5-30-1996

Segmenting Workstation Screen Images

Denis M. Serenyi
Dartmouth College

Segmenting Workstation Screen Images

A Computer Science Honors Thesis by
Denis M. Serenyi

Advised by
John M. Danskin

Dartmouth Computer Science
Technical Report Number
PCS-TR96-289

Submitted
May 30, 1996

Segmenting Workstation Screen Images

I. Introduction

Recently, many organizations have been attempting to develop an inexpensive “Network Terminal.” Such a device would have the same functionality as a PC without much of its hardware. In place of the RAM and hard drives that make current mulitmedia PCs so expensive, it would have a wide-bandwidth network connection and would use network servers to replicate functions previously implemented internally.

One design scheme for the Network Terminal is to simply have it display downloaded screen images. This Generic Imaging Terminal (GIT) has many advantages: for one thing, it can be almost as simple as a television set. In addition, the interface is not operating system dependent, so the user could run MacOS, Windows, and UNIX in the same box. The GIT has one major disadvantage, however: uncompressed screen images can be as large as 8–10 MB. Sending these fast enough to provide full interactivity (10–20 fps) over today’s networks is at best difficult and at worst impossible.

We believe, however, that the GIT is viable if the image data can be compressed well enough. Achieving a good compression ratio on screen images can be difficult because compression algorithms rely on accurate assumptions about data. Because screens incorporate a wide variety of image types, making useful assumptions is difficult. A text compression algorithm would not do well on a photograph, for instance. But if it were possible to segment the screen image into homogeneous areas, and send them individually to one of many compression algorithms, then we could avoid this problem. Our hope is to develop a screen image segmentation algorithm good enough to make the GIT possible even on standard Ethernet.

Objectives of Our Algorithm

Because screen images can incorporate many image types, the first thing we decided on is which ones our algorithm would attempt to identify. We selected these three:

- Noisy images: any images that are high in entropy. These usually correspond to photographs or complex screen backgrounds.
- Palletized images: any images that are low in entropy. Line art drawings, simple screen backgrounds, icons, and areas of solid color are some examples.
- Text.

The selection of these categories was driven by our goal to use the algorithm to improve compression ratios. Each one is tailored towards a specific type of image compression. Palletized images, with their few colors and low entropy, can be compressed well by GIF or other well known lossless algorithms. Noisy images, however, with their multitude of colors and high entropy, do not compress well losslessly. Lossy wavelet-transform algorithms usually perform better on these. Text images can be compressed extremely well with text-specific pattern recognition algorithms such as [22].

We should also mention that because this is not a computer vision problem, it is possible to simplify it in many ways. Computer vision refers to developing algorithms that allow machines to recognize patterns the same way our eyes do. Face recognition, motion planning, and many image segmentation problems fall into this field. Image segmentation for computer vision is concerned with identifying such things as photographs, walls, or people in an image based on a set of rules defining these items. These algorithms are complex and often ineffective.

Because we are only concerned with compression, classification *only* depends on entropy values. Although photographs are usually noisy images, if we happen to encounter one with low noise, there is no reason to classify it as one. By extension, even if only a part of a photograph is low in noise, it is preferable to break the image up rather than misclassify that one part. A good example of such a situation can be found below [fig. 1]. Because of this we are able to avoid many special cases that a computer visionist seeking to identify photographs would have to consider.

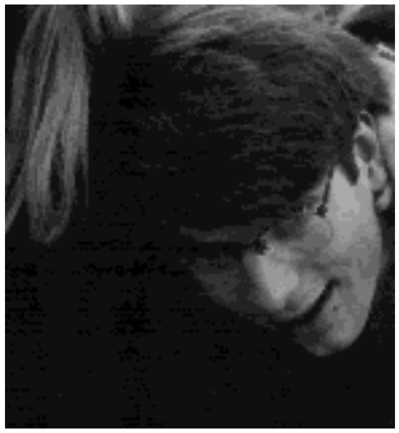


fig 1. Example of a photograph that has both noisy and palletized regions. In this photograph, most of the bottom portion is all black: it is only the face and the hair that is noisy. The all-black portion is a palletized image and should be separated from the top.

Previous Research on Image Segmentation

There has been a lot of recent research in the general area of image segmentation, but none specifically on screen images. For a good overview of the topic and the work that has been done since 1980, see [20].

Three categories of image segmentation routines exist: top-down, bottom-up, and a hybrid of the two, split-and-merge algorithms. The bottom-up approach, sometimes called "region growing," starts with small, isolated areas of the image. It then enlarges them by repeatedly concatenating neighboring pixels until a region border is found. This is a fast, greedy approach, but because it bases its decisions on a small amount of initial data, it is easy to end up with a bad segmentation if that data is not characteristic of the entire image. Good examples of bottom-up segmentation are [6, 21].

Top-down approaches avoid making bad initial decisions by considering the entire image in the first step. They recursively divide the image until all the regions are deemed homogeneous. Although these algorithms always make globally optimal decisions, they are always limited to splitting the entire image even when that does not yield the best result. Another problem is their performance: it is difficult to avoid repeating past work. Examples of the top down approach can be found in [9–12].

The last type, split-and-merge algorithms, combine the best features of each of the first two. First they act just like top-down algorithms, and recursively subdivide the image. Once all the regions are homogeneous, they are classified as one of the possible image types. Then the merging step, like bottom-up algorithms, combines neighboring regions of the same type. Although these algorithms provide the best of both worlds, they are usually too slow for any real-time application. Examples can be found in [9, 12].

The applications for image segmentation fall into two categories: texture segmentation and document image analysis. Texture segmentation algorithms, usually applied to aid image compression, attempt to isolate individual textures that comprise an image [1–8]. One of the common methods for doing this is to examine the entropy in various parts of the image. Texture borders lie where there are big jumps in entropy levels.

In document analysis, the goal is to extract text out of a document that may also include halftones or line art. There are several well known strategies for doing this. One is to analyze connected components to determine which ones are text elements [17–19]. The other is to first eliminate the blocks of white space on the page, and then segment the areas that remain in to text and halftones based on histogram analysis [14–16].

The three types of images we are hoping to identify in screen images, noisy images, palletized images, and text, cross the boundaries of these two applications. Separating noisy images and palletized images is a texture segmentation problem, whereas extracting text falls into document image analysis. Therefore, we decided to consider the two problems separately: first, extracting noisy images, then extracting text. Whatever remains is necessarily a palletized image. Because we define noisy images and palletized images in terms of entropy, using entropy analysis to separate these from one another made the most sense. A connected components algorithm was the most logical choice for identifying text, because it makes far fewer assumptions about text than histogram analysis.

The presentation of our findings are divided up into the following sections: Previous research (II), our top down approach for identifying noisy images (III), our bottom up approach for identifying noisy images (IV), connected components analysis for identifying text (V), and our conclusions (VI).

II. Previous Research

In this section, we will present a selection of methods that were most influential in our solution. First we will cover texture segmentation methods, which influenced our algorithm for extracting noisy images. Then we will discuss document analysis methods, which influenced our algorithm for extracting text.

Texture Segmentation

Many variants of top-down and bottom-up texture segmentation algorithms exist. Within the top-down category, the quadtree data structure often plays a central role. Tyagi and Bayoumi [9] use a quadtree to divide non-homogeneous regions into quarters. This makes the algorithm fast but also inflexible: quadtree region boundaries do not necessarily follow texture borders.

Chen and Pavlidis [12] use a quadtree and a co-occurrence matrix in a split-and-merge algorithm. As in [9], the quadtree is used for dividing the image. Once all the blocks are homogeneous, the algorithm decides whether adjoining rectangles can be merged based on statistics found in the co-occurrence matrix. Although the merging step is an improvement, this algorithm still has the same major flaw: texture borders usually do not coincide with quadtree boundaries.

This lack of adaptivity is solved by Wu [10], who performs texture segmentation for the purpose of image compression. Instead of simply dividing a heterogeneous block into quarters, he finds an optimal cut. This cut can lie at any row or column in the block, so no matter where a texture border lies, he will find it. He even allows for cuts to be at 45 or 135 degree angles, making the algorithm even more flexible.

Adams and Bischof [21], on the other hand, advocate region growing for texture segmentation. Remember that the key problem with bottom-up algorithms is the chance of starting with poor initial samples. They avoid this issue entirely by allowing the user to specify a set of seed pixels. Regions are then merged together based on differences in gray value. Although this does solve the major problem with region growing, having the user specify seed pixels is impractical for many applications, including ours.

A more sophisticated version of region-growing is proposed by Gong and Huang [6]. They use entropy instead of mean gray values to decide if two regions are similar to each other. Beginning with individual pixels, they scan through the image, applying a Markov chain measure of texture energy. When they find a group of pixels with high energy relative to those previously looked at, a region border has been discovered. Other similar algorithms are [2, 3, 5], with Derin and Elliot [2] using Gibbs Random Fields instead of Markov chains.

Document Image Analysis

Although there are several general approaches to extracting text from document images, most of them rely on making assumptions about the input data that do not apply to screen images. The most common restriction of these algorithms was that text and background could only be two colors (usually black and white). Toyoda, Noguchi, and Nishimura [16] make this assumption. Pavlidis and Zhou [14], on the other hand, make the invalid assumption that text does not vary significantly in size.

Connected components analysis, on the other hand, makes no assumption about text that does not hold in screen images. Fletcher and Kasturi [17] describe just such an algorithm. They define connected components as a set of uniform-colored pixels that are 8–connected to at least one other member of the set. Initially, they build them, storing each by its bounding rectangle. Then they eliminate all the non-text components based on a set of heuristics. Finally, the components are grouped into text blocks based on their proximity to one another. In their examples, only rarely did they miss a character.

Baird, Jones, and Fortune [18], also use connected components analysis for text identification. Their algorithm is supplemented, however, with document structure analysis and white-space removal. They first segment the page into columns by looking at large runs of white space. Once the columns are identified, they build the connected components. Their bounding rectangles are filled in with black, and the remaining white space is analyzed to determine where the text lies. Because this algorithm makes assumptions that are specific to document images, it is not as useful as [17].

III. Our Top-Down Approach to Noisy Region Identification

We decided to model our top-down algorithm for separating and identifying noisy images after Wu [10]. Like him, we elected not to use a quadtree. Instead, we find the optimal cut in the region and recurse on the two halves. Unlike Wu, we decided to only consider vertical and horizontal cuts, because 45 and 135 degree cuts would lead to non-rectangular regions. If only rectangular regions are possible, our representation of the regions is vastly simplified.

Entropy is our tool for finding the optimal cut. We define the entropy of a pixel as:

$$H = -\log_2\left(f_i \Big/ \sum_j f_j \right) - \log_2\left((f_i | f_j) \Big/ \sum_k (f_k | f_j) \right)$$

fig 2. The formula we used for entropy of a pixel. The first term in parentheses represents the probability of the pixel having a color i, and second parenthesized term represents the conditional probability of the pixel having color i given that the previous pixel has a color j.

This formula represents the sum of both zero order and first order entropy, with zero order represented as the first term and first order the second. We combined these two measures because we wanted a good balance between context-free and context-specific information. Zero order is context-free: the entropy of a pixel is simply based on the percentage of its color in the whole image. First order is context-specific: a pixel's entropy is based on the percentage of its color given the previous pixel's color. In the case of a pixel with color X, if color X is very rare in the image, then zero order entropy will be high. But if only color X appears after color Y, then its first order entropy will be low. Together, these two measures provide a good balance.

Using this measure of entropy, the optimal cut is found, as in Wu, by marching the cut across the region until a statistic is maximized or minimized. Once the optimal cut is found, the image is divided, and the process repeats recursively. The process is illustrated below [fig. 3].

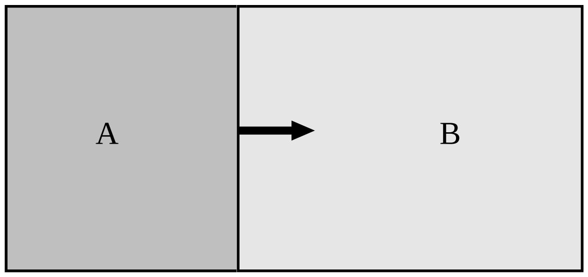


fig 3. Illustration of the search for an optimal cut in top-down algorithm. As the cut moves across the parent region, statistics are computed independently in A and B. When it is found, A and B become the new parent regions and the process repeats.

