i0++)
        a3s[0][i0] = a3s[0][i0-1] + 1;

    for (ccr[0][dim]=BIGNUM, i0=0; i0<dim; i0++)
        ccr[0][i0] = 0;

    nts = i4 = 1;
    dm  = dim - 1;

    for (i0=0; i0<numpts; i0++)
    {
        i1 = i7 = -1;
        i9 = 0;

        for (i11=0; i11<nts; i11++)
        {
            i1++;

            while (a3s[i1][0] < 0)
                i1++;

            xx = ccr[i1][dim];

            for (i2=0;i2<dim; i2++)
            {
                xx -= SQ(pts[i0][i2] - ccr[i1][i2]);
```

```
                    if (xx<0)
                        goto Corner3;
                }

                i9--;
                i4--;
                id[i4] = i1;

                for (i2=0; i2<dim1; i2++)
                {
                    ii[0] = 0;

                    if (ii[0] == i2)
                        ii[0]++;

                    for (i3=1; i3<dim; i3++)
                    {
                        ii[i3] = ii[i3-1] + 1;

                        if (ii[i3] == i2)
                            ii[i3]++;
                    }

                    if (i7>dm)
                    {
                        i8 = i7;

                        for (i3=0; i3<=i8; i3++)
                        {
                            for (i5=0;i5<dim;i5++)
                                if
(a3s[i1][ii[i5]]!=tmp[i3][i5])
                                    goto Corner1;

                            for (i6=0; i6<dim; i6++)
                                tmp[i3][i6] = tmp[i8][i6];

                            i7--;
                            goto Corner2;
Corner1:
                            ;
                        }
                    }

                    if (++i7 > tsz)
                    {
                        fprintf(stderr,
                            "\ntriangulate_2D: Temporary storage
exceeded - increase TSIZE");
```

```
                    exit(1);
                }

                for (i3=0; i3<dim; i3++)
                    tmp[i7][i3] = a3s[i1][ii[i3]];

Corner2:
                ;
            }

            a3s[i1][0] = -1;

Corner3:
            ;
        }

        for (i1=0; i1<=i7; i1++)
        {
            if (!(chl && tmp[i1][0] < numpts))
            {
                for (i2=0; i2<dim; i2++)
                {
                    for (wrk[i2][dim]=0, i3=0; i3<dim; i3++)
                    {
                        wrk[i2][i3] = pts[tmp[i1][i2]][i3] -
pts[i0][i3];
                        wrk[i2][dim] += wrk[i2][i3] *

(pts[tmp[i1][i2]][i3]+pts[i0][i3])/2;
                    }
                }

                if (dim < 3)
                {
                    xx = wrk[0][0] * wrk[1][1] - wrk[1][0] *
wrk[0][1];
                    ccr[id[i4]][0] = (wrk[0][2] * wrk[1][1]
- wrk[1][2]
                        * wrk[0][1]) / xx;

                    ccr[id[i4]][1] = (wrk[0][0] * wrk[1][2]
- wrk[1][0]
                        * wrk[0][2]) / xx;
                }
                else
                {
                    xx = (wrk[0][0] * (wrk[1][1] * wrk[2][2]
                        - wrk[2][1] * wrk[1][2]))
                        - (wrk[0][1] * (wrk[1][0] *
wrk[2][2]
```

```
                        - wrk[2][0] * wrk[1][2]))
                        + (wrk[0][2] * (wrk[1][0] *
wrk[2][1]
                        - wrk[2][0] * wrk[1][1]));

                    ccr[id[i4]][0]
=((wrk[0][3]*(wrk[1][1]*wrk[2][2]
                        -wrk[2][1]*wrk[1][2]))
                        -(wrk[0][1]*(wrk[1][3]*wrk[2][2]
                        -wrk[2][3]*wrk[1][2]))
                        +(wrk[0][2]*(wrk[1][3]*wrk[2][1]
                        -wrk[2][3]*wrk[1][1]))) / xx;


ccr[id[i4]][1]=((wrk[0][0]*(wrk[1][3]*wrk[2][2]
                        -wrk[2][3]*wrk[1][2]))
                        -(wrk[0][3]*(wrk[1][0]*wrk[2][2]
                        -wrk[2][0]*wrk[1][2]))
                        +(wrk[0][2]*(wrk[1][0]*wrk[2][3]
                        -wrk[2][0]*wrk[1][3]))) / xx;


ccr[id[i4]][2]=((wrk[0][0]*(wrk[1][1]*wrk[2][3]
                        -wrk[2][1]*wrk[1][3]))
                        -(wrk[0][1]*(wrk[1][0]*wrk[2][3]
                        -wrk[2][0]*wrk[1][3]))
                        +(wrk[0][3]*(wrk[1][0]*wrk[2][1]
                        -wrk[2][0]*wrk[1][1]))) / xx;
                }

                for (ccr[id[i4]][dim]=0, i2=0; i2<dim; i2++)
                {
                    ccr[id[i4]][dim] += SQ(pts[i0][i2] -
ccr[id[i4]][i2]);
                    a3s[id[i4]][i2] = tmp[i1][i2];
                }

                a3s[id[i4]][dim] = i0;
                i4++;
                i9++;
            }
        }
        nts += i9;
    }

