Bericht 70

M O O S E

Users' Manual
Implementation Guide
Evaluation

Steven A. Shafer

IfI-HH-B-70/80

April 1980

MOOSE

Users' Manual
Implementation Guide
Evaluation

Steven A. Shafer

16 April 1980

University of Hamburg
Fachbereich Informatik

The research that produced MOOSE was funded in part by Deutsche Forschungsgemeinschaft.

Table of Contents

1. What MOOSE Does

1.1 Segmentation

MOOSE is a program which performs a "segmentation" of an image. A segmentation, in general, is a process which attempts to abstract some two-dimensional cues from an image; hopefully, these cues are useful for some further processing. As a rule, segmentation techniques must make some assumptions about the relationship between "useful" cues (usually defined in terms of the scene domain) and the appearance of such cues in the (two-dimensional) image. Since the MOOSE program implements a segmentation procedure invented by Ohlander (section 1.3), the assumptions made by MOOSE are those of Ohlander's technique.

1.2 Ohlander's Assumptions

In Ohlander's segmentation algorithm, it is assumed that surfaces in the scene, which we desire to identify, will be represented in the image by connected sets of pixels with the same "color". Color, in this sense, may be any pixel-wise measure (such as some texture measure); usually, just spectral features are used.

It is easy to think of counter-examples to this assumption: a surface which is partly covered by a shadow from some object; two objects with similar colors whose projections in the image are adjacent; etc., etc., etc.

The above statement may be thought of as the primary assumption of Ohlander's technique; however, there are several more secondary assumptions related to the exact process used. You may imagine that these additional assumptions are related to overcoming some of the difficulties caused by the flaws in the primary assumption.

For example, each of these "patches" (connected sets of pixels) is assumed to produce a Gaussian peak when histogrammed over a spectral feature. In addition, the histogram of a collection of these patches, which therefore consists of a sum of such peaks, is assumed to clearly indicate the separations between these peaks. It is assumed that the "valleys" (relative minima) of the histogram will lie between such peaks rather than cutting peaks in the middle. In general, we believe that this assumption does not hold for images with many small patches; this may be called the "majority rule" problem, and is discussed in more detail in section 6.2.

Because Ohlander's algorithm deals with only one feature at a time, there is a further assumption that peaks which are defined in an n-dimensional histogram over all features will still be defined in at least one 1-dimensional histogram of some feature. It is clear that this cannot be true in general; however, we have no experience at Hamburg (or Carnegie-Mellon University) to contradict the usefulness of this assumption. On the contrary, the fact that we have used only black-and-white images so far in Hamburg demonstrates our faith in this assumption (to some extent), and our satisfaction with the results so far obtained confirms this. The computational simplification which comes from processing just one feature at a time seems to outweigh the cônceptual advantage of using truly n-dimensional histograms. In any event, we have no experience with this issue at Hamburg, since we have only used MOOSE with black-and-white images.

1.3 The Algorithm

The MOOSE algorithm differs slightly from Ohlander's own formulation. For a description of Ohlander's ideas, see his thesis [Ohlander 75]; for a more complete description of the algorithm used in MOOSE, see the KIWI report [Shafer and Kanade ??] which describes the same algorithm as implemented at Carnegie-Mellon University. Here follows a brief outline of the procedure.

1.3.1 The Control Structure

The basic idea of the algorithm is to split the image into regions, then split those regions into smaller regions, and so on, until only uniformly-colored or very small regions remain. The control structure is thus recursive. One way to view the segmentation is as the production of a tree in which the nodes represent regions of the image; the root node of the tree is the whole image, and the splitting step selects some terminal node of the tree, chops it into pieces, and creates a subtree of the split node which contains these new nodes. In the tree formulation, the final result of the segmentation consists of the collection of all the terminal nodes. In the MOOSE program, the terminal nodes are kept in a queue; the splitting step pulls the first node off of the queue, splits it into pieces, and puts these new nodes at the tail of the queue; this accomplishes a breadth-first generation of the tree.

The following paragraphs define in more detail how the splitting is accomplished for a single region.

1.3.2 Fetching a Region

First, some region must be selected from the head of the queue. There is a size criterion used to ensure that very small regions are not split further. The result of this step is either the selection of a region for segmentation; or the realization that no regions remain to be segmented (i.e. the segmentation is complete).

1.3.3 Histogramming

Each feature being used is histogrammed over the region selected. The result is a collection of histograms, one for each feature.

1.3.4 Peak Selection

Each histogram is analyzed to determine some good way to threshold the corresponding feature. Although this is called "peak selection", it is really a process of examining the relative minima (valleys) of the histogram, and might therefore be thought of as "valley selection". This analysis consists of several heuristic tests designed to identify some valley (or valleys) as potentially good thresholds. When all the features have been analyzed, each one is assigned some numeric score; the scores are compared, and one or more features may be selected as the "good" features for further processing. If no features qualify as good features, the region is declared terminal and segmentation of that region does not occur.

1.3.5 Thresholding

For each candidate (good) feature, the region is thresholded at the thresholds indicated by the peak selection for that feature. The result is a subimage containing the region, in which each pixel represents the "interval number" of that pixel according to the thresholds used. Thus, pixels below the value of the first threshold are in interval number 1, etc. This step is applied once for each candidate feature; it results in this subimage, called the "interval map".

1.3.6 Connected Patch Finding

This step is applied once for each interval of each candidate feature. It consists of a fast algorithm which scans the interval map, noting which pixels are within the current interval and which are not, and produces two descriptions of the pixels. The first description (the "patch" description), indicates, for each 4-connected patch of pixels in the interval, the MBR (minimum bounding rectangle) of the patch, its area, the number of holes in the patch and their area, the location of the centroid. A similar description is created for each 8-connected patch of pixels not in the interval. The other form of pixel description is the "run" list; a list of all the runs (adjacent pixels on one image row) which are in, or not in, the current interval. The runs for each patch form a linked list; each patch record contains a pointer to the list of runs for that patch. The results of this step are, for each interval of each candidate feature, a list of patches and a linked list of runs.

1.3.7 Noise Elimination

Thresholding features usually creates some amount of "noise" -- very small connected patches of pixels whose label does not seem to us to be appropriate. We may have thresholded a textured area or some pixels which have been subjected to digitization errors, for example. Usually, this means that we think the pixels should have been labelled in some other way; perhaps the same way as the patch surrounding the noise patch. Accordingly, in this step, we examine all the patch descriptions looking for noise patches. When a patch is identified (according to some criterion, such as size) as a noise patch, it is marked as being a noise patch and, in the future, we will consider it to be just a part of the patch which contains it. This step is applied once for each interval of each candidate feature; it just examines all the patch descriptions and indicates which ones are noise patches.

1.3.8 Feature Selection

Now, we have examined all candidate features; we have seen how many patches result from the selection of any of these features and how big they are; we are in a position to select the best of these features. This step is applied once for the region, after all features have been thresholded and examined for connected patches and for noise. The output from this step is simply one feature which has been selected as the best based upon some heuristic examination of the features. It is possible that no feature is considered acceptable; in this case, the region is declared to be terminal and segmentation of this region will not occur.

1.3.9 Region Collection

We have now selected some one feature as the best for segmenting the current region; we know which patches result from using this feature; we know all the runs of pixels which compose these patches. In this step, new region records are created for these patches, the "region map" is updated to indicate which pixels comprise all of these regions, and the new regions are added to the end of the queue of unexamined regions. After this, segmentation of the fetched region is complete, and the cycle begins again.

1.4 References

There are several previously published works which are relevant to the MOOSE program.

The idea of selecting thresholds from an examination of histograms is attributed to Judith Prewitt [Prewitt 70]. She, however, only used monochromatic (black-and-white) images in her work.

The idea of using several spectral features, histogramming each independently and selecting one feature as the "best", originated with Ohlander [Ohlander 75]. Since MOOSE follows this very idea, we refer to this algorithm as "Ohlander's algorithm".

Keith Price [Price 76] modified this algorithm by adding new techniques for dealing with texture, new kinds of image smoothing, and a sort of "planning" in which a reduced version of the image is segmented to provide guidance for segmenting the original picture. Since MOOSE does not make use of these enhancements to Ohlander's work, we view MOOSE as a parallel development with Price's ideas rather than as a direct descendant.

