DETERMINING A MINIMAL AND INDEPENDENT SET OF IMAGE PROCESSING OPERATIONS FOR A MULTIMEDIA DATABASE SYSTEM

Leonard Brown
Le Gruenwald

The University of Oklahoma
School of Computer Science
Norman, OK, 73069
lbrown@cs.ou.edu, gruenwal@cs.ou.edu

ABSTRACT

Instead of saving multiple versions of a large multimedia data file, it is more efficient to store only the instructions used by the editors to create the different versions. To uncompress files stored in this manner, the editor would simple perform the listed instructions. For these files to be portable, however, the set of operations used in the instructions must be standardized. To standardize a set of operations, there must be some criteria developed for the inclusion of each operation. This paper describes two desired properties of a set of image processing operations, namely minimality and independence. In addition, this paper demonstrates methods for testing for each of them.

1. MOTIVATION

Multimedia data can be categorized into various types such as audio, video, still images, text, and graphics. The one common attribute of all of these data types is that they all require large amounts of space (Khoshafian and Baker, 1996). For example, CD quality audio uses 1.4 Mb per second, NTSC quality video uses 1.92 Mb per frame, and an image stored as a bitmap can require 4,000,000 bytes (Aberer and Klas, 1992, Woelk, et. al, 1990).

One of the functions of a Multimedia Database Management System (MMDBMS) is to allow users to arbitrarily edit multimedia data (Grosky, 1994). The enormous size of the multimedia data described above causes problems for the database systems that must maintain it.

To illustrate these problems, we will consider three example applications. The first application is one for an interior designer. This application would allow the designer to decorate a room by editing a photo of it. Among the changes the designer could make are changing the color of the walls and carpet; adding, removing, or rearranging the furniture; and, finally, changing the lighting in the room. The designer will also want to save several different versions of the room and be able to retrieve them to show to the customer.

The second example application is one for a recording studio. This application would allow producers to enhance a recording by adding different sound effects to it. In addition, the producers could add other voices or sample from other songs to the recordings. The producers, of course, will want to save several different versions of the recordings to determine the ones that are the best to sell.

The final example is an application for the development of large scale software. During the coding and testing phases of the development, several modifications will be made to the source code. While debugging, the programmers will need to

save several different versions of the code, and then retrieve the later.

Each of these three examples stores a different type of multimedia data, namely, still images, audio, and text. Each application, however, has similar requirements. They must allow a user to start with a base object, create new objects by editing it, and save the new versions of the object.

Since multimedia data is space intensive, when users create and save several versions of these objects, they will quickly run out of storage. Thus, for these and other similar applications, it is necessary to use some abstractions of the data that will allow appropriate references to the multimedia objects, but use less space (Aberer and Klas, 1992, Klas and Aberer, 1995).

To solve this problem of editing multimedia data, researchers have proposed taking advantage of the similarity between the original and modified data files. In Gruenwald and Speegle (1996), it has been proposed to develop an MMDBMS that stores only the original data files with a series of instructions explaining how to generate the new data.

Figure 1 illustrates an example of 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.

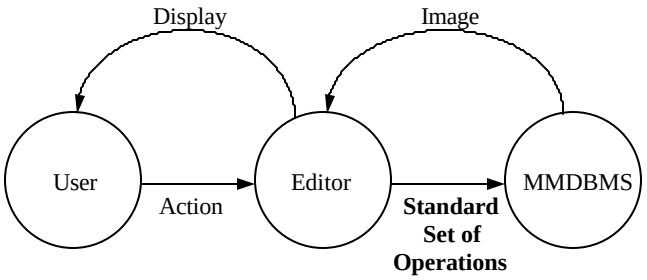

Figure 1 - View-Based MMDBMS Process for Images

This method has many advantages over traditional image data compression techniques. First, storing a sequence of instructions can potentially save much more space than any bitwise coding technique. Second, the given data compression technique is lossless since the derived objects can be recreated using the stored sequence of operations. Finally, since this technique does not specify how the base object should be stored, it can be used together with existing compression techniques. For example, a base image in an MMDBMS can be stored using the JPEG compression standard, and the derived images can be stored using only a sequence of image editing operations.