    FreeMatrixd(wrk);
    FreeMatrixi(tmp);
    FreeVecti(id);
    FreeMatrixd(ccr);
```

```
    /* copy points from working arrays to return points
lists */

    First = NULL;
    Last  = NULL;
    i0 = -1;
    for (i11=0; i11<nts; i11++)
    {
        i0++;
        while (a3s[i0][0] < 0)
            i0++;

        if (a3s[i0][0] < numpts)
        {  /* add triangle vertices to output triangle list
*/
            outptr = (TriangleInfo *) malloc (sizeof
(TriangleInfo));

            for (jj = 0; jj < 3; jj++)
            {
                XPos = (int) pts[(int)a3s[i0][jj]][0]+0.5;
                YPos = (int) pts[(int)a3s[i0][jj]][1]+0.5;

                for (gcpptr = GCPData;gcpptr != NULL;
                    gcpptr = gcpptr->Next)
                {
                    if ((gcpptr->RefX == XPos) &&
                        (gcpptr->RefY == YPos))
                    {
                        outptr->Vertices[jj] = gcpptr;
                    }
                }
            }

            outptr->Vertices[3] = outptr->Vertices[0];

            outptr->Next        = NULL;
            outptr->Forward.aX  = 0.0;
            outptr->Forward.bX  = 0.0;
            outptr->Forward.cX  = 0.0;
            outptr->Forward.aY  = 0.0;
            outptr->Forward.bY  = 0.0;
            outptr->Forward.cY  = 0.0;
            outptr->Reverse.aX  = 0.0;
            outptr->Reverse.bX  = 0.0;
            outptr->Reverse.cX  = 0.0;
            outptr->Reverse.aY  = 0.0;
            outptr->Reverse.bY  = 0.0;
            outptr->Reverse.cY  = 0.0;
            outptr->Reverse.cY  = 0.0;
```

```
                if (Last != NULL)
                    Last->Next = outptr;
                    else
                    First  = outptr;
                Last = outptr;
            }
        }

        FreeMatrixd(pts);
        FreeMatrixi(a3s);

        return (First);
    }
```

```
/*+*/
/*
/************************************************************
********
**
**  0) NAME:     WrGCPFile
**
**  1) PURPOSE:  To output a set of ground control points to
a
**               ascii file.
**
**
**  2) USAGE:
**          WrGCPFile();
**
**
**  3) ALGORITHM/METHOD:
**        1.
**
**  4) LIMITATIONS:
**
**  5) Create: 9/5/95
**
**  6) By:      Thomas Lundeen
**             Pacific Northwest Laboratory
**
**  7) $Revision$
**
**  8) $Log$
**
************************************************************
*******
*/
/*-*/
#ifndef lint
static char
sccs_info[] = "$Id$";
#endif

/*  System Includes */

#include <stdio.h>

/*  User Includes   */
```

```
#include "AutoReg.h"
#include "PRISMSErrors.h"

/*  Externals       */

extern int
PRISMSErrValue;

extern char
PRISMSErrMsg[];

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

/****************** WrGCPFile ******************/

int WrGCPFile (Name, GCPData)
char
Name[];
GCPInfo
*GCPData;
{
    FILE
        *fp;

    GCPInfo
        *gcpptr;

    int
    ret = FALSE;


    if ((fp = fopen (Name, "w")) == NULL)
    {
        ret = TRUE;
        (void) strcpy (PRISMSErrMsg,
            "Creating the output GCP file: ");
        (void) strcat (PRISMSErrMsg, Name);

        PRISMSErrValue = -1;
    }
    else
    {
        fprintf(fp,
```

```
            "# Ground Control Point file created by the
autoregistration routine\n");

        for  (gcpptr = GCPData; gcpptr != NULL; gcpptr =
gcpptr->Next)
        {
            fprintf (fp, "%6d   %6d  %6d  %6d\n",
                gcpptr->RefX, gcpptr->RefY,
                gcpptr->SrcX, gcpptr->SrcY);
        }