Ohlander and Price have summarized their work in an article [Ohlander et al. 78] which also presents some retrospective comments not present in their theses.

MOOSE does, however, have a direct precedent: the "KIWI" program [Shafer and Kanade ??] written at Carnegie-Mellon University. The author of MOOSE (and of this paper), as the implementor of KIWI, is in a position to draw direct comparisons; these appear in sections 5 and 6 of this paper.

Certain aspects of the MOOSE algorithm are related to phenomena described in other publications.

The connected patch extraction algorithm is due to Agrawala and Kulkarni [Agrawala and Kulkarni 77], although the formulation of the control and data structures differs slightly from theirs.

Dave Milgram [Milgram 77] used the relationship between threshold values and edges of the thresholded regions in his work; that relationship is related to some problems encountered in MOOSE.

The importance of various spectral features, and combinations of them, will be described in [Ohta et al. ??]. This analysis relates to essentially the same algorithm as that of Ohlander.

Finally, the idea of a "difference picture" describing differences between corresponding pixels of successive frames from an image sequence was developed by Jain et al. [Jain et al. 77]. This is currently used only for producing interesting displays in MOOSE; we hope, however, to make more use of this information in the future.

2. How to Use MOOSE

2.1 How to Run MOOSE on the MINCAL

To run MOOSE on the MINCAL, first turn on the MINCAL and start up the disks. You'll need to have both MINCAL terminals available, since MOOSE uses the alternate terminal. At the present time, MOOSE sits on the system (fixed) disk; so, it is probably best to put your data on the removable disk. You'll need to type "COD,SHA" so you can pretend to be me. When you're ready, just type "MOOSE" to the monitor, then go sit down at the alternate terminal and presto! magic will occur. See section 2.3 below for the next step.

2.2 How to Run MOOSE on the PDP-10

If you want to run MOOSE at super-high-speed, just configure it for the PDP-10 and compile it (see section 3 for details), and type "RU MOOSE" to the PDP-10 monitor. Alakazam! Magic will occur very quickly.

2.3 Messages on the Terminal

Needless to say, MOOSE wants to ask you a few questions before getting down to business. As a general rule, a message like this:

```
    (F)IRST CHOICE OR (S)ECOND CHOICE?  [F]
```

means that you must select one of the two choices. If you type "F-carriage return" you will choose the first; "S-carriage return" chooses the second. The "F" in square brackets at the end means that you can just type "carriage return" and this is the same as typing "F-carriage return". Typing anything else is also interpreted as the option in square brackets. Similarly, some yes-or-no questions look like this:

```
    DO YOU WANT MUSTARD ON YOUR APPLE?  [YES]
```

If you type "Y-carriage return" or "carriage return", you are answering "yes"; if you type "N-carriage return", you are answering "no". Again, typing anything else is the same as the option in square brackets.

2.3.1 Selecting an Image

The first thing MOOSE asks you is

```
    IMAGE NAME =
```

and you are supposed to type the file name of the image you want to segment. When segmentation of that image is complete, MOOSE makes a (very intelligent) guess as to the name of the next image in the sequence, and asks you if you want to process it. You answer with yes or no, and MOOSE does the right thing(!).

It is possible that several features exist for a single image. In such a case, MOOSE assumes that they all have names which are the same, except for a single letter which distinguishes the features (for example: "BBR01R", "BBR01G", "BBR01B"). To segment this image, you should type "BBR01#" to MOOSE, where "#" indicates the changing letter; then, MOOSE will construct the proper file names as needed.

After you have indicated the image to be segmented, MOOSE asks what features you want to use. Indicate each feature by a single letter, for example "D" for density (intensity), or "RGB" for red, green, and blue.

2.3.2 Configuring MOOSE

When you have selected an image, MOOSE wants to know exactly what to do with it. You may have several things to describe or choose.

If this image is not the first one of the sequence, you will have to choose whether you want a normal segmentation or region-matching. You probably want normal segmentation; see a description of region-matching elsewhere.

You will have to describe the display you want to see if you are running MOOSE using the COMTAL. See section 2.4 for details.

You will have to describe the things you want to have listed on the line printer. See section 2.5 for details.

You will have to tell MOOSE whether this image is full-sized (574 X 512) or a geo-picture (96 X 128). If it is full-sized, you can choose whether to reduce it (to 96 X 128) or crop some portion of it (no larger than 96 X 128) for processing. Reduction is performed by selecting the median from each 4 X 6 window; cropping is described below along with other display information. You will also have to select either full-frame or half-frame; when an image contains moving objects, the two half-frames (even and odd rows) will contain the object in differing positions, and segmenting one half-frame is probably desirable.

2.3.3 Progress of the Segmentation

As the segmentation proceeds, various things are printed on the tty to tell you what's happening. Each message is related to some step of the segmentation algorithm.

```
IGNORING REGIONS ...
```

Indicates that these regions were not selected for segmentation because they did not meet the necessary criteria (i.e. were too small).

```
SEGMENTING REGION ...
```

Tells you which region has been selected for segmentation.

```
HISTOGRAMMING ...
```

Reports the progress of the histogramming step.

```
PEAK SELECTION FOR A:P:S B:Q:T ...
```

Reports the progress of peak selection. A and B are features selected; P and Q represent the number of peaks for the respective features' histograms; S and T are the scores assigned to the features.

```
GOOD FEATURES ARE ...
NO GOOD FEATURES
```

one of these messages will appear after peak selection to tell you which features (if any) were selected as candidates for segmentation.

```
FEATURE X INTERVAL ABC...
```

The message "FEATURE X" appears when feature X is thresholded; as its intervals (A,B,...) are examined for connected patches and for noise, they are indicated in the message.

```
BEST FEATURE IS X
NO FEATURES ACCEPTABLE
```

One of these messages will appear during feature selection (after all features are examined for connected patches and noise), to tell you which feature (if any) has been selected for segmentation of the current region.

```
REGIONS A TO B CREATED
```

This message appears after region collection to tell you the newly created regions.

```
END OF DEPTH X
```

This appears during region-fetching to tell you that all regions at a certain depth in the segmentation tree have been examined.

```
SEGMENTATION COMPLETE
```

This tells you that the segmentation of the current image has ended.

```
SEGMENTATION TIME...
```

A chart is produced telling how many minutes and seconds of time were consumed for each step of the segmentation. The total time includes everything except the initial reading of the image data.

2.4 Color Displays

When you run MOOSE on the MINCAL you may enjoy gawking at the pretty color displays it can produce on the COMTAL. The image itself will always be shown in black-and-white; other stuff will be in color. There are three different types of display produced by MOOSE.

2.4.1 Cropping the Image

When you segment a full-size image by cropping some portion of it, you have to tell MOOSE what part you want to segment. MOOSE will then display the whole image on the COMTAL screen, and you must use the trackball and switches to indicate part of the image. Here's what you do: first, turn on the top switch. This tells MOOSE that you're ready. Then, use the trackball to move the little white dot over to some interesting part of the image. When you get there, turn on the bottom switch. This tells MOOSE that you want to include the point you're pointing at. MOOSE puts a green dot on the screen. As you move the trackball, the white dot moves as well, and MOOSE follows it putting little green dots on the screen. You are supposed to indicate the area you want to segment; MOOSE uses the MBR (minimum bounding rectangle) of all the green dots as the area to process. If you indicate too big an area, MOOSE beeps at you, puts a little red dot on the screen, and writes "too big" on the tty for you. It doesn't matter, just keep on going and remember that only the little green dots are important to MOOSE. When you are finished, turn off both switches. You only need to turn off the top one, but you'll be sorry unless you turn them both off. MOOSE will then ask you if the area you have encircled is ok; if you say "no" you get to try it again. When you tell MOOSE that it's ok, then it believes you and starts to segment.

2.4.2 Displays Showing the Progress of MOOSE

As the segmentation proceeds, lots of things may happen on the screen. You select what you want to see during the "configuration" of MOOSE. You can use the two switches on the trackball during the segmentation as well; the top switch makes the program stop after it finishes the current display (it beeps at you and flashes a cute message until you turn off the switch so it can proceed), and the bottom switch selects a "fast display". This fast display is a subset of the normal display which you have selected during configuration; after asking you what to include in the normal display, MOOSE will allow you to specify exactly what should be included in the fast display.

