Testing a Set of Image Processing Operations for Completeness

Leonard Brown
Le Gruenwald
The University of Oklahoma
School of Computer Science
Norman, OK, 73019
lbrown@cs.ou.edu, gruenwal@cs.ou.edu

Greg Speegle
Baylor University
Department of Computer Science
Waco, TX, 76798
speegle@mercury.baylor.edu

Abstract

In order to more efficiently represent image editing in multimedia databases, research has been performed to determine the usefulness of storing one image file as a base along with a set of operations that represent the modifications to the image that occured during editing. Because any editor should be able to produce and interpret them, these operations should be elements of some standardized set. One of the properties that this standardized set must have is completeness so that it is able to represent any of the modifications a software editor may produce. In this paper, we propose a method for testing whether or not any general set of image operations is complete. Part of this proposal includes formal definitions for both images and completeness. In addition, an example of this test is presented for a set of image processing operations.

1. Motivation

One of the functions of any database system is to facilitate editing of its data. This process becomes more complex in a multimedia database management system (MMDBMS) because of the size of multimedia data. Although multimedia data can be characterized into various types such as audio, video, still images, text, and graphics, they all have the characteristic of being space intensive [9]. For example, CD quality audio uses 1.4 Megabits per second, NTSC quality video uses 1.92 Megabits per frame, and an image stored as a bitmap can use 4,000,000 bytes of storage [1, 17].

To illustrate the problems with editing multimedia objects, imagine an application for an interior designer. This application allows the designer to view a photo of the room and decorate it by editing the picture. The designer could change the color of the walls and carpet, add, remove, or rearrange the furniture, and experiment with different lighting effects.

In addition to this example, imagine a different application that lets a recording studio edit one of its songs. The application would let the users select a song, then add several sound effects, alter the voices, or merge samples from other recordings.

In both of these examples, the users start with a base image or audio file and perform different editing operations on it. The users will want to see and hear each modification they perform. When the users begin to create new files that are modifications of the base files, then save both the old and new data, the MMDBMS may quickly run out of storage.

Researchers have noted that there is redundancy in storing both the base and modified data in an MMDBMS [7]. Because of the size of multimedia data, it may be more efficient to use data references which will act as abstractions of the data [1]. As an alternative to simply compressing the data, it has been proposed to store only

the base data with a series of instructions explaining how to generate the new data [7]. Not only would this method conserve much more space than compression, it would eliminate any degradation that results from compression algorithms that are lossy [7].

When the user creates the new data, the editor used must store the instructions explaining how it was created in the MMDBMS. In addition, if other users want to recreate the new data, their editors must know how to interpret the set of instructions they receive back from the database. For all of the editors to interpret the instructions correctly, the set of operations used in the instructions will have to be part of some previously defined standard set, called a logical model language (LML) by researchers [7].

Figure 1 illustrates this process for an image database. A user performs an action on an image such as clicking on the red option. The editor receives this action and translates it into a standard operation, such as 'change color red'. The editor sends the command to the MMDBMS which performs the corresponding operation (x, y, color) $\rightarrow$ (x, y, red) on the image. The MMDBMS then returns the modified image to the editor, which will then display the new image to the user.

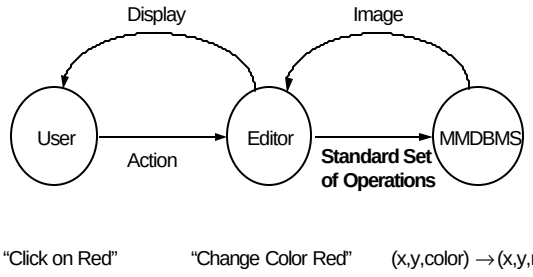

Figure 1 - Image Editing using Standardized Operations

For the LML to be useful, it must have the ability to represent any transformation performed by the user's editor. Since all of the possible image transformations are not known, it is difficult to test for this ability.

