INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®
PROGRESSIVE TRANSMISSION OF
DIGITAL IMAGES

by

Joseph V. Kraft
Bachelor of Science
North Dakota State University
1990

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science Degree
Department of Electrical Engineering
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
December 2000
Thesis Approval
The Graduate College
University of Nevada, Las Vegas

November 17, 2000

The Thesis prepared by

__________________________
Joseph V. Kraft

Entitled

Progressive Transmission of Digital Images

is approved in partial fulfillment of the requirements for the degree of

Master of Science

__________________________
Examination Committee Chair

__________________________
Dean of the Graduate College

__________________________
Examination Committee Member

__________________________
Examination Committee Member

__________________________
Graduate College Faculty Representative
ABSTRACT

Progressive Transmission of Digital Images

by

Joseph V. Kraft

Dr. Shahram Latifi, Examination Committee Chair
Professor of Electrical Engineering
University of Nevada, Las Vegas

New high-resolution sensors produce very large data sets and associated images requiring large bandwidths or extended times to transmit. Other techniques have been proposed to compress and sort an image to send the most significant portions of the image first, allowing the user to choose the amount of compression applied based on transmission time; but they transmit the image as a whole.

Many users are interested in only portions of the image. This case is better served by a transmission method with capabilities to browse the image at low resolution to first pick the portion of interest, followed by transmission of only the portion of interest. The image is decomposed using wavelet multiresolution decomposition; and stored as an increasing succession of resolutions, each building on the resolution before it. This provides a framework to allow selecting only data required to reconstruct the portion of interest.
# TABLE OF CONTENTS

ABSTRACT .................................................................................................................... iii

LIST OF FIGURES ...................................................................................................... vi

ACKNOWLEDGMENTS ............................................................................................. vii

CHAPTER 1 INTRODUCTION .................................................................................1

CHAPTER 2 BACKGROUND ...................................................................................4
  WaveletTransforms .................................................................................................4
  Region Covering ....................................................................................................10
  Literature Survey ..................................................................................................11

CHAPTER 3 MULTiresolution IMAGE FORMAT ...............................................14
  How Image is Decomposed and Stored ...............................................................14
  Multiresolution Image Class Descriptions .........................................................15
  Multiresolution File Format ...............................................................................16

CHAPTER 4 PROGRESSIVE TRANSMISSION ...................................................18
  Additional File Headers ......................................................................................18
  Flow of Progressive Transmission .....................................................................21

CHAPTER 5 SOFTWARE DESCRIPTIONS .........................................................23
  DIB Converter .......................................................................................................23
  Netscape Plug-in .................................................................................................24
  CGI Application ...................................................................................................26

CHAPTER 6 SUMMARY, RECOMMENDATIONS AND CONCLUSION ............28
  Summary ............................................................................................................28
  Recommendations for Future Work .................................................................29
  Conclusions .........................................................................................................30

APPENDIX 1 APPLICATION SOURCE CODE ................................................32
  DIB Converter ......................................................................................................32
  Netscape Plug-in .................................................................................................34
  CGI Server Code .................................................................................................53
LIST OF FIGURES

Figure 1. Haar Scaling Function ................................................................. 5
Figure 2. Haar Wavelet .............................................................................. 5
Figure 3. Single dimension multiresolution decomposition ...................... 7
Figure 4. Two Dimension Multiresolution Decomposition ....................... 9
Figure 5. Recursive 2-Dimensional Multiresolution Decomposition ....... 9
Figure 6. Region Filling ........................................................................ 10
Figure 7. Region Covering Worst Case ................................................. 11
Figure 8. Initial Request Flow ............................................................... 21
Figure 9. Fill Request Flow ................................................................ 22
ACKNOWLEDGMENTS

The author wishes to acknowledge the support of his wife and children, they sacrificed the most and had the least to gain. Many hours of playtime and quality time together were missed in the completion of this thesis. The support provided by his advisor and the rest of his graduate committee and willingness to accommodate his unique situation was also crucial. Without their support and willingness to be flexible, this thesis would probably never have been completed.

Finally, the author also wishes to acknowledge the support from NASA (in part) under grant NAG5-3994
CHAPTER 1

INTRODUCTION

With the advent of very high-resolution sensors, especially in the astronomical sciences, data sets and associated images are becoming increasingly larger. The Internet provides an easy way to transmit the images, but the amount of data contained in the images can easily bog down all but high-speed connections. To compound the problem, researchers often sift through many images looking at each one in enough detail to determine if it is relevant to their work. A method of reducing the amount of data transferred over the net, without removing data of interest, is required to efficiently use these large data sets. The problem is how to cut down the amount of information to be transmitted down to just what is necessary.

A typical approach to this problem currently is to store a small thumbnail (usually JPEG) version of the image for browsing, using some kind of compression scheme to send the entire image if necessary. This approach suffers from storage space problems and is not as easily maintainable. Multiple copies of each image must be maintained, and the only lossless version is the full size copy.

Another approach developed recently is progressive transmission of the image. The image is decomposed and the bits reordered. Therefore, for any
given number of bits, the portion of the image transmitted will have the minimum mean square error (MSE). This method works well for images already appropriately scaled for the size of display. However, astronomical images are often too large to display in their original resolution, and often researchers are interested in only a portion of the original image anyway.

A better solution in this case lies beyond just maintaining thumbnails of all large images and sending compressed images. What really should be done is size the image appropriately for display and remove everything but the area of interest before sending. This is the basis for the form of progressive image transmission described in this thesis. It is based on a multiresolution decomposition of the original image using a discrete wavelet transform. The areas of the decomposed image that have been retrieved from the server are tracked and a region-covering algorithm is used to eliminate requests for data already retrieved.

While originally intended for use with astronomical images, this work can be easily extended or adapted for use in other areas. Any discipline using large images of high resolution could benefit from the techniques demonstrated here. Some possibilities are medical imaging, teleradiology and satellite imagery.

The applications developed to demonstrate progressive multiresolution image transmission have their roots in the simple wavelet transform and the simple algorithm for region covering discussed in Chapter 2. A simple wavelet transform is used to create a multiresolution decomposition that is worked into the multiresolution image format discussed in Chapter 3. Chapter 4 combines the
multiresolution image with the region covering technique to enable progressive transmission of the multiresolution image. The programs developed in Chapter 5 demonstrate how progressive multiresolution images can be applied to the large images common to astronomical sciences.
CHAPTER 2

BACKGROUND

This chapter will introduce some of the concepts used to create a multiresolution decomposition using Haar wavelets, as well as introducing a simple region-covering algorithm. The discrete wavelet transform is often used to create a multiresolution decomposition of an image. The brief discussion of wavelet transforms is intended to give a general understanding of how a wavelet transform works; for more in-depth information see references [1], [5] and [7] in the Bibliography. A very simple form of the wavelet transform often used for demonstration and educational purposes is the Haar wavelet, which is presented next. Finally, a simple region covering method using rectangles and a “divide and conquer” scheme is discussed.

Wavelet Transforms

Wavelet transforms can be viewed as extensions of Fourier Transforms. The basic premise is that a signal can be decomposed to a sum of wavelets of different sizes and positions; just like the sines and cosines used in Fourier analysis. The wavelets are squeezed or stretched (dilated) to change their frequency; this dilation also adapts the wavelet to capture different frequency components of the signal. High frequency components are captured using a
high-frequency dilation of the wavelet with a small sampling window and low
frequency components are captured using a low-frequency wavelet with a large
sampling window. Each group of scaled wavelets (a basis) contains an infinite
number of wavelets formed by dilating and translating (shifting) the basic
waveform (mother wavelet). The scaling function used to create the wavelet
basis from the mother wavelet is very important in multiresolution decomposition.

Restricting the scaling and translating of the mother wavelet to discrete
amounts can create a discrete version of the wavelet transform. Most often
discrete wavelets are constructed where dilation is by a power of two. The
reasons for this will become evident after looking at how we construct a
multiresolution decomposition.

**Haar Wavelet**

The Haar Wavelet is a simple wavelet often used for demonstrational and
educational purposes. The Haar scaling function is shown in Figure 1, and the Haar
wavelet is shown in Figure 2. Translated and applied to a signal, the Haar scaling
function will give the average value of the data in the interval covered by the scaling
function. The wavelet gives half the difference between two adjacent values of
the data.

![Figure 1. Haar Scaling Function](image1)

![Figure 2. Haar Wavelet](image2)
**Multiresolution Decomposition**

The scaling function takes on much greater importance in the area of multiresolution image analysis. By using discrete wavelet transforms and judicious selection of the scaling function and wavelet, we can decompose the image into a series of images that differ by a multiple of two in resolution. A multiresolution representation can be created without redundancy and with arbitrary precision if the following four conditions are met:

1. **Translation.** The scaling function must be orthogonal to its translates by integers.

   \[
   \text{if } f(x) \in V_j \text{ then } f(x + k2^{-j}) \in V_j \quad \text{Equation 1}
   \]

2. **Nestedness.** The image at a given resolution contains all the information of the signal at lower resolutions.

   \[
   V_j \subset V_{j+1} \quad \text{Equation 2}
   \]

3. **Dilation.** The function 0 is the only object common to all the spaces, \( V_j \).

   \[
   \lim_{j \to \infty} V_j = \bigcap V_j = \{0\} \quad \text{Equation 3}
   \]

4. **Completeness.** Any signal can be approximated with arbitrary precision.

   \[
   \lim_{j \to \infty} V_j = L^2(\mathbb{R}) \quad \text{Equation 4}
   \]

The Haar wavelet as described by Haar[6] and Frische[3] meets these conditions for creating a multiresolution decomposition[7]. The scaling function provides an average of the values covered by the function. The Haar wavelet captures the difference between the first half and the last half of the values. When dilated by a factor of two the transform acts on two adjacent values. When
dilated by a factor of four, the scaling function averages the four values. The wavelet captures the difference between the sum of the first two values and the sum of the last two values.

Figure 3 demonstrates a one-dimensional multiresolution decomposition. One starts with a row of an even number of data points and applies the scaling function and the wavelet to each pair of data points. Then, the results of all the scaling function calculations are stored followed by the results of all the wavelet calculations. The first half (low frequency part) will be a ½ size representation of the original. The second half (high frequency part) must be retained to recombine with the first half to reconstruct the original. To obtain an even coarser representation of the original data set, the process can be repeated again on each resulting low frequency part. The process can be continued to the point where only one value remains.

Figure 3. Single dimension multiresolution decomposition.
Extending this into two dimensions is very similar to two-dimensional Fourier transforms. One can use the single dimensional transform to transform the rows, then use the same transform to transform the columns, or vice-versa[1]. The Haar Transform discussed above can be used to construct a two-dimensional transform known as the H-Transform[11]. The basic H-Transform for a $2^N \times 2^N$ image is as follows:

- Divide the image up into blocks of $2 \times 2$ pixels. Call the four pixels in each block $a_{00}$, $a_{10}$, $a_{01}$, and $a_{11}$.
- For each block compute 4 coefficients:
  \[
  h_0 = \frac{(a_{11} + a_{10} + a_{01} + a_{00})}{2} \quad \text{Equation 5}
  \]
  \[
  h_x = \frac{(a_{11} + a_{10} - a_{01} - a_{00})}{2} \quad \text{Equation 6}
  \]
  \[
  h_y = \frac{(a_{11} - a_{10} + a_{01} - a_{00})}{2} \quad \text{Equation 7}
  \]
  \[
  h_c = \frac{(a_{11} - a_{10} - a_{01} + a_{00})}{2} \quad \text{Equation 8}
  \]
- Construct a $2^{N-1} \times 2^{N-1}$ image from the $h_0$ values for each $2 \times 2$ block. Divide that image into $2 \times 2$ blocks and repeat the above calculation.
- Repeat this process $N$ times, reducing the image in size by a factor of 2 at each step, until only one $h_0$ value remains.

The H-Transform creates a two-dimensional multiresolution decomposition. Each iteration of $h_0$ data is $\frac{1}{4}$ the size of the previous iteration.

The data set is broken down into four pieces, each axis having a low frequency part and a high frequency part. The part containing the data points calculated by applying the scaling function in both $x$-axis and $y$-axis is denoted $LL$ because it contains the low frequency portion of both axes. It is a small ($1/4$ size)
representation of the original. The other three parts (HL, LL and HH) contain various residual values.

Figure 4 shows the progression from an image through the x-axis transform and then the y-axis transform. Assuming the low frequency information is to be concentrated to the top and left.

Like the single dimensional case, the transform can be repeated on the LL block to obtain a yet smaller data set (Figure 5). Each iteration is sometimes known as a subband.
Region Covering

The basic principle of a region-covering algorithm is to find the portion of a region covered or not covered by another region. There have been a number of algorithms presented on this topic for general region filling. A very simple "divide and conquer" method can be used in the case where all regions are rectangles. It is not the most elegant, but it is easy to understand.

To start, one locates an edge of the covering rectangle that crosses the covered rectangle. Divide the covered rectangle in two parts on that line. One part will still be partially covered, but the other will not be covered. Continue the same process with the rectangle that is partially covered, splitting out uncovered rectangles until the remaining portion is completely covered by the covering rectangle.

Figure 6. Region Filling.
The worst case for this method is when the covering rectangle is smaller than the covered rectangle, and completely contained within the covered rectangle (Figure 7). In any case, with a single covering rectangle at most four rectangular regions are required to completely cover the uncovered area.

**Literature Survey**

There have recently been two other research groups publishing papers on the subject of image compression, decomposition and progressive transmission of astronomical images. Dr. Aaron Kiely (NASA's Jet Propulsion Laboratory) and the team of Dr. Jeff Percival and Dr. Robert White (University of Wisconsin and Space Telescope Science Institute) both have approached the issue with similar yet distinct methods. Kiely’s report [8] discusses his approach as an image data compression strategy featuring progressive transmission. His method uses multiresolution decomposition, followed by arithmetic coding. Before transmission, the data is ordered so the data having the most effect on the quality of the image is sent first. Percival and White’s approach [11] is a three-part process similar to Kiely’s. The significant difference is the first step, to
equalize the noise levels across the image based on known noise levels for the data.

Kiely’s approach views the entire image at a constant size and at a resolution that increases as more data is transmitted. He initially uses an eighth order quadrature mirror filter to break down the image. The principle of representing filter banks as wavelets is discussed in [4]. In this case a quadrature mirror filter can be viewed as a 2-dimensional wavelet transform. Kiely makes note that for what he was doing, the actual transform does not matter as long as it decomposed the image into subbands.

From the subband decompositions each bitplane of each subband is encoded separately and the data is ordered by which bitplane will increase the quality (measured by mean square error (MSE)) the most per transmitted bit. When transmitted, the subbands are not necessarily transmitted in order. They are also not necessarily transmitted completely before the next subband is started. The transmission will alternate between subbands depending on which bitplane will increase the quality of the image the most.

Percival and White’s three-part process for compression and progressive transmission starts with intensity mapping to create an image with roughly equal noise in each pixel. This allows better compression of the transformed data. After the first step, the process closely follows Kiely’s method. A wavelet transform, in this case the H-Transform, is used for the multiresolution decomposition. Each plane of each subband is encoded separately, as Kiely did, but using quadtree encoding. Quadtree encoding was chosen over arithmetic
encoding in this case because it was computationally simpler, and the increased computational complexity of the arithmetic encoding was not justified for the small gain in compression.

Both of these methods for using multiresolution decomposition to enable progressive transmission of an image are targeted to applications where the images are already scaled appropriately for display. They address the issue of viewing an entire image in varying levels of quality and focus on increasing the quality at the highest rate per transmitted bit possible. The progressive transmission application to be discussed in this thesis involves scaling very large images for viewing in a relatively small (compared to their overall size) area thus requiring reduced viewing resolution. To display a high-resolution view of the image in a small area, only a portion of the image may be displayed at a time. One may also require a tradeoff in overall viewable image size and resolution to something in between the two extremes. This application is the focus for the rest of this paper.
CHAPTER 3

MULTIRESOLUTION IMAGE FORMAT

In order to take advantage of the properties of multiresolution decomposition, this chapter presents a new image format based on the multiresolution decomposition discussed in Chapter 2. It stores the image portions broken down by resolutions varying in size by a factor of two, from the full image down to the size chosen for the thumbnail image. The resolutions are stored in a building block format so the data contained in the lower resolutions is not repeated in the higher resolutions. The multiresolution format is implemented as a series of C++ classes that isolate the transform used for the decomposition. For testing/development purposes decomposition based on the Haar wavelet transform was used. Other transforms can be implemented and substituted for the Haar transform allowing optimization for a specific class of images.

How the Image is Decomposed and Stored

The data for each resolution is stored in up to four blocks. The image is decomposed using the H-Transform discussed in Chapter 2, with the exception of the division by two not being applied. This modification allows decomposition and recreation of each pixel using integer math. Each data store is sized so it is ½ the size in each dimension of the original image to be transformed. The
formulas are applied to each block of four pixels and the result stored in the data store for the thumbnail \((h_0)\), or the residual data stores \((h_x, h_y, h_C)\). The thumbnail is used as the original image for the next iteration of the decomposition. Once the next level of decomposition has been accomplished, the thumbnail no longer necessary because it may be reconstructed from the smaller thumbnail and its associated residual data blocks. The redundant thumbnail is deleted.

Multiresolution Image Class Descriptions

The multiresolution image structure is implemented as a set of C++ classes. The basic CImage class provides a common base for creating other image classes. The CMultiresImage class is built on the CImage class, integrating the transform provided and implementing the multiresolution image. The transform module implements the functions to conduct the transform and inverse transform on blocks of data.

CImage provides the basic framework of an image, which include height, width, data type and number of bits per pixel. It creates a data store sized appropriately to accommodate the type and amount of data in the image. It is a virtual class requiring the addition of functions to read and write a header to complete a usable class.

The CMultiresImage class provides the basic functionality of the multiresolution image. It is based on CImage, which provides the basic image framework, and extends it to support multiresolution images. The class is responsible for image management, such as linking the resolutions together in a list and managing the residual data stores, (the thumbnail is stored in the CI mage
data store). It also handles the file header and data reading/writing, as well as the progressive transmission functions discussed in the next chapter.

The Transform module contains the functions necessary for CMultiresImage to perform the chosen transform and inverse transform. This is the only module requiring changes to change the type of the transform used by CMultiresImage.

Multiresolution File Format

A format for a multiresolution image file was designed to store an image in the decomposed format. This allows an image to be decomposed one time instead of decomposing the image every time it is used. The format contains a file header (Table 1) followed by the image data. The image data is stored in order of increasing resolution, so the thumbnail (LL) data is stored first followed by the x-residual (HL), and the y-residual (LH) and the common-residual (HH) data for each resolution. So using the labels shown in Figure 5 (Pg. 9), the file would be composed of {Header, LL2, HL2, LH2, HH2, HL1, LH1, HH1} in that order.

