Large Block Inpainting by Color Continuation Analysis

Timothy K. Shih, Rong-Chi Chang, Liang-Chen Lu, and Louis H. Lin
Multimedia Information Network Lab
Department of Computer Science and Information Engineering
Tamkang University, Taiwan, R.O.C.
E-mail: tshih@cs.tku.edu.tw

ABSTRACT

Automatic inpainting is a mechanism which repairs damaged pictures using an approximation mechanism. The most difficult problem is to inpaint a large damaged area, without knowing its content. One possible solution is to use color interpolation or extrapolation on surrounding pixels. However, spatial characteristics such as edges and pixel continuations are hard to be restored. In this research, we propose a series of automatic algorithms, which is based on an analysis of color continuations. Large damaged blocks are repaired, before the rest smaller potions are repaired by a multi-resolution inpainting algorithm. The mechanism is tested on more than 2000 images, including cartoon drawing, photos, Chinese painting, and western painting. Our results prove that, the proposed automatic mechanism fixes damaged image up to a certain degree of satisfaction from the users. The demonstration of our work is available at: http://www.mine.tku.edu.tw/demos/inpaint.

Key words: digital inpainting, image restoration, large block inpainting, multi-resolution inpainting

1. Introduction

Automatic digital inpainting is a technique which restores damaged image or video by means of image interpolation. The technique can be used in photo restoration, zooming, and even image coding. Current techniques may base on the extrapolation of neighboring pixels, recovery of edges, curvature-driven diffusions (according to the connectivity principle in vision psychology) [3], and inpainting from multiple view points (i.e., image from movie, or image from different time and view point). A fast inpainting algorithm was proposed in [4]. Efficiency of the proposed method [4] is two to three orders of magnitude faster than those using partial differential equations. In addition to inpaint damaged pictures, the work presented in [1] can automatically inpaint a user-selected region, by using surrounding information. Another paper [2] takes the ideas from classical fluid dynamics to propagate isophote lines. Inpainting can be used in attacking a visible watermark [5]. The system discussed in [5] allows users to select a watermark area, and to produce an approximation to the original picture.

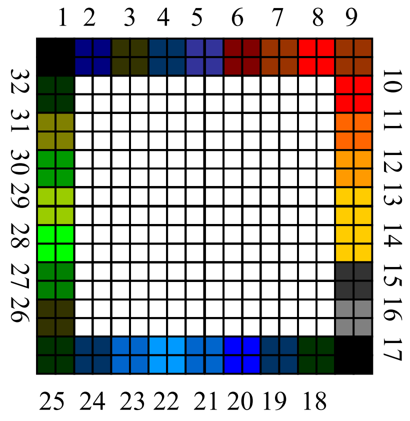
One of the most challenge problems of automatic inpainting is when there is only little or no information that can support pixel restoration of large damages areas. Most extrapolation or interpolation algorithms rely on surrounding pixels. However, if too much information is lost, automatic mechanism is impossible to recover all details. In this situation, user knowledge should be used by meanings of some drawing tools. In some cases, texture characteristics can be applied in restoration. However, if a picture contains important spatial characteristics, such as the horizontal line of the edge of a building, it is hard to restore vectors automatically.

We propose algorithms which take surrounding pixels, and compute possible color continuations. We also propose another method and an evaluation mechanism, based on different level of details of still image. We claim that, if an image region is seriously damaged, it is not realistic to rely on the extrapolation of neighboring pixels in any method. Instead, global information should be used. In addition, if the variance of pixel colors is large in an image block, it is possible that the block contain detailed shapes. Thus, a multi-resolution strategy should be considered. We present our algorithm and an evaluation mechanism. Analysis is given based on a test of more than 1000 pictures, with randomly generated noise. We conclude that, our algorithm consider different levels of details. And, the overall efficiency is higher then others using a singular resolution approach.

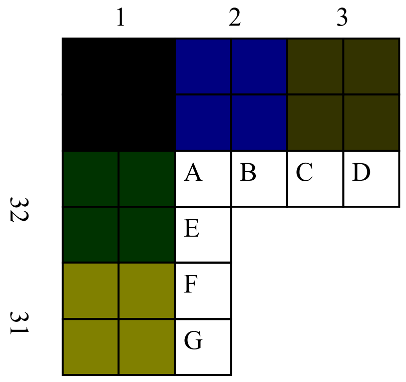
We deal with the special cases first in section 2. The main inpainting algorithm is presented in section 3, followed by an evaluation mechanism (section 4). We tested more than 2000 images of different kinds. Analysis is given in section 5 before the conclusion is presented.