It is the goal of this paper to solve this problem for logical model languages for image databases, which we call image LMLs. To accomplish this, it will give a precise definition of an image as well as define the completeness property for image LMLs. In addition, it will test a sample LML for completeness using the method presented.

The structure of the remainder of this paper is as follows: In section 2, we will describe and evaluate a previous definition of completeness. In section 3, we will state our definition of completeness as well as our definition of an image. In section 4, we will propose a methodology for testing an image LML for completeness. In section 5, we will describe a sample LML, and then in section 6, we will test it for completeness using the proposed technique. In section 7, we will summarize our research and describe the work that will be continued in the future.

2. Related work

Developing an image LML was the subject of research performed in [7] and [4]. To express all of the possible image transformations performed by a given editor, the researchers examined two sources, namely the existing image processing software and the image algebra described in [14] and [15]. While the operations in the LML are defined using the image algebra, the definition of completeness is based on the existing software.

The definition from the researchers is that a complete image LML has the capability of expressing any operation that can be performed by version 3.0 of Adobe Systems Incorporated's Photoshop. The reason this software package was selected is that it outsells competing software packages by a factor of four [4]. In addition, other software reviewers regard it as powerful when comparing its functionality to its competitors [10, 11, 12].

The main drawback to this definition is that it requires accepting two assumptions. The first assumption is that Photoshop 3.0 can express every image transformation. The second is that every operation in Photoshop can be expressed by the operations in the LML being evaluated. Proving the first assumption requires that all of the possible image transformations are known. Proving the second requires a list of the LML operations needed for each Photoshop command.

3. Definitions

Since our goal is to provide a methodology for determining whether a set of image operations is complete, we must precisely define what an image is, what an image operation is, and what completeness means. We will define an image by examining how it is used. Once the definition of an image is established, we will define image operations, then define completeness with respect to image operations.

3.1 Image Background

Despite how an image is logically stored in a computer, the goal is to visually display it. Computer screens are

composed of a grid of small, square, colored dots. These dots are called pixels. [2, 13].

For a computer to display an image, it must be digitized. This means that the image must be decomposed into a grid of pixels [5]. Since a computer must digitize an image in order to display it, we will define images as a grid of pixels.

An individual pixel can be thought of as having two components, its coordinate point and its value [15]. Both of these components must be defined. Since an image is a grid of pixels, we will restrict the coordinates of a single pixel to a 2-dimensional plane. The pixel's value is intended to represent its color.

Representing a color numerically is the goal of color theory. There are several models used for representing color. One widely used model is RGB which stands for Red, Green, and Blue. Combinations of these three colors, called channels, are used to create the other colors in the RGB model. This model is based on the way colors are perceived by humans in that red, green, and blue are the basis for all of the colors in nature [5].

Another widely used model is CMYK, which stands for Cyan, Magenta, Yellow, and blacK. This color model is based on the inks used in printing where the percentages of cyan, magenta, yellow, and black inks are specified for each printed dot [5]. So, to associate a pixel with a color in the CMYK model, a value between 0 and 100 is required for each of the four channels.

These two examples show that there are a different number of channels necessary to represent the color of a pixel depending on the model used. Because of this, a pixel must be a coordinate pair associated with a set of channels. This means an image is a set of coordinate points where each point has a set of channels and values.

3.2 Image Definition

As seen from the informal definition, an image is defined as a set of pixels where each pixel has a unique x and y coordinate pair. A pixel is defined as a set of tuples {<x, y, c, v>} where x represents the value of the x-coordinate and y represents the value of the y-coordinate. Since each pixel corresponds to exactly one x and y coordinate pair, all of the x values are equal, and all of the y values are equal in the set.