The data is written as it is stored in memory (row by row), each pixel data point is a 32 bit number composed of the red, green and blue values as unsigned bytes and one reserved byte. The residuals are stored the same, except that the red, green and blue values are signed bytes. There is currently no data compression applied to the data in the file.
This arrangement of data allows reading the image from the smallest to the largest resolution sequentially from the front of the file. If one is only interested in the thumbnail of the image, one can stop reading the file after loading the thumbnail. The data in the header provides all the information necessary to initialize the image structure prior to reading the data, nothing must be assumed.

Table 1. Multiresolution Image File Header.

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wFileHeaderSize</td>
<td>WORD</td>
<td>File header size in bytes.</td>
</tr>
<tr>
<td>wNumRes</td>
<td>WORD</td>
<td>Number of resolutions in the complete image</td>
</tr>
<tr>
<td>lOrigWidth</td>
<td>LONG</td>
<td>Width of the original image in pixels.</td>
</tr>
<tr>
<td>lOrigHeight</td>
<td>LONG</td>
<td>Height of the original image in pixels.</td>
</tr>
<tr>
<td>IThumbWidth</td>
<td>LONG</td>
<td>Height of the thumbnail image (smallest resolution) in pixels.</td>
</tr>
<tr>
<td>IThumbHeight</td>
<td>LONG</td>
<td>Height of the thumbnail (smallest resolution) in pixels.</td>
</tr>
<tr>
<td>wDataType</td>
<td>WORD</td>
<td>Value indicating the data type stored in the thumbnail. (Only value currently valid is JVK_RGB)</td>
</tr>
<tr>
<td>wBitsPerPixel</td>
<td>WORD</td>
<td>The number of bits required to store each pixel.</td>
</tr>
</tbody>
</table>
CHAPTER 4

PROGRESSIVE TRANSMISSION

The main purpose of developing the multiresolution image decomposition was to provide the base for the progressive image transmission developed in this chapter. Progressive transmission is used in applications where it is not necessary or not feasible to transmit the entire image through the data channel. To limit the amount of data sent, the image is sent in a size and resolution to be displayed directly in the available window. If one is only interested in the thumbnail of the image, only the thumbnail should be sent. The desire to limit the amount of data transferred also dictates that the data already sent to the client needs not be sent again. The scheme used for requesting additional information from the server tracks data already received by the client. Then, the client uses the region-covering algorithm to formulate requests to fill in areas that do not include the areas already received. The data received by the client is used to fill in the image data necessary to display the area of interest at an appropriate resolution.

Additional File Headers

The progressive transmission of multiresolution images has two distinct parts: initial data and fill-in data. The initial data sent to the client includes the
Table 2. Initial Progressive Multiresolution Header.

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wProgLongHeaderSize</td>
<td>WORD</td>
<td>File header size in bytes.</td>
</tr>
<tr>
<td>wNumRes</td>
<td>WORD</td>
<td>Number of resolutions in the complete image.</td>
</tr>
<tr>
<td>wResSent</td>
<td>WORD</td>
<td>Number of resolutions of residual data sent in this file (0 if just thumbnail).</td>
</tr>
<tr>
<td>lOrigWidth</td>
<td>LONG</td>
<td>Width of the original image in pixels.</td>
</tr>
<tr>
<td>lOrigHeight</td>
<td>LONG</td>
<td>Height of the original image in pixels.</td>
</tr>
<tr>
<td>IThumbWidth</td>
<td>LONG</td>
<td>Height of the thumbnail image (smallest resolution) in pixels.</td>
</tr>
<tr>
<td>IThumbHeight</td>
<td>LONG</td>
<td>Height of the thumbnail (smallest resolution) in pixels.</td>
</tr>
<tr>
<td>wDataType</td>
<td>WORD</td>
<td>Value indicating the data type stored in the thumbnail. (Only value currently valid is JVK_RGB which is 32 bit RGB)</td>
</tr>
<tr>
<td>wBitsPerPixel</td>
<td>WORD</td>
<td>The number of bits required to store each pixel.</td>
</tr>
</tbody>
</table>

Information needed to build the image structure and data for the initial portion of the image to be displayed. The second part is the fill-in data sent to add information to the image. This data may send the next resolution, if a user is zooming into the image, or it could send more of the current resolution if the user is panning to a previously undisplayed area. To support the two distinct parts of image transmission, two more file headers have been created.

The first header supports the initial data, and includes all the fields in the original file header; but adds a field informing the client how many complete resolutions of data to expect in the initial image (Table 2).
The second header is used for the fill-in data returned by each request to the server for additional data. Its structure is much simpler since it does not contain all the information to initialize the data structures. The only information the header needs to carry is the position information to insert the data into the correct storage locations in the image structure (Table 3).

The three different header types are differentiated by the first two bytes (WORD) in the data file or stream. The first WORD in all three headers contains the size of the header; each size is distinct. The read file/stream functions in the class will take different actions based on the header sensed. If the original file header is detected, the image structure is reset and the entire image is read from the file and marked valid. If the initial progressive multiresolution header is detected, the image structure is reset. However, only the number of resolutions indicated in the header are read and marked valid. The data in unread resolutions is left uninitialized and is not marked valid. Finally, if a fill-in progressive header is detected, the image structure is not reset, the data is

Table 3. Fill-in Progressive Multiresolution Header.

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wProgShortHeaderSize</td>
<td>WORD</td>
<td>File header size in bytes.</td>
</tr>
<tr>
<td>wRes</td>
<td>WORD</td>
<td>Resolution to which this data belongs.</td>
</tr>
<tr>
<td>rect</td>
<td>RECT</td>
<td>Rectangle structure (composed of 4 LONGs: left, top, right and bottom in that order) that defines the bounding rectangle for the data sent.</td>
</tr>
</tbody>
</table>
written to the appropriate place in the data stores, and only the area read is added to the list of valid areas.

Flow of Progressive Transmission

Initial Structure and Thumbnail

The server CGI script responds to the initial request for an image from the client with the initial progressive multiresolution header and at least the thumbnail resolution. The entire thumbnail is always sent in the initial portion of the image, and more complete resolutions of data may be sent if necessary. The client reads the response from the server and initializes the image structure. It then reads the thumbnail (Figure 8) and as many resolutions of residual data as were included in the initial portion of the image.

![Figure 8. Initial Request Flow](image)

Determining What Fills are Required

The client receives and stores the data it has requested and therefore is in the best position to know what data it has locally and what portions of the image have not been transmitted from the server yet. The valid rectangles are
maintained in a linked list attached to each resolution of residual data. The client confirms the region of the image it wishes to use is valid, that is, all the data required to reconstruct the area of interest has been sent from the server. If the image is not completely valid, the client uses the region-filling algorithm outlined in Chapter 2 to determine which areas need to be requested from the server.

**Fill-in Sections**

Once the areas have been defined, the requests are sent to the server (Figure 9). The server responds to each request individually with the information to fill in the portion of the image requested. The data is written to the appropriate place in the image structure and the appropriate area is marked valid. Once all requests have been filled and stored, the entire area of interest is valid and usable.
CHAPTER 5

SOFTWARE DESCRIPTIONS

Three applications were developed to demonstrate the utility of progressive image transmission. To create some multiresolution images to work with during development and testing, the first application was designed to convert an MS Windows device independent bitmap (DIB) to a multiresolution image. Next on the design list was the image browser client, implemented as a Netscape Plug-in, to read and display the multiresolution images. Finally, the server portion of the image browser was implemented as CGI code to be run on an existing World Wide Web (WWW) server. These applications were developed only to demonstrate the use of a multiresolution image and the utility of progressive transmission; they are in no way finished products.

DIB converter

The DIB converter does just what it sounds like it should do—convert DIBs to multiresolution images. It reads a DIB image using the CDIBImage class (another image class based on CImage), decomposes it into a multiresolution image and writes it to stdout or a file.

Because of the way the multiresolution image file format is currently defined, a 32-bit DIB remains about the same size when converted, but any other format
of DIB is expanded. In extreme cases with DIBs using less than 256 colors stored in a color table, this expansion can be very large; almost 300% in some cases. The 24-bit DIB is more efficient in terms of storage space than the 32-bit DIB. They both provide the same number of available colors, but the 24-bit DIB does not store the last empty byte of data for each pixel. A 24-bit DIB is expanded by 33% when stored using 32-bit data blocks for each pixel.

The original decision to use 32 bit data in the design of this software was because of a mistaken assumption that most DIBs were already in a 32-bit format. Its use was continued because it is easier to read the data 32 bits at a time. A substantial portion of the code was already completed and would have had to been rewritten to change it.

Netscape Plug-in

The Netscape Plug-in implements the client portion of the image display software. It was chosen because the Netscape browser is available for many different platforms, and provides most of the networking and data transmission functionality. The code for the plug-in is mostly portable from platform to platform; only the MS Windows specific routines need to be replaced by their platform specific counterparts. The plug-in reads the data and displays the thumbnail of the image, unless a specific resolution and region for display were requested in the <EMBED> HTML tag (Table 4) used to call the plug-in.
Table 4. Plug-in <EMBED> Tag format.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC</td>
<td>Initial URL to load. (Note this also includes a reference to the initial image to load)</td>
</tr>
<tr>
<td>HEIGHT</td>
<td>Height of plug-in window.</td>
</tr>
<tr>
<td>WIDTH</td>
<td>Width of plug-in window.</td>
</tr>
<tr>
<td>IMAGE</td>
<td>Name of the image on the server to retrieve</td>
</tr>
<tr>
<td>RES</td>
<td>Resolution for initial display.</td>
</tr>
<tr>
<td>LEFT</td>
<td>Defines the left edge of area to display.</td>
</tr>
<tr>
<td>TOP</td>
<td>Defines the top edge of area to display.</td>
</tr>
<tr>
<td>RIGHT</td>
<td>Defines the right edge of area to display.</td>
</tr>
<tr>
<td>BOTTOM</td>
<td>Defines the bottom edge of area to display.</td>
</tr>
</tbody>
</table>

The plug-in allows two actions to be taken on the image; a left mouse click to zoom in and a right mouse click to zoom out. The left mouse click will expand the image to the next resolution. The size of the image will be doubled, if the plug-in’s display area is large enough to accommodate the new size. If the display area is not large enough, the image will be clipped to fit. Once the area of the image to be displayed is calculated, the area is checked to see if any image data needs to be requested from the server. Once the plug-in has all the data it needs, it displays the new version of the image. The right mouse click will invoke similar actions, except the image resolution is halved instead of doubled.

The plug-in can also be used to read a local file. Since it is not working through the server, and has no way to request specific portions of the image, the
entire file is read into memory. In this case the plug-in is not embedded in a page of HTML so it is not constrained in size, it may use the entire size of the browser window. The plug-in’s initial display and the mouse clicks function the same.

When the plug-in is embedded using the HTML <EMBED> tag, the format for the tag is shown in Table 4. The required arguments are SRC, IMAGE, HEIGHT and WIDTH. If no initial resolution is given, the plug-in displays the thumbnail.

CGI Application

The server portion of the image display software is implemented as a CGI script to be used with an existing WWW server. The CGI script does not use any platform specific code, and should be easily portable to any platform with a C++ compiler. The CGI script developed for use in this project is very basic, with only a few inputs. It will reply to a specifically formatted request with the correct portion of an existing image. A more complex script could include conversion of other file formats to multiresolution images prior to use. It could also include (with corresponding changes to the plug-in) compression on the data stream to further reduce the amount of data transferred.

The CGI script will reply to a request with either the initial progressive multiresolution image header or the fill-in progressive multiresolution image header, depending on the arguments when it is called. If the script is called with a URL that includes an image (i.e. /scripts/multiresCGI.cgi?test.jvk), it will return the initial data to set up the image structure. If the script is called with the image name passed as a separate argument, accompanied with a resolution and
bounding rectangle (i.e. /scripts/MultiresCGI.cgi?IMAGE=test.jvk&LEFT=0&RIGHT=40&TOP=0&BOTTOM=30&RES=1); the script will return the fill data associated with the resolution and bounding rectangle.
CHAPTER 6

SUMMARY, RECOMMENDATIONS AND CONCLUSIONS

Summary

The programs developed to demonstrate progressive multiresolution image transmission have their roots in a simple wavelet transform and a simple method for region covering. The Haar wavelet is a very simple wavelet used to create the multiresolution decomposition of an image. This decomposition is the basis for the multiresolution image format where each resolution of the decomposition is stored separately in the image structure using a building block approach. The first resolution is reconstructed using the thumbnail image and the first set of residual data. Each subsequent resolution is reconstructed using the previous reconstructed resolution and the next set of residual data. The first program was developed to demonstrate the decomposition of a bitmap image creating a multiresolution image file.

Progressive transmission uses the multiresolution image to provide a set of discreet resolutions for the image and uses region covering to identify regions of the image to requiring data from the server. The second and third programs developed were the client and server portions of an image browser demonstrating one use of progressive transmission. An image browser is an
ideal program for demonstrating progressive transmission because many of the applications for progressive transmission are image related. In this case images are sent originally only as small thumbnails, with the rest of the image sent to the client only if necessary.

Recommendations for Future Work

Compression of Storage and Data Stream

The largest outstanding issue with the classes the way they are written now is the data storage, both internal and external. Using a 32-bit RGB representation as defined by Microsoft is a waste of storage space, both in memory and on disk. The 32-bit representation uses only 24 bits for storing the data reserving 8 bits for future use. Using a 24-bit RGB representation would save 25% of the space required to store the data in memory and on disk. To further decrease the amount of data passed through the transmission medium, some form of compression could be applied to the data stream at the server and decoded at the client.

Replacement of Transform

The transform used to decompose the image is the single biggest factor in the image quality of the lower resolutions. Choosing a transform that creates residual values very near zero will concentrate more of the information contained in the image into the thumbnail and usually provide better looking images. There is no single transform to create the best thumbnails for all images. However, some research to determine a wavelet transform providing good separation for a particular class of images will improve the results for that class of images.
Region Covering Algorithm

The region-filling algorithm currently incorporated into the multiresolution image class is a very simple "divide and conquer" scheme. A more advanced algorithm may provide a cover using fewer rectangles resulting in fewer data requests for the server to process.

Incorporate Other Image Formats for Conversion

The last major issue is to build image classes based on CImage for other common image formats. Providing these classes will allow easy conversion of a multiresolution image to and from other common formats such as FITS, JPEG and others.

Conclusions

The multiresolution image class developed is a useful tool for developing applications using progressive image transmission. The class is intended to be used with the wavelet predetermined to provide the best decomposition of image data for a particular class of images. The images may be decomposed losslessly (assuming a lossless transform is used) and reconstructed to any degree necessary to meet the needs of the user. All the information to reconstruct the lossless image is available from the server, if the user is willing to accept some loss, the image need not be transmitted at the highest resolution available.

The utility of the multiresolution image class is demonstrated in a Netscape plug-in. The plug-in demonstrates how progressive transmission can be successfully used to transmit and show just a portion of the entire image. The
plug-in was originally intended to be incorporated into an image browser for very large astronomical images, but could be easily used other contexts. The amount of data produced from current and planned high-resolution sensors requires new transmission techniques to reduce the bandwidth required to browse/use these data sets. Without these new techniques, the data sets will not realize their full potential as research material. Researchers without high-speed Internet connections will not have easy access to these large data sets, reducing the data’s usability to those researchers. To make the information more easily accessible to researchers without high-speed Internet connections, the transmission bandwidth must be reduced as well.
APPENDIX 1

APPLICATION SOURCE CODE

/* $Archive: /Multiresolution Plugin Thesis/MultiresolutionTranslator/MultiresTrans.cpp $ */
/* $Revision: 11 $ $Date: 11/08/00 8:42a $ */

#include "MultiresImage.h"
#include "DIBImage.h"

int main (int argc, char **argv)
{
    char* szProgName = new char[255];
    char* szInfile = new char[255];
    char* szOutfile = new char[255];

    // save program name to use in error messages
    szProgName = argv[0];
    if ((szProgName == NULL) || (szProgName[0] == 0))
        szProgName = "multiresrans";

    // parse the arguments
    if (argc == 3) {
        szInfile = argv[1];
        szOutfile = argv[2];
    } else if (argc == 1) {
        szInfile = szOutfile = "";
    } else {
        cerr << szProgName << ": Takes 0 or 2 args";
        exit (FALSE);
    }

    // read the DIB image and convert to 32 bit
    CDIBImage* pdi = new CDIBImage;
    pdi->ReadFile(szInfile);
    pdi->ConvertData(CDIBImage::JVK_RGB32);
}
// write the Multires Image
CMultiresImage* pMRI = new CMultiresImage(pdi);
pMRI->WriteFile(szOutfile);

// clean up
delete pdi;
delete pMRI;

return TRUE;
}
// npshell.cpp - Plug-in methods called from Netscape.  

#include <windows.h>
#include <npapi.h>
#include "MultiRes.h"
#include "PluginWindow.h"

// NPP_Initialize
//
NPError NPP_Initialize(void)
{
    return NPERR_NO_ERROR;
}

// NPP_Shutdown
//
void NPP_Shutdown(void)
{
}

// NPP_GetJavaClass - This plug-in does not use the Java Runtime Interface (JRI)
//
jref NPP_GetJavaClass (void)
{
    // If no java is used return NULL.
    return NULL;
}

// NPP_New - Create a new plug-in instance.
//
NPError NP_LOADDS NPP_New (NPMIMETYPE pluginType,  
NPP pInstance,  
uint16 mode,  
int16 argc,  
char* argv[],  
char* argn[]),  
NPSAVEDDATA* saved)
{
    if (pInstance == NULL)
        return NPERR_INVALIDINSTANCEERROR;
// Create a new CPluginWindow object
CMultiRes* pCMultiRes = new CMultiRes (pInstance);

// Attach it to the instance structure
pInstance->pdata = pCMultiRes;

// Save our plug-in's mode and parse args
pCMultiRes->mode = mode;
pCMultiRes->ParseHTMLArgs(argc, argn, argv);

return NPERR_NO_ERROR;


// NPP_Destroy - Destroy our plug-in instance.

NPError NP_LOADDS NPP_Destroy (NPP pInstance, NPSavedData** save)
{
    CMultiRes* pCMultiRes = (CMultiRes *) pInstance->pdata;

    if (pCMultiRes)
    {
        if (pCMultiRes->pWindow)
        {
            // Unsubclass the window, clean it up and delete it.
            pCMultiRes->pWindow->UnsubclassWindow();
pCMultiRes->pWindow->CleanupWindow();
        }

        delete pCMultiRes;
pInstance->pdata = NULL;
    }

    return NPERR_NO_ERROR;
}

// NPP_SetWindow - A window was created, resized, or destroyed.