2. Inpaint Large Damaged Blocks

When dealing with inpainting of large damaged blocks, a simple algorithm of color interpolation results badly. To cope with this problem, special treatment is necessary. We propose two recursive algorithms, which estimate color continuations of surrounding pixels with respect to a large damaged block and progressively inpaints the block by averaging colors. The algorithm is able to guess edge vectors which preserve shapes up to a degree of satisfaction. The difference between these two algorithms is on the definition of a mapping function, which decides colors for inpainting.



(a) Surrounding Pixel Groups



(b) Color Mappings

Figure 1: Surrounding Pixel Groups and Mapping

The fundamental concept of the algorithms assumes that a block (consists of *inpainting pixels*) to be filled is at least surrounded by *n* levels of *surrounding pixels*. In a boundary case, duplicated surrounding pixels can be added to ease computation. In figure 1(a), *n* is set to *2* for illustration. Inpainting pixels are shown in white and surrounding pixels are shown in non-white colors. Surrounding pixels are grouped into *n* by *n* groups, which are numbered from *1* to *32*. Each of these groups has a mean color *M*. Mean colors are used to estimate color continuations. The threshold of mean color differences, δ, depends on the color space used. For instance, in the HSV color space, the difference can rely on the value of hue. This threshold can be computed from the color variation of the entire picture. We present two recursive algorithms here. The first algorithm uses mean colors of surrounding pixel groups. It is easy to implement and runs faster. The second algorithm is CPU time consuming but has a better result.

Algorithm 1: Brute Force Color Extrapolation

Step 1: Inpaint: For each outer most inpainting pixel, inpaint according to the following mapping:

$f(A) = 1, f(B) = 2, f(C) = 3, f(D) = 3$
$f(E) = 32, f(F) = 31, f(G) = 31$

based on the illustrations in figure 1(b), where A to G are inpainting pixels, and numbers represent surrounding pixel groups. Other mappings can be computed in a similar manner. The mean colors of groups are used to inpaint the outer most inpainting pixels.

Step 2: Recursion: Shrink the inpainting pixel block by one level (i.e., removes the outer most pixels), and recursively call the algorithm until the size of inpainting pixel block is equal to 1 by 1 (or 0 by 0). Inpaint the last pixel (if any) by the mean of *M*s of the four surrounding pixel groups.

The disadvantage of the first algorithm includes error propagation. One of the solutions to solve this problem is to cut and paste surrounding pixel blocks, with a randomized control. However, the result fails to preserve special characteristics such as edges and gradient.

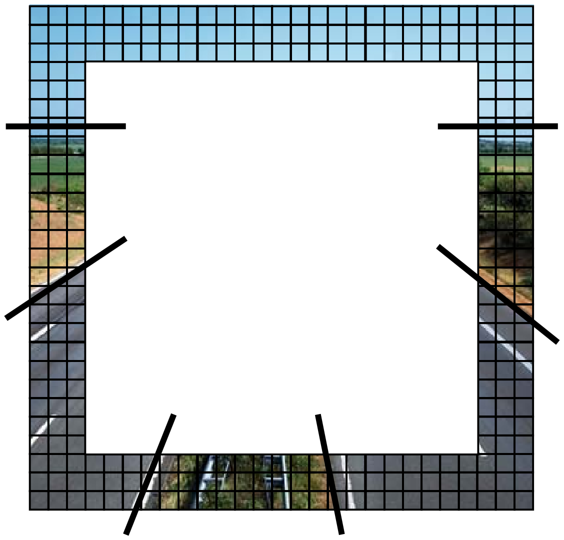
Algorithm 2: Color Extrapolation by Continuation

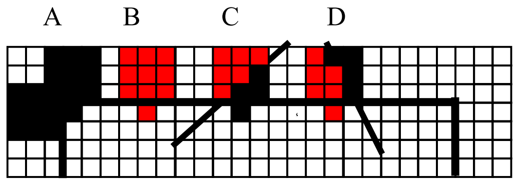
Step 1: Calculate mean color continuation: Subdivide all surrounding pixel groups into a number of group series, based on the similarity of mean colors, and the threshold δ. A surrounding pixel group at the boundary of a series is called a *boundary pixel group*.

Step 2: Inpaint: For each outer most inpainting pixel, inpaint according to the following mapping defined in table 1.