Selection of a Statistic

We developed three different statistics to use for determining optimality:

- Minimizing the maximum variance in entropy in either of the two subregions.
- Minimizing the total variance in entropy of the two subregions.
- Maximizing the difference in mean entropy between the two subregions.

When dividing heterogeneous regions, the optimal cut is the one that creates the two most homogeneous regions. This is why variance in entropy is such an important statistic: it measures exactly what we are interested in. A region with low variance will have entropy values bunched closely around a mean value, so it will be relatively homogeneous.

The first statistic, minimizing the maximum variance, is geared towards avoiding making cuts that would only separate a couple of rows. Any cut that is very near an edge of the region being divided will create a small region with very low variance, and a large region with very high variance. This situation would clearly not be considered optimal by this statistic.

The second method, minimizing the total variance, is motivated by our central goal: we try to make each cut reduce variance as much as possible. Although one of the two halves might still have a very high variance, on average the two subregions created will be the two most homogeneous ones possible.

The third method, maximizing the difference in mean entropy, is motivated by our desire to split the original region into the two which differ the most. If we can accomplish this, then we should eliminate all heterogeneous regions quickly. Take, for example, a heterogeneous region with one half a noisy image and the other half a palletized image. There is only one possible cut that will make both subregions homogeneous: the one lying right along the border of the two halves. When this cut is made, the difference in mean entropy will be at its peak. Any other cut would add some high entropy rows to the low entropy side and vice-versa, narrowing the gap between the means.

Stopping Condition and Classification

Finally, we need a stopping condition for the subdivision process, for at a certain point a region will be homogeneous enough to be classified as either palletized or noisy. Once again, variance was chosen as the best way to determine this. If it is low enough relative to the mean entropy, then the region is homogeneous and the subdivision stops. In order to differentiate noisy regions from palletized regions, the most straightforward method is simply to use mean entropy: if the mean is above a certain threshold, then it is noisy. We left the establishment of specific values for these tests up to the discretion of our experiments.

Experiments

We tested these three top-down algorithms on twenty screen images. The images were designed to vary as much as possible, but still consistently follow what a typical screen looks like. We also made sure to include images that tested situations we suspected would cause our algorithms to fail. One screen shot, for example, included four separate noisy images.

Two common windowing environments, MacOS and IRIX, along with many common applications were used as our primary test vehicles. These two windowing systems follow the characteristic patterns of all current ones: they include window frames, icons, scroll bars, menus, etc. The most common application we used was Netscape Navigator; web pages are not only a common item on computer screens, but they provide a challenging mixture of text, simple graphics, and complex photos. Other applications used were examples of word processing and e-mail programs, graphical editors, UNIX shell windows. We also tried to vary screen backgrounds as much as possible to see if they provided a challenge for our segmentation routines.

We judged the performance of the three algorithms based on a standard set of tests. To evaluate how correct the segmentation was, we asked:

- What percentage of the output regions were homogeneous?
- What percentage of the image area fell in a homogeneous region?

And to evaluate the correctness of our classification:

- What percentage of the screen area was classified correctly?
- Of misclassified rectangles, what percentage consisted of noisy images being misclassified? Palletized images being misclassified? Heterogeneous areas?

In order to properly evaluate correctness of segmentation, asking both those questions is important. The former gives too much weight to a large number of tiny heterogeneous blocks. The latter, on the other hand, weighs one large heterogeneous area too heavily. Even if the vast majority of pixels in that area are of one type, all of them would be labeled heterogeneous. So if we consider both questions, we can get a complete picture of how well our algorithms are performing.

One problem with evaluating correctness of segmentation is that any algorithm, even a random one, would eventually divide the screen into regions so small that they would all be homogeneous. To prevent this happening, we gave the algorithms a time limit of six iterations to achieve their goal.

Something else important to note is that the numbers we collected represent estimations rather than exact counts of pixels. Because we are really only concerned with the big picture, we felt it was satisfactory to just eyeball the output.

Experimental Results

The results of our top-down algorithms were mixed. Though we were able to properly segment the test images very well on average, a significant percentage of regions were misclassified. See figs. 12–14 for sample output.

Results of Segmentation Algorithms

Both the minimum maximum variance and the minimum total variance statistics failed to properly segment our test images consistently. It was common to have anywhere from 20% to 65% of the total image area in a heterogeneous rectangle [see figs. 5, 6 for complete data]. In both algorithms, on average only about half of the output regions would be homogeneous. They would fail even on test images with clearly identifiable noisy and palletized areas. The borders between these areas were ignored, and instead the cuts were randomly placed [fig. 12, 13].

The problem stemmed from using variance in entropy as a statistic. Although in theory it should be a very good measure of homogeneity, in practice it functioned quite differently. As the cut approached a border between noisy and palletized regions, no clear pattern emerged in variance. Rather than descending slowly to a minimum, or rising to a maximum, it would sway up and down, making it impossible to identify where the border lay [see fig. 11 to illustrate this].

The maximum mean difference algorithm, however, segmented our test images extremely well. Mean entropy, as opposed to variance in entropy, proved to be an extremely reliable and useful statistic. Notice in the graph below [fig. 11], as the cut approaches the region border, the mean entropies of the two subregions diverge. Their difference reaches a peak right at the border. The division, therefore, is made in the optimal location. This led an excellent average correctness for segmentation: 98.6% [see fig. 7 for complete data].

Results of Stopping Condition and Classification

The classification algorithm, on the other hand, did not perform well. Although we tried many different values for our thresholds, at best the algorithm only classified regions correctly 67% of the time [see fig. 8]. The mistakes fell into two categories:

- Small palletized regions were misclassified as noisy.

- Heterogeneous regions were prematurely classified as noisy images.

As can be seen from the scatter graph below [fig. 10], homogeneous misclassified regions were always extremely small. In addition, they always consisted of palletized regions being mistaken for noisy ones: not a single homogeneous noisy region was misclassified. The reason for these mistakes has to do with the statistic we use for classification, mean entropy. Very small regions, regardless of their type, will have a lot of randomness simply because of their size. With a lot of randomness comes a high mean entropy, so they end up looking like noisy images regardless of what they actually are.

The other problem was that sometimes heterogeneous regions would be prematurely classified as homogeneous noisy images. The culprit in this case was variance in entropy, the statistic we used as our stopping condition. In many cases a homogeneous noisy image would have a high variance in entropy, so that threshold could not be too strict. But heterogeneous regions, by definition, also have a high variance, so sometimes one would be mistaken for the other.

These errors hindered the previously outstanding performance of the segmentation algorithm. The percent of screen area in a homogeneous rectangle dropped from 98.6% down to 77.55% after the stopping condition was added [fig. 9]. Sometimes the entire screen was classified as a homogeneous noisy image, when in fact it was a combination of noisy regions and palletized regions.

Discussion

Although this performance is not poor, a correctness rate of two-thirds is not enough to significantly improve the compression ratios of screen images. In addition to the specific mistakes we encountered, the algorithm has fundamental problems: making an optimal linear cut does not lend itself to the rectangular structure of screen images. In some cases where windows overlap, even the best cut would necessarily split a noisy image into smaller fragments [fig. 4]. In light of these problems, we decided to try and improve our results with a bottom-up algorithm.

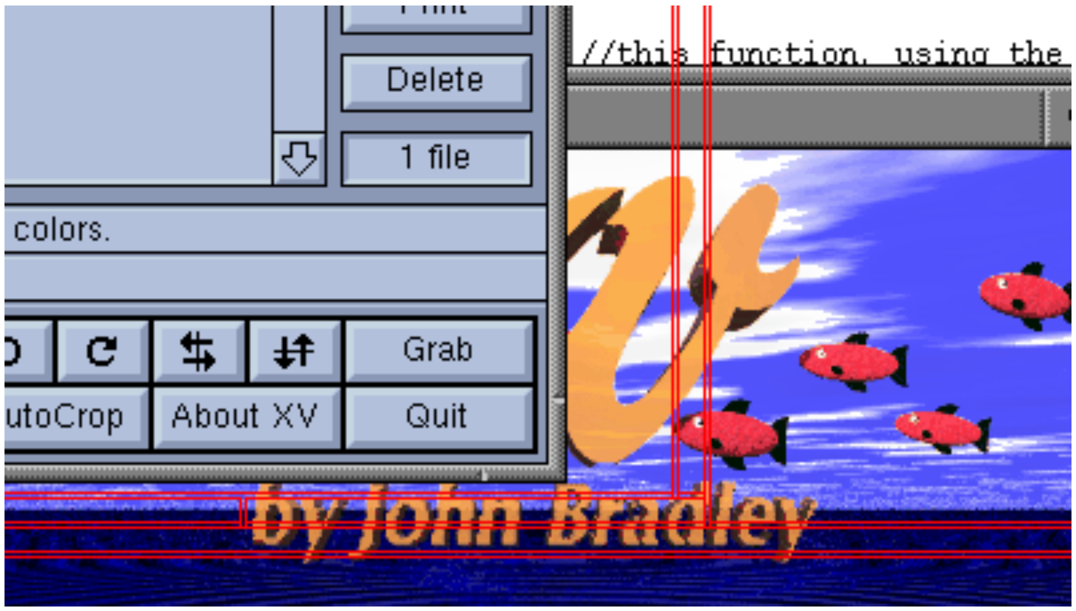


fig 4. Example of an image where no optimal cut can be made. In this situation, the best cut would fall right along the border between the gray window, a palletized image, and the blue noisy image behind it. It would necessarily split up this noisy image, regardless of whether the cut followed the horizontal or vertical edge. The actual cuts made by the maximum mean algorithm are shown in red.

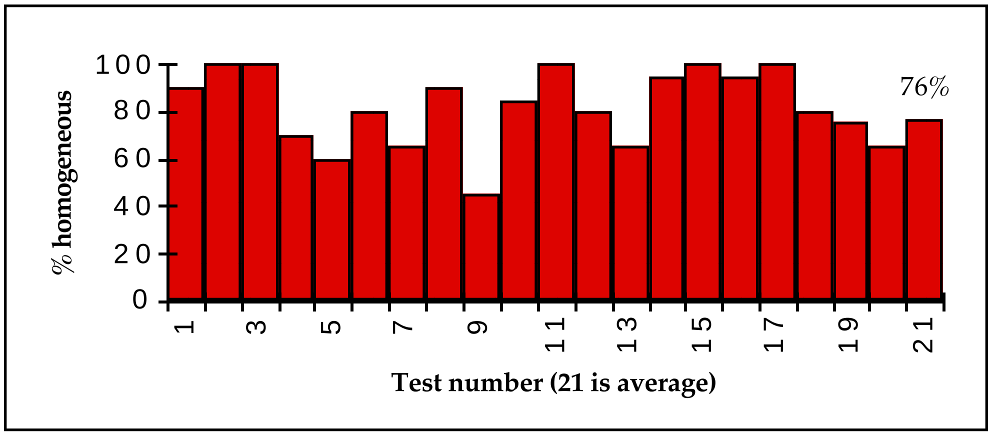


fig 5. Minimum maximum statistic. Percent of the image that was part of a homogeneous rectangle after six iterations of the algorithm. The twenty test images are represented by 1–20, with 21 being the average (76%).

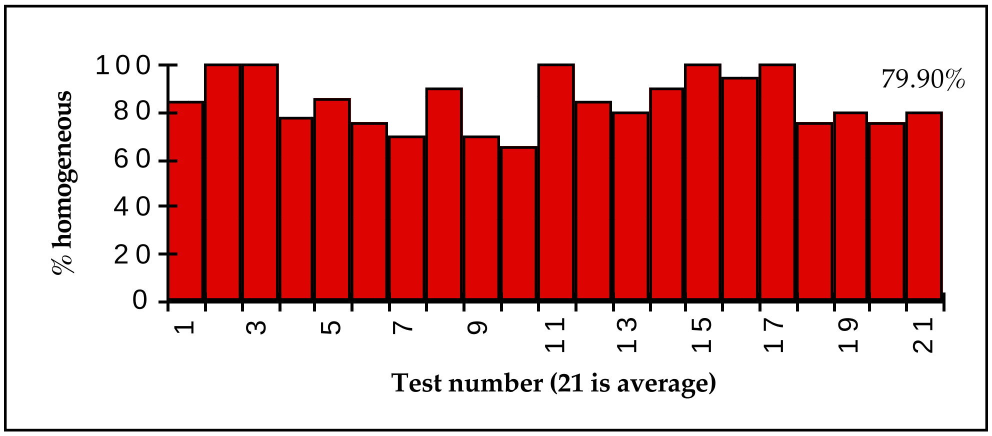


fig 6. Minimum variance statistic. Same format as above, with 79.9% being the average.