NPError NP_LOADDS NPP_SetWindow (NPP pInstance, NPWindow* window)
{
    if (!window)
        return NPERR_GENERIC_ERROR;

    if (!pInstance)
        return NPERR_INVALID_INSTANCE_ERROR;

    // Get instance data
    CMultiRes* pCMultiRes = (CMultiRes *) pInstance->pdata;
    if (!pCMultiRes)
        return NPERR_GENERIC_ERROR;

    // Spurious entry - just return
    if (!window->window && !pCMultiRes->pWindow)
        return NPERR_NO_ERROR;
}
// Window should have been destroyed, but because of a bug in
// Navigator, we consider this a spurious entry.
if (!window->window && pMultiRes->pWindow)
    return NPERR_NO_ERROR;

// OK, finally the combination that is good
if (!pMultiRes->pWindow && window->window)
{
    pMultiRes->pWindow = new CPluginWindow (pMultiRes);
    pMultiRes->pWindow->hWnd = (HWND)window->window;
    pMultiRes->pWindow->SubclassWindow();

    pMultiRes->pWindow->InitWindow();
}

// get the new window params
pMultiRes->GetWindowParams(window);
// Redraw the window.
InvalidateRect (pMultiRes->pWindow->hWnd, NULL, TRUE);
UpdateWindow (pMultiRes->pWindow->hWnd);

    return NPERR_NO_ERROR;


// NPP_NewStream - A new stream was created.
//
NPError NP_LOADDS NPP_NewStream(NPP pInstance,
    NPMIMEType type,
    NPStream* pStream,
    NPBool seekable,
    uint16* stype)
{
    if (!pInstance)
        return NPERR_INVALID_INSTANCE_ERROR;

    CMultiRes* pMultiRes = (CMultiRes*)pInstance->pdata;

    *stype = NP_ASFILE;

    if (pMultiRes)
        pMultiRes->Open (pStream);

    return NPERR_NO_ERROR;
}

// NPP_WriteReady - Returns amount of data we can handle for the next
NPP_Write
//
int32 NP_LOADDS NPP_WriteReady (NPP pInstance, NPStream *stream)
{
    return 0x0FFFFFFF;
}
// NPP_Write
//
int32 NP_LOADDS NPP_Write (NPP pInstance, NPStream *stream, int32 offset, int32 len, void *buffer)
{
    return len;
}

// NPP_StreamAsFile
//
void NP_LOADDS NPP_StreamAsFile (NPP pInstance, NPStream* stream, const char* fname)
{
    CMultiRes* pCMultiRes = (CMultiRes *)pInstance->pdata;

    if (pCMultiRes && fname)
        pCMultiRes->GotFileName (fname);
}

// NPP_DestroyStream
//
NPError NP_LOADDS NPP_DestroyStream (NPP pInstance, NPStream *stream, NPError reason)
{
    CMultiRes* pCMultiRes = (CMultiRes *)pInstance->pdata;

    if (pCMultiRes)
        pCMultiRes->EndOfStream ();

    return NPERR_NO_ERROR;
}

// NPP_Print - This plug-in does not print.
//
void NP_LOADDS NPP_Print (NPP pInstance, NPPrint* printInfo)
{
}

// NPP_URLNotify - Not Used.
//
void NPP_URLNotify(NPP pInstance, const char* szURL, NPReason nprReason, void* vpData)
{
    CMultiRes* pCMultiRes = (CMultiRes *)pInstance->pdata;

if (nprReason != NPRES_DONE)
    pCMultiRes->URL_Notify (szURL, vpData);
}
/* $Archive: /Multiresolution Plugin Thesis/Plugin/Multires.h $ * $Revision: 17 $ $Date: 11/11/00 1:51p $ */

#ifndef _MULTIRES_H_
define _MULTIRES_H_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include <npapi.h>
#include "MultiresImage.h"

// Forward declare

// CMultiRes class - Class that implements the Multiresolution Image plug-in. These methods are called by the Netscape methods in nshell.cpp.

class CMultiRes
{

public:
    CMultiRes (NPP pInstance);
    -CMultiRes ();


public:
    NPP pInstance;
    USHORT mode;
    NPStream* pStream;
    CPluginWindow* pWindow;

protected:
    char* szCGI;
    char* szImage;
    char* szSource;
    POINT ptWinTopLeft;
    LONG lWinHeight;
    LONG lWinWidth;
    LONG lCurTop;
    LONG lCurLeft;
    LONG lCurBottom;
    LONG lCurRight;
    SHORT sCurRes;
    CMultiresImage* pMRI;

private:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
// Member functions.
public:
    VOID URL_Notify (const char* szURL, VOID* vpData);
    VOID OnLeftClick(POINT pt);
    VOID OnRightClick(POINT pt);
    BOOL PostRequests (RECTREQ* pRR);
    BOOL GotFileName(const char* szFileName);
    BOOL EndOfStream (void);
    BOOL Close (void);
    BOOL Open (NPStream* pStr);
    VOID ParseHTMLArgs(int16 argc, char* argv[] , char* argv[]);
    VOID GetWindowParams (NPWindow* pNPW);

    LONG GetHeight();
    LONG GetWidth();
};

#endif /* _MULTIRES_H_ */
/* $Archive: /Multiresolution Plugin Thesis/Plugin/Multires.cpp $  
 * $Revision: 23 $ $Date: 11/11/00 1:51p $  
 */

#include "MultiRes.h"
#include "DIBImage.h"
#include "PluginWindow.h"
#include <stdlib.h>
#include <ctype.h>
#include <strstrea.h>

// Other functions

// strcmpend - Compares the last n characters of the two strings.
// strcmpend (const char* string1, const char* string2, int n)
//
// int pos1 = strlen (string1) - n;
// if (pos1 < 0) pos1 = 0;
//
// int pos2 = strlen (string2) - n;
// if (pos2 < 0) pos2 = 0;
//
// return (strcmp(string1 + pos1, string2 + pos2));

CMultiRes::CMultiRes (NPP pInstance)
{
    this->pInstance = pInstance;
pWindow = NULL;  // Don't have a window yet.
    lCurTop = lCurBottom = lCurLeft = lCurRight = 0;
sCurRes = 0;
szSource = szCGI = NULL;
pMRI = new CMultiresImage();
}

CMultiRes::~CMultiRes ()
{
}

// Functions called by Netscape methods in npshell.cpp

// Open - Get ready to recieve data, Called from NPP_Newstream.
//
BOOL CMultiRes::Open(NPStream * pStr)
{
pStream = pStr;
return TRUE;
}

// Close - Free all resources, this plugin is history. Called from
// NPP_DESTROY.
//
BOOL CMultiRes::Close()
{
    return TRUE;
}

// EndOfStream - Called from NPP_DestroyStream. The stream is done
// we have all the data.
//
BOOL CMultiRes::EndOfStream()
{
    // ignore this for now
    return TRUE;
}

// GotFileName - Called from NPP_StreamAsFile with a local file name.
// Read the file and create a Device Dependant Bitmap
// from the input file.
//
BOOL CMultiRes::GotFileName(const char* szFileName)
{
    pMRI->ReadFile(szFileName);
    sCurRes = pMRI->SetRes(sCurRes); // set current res
    LONG lOrigW, lOrigH; // find size of current res
    lOrigW = pMRI->GetResWidth();
    lOrigH = pMRI->GetResHeight();

    if (sCurRes == 0) { // nothing smaller than the thumbnail
        lCurLeft = 0;
        lCurRight = lOrigW - 1;
        lCurTop = 0;
        lCurBottom = lOrigH - 1;
    } else {
        if (lOrigW <= lCurLeft) lCurLeft = 0; // if you screw up the size set to
        if (lOrigW <= lCurRight) lCurRight = lOrigW - 1; // something that may work
        if (lOrigH <= lCurTop) lCurTop = 0;
        if (lOrigH <= lCurBottom) lCurBottom = lOrigH - 1;
    }

    RECT rect = {lCurLeft, lCurTop, lCurRight, lCurBottom};

    // convert the appropriate portion to a DIB
    if (pMRI->Validate()) {
CDIBImage* pDIB = pMRI->MakeDIB(rect);
pWindow->DIBtoDDB(pDIB);
delete pDIB;
}
return TRUE;
}

// ParseHTMLTags - Called by NPP_New to parse the arguments passed in the
// EMBED tag.
//
VOID CMultiRes::ParseHTMLArgs(int argc, char * argn [], char * argv []
{
int iArgCntr;

// args only valid in NP_EMBED mode
if (mode != NP_EMBED) return;

char* szStopString;
for (iArgCntr = 0; iArgCntr < argc; iArgCntr++) {
  if (strncmp(argn[iArgCntr], "TOP", 3) == 0) {
    iCurTop = strtol(argv[iArgCntr], &szStopString, 10);
  }
  else if (strncmp(argn[iArgCntr], "BOTTOM", 3) == 0) {
    iCurBottom = strtol(argv[iArgCntr], &szStopString, 10);
  }
  else if (strncmp(argn[iArgCntr], "RIGHT", 3) == 0) {
    iCurRight = strtol(argv[iArgCntr], &szStopString, 10);
  }
  else if (strncmp(argn[iArgCntr], "LEFT", 3) == 0) {
    iCurLeft = strtol(argv[iArgCntr], &szStopString, 10);
  }
  else if (strncmp(argn[iArgCntr], "RES", 3) == 0) {
    sCurRes = (WORD) strtol(argv[iArgCntr], &szStopString, 10);
  }
  else if (strncmp(argn[iArgCntr], "SRC", 3) == 0) {
    szSource = new char[strlen(argv[iArgCntr]+1)];
    strcpy(szSource, argv[iArgCntr]);
  }
  else if (!strncmpi(argn[iArgCntr], "image")) {
    szImage = new char[strlen(argv[iArgCntr]+1)];
    strcpy(szImage, argv[iArgCntr]);
  }
}
if (szSource) {
  char*pdest;
  int len;

  if (pdest = strchr(szSource, '?')) {
    len = pdest - szSource;
    sz CGI = new char[len];
    strncpy(szCGI, szSource, len);
    szCGI[len] = NULL;
  }
// PostRequests - Posts the requests to the server for the data in the
// lstReqRect lists.

BOOL CMultiRes::PostRequests(RECTREQ* pRR)
{
    RECT rect;
    NPError npErr;  // Error from netscape
    BOOL bRet = TRUE;
    char* buf;

    assert (szSource);

    int i;
    for (i=0; pRR[i].sRes != 0; i++) {
        istream ssData, ssOut;

        rect = pRR[i].rect;
        ssData << "IMAGE=" << szImage <<"&";
        ssData << "RES=" << pRR[i].sRes << "&";
        ssData << "TOP=" << rect.top << "&";
        ssData << "LEFT=" << rect.left << "&";
        ssData << "RIGHT=" << rect.right << "&";
        ssData << "BOTTOM=" << rect.bottom << "&";

        ssOut << szCGI << "?";
        ssOut << ssData.str();
        buf = strdup(ssOut.str());
        npErr = NPN_GetURLNotify (pInstance, ssOut.str(), NULL, buf);
    }
    return bRet;
}

VOID CMultiRes::OnLeftClick(POINT pt)
{
    sCurRes++;
    if (sCurRes == pMRI->SetRes(sCurRes)) {
        pt.x *= 2;
        pt.y *= 2;

        LONG lOrigW, lOrigH;
        lOrigW = pMRI->GetResWidth();
        lOrigH = pMRI->GetResHeight();

        lCurLeft = (pt.x - (lWinWidth/2) < 0) ? 0 : pt.x - (lWinWidth/2);
        lCurRight = (pt.x + (lWinWidth/2) > lOrigW) ? lOrigW-1 : pt.x +
                    (lWinWidth/2);
        lCurTop = (pt.y - (lWinHeight/2) < 0) ? 0 : pt.y - (lWinHeight/2);
        lCurBottom = (pt.y + (lWinHeight/2) > lOrigH) ? lOrigH-1 : pt.y +
                    (lWinHeight/2);

        RECTREQ* pRR;
        RECT rect = {lCurLeft, lCurTop, lCurRight, lCurBottom};
    }
if ((pRR = pMRI->Validate(rect)) == NULL) {
    CDIBImage* pDIB = pMRI->MakeDIB(rect);
    pWindow->DIBtoDDB(pDIB);
    delete pDIB;
} else {
    PostRequests(pRR);
}
else {
    sCurRes--;
}

// Function - Description.
//
VOID CMultiRes::OnRightClick(POINT pt)
{
    if (sCurRes > 0) {
        sCurRes--;
        pMRI->SetRes(sCurRes);
        pt.x /= 2;
        pt.y /= 2;
        
        LONG lOrigW, lOrigH;
        lOrigW = pMRI->GetResWidth();
        lOrigH = pMRI->GetResHeight();
        
        ICurLeft = (pt.x - (lWinWidth/2) < 0) ? 0 : pt.x - (lWinWidth/2);
        ICurRight = (pt.x + (lWinWidth/2) > lOrigW) ? lOrigW-1 : pt.x +
                    (lWinWidth/2);
        ICurTop = (pt.y - (lWinHeight/2) < 0) ? 0 : pt.y - (lWinHeight/2);
        ICurBottom = (pt.y + (lWinHeight/2) > lOrigH) ? lOrigH-1 : pt.y +
                     (lWinHeight/2);
        
        RECTREQ* pRR;
        RECT rect = {ICurLeft, ICurTop, ICurRight, ICurBottom};
        if ((pRR = pMRI->Validate(rect)) == NULL) {
            CDIBImage* pDIB = pMRI->MakeDIB(rect);
            pWindow->DIBtoDDB(pDIB);
            delete pDIB;
        } else {
            PostRequests(pRR);
        }
    }
}

VOID CMultiRes::URL_Notify(const char * szURL, VOID * vpData)
{
}

//

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
/ GetWindowParams - Gets the window position, height and width from
// the NPWindow structure.
//
VOID CMultiRes::GetWindowParams(NPWindow* pNPW)
{
    ptWinTopLeft.x = pNPW->x;
    ptWinTopLeft.y = pNPW->y;

    lWinWidth = pNPW->width;
    lWinHeight = pNPW->height;
}

// GetHeight/Width - Returns the window height/width.
//
LONG CMultiRes::GetWidth() {return lWinWidth;};
LONG CMultiRes::GetHeight() {return lWinHeight;};
/$Archive: /Multiresolution Plugin Thesis/Plugin/PluginWindow.h$
$Revision: 6 $ $Date: 11/08/00 7:11p $
*/

#include <windows.h>
#include "DIBImage.h"

class CMultiRes;

LONG WINAPI SubClassFunc(HWND hWnd, WORD Message, WORD wParam, LONG lParam);

class CPluginWindow
{

public:

CPluginWindow(CMultiRes* pMultiRes);
virtual ~CPluginWindow();

void UnsubclassWindow();
void SubclassWindow();
void CleanupWindow();
void InitWindow();
VOID DIBtoDDB (CDIBImage* pDIB);

public:

FARPROC IpfnOldWndProc;
HWND hWnd;
HBITMAP hBitmap;
CMultiRes* pMultiRes;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
protected:
};

#endif //
!defined(AFX_PLUGINWINDOW_H__E93C3F13_B9DC_11D2_A2FC_0080C80D1CF9_INCL
UDED_)
// PluginWindow.cpp: implementation of the CPluginWindow class.  

#include "PluginWindow.h"
#include "Multires.h"

CPluginWindow::CPluginWindow(CMultiRes* pMultiRes) 
{ 
    this->pMultiRes = pMultiRes;
}

CPluginWindow::~CPluginWindow() 
{
}

void CPluginWindow::InitWindow() 
{
}

void CPluginWindow::CleanupWindow() 
{
}

void CPluginWindow::SubclassWindow() 
{
    IpfnOldWndProc = (FARPROC) SetWindowLong (hWnd,
    GWL_WNDPROC,
    (DWORD) SubClassFunc);
    SetProp (hWnd, PROPERTY_NAME, (HANDLE)this);
}

// UnsubclassWindow -
void CPluginWindow::UnsubclassWindow()
{
    RemoveProp(hWnd, PROPERTY_NAME);
    SetWindowLong(hWnd, GWL_WNDPROC, (DWORD)lpfnOldWndProc);
}

// Plugin Message Handler

// SubClassFunc - Message Handler to subclass.

LONG WINAPI SubClassFunc(HWND hWnd,
    WORD Message,
    WORD wParam,
    LONG lParam)
{
    CPluginWindow* pPluginWindow = (CPluginWindow*)GetProp(hWnd,
        PROPERTY_NAME);
    static POINT ptMouseLoc;

    switch (Message) {
    case WM_PAINT:
        {
            static PAINTSTRUCT ps;
            static BITMAP bm;
            POINT ptSize, ptOrg;
            HDC hdc, hdcMem;

            hdc = BeginPaint(hWnd, &ps);
            hdcMem = CreateCompatibleDC(hdc);
            SelectObject(hdcMem, pPluginWindow->hBitmap);
            SetMapMode(hdcMem, GetMapMode(hdc));

            GetObject(pPluginWindow->hBitmap, sizeof(BITMAP), (LPVOID)&bm);

            ptSize.x = bm.bmWidth;
            ptSize.y = bm.bmHeight;
            DPtrToLP(hdc, &ptSize, 1);
            ptOrg.x = 0;
            ptOrg.y = 0;
            DPtrToLP(hdcMem, &ptOrg, 1);

            BitBlt(hdc, 0, 0, ptSize.x, ptSize.y,
                hdcMem, ptOrg.x, ptOrg.y, SRCCOPY);

            // white out any leftovers
            LONG lPIH = (pPluginWindow->pMultiRes)->GetHeight();
            LONG lPIW = (pPluginWindow->pMultiRes)->GetWidth();
            if (ptSize.x < lPIW) {
                PatBlt(hdc, ptSize.x, 0, lPIW - ptSize.x,
                    lPIH, WHITENESS);
            }
            if (ptSize.y < lPIH) {
        
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
PatBlt(hdc, 0, ptSize.y, ptSize.x, 
        lPIH - ptSize.y, WHITENESS);
}

DeleteDC(hdcMem);
EndPaint(hWnd, &ps);
break;
}
case WM_RBUTTONDOWN:
case WM_LBUTTONDOWN:
{
    ptMouseLoc.x = LOWORD(lParam);
    ptMouseLoc.y = HIWORD(lParam);
    break;
}
case WM_LBUTTONUP:
{
    HDC hdc;

    if ( (LOWORD(lParam) == ptMouseLoc.x) &&
         (HIWORD(lParam) == ptMouseLoc.y))
    {
        hdc = GetDC(hWnd);
        DPtoLP(hdc, &ptMouseLoc, 1);
        (pPluginWindow->pMultiRes)->OnLeftClick(ptMouseLoc);
        ReleaseDC(hWnd, hdc);
    }
    break;
}
case WM_RBUTTONUP:
{
    HDC hdc;

    if ( (LOWORD(lParam) == ptMouseLoc.x) &&
         (HIWORD(lParam) == ptMouseLoc.y))
    {
        hdc = GetDC(hWnd);
        DPtoLP(hdc, &ptMouseLoc, 1);
        (pPluginWindow->pMultiRes)->OnRightClick(ptMouseLoc);
        ReleaseDC(hWnd, hdc);
    }
    break;
}
default:
    break;
}

return CallWindowProc(pPluginWindow->lpfnOldWndProc,
        hWnd, 
        Message, 
        wParam, 
        lParam);


// Misc Functions

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
// DisplayDIB - Converts the DIB to a DDB and invalidates the area so windows will display it.