In each tuple, the element c represents one of the channels of the color mode used by the pixel, such as red in the rgb color mode. The element v represents the value of the channel for the pixel. This allows us to represent a single pixel by a set of tuples $\{<x_1, y_1, c_1, v_1>, <x_2, y_2, c_2, v_2>, \ldots, <x_k, y_k, c_k, v_k>\}$ where $k \geq 0$, $x_i = x_j$ and $y_i = y_j$ for all $i \geq 1$ and $i \leq k$ and for all $j \geq 1$ and $j \leq k$. Note that this means each tuple represents one channel, and adding or removing tuples corresponds to adding or removing channels.

The domains of x and y are the set of integers, Z, since x and y come from the discrete lattice Z^2 described in [14] and [15]. The domain of each c_i is composed of the various channels that can be defined on an image. The domain of each v_i is dependent on the value of the corresponding c_i in its tuple. For example, if c_i is red of the rgb color mode using 24-bit color, then the domain of v_i is all integers between 0 and 255. If c_i is yellow of the cmyk color model, then the domain of v_i is all integers between 0 and 100 [5, 13].

As an example of this image definition, say that we have a pixel in the rgb color model. The pixel is located at the point (5, 10) and has a red channel value of 15, a green channel value of 30, and a blue channel value of 40. According to the definition used in this paper, the pixel is represented as a set of three tuples, namely {<5, 10, red, 15>, <5, 10, green, 30>, and <5, 10, blue, 40>}.

3.3 Image Operations

The above definition of an image helps clarify what is meant by an image operation. In this paper, an image operation performs a transformation from an image A to another image B. This definition of an image operation excludes those operations whose result is not an image. For example, cardinality is an operation that can be performed on an image, but its result is an integer, namely the number of pixels in the image. As far as our definition is concerned, this is not an image operation.

3.4 Completeness Definition

Instead of comparing an image LML to a software package as in the related work, the approach used in this paper is to base the completeness definition on the definition of an image. To say that an image LML is complete, it must be able to express all transformations from one image to another. So, given any two images A and B, there should exist a sequence of LML operations to convert A to B and B to A.

4. Testing for Completeness

In this section, we will describe the method used to test if an image LML satisfies the completeness property. We will accomplish this by describing an algorithm for

converting any image A to another image B. If the operations in the image LML can perform all of the image operations needed by the algorithm, then it can convert any image A to another image B. By our definition, this means that it is complete.

4.1 Algorithm Description

The algorithm we will use is based on the fact that images are sets. Informally, given one set of elements, A, and another set of elements, B, we will convert A to B by first adding to or deleting elements from A until it has the same number of elements as B. Then, we will modify the values of each of the elements in A so that they are the same as the elements in B.

4.2 Algorithm

Formally, let A be any image and let the cardinality of A = m where $m \geq 0$. We can represent A as $\{P_1, P_2, \ldots, P_m\}$ where each P_i is a pixel. Similarly, let B be any image and let the cardinality of B = n, where $n \geq 0$. We can represent B as $\{Q_1, Q_2, \ldots, Q_n\}$ where each Q_i is a pixel.

Step 1: If $m \geq n$, then remove a pixel from A (m - n) times, otherwise, add a pixel to A (n - m) times. Upon completion of this step, if $m \geq n$, then the cardinality of A will equal m - (m - n), which equals n. If m < n, then the cardinality of A will equal m + (n - m), which also equals n. Thus, after this step, the cardinality of A will equal the cardinality of B.

Step 2: Since the cardinality of both images are now equal, we can now express A as $\{P_1, P_2, \ldots, P_n\}$ and B as $\{Q_1, Q_2, \ldots, Q_n\}$. In this step, for each i between 1 and n, we will set $P_i = Q_i$. Upon completion of this step, A will now contain the elements $\{Q_1, Q_2, \ldots, Q_n\}$.

In our definition, each P_i and Q_i are sets of tuples. This means that setting P_i equal to Q_i will be more complex than a simple assignment statement. For each of the pixels in image A, we will add or remove tuples from the pixel until its cardinality equals the cardinality of the corresponding pixel in image B. Then we will set the tuples in A equal to the corresponding tuples in B.