DISPLAY OF IMAGE WITH REGION BOUNDARIES

This puts the image in the upper left corner of the screen, and outlines all the regions in yellow as they are created. In addition, whenever a region is declared to be "terminal", a little yellow dot appears at its centroid.

DISPLAY OF IMAGE WITH NO REGION BOUNDARIES

Since outlining regions in yellow takes a long time, you may want to just show the image and not bother with the regions.

DISPLAY OF REGION SELECTED FOR SEGMENTATION

This shows the currently selected region in the lower left corner of the screen, blown up as big as possible, and outlined in yellow.

DISPLAY OF PEAK SELECTION

This uses another image buffer, so it's distinct from the displays described above and below. It consists of a picture of the histogram, with green lines indicating all potential threshold values. As the peak selection progresses, it tells you what it's doing and changes the green lines to reflect it's current feeling about the thresholds. This is real cute but takes a long time.

DISPLAY OF THRESHOLDS SELECTED

This is back on the main display; it puts a histogram in the upper right corner of the screen, with green lines indicating the thresholds selected. Below the histogram are two gray-scales: the first is simply a true gray-scale showing the pixel values represented by the appropriate columns of the histogram; the second uses a constant value for each interval, and indicates the grey values used for displaying the patches (see following paragraph).

DISPLAY OF PATCHES AFTER THRESHOLDING

This display puts a map of the patches in the lower right corner of the screen; the patches for each interval are shown in a constant gray value, and they're all outlined in green. The green outlines appear on the region in the lower left corner as well, so you can see how the patches really look in the image.

DISPLAY OF NOISE ELIMINATION RESULTS

After noise elimination, some patches are considered to be meaningful (i.e. not noise). These patches are marked (in both the lower left and lower right corners) by putting a little green dot at their centroids.

DISPLAY OF FINAL RESULTS OF SEGMENTATION

This is the final display (see next section for details).

2.4.3 The Final Display

When the segmentation is complete, if you have selected to see the final display, MOOSE gives you a truly impressive display of the results. This display has five parts:

(I)MAGE

The image itself, blown up to 384 X 512, in black-and-white.

(O)UTLINES

The outlines of the regions created, shown in yellow. Little noise regions are also outlined.

(L)ABELS

The region number for each region, written in purple digits at the centroid of the region.

(R)EFERENCE

The outlines of the regions in the preceding frame, shown in red.

(D)IFFERENCE

The "difference picture" indicating the significantly different pixels between the current frame and the previous frame; shown in black-and-white, where black means no difference and white means a significant difference [Jain et al. 77].

For various reasons, the "labels" and "reference" displays may not be seen at the same time; similarly, the "image" and "difference" displays are exclusive. In addition, the "reference" and "difference" displays are only meaningful for the second image you segment and all following images.

You get to turn these various displays on and off with the tty. A message appears something like this:

```
(I)MAGE OFF, (O)UTLINES OFF, (R)EFERENCE ON, ..., (E)XIT?
```

and you can type letters to indicate what you want to do. For example, typing "IO" would turn the image and outlines displays off; "OR" would turn the outlines off and the reference display on; "I" would just turn the image off; "E" would exit from the final display (and ask if you want to segment the next image...).

One caveat about the final display: you may think you can achieve similar effects by using the switches on the COMTAL front panel, but you'd be only partly correct. Take my word for it, using the tty will make you happier.

2.5 Producing Listings

You can get listings from MOOSE on either the MINCAL or the PDP-10. On the MINCAL, you have to have the printer hooked up the proper machine and turned on; on the PDP-10, a file called "??????.LPT" will be created which will be printed automatically when you log off. The various options for the listing are selected when you configure MOOSE.

REGION MAP

The region map indicates each pixel with a character telling which region contains that pixel. Above the map is a small scale showing which characters indicate which regions. The listing is about two pages, and is recommended whenever you get a listing.

SEGMENTATION TREE

This listing includes one line for each region and peak in the final segmentation tree, showing the structure of the tree. This is also about two pages, and is also recommended at all times.

MATCH DATA

This is only meaningful when you are performing region matching, and includes two parts: a table of match values during the matching process, and a picture of each region and its correspondng reference region. Useful for understanding what the matching is all about. Runs to many pages.

REGION PICTURES

This includes a drawing of each region resulting from the segmentation. The drawing indicates with an "X" each pixel in the region which is not included in any subregion, and with

"-" each pixel in the region which is included in some subregion. This may be many pages long.

REGION DESCRIPTIONS

If you get the region pictures or match data, this is automatically included; otherwise, you may choose it if you want. The region descriptions is simply a description of the parameters known for each region -- its position in the tree, and its shape (area, MBR, centroid, number of holes, etc.). Strongly recommended at all times.

SEGMENTATION TIME

The same chart that appears on the tty after segmentation is complete -- about one-half page.

PATCHES

This display is a dump of all patches which result from the connected patch finding algorithm. Many pages long; recommended only for debugging the program or if you need new wallpaper.

3. Implementation Notes

3.1 Program Organization

These are the (SOS) pages of the program and their contents:

1	introductory comments
2	comments about the data structures
3	constant declarations
4	constants for COMTAL library (etc.)
5	types -- subranges and scalars
6	types -- records and arrays
7	types for COMTAL library (etc.)
8	variable declarations
9-10	initprocedures
11-20	COMTAL library procedures and utility procedures
21-22	listing and color display procedures
23-26	initialization procedures
27-28	fetching a region
29-30	histogramming
31-33	peak selection
	32 = comments explaining algorithm
34-35	finding matching intervals (region-matching)
36-37	thresholding
38-40	connected patch finding
	39 = comments explaining algorithm
41-42	noise elimination
43-44	feature selection
45-46	region collection
47-48	region matching
49-50	saving reference frame
51-52	main program

3.2 Explanations of Data Structures and Algorithms

The primary data structures are described in the program, on page 2 of the code. The two tricky algorithms are peak selection and connected patch finding, described in the code on pages 32 and 39, respectively.

There are two data values not mentioned specifically in the code, but which may be important in the future. They are "IMROFFSET" and "IMCOFFSET", the coordinates of the upper left-hand pixel of the subimage segmented. These are always 1 for geo-pictures or reduced images; however, when a full-sized image is cropped, they may be different. Inside the program all row and column indices are relative to these values; so, row 1 actually refers to row (1+IMROFFSET-1) of the image; row 2 refers to row (2+IMROFFSET-1), etc. In producing listings or registering sequential images segmented over different areas, it may be necessary to account for these two values.

3.3 Arbitrary Constants

Here follows a description of the constants randomly thrown into MOOSE, and how they might be changed. The actual heuristics used in the program are described in section 4, and are not mentioned specifically here. All of these constants appear on page 3 of the source code.

3.3.1 Implementation Limits

PIX_MAX

This is the largest gray-value allowed in a pixel (e.g. 255).

ROW_MAX, COL_MAX

These are the parameters which define the maximum area which can be segmented. If they are changed, then the initialization for geo-pictures and reduced images will have to be changed so that the proper dimensions are noted when the image is read.

TVROW_MAX, TVCOL_MAX

These values define the size of a full-sized TV frame.

REDUCE_ROWS, REDUCE_COLS

These define the size of the window used for reducing a full-sized frame to fit within the maximum allowed area.

AREA_MAX

This is simply ROW_MAX times COL_MAX; the largest area value.

FEAT_MAX

Tells how many features are allowed for each image.

CAND_MAX

Tells how many features may be selected as candidates during peak selection.

REG_MAX

Tells how many regions may result from segmentation. If this value is changed to above 255, then several data records will become larger and more core will be used at run-time.

PEAK_MAX

Tells how many peaks may result from segmentation. The same size considerations apply here as for REG_MAX.

PAT_MAX

How many patches may be present at a given time. This includes all patches from all intervals of all candidate features for a single region. This value should probably be at least 50 times CAND_MAX times INT_MAX.

RU_MAX

How many runs may be in the run list at once. Should probably be at least 6 times PAT_MAX.