VOID CPluginWindow::DIBtoDDB(CDIBImage * pDIB)
{
    HDC hdc = GetDC(hWnd);

    hBitmap = CreateDIBitmap(hdc,
                         (PBITMAPINFOHEADER) pDIB->GetBMIPtr(),
                         CBM_INIT,
                         pDIB->GetDataPtr(),
                         (PBITMAPINFO)pDIB->GetBMIPtr(),
                         DIB_RGB_COLORS);

    ReleaseDC(hWnd, hdc);
    InvalidateRect(hWnd, NULL, TRUE);
}
/* $Archive: /Multiresolution Plugin Thesis/MultiresCGI/MultiresCGI.cpp $  
  * $Revision: 11 $  $Date: 11/08/00 9:02a $ */
#define _WIN32_WINNT 0x0400 // CGI debug stuff from article Q238788

#include <assert.h>
#include <iostream.h>
#include <windows.h>
#include "Multireslmage.h"

extern "C" char **getcgivars();

void main (int argc, char **argv) {
    // CGI debug stuff from TechNet article Q238788
    #ifdef _DEBUG
        char szMessage [256];
        wsprintf (szMessage, "Please attach a debugger to the process 0x%X (%s) and click OK",
                  GetCurrentProcessId(), argv[0]);
        MessageBox(NULL, szMessage, "CGI Debug Time!",
                   MB_OK|MB_SERVICE_NOTIFICATION);
    
    // strcmpend - Compares the last n characters of the two strings. 
    // Returns
    //
    // strcmpend (const char* string1, const char* string2, int n)
    //
    //
    // strcmpend (const char* string1, const char* string2, int n)
    //
    return (strcmp(string1 + pos1, string2 + pos2));
}
#endif

char* szStop;
char* szImage = NULL;
RECT rect = {0,0,0,0}; // a safe value, "in theory"
SHORT sRes = 0;
char** pszArgString = getgivars();
BYTE bPos = 0;

// check to see if there's only one arg
if (!strcmpi(pszArgString[1], "")) {
    if (!strcmpend(pszArgString[0], ".jvk", 4)) {
        szImage = new char[strlen(pszArgString[0])+1];
        strcpy(szlmage, pszArgString[0]);
    }
} else {
    while (pszArgString[bPos] != NULL) {
        if (!strcmpi(pszArgString[bPos], "top")) {
            rect.top = strtol(pszArgString[bPos+1], &szStop, 10);
        } else if (!strcmpi(pszArgString[bPos], "bottom")) {
            rect.bottom = strtol(pszArgString[bPos+1], &szStop, 10);
        } else if (!strcmpi(pszArgString[bPos], "left") &&
            rect.left = strtol(pszArgString[bPos+1], &szStop, 10);
        } else if (!strcmpi(pszArgString[bPos], "right") &&
            rect.right = strtol(pszArgString[bPos+1], &szStop, 10);
        } else if (!strcmpi(pszArgString[bPos], "res")) {
            // type cast to WORD, shouldn't ever be more
            sRes = (SHORT) strtol(pszArgString[bPos+1], &szStop, 10);
        } else if (!strcmpi(pszArgString[bPos], "image")) {
            szImage = new char[strlen(pszArgString[bPos+1])+1];
            strcpy(szlmage, pszArgString[bPos+1]);
        }
    }
    bPos += 2;
}

cout << binary << "Content-type: image/x-jvk\n\n";
    // open the image
assert (szImage);
CMultiresImage* pMRI = new CMultiresImage(szImage);

    // write the required info to stdout
pMRI->SetRes(sRes);
pMRI->WriteProgStream (cout, rect);
cout << flush;

    // cleanup and quit
delete pMRI;
getcgivars.C-- routine to read CGI input variables into an array of strings.

Written in 1996 by James Marshall, james@jmarshall.com, except that the x2c() and unescape_url() routines were lifted directly from NCSA's sample program util.c, packaged with their HTTPD.

For the latest, see http://www.jmarshall.com/easy/cgi/.

/***********************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/** Convert a two-char hex string into the char it represents **/
char x2c(char *what) {
     register char digit;

digit = (what[0] >= 'A' ? ((what[0] & 0xdf) - 'A')+10 : (what[0] - '0'));
    digit *= 16;
    return(digit);
}

/** Reduce any %xx escape sequences to the characters they represent **/
void unescape_url(char *url) {
   register int i,j;

   for(i=0,j=0; url[j]; ++i,++j) {
      if((url[i] = url[j]) == '%') {
         url[i] = x2c(&url[j+1]) ;
         j+= 2 ;
      }
   }
   url[i] = '\0' ;
}
/** Read the CGI input and place all name/val pairs into list.  
**/  
/** Returns list containing name1, value1, name2, value2, ... , NULL  
**/  
char **getcgivars() {  
    register int i ;  
    char *request_method ;  
    int content_length;  
    char *cgiinput ;  
    char **cgivars ;  
    char **pairlist ;  
    int paircount ;  
    char *nvpair ;  
    char *eqpos ;  
  }  
  
  /** Depending on the request method, read all CGI input into  
  cgiinput **/  
  /** (really should produce HTML error messages, instead of  
  exit()ing) **/  
  request_method= getenv("REQUEST_METHOD") ;  
  if (!strcmp(request_method, "GET") || !strcmp(request_method, "HEAD") )  
      cgiinput= strdup(getenv("QUERY_STRING")) ;  
  else if (!strcmp(request_method, "POST")) {  
      char* wow ;  
      wow = strdup(getenv("CONTENT_TYPE")) ;  
      if (wow == NULL) exit(1);  
      /* strcasecmp() is not supported in Windows-- use strcmpi()  
      instead */  
      if ( strcmpi(wow, "application/x-www-form-urlencoded") ) {  
          printf("getcgivars(): Unsupported Content-Type.\n") ;  
          exit(1) ;  
      }  
      if ( !(content_length = atoi(getenv("CONTENT_LENGTH")))) {  
          printf("getcgivars(): No Content-Length was sent with the  
          POST request.\n") ;  
          exit(1) ;  
      }  
      if ( !(cgiinput= (char *) malloc(content_length+1)) ) {  
          printf("getcgivars(): Could not malloc for cgiinput.\n") ;  
          exit(1) ;  
      }  
      if (!fread(cgiinput, content_length, 1, stdin)) {  
          printf("Couldn't read CGI input from STDIN.\n") ;  
          exit(1) ;  
      }  
      cgiinput[content_length]='\0' ;  
  }
else {
    printf("getcgivars(): unsupported REQUEST_METHOD\n")
    exit(1);
}

/** Change all plusses back to spaces **/
for(i=0; cgiinput[i]; i++) if (cgiinput[i] == ' + ') cgiinput[i] = '

/** First, split on ";" to extract the name-value pairs into
pairlist **/
pairlist= (char **) malloc(256*sizeof(char **))
paircount= 0
nvpair= strtok(cgiinput, ";")
while (nvpair) {
    pairlist[paircount++]= strdup(nvpair);
    if ((! (paircount%256))
        pairlist= (char **) realloc(pairlist,(paircount+256)*sizeof(char **)) ;
    nvpair= strtok(NULL, ";")
}
pairlist[paircount]= 0;  /* terminate the list with NULL */

/** Then, from the list of pairs, extract the names and values **/
cgivars= (char **) malloc((paircount*2+1)*sizeof(char **))
for (i= 0; i<paircount; i++) {
    if (eqpos=strchr(pairlist[i], ' =')) {
        *eqpos= '\0' ;
        unescape_url(cgivars[i*2+1] = strdup(eqpos+1))
    } else {
        unescape_url(cgivars[i*2+1] = strdup(""))
    }
    unescape_url(cgivars[i*2] = strdup(pairlist[i]));
}
.cgivars[paircount*2] = 0;  /* terminate the list with NULL */

/** Free anything that needs to be freed **/
free(cgiinput) ;
for (i=0; pairlist[i]; i++) free(pairlist[i])
free(pairlist) ;

/** Return the list of name-value strings **/
return cgivars ;
}

/******************** end of the getcgivars() module *******************/
APPENDIX 2

IMAGE CLASS SOURCE CODE

/* $Archive: /Multiresolution Plugin Thesis/Multiresolution Translator/Image.h $ */
/* $Revision: 26 $ $Date: 11/06/00 10:02a $ */

// Image.h: interface for the CImage class.

#ifndef AFX_IMAGE_H_5291DB84_CC23_11D2_A302_0080C80D1CF9_INCLUDED_
#define AFX_IMAGE_H_5291DB84_CC23_11D2_A302_0080C80D1CF9_INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include <windows.h>

// Additional Stream I/O functions.

// ReadValue - Template for reading values from a stream.

template <class T>
VOID ReadValue (istream& ist, T& val)
{
    ist.read ((PBYTE)(&val), sizeof(T));
}

// WriteValue - Template for writing values to a stream.

template <class T>
VOID WriteValue (ostream& ost, T& val)
{
    ost.write ((PBYTE)(&val), sizeof(T));
}

// CImage Class - A basic class which stores an image with a minimal

58
// amount of information about the composition of the
// image.

class CImage
{

// Constructor/Destruction
public:
CImage(LONG w = 0, LONG h = 0, WORD dt = JVK_NONE, WORD bits = 0);
CImage(CImage& image);
virtual ~CImage();

// Member functions
public:
RGBQUAD* GetDataPtr();
VOID SetDataPtr(RGBQUAD* prgb);
DWORD GetDataBytes();
VOID SetDataBytes(DWORD dwLen);
LONG GetWidth();
LONG GetHeight();
virtual BOOL ReadData (istream& ist);
virtual BOOL ReadHeader (istream& ist) = 0;
virtual BOOL WriteData (ostream& ost);
virtual BOOL WriteHeader (ostream& ost) = 0;

protected:
virtual BOOL CreateDataStore();

// Data members
public:
enum DataType {JVK_NONE,
JVK_RGB32,
JVK_RGB24,
JVK_MRI30};

protected:
WORD wDataType;
LONG lWidth;
LONG lHeight;
WORD wBitsPerPixel;
DWORD dwDataBytes;
RGBQUAD * pData;

};
// Image.cpp: implementation of the CImage class.
//
#include <windows.h>
#include <iostream.h>
#include <strstrea.h>
#include "Image.h"

CImage::CImage(LONG w, LONG h, WORD dt, WORD bits)
{
  lWidth = w;
  lHeight = h;
  wDataType = dt;
  wBitsPerPixel = bits;
  pData = NULL;
  dwDataBytes = 0;
}

CImage::CImage(CImages image)
{
  lWidth = image.lWidth;
  lHeight = image.lHeight;
  wDataType = image.wDataType;
  wBitsPerPixel = image.wBitsPerPixel;
  dwDataBytes = image.dwDataBytes;

  CreateDataStore();
  ostrstream ost = *( new ostrstream((char *)pData, dwDataBytes));
  ost.write((char *)image.pData, dwDataBytes);
}

CImage::~CImage()
{
  if (pData) free(pData);
}

BOOL CImage::WriteData(ostream& ost)
{
// check to see if there is data to write
if (!pData) return FALSE;

// write the data to the stream
if (!ost.write ((const BYTE *)pData, dwDataBytes)) {
    return FALSE;    // there was a write error
}
return TRUE;


// ReadData - Reads the data from the stream. DataSize must
// be already initialized.
//
// BOOL CImage::ReadData(istream& ist)
//{
//    // check to see if the data structure is initialized
//    // if it’s not, create it.
//    if (!pData) {
//        CreateDataStore();
//    }
//    if (dwDataBytes > 0) {
//        if (!ist.read ((BYTE *)pData, dwDataBytes)) {  
//            return FALSE;    // read error
//        }
//    }
//    return TRUE;
//}

// Data Accessors

// GetHeight - Accessor for IHeight.
//
// LONG CImage::GetHeight()
//{
//    return IHeight;
//}

// GetWidth - Accessor for IWidth.
//
// LONG CImage::GetWidth()
//{
//    return IWidth;
//}

// GetDataBytes - Returns the number of data bytes in pData.
//
// DWORD CImage::GetDataBytes()
//{
//    return dwDataBytes;
//}
VOID CImage::SetDataBytes(DWORD dwLen)
{
    dwDataBytes = dwLen;
}

// GetDataPtr - Returns a pointer to the start of the data.
RGBQUAD* CImage::GetDataPtr()
{
    return pData;
}

// SetDataPtr - Sets the data pointer.
VOID CImage::SetDataPtr(RGBQUAD * prgb)
{
    pData = prgb;
}

// CreateDataStore - Creates the pData array, based on the information
// contained in the class.
BOOL CImage::CreateDataStore()
{
    if (!pData) {
        if (dwDataBytes == 0) {  // figure out how much data there should be
            dwDataBytes = IWidth*IHeight*(wBitsPerPixel/8);
        }

        if (!((RGBQUAD *)malloc(dwDataBytes))) {
            dwDataBytes = 0;
            return FALSE;
        }
    }
    return TRUE;
}
#ifndef AFX_MULTITRESIMAGES_H__5291DB88_CC23_11D2_A302_0080C80D1CF9_INCLUDED_
#define AFX_MULTITRESIMAGES_H__5291DB88_CC23_11D2_A302_0080C80D1CF9_INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include <windows.h>
#include "DIBImage.h"
#include "Image.h"
#include "List.hpp"
#include "MRIRes.h"
#include "PixelMath.h"

// Structure for passing a list of rectangles to calling program
typedef struct tagRECTREQ {
  SHORT sRes;
  RECT rect;
} RECTREQ;

// Multiresolution Image Class
class CMultiresImage : public CImage {
public:
  CMultiresImage(LONG w = 0, LONG h = 0, WORD dt = 0, WORD bits = 0);
  CMultiresImage(char* szFileName);
  CMultiresImage(CImage* image, LONG lXmax=THUMB_XMAX, LONG lXmin=THUMB_YMAX);
  virtual ~CMultiresImage();
  CMultiresImage(CMultiresImage& mri);

  // Member functions.
  //
  // Stream IO
public:
  virtual VOID WriteProgStream (ostream& ost, RECT rect, SHORT sResSent = 0);
protected:
  virtual VOID ReadStream(istream & ist);
  virtual BOOL ReadData(istream & ist);
private:
  virtual BOOL ReadHeader (istream & ist);
  virtual BOOL WriteHeader (ostream & ost);
  virtual VOID WriteProgShortHeader (ostream & ost, RECT rect);
  virtual VOID WriteProgLongHeader (ostream& ost, SHORT sResSent);

  //
  // File IO
  //
public:
  BOOL ReadFile (const char* szFileName);
  BOOL WriteFile(const char* szFileName);
protected:
private:

  //
  // Data Accessors
  //
public:
  LONG GetOrigWidth();
  LONG GetOrigHeight();
  LONG GetResWidth();
  LONG GetResHeight();
  virtual VOID SetData (LONG x, LONG y, RGBQUAD rgb);
  virtual VOID SetData (LONG x, LONG y, TRANSRESID tr, BYTE bDataLoc);
  virtual RGBQUAD GetData (LONG x, LONG y);
  virtual TRANSRESID GetData (LONG x, LONG y, BYTE bDataLoc);
  RGBQUAD Value (LONG x, LONG y, SHORT sRes=-l);
  CDIBImage* MakeDIB(RECT rect);
  WORD SetRes(SHORT sRes = -1);
protected:
  virtual CMRIRes* FindResolution(SHORT sRes);
private:

  //
  // Rectangle Validation
public:
  RECTREQ* Validate (RECT rect);
  bool Validate();
protected:
private:

  //
  // Member data
  //
public:
protected:
  List<CMRIRes*> lstResList;
  LONG lOrigHeight, lOrigWidth;
  SHORT sNumRes, sCurRes;
const static LONG THUMB_XMAX;
const static LONG THUMB_YMAX;
const static WORD wFileHeaderSize;
const static WORD wProgLongHeaderSize;
const static WORD wProgShortHeaderSize;
private:
   RECTREQ* pRR;
};

#ifndef AFX_MULTIRESIMAGE_H__5291DB88_CC23_11D2_A302_0080C80D1CF9__INCLUDED_
// MultiresImage.cpp: implementation of the CMultiresImage class.