For each i between 1 and n, perform the following sub-steps:

Step 2.1: Let h be the cardinality of Q_i and k be the cardinality of P_i. If $k \geq h$, then remove (k - h) tuples from P_i, otherwise add (h - k) tuples to P_i. Upon completion of this step, if $k \geq h$, then the cardinality of P_i will equal k - (k - h), which equals h. If k < h, then the cardinality of P_i will equal k + (h - k), which also equals h. Thus, after this step, the cardinality of P_i will equal the cardinality of Q_i.

Step 2.2: Now, we will set P_i equal to Q_i. Since the cardinalities of both P_i and Q_i equal h, we will represent P_i as $\{P_{i1}, P_{i2}, \ldots, P_{ih}\}$ and Q_i as $\{Q_{i1}, Q_{i2}, \ldots, Q_{ih}\}$, where each P_{ij} and Q_{ij} is a 4-tuple described earlier.

For each j between 1 and h, perform the following:

Step 2.2.1: Set the value of the x-variable in P_{ij} equal to the value of the x variable in Q_{ij}.

Step 2.2.2: Set the value of the y-variable in P_{ij} equal to the value of the y variable in Q_{ij}.

Step 2.2.3: Set the value of the c-variable in P_{ij} equal to the value of the c variable in Q_{ij}.

Step 2.2.4: Set the value of the v-variable in P_{ij} equal to the value of the v variable in Q_{ij}.

4.3 Algorithm Correctness

After completion of the substeps 2.2.1 - 2.2.4, the tuple P_{ij} will equal the tuple Q_{ij}. This is true because each of the 4 corresponding components of the tuples will be equal.

After these substeps are performed for each of the j tuples in P_i, then each tuple P_{ij} will equal Q_{ij} for all j between 1 and h. So, if a 4-tuple <x, y, c, v> is in Q_i, then there will be a 4-tuple <x, y, c, v> in P_i. So, $Q_i \subseteq P_i$. Similarly, if a 4-tuple <x, y, c, v> is in P_i, then there will be a 4-tuple <x, y, c, v> in Q_i. So, $P_i \subseteq Q_i$. Therefore $P_i = Q_i$.

This means that when step 2 completes for each of the n pixels in image A, P_i will equal Q_i for all i between 1 and n. So, if there is a pixel $Q_i = \{Q_{i1}, Q_{i2}, \ldots, Q_{ih}\}$ in B, there will be a pixel $\{Q_{i1}, Q_{i2}, \ldots, Q_{ih}\}$ in A. So, $B \subseteq A$. Similarly, since $P_i = Q_i$ for all i, if there is a pixel $Q_i = \{Q_{i1}, Q_{i2}, \ldots, Q_{ih}\}$ in A, there will be a pixel $\{Q_{i1}, Q_{i2}, \ldots, Q_{ih}\}$ in B. So, $A \subseteq B$. Therefore, A = B.

This means that at the conclusion of this algorithm, image A will be the same as image B.

4.4 Image Operation List

The algorithm given above uses eight image operations. In step 1, a pixel is either added to or removed from an image. In step 2.1, a tuple is either added to or removed from a pixel. In step 2.2.1, the x variable of a tuple is modified. In step 2.2.2, the y variable of a tuple is modified. In step 2.2.3, the c variable of a tuple is modified. In step 2.2.4, the v variable of a tuple is modified.

Table 1 summarizes the image operations used in our algorithm.

1.	An operation to add a pixel to an image
2.	An operation to remove a pixel from an image
3.	An operation to add a tuple to a pixel
4.	An operation to remove a tuple from a pixel
5.	An operation to change pixel element x
6.	An operation to change pixel element y
7.	An operation to change pixel element c
8.	An operation to change pixel element v

Table 1 - Image Operations Used in the Testing for Completeness Algorithm

So, if an image LML can perform each of these eight image operations, then it can express every transformation from one image to another. This means that it is complete.