For this method of data storage to be useful, it must be portable. This means that any sequence of instructions generated by a multimedia editor must be interpreted by any other editor. For different editors to interpret the instructions correctly, the set of operations used must be a part of some previously standardized set. In Gruenwald and Speegle (1996), this set of operations is called a Logical Model Language (LML).

The goal of this paper is to provide a theoretical basis for establishing such a standardized set of image processing operations. We will focus on two properties of a set of operations, namely independence and minimality. We will provide formal definitions of each of these properties, and describe procedures for testing for them.

The remainder of this paper is organized in the following manner: In section 2, we will discuss the desired properties of a set of image processing operations. In section 3, we will define minimality, and demonstrate the testing procedure for it. We will do the same for independence in section 4. Finally, in section 5, we will summarize our work.

2. RELATED WORK

Several researchers are compiling sets and lists of image processing operations. Some are compiling the list of operations for a particular application (Webb, 1992, Joseph and Cardenas, 1988, Bengtsson et. al., 1981), while others are creating the list for general use (Klette and Zamperoni, 1996). To be able to argue that their lists are complete, however, the researchers include as many image processing operations as possible.

In this paper, we use the definition of completeness presented by Brown et. al. (1997). Specifically, a complete set of image processing operations has the ability to express any transformation from one image to another. In addition, Brown et. al. (1997) demonstrates that although it may be inefficient, this ability can be performed with a small number of operations, specifically six.

To extend the work performed by Brown et. al. (1997), other useful properties of an LML must be determined. As in Webb (1992), we view the set of image processing operations as a *little language*. Little languages are defined as languages that are used for a specific problem, but do not have the features of traditional programming languages (Bentley, 1986).

Bentley (1986) lists several goals of the design of a little language. In addition to the ability of describing all possible objects, the author states that the removal of any unnecessary operations should also be a goal. To satisfy these two goals,

we will determine when an image processing operation in a set is necessary by defining when a particular operation is independent, and when the entire set is minimal.

3. MINIMALITY

In Brown et. al. (1997), we provide a method for determining if a given set of image processing operations is complete. However, to justify that each operation in a complete set is necessary, we must also show that the removal of any operation from the set means that it is no longer complete. More formally, we must show that no subset of the set of operations is complete. This means that the set is minimal.

If a set is complete, then the addition of any operator will result in a complete set. This can be proven from the fact that since the original set of image processing operations is complete, it can express all image transformations. Since adding operations does not reduce the number of transformations a set can perform, adding operations to a complete set will always create another complete set.

Conversely, if the addition of operators does not result in a complete set, the original set was not complete. This implies that if a set is not complete, then no subset of it will be complete either. Using this information, we have developed the following procedure to determine if a set is minimal:

Let S be a set $\{O_1, O_2, ..., O_N\}$, where each O_i is an image procession operation.
Show S is complete (*Otherwise S is not minimal*)
For i = 1 to N
Let S' = S - $\{O_i\}$.
Show S' is not complete (*Otherwise S is not minimal*).

We will apply this test for minimality on the LML described by Brown et. al. (1997). This set consists of six image processing operations called merge, define, mutate, modify, combine, and applyfunction. The merge operation combines two images together to form a new image. The define operation creates a new image by selecting a subset of an image and creating new pixels outside the image. The mutate operation changes the location of some of the pixels in the image. The modify operation changes the color of the pixels in an image explicitly, while the combine operation changes the color by calculating new values from the neighbors. Finally, the applyfunction operation manipulates the color mode used by the pixels.

These six image processing operations are our set, S. According to the procedure described above, the first step in determining if S is minimal is to prove that S is complete. As an example used to illustrate the testing procedure, this set was proven complete by Brown et. al. (1997).

The next step is to repeatedly remove an operation from S, and test the remaining five operations for completeness. So, defining O_1 to be the merge operation, we will let S' = S - {merge}, then test if S' is complete.