PREG_MAX

The largest number of partial regions that can exist at once. This only pertains to a single interval of a single feature, so the value is independent of all other parameters. Normally, 50 is plenty. However, in bad cases, several hundred partial regions may exist at once. So, a value of 200 to 400 is recommended.

INT_MAX

The largest number of intervals into which a single feature may be divided. For safety, 2 is a good value (i.e. only one threshold per feature); however, just to be interesting you may try 3 or even more.

3.3.2 Flag Values

These are used only to indicate special situations. For example, the number of runs is "RU_MAX", so a run-number need only take on the values "0..RU_MAX". However, since runs form a linked list, some sort of "nil" value is needed. So, "RU_NIL" is defined to be -1; then the type "RU-LINK" is defined as "RU-NIL..RU-MAX".

3.3.3 Constants Used in Heuristics

K_SPLITSIZE, K_DEPTHMAX

These are used in region-fetching, to determine if a region can be segmented. K_SPLITSIZE is the more important of these; it should usually be about 1% of the image size. K_DEPTHMAX is used for achieving special effects, for example, only segmenting one or two levels of the tree and then stopping.

**K_MAXMIN, K_HEIGHT, K_RELAREA, K_ABSAREA, K_ABSMIN,
K_ABSSCORE, K_RELSCORE**

These parameters are used in peak selection. They pertain to histogram characteristics, and are (except for K_ABSAREA) therefore relatively insensitive to the image size itself. Tuning these parameters may produce dramatic differences in the resulting segmentation.

K_NOISE

This determines how small a "noise" region is; about 0.1% of the image is a reasonable value.

K_SPLITAREA

This determines how much of a region's area must be preserved (i.e. not lost to noise) for a feature to be considered acceptable. It is relative to the size of the region; therefore it is somewhat insensitive to the size of the image itself.

3.4 How to Compile MOOSE for the MINCAL and the PDP-10

3.4.1 Differences in the Source Code

There is only one change which must be made in the source code for switching between the two machines: the declarations of COMTAL_IO and COMTAL_ERROR in the COMTAL library. These definitions must be enclosed in comments for the MINCAL; they must not be within comments for the PDP-10. Lines 800 and 1400 of the source code on the appropriate page are comment brackets: they should contain "(*" and "*)" for the MINCAL, "(* *)" and "(* *)" for the PDP-10.

3.4.2 Compiling for the PDP-10

This sequence of typing compiles MOOSE for the PDP-10:

```
.R PASCAL
OBJECT CODE =   MOOSE.REL
LISTING =
SOURCE CODE =   MOOSE.PAS
.LOAD MOOSE
.SAVE MOOSE
```

3.4.3 Compiling for the MINCAL

You must type this to compile MOOSE for the MINCAL:

```
.RU PASMIN [306,100]
*
SOURCE CODE =    MOOSE.PAS
LISTING =
OBJECT CODE =
.RU MINLOAD [306,100] 100
*
```

This all takes a while, so bring a newspaper if you can.

3.5 Caveat Implementor!

Here are some particular pitfalls to watch out for in changing the code.

3.5.1 Integer v. Subranges of Integer

In general, you may use subranges for row number, column number, region number, etc.; In fact, this is done in most of the program. However, there is one particular case in which such a substitution cannot be safely made: RDESC_RCENT and RDESC_CCENT, the position of the centroid of a region. The reason for this is that the procedure "DOCONNECT", which finds connected patches, uses these fields as accumulators for the sum of all row numbers (taken over all pixels in the patch) and all column numbers, respectively. These sums may be approximately (area of the image) * (maximum row number) / 2 (respectively, maximum column number); clearly, too large to fit within a value declared as "0..ROW_MAX".

3.5.2 Changing the Maximum Image Size

To change the maximum image size allowed, you must change several things. First, the parameters "ROW_MAX" and "COL_MAX" must be changed. Then, "REDUCE_ROWS" and "REDUCE_COLS" need to be changed. "AREA_MAX" and various of the "K_..." constants will need to be updated. For initializing a geo-picture, see lines 350-360 of the page containing the "CONFIGURE" procedure. Last, all displays (including line-printer listings) may need to be changed, since the display code is highly dependent upon the maximum allowed image size.

3.5.3 Storage Allocation on the Heap

MOOSE allocates almost all heap data structures at the beginning of the program. So, if it successfully initializes, you're home free. There is only one exception to this: during the listing procedures which involve a region-map, an image-sized data structure is allocated and later freed in the same procedure.

3.5.4 Compiler Bugs

Oh, there are several of these.

One of the few worth mentioning here pertains to the clock; statements of the form:

```
IF CLOCK + A < B + C THEN ...
```

are liable to be erroneous; you have to do this instead:

```
X := CLOCK + A; IF X < B + C THEN ...
```

The most serious bug pertains to this situation: a with statement for a record which is an element of an array on the heap. Here is the bad situation:

```
TYPE
  FOO = RECORD ... FOOFIELD:INTEGER ... END;
  FOOINDEX = 0..100;
  FOOARRAY = ARRAY [FOOINDEX] OF FOO;
  FOOPTR = ^ FOOARRAY;
VAR
  P: FOOPTR;
...
PROCEDURE USEFOO;
  BEGIN
    WITH P^[X] DO BEGIN
      ... FOOFIELD := FOOFIELD + 2; ...
    END;
  END;
```

This causes no end of problems. The solution is simple but slows down the program:

```
PROCEDURE USEFOO;
  VAR
    F:  FOO;
  BEGIN
    F := P^[X];
    WITH F DO BEGIN
      ... FOOFIELD := FOOFIELD + 1; ...
    END;
    P^[X] := F;
  END;
```

Usually, you can skip either "F := ..." or "... := F"; if you need to do both, you may consider just avoiding the with statement and writing:

```
P^[X].FOOFIELD := P^[X].FOOFIELD + 1;
```

Sorry about this, but that's life.

Another bug concerns the "DIV" operator; a statement like this:

```
    A := (A + (B DIV 2)) DIV B;
```

unfortunately causes incorrect code. You may recognize this as the simple idea of "A DIV B, rounding to the nearest integer". The solution is to break it into pieces:

```
    X := B DIV 2;   X := X + A;   A := X DIV B;
```

Note that the above bugs exist only in the "PASMIN" compiler, and not on the PDP-10. I don't know many systematic bugs on the PDP-10.

(Note: The bugs in the PASMIN compiler have been removed in the meantime)

4. Heuristics Used in MOOSE

4.1 Fetching a Region

In fetching a region, the decision must be made for each region: "Can this region be segmented?" This question is answered by examining two parameters in MOOSE: K_SPLITSIZE and K_DEPTHMAX. K_SPLITSIZE is the minimum number of pixels which a region must contain in order to be segmentable. K_DEPTHMAX is used mainly to control the depth of the tree; regions deeper than K_DEPTHMAX will not be segmented.

4.2 Peak Selection

The peak selection process uses seven constants. The entire procedure, including the use of these constants, is described in comments in the code; this description is not repeated here.

4.3 Noise Region Elimination

The noise elimination procedure considers each interval of each candidate to define a binary image: 1's for pixels in the interval, and 0's for all other pixels. The connected patches are extracted before noise elimination -- 4-connected patches of 1's and 8-connected patches of 0's.

A "noise" patch is one whose area is smaller than K_NOISE. Such a patch is marked with the "melted" status, indicating that this patch is conceptually to be "melted" into the patch containing it. Thus an isolated 0 or group of 0's will be merged with the surrounding patch of 1's; or a group of 1's will be merged into the surrounding 0's.

Note that some patches may be lost forever: a patch from interval a which lies at the edge of the region being segmented will be considered "noise" and ignored for interval a; since it does not indicate a hole in a patch in interval b, it will not be included with the adjacent patch from interval b. Thus, the small patch will be simply "lost to noise".

4.4 Feature Selection

When all features have been analyzed for connected patches and for noise, the features are compared against each other to determine which candidate is the best. A feature must, however, fulfill two criteria to be considered acceptable: there must be at least one non-noise patch remaining for each interval; and the total number of pixels lost to "noise" patches must not exceed some fraction of the original area. This fraction, expressed as a percentage and complemented (i.e. 100 - X), is K_SPLITAREA.