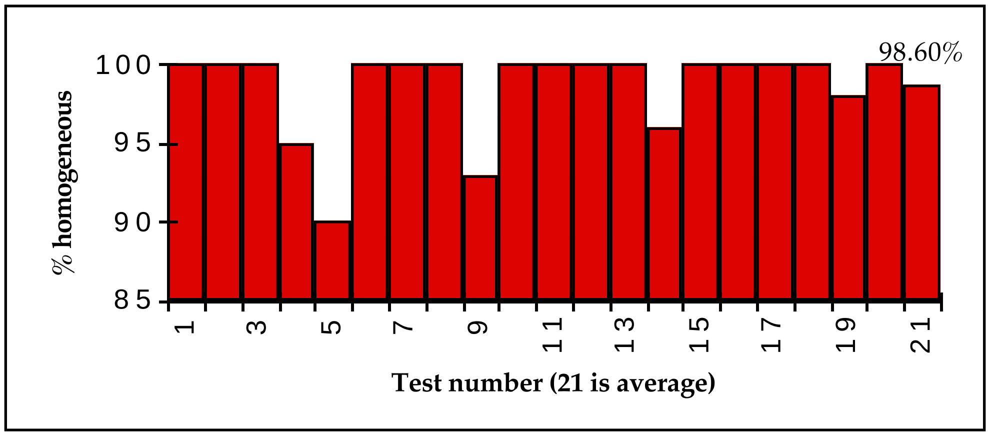


fig 7. Maximum mean difference statistic. Same format as above graphs, with the average being 98.6%.

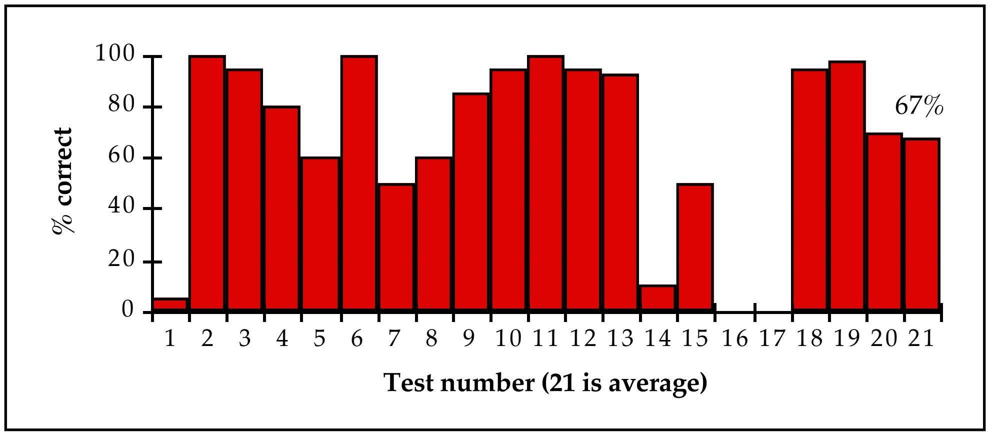


fig 8. Percent of the image that was classified correctly, after using the maximum mean difference algorithm to split it into homogeneous regions. The twenty test images are represented by 1–20, with 21 being the average (67%).

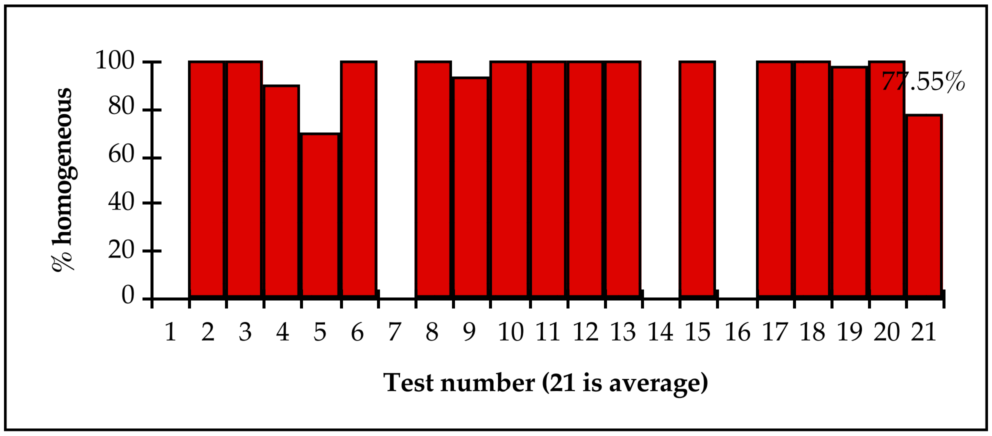


fig 9. Maximum mean difference statistic after the classification step has been added. The differences between this and the previous homogeneity graph resulted from heterogeneous regions that were prematurely classified.

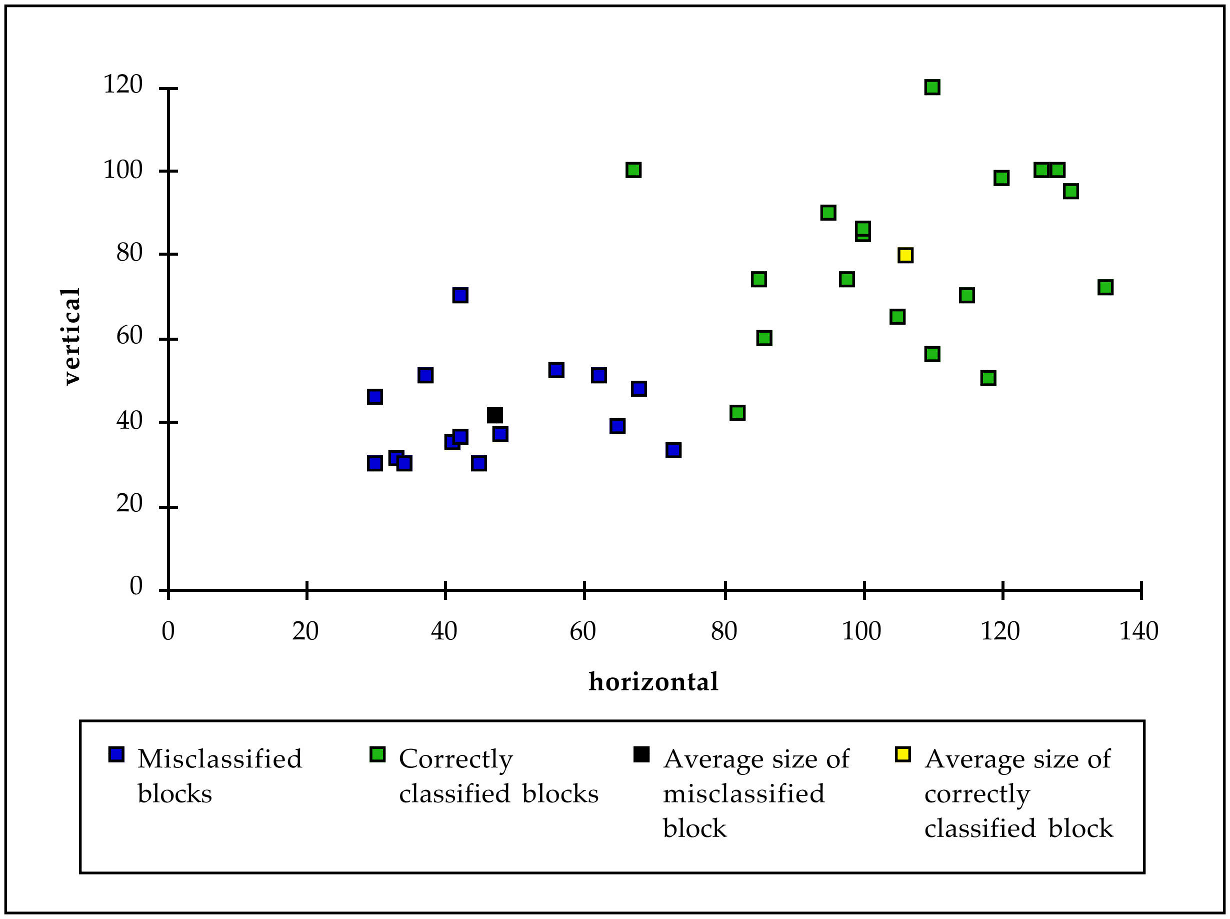
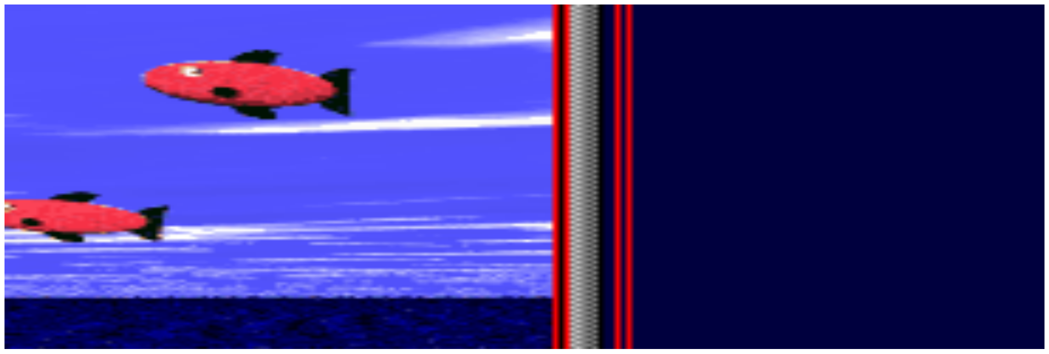


fig 10. Size of homogeneous misclassified blocks versus size of correctly classified blocks. Each blue mark represents the average size of misclassified blocks in each of the 20 test images. The black dot is the overall average. The green marks represent the average size of correctly classified blocks in the test images. The yellow is the overall average. There are fewer than 20 red and green points because for some of the tests there were either no homogeneous misclassified blocks, or no correctly classified ones.



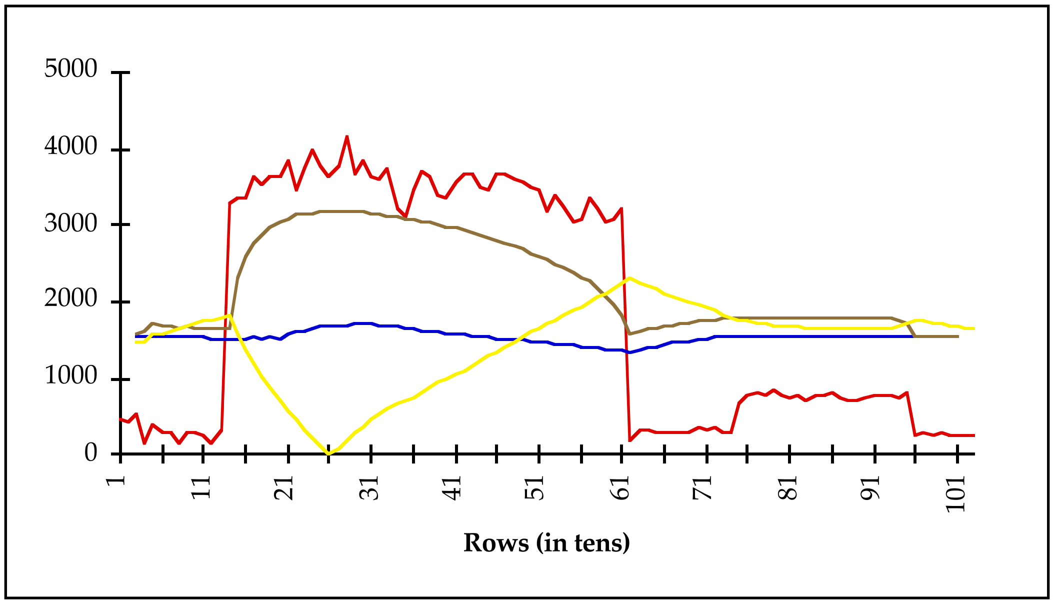


fig 11. The graph shows the change in various statistics as the cut was moved across a test image. The blue line is the maximum variance. The brown line is total variance. The yellow line is absolute difference in mean entropy. The red line represents per column entropy values, with the large raised area being a noisy image between columns 125 and 610. Note how the mean difference reaches a sharp spike right at the edge of the large noisy region, whereas the other two statistics waver inconclusively. In the portion of the actual image shown above, you can see this border between the noisy image on the left and the palletized image on the right. The double red line on the left, falling exactly on the border of the two regions, is where the maximum mean difference was.

