IDL Tutorial

Advanced Image Processing

The IDL Intelligent Tools (iTools)

The IDL Intelligent Tools (**iTools**) are a set of interactive utilities that combine data analysis and visualization with the ability to produce presentation quality graphics. The iTools allow users to continue to benefit from the control of a programming language, while enjoying the convenience of a point-and-click environment. There are 7 primary iTool utilities built into the IDL software package. Each of these seven tools is designed around a specific data or visualization type :

- Two and three dimensional plots (line, scatter, polar, and histogram style)
- Surface representations
- Contours
- Image displays
- Mapping
- Two dimensional vector flow fields
- Volume visualizations

The iTools system is built upon an **object-oriented** component framework architecture that is actually comprised of only a single tool, which adapts to handle the data that the user passes to it. The pre-built *iPlot*, *iSurface*, *iContour*, *iMap*, *iImage*, *iVector* and *iVolume* procedures are simply shortcut configurations that facilitate ad hoc data analysis and visualization. Each pre-built tool encapsulates the functionality (data operations, display manipulations, visualization types, etc.) required to handle its specific data type. However, users are not constrained to work with a single data or visualization type within any given tool. Instead, using the iTools system a user can combine multiple dataset visualization types into a single tool creating a hybrid that can provide complex, composite visualizations.

Digital Images and Advanced iImage Operations

IDL provides a powerful environment for image processing and display. Digital images are easily represented as two-dimensional arrays in IDL and can be processed just like any other array. Within an image array the value of each pixel represents the intensity and/or color of that position in the scene. Images of this form are known as sampled or **raster** images, because they consist of a discrete grid of samples. IDL contains many procedures and functions specifically designed for image display and processing. In addition, the *iImage* tool allows the user great flexibility in manipulating and visualizing image data.

In the following exercise, the image from the example data file "*meteorite.bmp*" will be input into IDL. This example data file is located in the "*data*" subfolder.

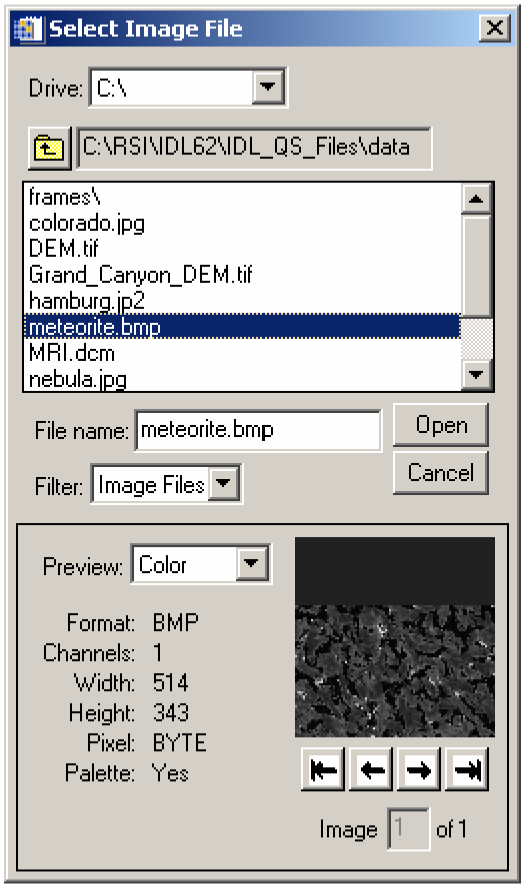
The file is in Windows bitmap format and contains an image of a thin section taken through the Shergotty meteorite that is believed to represent a sample of the surface of Mars. Input the image data into the current IDL session by utilizing the *Import Image* macro built into the IDL Development Environment :

1. IDL> `import_image`
2. Navigate to the "*data*" subfolder and select the "*meteorite.bmp*" file. Information on the image and a small preview will be displayed in the bottom of the *Select Image File* dialog [Fig. 1].
3. Press the "*Open*" button to read the image data into IDL and dismiss the *Import Image* wizard.

Once the *Import Image* macro is finished running the user will be returned to the main IDLDE window where a new variable named "*meteorite_image*" is now present within the current IDL session. The *HELP* procedure can be used to obtain information on this variable :

4. IDL> `HELP, meteorite_image`

```
METEORITE_IMAGE STRUCT    = -> <Anonymous> Array[1]
```



Format: BMP
Channels: 1
Width: 514
Height: 343
Pixel: BYTE
Palette: Yes

Figure 1: The Import Image macro dialog

The output from the *HELP* procedure shows that the "*meteorite_image*" variable is actually a structure containing multiple pieces of data and information read in from the BMP image file. To obtain information on the contents of this structure variable the *HELP* procedure must be executed with the *STRUCTURE* keyword set :

5. IDL> `HELP, meteorite_image, /STRUCTURE`

```
** Structure <14ca450>, 5 tags, length=177112, data
length=177106, refs=1:
   IMAGE           BYTE      Array[514, 343]
   R               BYTE      Array[256]
   G               BYTE      Array[256]
   B               BYTE      Array[256]
   QUERY           STRUCT    -> <Anonymous> Array[1]
```

The actual image data from the BMP file is stored in the *IMAGE* tag of the structure variable, which contains a 2-dimensional array that has 514 columns and 343 rows with an 8-bit (BYTE) data type. The *R*, *G*, and *B* tags within the structure variable are provided to store the color table vectors that can be stored within 8-bit BMP files, which in this case are not necessary since the image is in simple grayscale mode. Furthermore, the *QUERY* tag stores yet another sub-structure that contains other useful information on the BMP image file.

In order to access the data that is stored within the fields of this "*meteorite_image*" structure, the period "." character must be used to reference the tags. For example, to view the information within the *QUERY* sub-structure field the following syntax must be utilized :

```
6. IDL> HELP, meteorite_image.query, /STRUCTURE
   ** Structure <14ca298>, 7 tags, length=40, data length=36,
   refs=3:
      CHANNELS        LONG                     1
      DIMENSIONS      LONG      Array[2]
      HAS_PALETTE     INT                  1
      NUM_IMAGES      LONG                     1
      IMAGE_INDEX     LONG                     0
      PIXEL_TYPE      INT                  1
      TYPE            STRING    'BMP'
```

The *QUERY* field contains some useful information on the BMP image file. In order to access the actual image data stored within the "*meteorite_image*" structure in a manner that does not require a lot of typing, extract the *IMAGE* field of the structure and assign it to a new variable named "*image*" :

```
7. IDL> image = meteorite_image.image
```

Once this is accomplished a new variable is created at the main IDL level that is simply called "*image*" :

```
8. IDL> HELP, image
   IMAGE           BYTE      = Array[514, 343]
```

Now that the image data has been extracted into a simple variable it can be easily visualized by loading it into the *iImage* utility :

```
9. IDL> iImage, image
```

The resulting *IDL iImage* visualization window should look similar to Fig. 2.

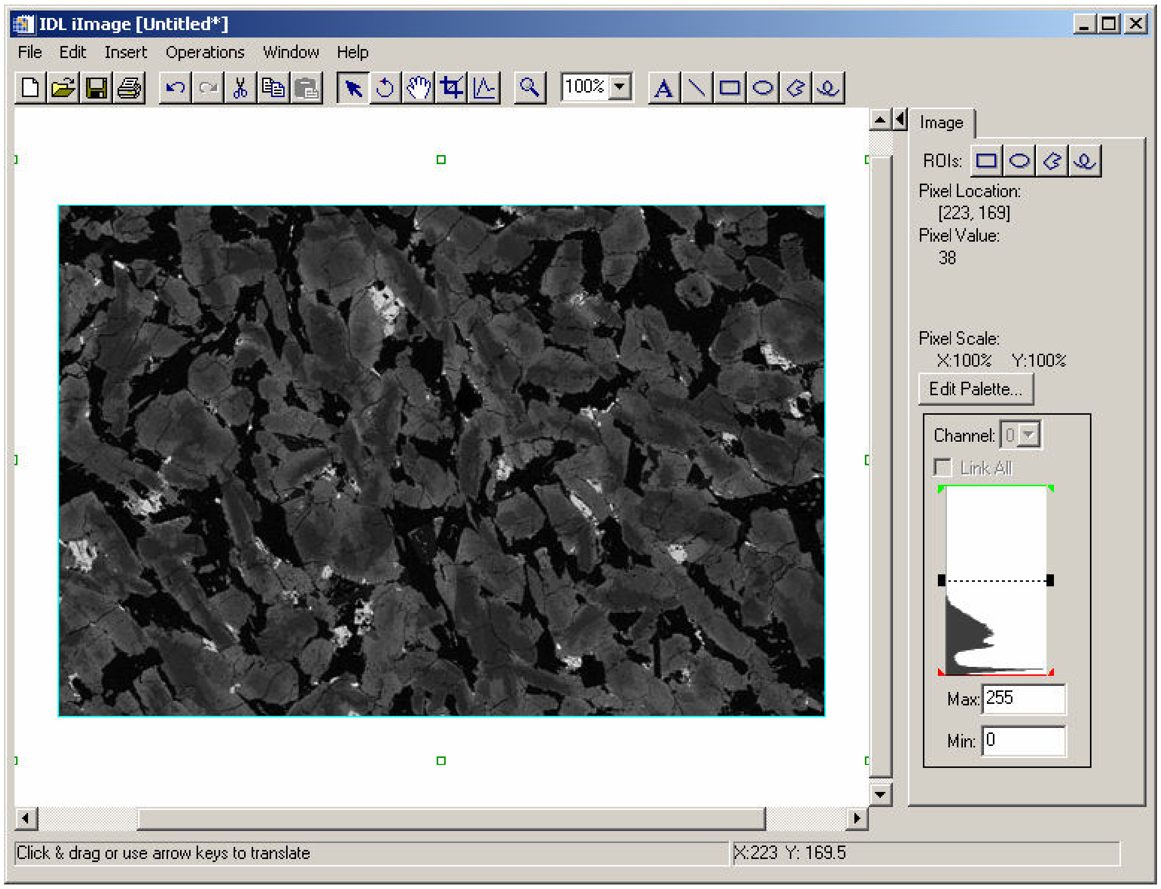


Figure 2: Display of the Shergotty meteorite image within the iImage utility

On the right-hand side of the *IDL iImage* window the "Min:" and "Max:" boxes show that the pixels in the image range throughout the full 8-bit range (0 → 255). However, the histogram plot window illustrates that most of the pixels within the image have brightness values in the lower half of the data range. The histogram plot is essentially portraying the overall dark gray to black appearance of the image.

A simple form of **image enhancement** can be obtained by moving the histogram threshold bars within the *iImage* utility. This will adjust the range of pixel data values that are mapped to the 256 levels of gray displayed on the screen. The **stretching** of the image in the defined range is performed in a linear fashion and this provides a form of **contrast** enhancement.

10. Within the "Max:" field box, type a pixel data value of 110 and press the *Enter* key on the keyboard. This will saturate all pixels in the image with a value of 110 or higher to white, while stretching the pixels with values 0 → 109 throughout the full range of the grayscale display.

The resulting *IDL iImage* visualization window should look similar to Fig. 3.

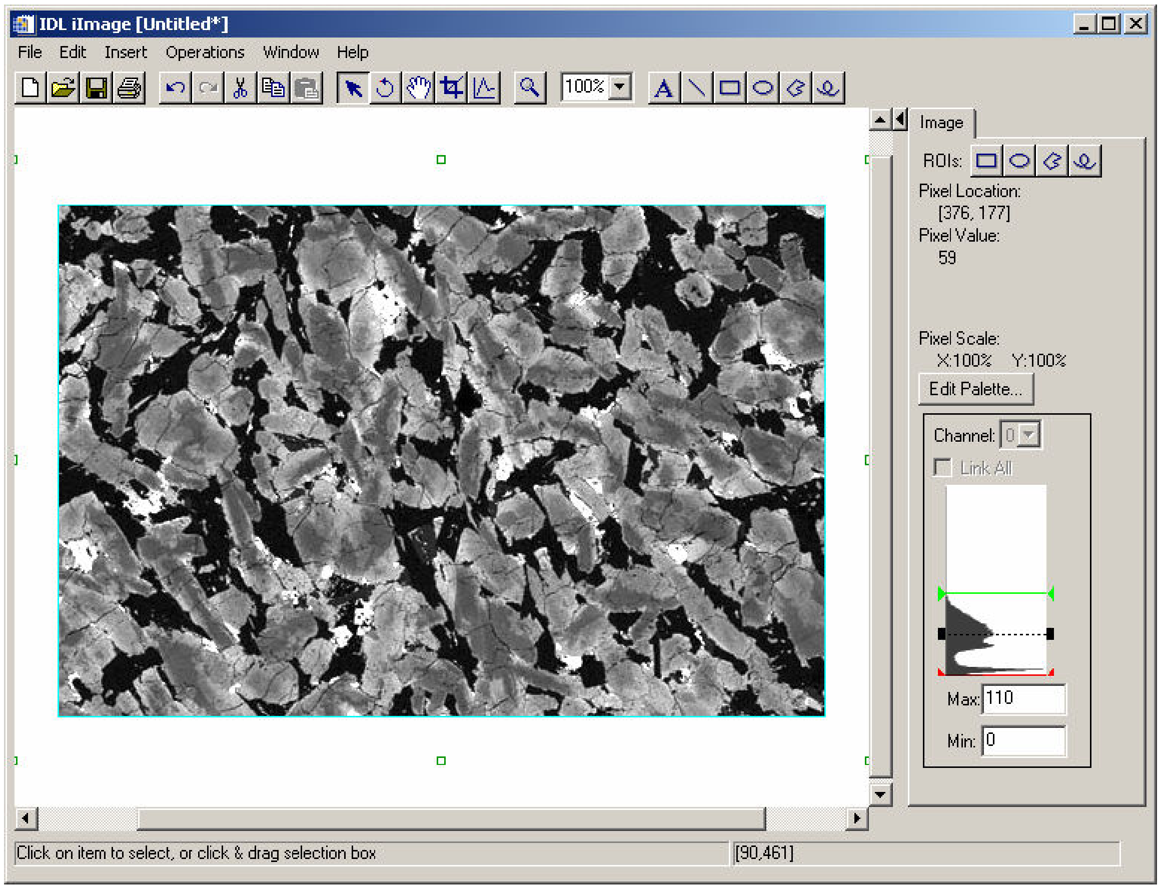


Figure 3: Contrast enhancement of the image via a linear stretch

This manipulation of the histogram stretch bars only affects the display of the image and does not change the actual pixel values for the image dataset. Notice that the "Pixel Value:" field within the Image panel now displays a number in parentheses next to the actual pixel data value. This number in parentheses is the output grayscale intensity for the current pixel according to the stretch that is being applied.