#include "MultiresImage.h"
#include <strstream.h>
#include <fstream.h>
#include "HTransform.h"
#include <assert.h>

const LONG CMultiresImage::THUMB_XMAX = 50;
const LONG CMultiresImage::THUMB_YMAX = 50;

const WORD CMultiresImage::wFileHeaderSize =
    sizeof(WORD /*wFileHeaderSize*/) +
    sizeof(SHORT /*sNumRes*/) +
    sizeof(LONG /*lOrigWidth*/) +
    sizeof(LONG /*lOrigHeight*/) +
    sizeof(LONG /*lThumbWidth*/) +
    sizeof(LONG /*lThumbHeight*/) +
    sizeof(WORD /*wDataType*/) +
    sizeof(WORD /*wBitsPerPixel*/);

const WORD CMultiresImage::wProgLongHeaderSize =
    sizeof(WORD /*wProgLongHeaderSize*/) +
    sizeof(SHORT /*sNumRes*/) +
    sizeof(SHORT /*sResSent*/) +
    sizeof(LONG /*lOrigWidth*/) +
    sizeof(LONG /*lOrigHeight*/) +
    sizeof(LONG /*lThumbWidth*/) +
    sizeof(LONG /*lThumbHeight*/) +
    sizeof(WORD /*wDataType*/) +
    sizeof(WORD /*wBitsPerPixel*/);

const WORD CMultiresImage::wProgShortHeaderSize =
    sizeof(WORD /*wProgShortHeaderSize*/) +
    sizeof(SHORT /*sRes*/) +
    sizeof(RECT /*rect*/);
// Compare functions for sorting lists
// Compare (const RECT &rectA, const RECT &rectB)
{
    // compare the first that aren't the same
    if (rectA.left != rectB.left)
        return (rectA.left < rectB.left) ? -1 : 1;
    if (rectA.top != rectB.top)
        return (rectA.top < rectB.top) ? -1 : 1;
    if (rectA.right != rectB.right)
        return (rectA.right < rectB.right) ? -1 : 1;
    if (rectA.bottom != rectB.bottom)
        return (rectA.bottom < rectB.bottom) ? -1 : 1;
    return 0; // it was the same
}

// swap - Template to swap two numbers.
//
template <class T>
VOID swap (T& t1, T& t2) {
    T t3;
    t3 = t1;
    t1 = t2;
    t2 = t3;
}

CMultiresImage::CMultiresImage(LONG w, LONG h, WORD dt, WORD bits)
    :CImage((w+1)/2, (h+1)/2, dt, bits)
{
    lOrigWidth = w;
    lOrigHeight = h;

    sNumRes = 0;
    sCurRes = 0;
    pRR = NULL;
}

CMultiresImage::CMultiresImage (char* szFileName) : CImage()
{
    lOrigHeight = lOrigWidth = 0;

    sNumRes = 0;
    sCurRes = 0;
    pRR = NULL;

    ReadFile(szFileName);
}

// Constructor - Converts an Image to a MultiresImage.
CMultiresImage::CMultiresImage(CImage* image, LONG lXmax, LONG lYmax)
  : CImage(image->GetWidth(), image->GetHeight(), JVK_RGB32, 32)
{
    lOrigWidth = image->GetWidth(); // initial values setup
    lOrigHeight = image->GetHeight();
    sNumRes = sCurRes = 0;
    pRR = NULL;
    CreateDataStore();
    // copy the image data.
    ostrstream ost = *(new ostrstream ((char*)pData, dwDataBytes));
    ost.write((char*)image->GetDataPtr(), image->GetDataBytes());

    CMRIRes* pRes;
    RGBQUAD* pNewData;
    RECT rect;
    Iter<CMRIRes*> lstRes(lstResList);

    while ((lWidth > lXmax) || (lHeight > lYmax)) {
        pRes = new CMRIRes(lWidth, lHeight);
        lstRes.addFirst(pRes);
        sNumRes++;
        // create the data stores for the residuals
        pRes->CreateResidXDataStore();
        pRes->CreateResidYDataStore();
        pRes->CreateResidCDataStore();

        // Calculate the new thumbnail and residuals
        pNewData = Transform(pData, lWidth, lHeight,
            &pRes->vpTransResidX),
            &pRes->vpTransResidY),
            &pRes->vpTransResidC));

        free(pData); // delete the old thumbnail and attach the new
        pData = pNewData;
        lWidth = pRes->lTransWidth;
        lHeight = pRes->lTransHeight;
        dwDataBytes = lWidth * lHeight * sizeof(RGBQUAD);

        Iter<RECT> lstValRect(pRes->lstValRectList);
        rect.top = rect.left = 0;
        rect.right = lWidth-1;
        rect.bottom = lHeight-1;
        lstValRect.addsorted(rect, compare);
    }
}

CMultiresImage::~CMultiresImage()
{
    // delete all the residual data
    int rc;
    Iter<CMRIRes*> lstRes(lstResList);
    rc = lstRes.first();
    while (rc = TR_OK) {
        delete lstRes.atNode(); // confirm this hits the CMRIRes destructor.
    }
}
rc = lstRes.first();
}
}

// Basic Copy constructor. NOT IMPLEMENTED YET
// CMultiresImage::CMultiresImage(CMultiresImage& mri)
{
    assert(FALSE); // duplicate an image
}

/////////////////////////////////////////
// Stream I/O
/////////////////////////////////////////

// ReadHeader - Stub. Not used at this time.
// BOOL CMultiresImage::ReadHeader(istream & ist)
{
    assert(FALSE); // ain't supposed to be here
    return (FALSE); // definately shouldn't have been here
}

VOID CMultiresImage::ReadStream(istream & ist)
{
    WORD wHeaderSize;

    ReadValue (ist, wHeaderSize);

    switch (wHeaderSize) {
    case wFileHeaderSize: // Read the basic file header
        ReadValue (ist, sNumRes);
        ReadValue (ist, lOrigWidth);
        ReadValue (ist, lOrigHeight);
        ReadValue (ist, lWidth);
        ReadValue (ist, lHeight);
        ReadValue (ist, wDataType);
        ReadValue (ist, wBitsPerPixel);

        { 
            Iter<CMRIRes*> IstRes (lstResList); // Init Res list
            LONG lH = lOrigHeight;
            LONG lW = lOrigWidth;

            for (sCurRes = sNumRes; sCurRes > 0; sCurRes--) {
                IstRes.addFirst(new CMRIRes(lH, lW));
                lW = (lW+1)/2;
                lH = (lH+1)/2;
            }
        // reads the data for each resolution
    }
for (sCurRes = 0; sCurRes <= sNumRes; sCurRes++) {
    ReadData(ist);
}
}
break;

case wProgLongHeaderSize:
{
    SHORT sResSent;
    ReadValue (ist, sNumRes);
    ReadValue (ist, sResSent);
    ReadValue (ist, lOrigWidth);
    ReadValue (ist, lOrigHeight);
    ReadValue (ist, lWidth);
    ReadValue (ist, lHeight);
    ReadValue (ist, wDataType);
    ReadValue (ist, wBitsPerPixel);

    Iter<CMRIRes*> lstRes(lstResList);
    LONG lH = lOrigHeight;
    LONG lW = lOrigWidth;

    for (sCurRes = sNumRes; sCurRes > 0; sCurRes--) {
        lstRes.addFirst(new CMRIRes(lH, lW));
        lW = (lW+1)/2;
        lH = (lH+1)/2;
    }

    for (sCurRes=0; sCurRes <= sResSent; sCurRes++) {
        ReadData(ist);
    }
}
break;

case wProgShortHeaderSize:
{
    RECT rectData;
    ReadValue (ist, sCurRes);
    ReadValue (ist, rectData);
    CMRIRes* pRes = FindResolution (sCurRes);
    TRANSRESID tr;
    LONG x,y;

    // read the X-Residual data
    for (y=rectData.top; y<= rectData.bottom; y++) {
        // should change this to read by line.
        for (x=rectData.left; x<= rectData.right; x++) {
            ReadValue (ist, tr);
            pRes->SetData (x,y, tr, MRI_XRESID);
        }
    }

    // read the Y-residual Data
    for (y= rectData.top; y<= rectData.bottom; y++) {
        // should change this to read by line.
        for (x=rectData.left; x<= rectData.right; x++) {
            ReadValue (ist, tr);
            pRes->SetData (x,y, tr, MRI_YRESID);
        }
    }
}
for (x = rectData.left; x <= rectData.right; x++) {
    ReadValue (ist, tr);
    pRes->SetData (x, y, tr, MRI_YRESID);
}

// read the C-residual Data
for (y = rectData.top; y <= rectData.bottom; y++) {
    // should change this to read by line.
    for (x = rectData.left; x <= rectData.right; x++) {
        ReadValue (ist, tr);
        pRes->SetData (x, y, tr, MRI_CRESID);
    }
}

// validate the section just read
Iter<RECT> lstValRect (pRes->lstValRectList);
Iter<RECT> lstReqRect (pRes->lstReqRectList);
trOp rc;
rc = lstReqRect.locate (rectData, compare);
if (rc == CURR) {
    // remove from request list
    delete lstReqRect.atNode();
}
lstValRect.addsorted (rectData, compare); // add to valid list
}
break;

default:
    cerr << "Error...Unknown header type." << endl;
} // case

// ReadData - Reads data from the stream into the data store
// indicated by sCurRes.
//
// BOOL CMultiresImage::ReadData(istream &ist)
{
    switch (sCurRes) {
    case 0:
        // read the thumbnail
        CImage::ReadData (ist);
        break;

    default:
    { // read the residuals
        CMRIRes* pRes = FindResolution (sCurRes);
        if (pRes)
            pRes->ReadResidual (ist);
        else
            return FALSE;
    } // end case default
    } // end switch

    return TRUE;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
// WriteHeader - Writes a Standard Multires Header.
//
BOOL CMultiresImage::WriteHeader(ostream & ost)
{
    WriteValue (ost, wFileHeaderSize);
    WriteValue (ost, sNumRes);
    WriteValue (ost, lOrigWidth);
    WriteValue (ost, lOrigHeight);
    WriteValue (ost, IWidth);
    WriteValue (ost, IHeight);
    WriteValue (ost, wDataType);
    WriteValue (ost, wBitsPerPixel);

    if (ost.fail()) {
        return FALSE;
    } else {
        return TRUE;
    }
}

// WriteProgShortHeader - Writes the short header for progressive transmissions used to pass updates.

VOID CMultiresImage::WriteProgShortHeader(ostream & ost, RECT rect)
{
    WriteValue (ost, wProgShortHeaderSize);
    WriteValue (ost, sCurRes);
    WriteValue (ost, rect);
}

// WriteProgLongHeader - Writes the long header for progressive transmission that passes all the image parameters to set up the structures for future use.

VOID CMultiresImage::WriteProgLongHeader(ostream & ost, SHORT sResSent)
{
    WriteValue (ost, wProgLongHeaderSize);
    WriteValue (ost, sNumRes);
    WriteValue (ost, sResSent);
    WriteValue (ost, lOrigWidth);
    WriteValue (ost, lOrigHeight);
    WriteValue (ost, IWidth);
    WriteValue (ost, IHeight);
    WriteValue (ost, wDataType);
    WriteValue (ost, wBitsPerPixel);
}
// WriteProgStream - Determines which header and what data to write to
// the
// stream based on the resolution and clipping
// rectangle.
// The rectangle is given is transform coordinates
//
VOID CMultiresImage::WriteProgStream(ostream & ost, RECT rect, SHORT
sResSent)
{
    CMRIRes* pRes;

    if (sCurRes == 0) {  // Write the basic header and thumbnail
        WriteProgLongHeader (ost, sResSent);
        WriteData (ost);
    }
    else {  // Write the residuals for the res/rect passed
        WriteProgShortHeader (ost, rect);
        pRes = FindResolution(sCurRes);
        pRes->WriteProgData (ost, rect);
    }
}

// File I/O

// ReadFile - Reads a file into an uninitialized MultiresImage.
//
BOOL CMultiresImage::ReadFile (const char * szFileName)
{
    istream* pist;
    ifstream ifst;
    if (strcmp (szFileName, "", 1)) {
        ifst.open (szFileName, ios::ncreate|ios::binary);
        pist = &ifst;
    } else {
        pist = &cin;
    }
    if (pist->fail()) return FALSE;

    ReadStream(*pist);
    if (pist->fail()) return FALSE;

    return TRUE;
}

// WriteFile - Writes the multires image to a file. It will store the
// file
// header which contains enough information to reconstruct all the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
// resolutions of the image. The data is stored in the following order: The data for the thumbnail, the X-residual values any Y- residual values, then C-residual values for each resolution increasing in size.

BOOL CMultiresImage::WriteFile(const char * szFileName)
{
    // figure out where to output
    ostream* post; // the output stream
    ofstream ofst;
    if (strncmp(szFileName, "", 1)) {
        ofst.open(szFileName, ios::binary);
        post = &ofst;
    } else {
        post = &cout;
    }
    if (post->fail()) return FALSE;
    WriteHeader(*post);
    CImage::WriteData(*post);
    {
        CMRIRes* pRes;
        SHORT sRes;
        for (sRes=1; sRes <= sNumRes; sRes++) {
            pRes = FindResolution(sRes);
            pRes->WriteResidual(*post);
        }
    }
    if (post->fail()) {
        return FALSE;
    } else {
        return TRUE;
    }
}

Data Accessors

GetOrigHeight/GetOrigWidth - Gee, what do you think they do? //
LONG CMultiresImage::GetOrigHeight()
{
    return lOrigHeight;
}

LONG CMultiresImage::GetOrigWidth()
{
    return lOrigWidth;
}
// GetResHeight/GetResWidth - Returns the height and width of the current
// resolution.

LONG CMultiresImage::GetResHeight()
{
    if (sCurRes == 0) return lHeight;

    CMRIRes* pRes = FindResolution(sCurRes);
    return pRes->lOrigHeight;
}

LONG CMultiresImage::GetResWidth()
{
    if (sCurRes == 0) return lWidth;

    CMRIRes* pRes = FindResolution(sCurRes);
    return pRes->lOrigWidth;
}

// FindResolution - Returns the pointer to the resolution specified.
//
CMRIRes* CMultiresImage::FindResolution(SHORT sRes)
{
    assert (sRes >= 0);
    assert (sRes <= sNumRes);

    if (sRes == 0) return NULL;

    int i;
    int rc;
    Iter<CMRIRes*> lstRes(lstResList);
    rc = lstRes.first();
    for (i = 1; i<sRes; i++) rc = lstRes.next();

    if (rc == TR_OK) return lstRes.data();
    else return NULL;
}

// SetData (transform) - Sets the data value in the CImage data store.
//
VOID CMultiresImage::SetData(LONG x, LONG y, RGBQUAD rgb)
{
    RGBQUAD* pRGB = NULL;

    if (!pData) CreateDataStore();
    pRGB = (RGBQUAD*)pData;

    assert (x < lWidth);
    assert (y < lHeight);

    pRGB += (y*lWidth + x);
    *pRGB = rgb;
}
// SetData (residual) - Sets the data in the residual data stores.
// VOID CMultiresImage::SetData(LONG x, LONG y, TRANSRESID tr, BYTE bDataLoc)
// {
// CMRIRes* pRes = FindResolution(sCurRes);
// pRes->SetData(x, y, tr, bDataLoc);
// }

// GetData - Returns data from the pData store.
// RGBQUAD CMultiresImage::GetData(LONG x, LONG y)
// {
// RGBQUAD rgb;
// RGBQUAD* pRGB = (RGBQUAD*) pData;
// pRGB += (y*IWidth + x);
// rgb = *pRGB;
// return rgb;
// }

// GetData (residual) - Gets the data from the residual data stores.
// TRANSRESID CMultiresImage::GetData(LONG x, LONG y, BYTE bDataLoc)
// {
// TRANSRESID tr;
// CMRIRes* pRes = FindResolution(sCurRes);
// tr = pRes->GetData(x, y, bDataLoc);
// return tr;
// }

// Value - Recreates the pixel (x,y) from the transform and residual data.
// RGBQUAD CMultiresImage::Value(LONG x, LONG y, SHORT sRes)
// {
// RGBQUAD rgb;
// RGBQUAD *pRGB;
// if (sRes == -1) sRes = sCurRes; // set to current if no input, uses
// if (sRes == 0)
// rgb = GetData (x,y);
// else {
// CMRIRes* pRes = FindResolution(sRes);
// SHORT oddX = x%2;
// SHORT oddY = y%2;
// LONG posX = x/2;
// }

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
LONG posY = y/2;

RGBQUAD rgbData = Value(posX, posY, sRes-1);
TRANSRESID trX = pRes->GetData (posX, posY, MRI_XRESID);
TRANSRESID trY = pRes->GetData (posX, posY, MRI_YRESID);
TRANSRESID trC = pRes->GetData (posX, posY, MRI_CRESID);

pRGB = ITransform (&rgbData, &trX, &trY, &trC,1,1,1,oddX, oddY);
rgb = *pRGB;
free (pRGB);
return rgb;
}

// MakeDIB - Creates a DIB from the current resolution, cropped based
// on the values passed in rect.
//
CDIBImage* CMultiresImage::MakeDIB (RECT rect)
{
if (rect.right < rect.left) // make A the smallest (x,y)
    swap (rect.right, rect.left);
if (rect.bottom < rect.top)
    swap (rect.bottom, rect.top);

LONG IW = rect.right - rect.left + 1; // add one because start and end
    are
LONG IH = rect.bottom - rect.top + 1; // both to be included

CDIBImage* dib = new CDIBImage (IW, IH, wDataType, wBitsPerPixel);
RGBQUAD rgb;
LONG x,y;
for (y = rect.top; y <= rect.bottom; y++) {
    for (x = rect.left; x <= rect.right; x++) {
        rgb = (Value (x,y));
        dib->SetData (x - rect.left, y - rect.top, rgb);
    }
}
return dib;
}

// SetRes - Sets the current resolution, if called with default
// argument of -1
// it will return the current resolution. If set to -2 will
// set the current res to the max res.
//
WORD CMultiresImage::SetRes (SHORT sRes)
{
    switch (sRes) {
        case -1:
            return sCurRes;
        case -2:
            sCurRes = sNumRes;
            break;
        default:
            break;
    }
    return sCurRes;
}
default:
    if (sRes < -2) sCurRes = 0;
    else if (sRes > sNumRes) sCurRes = sNumRes;
    else sCurRes = sRes;
    break;
} return sCurRes;
}

// Rectangle Validation Functions


// Validate (rect) - Validates a region (rect) given in relation to the original size of the current resolution. Validates each smaller resolution in succession creating a list of rectangles in each resolution to request which will make the entire region valid.