        (void) fclose(fp);
    }

    return (ret);
}
```

```
/*+*/
/*
*************************************************************
*******
**
**  0) Name           : AutoReg.h
**
**  1) Description    : Defines the various subroutines and
data
**                      structures for the autoregistration
program.
**
**  2) $Revision$
**
**  3) $Log$
**
*************************************************************
*******
*/
/*-*/

/*   User Includes                                          */

#include "DataSetInfo.h"

/*   Control point information                              */

typedef struct {
     int
          XLoc,
          YLoc;
     float
          InterestValue;
     void
          *Next;
     } CPInfo;

/*   Ground Control Point Pairs
     */

typedef struct {
     int
          SrcX,
          SrcY,
          RefX,
```

```
          RefY;
     void
          *Next;
     } GCPInfo;

/*   Transform Coefficents
     */

typedef struct {
     double
          aX,
          bX,
          cX,
          aY,
          bY,
          cY;
     } TransCoef;

/*   Triangle
     */

typedef struct {
     GCPInfo
          *Vertices[4];

     TransCoef
          Forward,
          Reverse;

     void
          *Next;
     } TriangleInfo;

/*   Program Definitions
     */

#define   FORWARD   1
#define   REVERSE   0
#define   CORRELATE 0
#define   APD  1
#define   CODE 2

/*   Boolean Definitions
     */
#ifndef   TRUE
#define   TRUE      1
#endif

#ifndef   FALSE
#define   FALSE     0
```

```
#endif

int AutoRegProc(DataSetInfo *, DataSetInfo *, GCPInfo *,
CPInfo *,
     float, float, int, int, int, int);

GCPInfo *RdGCPFile (char *);
int        WrGCPFile (char *, GCPInfo *);

CPInfo *RdOCPFile (char *);
CPInfo *FindCCPS(float *, int, int, int, int, int, int, int
*, float);

void FindSearchRadius(CPInfo *, int, int *, int *);
void      FindBBox(GCPInfo *, int *, int *, int *, int *,
int);
int       FindLCPs(GCPInfo *, CPInfo *, CPInfo *, CPInfo *,
int);

TriangleInfo   *Triangulate2D(GCPInfo *);
void           CalcTransform(TriangleInfo *);
int                 PointInTriangle (TriangleInfo *, int,
int);
void                FreeTriInfo(TriangleInfo *);

void      FreeGCPInfo(GCPInfo *);
void      FreeCPInfo(CPInfo *);
int       CPCompare(CPInfo *, CPInfo *);

void FreeVecti(int *);
void FreeMatrixi(int **);
void FreeMatrixd(double **);
int       *IntVect(int ncols);
int       **IntMatrix(int , int);
double  **DoubleMatrix(int , int);

int       ReadRefPatch(DataSetInfo *, float *, float *, int,
          double, double, TriangleInfo *);

int       ReadSrcPatch(DataSetInfo *, float *, float *, int,
int,
          int);
void *ProcessPatch (void *, float *, int, int);
float     ComparePatch(void *, void *, int, int);

void *ProcessApdPatch(float *Processed, float *Patch, int
PatchSize);
float ApdMatchProc(float *RefFVec, float *SrcFVec, int
PatchSize);
```