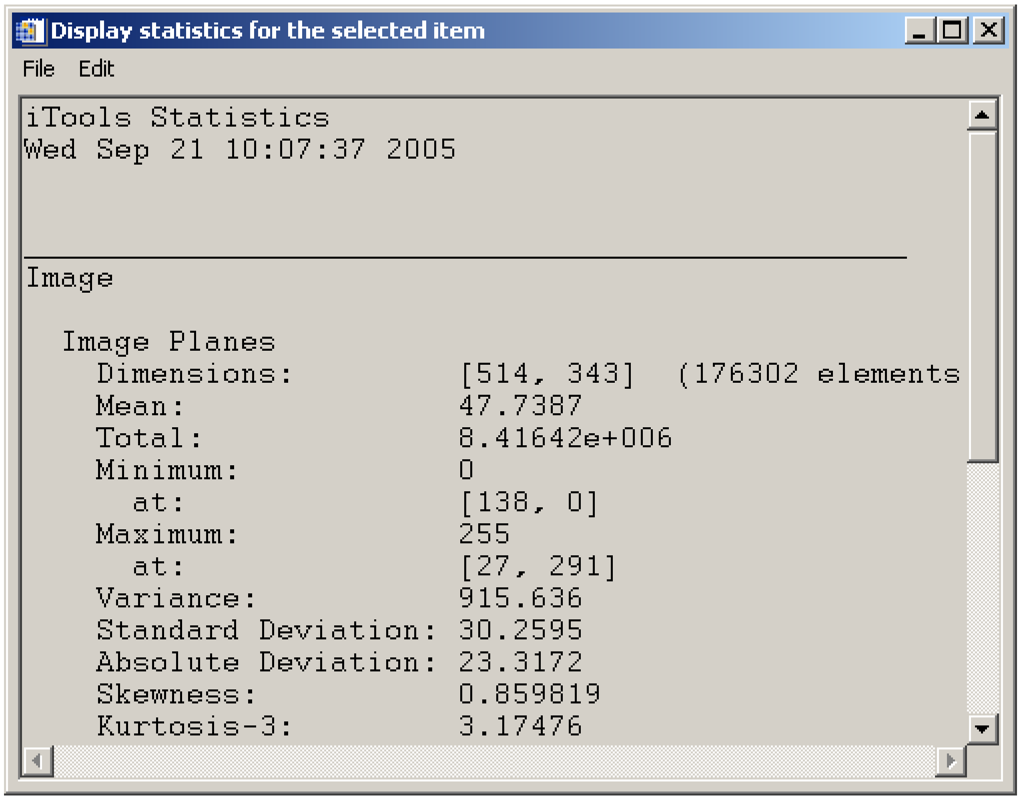
11. Change the "Max:" field back to "*255*" by either typing in the text box or clicking on the green stretch bar and dragging it back up to the top of the histogram display window.

There are a number of analysis tools found within the *Operations* menu of the *iImage* utility. The operations that are built into the iTools system represent some of the most common image processing tasks.

12. While the image object is selected within the *IDL iImage* window, select "*Operations > Statistics…*" from the menu system.

A separate dialog will appear that displays some statistical information on the current image dataset [Fig. 4]. Notice that the average (mean) pixel value for this image is 47.7387, which explains the relatively dark appearance of the original image.

13. Once finished viewing the image statistics, close the *Display statistics for the selected item* dialog.



```
iTools Statistics
Wed Sep 21 10:07:37 2005

Image

  Image Planes
    Dimensions:           [514, 343]  (176302 elements
    Mean:                 47.7387
    Total:                8.41642e+006
    Minimum:              0
      at:                 [138, 0]
    Maximum:              255
      at:                 [27, 291]
    Variance:             915.636
    Standard Deviation:   30.2595
    Absolute Deviation:   23.3172
    Skewness:             0.859819
    Kurtosis-3:           3.17476
```

Figure 4: Statistics for the meteorite thin section image

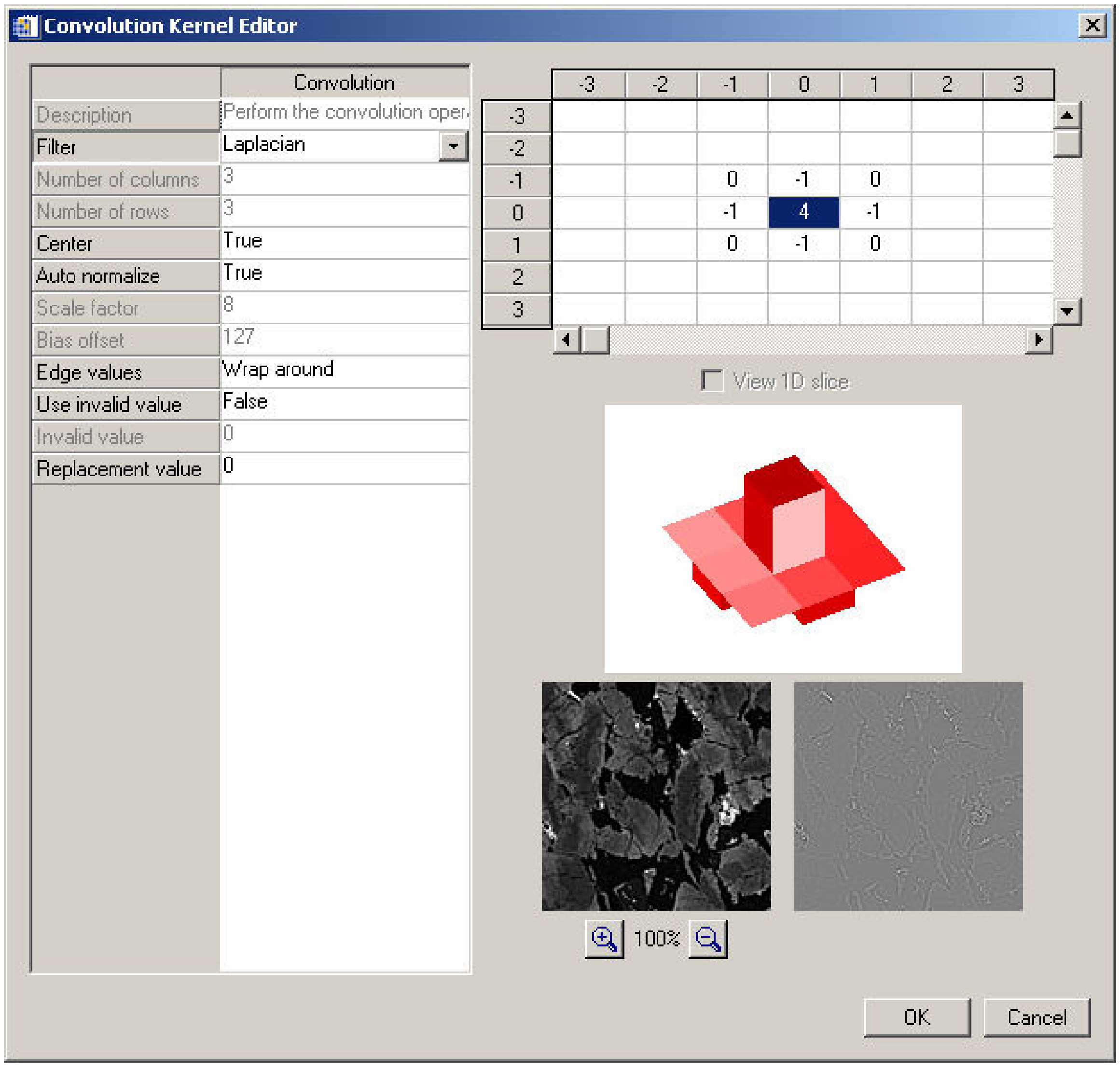
One of the operations built into the *iImage* utility is the **Unsharp Masking** technique, which applies a **sharpening** filter to the image. Digital Unsharp Masking is a digital image processing technique that increases the contrast where subtle details are set against a diffuse background. This operation suppresses features which are smooth (those with structures on large scales) in favor of sharp features (those with structure on small scale), resulting in a net enhancement of the contrast of fine structure in the image.

14. From the *IDL iImage* window menu system select "*Operations > Filter > Unsharp Mask*".
15. Within the *Unsharp Mask* dialog that pops up, leave all parameters set to their default values and simply press "*OK*".

Notice that the fine detail within the image is enhanced by applying the Unsharp Mask operation.

The *iImage* utility also has a built-in tool for convolving an image array with a kernel. **Convolution** is a simple matrix algebra operation that can be used for various types of smoothing, shifting, differentiation, edge detection, etc..

16. Select "*Operations > Filter > Convolution*" from the *iImage* menu system. A separate dialog entitled *Convolution Kernel Editor* will appear [Fig. 5].



	Convolution
Description	Perform the convolution oper.
Filter	Laplacian
Number of columns	3
Number of rows	3
Center	True
Auto normalize	True
Scale factor	8
Bias offset	127
Edge values	Wrap around
Use invalid value	False
Invalid value	0
Replacement value	0

	-3	-2	-1	0	1	2	3
-3							
-2							
-1			0	-1	0		
0			-1	4	-1		
1			0	-1	0		
2							
3							

Figure 5: The iTools Convolution Kernel Editor dialog

The *Convolution Kernel Editor* window allows the user to select from a list of pre-defined kernels, or define their own user-defined kernel. The convolution of these different kernels will have a wide variety of effects on the resulting image display.

The **Laplacian** filter can be applied by convolving a Laplacian kernel with the image. A Laplacian filter is an edge enhancement filter that operates without regard to edge direction. Laplacian filtering emphasizes maximum values within the image by using a kernel with a high central value typically surrounded by negative weights in the up down and left-right directions and zero values at the kernel corners. The Laplacian kernel convolution is a form of **high pass** filter, which removes the low frequency components of an image while retaining the high frequency (local variations). It can be used to **enhance edges** between different regions as well as to sharpen an image.

17. Using the droplist next to the "Filter" parameter in the upper left hand corner of the *Convolution Kernel Editor* dialog, select the "*Laplacian*" kernel.
18. Notice that a surface representation of the current kernel is displayed within this dialog. The user can click on this surface and rotate it in order to visualize the structure of the kernel.
19. Once the "*Laplacian*" kernel has been selected press "*OK*" to apply the convolution operation and dismiss the *Convolution Kernel Editor* dialog.
20. At this point, it is beneficial to change the range for the current stretch to the following values :
    - Max: 140
    - Min: 115

The application of the Laplacian filter will enhance the edges between different regions (in this case mineral grains) within the image. The resulting *IDL iImage* visualization window should look similar to Fig. 6.

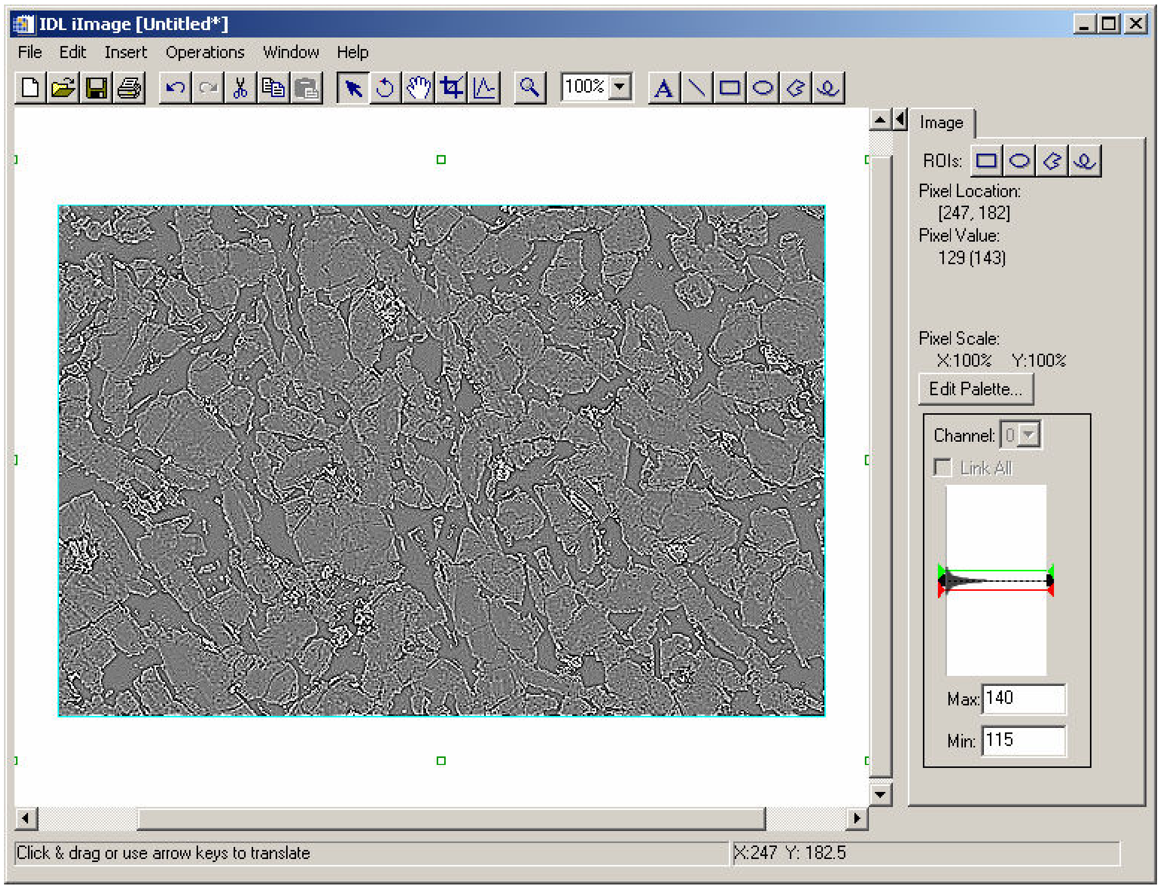
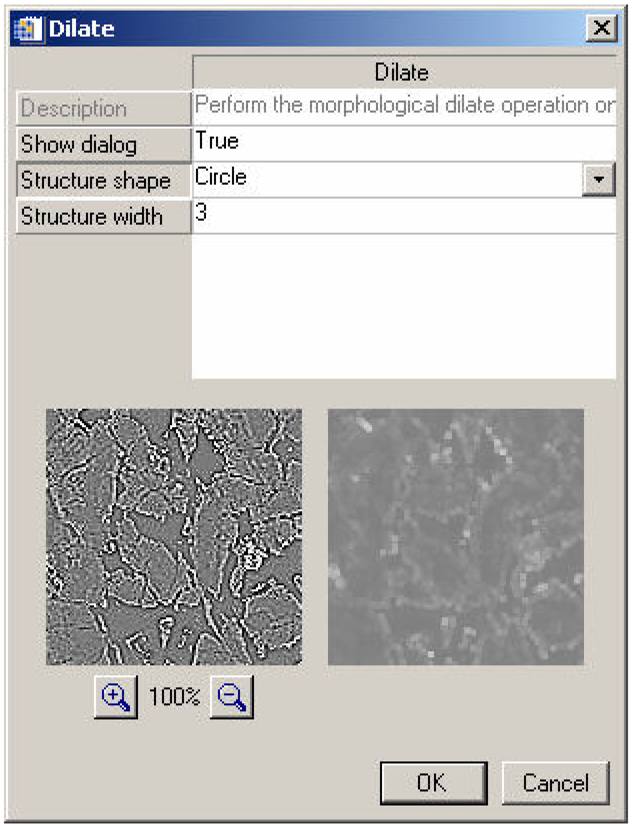


Figure 6: Application of a Laplacian filter convolution

IDL also has a number of morphological image operators built into its library of routines. Mathematical morphology is a method of processing digital images on the basis of shape. Some of these morphological algorithms have been added to the operations within the iTools system. For example, the dilate operator, which is commonly known as the "fill", "expand", or "grow" operator, can be used to further enhance the boundaries between mineral grains in the current image.