According to the completeness test, we must show that S' can perform eight simple operations, otherwise it is not complete. The eight operations are adding a pixel, removing a pixel, adding a channel, removing a channel, changing the horizontal position of the pixel, changing the vertical position of the pixel, modifying a channel, and modifying the value of the channel.

These operations come from the definition of an image used by Brown et al. (1997). This definition is formed from the traditional notion of an image $f(x, y) = (v_1, v_2, ..., v_n)$, where x is the horizontal position, y is the vertical position, and each v_i is a value of a channel of the pixel (Gonzales and Woods, 1992). We convert this equation of an image to one describing an image as a set of pixels, where each pixel is a set of triples $\{<x, y, v_i>\}$. When we explicitly represent each channel by a variable c_i, we define each pixel to be a set of 4-tuples $\{<x, y, c_i, v_i>\}$.

Both adding and removing a pixel can be performed by the define operation in S'. Since one of its functions is to create a new image by selecting a superset of an image, we can add one new pixel to an image. Similarly, since one of the functions of the define operation is to create a new image by selecting a subset of an image, we can select all of the pixels except the desired one in an image. Adding, removing, and modifying a channel can all be performed by the applyfunction operation, depending on the parameters used. The mutate operation can be used to change both the horizontal and vertical positions of a pixel. Finally, the modify operation can be used to change the value of a channel of a pixel.

Since define, applyfunction, mutate, and modify are all in S', then S' is complete. According to our procedure, since S' is complete, S is not minimal.

As a different example, we will examine a new set containing only three operations. These operations are define, modify, and applyfunction, and they are defined in the same way as above. We will call the set {define, modify, applyfunction} T.

Our first step in showing that T is minimal is to show that T is complete. As stated above, adding and removing a pixel can be performed by the define operation, and adding, removing, and modifying a channel can be performed by the applyfunction operation. Changing the value of a channel in the pixel can be performed by modify.

To change the horizontal and vertical positions of a pixel in an image, we can use a combination of all three operations. We can use define to remove the pixel from its current position,

then add it to its new position. To recreate the appropriate channels and values, we can use the modify and applyfunction operations.

Since we can perform all of the eight simple operations using the elements of T, the set T is complete. Next, we will test T - $\{O_i\}$ for completeness for values of i from 1 to 3.

Let T' = T - {define}. Since the remaining operations in T', modify and applyfunction, only edit the channels and their values, it is not possible to add or remove a pixel from the image. So, T' is not complete.

Next, we will set T' = T - {modify}. Neither define nor applyfunction can change the value of a channel in a pixel. Thus, the set T' cannot perform all of the eight simple transformations, which means that it is not complete.

Finally, we will set T' = T - {applyfunction}. Neither of the operations left in T', define and modify, can add or remove a channel from a pixel. So, like the previous cases, T' is not complete.

Summarizing, the set T = {define, modify, applyfunction} is complete. In addition, removing any operation from T creates a new set T' that is not complete. By our definition, then, the set T is minimal.

4. INDEPENDENCE

In mathematics, an axiom in a set is independent if it cannot be derived from the other axioms in the set (Smart, 1988). We define an operation as independent using the same ideas. Specifically, an operation in a set is independent if it performs a transformation that cannot be expressed by any combination of the other operations. We define the entire set to be independent if all of the operations in the set are independent of each other.

This notion of independence is related to the concept of minimality. Another way of stating that a set is minimal is stating that a set is complete, and each operation in the set is independent of the others. Thus, the method for proving that a set of image processing operations is independent is similar to the method of proving that a set is minimal. Formally, the steps for proving a set independent are:

Let S be a set $\{O_1, O_2, ..., O_M\}$, where each O_i is an image procession operation.

For i = 1 to M

Let S' = S - $\{O_i\}$.