Step 3: Recursion: Shrink the inpainting pixel block by one level (i.e., removes the outer most pixels), and recursively call the algorithm until the size of inpainting pixel block is equal to 1 by 1 (or 0 by 0). Inpaint the last pixel (if any) by the mean of *M*s of the four surrounding pixel groups.

The mapping function for algorithm 2 is based on the definition of two patterns. In figure 3, pattern A and pattern B are used as the sources of inpainting color computation, for a pixel in a corner and a pixel not in a corner, respectively. Both of these patterns has a consist color of pixels (i.e., A and B), or multiple colors (i.e., C and D). A group of pixels are consistent if all pixels have their colors differ in at most δ (the threshold of color difference). Usually, color differences large then the threshold occur in boundary pixel groups. Table 1 summarizes the pre-image and image of a mapping function, which finds inpainting colors. The second algorithm relies on the mapping, to find a suitable color for each inpainting pixel.


(a) Separation of Color Continuations



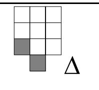
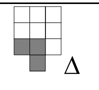
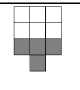
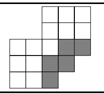
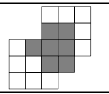
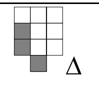
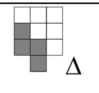
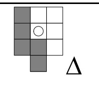
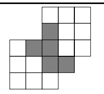
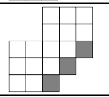
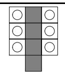
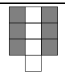
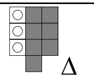
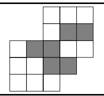
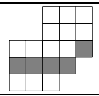
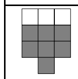
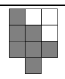
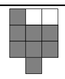
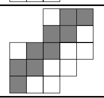
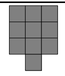
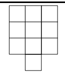
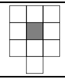
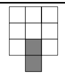
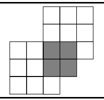
(b) Patterns of Inpainting Colors

Figure 3: Color continuation of surrounding pixels and Patterns for inpainting color computation

Since table 1 is designed according to human perception on the boundary of color changes, luckily, it is possible to preserve spatial characteristics such as edges and gradient.

The initial step is to handle large damaged area. However, in a typical damaged picture, damaged areas are by scratch lines, inks, and other small effects. Frankly speaking, if the damaged area is too large, too much information is lost. It is impossible to automatically restore the damage. However, for damages of small regions, automatic restoration is possible, as we should discuss in the next section.

Table 1: Color Mapping Function of Algorithm 2

	Boundary			Corner	
					
					
					
					
					
default					

3. Inpaint by Color Interpolation

To restore damages areas automatically, we design a recursive algorithm. The algorithm works as the following. Let **DIB** be a damaged image block. We subdivide **DIB** into n by n image blocks (i.e., **IB**s). The default value of n is set to 16. There are three adjustable thresholds used: $\boldsymbol{\alpha}$ is a threshold of variance of pixel colors, and $\boldsymbol{\beta_1}$ and $\boldsymbol{\beta_2}$ are the thresholds of percentages. We assume that, an image block has $i*j$ pixels, and a color variance, var, can be calculated as

$$\bar{x} = \frac{\sum_{\forall i}\sum_{\forall j} x_{ij}}{i \times j}, \quad \text{var} = \Big(\sum_{\forall i}\sum_{\forall j}(x_{ij} - \bar{x})^2\Big)/(i \times j - 1),$$