21. Select "*Operations* > *Morph* > *Dilate*" from the *iImage* menu system. A separate dialog entitled *Dilate* will appear [Fig. 7]. The parameters associated with the dilate operation are displayed in this dialog along with a preview of the operation.
22. Click on the box to the right of the "Structure shape" field and change the setting to "Circle" [Fig. 7].
23. Press the "*OK*" button to apply the dilate operation and dismiss the *Dilate* dialog.



	Dilate
Description	Perform the morphological dilate operation or
Show dialog	True
Structure shape	Circle
Structure width	3

Figure 7: The iTools Dilate operation dialog

The resulting *IDL iImage* visualization window should look similar to Fig. 8.

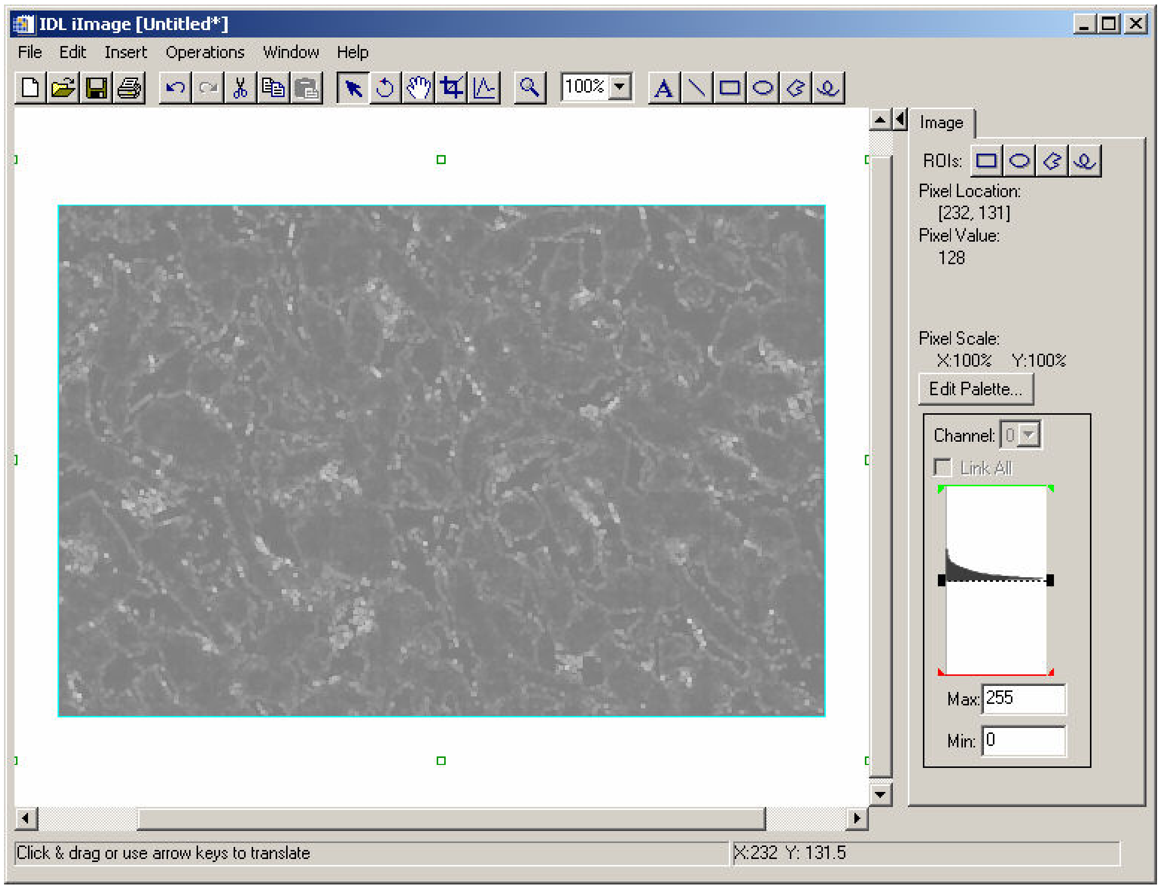


Figure 8: Application of the Dilate morphological operation

24. Once finished viewing the processed image, close the *IDL iImage* window.

In addition to sharpening, high pass filtering, and edge enhancement techniques, IDL also offers operations to perform image **smoothing**, **low pass filtering**, and **noise removal**. Re-launch the *iImage* utility with the original image so these techniques can be investigated :

25. IDL> `iImage, image`

The median operation replaces each pixel with the median of the two-dimensional neighborhood of the specified width. In an ordered set of values, the median is a value with an equal number of values above and below it. Median filtering is effective in removing salt and pepper noise (isolated high or low values). The resulting image will have a less grainy appearance than the original.

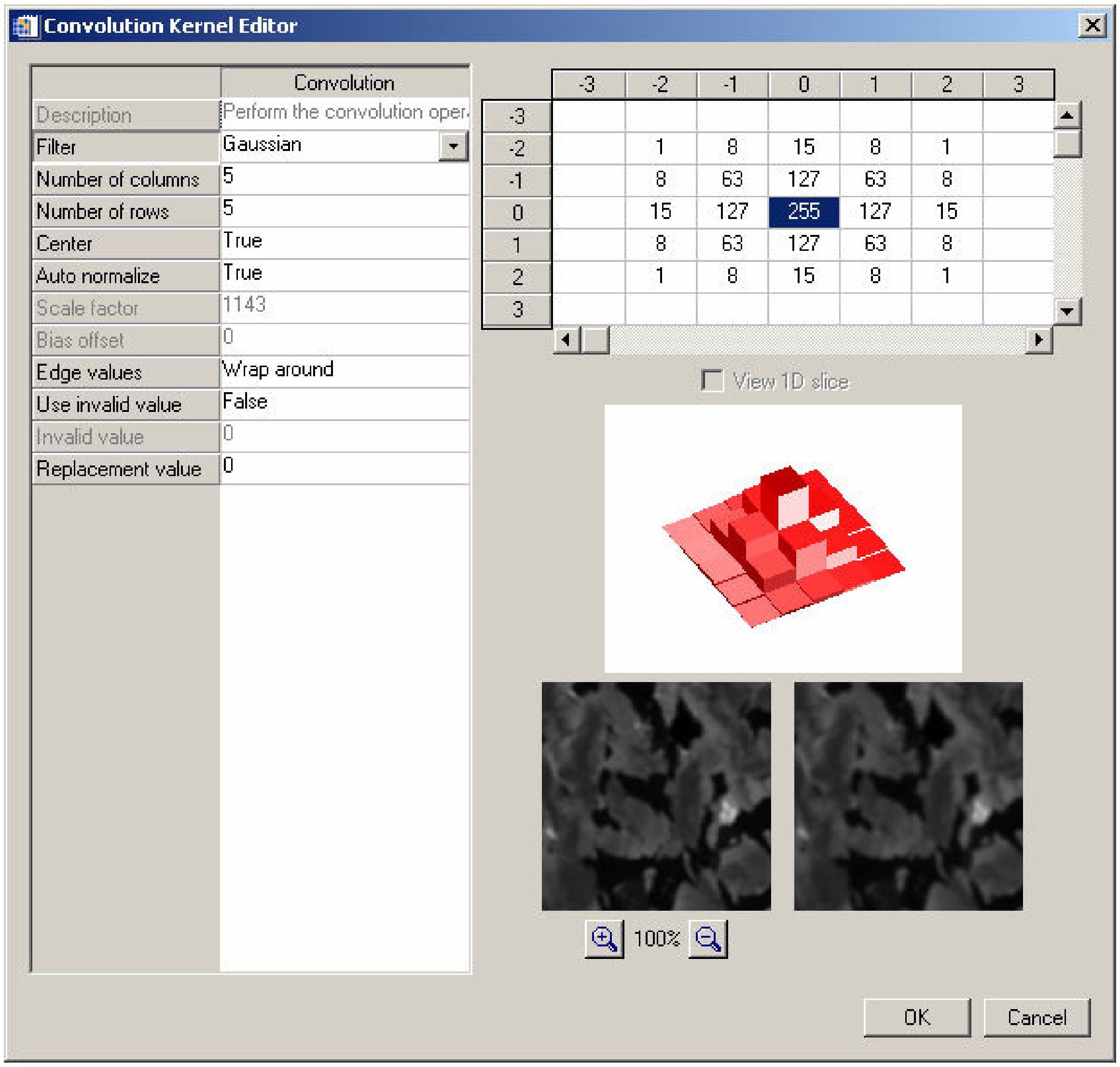
26. From the *iImage* menu system, select "*Operations > Filter > Median*".
27. Within the *Median* dialog window, leave all of the default settings and press "*OK*".

The smooth operation will compute the boxcar average of a specified width for the image. Smoothing is similar to the median filter except the pixels are replaced with the average (mean) value across the neighborhood. This tends to have the effect of blurring the edges within the image and making them more diffuse.

28. From the *iImage* menu system, select "*Operations > Filter > Smooth*".
29. Press the "*OK*" button to apply the smoothing operation and dismiss the *Smooth* dialog window.

Finally, the Convolution tool can be used once again to apply a low pass filter to the image. The Gaussian kernel provides a form of low pass filtering that preserves the low frequency components of an image.

30. Select "*Operations > Filter > Convolution*" from the *iImage* menu system.
31. Within the *Convolution Kernel Editor* dialog, change the "Filter" selection droplist to "*Gaussian*" [Fig. 9].
32. Edit the number of columns and rows fields so the kernel has a size of 5 x 5 [Fig. 9].
33. Press the "*OK*" button to apply the Gaussian filter and dismiss the dialog.

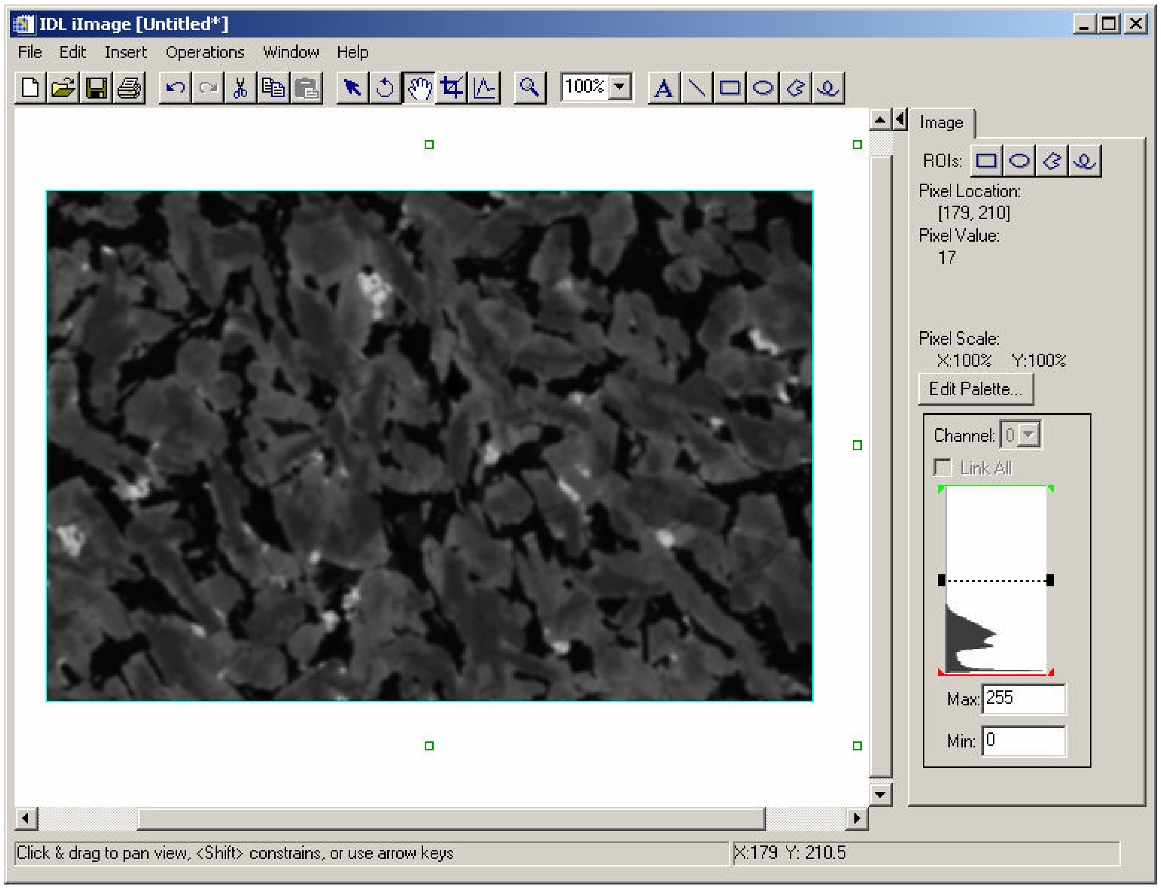


	Convolution
Description	Perform the convolution oper
Filter	Gaussian
Number of columns	5
Number of rows	5
Center	True
Auto normalize	True
Scale factor	1143
Bias offset	0
Edge values	Wrap around
Use invalid value	False
Invalid value	0
Replacement value	0

	-3	-2	-1	0	1	2	3
-3							
-2		1	8	15	8	1	
-1		8	63	127	63	8	
0		15	127	255	127	15	
1		8	63	127	63	8	
2		1	8	15	8	1	
3							

Figure 9: Application of the Gaussian filter convolution

The resulting *IDL iImage* visualization window should look similar to Fig. 10.



Pixel Location:
[179, 210]
Pixel Value:
17

Pixel Scale:
X:100% Y:100%

Max: 255
Min: 0

Click & drag to pan view, <Shift> constrains, or use arrow keys
X:179 Y: 210.5

Figure 10: Image display after noise removal, smoothing, and low pass filtering

34. Once finished viewing the processed image, close the *IDL iImage* window.

Thresholding, Clipping, and Histogram Equalization

Although the *iImage* utility has a lot of built-in analytical techniques, the bulk of IDL's image processing capabilities must be accessed using routines within the IDL language. IDL's image processing library contains a number of routines for contrast enhancement, filtering, feature extraction, image segmentation, geometry transformations, and regions of interest analysis. In addition, the IDL language has built-in operators that can be utilized to perform simple image processing techniques such as masking and stretching.

Thresholding (also known as masking) is used to isolate features within an image above, below, or equal to a specified pixel value. The value (known as the threshold level) determines how the masking occurs. In IDL, thresholding is performed using the relational operators. IDL's relational operators are illustrated in Table 7-1 :

OPERATOR	DESCRIPTION
EQ	Equal to
NE	Not equal to
GE	Greater than or equal to
GT	Greater than
LE	Less than or equal to
LT	Less than

Table 7-1 : IDL's Relational Operators

For example, in order to threshold the Shergotty meteorite image and identify the pixels that have a value greater than 70 (byte) simply execute the following statement :

1. IDL> `mask = image GT 70B`

This expression creates a new variable named "*mask*" that is a 2-dimensional array of the same size as the original image. This new "*mask*" variable contains a binary image where each pixel has a value of either one (original image pixel value was greater than 70) or zero (original image pixel value was equal to or less than 70). At this point, the user may wish to view this binary threshold image by loading it into the *iImage* utility :

2. IDL> `iImage, mask`

The resulting image display within the *IDL iImage* window should appear completely black. This is due to the fact that all of the pixels within the "*mask*" binary image have a value of 0 or 1, which are very difficult to discern (and very dark) within a 0 → 255 grayscale display. Consequently, the *BYTSCL* function should be utilized when displaying binary images so the pixels with a value of 1 are actually mapped to 255 (white). The *BYTSCL* function scales all values of an array into a specified range (0 → 255 by default) :

3. Close the existing *IDL iImage* window.
4. Re-issue the *iImage* statement, but this time wrap the mask image variable with a dynamic call to the *BYTSCL* function :

   IDL> `iImage, BYTSCL (mask)`

The resulting *IDL iImage* visualization window should look similar to Fig. 11.