Show $\exists$ a transformation performed by {Oi} that cannot be performed by S'

(*Otherwise S is not independent*)

We will demonstrate this procedure by testing the set of six image processing operations used earlier, namely merge, define, modify, combine, and applyfunction. To perform this procedure, we must characterize all of the possible functions of each of the operations. We will do this in terms of the eight simple operations described above.

The merge operation alters an image by combining it with a new one. This means that it can modify the base image in several ways depending on the parameters used with the operation. By merging an image with a new pixel, it can add a pixel to the base image, which means that it could add several pixels to an image. By merging a pixel in the image with another pixel at the same location, it could replace the existing pixel or simply alter the channels or values of the channels of that pixel. This means that it can add, remove, or modify channels of a pixel as well as modify its values.

Merge cannot, however, delete pixels from the base image. By the definition of the merge operation given in Gruenwald and Speegle (1996), merge only adds new pixels or changes the color of existing pixels. Although it could change the color of a pixel to match the background color, the pixel will still be defined in the image. The definition of merge also implies that it has no way of changing the horizontal and vertical positions of a pixel.

As stated earlier, the define operation transforms an image by selecting a subset or superset of the image. This implies that it can only add and delete pixels, and not change the color of a pixel, nor change the channels on which the pixel operates.

The mutate operation shifts, rotates, enlarges, or shrinks an image. So, it can change the horizontal and vertical positions of an image. When it shrinks an image, it removes pixels, and when it enlarges an image, it creates new pixels. This operation, then, could be used to add or remove pixels as well. Mutate does not affect the channels of the pixels nor their values.

The modify operation allows users to change the values of the channels in the pixels. It does not, however, affect the position of the pixels, and it does not add or remove pixels. In addition, modify cannot alter the channels used by a pixel, only its values.

The combine operation is similar to the modify operation except that it will calculate the new value of a pixel using information from its neighbors. Like modify, it does not affect the position of the pixels, the number of pixels in the image, or the channels used by the pixels.

Finally, the applyfunction operation edits the channels used by an image. It can change, add, or remove channels from some or all of the pixels. This operation, however, does not add or remove pixels, change the position of the pixels, nor change the values of the channels.

Now that we have identified the basic functions of each operation, we can determine whether each is independent. So, we will let S, our set, equal {merge, define, mutate, modify,

combine, applyfunction}, and let S' = S - {merge}, which equals to {define, mutate, modify, combine, applyfunction}.

As stated above, the merge operation can add pixels, edit the colors of the pixels, and edit their channels. These operations can be performed with the remaining operations in S'. The define operation can add pixels, the combine operation can change the color of the pixels, and the applyfunction operation can edit their channels. So, each operation that can be performed by merge can be performed by the operations in S', which means that merge is not independent.

Removing {define} from S yields the new set S' = {merge, mutate, modify, combine, applyfunction}. The define operation can add and remove pixels, and these operations can be performed by merge and mutate, respectively. Since the operations that can be performed by define can be performed by the operations in S', define is not independent.

When S' = S - {mutate}, S' becomes {merge, define, modify, combine, applyfunction}. The mutate operation adds and removes pixels as well as changes the horizontal and vertical positions of the pixels. To add and remove operations, we can use the define operation. To change the position of pixels, however, we must use a combination of the other operators.

Changing the position of a pixel can be duplicated by removing the pixel, adding a new one in the appropriate position, then setting the channels and colors. The define operation can perform the removal and addition of pixels from the necessary positions. To set the appropriate channels and colors, we can use the applyfunction and modify operations. This means that the transformations performed by the mutate operation can be duplicated by the operations in S'. So, mutate is not independent.

The only transformation that modify performs is to change the color of the pixels in an image. This can also be performed by the combine operation. When S' = S - {modify}, S' = {merge, define, mutate, combine, applyfunction}. Since combine is in S', S' can perform all of the transformations that modify can perform, and that means modify is not independent. Similarly, when S' = S - {combine}, S' = {merge, define, mutate, modify, applyfunction}. So, S' can perform all of the transformations that combine can perform, which means that combine is also not independent.