```
void *ProcessCodePatch(unsigned char *Processed, float
*Patch, int PatchSize);
float CodeMatchProc(unsigned char *RefPatch, unsigned char
*SrcPatch, int PatchSize);
int BitCount(unsigned char x);
void ShellSort2(float arr1[], unsigned arr2[]);
```

APPENDIX C. DESCRIPTION OF THE IMAGE INTERSECTION MODULE

The purpose of ClipImages is to determine the intersection of two images. It is assumed that both image files are unwarped right rectangles (Figure C.1). The algorithm takes the minimum and maximum x,y coordinates of the left, right, top, and bottom bounds of both images and a file of minimally 4 GCP's (Ground Control Points). By convention the origin is assumed to be at the upper left and all input and output has the source image coordinates first followed by the reference image coordinates. All computations are done in double precision floating point.

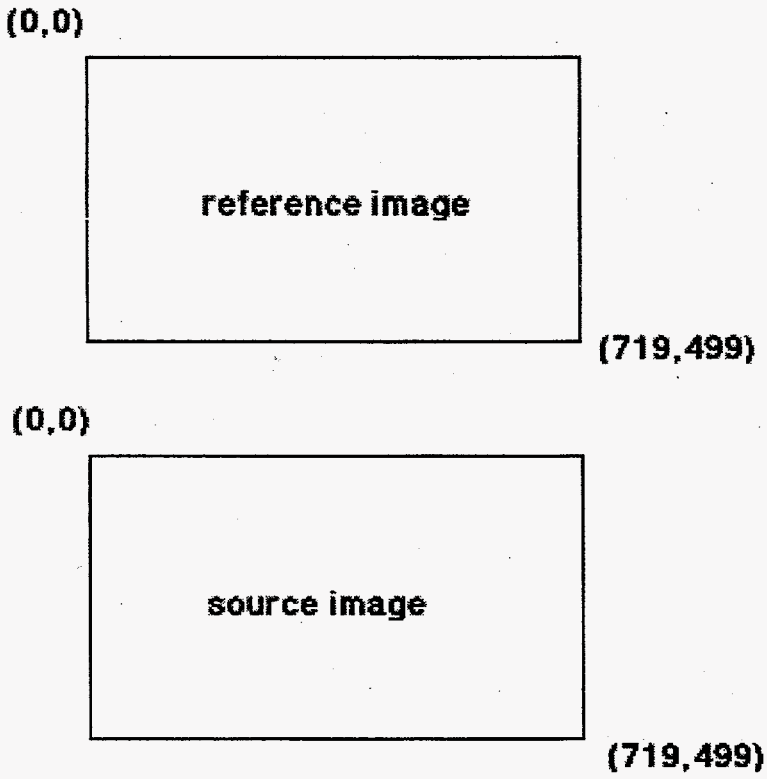


Figure C.1.

The GCP's are used to compute a 1st degree affine transformation from the source to the reference image. This is the reverse of how resampling is normally done. When the transformation was done as usual, reference image into source image space, and the intersection coordinates were found in terms of source image space, then when those coordinates were transformed back to reference image space for output off-of-image round-off errors were encountered. This problem was eliminated by finding the intersection in reference image space (Figure C.2). The accuracy of the source image coordinates depends on how accurately the GCP's were selected.

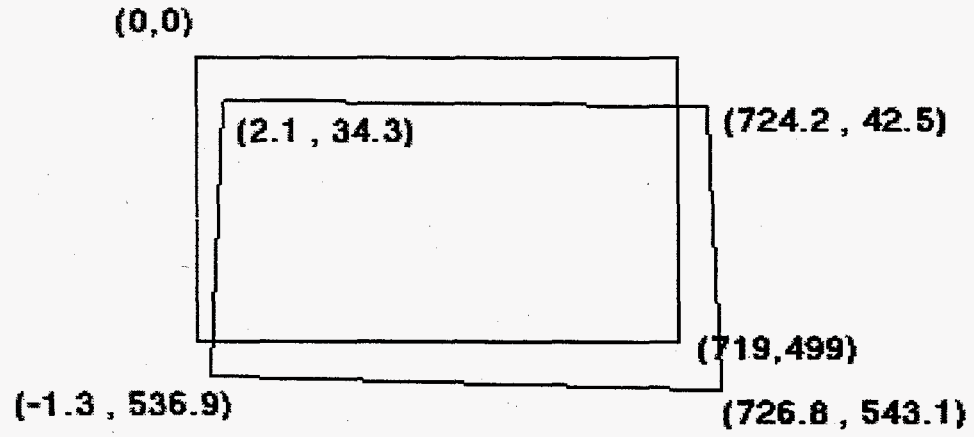


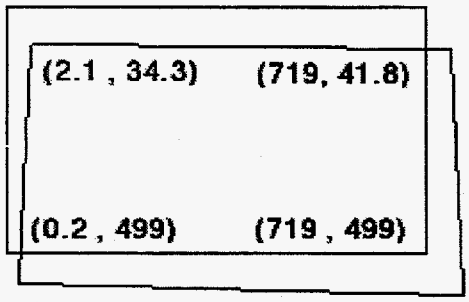
Figure C.2.

Once the source image is warped to reference image space a test is done for the trivial case where one image is contained completely within the other and, if found, the cornerpoints of the contained image are returned both in source and reference image coordinates.

If the trivial case is not found the images are assumed to overlap. The reference image is viewed as a clipping rectangle and the source image's sides are viewed as a series of lines to be clipped against the window. Beginning with the top sides and proceeding clockwise around both images, the source image is clipped against the reference image,

using the Cohen-Sutherland (see Foley et al. 1994) line clipping algorithm, returning the endpoints of the clipped source image line. A list of intersection points in reference image space is kept and, as each side of the source image is clipped, the endpoints are either added to or replace previously found endpoints in the list.

When the last side has been clipped and the list of endpoints is complete it is output both in reference image space and transformed back into source image space (Figure C.3).



Source image clipped to reference image.

Figure C.3.

REFERENCES

Foley, J.D., A. van Dam, S.K. Feiner, J.F. Hughes, R.L. Phillips. 1994. Introduction to Computer Graphics. 2nd ed. Addison Wesley, Reading MA.