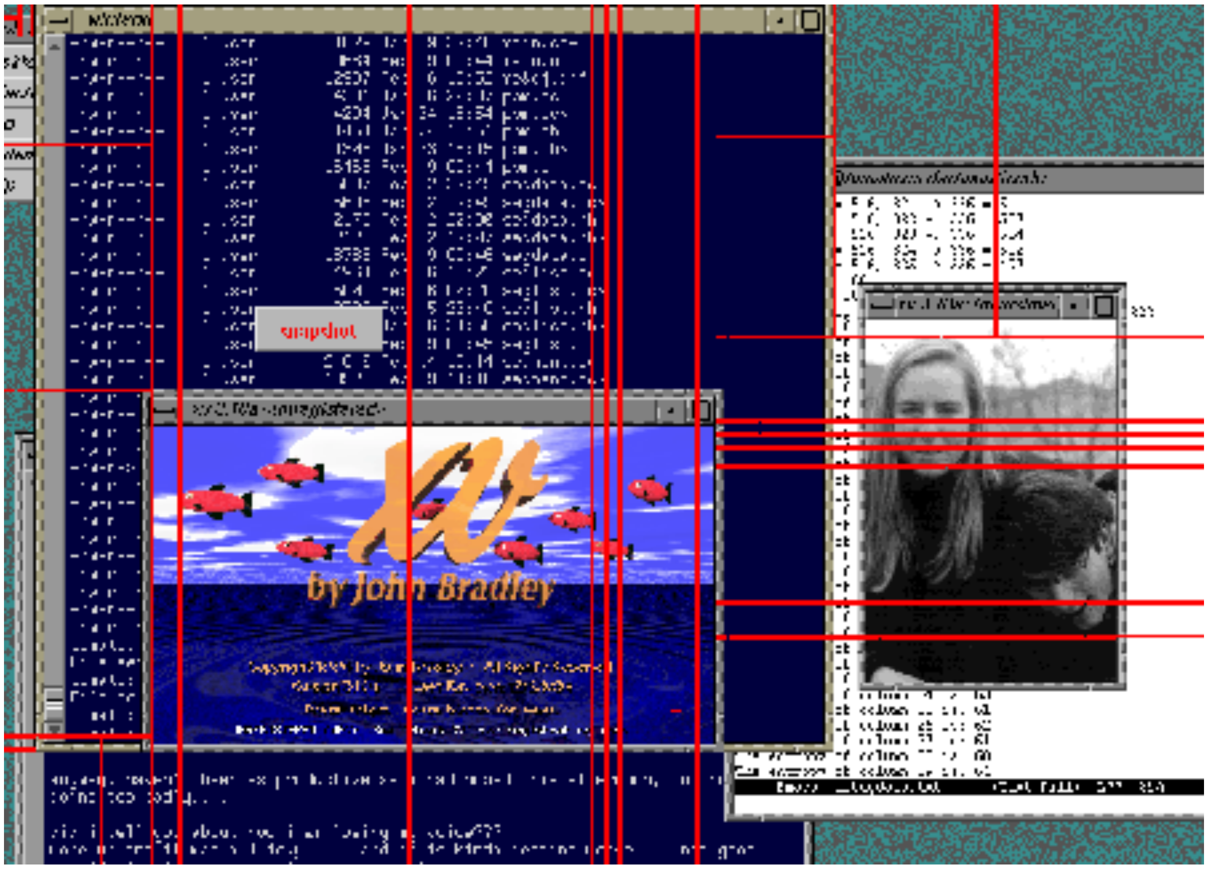


fig 12. Example output of the minimum maximum variance algorithm. In this test, there are clearly two noisy images surrounded by some text windows and a screen background. What we would like to see are the two noisy images isolated from the rest. Instead, the cuts (shown in red) appear random. Note that all these sample images have been reduced from their original size, obscuring some details.

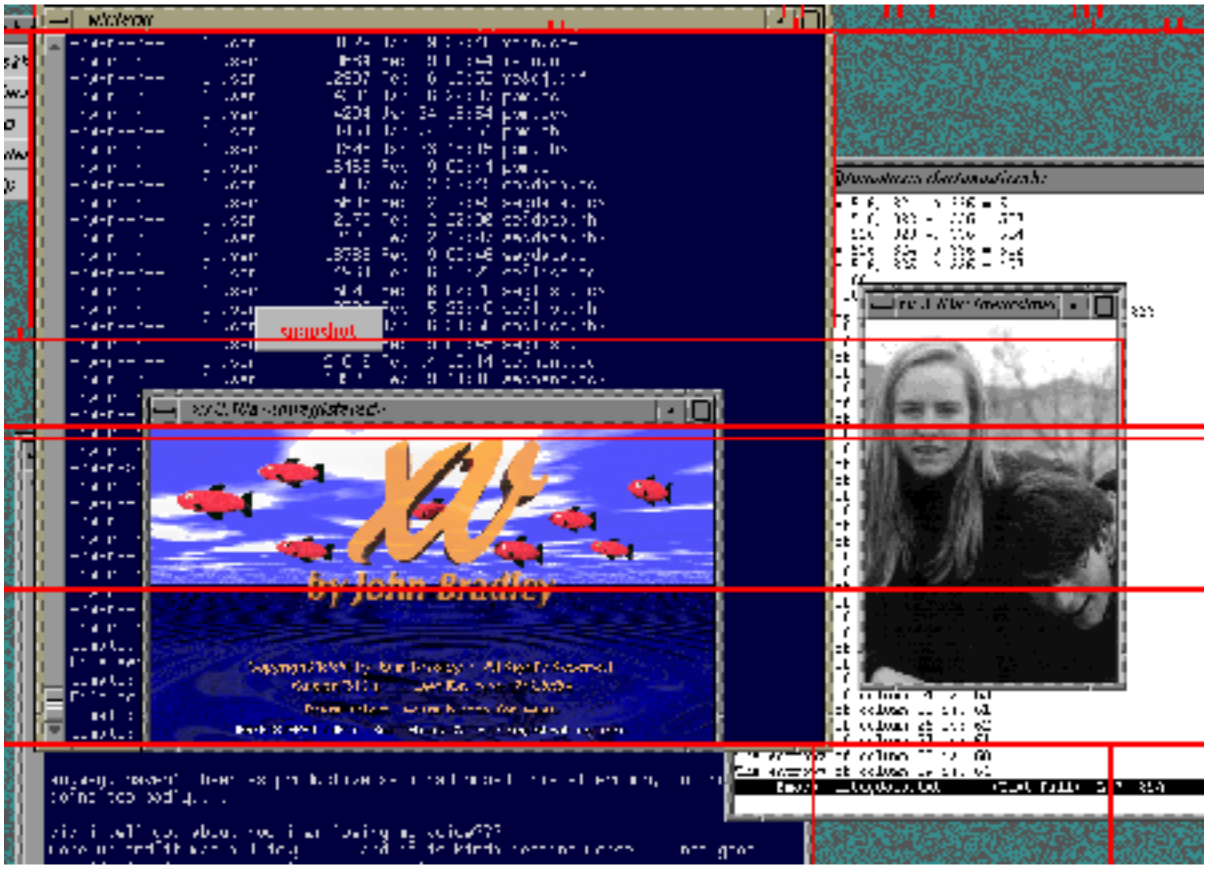


fig 13. Output of the minimum total variance algorithm. Like in the minimum maximum variance algorithm above, the two noisy images in this example have not been isolated from the palletized images surrounding them.

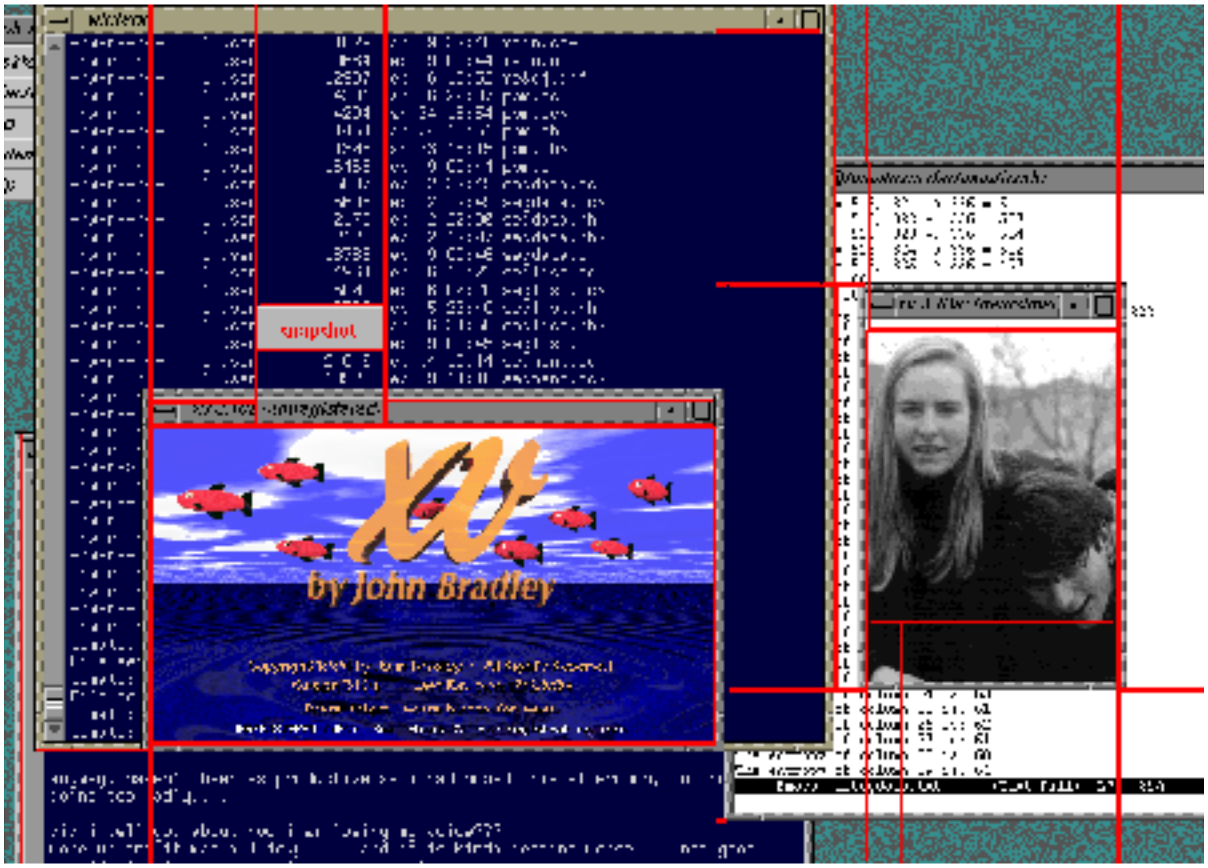


fig 14. Example of the maximum mean difference algorithm. Here the cuts are exactly what we want. The noisy image on the left has been completely isolated. The photograph on the right has been isolated except for the bottom section, which is almost all black and therefore not noisy anyway. Some cuts were obscured when the image was shrunk.

IV. Our Bottom-Up Approach to Noisy Region Identification

As stated above, a bottom-up, or region growing algorithm begins with isolated pixels and repeatedly combines similar surrounding ones, building up a homogeneous region until a border is found. Once one region has been grown as much as possible, the same process is repeated on the remaining portions of the image until the entire image is divided. Bottom-up approaches are fast because they are greedy, but they can perform poorly if they make bad initial decisions.

Our bottom-up algorithm begins by growing a thirty-by-thirty region in the upper left hand corner of the screen. We felt this was a safe choice for an initial region: borders between noisy and palletized areas do not usually lie in corners. An initial block of that size was chosen for several reasons:

- Smaller blocks are not useful for compression.
- Smaller blocks have too few pixels for good statistics.
- Larger blocks increase the chances that they include both palletized and noisy areas.

Once our initial region is chosen, we enlarge it until a border is encountered. The adjoining row on the bottom and adjoining column on the right are compared with the region, and if they are similar then they are merged [fig. 15]. A statistic, or some basis for comparison, must be used to determine similarity; we will discuss our choice of methods below. This process repeats row by row and column by column until no more growth is possible.

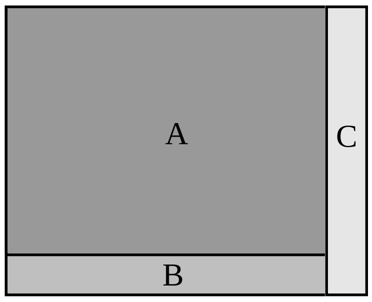


fig 15. Illustration of region growing. A is a homogeneous region. B and C are both tested to see if they are similar to A. If they are, then they are added to A.