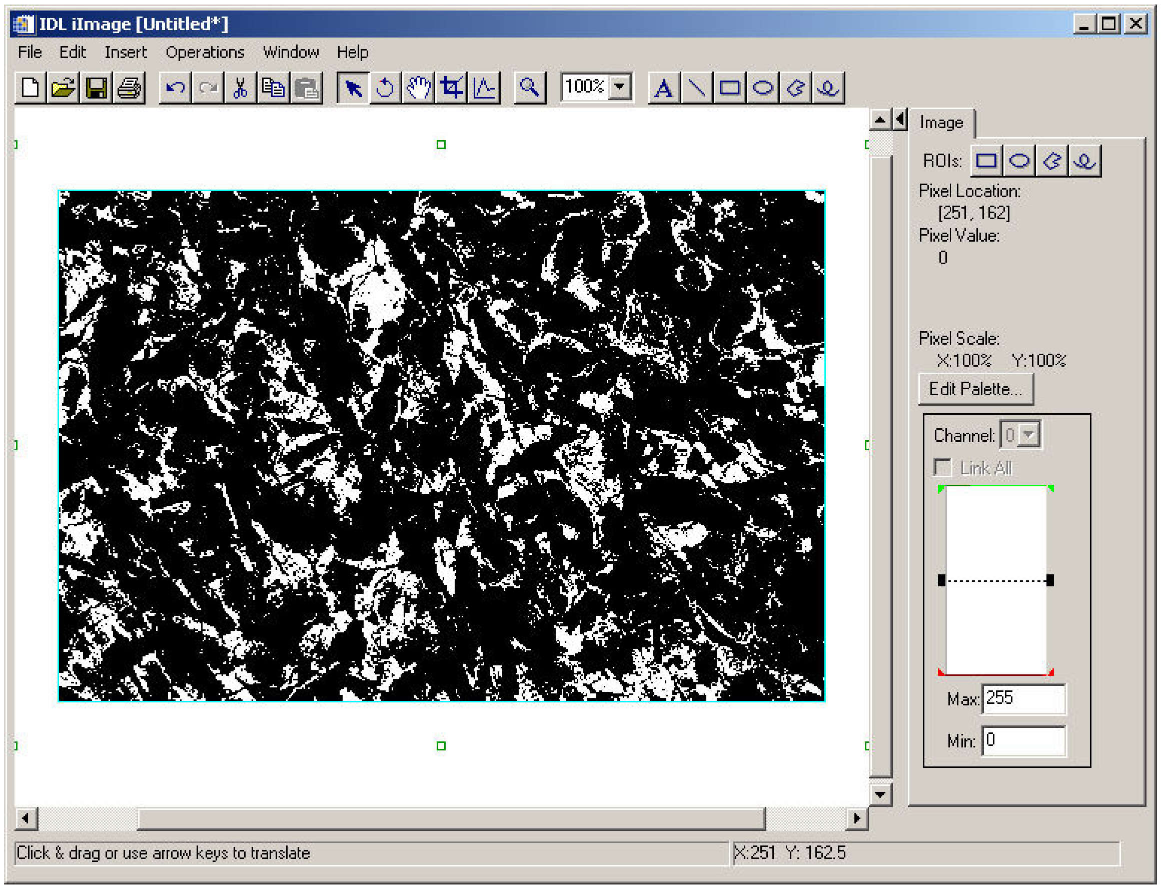


Figure 11: Threshold image showing pixels with values greater than 70 (white) and less than or equal to 70 (black)

5. Once finished viewing the binary threshold image, close the *IDL iImage* window.

Binary threshold images can also be used to **mask-out** the pixels in an image that do not qualify based on the given expression. For example, in order to display only those pixels from the original image that have a value greater than 70 simply execute the following statements :

6. IDL> `masked = image * mask`
7. IDL> `iImage, masked`

The resulting *IDL iImage* visualization window should look similar to Fig. 12.

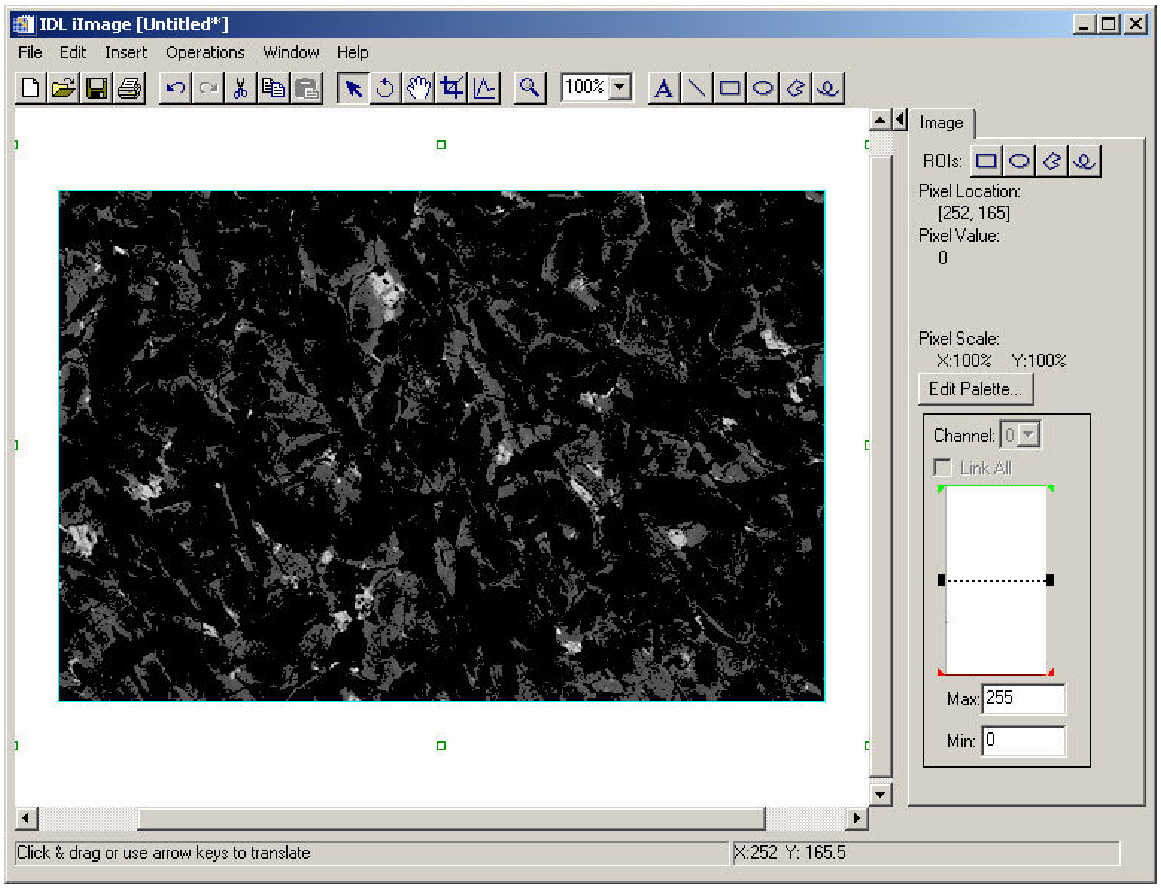


Figure 12: Display of the original image with all pixels that have a value of 70 or less masked-out (i.e. displayed as black)

8. Once finished viewing the masked image, close the *IDL iImage* window.

The user can also provide both upper and lower bounds when creating threshold images by using the Boolean operators built into IDL (*AND*, *NOT*, *OR*, and *XOR*). For example, create a threshold image that identifies those pixels which have a data value between 50 and 70 :

9. IDL> `mask = (image GE 50B) AND (image LE 70B)`
10. IDL> `iImage, BYTSCL (mask)`

The resulting *IDL iImage* visualization window should look similar to Fig. 13.

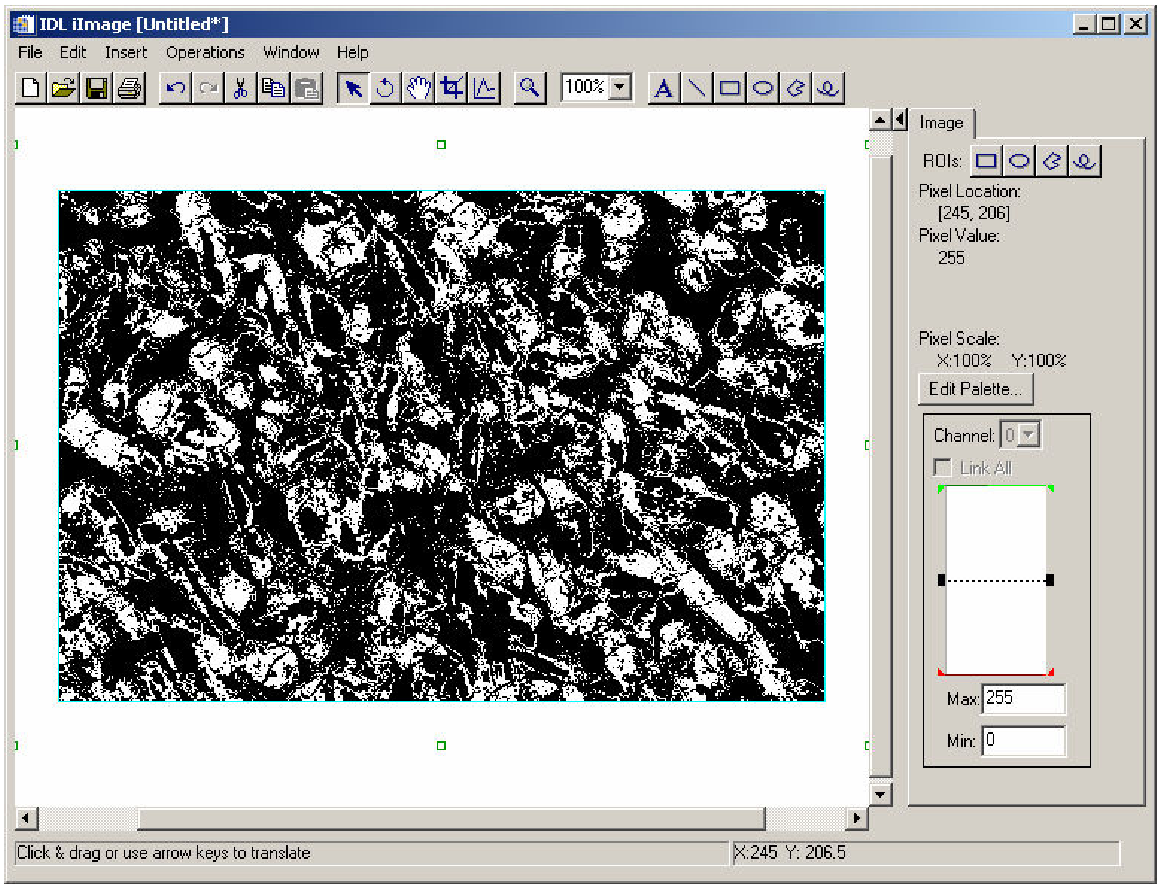


Figure 13: Threshold image showing all pixels with data values between 50 and 70

11. Once finished viewing the threshold image, close the *IDL iImage* window.

Clipping is similar to thresholding because pixels with data values above or below a specified level are all set to the same value. However, when clipping an image the pixels that do not satisfy the expression are set to the selected level and the resulting image is not binary in nature. Clipping can be used to enhance features within an image.

In IDL, clipping is performed with the minimum (<) and maximum (>) operators. In order to clip an image the user must design an expression that contains an image array, the appropriate operator, and the clipping level. For example, to clip the meteorite thin section image so that all pixels with a value greater than or equal to 50 are set to a value of 50 simply execute the following statements :

12. IDL> `clipped = image < 50B`
13. IDL> `iImage, clipped`

The resulting *IDL iImage* visualization window should look similar to Fig. 14.

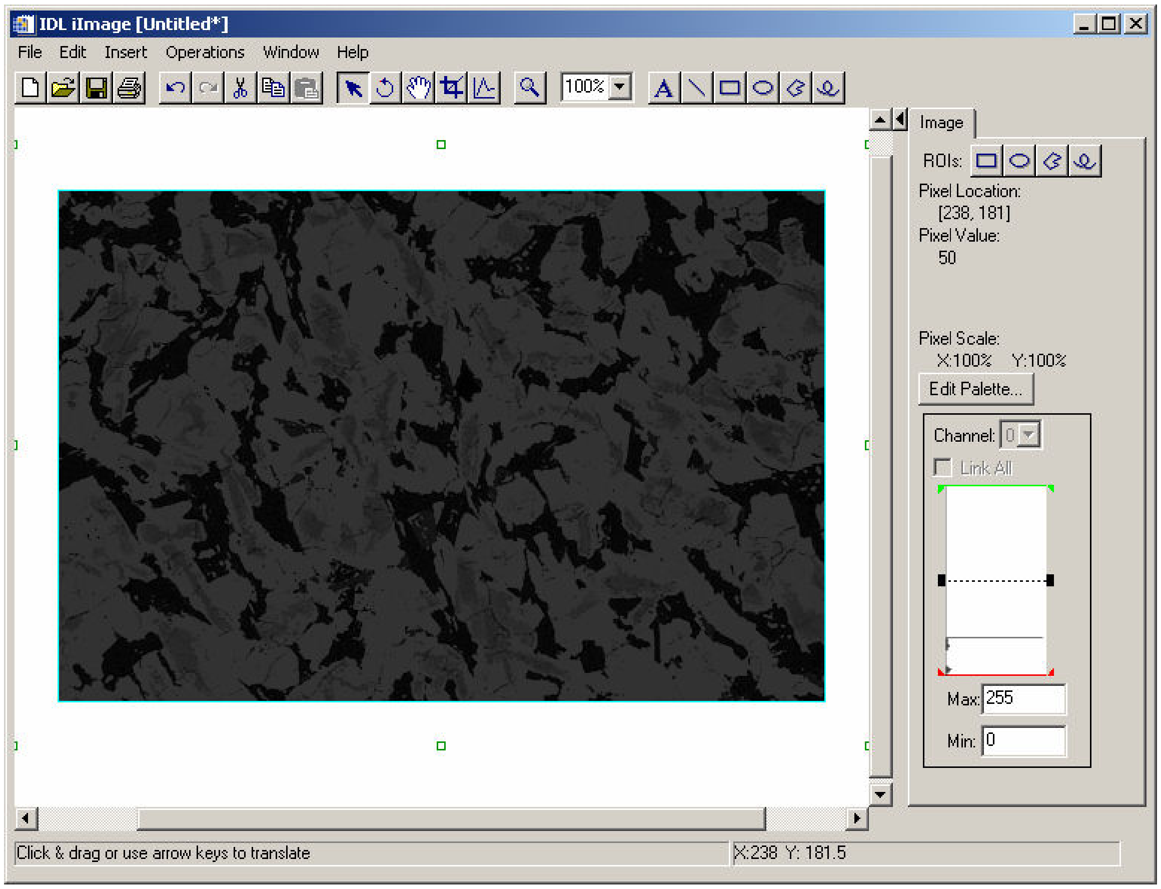

IDL iImage [Untitled*]
File Edit Insert Operations Window Help
100%
Image
ROIs:
Pixel Location: [238, 181]
Pixel Value: 50
Pixel Scale: X:100% Y:100%
Edit Palette...
Channel: 0
Link All
Max: 255
Min: 0
Click & drag or use arrow keys to translate
X:238 Y: 181.5

Figure 14: Clipped image showing all pixels with data values 50 or higher set to a brightness level of 50

14. Once finished viewing the clipped image, close the *IDL iImage* window.

When clipping is used in conjunction with byte-scaling it is equivalent to performing a stretch on an image. For example, in order to stretch the image between the range of 25 → 100 simply execute the following statements :

```
15. IDL> stretched = BYTSCL (image > 25B < 100B)
16. IDL> iImage, stretched
```

It is worth mentioning that the same stretching technique can be obtained by utilizing the *MIN* and *MAX* keywords to the *BYTSCL* function :

```
17. IDL> stretched = BYTSCL (image, MIN=25, MAX=100)
18. IDL> iImage, stretched
```

The resulting *IDL iImage* visualization window(s) should look similar to Fig. 15.