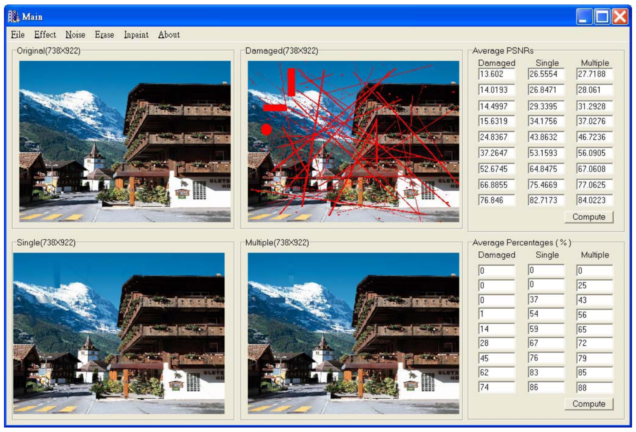
where $\bar{x}$ is the average of pixel color. Color variance has a strong indication of the degree of details in an **IB**. The threshold $\boldsymbol{\alpha}$ sets the criterion of whether a multi-resolution inpainting is required. In our implementation, the value of $\boldsymbol{\alpha}$ is a percentage in the range between 0 and 100 (the maximum var) of an **IB**. Another criterion is the percentage of potential damaged pixels. We argue that, if the percentage is too high, using surrounding color information to fix a pixel is less realistic as compared to using a global average color. In some severe cases, it is impossible to use neighborhood colors. Note that, both thresholds are adjustable for the sake of analysis. The recursive algorithm iterates through each of the **IB**s in a **DIB**. If the color variance of IB is below the threshold $\boldsymbol{\alpha}$, there is not much difference of pixels in IB. No subdivision is required (i.e., no need of looking at the next level of details). Thus, the algorithm further divides **IB** into several pixel blocks (i.e., **PB**s). If the percentage of damaged pixels in a **PB** is too high (i.e., greater than $\boldsymbol{\beta_2}$), the mean color of **IB** is used. One example is that the entire **PB** is damaged (thus we need to use the mean color of **IB**). Alternatively, if the percentage is still high (i.e., greater than β_1), the mean color of **PB** is used. Note that, the computation of mean colors does not take damaged pixels into the account. If the percentage is low, neighbor pixels are used for inpainting. Finally, if the color variance of **IB** is not below the threshold α, the algorithm is called recursively to handle the next level of details. The following algorithm is implemented on MS Windows to test our justification.

Algorithm 3: Inpainting by Color Interpolation

```
Let DIB be a damaged image block
Let $\alpha$ be a threshold of variance
Let $\beta_1$, $\beta_2$ be a threshold of percentage, $\beta_1 < \beta_2$

Algorithm inPaint(block DIB)
  use color extrapolation inpainting in large damaged blocks
  if DIB is a small block then return
  divide DIB into $n*n$ image blocks
  for each image block IB
    let var be the color variance of IB
    let Mcolor be the mean color of IB
      if $var < \alpha$ then
        {
        let PB be an $x*y$ pixel block in IB
    let Ncolor be the mean color of PB
        for each PB in the image block
          {
          if the percentage of damaged pixels in PB > $\beta_2$
            inpaint the damaged pixels using Mcolor
     else if the percentage of damaged pixels in PB > $\beta_1$
            inpaint the damaged pixels using Ncolor
          else
            inpaint the damaged pixels using neighbor pixels
          }
        for each pixel in the boundary of each PB
          smooth boundary pixels using neighbor pixels
        }
      else
        call inPaint(IB)
```

We use this algorithm to design a simple inpainting tool (see figure 4), the tool allows one to load a picture, and to damage the picture on purpose (by using line, simple graphics object, spray, and even randomly generated noises). A naïve single-resolution inpainting function and our multi-resolution inpainting function discussed above are both implemented. The damaged picture and the two inpainted pictures are compared with the original picture to obtain multi-level evaluation values (shown on the right side). The tool is used to test the efficiency of the algorithm. Interested readers can download the tool from our demo Web site. However, to evaluate the results, we also implemented a batch process system, which reads 1000 bit-mapped pictures, and produces a line chart using MATLAB. We discuss our evaluation strategy in the next section. For the interests of the reader, figure 5 shows a damaged picture, a result from single resolution inpainting, and another from multi-resolution inpainting. Note that, the result from single resolution inpainting in figure 5 has a strip from the sky to the mountain area. The error effect is less in the result of multi-resolution inpainting. We have 1000 sets of results available on the demo Web. Interested readers should check these results.



Average PSNRs

Damaged	Single	Multiple
13.602	26.5554	27.7188
14.0193	26.8471	28.061
14.4997	29.3395	31.2928
15.6319	34.1756	37.0276
24.8367	43.8632	46.7236
37.2647	53.1593	56.0905
52.6745	64.8475	67.0608
66.8855	75.4669	77.0625
76.846	82.7173	84.0223

Average Percentages (%)

Damaged	Single	Multiple
0	0	0
0	0	25
0	37	43
1	54	56
14	59	65
28	67	72
45	76	79
62	83	85
74	86	88

Figure 4: A multi-resolution inpainting tool

4. The Evaluation Procedure

In a practical situation, it is impossible to compare a damaged picture with its original. However, the strategy of our evaluation is based on the following assumption. Suppose that there are two copies of the original picture. The first is damaged, and we will like to recover the damage as much as possible such that the inpainted picture will look almost the same as the second original. Thus, the evaluation of an inpainting algorithm can take in to account the PSNR value of the damaged picture, and the PSNR value of the inpainted picture. However, using a single PSNR value of each picture fails to analyze the details. It is possible that, an important small portion of the inpainted picture is not well-repaired. But, the overall PSNR value of the picture is still high. Thus, we use a set of multilevel PSNR values as well as the concept of "the percentage of good picture portions."