Each acceptable feature is assigned a score which is the average area of the resulting regions; the feature with the largest score is selected as the best feature.

5. Comparing MOOSE and KIWI

5.1 Heuristics

The MOOSE algorithm is the same as that of KIWI, with some exceptions. Two particular heuristics have been improved in MOOSE: the peak selection algorithm, and the feature selection criteria.

The peak selection process is similar in KIWI and MOOSE, but certain of the tests (the peak and valley tests) have been modified to make them relative (i.e. between the possible thresholds) rather than absolute (i.e. comparing possible thresholds to constants). The new test were slightly re-ordered. The results seem somewhat better with the new tests, but they continue to produce some unpleasant artifacts.

The feature selection criteria of KIWI demand that a feature have at least one region result from each interval. However, KIWI has no criterion relating to the acceptable amount of noise. Such a criterion has been introduced in MOOSE, and seems to work somewhat well against degenerate histograms (as identified by Price [Price 76] and by Shafer and Kanade [Shafer and Kanade ??]. There are still cases not dealt with by this criterion, however; if a noise-cleaning post-processing step is implemented, then this heuristic may hopefully be eliminated.

5.2 Features Provided by KIWI and MOOSE

There are several differences in the features of the two segmentation programs.

KIWI allows the processing of images of any size, for up to 20 features. MOOSE allows only a single size (and other sizes if appropriately reduced to that single size) to be processed. The price paid by KIWI for this is the need for a sophisticated pixel access mechanism, and the fact that all images must reside on disk throughout the processing, except for buffers in core.

Kiwi allows all parameters to be specified by a file as well as by the user; the program can be run by another program if desired, and that other program has complete freedom to configure KIWI. MOOSE, however, requires that a user be present to answer questions on the tty.

KIWI allows the segmentation of an image to be interrupted at any point, and only needs to backtrack to the last step of the segmentation. The user can then change parameters, look at partial results, etc., and restart the segmentation at any point. If MOOSE is interrupted by the user, the segmentation must begin again. Note that it is important to be able to interrupt the segmentation in KIWI, since large images (e.g. 600 X 800) may require a very long time to process; in MOOSE, however, since the segmentation time is always short, it is no hardship to begin again. The ability to interrupt KIWI contributes to its usefulness in analyzing the partial results of the segmentation and for investigating alternatives; these are not part of the design criteria of MOOSE.

KIWI allows the maintenance of a "database" of segmentations of the same image, for ease of comparing alternative parameters or processes or feature sets; MOOSE keeps no data from one segmentation to the next. KIWI also remembers the segmentation parameters on successive runs.

KIWI can produce printer-ready hard-copy of histograms, thresholded images, segmentation results, etc.; MOOSE produces only line-printer listings which are useful for debugging but not for publishing. Images and histograms produced by KIWI may be scaled to occupy a certain size on the printed page, as well.

MOOSE, however, is capable of segmenting several images from a sequence. This is one of the most important facilities of MOOSE. KIWI is not capable of successively segmenting several images. MOOSE includes some facilities for comparing successive segmentations, and even for using one segmentation to guide the next.

MOOSE can produce quite sophisticated color displays, showing all steps of the segmentation as well as displaying the final results in some detail. KIWI produces only simple color displays; although they can illustrate the processes as they occur (MOOSE shows only results which have been completely computed), they have little detail and are only moderately useful in understanding the segmentation process. This is partly due to the fact that KIWI can process images of any size (as noted before, the displays in MOOSE are very sensitive to the image size), and partly due to the poor resolution of the color display used by KIWI. An additional factor influencing the display question is that MOOSE keeps all data structures in core, while KIWI consists of separate programs which communicate via data structures on disk. Since sophisticated displays frequently require access to many different data structures at the same moment, it would be very difficult for KIWI to match the quality of the displays produced by MOOSE. However, KIWI does produce color displays of color images, while MOOSE is faced with the simpler task of producing black-and-white displays.

5.3 Execution Speed

5.3.1 Comparison of Speeds

The KIWI program runs on a PDP-11 with a megabyte of core. The machine runs a time-sharing operating system with no virtual memory facility. MOOSE runs on two machines: a PDP-10 and the MINCAL X2 (similar to a PDP-11) with 372k bytes of memory. The pascal run-time system on the MINCAL provides for virtual memory for object code; since the machine does not run a time-sharing operating system during execution of pascal programs, the entire memory is available for data structures.

On a PDP-11, KIWI requires approximately 20 minutes to segment a 110 X 120 image (13200 pixels). On the MINCAL, MOOSE requires approximately 90 seconds for a 96 X 128 image; on the PDP-10, MOOSE requires about 30 seconds for the same size image. This represents about a factor of 13 speed-up over KIWI.

5.3.2 Analysis of Speedup

Several factors contribute to this increase of speed.

KIWI spends about 40% of its time switching between programs, because of the lack of virtual memory. MOOSE, with demand-paging of object code, spends only disk-read time to fetch new code. Since most of the code fits within one core-image on the MINCAL, there is not too much time spent swapping pages.

KIWI is allowed to process any image size. The system which implements this requires approximately 40 microseconds for an average pixel access, including all array indexing, page swapping, and bit-operations. MOOSE only operates on a single image size; the image is kept in core, and a few microseconds suffice to fetch a pixel. Since pixel operations account for perhaps 1/3 of the execution time of KIWI, this represents approximately a 30% saving of time.

KIWI has been used to process images with 6 features; MOOSE has only processed one feature. This probably accounts for a factor of at least 3 in the speed.

In summary, for 100 operations required in KIWI: 40 are program switching not present in MOOSE; 33 are pixel accesses which require about 3 operations in MOOSE; this results in about 30 operations in MOOSE for 100 in KIWI. Considering the factor of 3 due to the use of a smaller number of features, we might easily expect that 10 operations of MOOSE correspond to 100 of KIWI: a factor of 10 speedup. This is quite close to the observed factor of 13; since all of the above estimates are very rough, we have probably accounted for all of the important factors in the speed difference.

It should also be noted that running MOOSE with all displays requires between 20 and 25 minutes for a 96 X 128 image.

6. Insight Gained from MOOSE

6.1 How MOOSE has Provided Some Insight

Several things have been learned about the segmentation procedure through the construction and use of MOOSE. These things are primarily attributable to two factors: the quality of the displays produced by MOOSE, and the existence of the image-cropping capability of MOOSE. Both of these circumstances helped to bring about an increased awareness of some of the problems and promise of Ohlander's segmentation algorithm, as implemented in KIWI and MOOSE.

6.2 Majority Rule

The most dramatic insight gained by MOOSE was a problem which might be called the "majority rule" problem. It arises when there are very large regions and small regions together in the image.

The problem has two parts which are somewhat complementary. Consider the histogram of two very large regions, with clearly defined peaks and a low valley separating them (Figure 6-1). The ideal spot for a threshold is clear. Now, imagine a small region whose grey-values lie just between the large regions, but a little bit to the left of the original valley. The small region will blend with the peak on the left, but be partially separated from the peak on the right; the valley will thus be moved perhaps several pixel values to the right. This is probably ok as far as segmenting the small region is concerned; but, it will chop some area off of the large region whose peak lies to the right. So, one of the large regions will not be optimally thresholded. In fact, this problem will create noise areas along the edges of the region, in general, which is difficult to deal with (see section 6.3).