Once one region has been expanded as much as possible, we continue segmenting by making its upper right hand and lower left hand corners into starting points for two new ones [fig 16]. These locations make sense for three reasons:

- We already know that these places are the start of new regions, because they are where the parent region stopped expanding.
- Choosing them ensures that regions only have to grow down and to the right, simplifying the algorithm.
- Starting anywhere else would require the algorithm to consider a complex polygon somewhere in the screen image. Avoiding this again simplifies the algorithm, and eliminates many of the situations where the two regions might overlap.

They could overlap, however, if the upper-right hand rectangle grows to be taller than the original region and the lower-left hand one grows to be wider [fig 17]. To prevent this from happening, we simply constrain the upper-right hand rectangle to be no taller than its parent [illustrated by D in fig. 16].

This process of creating child regions and growing them repeats until the entire screen has been segmented.

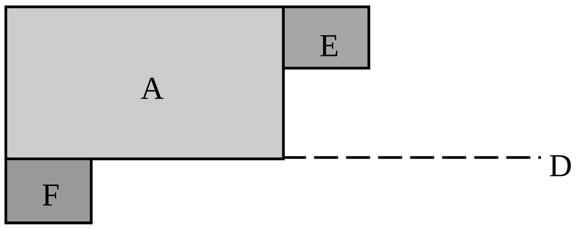


fig 16. Illustration of picking region starting points. If B and C (in fig. 15) are not similar to A, A is a finished region. E and F become the starting rectangles for two new regions, which are then grown in the same manner. Because it is possible for E and F to overlap, C's growth is limited to the bottom of A, represented by the dashed line.

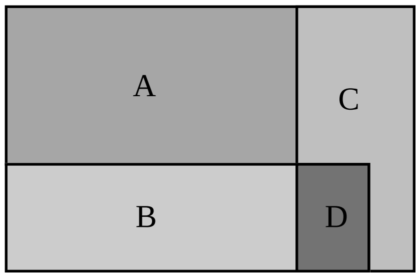


fig 17. Example of the region growing overlap problem. A is a region that has finished growing. B and C are two child regions. If both B and C are not restricted, the overlap region D is possible.

Heuristics for Determining Homogeneity

This algorithm requires that we have some method for deciding whether new rows and columns are similar to the region being grown. As we shall show below, this method is intertwined with actually classifying the blocks, so the steps of region subdivision and classification merge into one. We developed two methods for making this decision, both based on entropy. One is based on global data, the other based on local data.

Global Entropy Approach

This method consists of first building up a corresponding “entropy image” from the original screen shot [sample shown in fig. 18]. In this image, per-pixel entropy values from the original correspond to gray values, with entropy being defined the same way it was in our top-down algorithm [see section III above]. Previously, however, we calculated entropy in one pass, so the probability of a pixel being a certain color was only based on the ones before it. For the entropy image, we make two passes. The first builds a histogram of probabilities for all the colors in the image. The second calculates per-pixel entropy according to these probabilities. Therefore, a pixel’s zero-order entropy value will be fixed independent of its location. What will result from this is an image with relatively bright gray areas corresponding to noisy regions, and relatively dark gray areas corresponding to palletized ones. Just from eyeballing it, we can easily see what the segmentation should be.

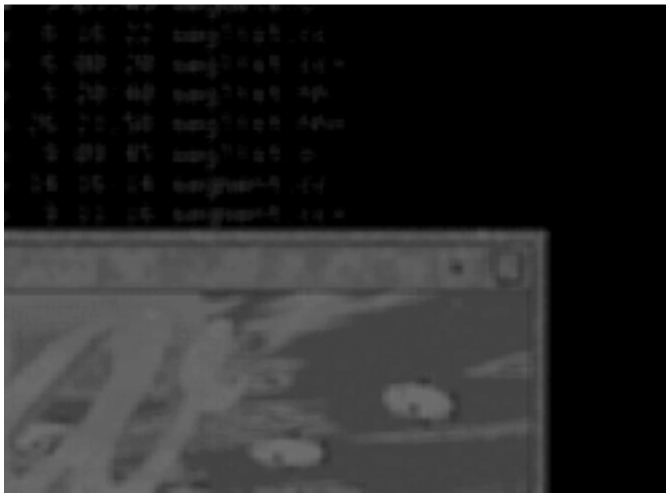


fig 18. A sample entropy image. The light area on the bottom right represents a noisy image, the dark area around it a palletized image.

Because pixel values in the entropy image imply both a segmentation and a classification, we can combine the two steps. When beginning with the initial thirty-by-thirty region, the mean entropy value is used to make a classification. This implies some threshold, below which the region is classified as palletized, and above which it is classified as noisy. This threshold was established through our tests described below; for now its actual value is unimportant. When growing the region, testing rows and columns for similarity can be as simple as calculating their mean gray value. If a row's mean value is on the same side of the threshold as the region's, then they can be combined.

There is one special case that we must be aware of: sometimes a large palletized region might border a small noisy one. One example of this is a small photograph on a web page [fig. 19]. When growing the palletized region, this noisy image could be missed if it only occupies a small fraction of the width in the row to be added. To solve this problem, we only consider the mean entropy level within a sliding window of thirty pixels. As this window moves across the row, if at any time the mean is above our threshold, then the row is considered to be a region border. The converse situation, a large noisy image surrounding a small palletized image, happens very rarely in screen images, so we felt it was unnecessary to consider it.

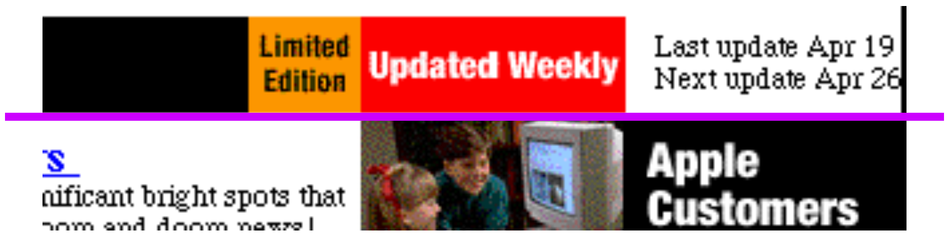


fig 19. Example of the special case for global entropy region growing. This is a small clip of a larger screen shot: note the small noisy photograph in the lower middle part of the image. If everything above the purple line is part of one palletized region, then the noisy image could easily be overlooked because it is so small.

Local Entropy Approach

The global entropy approach provides a good method of separating noisy images from palletized images, but ideally we would like our segmentation algorithm to do more than just that. For example, the global approach would lump together a region of black-on-white line art with a region of white-on-black line art. It would not be wrong to do this, for both the regions are palletized. Keeping in mind our goal to increase compression ratios, however, it would be advantageous to isolate these two from one another. They would certainly compress better if separated, because the average entropy would be lower. So although it is not essential to isolate varying palletized regions, it would be advantageous to do so. The local entropy approach hopes to accomplish this.

Instead of building up an entropy image, this algorithm only uses data within each region to measure entropy. We build a histogram, like the one in the global approach, but only for pixels within the initial region. Per-pixel entropy is then calculated based on that histogram. If the mean entropy value is below a certain threshold, it is classified as palletized, otherwise it is classified as noisy.

Testing new rows for similarity is done differently depending on the classification type. For palletized images, we measure a new row's entropy in terms of the region's histogram. In this scenario, a low average entropy means that the row has the same colors and patterns as the region. Conversely, a high average entropy implies that the two have little in common. If the entropy is in fact low enough (below some threshold), they are deemed similar and merged. The row is also added to the histogram, so that the histogram grow along with the region.

Measuring entropy in this manner will identify not only a border between a palletized and noisy region, but also a border between two dissimilar palletized regions. To illustrate this, consider the two line art images from above: one white on black, the other black on white. If the initial block falls within the white on black area, then any row added within that image will have a low average entropy. When the black on white region is encountered, however, most of the pixels will be white. This will be unexpected, so the entropy will be high.

In noisy regions, on the other hand, because the entropy is consistently high, it would be impossible to judge similarity in the same way. The first row of a palletized region, when compared to the noisy region, would seem just as random. Therefore, in many cases palletized and noisy areas would get lumped together. To remedy this, in addition to measuring entropy in terms of the existing region, we measure it in terms of the next couple of rows following the region. If the entropy is low for the next couple rows, then this row is considered to be the start of a palletized region.

Experiments

In order to keep a level playing field for comparing our top-down and bottom-up algorithms, we used the same set of twenty images. Because the bottom-up algorithms combine the segmentation and classification steps, however, it was not necessary to ask the same questions. We were primarily interested in:

- What percentage of the screen area was classified correctly?
- What percentage of rectangles consisted of noisy images being misclassified? Palletized images being misclassified? Heterogeneous areas?

For the local entropy approach:

- What percentage of palletized images fell within homogeneous rectangles?

Results of the Global Entropy Approach

As can be seen from the graph below [fig. 21], the global bottom-up algorithm performed significantly better than any of the top-down algorithms. On average, 96% of the area in our test images was classified correctly, an improvement of 29% over the previous best. The algorithm was perfect at identifying noisy images: over all our tests, not one mistake was made. After trying several different thresholds for mean entropy, we settled on a value of six. Any average entropy value greater than that meant that the image was noisy, anything less and it was palletized. Though this did not prove ideal in all situations, as we shall see below, overall it resulted in the best output [see fig. 25 for sample output].

We encountered three major problems with this method: misclassification of small palletized regions, fragmentation, and misclassification of screen backgrounds. Although mistakes were infrequent, when they did occur it was usually a small, isolated palletized region being labeled noisy. Raising the threshold did not remedy this situation: if it was any bigger than six, noisy regions would be misclassified. Instead, our solution was to add a pruning step to the algorithm. Any isolated noisy region smaller than forty-five by forty-five pixels was assumed to be a mistake, and reclassified as palletized. This solved the problem in most cases, and in no test did it introduce new errors.

Another problem which we encountered was that the screen was fragmented into many tiny blocks. When growing a palletized region, sometimes an isolated row would be above entropy threshold. The algorithm would mistake this anomaly for a region border. The result was hundreds of thirty by thirty palletized regions. To prevent this, if a row was above the threshold, before deciding that it was a region border we would look ahead four rows. Only if all those rows were also above the threshold would we consider this row a border. This reduced the problem somewhat, and in no cases did it introduce misclassifications.

The final problem we encountered had to do with screen backgrounds. Simple backgrounds, with only a few colors, were correctly segmented and classified as palletized [fig. 20]. However, complex backgrounds, multi-colored granite textures for example, were classified as noisy images. This happened simply because those backgrounds had entropies above the threshold. In the end, we elected to ignore this problem because it was questionable if the "misclassified" backgrounds were indeed palletized. Because the entropy values were so high, a lossless compression algorithm might not be able to compress them very well. If that is indeed the case, then they should be classified as noisy.

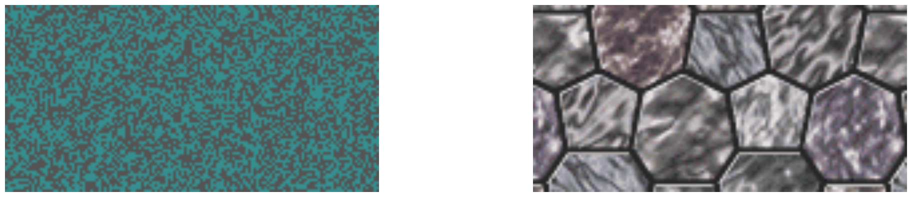


Fig 20. Noisy vs. non-noisy backgrounds. The image on the left is a sample of a background that was classified as a palletized image by the global bottom-up algorithm. The one on the right was classified as noisy.

Despite these problems, the overall performance of the algorithm was excellent. In fact, if we ignore the questionable misclassification of some screen backgrounds, the mean correctness for segmentation rises from an already acceptable 96% to 98.5% [see fig. 23]. After seeing these results on such a wide range of test cases, we find it unlikely that this algorithm will run into a screw case.

Results of The Local Entropy Approach

The local entropy approach did not perform nearly as well as its counterpart. Although it classified on average 87.6% of screen area correctly, that in itself is deceiving [fig. 22]. Its major problem was with identifying noisy images, which only made up between zero and twenty percent of the area in our test images. A more telling statistic is the amount of noisy image area that was identified. This was a dismal 7.5% [fig. 24]. The problem was not with the threshold we set; in fact, we tried many different values and got equally poor results. The cause was instead the size of the sample upon which classification was based. Remember that we classify a region before expanding it: the decision is made based on data within the initial thirty by thirty block. Even in a region that is noisy overall, entropy in such a small part is usually low, making noisy and palletized images indistinguishable.