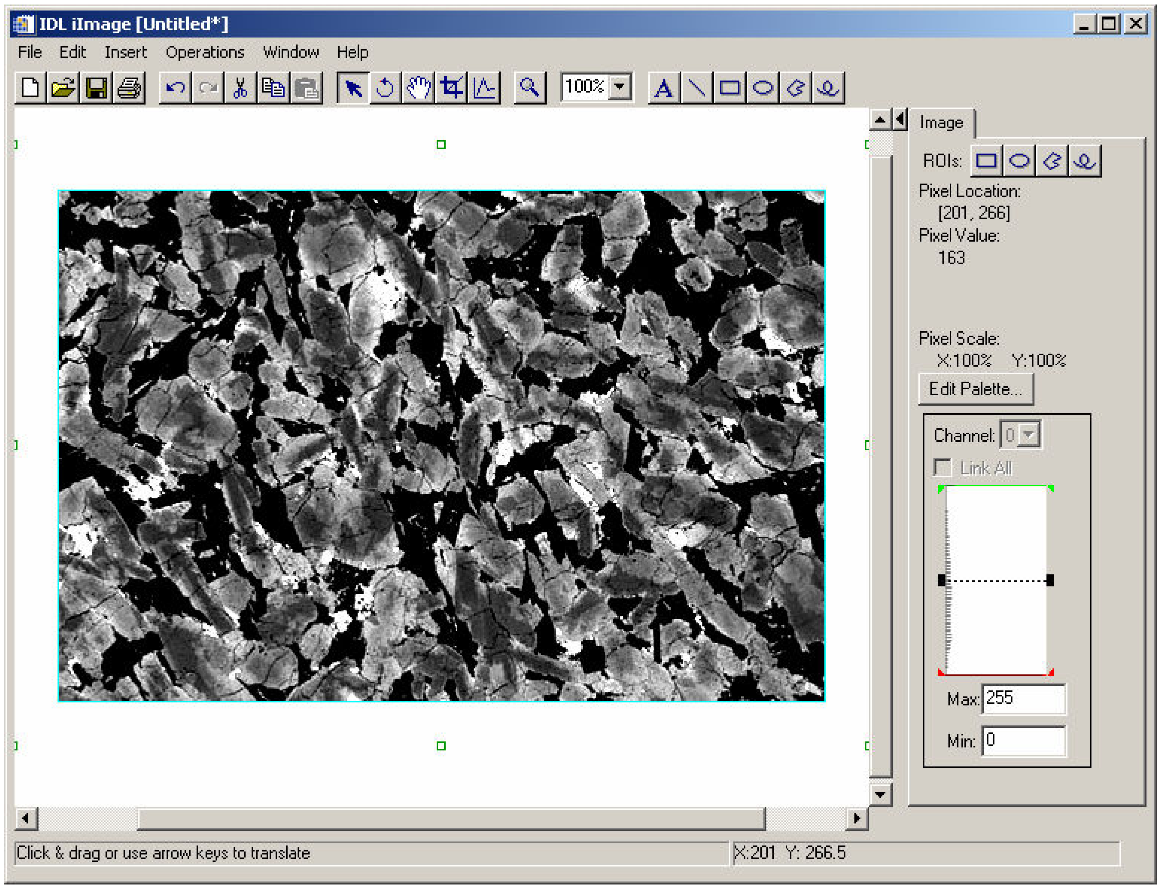


Figure 15: Stretched image that highlights all pixels with data values between 25 and 100

19. Once finished viewing the stretched image, close the *IDL iImage* window(s).

In addition to simple linear stretching techniques, IDL also has routines that allow the user to stretch the image using other histogram manipulations. For example, the *HIST_EQUAL* function can be used to apply a histogram equalization stretch to the image data. Histogram equalization employs a monotonic, non-linear mapping which re-assigns the intensity values of pixels in the input image such that the output image contains a uniform distribution of intensities (i.e. a flat histogram). Execute the following statements in order to derive and display the histogram-equalized version of the meteorite thin section image :

20. IDL> `equalized = HIST_EQUAL (image)`
21. IDL> `iImage, equalized`

Notice that the resulting image has improved contrast and the histogram has a very even distribution throughout the 0 → 255 range. The resulting *IDL iImage* visualization window should look similar to Fig. 16.

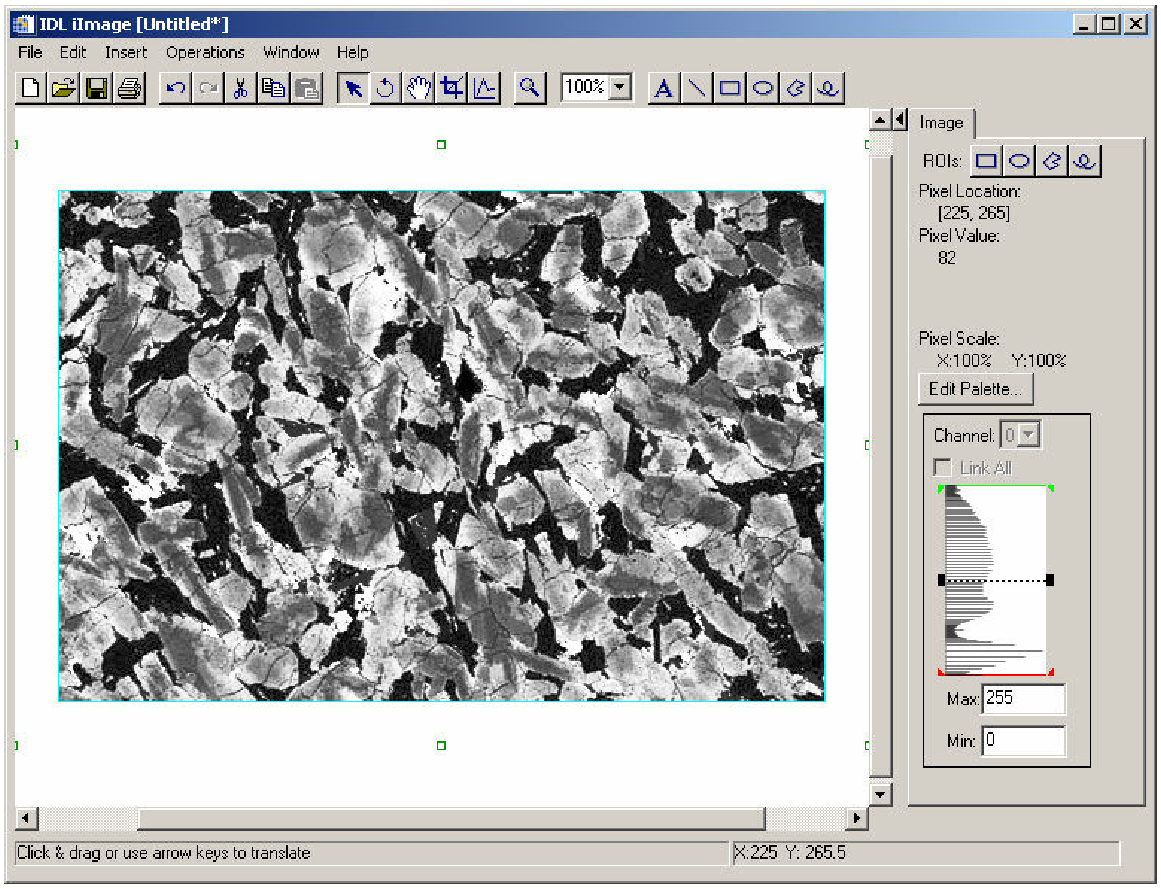


Figure 16: Image display with a histogram equalization stretch

22. Once finished viewing the histogram-equalized image, close the *IDL iImage* window.

In addition to the standard histogram equalization provided by the *HIST_EQUAL* function, IDL also provides the *ADAPT_HIST_EQUAL* function which performs adaptive histogram equalization (a form of automatic image contrast enhancement). Adaptive histogram equalization involves applying contrast enhancement based on the local region surrounding each pixel. Each pixel is mapped to an intensity proportional to its rank within the surrounding neighborhood. This method of automatic contrast enhancement has proven to be broadly applicable to a wide range of images and to have demonstrated effectiveness. Execute the following statements in order to apply the adaptive histogram equalization and display the resulting image :

23. IDL> `adaptive = ADAPT_HIST_EQUAL (image)`
24. IDL> `iImage, adaptive`

The resulting *IDL iImage* visualization window should look similar to Fig. 17.

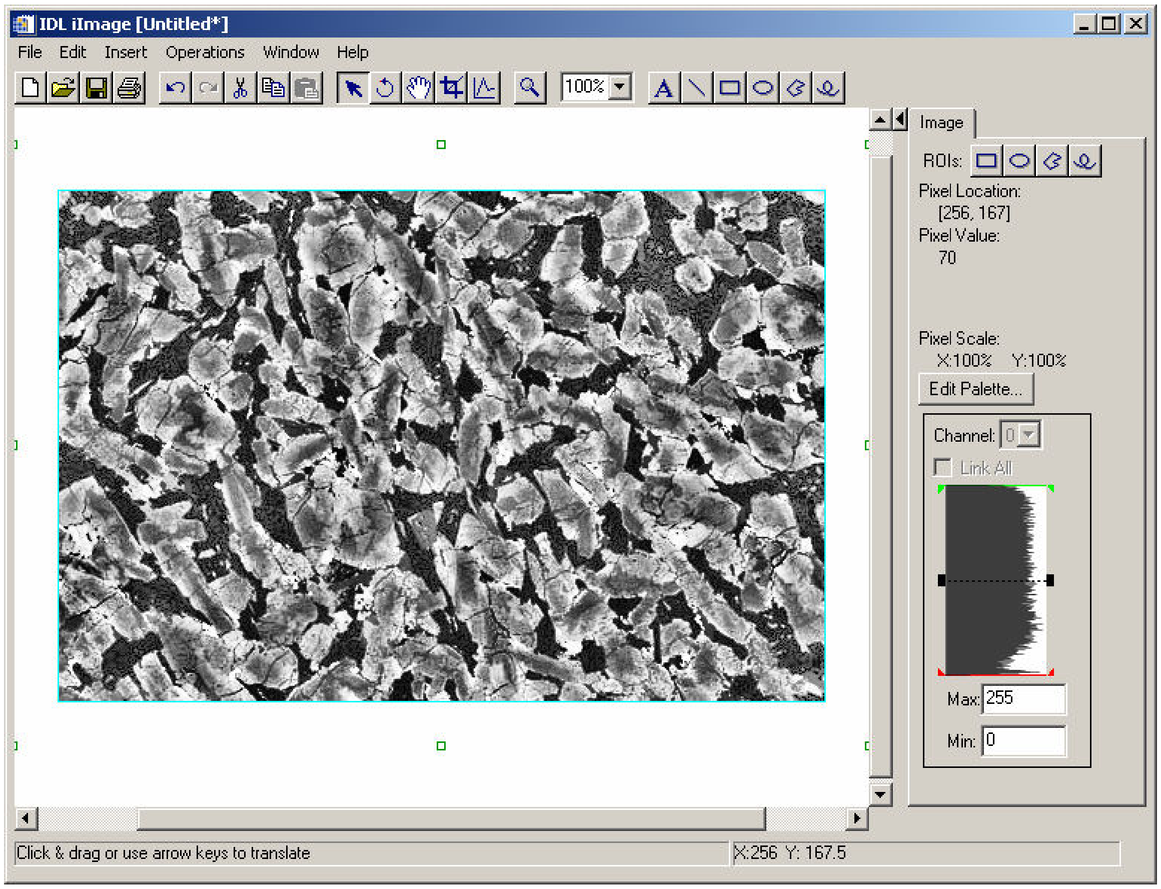


Figure 17: Image display with an adaptive histogram equalization stretch

25. Once finished viewing the adaptive histogram-equalized image, close the *IDL iImage* window.

Morphological Operations and Image Segmentation

Morphological image processing operations reveal the underlying structures and shapes within binary and grayscale images. While individual morphological operations perform simple functions, they can be combined to extract specific information from an image. Morphological operations often precede more advanced pattern recognition and image analysis operations such as segmentation. Shape recognition routines commonly include image thresholding or stretching to separate foreground and background image features.

Morphological operations apply a **structuring element** or morphological mask to an image. A structuring element that is applied to an image must be 2 dimensional, having the same number of dimensions as the array to which it is applied. A morphological operation passes the structuring element, of an empirically determined size and shape, over an image. The operation compares the structuring element to the underlying image and generates an output pixel based upon the function of the morphological operation. The size and shape of the structuring element determines what is extracted or deleted from an image. In general, smaller structuring elements preserve finer details within an image than larger elements.

Start by thresholding the Shergotty meteorite image in order to identify the dark mineral grains with a pixel value less than or equal to 20 :

1. IDL> `minerals = image LE 20B`

Next, create a structuring element array with a square shape that will help extract objects with sharp rectangular edges :

2. IDL> `structElem = BYTARR (3,3) + 1B`
3. IDL> `PRINT, structElem`

```
   1   1   1
   1   1   1
   1   1   1
```

The *MORPH_CLOSE* function can be used with this structuring element to apply the closing operator to the binary threshold image. The closing operator has the effect of clumping the threshold image, thereby filling in holes within and connecting gaps between neighboring regions. In addition, the *MORPH_OPEN* function can be subsequently used to apply the opening operator, which will have a sieving effect on the image that helps to remove small isolated regions. Apply these morphological operations and visualize the results in comparison to the original image :

4. IDL> `clumped = MORPH_CLOSE (minerals, structElem)`
5. IDL> `sieved = MORPH_OPEN (minerals, structElem)`
6. IDL> `iImage, image, VIEW_GR=[2,2]`
7. IDL> `iImage, BYTSCL (minerals), /VIEW_NE`
8. IDL> `iImage, BYTSCL (clumped), /VIEW_NE`
9. IDL> `iImage, BYTSCL (sieved), /VIEW_NE`
10. Click on each individual window pane and change the canvas zoom droplist to "*50%*".