5. An Example LML

To demonstrate our technique for testing for completeness of an image LML, we will use the LML proposed in [7] which contains six operations. These operations are called merge, define, mutate, modify, combine, and applyfunction.

5.1 Merge

The merge operation combines a base image, A, with a new image, B. The result is the union of the pixels in the two images. This operation is flexible enough to be either opaque or transparent. If it is transparent, whenever images A and B have pixels at the same coordinates, the value of the pixels in A will be used. In the same situations, if it is opaque, the values of the pixels in B will be used.

Formally, let A be the base image where $A = \{P_1, P_2, \ldots, P_m\}$ with $m \geq 0$, and B be an image $\{Q_1, Q_2, \ldots, Q_n\}$ with $n \geq 0$. Recall from section 3 that a pixel is defined as a set of 4-tuples $\{<x, y, c, v>\}$ where all of the values of x are equal, and all of the values of y are equal. The result of merging A with B returns the union of their pixels based on the x and y coordinates, meaning that if there is a pixel P_i in A where $P_i = \{<x_i, y_i, c_{i1}, v_{i1}>, <x_i, y_i, c_{i2}, v_{i2}>, \ldots, <x_i, y_i, c_{ik}, v_{ik}>\}$, and there is a pixel Q_j in B where $Q_j = \{<x_j, y_j, c_{j1}, v_{j1}>, <x_j, y_j, c_{j2}, v_{j2}>, \ldots, <x_j, y_j, c_{jh}, v_{jh}>\}$, such that $x_i = x_j$ and $y_i = y_j$, then we will include only one of the pixels in the resultant union.

5.2 Define

The define operation selects a subset of an image. This operation is frequently referred to as cropping.

Formally, let A be an image where $A = \{P_1, P_2, \ldots, P_m\}$ with $m \geq 0$. Performing the define operation on A will create a new image $\{P_1, P_2, \ldots, P_r\}$ with $m \geq r$.

5.3 Mutate

The mutate operation changes the position of pixels in an image. This means that it assigns new values to the x and y coordinates of some of the pixels. In terms of our image definition, this operation will assign new values to the x and y variables of the tuples in a pixel. Since the result of this operation must be an image, there cannot be another pixel in the resulting image with the same x and y coordinates.

5.4 Modify

The modify operation affects the values of the channels of an image. It will change the color of some of the pixels in an image to a new color. In terms of our image definition, this corresponds to changing the value of the v variable in a tuple.

5.5 Combine

The combine operation is similar to the modify operation in that is also changes the colors of an image. The difference is that it computes the new color of a pixel by using the colors of adjacent pixels. As with the modify operation, this corresponds to changing the value of the v variable in a tuple in terms of our image definition.

5.6 ApplyFunction

Finally, the applyfunction operation manipulates the channels used by an image. Unlike the modify and combine operations, it edits the channels themselves, not their values. Among the functions this operation performs is adding a channel, removing a channel, and replacing a channel. In terms of our image definition, it affects the values of the c variables in the tuples.

6 Testing the sample LML

For an image LML to be complete, it must be able to perform each of the eight low level image operations needed by the algorithm described in section 4. In this section, we will show that the sample LML described in section 5 can perform each of the eight operations.

6.1 Adding a pixel

Adding a pixel to an image can be performed by the merge operation in the LML by merging a base image with an image containing a single pixel provided that the pixel does not have the same coordinates as a pixel in the base image. Formally, let A be any image where A = $\{P_1, P_2, \ldots, P_m\}$. To transform A into $\{P_1, P_2, \ldots, P_m, Q_1\}$, we can merge A with an image B = $\{Q_1\}$, where Q_1 = {<x', y', c_1', v_1'>, <x', y', c_2', v_2'>, ... , <x', y', c_h', v_h'>}, and there is not a pixel $\{<x, y, c_1, v_1>, <x, y, c_2, v_2>, \ldots, <x, y, c_k, v_k>\}$ in A such that x = x' and y = y'.