In classifying palletized regions, on the other hand, the local entropy approach was reasonably successful. In many cases, screen backgrounds, windows, buttons, and icons of different colors would be separated from each other as they logically should be. There was also a limited amount of fragmentation. As opposed to the global approach, which would often create ten regions where only one was needed, when this algorithm started a new region it was usually not without reason.

In some cases it did make mistakes, however. There was a tendency to create either extremely large heterogeneous regions or a group of tiny unnecessary ones. Because entropy calculations were based only on the heuristics of the region, large regions would tend to be more lenient in adding rows than small ones. Overall, the improvements made were neither significant nor regular enough to offset the problems. The global entropy approach, therefore, was our algorithm choice for segmentation and classification of noisy and palletized regions.

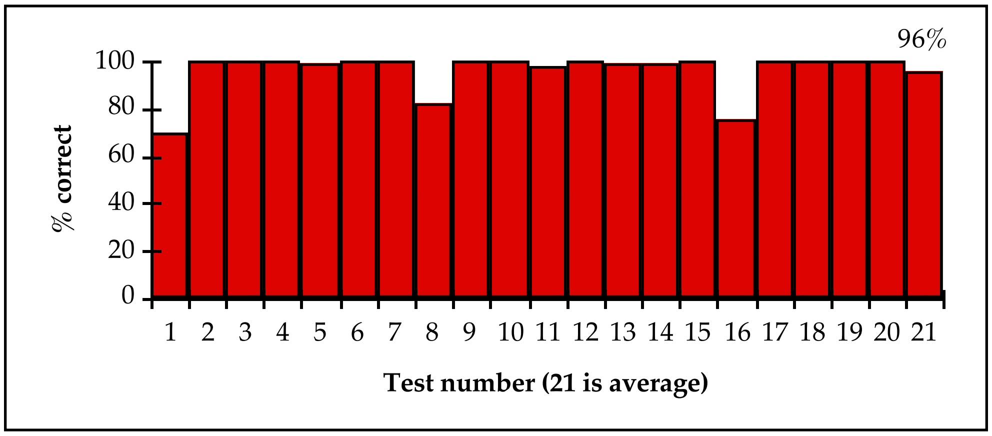


fig 21. Percent of screen area classified correctly by the global entropy approach. Each test is represented by a separate bar, and the average is shown as bar 21 (96%).

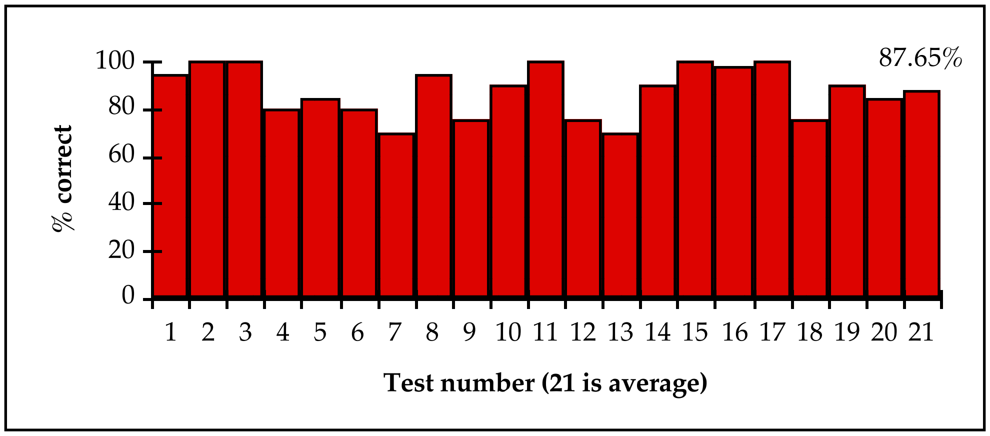


fig 22. Same test as above performed on the local entropy approach. The average correctness is 87.65%

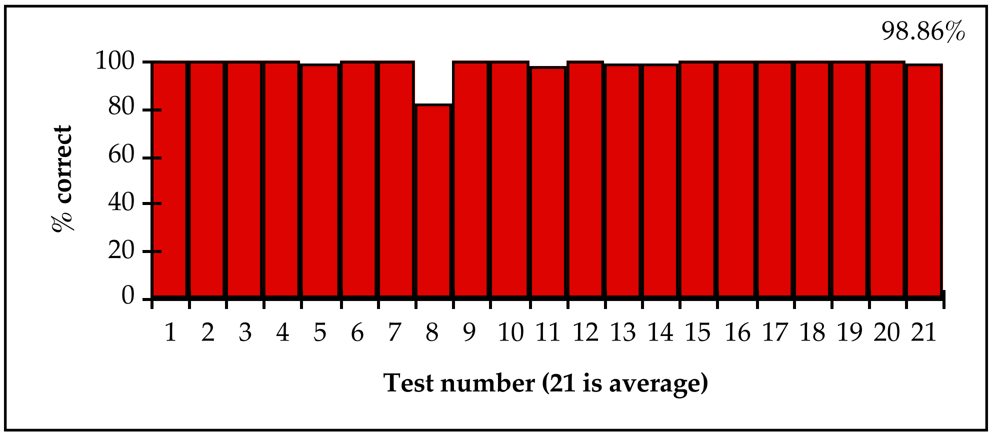


fig 23. Percent of screen area classified correctly by the global entropy approach if screen backgrounds are considered noisy when they are classified as such. Same format as above graphs, with an average of 98.86%

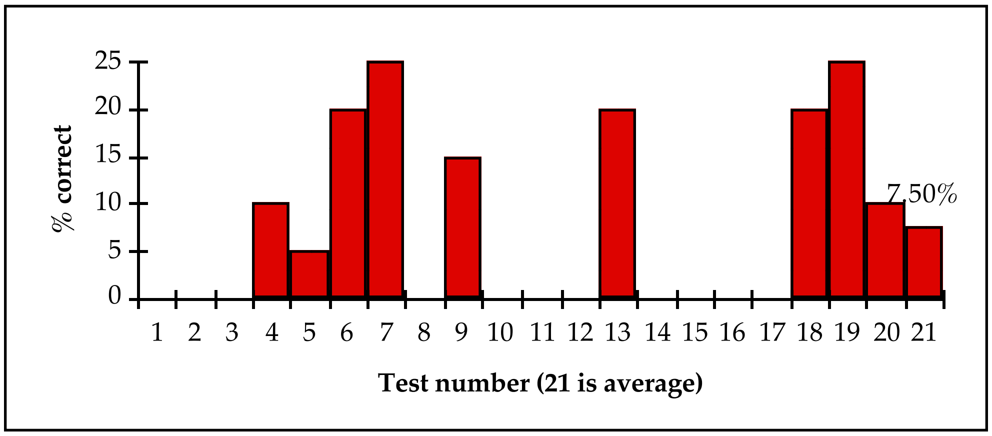


fig 24. Percent of noisy images correctly classified in the local entropy approach. Same format as above graphs, with an average of 7.5%.

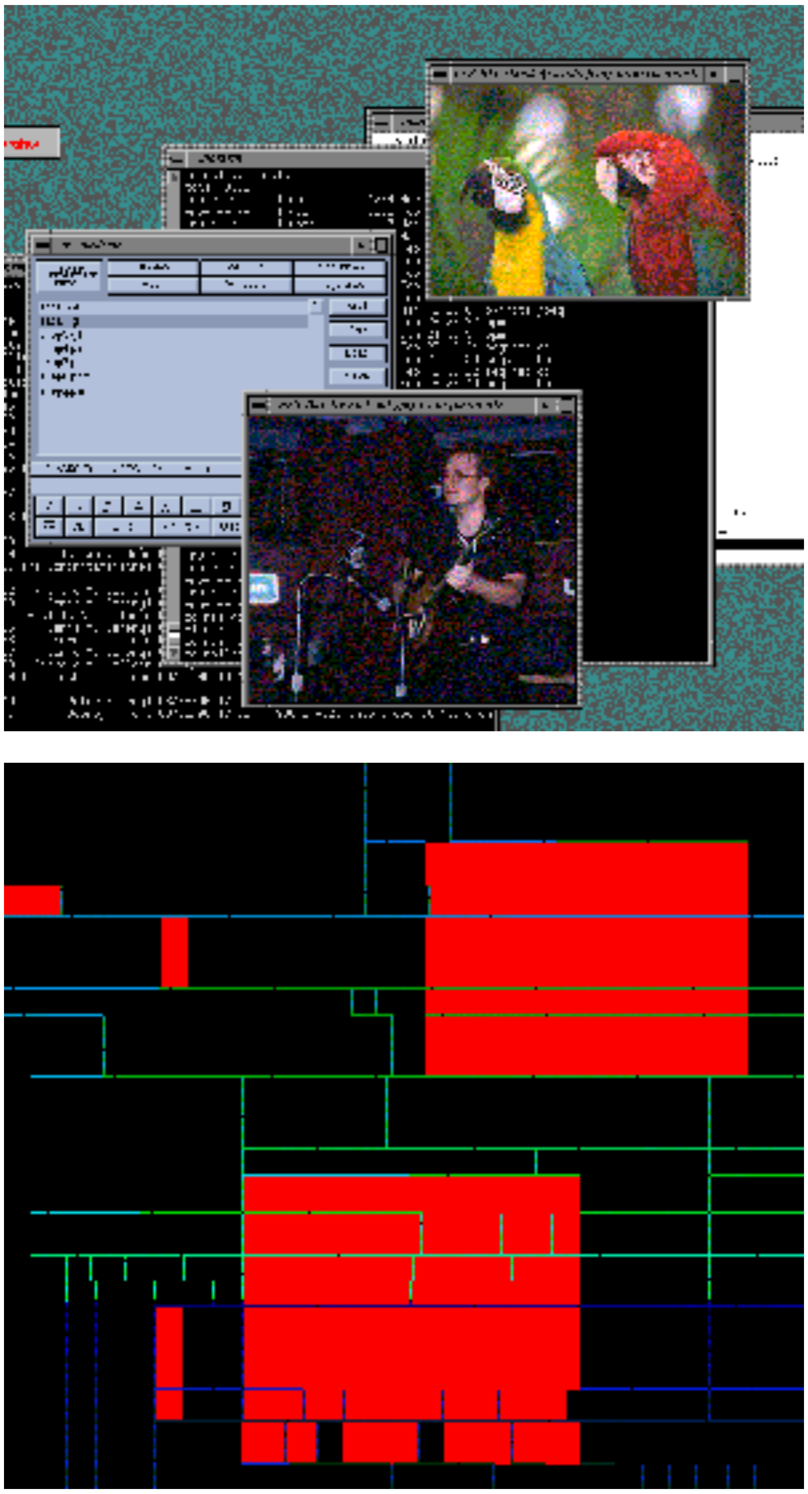


fig 25. Output of the global entropy region growing algorithm. Red rectangles correspond to noisy images, and black correspond to palletized images. The original image, shown above, clearly has two noisy images. In the output, these have been properly segmented. Note the fragmentation occurring in the palletized region, and some small, isolated misclassifications of palletized images as noisy.

V. Our Connected Components Approach to Text Identification

We modelled our connected components algorithm for identifying text in palletized images closely after Fletcher and Kasturi [17]. Like theirs, our algorithm consists of three phases: first, we identify all the connected components in the image. Then, based on a set of heuristics, we eliminate all non-text components. Finally, we group the individual components into text blocks.

Growing Connected Components

Through the clever use of data structures, we can accomplish the first step with one top-to-bottom linear pass through the region. Each component is described with a color, a bounding rectangle, and a list of "active points." If we are currently on row n of the region, then the active points for a component are the pixels that belong to it in row n-1. Because a component's pixels must be 8–connected to one another, if it has no active points, then there is no way it can grow anymore. Therefore, at row n we only need to be concerned with the components that have active points.

For each pixel along the current row, we see if it is 8–connected to any components of the same color by using their active points. If so, that component's bounding rectangle is extended to include row n. If not, then we create a new connected component for this pixel. We also need to be aware of special cases: for instance, a pixel could link together two connected components, combining them into one [fig. 26 illustrates this occurrence]. After repeating this process for every pixel along row n, and for all rows in the image, we are left with the bounding rectangles of every connected component. Because there are usually thousands in any sizeable image, an efficient implementation of the data structure is essential (we chose hash tables).