The resulting *IDL iImage* visualization window should look similar to Fig. 18.

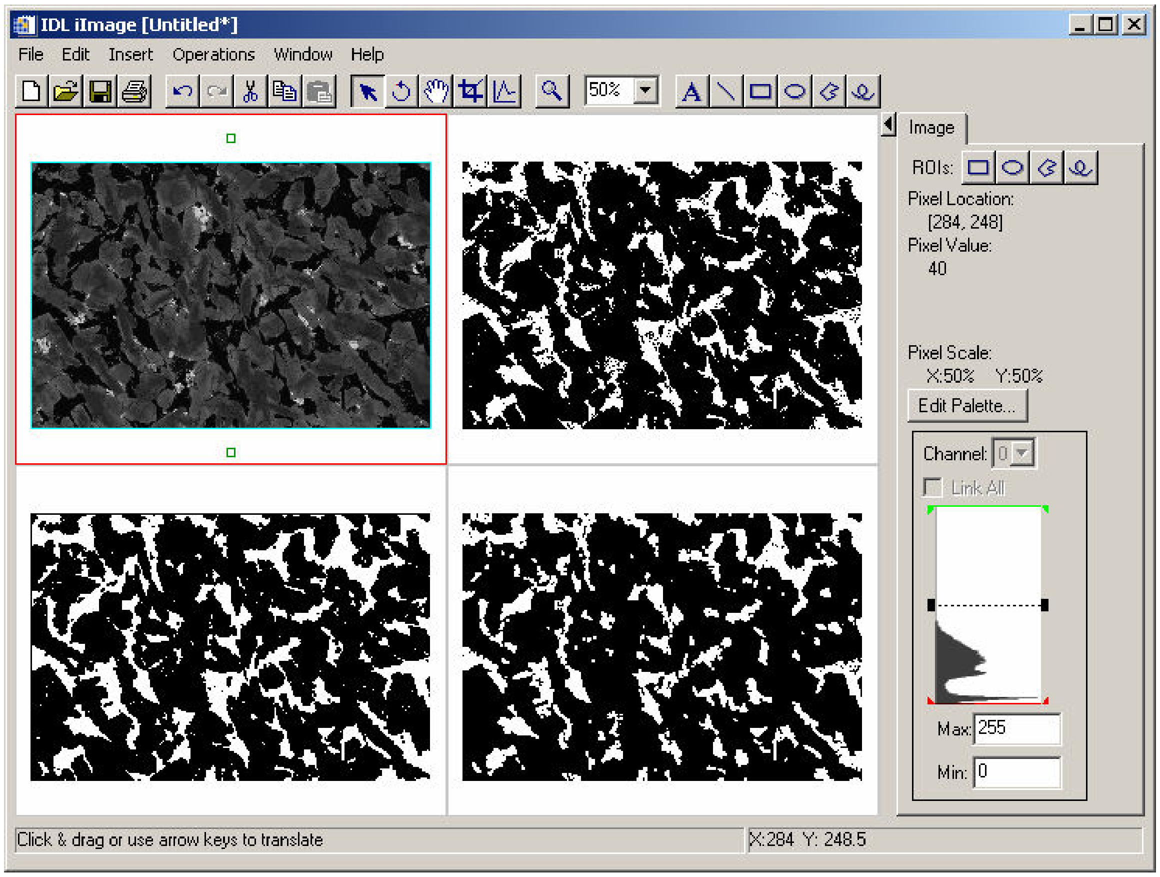


Figure 18: Display of original image (upper left), binary threshold image of dark mineral grains (upper right), application of a clumping operation (lower left), followed by a sieving operation (lower right)

Finally, the *LABEL_REGION* function can be used to perform image segmentation, which will consecutively label all of the regions, or *blobs*, of the clumped and sieved binary image with a unique region index. The resulting segmentation image can be displayed with a color table in order to visualize the separate distinct mineral grains within the meteorite image :

11. IDL> `segmented = LABEL_REGION (sieved)`
12. IDL> `iImage, segmented`
13. Within the Image panel on the right hand side of the *IDL iImage* window, press the "*Edit Palette…*" button.
14. Within the *Palette Editor* dialog, click on the "*Load Predefined…*" droplist and select "*Rainbow18*" from the dropdown menu.
15. Press the "*OK*" button to dismiss the *Palette Editor* dialog.

The resulting *IDL iImage* visualization window should look similar to Fig. 19.

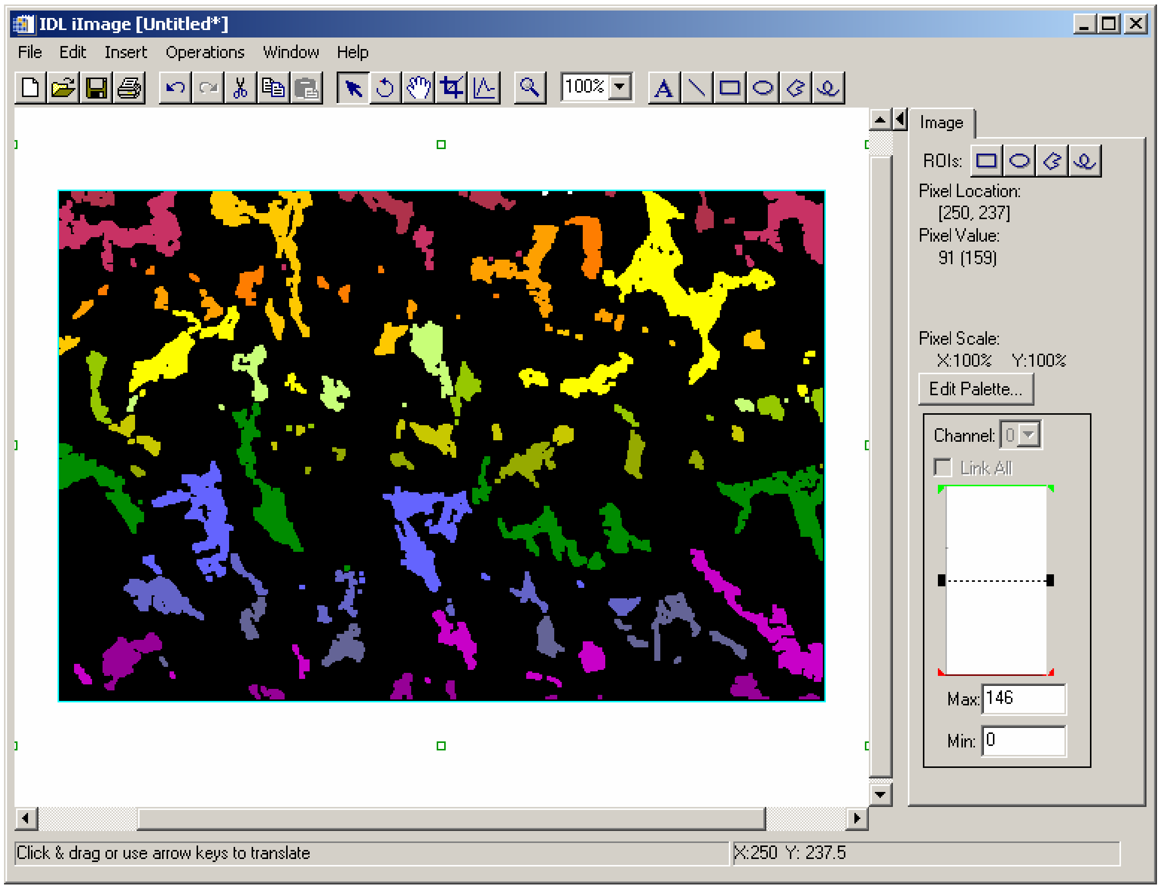


Figure 19: Image segmentation of the separate dark mineral grains

26. Once finished viewing the segmentation image, close the *IDL iImage* window.

Processing Images in Alternate Domains

So far all of the processing and visualization of image data has been performed in the **spatial domain**. This means that the digital image is represented by pixel values that have a particular spatial location (i.e. column and row). However, a pixel's value and location can also be represented in other domains. Transforming an image into an alternate domain can provide a basis for performing image filters, noise removal, sharpening, or feature extraction. In addition, domain transformations also provide additional information about an image and can enable robust image compression techniques.

In the frequency or **Fourier domain**, the value and location are represented by sinusoidal relationships that depend upon the frequency of a pixel occurring within an image. In this domain, pixel location is represented by its X and Y frequencies and its value is represented by an amplitude. Images can be transformed into the frequency domain to determine which pixels contain the most important information and whether repeating patterns occur.

In addition to the Fourier domain, IDL also has the ability to transform images into the wavelet (time-frequency), Hough, and Radon domains. In the **wavelet domain** the value and location of pixels are represented by sinusoidal relationships that only partially transform the image into the frequency domain. The wavelet transformation process is the basis for many image compression algorithms. The image information within the **Hough domain** shows the pixels of the original (spatial) image as sinusoidal curves. If the points of the original image form a straight line, their related sinusoidal curves in the Hough domain will intersect. Masks can be easily applied to the image within the Hough domain to determine if and where straight lines occur. The image information within the **Radon domain** shows a line through the original image as a point. Specific features and geometries within the original image will produce peaks within the Radon domain and can be easily identified.

In IDL, the *FFT* routine can be utilized to perform a **Fast Fourier Transformation** and convert an image from the spatial domain into the frequency domain. In the following exercise, the image data from the example data file "*hamburg.jp2*" will be input into the *iImage* utility and subsequently transformed into the Fourier domain. This example data file is located in the "*data*" subfolder.

The file "*hamburg.jp2*" is in JPEG2000 format and contains a satellite image of the loading docks at the port in Hamburg, Germany. Start by loading this image into a new *iImage* utility :

1. IDL> `iImage`
2. From the *iImage* menu system select "*File > Open…*".
3. Select the "*hamburg.jp2*" file and hit "*Open*".

The resulting image display should look similar to Fig. 20. Notice the linear and rectangular patterns that are prevalent in this image in both diagonal directions.

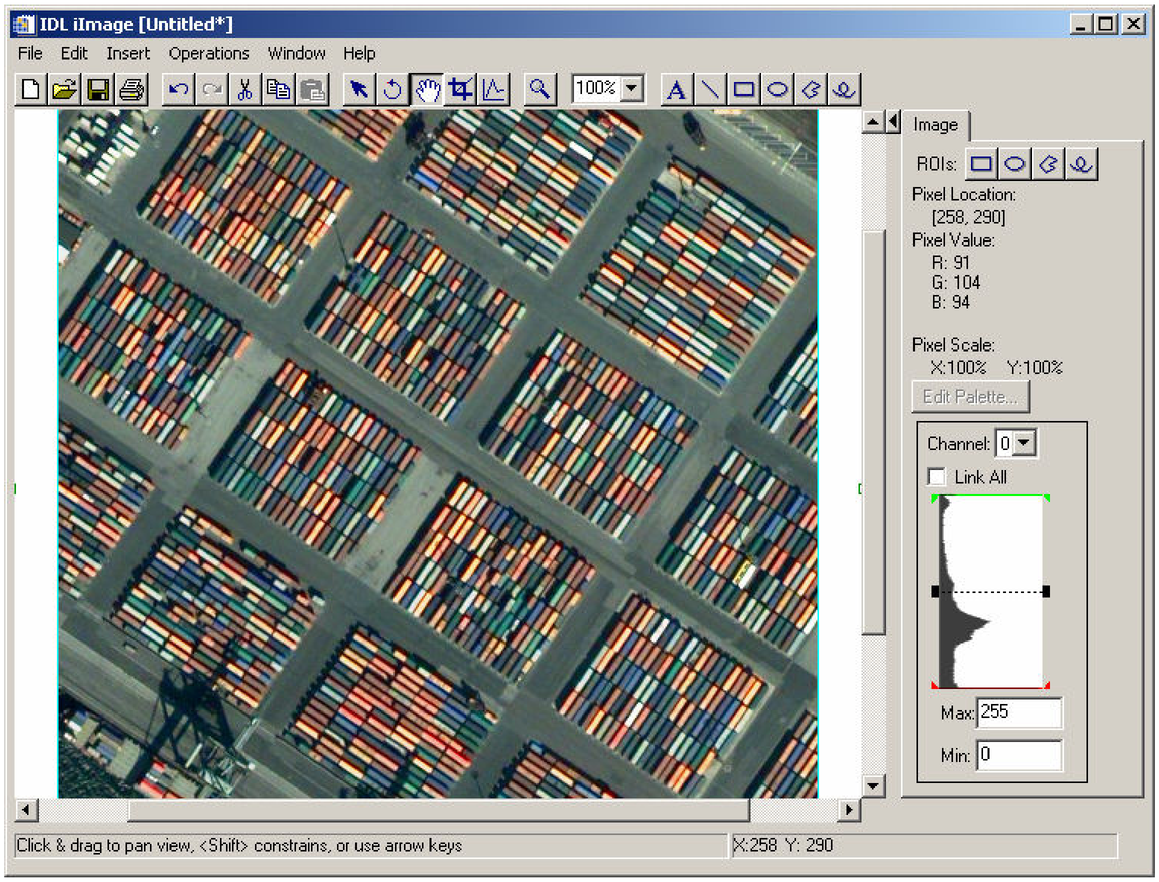
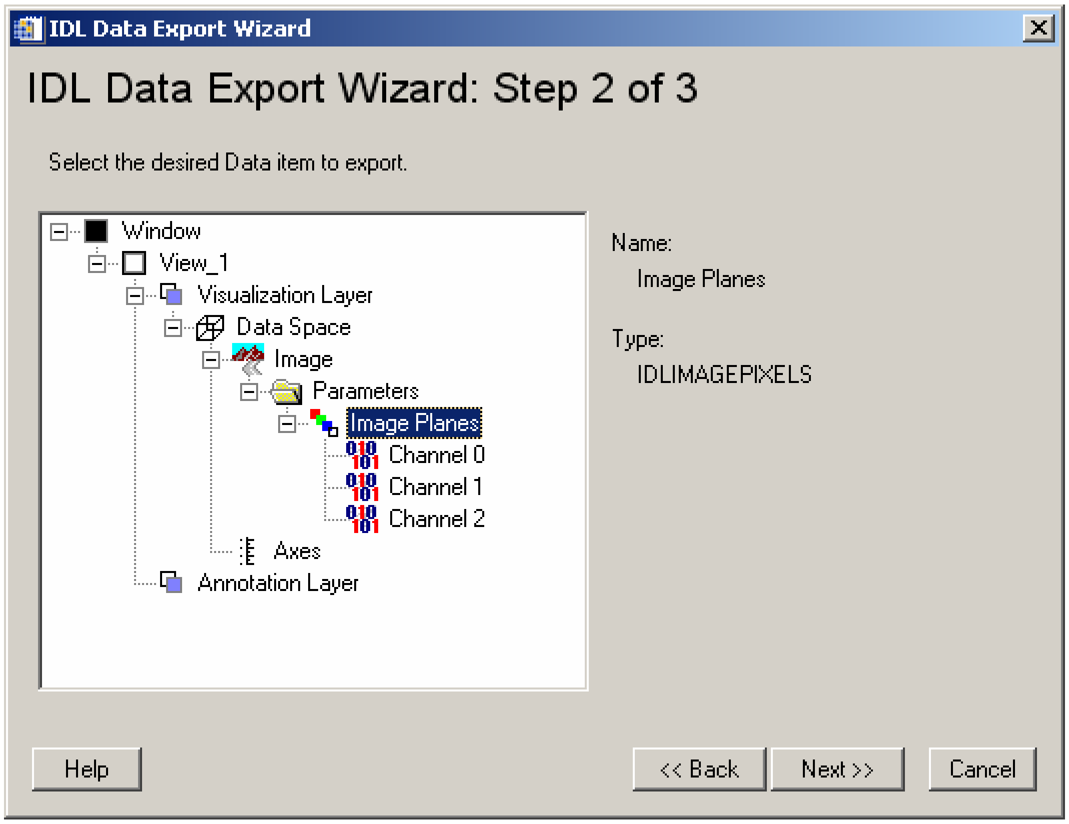


Figure 20: Image of the port in Hamburg, Germany

Once the image has been loaded into the *iImage* utility, it can be exported to an IDL variable for processing at the IDL> command prompt. Use the following steps to create a variable for this image at the main IDL level :

4. Select "*File > Export…*" from the *iImage* menu.
5. In Step 1 of 3 of the *IDL Data Export Wizard* select "To an IDL Variable" and press the "*Next >>*" button.
6. In Step 2 of 3 of the *IDL Data Export Wizard* select the "Image Planes" parameter and press the "*Next >>*" button [Fig. 21].
7. In Step 3 of 3 of the *IDL Data Export Wizard* change the "IDL Variable Name:" field to "*hamburg*" and press the "*Finish*" button.
8. Once this is accomplished, close the *IDL iImage* window and return to the main IDL Development Environment.