Now, imagine a somewhat smaller region histogrammed along with the two large regions, and imagine that its center lies just in the deep point of the original valley. Since this is a (relatively) very small region whose peak is symmetric with respect to the original valley, the presence of this region will probably not significantly disturb the location of the threshold. As a result, the peak of this small region will be cut right in half, and the small region will almost assuredly be riddled with noise after thresholding. This phenomenon leads to the name "majority rule" -- it is the large regions which dominate the choice of a threshold, and small regions may be cut up almost randomly. It is one of our most dramatic observations with MOOSE that cropping a large image around a small object, even in a very approximate way, yields a fairly reasonable segmentation of the object, while processing the whole image may find little if any of the structure of the small object. This is even true when the small object occupies say, 5% of the image area -- quite a large part of the image for a single

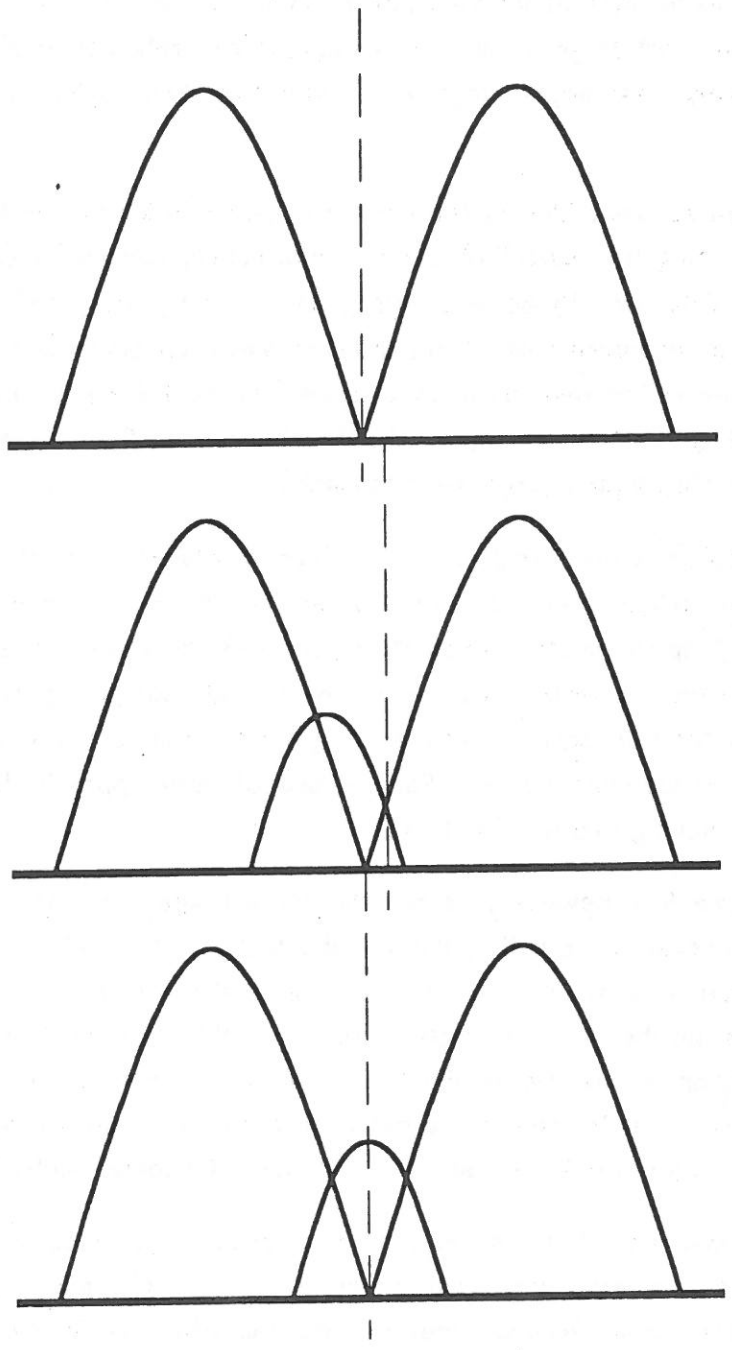

Figure 6-1: Histograms of two and three regions

object in a complex scene.

The problems mentioned above are actually more severe than the discussion has indicated. Note, for example, that almost all of the above observations will hold true when the two large regions are not large at all, but merely demonstrate noticeable peaks in the histogram. It seems likely that any histogram of more than two regions at once may suffer from this problem.

As seen above, reducing the number of peaks in a single histogram should have the effect of highlighting the important spectral information, allowing us to find significant objects more readily. One way to achieve this is by cropping the image around suspected objects of interest, as in figure 6-2. Here, an image was cropped around an interesting automobile; this is reflected in the histogram, which shows the peak for the auto on the left and that for the surrounding road on the right. The thresholded result, also seen in that figure, demonstrates a reasonable boundary for the automobile.

A problem with cropping is that some detailed knowledge of "interesting" objects is necessary in advance. Is this really required, or is the process robust enough to deal with extremely approximate cropping? Figure 6-3 shows the same image, but the cropped area includes other automobiles and much more road. Comparing this with figure 6-2 reveals that the peak for the large car in the foreground is still readily visible, and that thresholding still yields a reasonable result. So, the use of very approximate cropping has not hurt the decision-making process too badly.

In figure 6-4, however, we see the whole image processed. The image has been reduced for this procedure, but that should not affect the shape of the histogram significantly. Notice in this figure, that the peak for the car in the foreground is completely hidden by noise! There is no hope of extracting the automobile with reliability from this image; and the thresholding in that figure illustrates the fact. Furthermore, in a recursive region-splitting scheme such as Ohlander's, the bad decisions made in early stages of processing (such as that shown in figures 6-2, 6-3, and 6-4) will never be corrected by later processing.

It seems, then that the "majority rule" problem in histogramming an image stems from the presence of a large number of peaks in the same histogram. It may arise any time several peaks are histogrammed together, and the effect is to make any single threshold value potentially harmful. The problem may be reduced by cropping an image, so that fewer regions are included in the same histogram. This cropping need not be exact to be useful, but seems to help immensely, even if the cropping is poorly done.

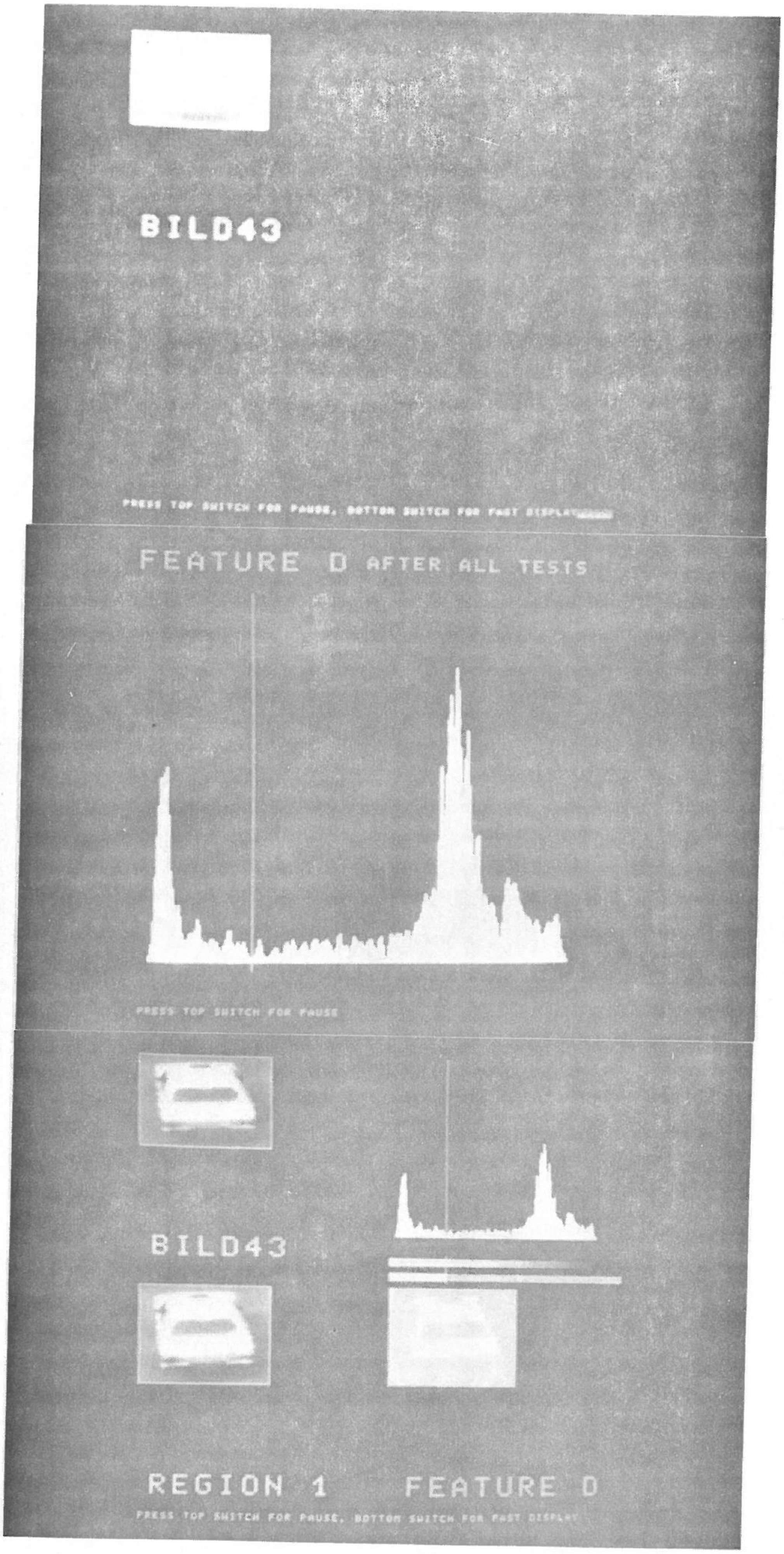
BILD43
PRESS TOP SWITCH FOR PAUSE, BOTTOM SWITCH FOR FAST DISPLAY
FEATURE D AFTER ALL TESTS
PRESS TOP SWITCH FOR PAUSE
BILD43
REGION 1
FEATURE D
PRESS TOP SWITCH FOR PAUSE, BOTTOM SWITCH FOR FAST DISPLAY