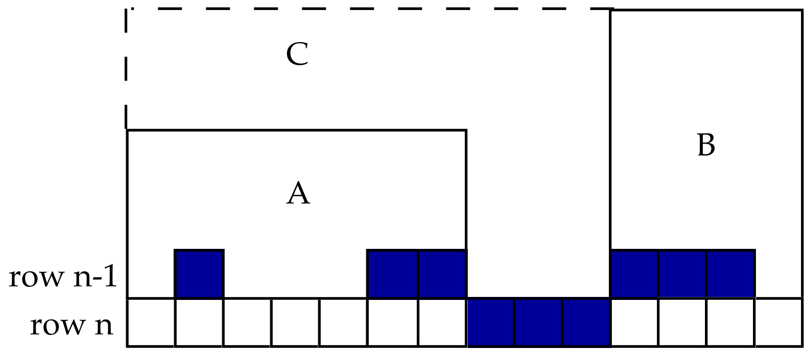


fig 26. Two components being combined. A and B are two connected components, with the large rectangles their bounding rectangles. The blue pixels in row n-1 are their active pixels. The group of blue pixels in row n are 8–connected to both A and B, so the two should be merged. This creates the new component C.

Eliminating Non-Text Components

The next step after building the components is to decide which ones are text. We do this by applying a set of heuristics. This is the most important and most difficult step of the algorithm: poor heuristics will lead directly to misclassifications. We believe, however, that the choices we made are sufficiently general so that only a small percentage will be ignored, but particular enough so that all palletized image components will be eliminated.

It is important to note that through establishing these heuristics we restricted ourselves to looking for large groups of text. One reason for this is simple necessity. If we are hoping to identify text components based on their shape alone rather than their proximity to others, then our algorithm will become as complex as OCR. It is also unlikely that identifying a totally isolated text character will improve compression ratios. Because text compression algorithms rely on repeated occurrences of the same pattern, chances are it would compress an isolated character exactly the same way an image compression algorithm would. Finally, because most characters occur as part of some kind of text document, even if we ignore isolated ones we will still be considering the vast majority of them.

The first assumption we made is that text components only come in a limited range of sizes. This is not actually true, of course: often text characters can be huge. However, in almost all cases, large characters do not appear in large groups. The physical constraint of image size makes such an occurrence impossible. And generally, in any situation where lots of text appears, such as a word processing document or e-mail message, text will usually be around 12-point. We decided, therefore, that we would only consider components larger than 3–by–3 and smaller than 24–by–24. Because we store the bounding rectangle of each component, it is simple to remove all that fall outside this range in one quick pass.

The second assumption is that text components appear in horizontal rows. Once again, this is not true in all cases: there are always situations where one or two characters appear in isolation. Text in screen images can also occur vertically or at other angles. But in most cases, large groups of text will be horizontal.

We grouped components into rows using two filters; the first is a vertical proximity test, the second is a ratio of areas. Two components can be in the same row only if the difference in Y-value of their center points is below a threshold. We normalize this threshold by the area of the components, making it more flexible for larger font sizes. Secondly, two components cannot be in the same row if the ratio of their areas is greater than 3:1. Given a font and font size, character size never varies significantly. So if the two are outside this bound, chances are they are not part of the same text block. Using these two tests, if we can find a grouping of at least seven components on one row, then all of them are considered text.

The final step tests horizontal proximity. Even if a component has the same Y-value as many others, it could be all the way on the left side of the image with all the others on the right. In this case, that one component is certainly not part of a text block and should be eliminated. We determine proximity by building up chains of nearby components: if more than seven border each other closely, then they are all text.

An illustration of the pruning process is shown below [fig. 27].

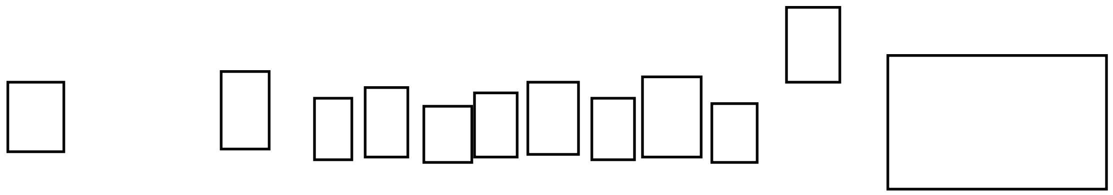
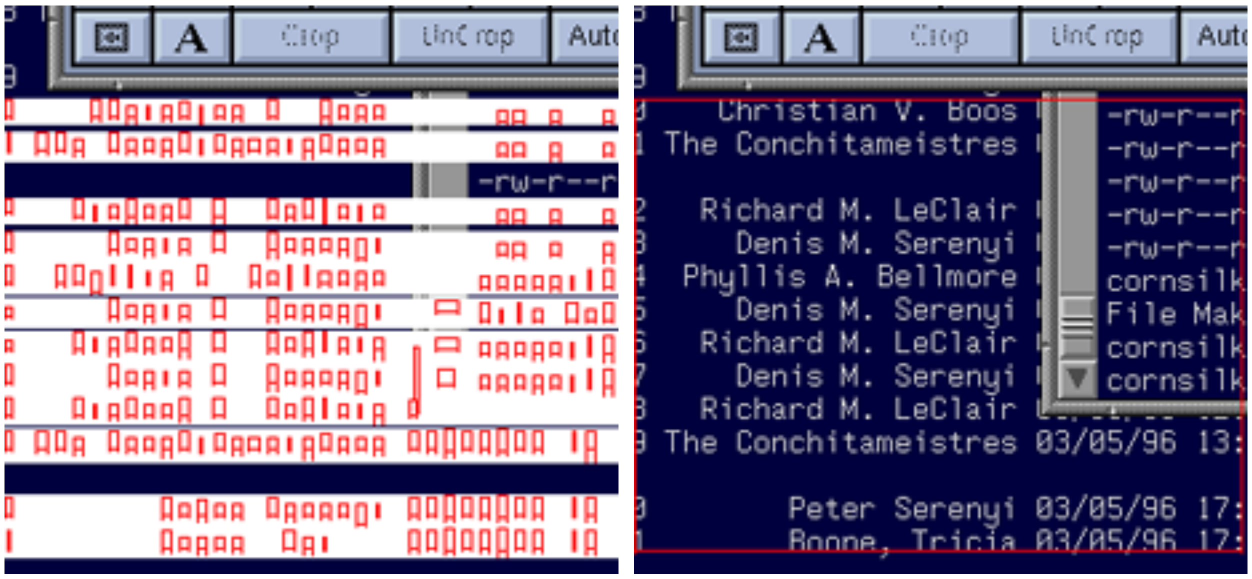


fig 27. Elimination of non-text components (marked with 'X'). The one on the right does not fit into the size constraint, the second from the right is not in the same row as the others, and finally the one on the left is not close enough to be within horizontal proximity of any others.

Grouping Components Into Text Blocks

Now that all non-text components have been eliminated, we only have to group the remaining ones into text blocks. This task is simple if they are sorted. Beginning with the upper-left hand corner, we simply combine any two components within a certain distance of one another. The bounding rectangle around the previous two becomes the bounding rectangle of the new component. As this process repeats, whole groups of separate text components will be merged into one. Isolated blocks remain separated, however, because they will never be within the distance threshold we have set. An example of this process is shown below [fig. 28].

This algorithm is fast but also ignores some possible scenarios. If a text region surrounds an image, for instance, the rectangle created might mistakenly include that image. Text that borders a palletized region in an L shape might also cause a misclassification. We believe, however, that in general text occurs in rectangular regions on screens, so in most cases no errors will occur.



```
   Christian V. Boos      -rw-r--r
 The Conchitameistres     -rw-r--r
                          -rw-r--r
   Richard M. LeClair     -rw-r--r
     Denis M. Serenyi     -rw-r--r
  Phyllis A. Bellmore     cornsilk
     Denis M. Serenyi     File Mak
   Richard M. LeClair     cornsilk
     Denis M. Serenyi     cornsilk
   Richard M. LeClair
 The Conchitameistres 03/05/96 13:

        Peter Serenyi 03/05/96 17:
        Boone, Tricia 03/05/96 17:
```

fig 28. Connected components in action. In the picture on the right, you can see a piece of a screen image with a large amount of text. The red rectangle surrounding the text corresponds to the region where text was identified. The image on the left shows the raw connected components. At this step, all the non-text components have been removed.

Experiments

We used the same 20 images employed in previous tests for our connected components algorithm. All but one contain some text; many only have text in isolated areas, such as underneath file icons and in window title bars. Others have large text regions, either as a word processing document, e-mail, or a web page. All in all, we feel that the text in our tests is an accurate reflection of where it appears in most screen images. Because we are only aiming to identify text in palletized regions, the image was first processed by the bottom-up global entropy algorithm. The noisy regions were then blacked out so they would be ignored.

We measured performance based on the following set of questions:

- Of the total number of text characters on the screen, what percentage were classified as noisy? What percentage occurred in large groups? What percentage occurred in isolated areas?
- Of the text in large groups, what percentage was classified as text? Of the text in isolated areas, what percentage was classified as text?
- What percentage of the text blocks actually consisted of text regions?

The first set of questions, rather than evaluating the performance of the algorithm, measure the accuracy of our initial assumptions. Were we correct in believing that only a marginal amount of text occurs in noisy regions? Were we correct in believing that most text occurs in large groups? The second and third set of questions measures the algorithm's performance. We are hoping that all the large groups of text are identified, and that a minimal number of palletized images are misclassified. In all of these tests, so as not to unfairly weight an image with only a minuscule number of characters on it, we computed averages in terms of total number of characters.

Experimental Results

Our assumptions about text in screen images were correct: most of it appeared in large groups within palletized regions. Overall, a mere 1% of text was segmented as noisy, and 11.7% appeared in isolation [fig. 30]. The former usually resulted from a misclassification by the bottom-up algorithm; only very rarely did a couple of characters actually appear in a noisy image. Text in isolation, as expected, consisted mostly of title bars, pull-down menus, and icons. Even though less than 50% of this was identified by our algorithm, it is unlikely that an isolated word would compress better if classified as text rather than palletized image.

Although the algorithm did not perform as well as we had hoped, it was able to identify 89.96% of the text occurring in large groups. In general, the second step correctly eliminated all non-text components, while leaving the text alone. This can be attributed to our good choice of heuristics. Text components were similar in size, occurred in rows, and were within a short distance of one another. Non-text components, on the other hand, followed much more random patterns. The following thresholds yielded the best results: for grouping text into rows, we used a Y-distance of 3 pixels and a horizontal proximity of 10 pixels [see above for details on how these function]. For grouping components into text blocks, we used a distance of 25 pixels.

The algorithm's major problem was mistaking palletized images for text. On average, only 64.5% of text block area actually contained text [fig. 31]. In most cases, this did not happen because non-text components were being misclassified. Rather, the problem stemmed from the last stage of the algorithm, when individual characters are combined into regions. In many of our test images, especially those that included web pages, text occurred in non-rectangular blocks. When these characters were grouped together, a bounding rectangle was placed around them which would necessarily include surrounding areas of palletized image.