RECTREQ* CMultiresImage::Validate(RECT rect)
{
    BOOL bRtn = TRUE;
    SHORT sRes;
    CMRIRes* pRes;
    // generate the lists for each resolution
    for (sRes = sCurRes; sRes >= 1; sRes--) {
        pRes = FindResolution(sRes);
        rect.left /= 2;
        rect.right /= 2;
        rect.top /= 2;
        rect.bottom /= 2;
        if (pRes->Validate(rect) == FALSE) bRtn = FALSE;
    }

    int count = 0;
    if (bRtn) // there wasn't any
        pRR = NULL;
    else {
        // count how many
        for (sRes = 1; sRes <= sNumRes; sRes++) {
            pRes = FindResolution(sRes);
            count += (pRes->lstReqRectList).items();
        }
        // delete the old array/create new onw
        if (pRR) delete pRR;
        pRR = new RECTREQ[count + 1];
    }

    int i = 0;
    int rc; // populate the array
    for (sRes = 1; sRes <= sNumRes; sRes++) {
        pRes = FindResolution(sRes);
        Iter<RECT> lstReqRect(pRes->lstReqRectList);
        rc = lstReqRect.first();
        while (rc == TR_OK) {
            pRR[i].sRes = sRes;
            pRR[i].rect = lstReqRect.data();
            i++;
        }
rc = lstReqRect.next();
}
}
pRR[i].sRes = 0; // marks the end of the list
return pRR;
}

// Validate() - Returns true if no requests pending in lstReqRect.
//
bool CMultiresImage::Validate()
{
    CMRIRes* pRes;
    SHORT sRes;
    for (sRes=1; sRes <= sNumRes; sRes++) {
        pRes = FindResolution(sRes);
        Iter<RECT> IstReqRect(pRes->lstReqRectList);
        // if there's any elements there are requests pending
        if (IstReqRect.first() == TR_OK) return FALSE;
    }
    return TRUE;
}
/* $Archive: /Multiresolution Plugin Thesis/Multiresolution Translator/MRIRes.h $  
* $Revision: 3 $ $Date: 11/06/00 3:19p $  
*/  
////////////////////////////////////////////////////////////////////////
// MRIRes.h: interface for the CMRIResolution class.  
////////////////////////////////////////////////////////////////////////

#if ! defined (AFX_MRIRESOLUTION_H_E75725C1_A73F_11D4_9779_964735000000 INCLUDED_)  
define  
AFX_MRIRESOLUTION_H_E75725C1_A73F_11D4_9779_964735000000 INCLUDED_  
#endif // _MSC_VER >= 1000  
#pragma once  
#if _MSC_VER >= 1000
#endif // _MSC_VER >= 1000

#include <windows.h>  
#include <iostream.h>  
#include "List.hpp"  
#include "pixelmath.h"  
#include "Image.h" // just for Read/WriteValue functions

////////////////////////////////////////////////////////////////////////
// Forward declarations  
////////////////////////////////////////////////////////////////////////

class CMultiresImage;

enum dataLoc {MRI_XRESID,  
MFI_YRESID,  
MFI_CRESID};

////////////////////////////////////////////////////////////////////////
// Resolution class for CMultiresImage  
////////////////////////////////////////////////////////////////////////

class CMRIRes  
{
friend class CMultiresImage;

////////////////////////////////////////////////////////////////////////
// Construction/Deconstruction  
////////////////////////////////////////////////////////////////////////
  public:
    CMRIRes(LONG w=0, LONG h=0);  
    virtual ~CMRIRes();

protected:
    virtual BOOL ReadResidual (istream & ist);  
    virtual BOOL WriteResidual (ostream & ost);  
    virtual VOID WriteProgData (ostream & ost, RECT rect);

TRANSRESID GetData(LONG x, LONG y, BYTE bDataLoc);  
VOID SetData(LONG x, LONG y, TRANSRESID tr, BYTE bDataLoc);

virtual BOOL CreateResidXDataStore ();  
virtual BOOL CreateResidyDataStore ();
virtual BOOL CreateResidCDataStore();

BOOL Validate(RECT rect);

/////////////////////////////////////////////////////////////////////
// Member data
/////////////////////////////////////////////////////////////////////
public:
protected:
List<RECT> lstValRectList;
List<RECT> lstReqRectList;
LONG lOrigHeight, lOrigWidth;
LONG lTransHeight, lTransWidth;
TRANSRESID* vpTransResidX;
TRANSRESID* vpTransResidY;
TRANSRESID* vpTransResidC;
ULONG lTransResidXBytes;
ULONG lTransResidYBytes;
ULONG lTransResidCBytes;

};

#endif // !defined(AFX_MRIRESOLUTION_H__E75725C1_A73F_11D4_9779_964735000000__INC
LUDED_)
#include "MRIRes.h"
#include <assert.h>

extern compare (const RECT &rectA, const RECT &rectB);

template <class T>
VOID swap (T& t1, T& t2) {
    T t3;
    t3 = t1;
    t1 = t2;
    t2 = t3;
}

CMRIRes::CMRIRes(LONG w, LONG h)
{
    lOrigHeight = h;
    lOrigWidth = w;
    lTransHeight = (h+1)/2;
    lTransWidth = (w+1)/2;
    vpTransResidX = vpTransResidY = vpTransResidC = NULL;
    lTransResidXBytes = lTransResidYBytes =
        lTransResidCBytes = 0;
}

CMRIRes::~CMRIRes()
{
    if (vpTransResidX) free(vpTransResidX);
    if (vpTransResidY) free(vpTransResidY);
    if (vpTransResidC) free(vpTransResidC);
}
// ReadResidualData - Reads and stores the residual data from the stream
//
BOOL CMRIRes::ReadResidual(istream & ist) {
    // X-residual
    // check to see if the data structure is already initialized
    if (!vpTransResidX)
        CreateResidXDataStore();
    // check to see if it's the right size
    if (lTransResidXBytes != ITransHeight * ITransWidth *
        (sizeof(TRANSRESID)) ) {
        free (vpTransResidX);
        CreateResidXDataStore();
    }
    // read the data
    if (vpTransResidX > 0) {
        if (! (ist.read ((BYTE *)vpTransResidX, lTransResidXBytes)) ) {
            return FALSE; // read error
        }
    }

    // Y-Residual
    // check to see if the data structure is already initialized
    if (!vpTransResidY)
        CreateResidYDataStore();
    // check to see if it's the right size
    if (lTransResidYBytes != ITransHeight * ITransWidth *
        (sizeof(TRANSRESID)) ) {
        free (vpTransResidY);
        CreateResidYDataStore();
    }
    // read the data
    if (vpTransResidY > 0) {
        if (! (ist.read ((BYTE *)vpTransResidY, lTransResidYBytes)) ) {
            return FALSE; // read error
        }
    }

    // C-Residual
    // check to see if the data structure is already initialized
    if (!vpTransResidC)
        CreateResidCDataStore();
    // check to see if it's the right size
    if (lTransResidCBytes != ITransHeight * ITransWidth *
        (sizeof(TRANSRESID)) ) {
        free (vpTransResidC);
        CreateResidCDataStore();
    }
    // read the data
    if (vpTransResidC > 0) {
        if (! (ist.read ((BYTE *)vpTransResidC, lTransResidCBytes)) ) {
            return FALSE; // read error
        }
    }

    // Validate the rectangle for the data just read

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Iter<RECT> lstValRect(lstValRectList);
RECT rect = {0,0,0,0};
rect.right = lTransWidth-1;
rect.bottom = lTransHeight-1;
lstValRect.addsorted(rect, compare);

return TRUE;
}

// WriteResidual - Writes the residual data to the stream
//
BOOL CMRIRes::WriteResidual(ostream & ost)
{
    //X-Residual
    // check to see if there is data to write
    if (!vpTransResidX) return FALSE;

    // write the data to the stream
    if (!(ost.write ((const BYTE *) vpTransResidX, lTransResidXBytes)))
    { return FALSE; // there was a write error
    }

    // Y-Residual
    // check to see if there is data to write
    if (!vpTransResidY) return FALSE;

    // write the data to the stream
    if (!(ost.write ((const BYTE *) vpTransResidY, lTransResidYBytes)))
    { return FALSE; // there was a write error
    }

    //C-Residual
    // check to see if there is data to write
    if (!vpTransResidC) return FALSE;

    // write the data to the stream
    if (!(ost.write ((const BYTE *) vpTransResidC, lTransResidCBytes)))
    { return FALSE; // there was a write error
    }

    return TRUE;
}

// WriteProgData - Calculates the portion of the data stream required
// to reconstruct the portion of the image bounded by
// the rectangle passed.
//
VOID CMRIRes::WriteProgData(ostream & ost, RECT rect)
{
    TRANSRESID tr;
    LONG x,y;

    for (y = rect.top; y <= rect.bottom; y++)
    {
for (x = rect.left; x <= rect.right; x++) {
    assert (x < lTransWidth);
    assert (y < lTransHeight);

    tr = GetData(x, y, MRI_XRESID);
    WriteValue(ost, tr);
}

for (y = rect.top; y <= rect.bottom; y++) {
    for (x = rect.left; x <= rect.right; x++) {
        assert (x < lTransWidth);
        assert (y < lTransHeight);

        tr = GetData(x, y, MRI_YRESID);
        WriteValue(ost, tr);
    }
}

for (y = rect.top; y <= rect.bottom; y++) {
    for (x = rect.left; x <= rect.right; x++) {
        assert (x < lTransWidth);
        assert (y < lTransHeight);

        tr = GetData(x, y, MRI_CRESID);
        WriteValue(ost, tr);
    }
}

/***************************************************************************/
// Data Accessors
/***************************************************************************/

// GetData - Returns the residual data from the specified bDataLoc.
// TRANSRESID CMRIRes::GetData(LONG x, LONG y, BYTE bDataLoc)
{
    TRANSRESID* pTRData = NULL;
    TRANSRESID tr;

    switch (bDataLoc)
    {
    case MRI_XRESID:
        pTRData = (TRANSRESID*)vpTransResidX;
        break;
    case MRI_YRESID:
        pTRData = (TRANSRESID*)vpTransResidY;
        break;
    case MRI_CRESID:
        pTRData = (TRANSRESID*)vpTransResidC;
        break;
    }

    assert (pTRData);
assert (x < lTransWidth);
assert (y < lTransHeight);

pTRData += (y*lTransWidth + x);
tr = *pTRData;

return (tr);
}

// SetData (residual) - Sets the data in the residual data stores.
//
VOID CMRIRes::SetData(LONG x, LONG y, TRANSRESID tr, BYTE bDataLoc)
{
TRANSRESID* pTR= NULL;

switch (bDataLoc)
{
case MRI_XRESID:
    if (!vpTransResidX) CreateResidXDataStore();
    pTR = (TRANSRESID*)vpTransResidX;
    break;
case MRI_YRESID:
    if (!vpTransResidY) CreateResidYDataStore();
    pTR = (TRANSRESID*)vpTransResidY;
    break;
case MRI_CRESID:
    if (!vpTransResidC) CreateResidCDataStore();
    pTR = (TRANSRESID*)vpTransResidC;
    break;
}

assert (x < lTransWidth);
assert (y < lTransHeight);

pTR += (y*lTransWidth + x);
*pTR = tr;
}

// Data store initialization

// CreateXDataStore - Sets up the X-residual data store.
// Calculates the height and width of the base image, based
// on the size of the original which will be stored in the
// multiresolution image.
//
BOOL CMRIRes::CreateResidXDataStore()
{
    if (!vpTransResidX)
    { lTransResidXBytes = lTransHeight * lTransWidth * (sizeof(TRANSRESID));

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
if (! (vpTransResidX = (TRANSRESID*) malloc(lTransResidXBytes))) {
    lTransResidXBytes = 0;
    return FALSE;
}
}
return TRUE; // there weren't any problems with allocation

/
// CreateYDataStore - Sets up the Y-residual data store.
// Calculates the height and width of the base image,
// based
// on the size of the original which will be stored in
// the
// multiresolution image.
//
BOOL CMRIRes::CreateResidYDataStore()
{
    if (!vpTransResidY) {
        lTransResidYBytes = lTransHeight * lTransWidth *
            (sizeof(TRANSRESID));
        if (! (vpTransResidY = (TRANSRESID*) malloc(lTransResidYBytes))) {
            lTransResidYBytes = 0;
            return FALSE;
        }
    }
    return TRUE; // there weren't any problems with allocation
}

/
// CreateCDataStore - Sets up the C-residual data store.
// Calculates the height and width of the base image,
// based
// on the size of the original which will be stored in
// the
// multiresolution image.
//
BOOL CMRIRes::CreateResidCDataStore()
{
    if (!VpTransResidC) {
        lTransResidCBytes = lTransHeight * lTransWidth *
            (sizeof(TRANSRESID));
        if (! (vpTransResidC = (TRANSRESID*) malloc(lTransResidCBytes))) {
            lTransResidCBytes = 0;
            return FALSE;
        }
    }
    return TRUE; // there weren't any problems with allocation
}

///////////////////////////////////////////////////////////////////
// Rectangle Validation
///////////////////////////////////////////////////////////////////

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
// Validate (RECT) - Checks the rectangle passed and builds the list of
// rectangles which must be validated before the
// larger
// rectangle will be valid for this resolution.

BOOL CMRIRes::Validate(RECT rect)
{
    assert (rect.left >= 0);
    assert (rect.right < ITransWidth);
    assert (rect.top >= 0);
    assert (rect.bottom < ITransHeight);

    // normalize the rectangle so top, bottom, right and left make sense
    if (rect.left > rect.right) swap (rect.left, rect.right);
    if (rect.top > rect.bottom) swap (rect.top, rect.bottom);
    assert (rect.left < rect.right);
    assert (rect.top < rect.bottom);

    int rcR, rcV;  // result codes
    List<RECT> lstRectRList;
    Iter<RECT> lstRectR (lstRectRList);
    Iter<RECT> lstValRect (lstValRectList);
    Iter<RECT> lstReqRect (lstReqRectList);
    BOOL ok;
    BOOL bRtn = TRUE;
    RECT rectV, rectR, rectRl, rectR2;

    lstRectR.addFirst (rect);  // start the working rectangle request list
    rcR = lstRectR.first();

    while (rcR == TR_OK) {  // as long as there's a rectangle on the
        rectR = lstRectR.data();
        ok = TRUE;

        // Work through all valid rectangles
        for (rcV = lstValRect.first(); rcV == TR_OK; rcV = lstValRect.next())
        {
            rectV = lstValRect.data();

            // left edge of valid rect falls in requested rect
            if (  ((rectV.left > rectR.left) && (rectV.left < rectR.right)) )
            {
                rectRl = rectR2 = rectR;
                rectRl.right = rectV.left - 1;  // split on left edge
                rectR2.left = rectV.left;

                delete lstRectR.atNode();  // remove current request from list
                lstRectR.addFirst (rectR2);  // add the two split out to the list
                lstRectR.addFirst (rectRl);

                ok = FALSE;
                break;
            }

            // right edge of valid rect falls in requested rect
            else if (  ((rectV.right < rectR.right) && (rectV.right > rectR.left)) )
            {
rectR1 = rectR2 = rectR;
rectR1.right = rectV.right;  // split on right edge
rectR2.left = rectV.right + 1;

delete lstRectR.atNode();  // remove current request
lstRectR.addFirst (rectR1);  // add the two split rects to list
lstRectR.addFirst (rectR2);

ok = FALSE;
break;
}
// top edge of valid rect falls in requested rect
else if ((rectV.top > rectR.top) && ( rectV.top < rectR.bottom))  {
rectR1 = rectR2 = rectR;
rectR1.bottom = rectV.top - 1;  // split on top edge
rectR2.top = rectV.top;

delete lstRectR.atNode();  // remove the current request
lstRectR.addFirst (rectR2);  // add requests for the split rects
lstRectR.addFirst (rectR1);

ok = FALSE;
break;
}
// bottom edge of valid rect falls in requested rect
else if ((rectV.bottom < rectR.bottom) && (rectV.bottom > rectR.top))  {
rectR1 = rectR2 = rectR;
rectR1.bottom = rectV.bottom;  // split request on bottom edge
rectR2.top = rectV.bottom + 1;

delete lstRectR.atNode();  // remove current request
lstRectR.addFirst (rectR1);  // add requests for split rects
lstRectR.addFirst (rectR2);

ok = FALSE;
break;
}
// valid rect completely contains the requested rect
else if ((rectV.left <= rectR.left) && (rectV.right >= rectR.right)
&& (rectV.top <= rectR.top) && (rectV.bottom >= rectR.bottom))  {
delete lstRectR.atNode();  // remove current request
ok = FALSE;
break;
}
// if a rectangle passes all the wickets here, add it to the real
// list to request, because it was compared to all other valid
// rectangles  
// so it's definitely not valid.
if (ok) {
 lstReqRect.addsorted(rectR, compare);

 delete lstRectR.atNode();
 bRtn = FALSE;
rcR = lstRectR.first(); // work the working request list from the top down
return bRtn;
/* $Archive: /Multiresolution Plugin Thesis/Multiresolution Translator/DIBImage.h$
 * $Revision: 19 $ $Date: 3/18/00 6:47p$
 */

#ifndef _DIBCLASS_H_
define _DIBCLASS_H_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

// load headers
#include <windows.h>
#include <fstream.h>
#include "Image.h"

// CDIBImage class - The class that implements the Device Independent Bitmap (DIB) functionality.

class CDIBImage : public CIImage {

// Construction/Destruction

public:
  CDIBImage (LONG w = 0, LONG h = 0, WORD dt = JVK_NONE, WORD bits = 0);
  virtual ~CDIBImage ();

// Member Functions

public:
  VOID SetData (LONG x, LONG y, RGBQUAD rgb);
  RGBQUAD GetData(LONG X, LONG y);
  VOID ConvertData (WORD wNewDT);
  VOID* GetBMIPtr (  );
  BOOL WritePile (const char* szFileName);
  BOOL ReadFile (const char* szFileName);

protected:
  WORD PaletteSize ();
  WORD NumColors ();
  BOOL WriteHeader(ostream & ost);
  BOOL ReadHeader(istream & ist);
  inline BOOL IS_WIN30_DIB(VOID* pbmi) {
    return (*(LPDWORD)pbmi == BMI_STRUCT_SIZE);
  }

  inline long WIDTHBYTES(long bits) {
    return ((bits + 31) / 32 * 4);
  }

};

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
protected:
    DWORD dwColorsImportant;
    DWORD dwColorsUsed;
    LONG lYPelsPerMeter;
    LONG lXpelsPerMeter;
    DWORD dwCompression;
    WORD wPlanes;
    DWORD dwStructSize;
    VOID* vpBMI;
    const static WORD DIB_HEADER_MARKER;
    const static DWORD BMI_STRUCT_SIZE;
    const static DWORD BMI_CORE_STRUCT_SIZE;
};

#endif /* _DIBCLASS_H */
/* $Archive: /Multiresolution Plugin Thesis/TestMRl/DIBImage.cpp $ 
* $Revision: 26 $ $Date: 11/07/00 9:18a $ */ 

#include <strstrea.h> 
#include <assert.h> 
#include "DIBImage.h" 

const WORD CDIBImage::DIB_HEADER_MARKER = 'M'<<8 | 'B'; 
const DWORD CDIBImage::BMI_STRUCT_SIZE = sizeof(DWORD)+ sizeof(LONG)+ sizeof(LONG)+ sizeof(WORD)+ sizeof(WORD)+ sizeof(DWORD)+ sizeof(DWORD)+ sizeof(DWORD)+ sizeof(DWORD)+ sizeof(DWORD)+ sizeof(DWORD)+ sizeof(DWORD)+ sizeof(DWORD)+ sizeof(DWORD)+ sizeof(DWORD); 
const DWORD CDIBImage::BMI_CORE_STRUCT_SIZE = sizeof(DWORD)+ sizeof(LONG)+ sizeof(LONG)+ sizeof(WORD)+ sizeof(WORD); 

CDIBImage::CDIBImage (LONG w, LONG h, WORD dt, WORD bits) : CImage(w, h, dt, bits) 
{ 
  dwColorsImportant = 0; 
  dwColorsUsed = 0; 
  dwCompression = 0; // BI_RGB, basic no compression DIB. 
  // Need to fix later for more robust class. 
  dwStructSize = BMI_STRUCT_SIZE; 
  lXPelsPerMeter = 0; 
  lYPelsPerMeter = 0; 
  vpBMI = NULL; 
  wPlanes = 1; 
} 