6.2 Removing a pixel

Removing a pixel can be accomplished with the define operation. Informally, we will define a new image from the base image by selecting all of the pixels in the base image except for a single pixel. Formally, let A be an image $\{P_1, P_2, \ldots, P_m\}$. To remove a pixel P_j from A, we define a new image A' = $\{P_1', P_2', \ldots, P_{m-1}'\}$ where $P_i' = P_i$ if i < j and $P_i' = P_{i+1}$ if i ≥ j.

6.3 Adding a tuple

Recall from section 3.2 that adding a tuple to a pixel corresponds to adding a new channel. To add a tuple to a pixel, we will have to define a new channel at the same coordinates and assign it a value. Since the applyfunction operation in the proposed image LML can add a channel, this task can be accomplished by it. Formally, let P be a pixel $\{<x, y, c_1, v_1>, <x, y, c_2, v_2>, \ldots, <x, y, c_k, v_k>\}$. Applyfunction can add a tuple $<x, y, c_{k+1}, v_{k+1}>$ to P creating a new pixel $\{<x, y, c_1, v_1>, <x, y, c_2, v_2>, \ldots, <x, y, c_k, v_k>, <x, y, c_{k+1}, v_{k+1}>\}$.

6.4 Removing a tuple

Removing a tuple from a pixel corresponds to removing a channel from a pixel. Since the applyfunction operation in the proposed image LML can remove a channel, this task can be accomplished by it. Formally, let P be a pixel $\{<x, y, c_1, v_1>, <x, y, c_2, v_2>, \ldots, <x, y, c_k, v_k>\}$. Applyfunction can remove a tuple $<x, y, c_j, v_j>$ from P creating a new pixel $\{<x, y, c_1', v_1'>, <x, y, c_2', v_2'>, \ldots, <x, y, c_{k-1}', v_{k-1}'>\}$ where each $c_i' = c_i$ if i < j and $c_i' = c_{i+1}$ if i ≥ j.

6.5 Modifying x and y

Changing the x and y coordinates in a tuple can be performed by the mutate operation. Informally, we assign new values to the x and y components of each tuple in a pixel. Of course, the resulting image cannot contain two pixels with the same x and y coordinates.

Formally, let P be a pixel P = $\{<x, y, c_1, v_1>, <x, y, c_2, v_2>, \ldots, <x, y, c_k, v_k>\}$. The mutate operation will set $<x, y, c_i, v_i>$ equal to $<x', y', c_i, v_i>$ for all i = 1, k. In order for this operation to be valid, there can be no pixel $\{<x'', y'', c_1, v_1>, <x'', y'', c_2, v_2>, \ldots, <x'', y'', c_l, v_l>\}$ in the original image such that x'' = x' and y'' = y'.

6.6 Modifying c

Changing the c value in a tuple corresponds to replacing a channel with another in a tuple. This function can be performed by the applyfunction operation

according to its definition. Formally, let P be a pixel P = {<x, y, c_1, v_1>, <x, y, c_2, v_2>, ... , <x, y, c_k, v_k>}. To replace a channel c_j with some c', we will set the tuple <x, y, c_j, v_j> equal to <x, y, c', v_j>. Note that if v_j is not in the domain of c', then it will also have to be modified.

6.7 Modifying v

The modify and combine operations change the value v in a tuple. The only restriction on changing the value of v is that the new value must remain in the domain of the c value in its tuple.

Formally, let P be a pixel {<x, y, c_1, v_1>, <x, y, c_2, v_2>, ... , <x, y, c_k, v_k>}. To change the value v_i in a tuple <x, y, c_i, v_i> to some value v' in the domain of c_i, we simply replace the tuple by <x, y, c_i, v'>.

6.8 Completeness Summary

To summarize, the 8 possible general operations that can be performed on an image can be expressed in the operations defined in the image LML in [7]. This means that the proposed image LML is complete. The general operations and their corresponding LML operations are given in table 2.