Figure 6-2: Image cropped closely around object

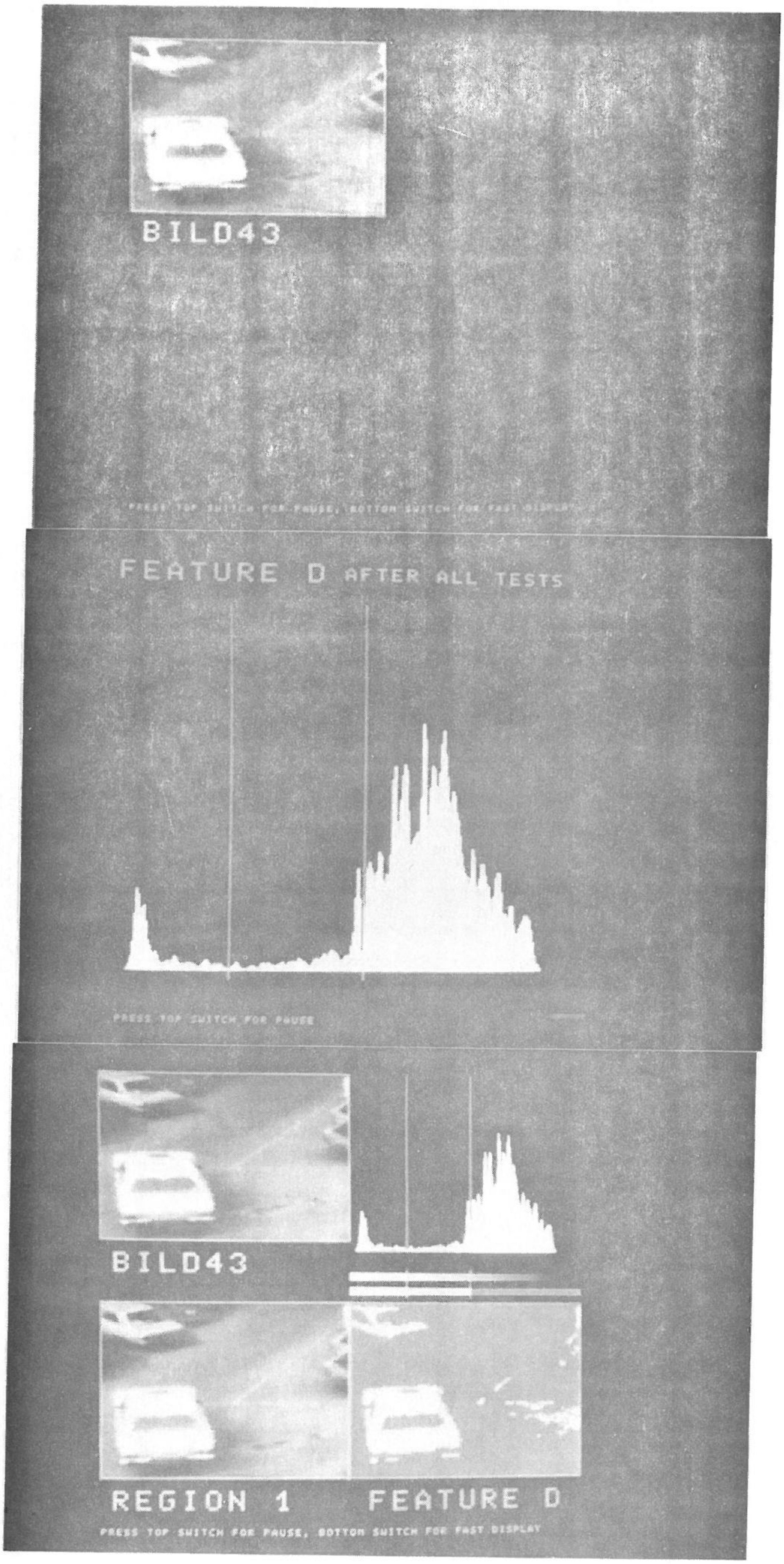
BILD43
PRESS TOP SWITCH FOR PAUSE, BOTTOM SWITCH FOR FAST DISPLAY
FEATURE D AFTER ALL TESTS
PRESS TOP SWITCH FOR PAUSE
BILD43
REGION 1
FEATURE D
PRESS TOP SWITCH FOR PAUSE, BOTTOM SWITCH FOR FAST DISPLAY

Figure 6-3: Image cropped approximately

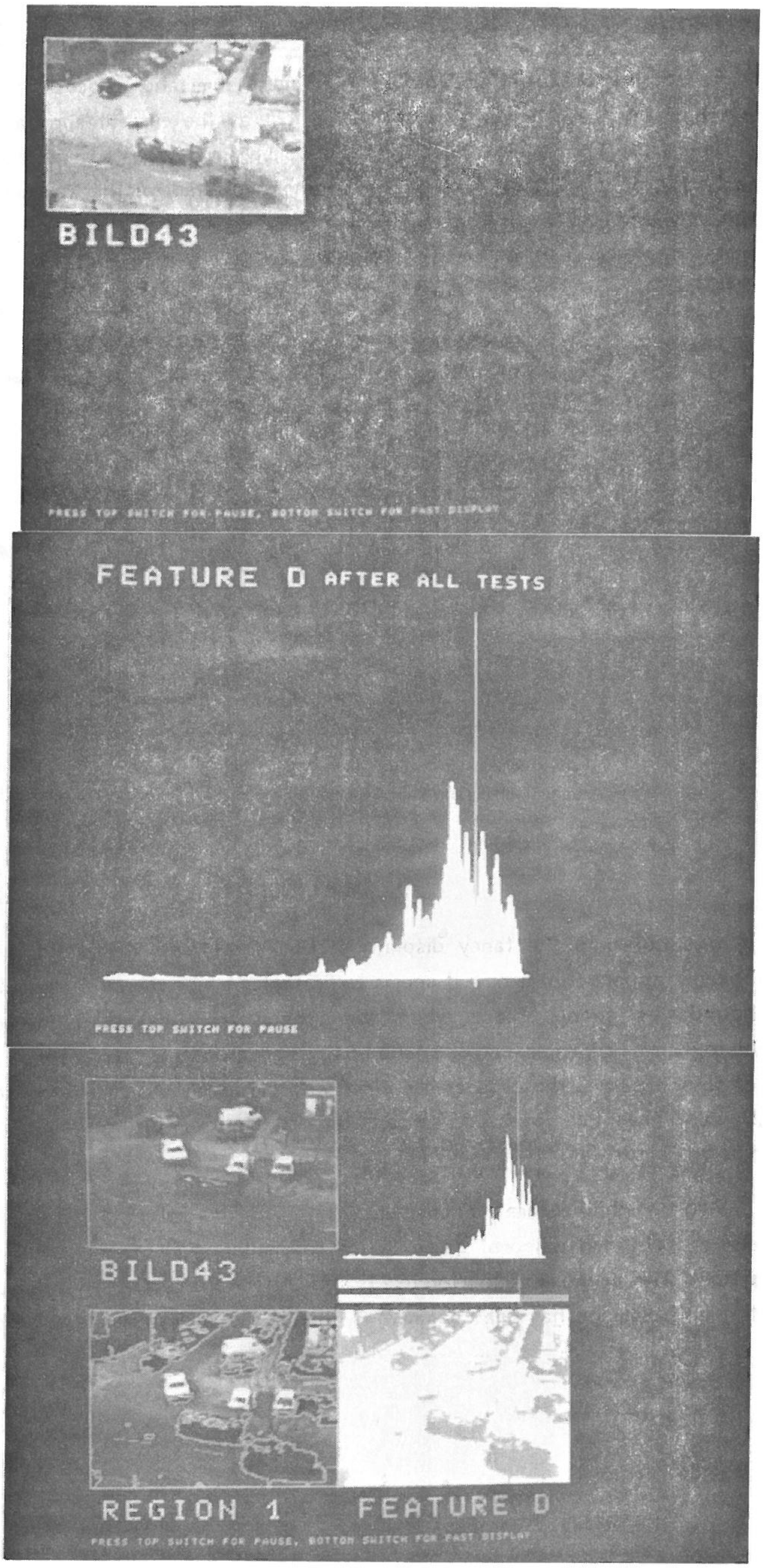
BILD43
PRESS TOP SWITCH FOR PAUSE, BOTTOM SWITCH FOR FAST DISPLAY
FEATURE D AFTER ALL TESTS
PRESS TOP SWITCH FOR PAUSE
BILD43
REGION 1
FEATURE D
PRESS TOP SWITCH FOR PAUSE, BOTTOM SWITCH FOR FAST DISPLAY