IDL Data Export Wizard

IDL Data Export Wizard: Step 2 of 3

Select the desired Data item to export.

```
Window
  View_1
    Visualization Layer
      Data Space
        Image
          Parameters
            Image Planes
              Channel 0
              Channel 1
              Channel 2
        Axes
    Annotation Layer
```

Name:
Image Planes

Type:
IDLIMAGEPIXELS

Figure 21: Step 2 of 3 of the IDL Data Export Wizard

A new variable named "hamburg" now exists at the main IDL level :

```
 9. IDL> HELP, hamburg
    HAMBURG         BYTE      = Array[3, 500, 500]
```

In order to work with this image in the frequency domain it is beneficial to extract the individual color channel images. This can be accomplished using IDL's standard array subscripting syntax in conjunction with the *REFORM* function, which is used to remove the first dimension (that has a size of one) and return a simple two-dimensional array :

```
10. IDL> r = REFORM (hamburg[0,*,*])
11. IDL> g = REFORM (hamburg[1,*,*])
12. IDL> b = REFORM (hamburg[2,*,*])
13. IDL> HELP, r, g, b
    R               BYTE      = Array[500, 500]
    G               BYTE      = Array[500, 500]
    B               BYTE      = Array[500, 500]
```

Note: Remember that the up-arrow and down-arrow keys on the keyboard can be used to perform command recall within IDL, which may be beneficial during these exercises.

Once this is accomplished, the *FFT* routine can be used to transform the image planes into the frequency domain :

```
14. IDL> rFFT = FFT (r)
15. IDL> gFFT = FFT (g)
16. IDL> bFFT = FFT (b)
```

The Fast Fourier Transform decomposes an image into sines and cosines of varying amplitudes and phases. The values of the resulting transform represent the amplitudes of particular horizontal and vertical frequencies. The data type of the array returned by the *FFT* function is complex, which contains real and imaginary parts :

```
17. IDL> HELP, gFFT
    GFFT            COMPLEX   = Array[500, 500]
```

The amplitude is the absolute value of the FFT, while the phase is the angle of the complex number, computed using the arctangent. In most cases, the imaginary part will look the same as the real part.

The image information in the frequency domain shows how often patterns are repeated within an image. Within the Fourier domain, low frequencies represent gradual variations in an image, while high frequencies correspond to abrupt variations in the image. The lowest frequencies usually contain most of the information, which is shown by the large peak in the center of the result. If the image does not contain any background noise, the rest of the data frequencies are very close to zero.

The results of the *FFT* function are often shifted to move the origin of the X and Y frequencies to the center of the display. Furthermore, the range of values from the peak to the high frequency noise is usually extreme. Consequently, a logarithmic scale is often utilized in order to visualize the image in the frequency domain. Since the logarithmic scale only applies to positive values, the **power spectrum** should be computed since it is the absolute value squared of the Fourier transform.

Visualize the power spectrum of the Fourier domain image for the green channel by executing the following statements :

```
18. IDL> center = 500 / 2 + 1
19. IDL> gShift = SHIFT (gFFT, center, center)
20. IDL> gPowSpec = ABS (gShift) ^ 2
21. IDL> gScaled = ALOG10 (gPowSpec)
22. IDL> iImage, gScaled, TITLE='Log-Scaled FFT Power Spectrum (G)'
```

The resulting *IDL iImage* visualization window should look similar to Fig. 22.

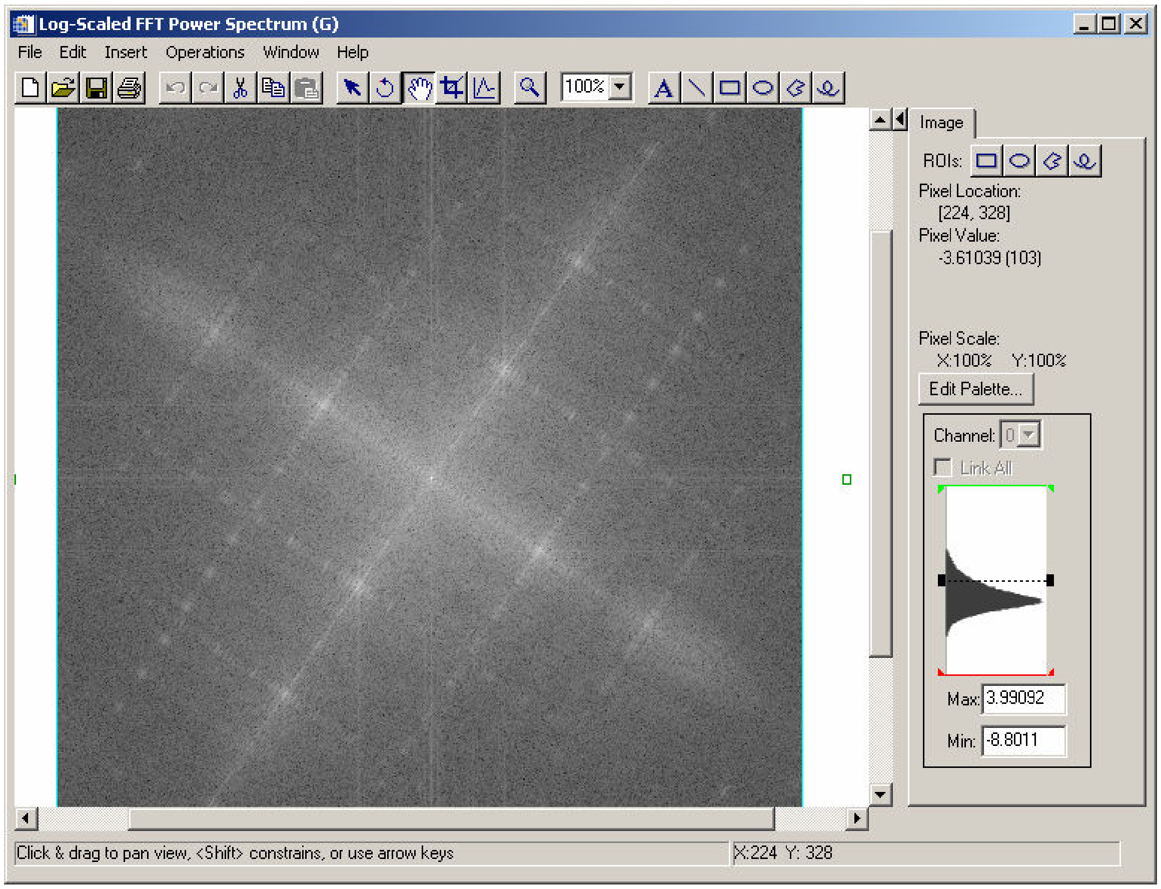


Figure 22: Power spectrum for the green channel image in the frequency domain

Notice the orientation of spatial patterns within the power spectrum image in both of the diagonal directions (just like the original image).

23. Once finished viewing the power spectrum image, close the *IDL iImage* window.

It may also be beneficial to visualize the power spectrum as a surface. Use the *REBIN* function to sub-sample the power spectrum in order to suppress some of the noise and set the shading for the surface to Gouraud :

24. IDL> `iSurface, REBIN (gScaled, 100, 100), SHADING=1`

The resulting *IDL iSurface* visualization window should look similar to Fig. 23.

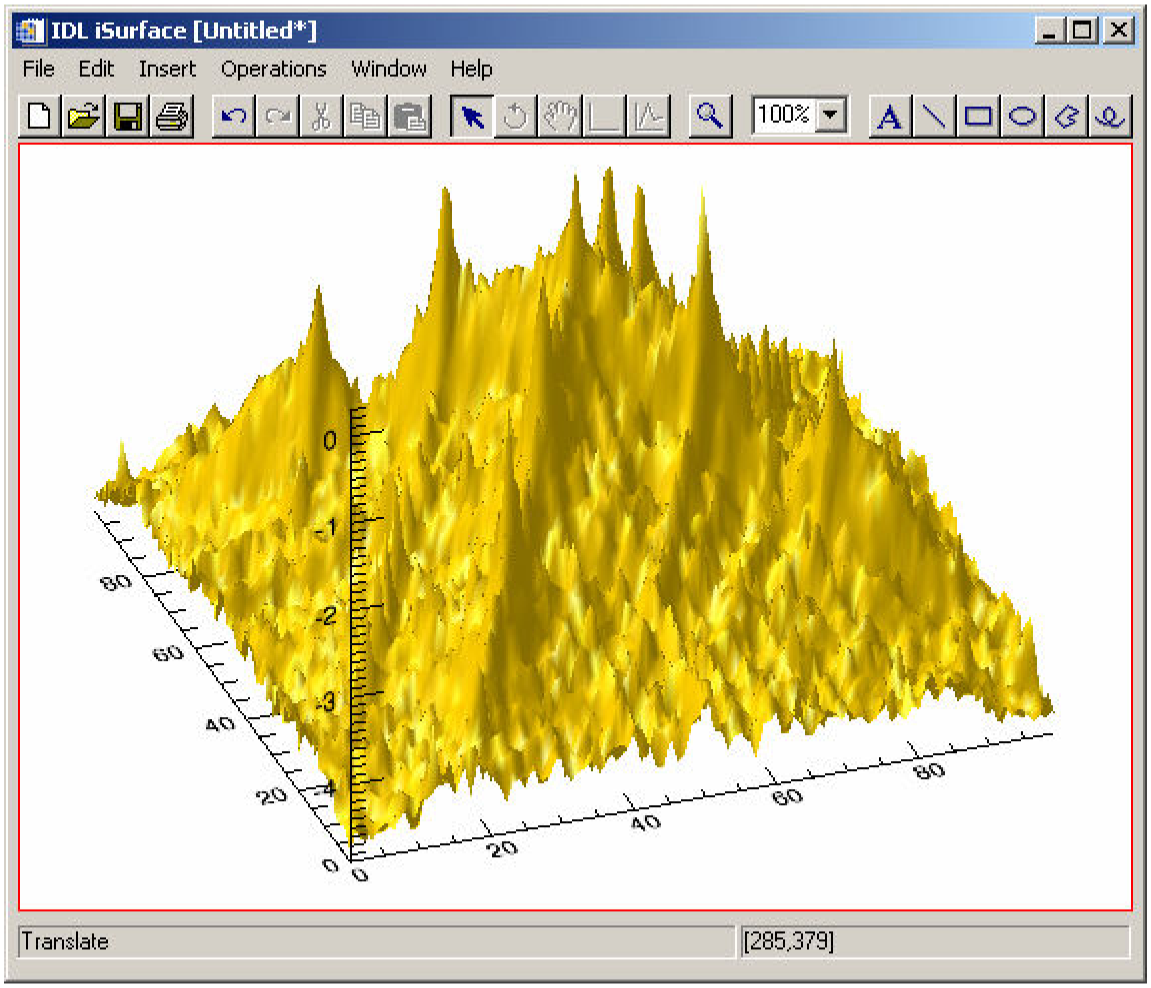


Figure 23: Power spectrum displayed as a surface

25. Once finished viewing the power spectrum surface, close the *IDL iSurface* window.

Low frequencies within the image tend to contain the most information because they determine the overall shape or patter in the image. High frequencies provide detail in the image, but they are often contaminated by the spurious effects of noise. Consequently, masks can be easily applied to an image within the frequency domain in order to remove noise.

Create a mask for the low spatial frequency components based on the highest values within the power spectrum for the green channel image :

26. IDL> `lsfMask = REAL_PART (gScaled) GT -2.5`

Note: The threshold value of –2.5 was arbitrarily selected based on the surface visualization above.

Visualize this mask by loading it into the *iImage* utility :

27. IDL> `iImage, BYTSCL (lsfMask)`

The resulting *IDL iImage* visualization window should look similar to Fig. 24.