Image Operations	***Corresponding LML Operations***
Add a pixel to an image	Merge
Remove a pixel from an image	Define
Add a tuple to a pixel	Applyfunction
Remove a tuple from a pixel	Applyfunction
Change x	Mutate
Change y	Mutate
Change c	Applyfunction
Change v	Modify or Combine

Table 2 - Image Operations and Their Corresponding LML Operations

7. Conclusion

To reduce the amount of space necessary to store a series of edited versions of an image, some of the versions can be stored as a set of instructions. So that the images can be interpreted by any editor, the set of image processing operations in the instructions must be part of some standardized set called a Logical Model Language (LML) [7]. To demonstrate that the standardized set of operations is useful, it must be proven to be complete.

In this paper, we have proposed a methodology for testing whether or not a set of image processing operations is complete. As part of accomplishing this, we have provided a definition of completeness and a formal definition of an image. In addition, we have demonstrated this method using a sample set of image operations.

Completeness is not the only property needed to establish whether or not an image LML is an acceptable standard. In addition, the LML should be independent and minimal [7, 4]. A precise definition and testing method for each of these properties is part of continuing research. Also, this paper has only addressed LMLs for image databases. This work needs to be extended to other multimedia data types, namely audio, video, text, and graphics data.

References

1. Aberer, Karl and Wolfgan Klas, "*The Impact of Multimedia Data on Database Management Systems*", International Computer Science Institute, Berkeley, 1992.

2. Adobe Photoshop 3.0 User Guide, Adobe Systems Inc., 1994.

3. Banjerjee, Jay et. al., *"Data Model Issues for Object-Oriented Applications*", Readings in Object-Oriented Databases, 1990, pp. 197-208.

4. Basit, Mujeeb, Paul Parker, and Greg Speegle, "*Application of Image Algebra to Views of Images in Multimedia Databases*", Honors Thesis, The University of Baylor, April, 1996.

5. Greenberg, Adele Droblas and Seth Greenberg, Fundamental Photoshop, McGraw-Hill, Inc., Berkeley, 1995.

6. Gruenwald, Le and Greg Speegle, "*Views of Multimedia: A Mechanism for Supporting Media Editing with Databases*", NSF Proposal, September, 1996.

7. Gruenwald, Le and Greg Speegle, *"Research issues in View-Based Multimedia Databases"*, 2nd World Conference on Integrated Design and Process Technology, December, 1996.

8. Gruenwald, Le, Greg Speegle, and Wang, *"A Meta-Structure for Supporting MM Editing in OODBMS"* , Work in progress, 1997.
9. Khoshafian, Setrag and Brad Baker, Multimedia and Imaging Databases, Morgan Kaufman, 1996.

10. Marshall, Patrick, *"Electric Results"*, *Info World*, Vol. 17, Issue 11, March 13, 1995.

11. Marshall, Patrick, *"Electric Results"*, *Info World*, Vol. 17, Issue 19, May 8, 1995.

12. Marshall, Patrick, *"Electric Results"*, *Info World*, Vol. 17, Issue 26, June 26, 1995.

13. McClelland, Deke, Photoshop 3 Bible, IDG Books Worldwide, Inc., Chicago, 1994.

14. Ritter, Gerhard X., "Image Algebra", Unpublished manuscript available anonymous ftp at ftp.cis.ufl.edu in the /pub/src/ia/documents directory.

15. Ritter, Gerhard X. and Joseph N. Wilson, Handbook of Computer Vision Algorithms in Image Algebra, CRC Press, New York, 1996.

16. Speegle, Greg, *"Views of Media Objects in Multimedia Database Management Systems"*, Proceedings of the 1st International Workshop on MMDBMS, August, 1995.

17. Woelk, Darrell, W. Kim, and W. Luther., *"An Object-Oriented Approach to Multimedia Data"*, Readings in Object-Oriented Database Systems, Morgan Kaufman, 1990.