CDIBImage::~CDIBImage () 
{ 
  if (vpBMI) free (vpBMI); 
} 

// File I/O 

// LoadFile - Loads the image from a given file name 
// 
BOOL CDIBImage::ReadFile (const char* szFileName) 
{ 
  WORD wHeaderMarker;

DWORD dwFileSize;
WORD wReserved1;
WORD wReserved2;
DWORD dwOffsetToBits;

// figure out where to output
istream* pist; // pointer to input stream
ifstream ifst; // input fstream
if (szFileName != "") {
    ifst.open(szFileName, ios::nocreate|ios::binary);
pist = &ifst;
} else {
pist = &cin;
}
if (pist->fail()) return FALSE;

// read the file header
ReadValue(*pist, wHeaderMarker);
if (wHeaderMarker != DIB_HEADER_MARKER) { return FALSE;
}
ReadValue(*pist, dwFileSize);
ReadValue(*pist, wReserved1);
ReadValue(*pist, wReserved2);
ReadValue(*pist, dwOffsetToBits);

// read the BMI header
ReadHeader(*pist);

// if not 32-bit color, need to add color table here

// read the image data
ReadData(*pist);

if (pist->fail()) { // check to see if something went wrong
    return FALSE; // while reading the data stream
} else {
    return TRUE;
}

// SaveFile - Saves the DIB to the given file name.
//
BOOL CDIBImage::WriteFile(const char * szFileName)
{
    WORD wHeaderMarker = DIB_HEADER_MARKER;
    DWORD dwFileSize;
    WORD wReserved1 = 0;
    WORD wReserved2 = 0;
    DWORD dwOffsetToBits;

    // calculate the header fields
dwDataBytes = WIDTHBYTES(lWidth * lHeight * wBitsPerPixel);
dwFileSize = dwStructSize + PaletteSize() +
dwDataBytes + sizeof(BITMAPFILEHEADER);
dwOffsetToBits = sizeof(BITMAPFILEHEADER) + dwStructSize + PaletteSize();

    // figure out where to output
    ostream* post; // the output stream
    ofstream ofst;
    if (szFileName != "") {
        ofst.open(szFileName, ios::binary);
        post = &ofst;
    } else {
        post = &cout;
    }
    if (post->fail()) return FALSE;

    // write the file header, bitmap header and data
    WriteValue(*post, wHeaderMarker);
    WriteValue(*post, dwFileSize);
    WriteValue(*post, wReserved1);
    WriteValue(*post, wReserved2);
    WriteValue(*post, dwOffsetToBits);

    WriteHeader(*post);
    // if not working with 32-bit color, need to add color table
    WriteData(*post);

    if (post->fail()) { // see if anything went wrong with the
        return FALSE; // output stream
    } else {
        return TRUE;
    }
}

/
// Stream I/O
/
//////

// ReadHeader - Reads the DIB header from the istream and saves it
// in the class values, then stores it in a bmiHeader
// structure to be used in OS calls.
//
// BOOL CDIBImage::ReadHeader(istream & ist)
// {
//     ReadValue(ist, dwStructSize);
//     assert (dwStructSize == BMI_STRUCT_SIZE); // core headers not
// supported
//     ReadValue(ist, lWidth);
//     ReadValue(ist, lHeight);
//     ReadValue(ist, wPlanes);
//     assert (wPlanes == 1); // must be 1, don't know why
//     ReadValue(ist, wBitsPerPixel);
//     assert ((wBitsPerPixel == 24) || (wBitsPerPixel == 32)); // only
// 24/32 bit
//     switch (wBitsPerPixel) {

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
case 32:
    wDataType = JVK_RGB32;
    break;
case 24:
    wDataType = JVK_RGB24;
    break;
default:
    wDataType = JVK_NONE;
}

ReadValue(ist, dwCompression);
    assert (dwCompression == 0); // no compression supported
ReadValue(ist, dwDataBytes);
ReadValue(ist, lXPelsPerMeter);
ReadValue(ist, lYPelsPerMeter);
ReadValue(ist, dwColorsUsed);
    assert (dwColorsUsed == 0); // color table not supported
ReadValue(ist, dwColorsImportant);

if (ist.fail()) { // check to see if there's a stream error
    return FALSE;
} else {          
    return TRUE;
}

// WriteHeader - Writes the BITMAPINFO structure to the ostream.
//
BOOL CDIBImage::WriteHeader(ostream & ost)
{
    WriteValue(ost, dwStructSize);
    WriteValue(ost, lWidth);
    WriteValue(ost, lHeight);
    WriteValue(ost, wPlanes);
    WriteValue(ost, wBitsPerPixel);
    WriteValue(ost, dwCompression);
    WriteValue(ost, dwDataBytes);
    WriteValue(ost, lXPelsPerMeter);
    WriteValue(ost, lYPelsPerMeter);
    WriteValue(ost, dwColorsUsed);
    WriteValue(ost, dwColorsImportant);

    if (ost.fail()) { // check to see if there's a stream error
        return FALSE;
    } else {
        return TRUE;
    }
}

// Misc Functions

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
// ConvertData - Converts the data from one format of a DIB to another.
//
VOID CDIBImage::ConvertData(WORD wNewDT)
{
    switch (wNewDT) {
    case JVK_RGB32:
        switch (wDataType) {
        case JVK_RGB32:
            break;
        case JVK_RGB24:
        {
            RGBQUAD* newData = new RGBQUAD[lHeight*lWidth];
            LONG x, y;
            for (y=0; y < lHeight; y++) {
                for (x=0; x < lWidth; x++) {
                    newData[y*lWidth + x] = GetData(x, y);
                }
            }
            free (pData);
            pData = newData;
            dwDataBytes = lHeight * lWidth * sizeof(RGBQUAD);
            wDataType = JVK_RGB32;
            wBitsPerPixel = 32;
        } break;
        default: break;
        }
    default: break;
    }
}

//
// PaletteSize - Gets the size required to store the DIB's palette by
// multiplying the number of colors by the size of an
// RGBQUAD (3.0 style DIB) or by th size of RGBTRIPLE
// (for an other-style DIB)
// (NOTE: Only works with 3.0 style DIBs now.)
WORD CDIBImage::PaletteSize ()
{
    return (WORD)(NumColors() * sizeof(RGBQUAD));
}

//
// NumColors - Calculates the number of colors in the DIB's color table
// by finding the bits per pixel.
//
WORD CDIBImage::NumColors()
{
    // If this is a Windows-style DIB, the number of colors in the
// color table can be less than the number of bits per pixel
// allows for (i.e. lpbi->biClrUsed can be set to some value).
// If this is the case, return the appropriate value.
if (dwColorsUsed != 0)
{
    return (WORD)dwColorsUsed;
}

// Calculate the number of colors in the color table based on the
// number of bits per pixel for the DIB.
switch (wBitsPerPixel)
{
    case 1:
        return 2;
    case 4:
        return 16;
    case 8:
        return 256;
    default:
        return 0;
}

// Accessor Functions

VOID* CDIBImage::GetBMIPtr()
{
    if (!vpBMI) {
        if (!((vpBMI = malloc(dwStructSize))) )
            return (vpBMI); // don't try to write the header if you
        // get a bum pointer back
    }

    // always rewrite the header in case the data has changed
    stringstream* sstr = new stringstream((char*) vpBMI, dwStructSize,
                ios::out);
    WriteHeader (*sstr);
    delete sstr;

    return (vpBMI);
}

RGBQUAD CDIBImage::GetData(LONG x, LONG y)
{
    RGBQUAD rgb;
    BYTE* pubData;
    LONG lOffset;

    // GetData - Returns data from the specified location.
    // RGBQUAD CDIBImage::GetData(LONG x, LONG y)
    //
pubData = (BYTE*)pData;

    // some error checking, in case you're being stupid
    assert (x<lWidth);
    assert (y<lHeight);

    switch (wBitsPerPixel)
    {
    case 32:
    case 24:
        lOffset = (y * WIDTHBYTES(lWidth*wBitsPerPixel)) + (x * wBitsPerPixel/8);
        rgb.rgbBlue = *(pubData + lOffset);
        rgb.rgbGreen = *(pubData + lOffset + 1);
        rgb.rgbRed = *(pubData + lOffset + 2);
        rgb.rgbReserved = 0;
        break;
    case 16:
    case 8:
    case 4:
    case 1:
        assert (FALSE); // only 32/24 bit supported ATT
        break;
    }

    return (rgb);
}

// SetData - Writes the specified pixel data into the data store.
///*****/////***** don't think this will work for 24 bit DIBS
*****/////*****\\

// void CDIBImage::SetData(long x, long y, RGBQUAD rgb)
{
    RGBQUAD* rgbpData = NULL;
    LONG lLineWidth;

    if (!pData)
        CreateDataStore();

    rgbpData = (RGBQUAD*)pData;
    lLineWidth = lWidth;

    assert (x < lLineWidth);
    assert (y < lHeight);

    rgbpData += (y*lLineWidth + x);
    *rgbpData = rgb;
}
APPENDIX 3

MISCELLANEOUS SOURCE CODE

/* $Archive: /Multiresolution Plugin Thesis/TransformTest/HTransform.h */
/* $Revision: 7 $ $Date: 11/06/00 8:07a $ */

//
/////////////////////////////////////////////////////////////////////
#if ! defined (AFX HTRANSFORM H 5291DB88_CC23_11D2_A302_0080C80D1CF9 INCLUDED_)
#define AFX HTRANSFORM H 5291DB88_CC23_11D2_A302_0080C80D1CF9 INCLUDED_
#endif // _MSC_VER >= 1000

#define _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////
// Includes
/////////////////////////////////////////////////////////////////////
#include <windows.h>
#include "PixelMath.h"

/////////////////////////////////////////////////////////////////////
// 2-D Transform functions
/////////////////////////////////////////////////////////////////////
RGBQUAD* Transform (RGBQUAD* prgb, // input data
                   LONG lWidth, // input width
                   LONG lHeight, // input height
                   TRANSRESID** ptrX = NULL, // pointers to store the
                   TRANSRESID** ptrY = NULL, // residual data if you
                   TRANSRESID** ptrC = NULL, // want to keep it
                   LONG lOffset = 0, // row offset for only
                   LONG lStartX = 0, // working part of an input area.
                   LONG lStartY = 0);

RGBQUAD* ITransform (RGBQUAD* prgb,
                     TRANSRESID* ptrX,
                     TRANSRESID* ptrY,
                     TRANSRESID* ptrC,
                     LONG lWidth,
LONG LHeight,
LONG lOffset = 0,
LONG lStartX = 0,
LONG lStartY = 0);

#endif //
!defined(AFX_HTRANSFORM_H__5291DB88_CC23_11D2_A302_0080C80D1CF9__INCLUDED_)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
// HTransform.cpp: Computes 1-D and 2-D H-Transforms.

#include "HTransform.h"

// 2-D Transform functions

RGBQUAD* Transform (RGBQUAD* prgbIn, LONG lWidth, LONG lHeight,
TRANRESID** ptrX, TRANSRESID** ptrY,
TRANRESID** ptrC, LONG lOffset,
LONG lStartX, LONG lStartY)
{
    BOOL bResidual = FALSE;
    LONG lTransWidth = (lWidth+1) / 2;
    LONG lTransHeight = (lHeight+1) / 2;
    RGBQUAD* prgbOut = new RGBQUAD[lTransWidth*lTransHeight];

    if (*ptrX & *ptrY & *ptrC) {
        bResidual = TRUE;
        *ptrX = new TRANSRESID[lTransWidth*lTransHeight];
        *ptrY = new TRANSRESID[lTransWidth*lTransHeight];
        *ptrC = new TRANSRESID[lTransWidth*lTransHeight];
    }

    LONG IW, IH, index;
    RGBQUAD rgbA, rgbB, rgbC, rgbD;
    TRANSDATA td1, td2, ho;
    TRANSRESID tr1, tr2, hx, hy, hc;

    for (IH = 0; IH < lTransHeight; IH++) {
        for (IW = 0; IW < lTransWidth; IW++) {
            index = ((lStartY + 1H*2) * (lWidth + lOffset)) + lStartX + 1W*2;
            if (((1W*2+1) < lWidth) && ((1H*2+1) < lHeight)) {
                rgbA = prgbIn[index];
                rgbB = prgbIn[index + 1];
                rgbC = prgbIn[index + (lWidth+lOffset)];
                rgbD = prgbIn[index + (lWidth+lOffset) + 1];
            } else if (((1W*2+1) < lWidth) {
                rgbA = rgbC = prgbIn[index];
                rgbB = rgbD = prgbIn[index + 1];
            } else if (((1H*2+1) < lHeight) {
                rgbA = rgbB = prgbIn[index];
                rgbC = rgbD = prgbIn[index + (lWidth+lOffset)];
            } else {
rgbA = rgbB = rgbC = rgbD = prgbIn[index];
}
d1 = rgbD + rgbC;
d2 = rgbB + rgbA;
ho = d1 + d2;
prgbOut[lH*lTransWidth + lW] = TDtoRGB(mroundTD(ho, 2));
if (bResidual) {
  tr1 = rgbD - rgbC;
  tr2 = rgbB - rgbA;
  hx = d1 - d2;
  hy = tr1 + tr2;
  hc = tr1 - tr2;
  (*ptrX)[lH*lTransWidth + lW] = mroundTR(hx, 1);
  (*ptrY)[lH*lTransWidth + lW] = mroundTR(hy, 1);
  (*ptrC)[lH*lTransWidth + lW] = hc;
}
}
return (prgbOut);
}
RGBQUAD* ITransform (RGBQUAD* prgbln, TRANSRESID* ptrX, TRANSRESID* ptrY,
  TRANSRESID* ptrC, LONG lWidth, LONG lHeight, LONG lOffset,
  LONG lStartX, LONG lStartY)
{
  LONG lW, lH, lWPos, lHPos, lPos;
  RGBQUAD* prgbOut = new RGBQUAD[lWidth*lHeight];
  TRANSDATA tdA, bit0, bit1;
  TRANSRESID trX, trY, trC;
  lTransWidth = (lWidth + lOffset +1)/2;
  for (lH = lStartY; lH < (lStartY + lHeight); lH++) {
    lHPos = (lH)/2;
    for (lW = lStartX; lW < (lStartX + lWidth); lW++) {
      lWPos = (lW)/2;
      lPos = lHPos * lTransWidth + lWPos;
      // read the data and shift the zeros back in
      tdA = prgbln[lPos] <<2;
      trX = ptrX[lPos] <<1;
      trY = ptrY[lPos] <<1;
      trC = ptrC[lPos];
      // propagate bit0 from the C-residual
      bit0 = trC & 0x01;
      trX = bitprop(trX, bit0);
      trY = bitprop(trY, bit0);
      // propagate bit1 and bit0 to ho

bit1 = (trC^trX^trY) & 0x02;
tdA = bitprop(tdA, bit0, bit1);

prgbOut[((IY-IStartY) * IWidth) + (IW-IStartX)] =
(tdA + ((IY%2 ? 1:-1) * trX) +
  ((IW%2 ? 1:-1) * trY) +
  ((IY%2 ? 1:-1) * (IW%2 ? 1:-1) * trC) ) / 4;
}
}

return prgbOut;
}
// PixelMath.h: defines functions and structures for calculations on pixels.

////////////////////////////////////////////////////////////////////
#ifndef  PIXELMATH_H
#define  PIXELMATH_H
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

////////////////////////////////////////////////////////////////////
// Includes
////////////////////////////////////////////////////////////////////
#include <windows.h>

////////////////////////////////////////////////////////////////////
// Data type used for saving residual data.
////////////////////////////////////////////////////////////////////
typedef struct tagTRANSDATA {
    unsigned blue : 10;
    unsigned green: 10;
    unsigned red : 10;
} TRANSDATA;

typedef struct tagTRANSRESID {
    signed blue :10;
    signed green:10;
    signed red :10;
} TRANSRESID;

////////////////////////////////////////////////////////////////////
// RGBQUAD Operators
////////////////////////////////////////////////////////////////////
RGBQUAD operator+ (RGBQUAD rgbA, int i);
RGBQUAD operator+ (int i, RGBQUAD rgbA);
RGBQUAD operator* (int i, RGBQUAD rgbA);
RGBQUAD operator* (RGBQUAD rgbA, int i);
RGBQUAD operator/ (TRANSDATA tdA, int i);

TRANSDATA operator+ (TRANSDATA tdA, TRANSDATA tdB);
TRANSDATA operator+ (TRANSDATA tdA, TRANSRESID trA);
TRANSDATA operator+ (TRANSRESID trA, TRANSDATA tdA);
TRANSDATA operator+ (RGBQUAD rgbA, RGBQUAD rgbB);
TRANSDATA operator+ (TRANSDATA tdA, int i);
TRANSDATA operator+ (int i, TRANSDATA tdA);
TRANSDATA operator<< (RGBQUAD rgbA, int i);

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
TRANSDATA operator& (TRANSDATA tdA, int i);
TRANSDATA operator& (TRANSRESID trA, int i);

TRANSRESID operator+ (TRANSRESID trA, TRANSRESID trB);
TRANSRESID operator- (RGBQUAD rgba, RGBQUAD rgbB);
TRANSRESID operator- (TRANSDATA tdA, TRANSDATA tdB);
TRANSRESID operator- (TRANSRESID trA, TRANSRESID trB);
TRANSRESID operator* (int i, TRANSRESID trA);
TRANSRESID operator* (TRANSRESID trA, int i);
TRANSRESID operator<< (TRANSRESID trA, int i);
TRANSRESID operator^ (TRANSRESID trA, TRANSRESID trB);

bool operator!= (TRANSRESID trA, TRANSRESID trB);
bool operator!= (RGBQUAD rgba, RGBQUAD rgbB);

///////////////////////////////////////////////////////////////////////////////
// Misc functions declarations
///////////////////////////////////////////////////////////////////////////////
BYTE limit(int i);

TRANSDATA mroundTD (TRANSDATA tdA, unsigned int i);
TRANSRESID mroundTR (TRANSRESID trA, unsigned int i);

RGBQUAD TDtoRGB (TRANSDATA tdA);

TRANSRESID bitprop (TRANSRESID trA, TRANSDATA bit0);
TRANSDATA bitprop (TRANSDATA tdA, TRANSDATA bit0, TRANSDATA bit1);