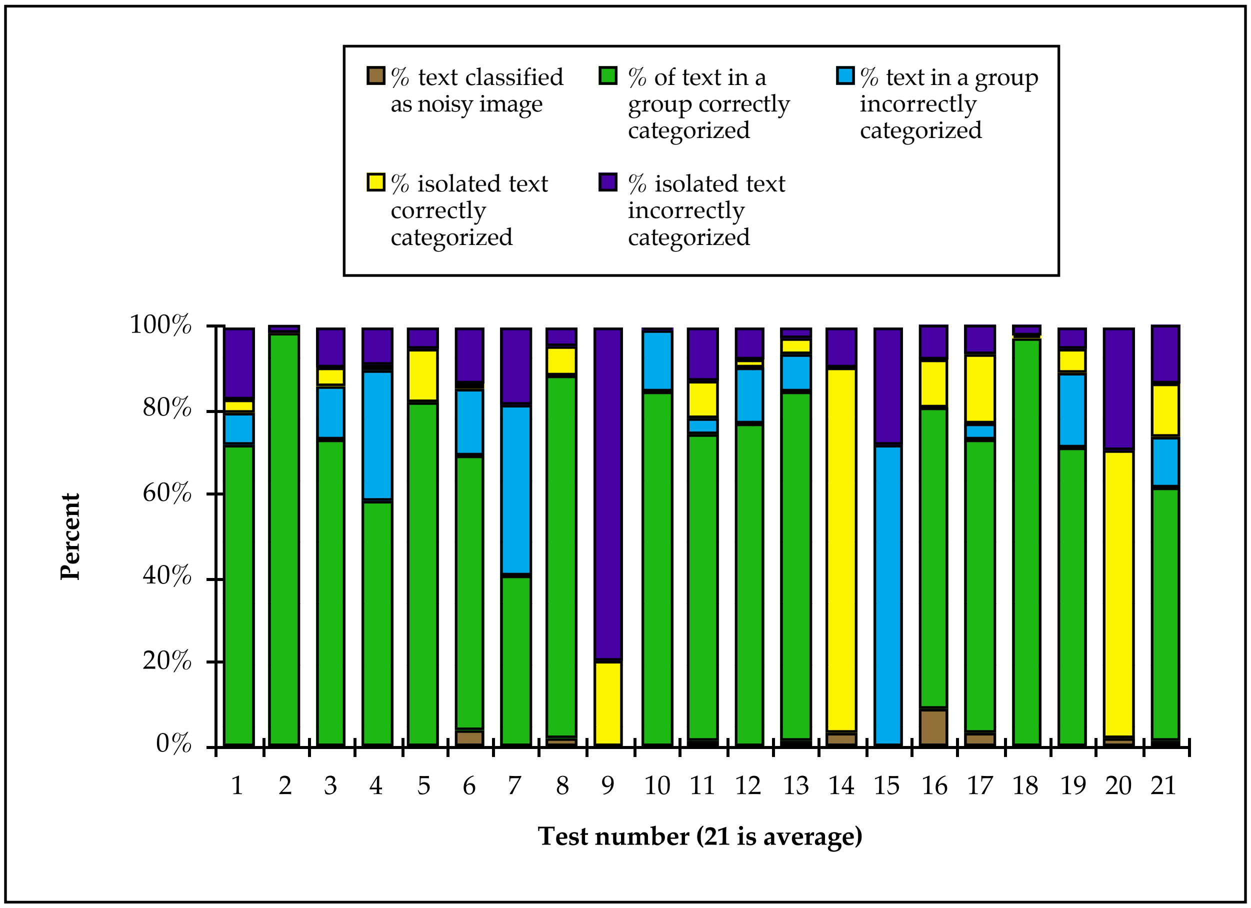


fig 29. Combined text extraction results. Here we see the total breakdown of how text in our twenty test images was classified: each color represents the percentage of text characters placed in one of five possible categories. The occasional brown bar on the bottom represents text classified as a noisy image. The large green bars represent the percentage of text occuring in large groups, such as in a word processing document, that was properly identified. The blue bars represent text occuring in large groups that was incorrectly classified. The yellow bars represent the percentage of text occuring in isolation, such as a window title or icon name, that was properly identified. Finally, the purple bars represent the percentage of text occurring in isolation that was incorrectly classified. The average of the twenty tests is column 21.

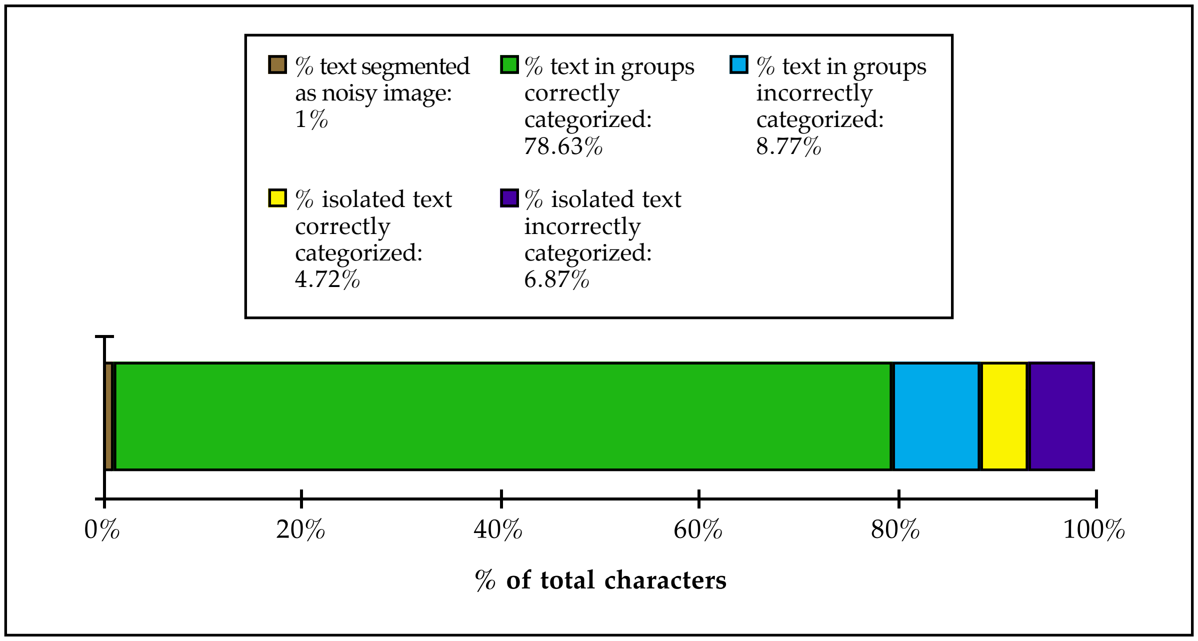


fig 30. The data in fig. 29 does not consider that the number of characters varies greatly from test to test. Here are the results based on total number of characters over all twenty tests.

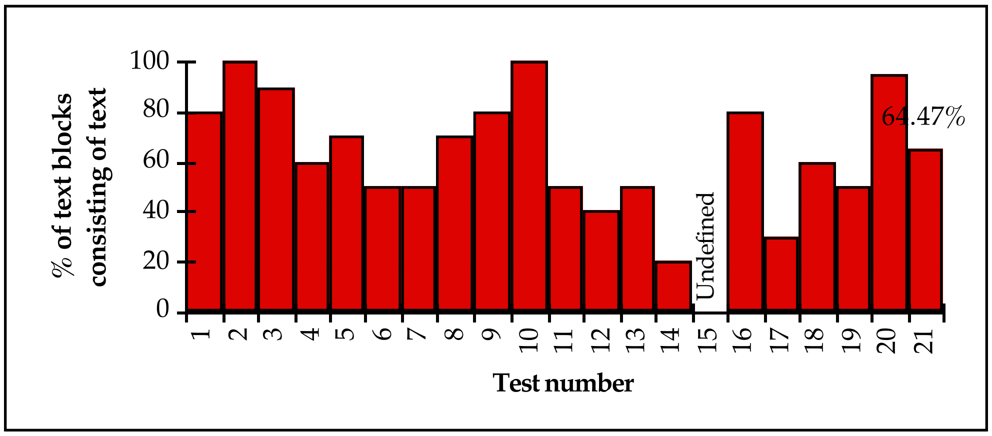


fig 31. Percent of text blocks actually consisting of text: the average over all tests is 64.47%. Test 15 is undefined because there was no text in that image.

VI. Conclusion

We have just described a successful algorithm for segmenting screen images and classifying individual regions as noisy images, palletized images, and text. Of the various methods tried for separating noisy images from palletized images, the bottom-up "global entropy" algorithm performed the best, achieving a mean correctness rate of 98.86%. Although the connected components algorithm proposed for separating text and palletized images did not perform as well, it identified a reasonable 89.9% of the large groups of text in screen images. Its major problem, misclassifying palletized images, can hopefully be fixed with a minor tuning of the algorithm.

Though this method is not perfect, we believe that it will be able to significantly improve compression ratios for screen images. Instead of compressing a number of widely differing image types with one compression algorithm, it is now possible to identify noisy images, palletized images, and text, and send these regions individually to customized compression routines. With the improved results, the Generic Imaging Terminal described above will hopefully be able to achieve high frame rates even over low-bandwidth networks.

Bibliography

1. M.E. Jernigan and F. D'Astous, "Entropy Based Texture Analysis in the Spatial Frequency Domain," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 4[2], 1984. pp. 237–243.

2. Haluk Derin and Howard Elliot, "Modeling and Segmentation of Noisy and Textured Images Using Gibbs Random Fields," IEEE Transactions on Pattern Analysis and Machine Intelligence, Jan. 1987, pp. 39–55.

3. N. K Huang, "Markov Model for Image Segmentation," 22nd Allerton Conference on Communication, Control, and Computing, October 3–5, 1984, pp. 775–781.

4. J. E. Bevington and R. M. Mersereau, "A Random Field Model Based Algorithm for Textured Image Segmentation," Proceedings of the 3rd European Signal Processing Conference, September 2–5, 1986, pp. 909–912.

5. N. K. Huang and X. Gong, "Textured Image Recognition Using the Hidden Markov Model," Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, April 11–14, 1988, pp. 1128–1131.

6. X. Gong and N. K. Huang, "Texture Segmentation Using Iterative Estimate of Energy States," Proceedings, 9th International Conference on Pattern Recognition, November 14–17, 1988, pp. 51–55.

7. R. N. Sutton and F. L. Hall, "Texture Measures for Automatic Classification of Pulmonary Disease," IEEE Transactions on Computers, Vol. 21, 1972, pp. 667–676.

8. J. S. Weszka, C. R. Dyer, and A. Rosenfeld, "A Comparative Study of Texture Measures for Terrain Classification," IEEE Transactions on Systems, Man, and Cybernetics, Vol. 6[4], 1976, pp. 269–285.

9. Aakash Tagi and Magdy A. Bayoumi, "Image Segmentation On a 2-D Array by a Directed Split and Merge Procedure, " IEEE transactions on Signal Processing, Vol. 40[11], November 1992, pp. 2804–2813.

10. Xiaolin Wu, "Image Coding by Adaptive Tree-Structured Segmentation," IEEE Transactions on Information Theory, Vol. 38[6], 1992, pp. 1755–1767.

11. T. Gevers and K. Kajcovski, "Image Segmentation by Directed Region Subdivision," Proceedings, 12th International Conference on Pattern Recognition, October 9–13, 1994, Vol. 1, pp. 342–346.

12. Patrick C. Chen and Theodosios Pavlidis, “Segmentation by Texture Using a Co-Occurrence Matrix and a Split and Merge Algorithm,” Computer Graphics and Image Processing, Vol. 10, 1979, pp. 172–182.

13. Trygve Randen and John Hakon Husoy, “Multichannel Filtering for Image Texture Segmentation,” Optical Engineering, Vol. 33[8], 1994, pp. 2617–2625.

14. Theo Pavlidis and Jiangying Zhou, “Page Segmentation and Classification,” Computer Vision, Graphics, and Image Processing, Vol. 54[6], pp 484–496, 1992.

15. Naoki Kuwata, Yasuhiko Murayama and Shoichi Ilno, “A New Bi-Level Quantizing Method for Document Images,” IEEE Transactions on Consumer Electronics, Vol. 38[3], pp. 718–724, 1992.

16. J. Toyoda, Y. Noguchi, and Y. Nishimura, “Study of Extracting Japanese Newspaper Article,” 6th International Conference on Pattern Recognition, October 1982, pp. 1113–1115.

17. Lloyd Alan Fletcher and Rangachar Kasturi, “A Robust Algorithm for Text String Separation from Mixed Text / Graphics Images,” IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 10[6], pp. 910–918, 1988.

18. Henry S. Baird, Susan E. Jones, and Steven J. Fortune, “Image Segmentation by Shape-Directed Covers,” Proceedings of the 10th International Conference on Pattern Recognition, June 1990, pp. 820–825.

19. Hiromichi Fujisawa, Yasuaki Nakano, and Kiyomichi Kurino, “Segmentation Methods for Character Recognition: From Segmentation to Document Structure Analysis,” Proceedings of the IEEE, Vol. 80, July 1992, pp. 1079–1092.

20. Todd R. Reed and J. M. Hans Du Buf, “A Review of Recent Texture Segmentation and Feature Extraction Techniques,” Computer Vision, Graphics, and Image Processing: Image Understanding, Vol. 57[3], pp 359–372, 1993.

21. Rolf Adams and Leanne Bischof, “Seeded Region Growing,” IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 16[6], pp. 641–647, 1994.

22. Qin Zhang and John M. Danskin, “Entropy-Based Pattern Matching for Document Image Compression,” IEEE International Conference on Image Processing, September 16–19, 1996.