Finally, let S' = S - {applyfunction}. This means that S' = {merge, define, mutate, modify, combine}. The applyfunction operation adds, removes, or modifies the channels that an image uses. Since the merge operation can also add, remove, or modify a channel, S' can perform all of the transformations that applyfunction can. So, like the other operations in S, applyfunction is not independent.

If there were an image transformation that one of these operations, say O_i, performed that could not be duplicated by the other operations in S, then O_i would be independent. As demonstrated, this is not true for any of the operations in the example LML. Since there exists an operation in S that is not independent of the others, the entire set is *not* independent.

5. CONCLUSION AND FUTURE WORK

In the preceding sections, we have presented methods for determining if a set of image processing operations is minimal, and if it is independent. To accomplish this, we have explicitly defined minimality and independence, and what it means for a single operation to be independent of others in a set. In addition, we have shown that the Logical Model Language (LML) presented by Gruenwald and Speegle (1996) is neither minimal nor independent. Furthermore, none of the image processing operations in the LML are independent of the others. This means that to justify their inclusion in a standardized LML, we must develop different criteria.

One possible method of justification is to count the number of operations it takes to duplicate the transformations of a particular operation. This would give us a basis for measuring efficiency. Even though it may be obvious that it is more efficient to have some specific operation, we would be able to state the number of transformations saved by including it. Instead of determining when an operation is independent, we would determine what transformations cannot be duplicated in k operations, where $k \geq 0$.

In addition, we must extend our work to the other types of multimedia data listed earlier. Using similar criteria to the ones used earlier, we must develop LMLs for audio, video, and text. In addition, a prototype application must be developed to determine the space savings generated by a view-based multimedia database management system.

REFERENCES

Aberer, K., and Klas, W., 1992, "*The Impact of Multimedia Data on Database Management Systems*", International Computer Science Institute, Berkeley, California.

Bengtsson, E. et. al., 1981, "*Cello: An Interactive System for Image Analysis*", Lecture Notes in Computer Science, 109, Digital Image Processing Systems, Springer-Verlag, New York, pp. 21-45.

Bentley, J., 1986, "*Little Languages*", Communications of the ACM, Vol. 29, No. 4, August, pp. 711-721.

Brown, L., Gruenwald, L., and Speegle, G., 1997, "*Testing a Set of Image Processing Operations for Completeness*", Proceedings of the 2nd International Conference on Multimedia Information Systems, pp. 127-134.

Gonzales, R. C., and Woods, R. E., 1992, Digital Image Processing, Addison-Wesley, Reading, Massachusetts.

Grosky, W., 1994, "*Multimedia Information Systems*", IEEE Multimedia Systems, Spring, pp. 12-24.

Gruenwald, L., and Speegle, G., 1996, "*Research Issues in View-Based Multimedia Database Systems*", Proceedings of the 2nd World Conference on Integrated Design and Process Technology.

Joseph, T., and Cardenas, A. F., 1988, *"PICQUERY: A High-Level Query Language for Pictorial Database Management"*, IEEE Transactions on Software Engineering, Vol. 14, No. 5, pp. 630-638.

Khoshafian, S., and Baker, B., 1996, Multimedia and Imaging Databases, Morgan Kaufmann, San Francisco, California.

Klas, W., and Aberer, K., 1995, "*Multimedia Applications and their Implications on Database Architectures*", Advanced Course on Multimedia Databases in Perspective, University of Twente, The Netherlands.

Klette, R. and Zamperoni, P., 1996, Handbook of Image Processing Operations, John Wiley & Sons, New York.

Smart, J. R., 1988, Modern Geometries, Brooks/Cole Publishing, Pacific Grove, California.

Webb, J. A., 1992, "*Overcoming the Barriers to Architecture-Independent Image Processing*", Proceedings: Image Understanding Workshop.

Woelk, D., Kim, W., and Luther, W., 1990, *"An Object-Oriented Approach to Multimedia Data"*, Readings in Object-Oriented Systems, Morgan Kaufmann, San Mateo, California, pp. 592 - 606.