Figure 6-4: Image reduced without cropping

6.3 Edge Noise Problem

As pointed out in the description of the noise elimination process of MOOSE and KIWI, noise patches along the edges of a region are lost by the current approach to noise elimination. Noise in the middle of a region will always be simply merged with the surrounding patches (actually, only if there was a single threshold selected for that feature), but noise along the edge of a region is lost (Figure 6-5).

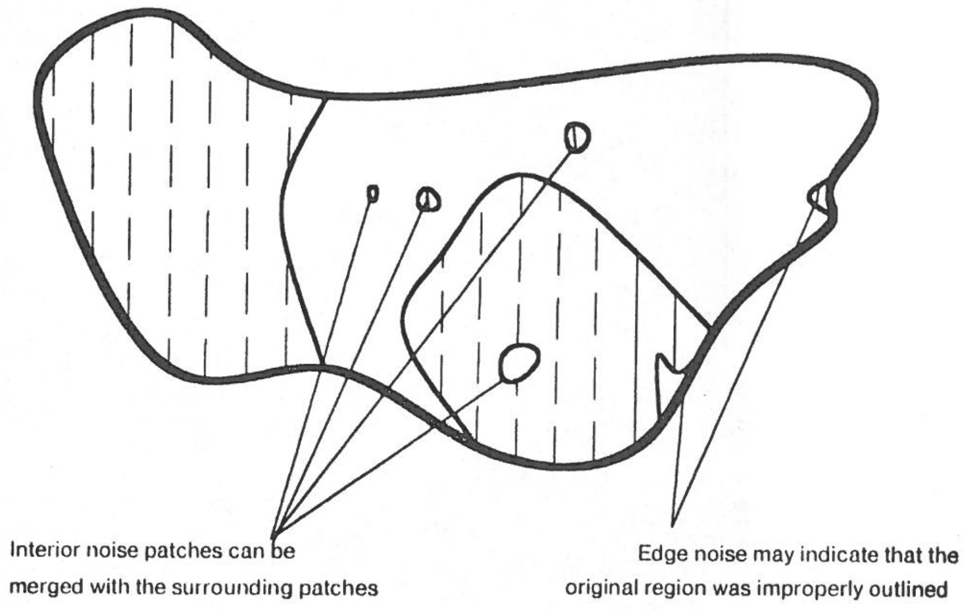
Interior noise patches can be merged with the surrounding patches
Edge noise may indicate that the original region was improperly outlined

Figure 6-5: Edge noise

It was not until the fancy displays of MOOSE were used that the extent of this problem became apparent, but now it seems to present a major source of uncertainty along all region boundaries except those which are very sharply defined. In fact, certain kinds of histogramming problems (see section 6.2) can contribute a great deal of noise along the edges of regions, since these areas are most sensitive to the choice of a particular threshold. (N.B.: Dave Milgram has observed this sensitivity of the edges of thresholded regions, and used it in various algorithms [Milgram 77].)

Two possibilities are obvious for dealing with this noise. It is conceiveable that it can be merged with an adjacent patch during segmentation even if it is not surrounded with that patch. The implementation for this is straightforward. As an alternative, we could wait until the entire image has been segmented, then see if we can merge noise areas with any of the several adjacent regions on the basis of similarity of color, etc. The second approach seems highly preferable, since noise along the edges of a region usually means that that region was not properly (or clearly) defined when it was thresholded, and the noise is likely to be better classified with some other adjacent region. In addition, some post-processing step which tries to account for the noise might also look to see if regions can be merged on the basis of similarity; this might help to alleviate some of the problems caused by the second variety of "majority-rule" problem (see above), in which small regions are hopelessly and meaninglessly chopped up.

6.4 Constant Non-Zero Intensity Gradient

While Ohlander's technique can deal with surfaces whose intensity is constant, it has difficulties when the surface has even a linear intensity change from one edge to the other. Since this kind of constant (non-zero) gradient in the intensity seems like a simple notion, we would like to capture this information in such a way that the algorithm can use it. Unfortunately, the intensity gradient requires two parameters for description (direction and magnitude of change). We would like to express this somehow in one-dimensional features, but we have not yet discovered how (or if) this can be done.

6.5 Peak Selection

The peak selection heuristics have been improved slightly, but this does not seem to have grossly altered the nature of the segmentation algorithm or results.

6.6 Degenerate Histograms

Price [Price 76] identified a phenomenon he calls "degenerate histograms", in which the histogram of a small region has (relatively) well-defined peaks which in fact are due to some noise or slight texture. Such a phenomemon is a combination of spatial and spectral effects, and is therefore difficult to deal with. The approach in KIWI is to pretend the threshold is good, then examine the spatial effects to see if any sense can be made of the results; however, this examination is performed in a very simple-minded way. In MOOSE, the feature-acceptability heuristics were amended to include the requirement that no more than some fixed percentage of the area of a region could be lost to noise by a thresholding; this seems to have the desired effect of rejecting truly degenerate histograms. Like all such simple tests, however, it is too conservative in some cases and too liberal in others. No systematic attempt to understand and analyze the degenerate histogram phenomenon has been made for KIWI or for MOOSE.

7. Highlights

The construction of MOOSE has been largely a success. The program was implemented, has features which render it useful for its intended environment, and has provided some insights into potentially powerful applications for the segmentation.

MOOSE achieved a considerable speedup over its predecessor, KIWI, partly through an absence of flexibility, and partly through the sophistication of the underlying (PASCAL) implementation. The display facility of MOOSE also overshadows the display capability of KIWI.

We have learned some important things about the segmentation algorithm itself from MOOSE. Some phenomema have been identified which seem inherent in histogramming images, especially along a single dimension, and some aspects of the noise elimination problem have been clarified. Some heuristics have been added to the program.

MOOSE appears to be useful for the implementation of a new method of object tracking. This technique is still being developed, and some of the remaining problems are described under separate cover. Although it is a long way from final implementation, we feel confident that the segmentation provided by MOOSE will be adequate to fill the required role.

Not least important, it was lots of fun to write MOOSE.

7.1 Acknowledgements

MOOSE was developed with considerable help from the staff and faculty of the DIG at the Fachbereich Informatik. I am especially grateful to Leonie Dreschler for her collaboration in developing a useful set of procedures for producing color displays.

Dr. Hans-Hellmut Nagel made MOOSE possible. His insight, guidance, and enthusiasm are largely responsible for its success. Thank you, Dr. Nagel.