#endif // PIXELMATH_H__
# include "PixelMath.h"
# include <assert.h>

/* RGBQUAD operators */
RGBQUAD operator+ (RGBQUAD rgbA, int i) {
    RGBQUAD rgb;
    rgb.rgbRed = rgbA.rgbRed + i;
    rgb.rgbBlue = rgbA.rgbBlue + i;
    rgb.rgbGreen = rgbA.rgbGreen + i;
    rgb.rgbReserved = rgbA.rgbReserved + i;
    return rgb;
}
RGBQUAD operator+ (int i, RGBQUAD rgbA) {return (rgbA + i);}
RGBQUAD operator* (int i, RGBQUAD rgbA) {
    RGBQUAD rgb;
    rgb.rgbRed = i * rgbA.rgbRed;
    rgb.rgbBlue = i * rgbA.rgbBlue;
    rgb.rgbGreen = i * rgbA.rgbGreen;
    rgb.rgbReserved = i * rgbA.rgbReserved;
    return rgb;
}
RGBQUAD operator* (RGBQUAD rgbA, int i) {return (i * rgbA);}
RGBQUAD operator/ (TRANSDATA tdA, int i) {
    RGBQUAD rgb;
    rgb.rgbBlue = tdA.blue / i;
    rgb.rgbGreen = tdA.green / i;
    rgb.rgbRed = tdA.red / i;
    rgb.rgbReserved = 0;
    return rgb;
}

/* TRANSDATA operators */
TRANSDATA operator+ (TRANSDATA tdA, TRANSDATA tdB) {
    TRANSDATA td;
    td.blue = tdA.blue + tdB.blue;
    td.green = tdA.green + tdB.green;
    td.red = tdA.red + tdB.red;
    return td;
}
TRANSDATA operator+ (TRANSDATA tdA, TRANSRESID trA) {
TRANSDATA td;
    td.blue = tdA.blue + trA.blue;
    td.green = tdA.green + trA.green;
    td.red = tdA.red + trA.red;
    return td;
}

TRANSDATA operator+ (TRANSRESID trA, TRANSDATA tdA) {return (tdA + trA);}

TRANSDATA operator+ (RGBQUAD rgbA, RGBQUAD rgbB) {
    TRANSDATA td;
    td.blue = rgbA.rgbBlue + rgbB.rgbBlue;
    td.green = rgbA.rgbGreen + rgbB.rgbGreen;
    td.red = rgbA.rgbRed + rgbB.rgbRed;
    return td;
}

TRANSDATA operator+ (TRANSDATA tdA, int i) {
    TRANSDATA td;
    td.blue = tdA.blue + i;
    td.green = tdA.green + i;
    td.red = tdA.red + i;
    return td;
}

TRANSDATA operator+ (int i, TRANSDATA tdA) {return (tdA + i);}

TRANSDATA operator<< (RGBQUAD rgbA, int i) {
    TRANSDATA td;
    td.blue = rgbA.rgbBlue <<i;
    td.green = rgbA.rgbGreen <<i;
    td.red = rgbA.rgbRed <<i;
    return td;
}

TRANSDATA operator& (TRANSDATA tdA, int i) {
    TRANSDATA td;
    td.blue = tdA.blue & i;
    td.green = tdA.green & i;
    td.red = tdA.red & i;
    return td;
}

TRANSDATA operator& (TRANSRESID trA, int i) {
    TRANSDATA td;
    td.blue = trA.blue & i;
    td.green = trA.green & i;
    td.red = trA.red & i;
    return td;
}

_*/*******************************************************************************/
* TRANSRESID operators
*******************************************************************************/

TRANSRESID operator+ (TRANSRESID trA, TRANSRESID trB) {
    TRANSRESID tr;
    tr.blue = trA.blue + trB.blue;
    tr.green = trA.green + trB.green;
    tr.red = trA.red + trB.red;
    return tr;
}
TRANSRESID operator- (RGBQUAD rgbA, RGBQUAD rgbB) {
TRANSRESID tr;
tr.blue = rgbA.rgbBlue - rgbB.rgbBlue;
tr.green = rgbA.rgbGreen - rgbB.rgbGreen;
tr.red = rgbA.rgbRed - rgbB.rgbRed;
return tr;
}

TRANSRESID operator- (TRANSDATA tdA, TRANSDATA tdB) {
TRANSRESID tr;
tr.blue = tdA.blue - tdB.blue;
tr.green = tdA.green - tdB.green;
tr.red = tdA.red - tdB.red;
return tr;
}

TRANSRESID operator- (TRANSRESID trA, TRANSRESID trB) {
TRANSRESID tr;
tr.blue = trA.blue - trB.blue;
tr.green = trA.green - trB.green;
tr.red = trA.red - trB.red;
return tr;
}

TRANSRESID operator* (int i, TRANSRESID trA) {
TRANSRESID tr;
tr.blue = trA.blue * i;
tr.green = trA.green * i;
tr.red = trA.red * i;
return tr;
}

TRANSRESID operator* (TRANSRESID trA, int i) {return (i*trA);}

TRANSRESID operator<< (TRANSRESID trA, int i) {
TRANSRESID tr;
tr.blue = trA.blue <<i;
tr.green = trA.green <<i;
tr.red = trA.red <<i;
return tr;
}

TRANSRESID operator^ (TRANSRESID trA, TRANSRESID trB) {
TRANSRESID tr;
tr.blue = trA.blue ^ trB.blue;
tr.green = trA.green ^ trB.green;
tr.red = trA.red ^ trB.red;
return tr;
}

////////////////////////////////////////////////////////////////////////////////////////
// Boolean Operators

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
bool operator!=(TRANSRESID trA, TRANSRESID trB) {
    bool b=true;
    if (trA.blue == trB.blue) b = false;
    if (trA.green == trB.green) b = false;
    if (trA.red == trB.red) b = false;
    return b;
}

bool operator1!=(RGBQUAD rgbA, RGBQUAD rgbB) {
    bool b = true;
    if (rgbA.rgbBlue == rgbB.rgbBlue) b = false;
    if (rgbA.rgbGreen == rgbB.rgbGreen) b = false;
    if (rgbA.rgbRed == rgbB.rgbRed) b = false;
    return b;
}

BYTE limit(int i) {
    if (i>255) return (255);
    else if (i<0) return (0);
    else return (i);
}

TRANS DATA mroundTD(TRANSDATA tdA, unsigned int i) {
    int prnd = l«(i-1);
    int nrnd = prnd -1;
    TRANSDATA td;
    td.blue = (tdA.blue >= 0 ? tdA.blue + prnd : tdA.blue + nrnd) >> i;
    td.green = (tdA.green >= 0 ? tdA.green + prnd : tdA.green + nrnd) >> i;
    td.red = (tdA.red >= 0 ? tdA.red + prnd : tdA.red + nrnd) >> i;
    return td;
}

TRANSRESID mroundTR(TRANSRESID trA, unsigned int i) {
    int prnd = l«(i-1);
    int nrnd = prnd -1;
    TRANSRESID tr;
    tr.blue = (trA.blue >= 0 ? trA.blue + prnd : trA.blue + nrnd) >> i;
    tr.green = (trA.green >= 0 ? trA.green + prnd : trA.green + nrnd) >> i;
    tr.red = (trA.red >= 0 ? trA.red + prnd : trA.red + nrnd) >> i;
    return tr;
}

RGBQUAD TDtoRGB(TRANSDATA tdA) {
    RGBQUAD rgb;
    assert (tdA.blue < 256);
    assert (tdA.green < 256);
    assert (tdA.red < 256);
}
rgb.rgbBlue = (BYTE)tdA.blue;
rgb.rgbGreen = (BYTE)tdA.green;
rgb.rgbRed = (BYTE)tdA.red;
rgb.rgbReserved = 0;
return rgb;
}

TRANSRESID bitprop(TRANSRESID trA, TRANSDATA bitO) {
  TRANSRESID tr;
  tr.blue = (trA.blue >=0) ? (trA.blue - bitO.blue) : (trA.blue + bitO.blue);
  tr.green = (trA.green >=0) ? (trA.green - bitO.green) : (trA.green + bitO.green);
  tr.red = (trA.red >=0) ? (trA.red - bitO.red) : (trA.red + bitO.red);
  return tr;
}

TRANSDATA bitprop(TRANSDATA tdA, TRANSDATA bitO, TRANSDATA bitl) {
  TRANSDATA td;
  td.blue = (tdA.blue >=0) ? (tdA.blue + bitO.blue - bitl.blue) :
            (tdA.blue + ((bitO.blue==0) ? bitl.blue : (bitO.blue - bitl.blue)));
  td.green = (tdA.green >=0) ? (tdA.green + bitO.green - bitl.green) :
            (tdA.green + ((bitO.green==0) ? bitl.green : (bitO.green - bitl.green)));
  td.red = (tdA.red >=0) ? (tdA.red + bitO.red - bitl.red) :
            (tdA.red + ((bitO.red==0) ? bitl.red : (bitO.red - bitl.red)));
  return td;
}
class list;

    // Generic 'node' class

class node
{
    friend class iter;
public:
    node( list * L =0, node * prv =0, node * nxt =0 )
        : mylist( 0 ), Prev( 0 ), Next( 0 )
    { link( L, prv, nxt ); }
    virtual ~node( void )
    { unlink(); }
    void link( list * L, node * prv, node * nxt );
    void unlink( );
    node * prevNode( void ) const
    { return Prev; }
    node * nextNode( void ) const
    { return Next; }

    // These classes allow any arbitrary type to be contained in a
type-safe linked list. All of the common code for list
management itself is contained in a common set of classes:
node, list and iter. Template classes derived from these
allow inline access to the underlying base classes via a
type-safe front-end.

#if ! defined( _list_h )
define _list_h

class list;

    // Generic 'node' class

class node
{
    friend class iter;
public:
    node( list * L =0, node * prv =0, node * nxt =0 )
        : mylist( 0 ), Prev( 0 ), Next( 0 )
    { link( L, prv, nxt ); }
    virtual ~node( void )
    { unlink(); }
    void link( list * L, node * prv, node * nxt );
    void unlink( );
    node * prevNode( void ) const
    { return Prev; }
    node * nextNode( void ) const
    { return Next; }

    // These classes allow any arbitrary type to be contained in a
type-safe linked list. All of the common code for list
management itself is contained in a common set of classes:
node, list and iter. Template classes derived from these
allow inline access to the underlying base classes via a
type-safe front-end.

#if ! defined( _list_h )
define _list_h

class list;

    // Generic 'node' class

class node
{
    friend class iter;
public:
    node( list * L =0, node * prv =0, node * nxt =0 )
        : mylist( 0 ), Prev( 0 ), Next( 0 )
    { link( L, prv, nxt ); }
    virtual ~node( void )
    { unlink(); }
    void link( list * L, node * prv, node * nxt );
    void unlink( );
    node * prevNode( void ) const
    { return Prev; }
    node * nextNode( void ) const
    { return Next; }

    // These classes allow any arbitrary type to be contained in a
type-safe linked list. All of the common code for list
management itself is contained in a common set of classes:
node, list and iter. Template classes derived from these
allow inline access to the underlying base classes via a
type-safe front-end.

#if ! defined( _list_h )
define _list_h

class list;

    // Generic 'node' class

class node
{
    friend class iter;
public:
    node( list * L =0, node * prv =0, node * nxt =0 )
        : mylist( 0 ), Prev( 0 ), Next( 0 )
    { link( L, prv, nxt ); }
    virtual ~node( void )
    { unlink(); }
    void link( list * L, node * prv, node * nxt );
    void unlink( );
    node * prevNode( void ) const
    { return Prev; }
    node * nextNode( void ) const
    { return Next; }
private:
    list * mylist;
    node * Prev, * Next;
};

// template node frontend

template<class T>
class Node : public node
{
public:
    Node( T data, list * L = 0, node * prv = 0, node * nxt = 0 )
        : node( L, prv, nxt ), Data( data ) {}
    Node<T> * next( void ) const
        { return (Node<T> *)nextNode(); }
    Node<T> * prev( void ) const
        { return (Node<T> *)prevNode(); }
    T & ref2data( void ) const
        { return (T & )Data; }
    T * ptr2data( void ) const
        { return (T *) &Data; }
    T data( void ) const
        { return Data; }
private:
    T Data;
};

// Generic 'list' class

class list
{
public:
    list( void )
        : First( 0 ), Last( 0 ), nodes( 0 ) {}
    virtual ~list( void )
        { purge(); }
    void purge( void );
    long items( void ) const
        { return nodes; }
    void addatstart( node * n )
        { n->link( this, 0, First ); }
    void addatend( node * n )
        { n->link( this, Last, 0 ); }
    void addafter( node * n, node * prv )
        { n->link( this, prv, 0 ); }
    void addbefore( node * n, node * nxt )
        { n->link( this, 0, nxt ); }
    node * firstNode( void ) const
        { return First; }
    node * lastNode( void ) const
        { return Last; }
protected:
    node * First, * Last;
    long nodes;
};

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
template<class T>
class List : public list
{
    public:
    List( void ) : list() {}
    Node<T> * add( T data, Node<T> * prv =0, Node<T> * nxt =0 )
    { return new Node<T>( data, this, prv, nxt ); }
    Node<T> * first( void ) const
    { return (Node<T> *)First; }
    Node<T> * last( void ) const
    { return (Node<T> *)Last; }
};

enum trOp
{
    FIRST, LAST, PREV, NEXT, CURR
};
#define TR_OK 0
#define TR_EMPTY -2
#define TR_NOMORE -3

class iter
{
    public:
    iter( list & L )
    : mylist( L ), nptr( 0 ) {}
    iter( iter const & I )
    : mylist( I.mylist ), nptr( I.nptr ) {}
    iter & operator=( iter const & I )
    { if ( &I.mylist == &mylist ) nptr = I.nptr; return *this; }
    void reset( void )
    { nptr = 0; }
    int traverse( trOp op );
    int current( void )
    { return traverse( CURR ); }
    int first( void )
    { return traverse( FIRST ); }
    int last ( void )
    { return traverse( LAST ); }
    int prev( void )
    { return traverse( PREV ); }
    int next( void )
    { return traverse( NEXT ); }

    protected:
    list & mylist;
    node * nptr;
};

// Iterator

template<class T>
class Iter : public iter
{
    public:
typedef int (*comparator)(const T&, const T&);

Iter( List<T> & L )
: iter( L ) {}
Iter( Iter<T> const & I )
: iter( I ) {}    
Iter<T> & operator=( Iter<T> const & I )
{ iter::operator=( I ); return *this; }
List<T> & myList( void ) const
{ return ( List<T> & )mylist; }
Node<T> * atNode( void ) const
{ return ( Node<T> * )nptr; }
T & ref2data( void ) const
{ return atNode()->ref2data(); }
T * ptr2data( void ) const
{ return atNode()->ptr2data(); }
T data( void ) const
{ return atNode()->data(); }
void addFirst( T data )
{ myList().addatstart( new Node<T>( data ) ); }
void addLast( T data )
{ myList().addatend( new Node<T>( data ) ); }
void addAfter( T data )
{ myList().addafter( new Node<T>( data ), nptr ); }
void addBefore( T data )
{ myList().addbefore( new Node<T>( data ), nptr ); }
void add( T data, trOp op );
trOp locate( T & data, comparator compare );
int addsorted( T data, comparator compare, int adddupe =0 );

template<class T> void Iter<T>::add( T data, trOp op )
{
  switch( op )
  {
  case FIRST: addFirst( data ); break;
  case LAST: addLast( data ); break;
  case PREV: addBefore( data ); break;
  case CURR: case NEXT: addAfter( data ); break;
  }
}

template<class T>
trOp
Iter<T>::locate( T & data, comparator compare )
{
  register trOp rc;
  register Node<T> * n = myList().first();

  if ( n == 0 ) // Add to start of empty list
    rc = FIRST;
  else
  {
    rc = LAST;
    while ( rc == LAST && n != 0 )
    {
      int r = compare( data, n->ref2data() );
    }
if ( r == 0 ) // Found an exact match
    rc = CURR;
else if ( r < 0 ) // We've gone past it
    rc = PREV;
else
    n = n->next();

nptr = n;
return rc;

}template<class T>
int
Iter<T>::addsorted( T data, comparator compare, int adddupe )
{
    trOp r;

    if ((( r = locate( data, compare )) != CURR ) || adddupe )
    {
        add( data, r );
        return 1;
    }
    return 0;
}
#endif // _list_h
// MODULE
// list.cpp
// CREATED
// davidn 03 Dec 1994 23:59
// David L. Nugent
// This class implementation is donated to the public domain
// DESCRIPTION
// Implementation of class node, list & iter
// FUNCTIONS
// node::unlink()
// Destroys linkage from a list and removes it from a
// linked list
// node::link()
// Links a node into a linked list
// list::purge()
// Removes all linked list entries
// iter::traverse()
// Provides full traversal functions for nodes in a
// linked list
// Implementation of class list & friends

#include "list.hpp"

void
node::unlink()
{
  if ( mylist )
  {
    // Unlink from previous
    if ( Prev )
      Prev->Next = Next;
    else
      mylist->First = Next;

    // Unlink from next
    if ( Next )
      Next->Prev = Prev;
    else
      mylist->Last = Prev;

    mylist->nodes--; 
    mylist = 0; 
    Prev = Next = 0;
  }
}

void
node::link( list * L, node * prv, node * nxt )
{
  // If currently linked, then unlink it

  // Code...
}
if ( mylist )
    unlink();

    // Link it to the list

if ( L )
{
    mylist = L;
    // Add after previous
    if ( prv )
    {
        Prev = prv;
        Next = prv->Next;
    }
    // Add before next
    else if ( nxt )
    {
        Next = nxt;
        Prev = nxt->Prev;
    }
    // Else just add to end
    else
    {
        Next = 0;
        Prev = L->Last;
    }

    if ( Prev )
        Prev->Next = this;
    else
        L->First = this;

    if ( Next )
        Next->Prev = this;
    else
        L->Last = this;

    mylist->nodes++;
}

void
list::purge( void )
{
    while ( First )
        delete First;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
int iter::traverse( trOp op )
{
    if ( mylist.firstNode() == 0 )
        return TR_EMPTY;

    switch ( op )
    {
        case NEXT:
            if ( nptr )
            {
                nptr = nptr->Next;
                break;
            }
        case FIRST:
            nptr = mylist.firstNode();
            break;
        case PREV:
            if ( nptr )
            {
                nptr = nptr->Prev;
                break;
            }
        case LAST:
            nptr = mylist.lastNode();
            break;
        case CURR:
            break;
    }
    return ( nptr ) ? TR_OK : TR_NOMORE;
}
BIBLIOGRAPHY


VITA

Graduate College
University of Nevada, Las Vegas

Joseph V. Kraft

Home Address:
6221 NW 77th St.
Oklahoma City, OK 73132

Degrees:
Bachelor of Science, Electrical and Electronic Engineering, 1990
North Dakota State University, Fargo, ND

Thesis Title: Progressive Transmission of Digital Images

Thesis Examination Committee:
Chairperson, Dr. Shahram Latifi, Ph.D.
Committee Member, Dr. Henry Selvaraj, Ph.D.
Committee Member, Dr. Eugene McGaugh, Ph.D.
Graduate Faculty Representative, Dr. Evangelos Yfantis, Ph.D.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.