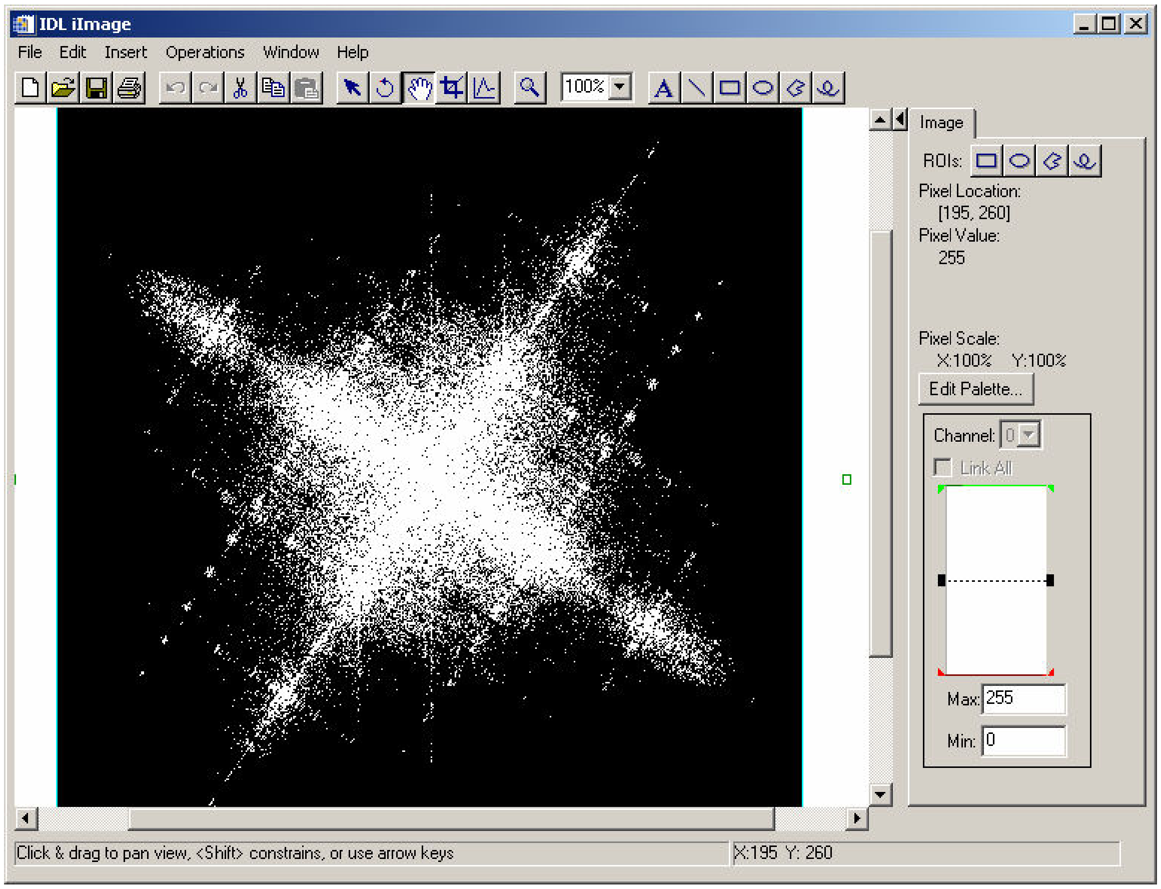


Figure 24: Mask of the low spatial frequency components (white) within the power spectrum for the green color channel

Notice that the low frequency components are found predominantly in the center of the power spectrum.

28. Once finished viewing the mask image, close the *IDL iImage* window.

In order to remove the high spatial frequency noise from the image, this mask must be applied to the Fourier transform data and then the inverse FFT must be computed. Applying the low spatial frequency mask allows these components to be converted back to the spatial domain during the inverse transform, while the high spatial frequency components are *masked out*.

First, the mask image must be shifted back to the original location of the Fourier transform :

29. IDL> `lsfMask = SHIFT (lsfMask, -center, -center)`

Once this is accomplished, the mask can be applied to the FFT results for the 3 color channels :

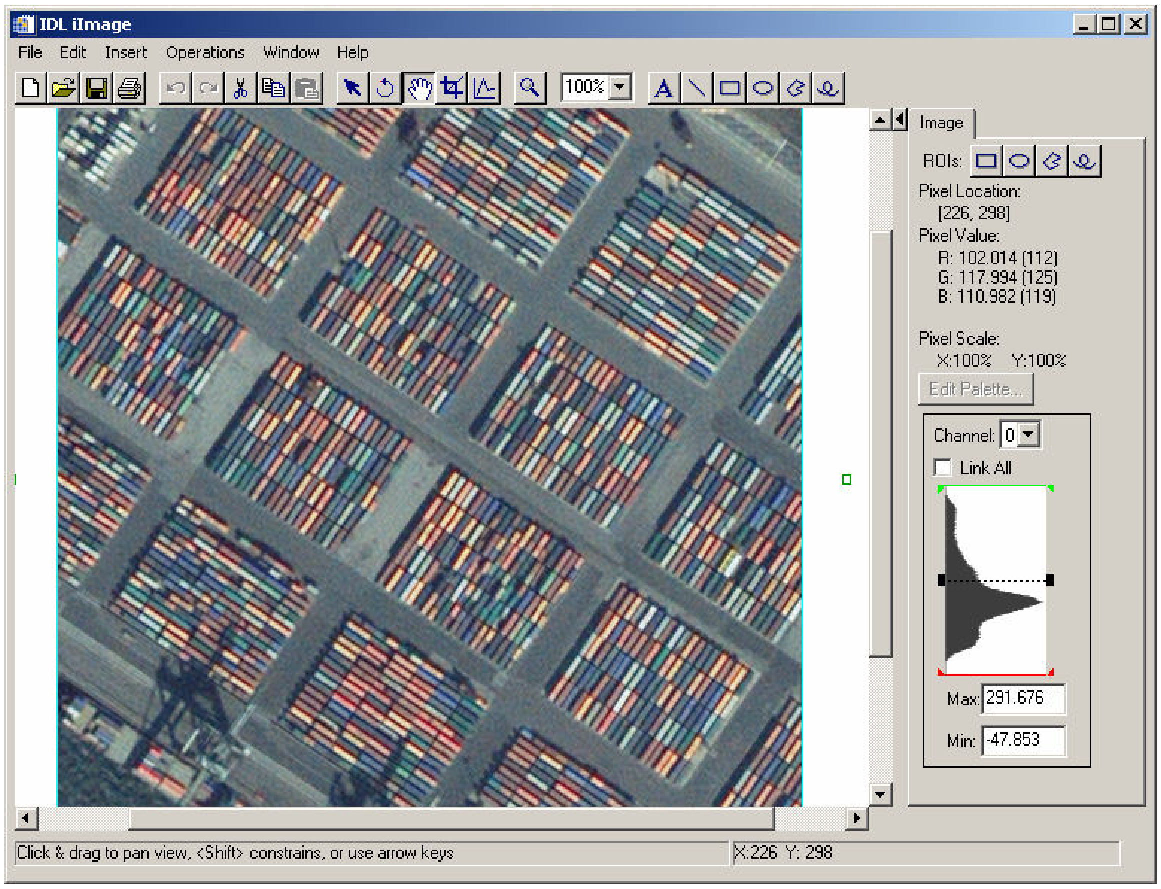
```
30. IDL> rMasked = rFFT * lsfMask
31. IDL> gMasked = gFFT * lsfMask
32. IDL> bMasked = bFFT * lsfMask
```

The inverse FFT can be used in conjunction with the *REAL_PART* function in order to convert the images back into the spatial domain :

```
33. IDL> rInvert = REAL_PART (FFT (rMasked, /INVERSE) )
34. IDL> gInvert = REAL_PART (FFT (gMasked, /INVERSE) )
35. IDL> bInvert = REAL_PART (FFT (bMasked, /INVERSE) )
```

The result can be visualized by loading the individual color channel images into the *iImage* utility [Fig. 25] :

```
36. IDL> iImage, RED=rInvert, GREEN=gInvert, BLUE=bInvert
```



Pixel Location:
[226, 298]
Pixel Value:
R: 102.014 (112)
G: 117.994 (125)
B: 110.982 (119)
Pixel Scale:
X:100% Y:100%
Channel: 0
Max: 291.676
Min: -47.853
Click & drag to pan view, <Shift> constrains, or use arrow keys
X:226 Y: 298

Figure 25: Result of the inverse FFT after the high spatial frequency components have been masked out (low pass filter)

37. Once finished viewing the inverse FFT image, close the *IDL iImage* window.

The high spatial frequency components of an image can also be enhanced using masking techniques in the frequency domain. A **circular-cut** (high pass) filter can be created by utilizing the DIST function in IDL and the appropriate threshold value :

38. IDL> `hsfMask = DIST (500) GE 50`

Visualize this mask by shifting it into the appropriate location and loading the result into the iImage utility [Fig. 26] :

39. IDL> `iImage, BYTSCL (SHIFT (hsfMask, center, center) ), $`
`BACKGROUND=[80,80,80]`

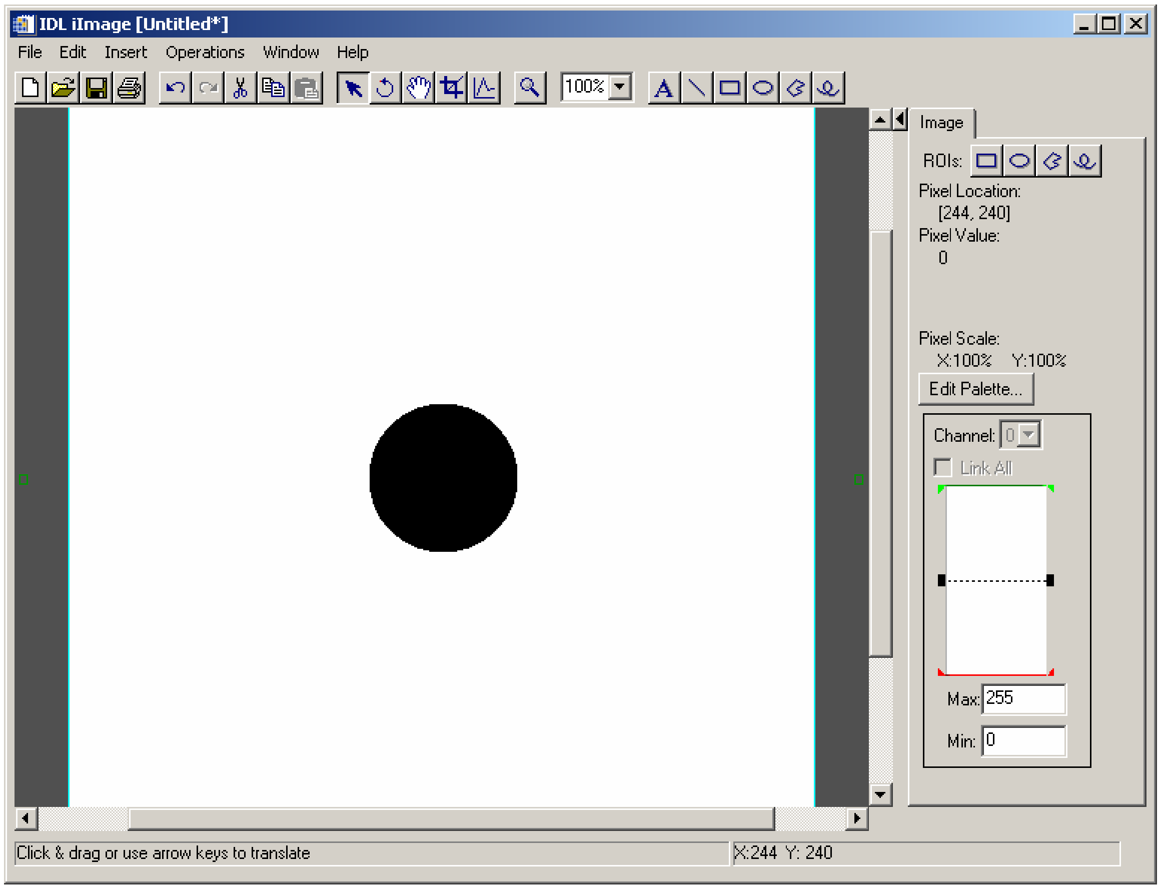


Figure 26: Circular-Cut filter for the high spatial frequency components (white) within the image

Notice that the high frequency components are found around the outer edges of the transform.

40. Once finished viewing the mask image, close the *IDL iImage* window.

Use the same methodology as before to apply the high pass filter, compute the inverse FFT, and display the result :

41. IDL> `rMasked = rFFT * hsfMask`
42. IDL> `gMasked = gFFT * hsfMask`
43. IDL> `bMasked = bFFT * hsfMask`
44. IDL> `rInvert = REAL_PART (FFT (rMasked, /INVERSE) )`

45. IDL> `gInvert = REAL_PART (FFT (gMasked, /INVERSE) )`
46. IDL> `bInvert = REAL_PART (FFT (bMasked, /INVERSE) )`
47. IDL> `iImage, RED=rInvert, GREEN=gInvert, BLUE=bInvert`

The resulting *IDL iImage* visualization window should look similar to Fig. 27.

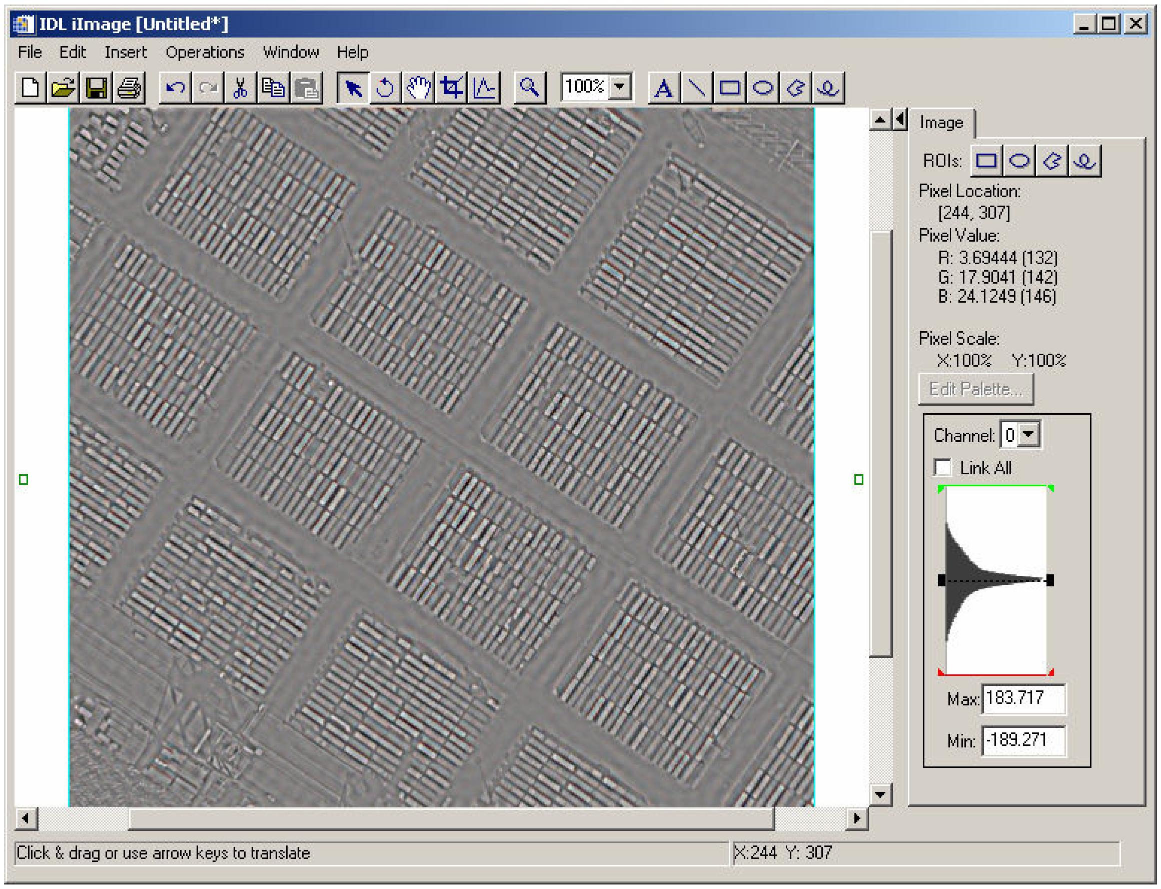


Figure 27: Result of the inverse FFT after the low spatial frequency components have been masked out (high pass filter)

48. Once finished viewing the inverse FFT image, close the *IDL iImage* window.