We decompose a picture using a complete quad-tree representation. Each level of decomposition equally divides an area into 4 quadrants. The number of levels depends on the actual size of a picture. As a practical situation, we decompose pictures up to 8 levels for evaluation. Note that, this decomposition layout is different from the multi-resolution inpainting algorithm, in order to test the results from a different perspective. Let n be the level number (from *1* to a maximum value). The average PSNR value at each level can be computed as:

$$PSNR_n = \sum_{\forall i \forall j} \left(\frac{PSNR_n^{i,j}}{4^{n-1}} \right),$$

where $PSNR_n^{i,j}$ is the PSNR value of the decomposed block$_{i,j}$ at level n. And, PSNR$_1$ is the PSNR value of the entire picture. According to the PSNR function, the average PSNR value at level $n+1$ is equal to the PSNR value at level n. This can be proved mathematically. Or, comparing one blank picture to another with a uniform distribution of noise will prove the assumption. We have tested both approaches. Note that, the decomposition of PSNR values is to test the quality of inpainting in the next level of detail. It is interesting to know that, in general, the average PSNR values at level $n+1$ are higher then the PSNR values at level n. The PSNR values are used to compare different inpainting algorithms on an equal base. In addition, we count the percentage of areas with their PSNR values greater than or equal to 30dB, w. r. t. each level of decomposition. In general, the value (i.e., 30dB) denotes a good picture quality. The testing tool shows PSNR values and percentages at each level (see figure 4).

There are three thresholds in the multi-resolution inpainting algorithm, $\boldsymbol{\alpha}$, $\boldsymbol{\beta_1}$ and $\boldsymbol{\beta_2}$. We use all combinations of the following values:

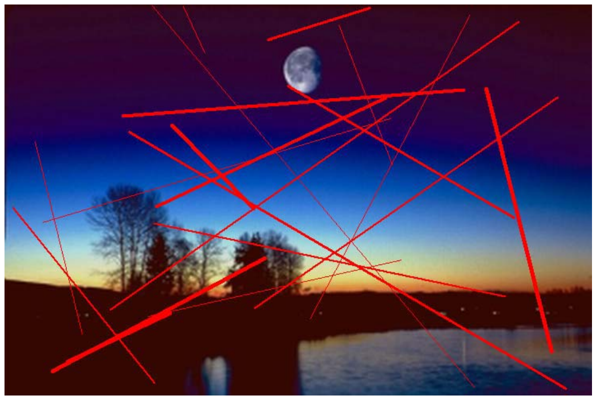
$\boldsymbol{\alpha}$ = 50%, 60%, 70%, 80%
$\boldsymbol{\beta_1}$ = 60%, 65%, 70%, 75%, 80%, 85%, 90%
$\boldsymbol{\beta_2}$ = 95%

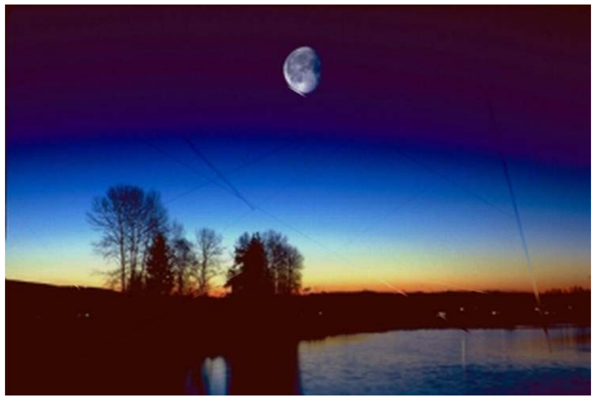
The selection of $\boldsymbol{\beta_2}$ is to test the usage of *Mcolor* (i.e., the mean color of the outside image block). Unless a pixel block is seriously damaged, otherwise, *Mcolor* should not be used. Thus, the selection of $\boldsymbol{\beta_2}$ should be high. Since $\boldsymbol{\beta_1}<\boldsymbol{\beta_2}$, we select the values of $\boldsymbol{\beta_1}$ accordingly. The threshold $\boldsymbol{\alpha}$ is to check the variance. We try to cover a wide spectrum. We run through the above combinations for 1000 bit-mapped image. The result is discussed in the next section.

5. Analysis

The values of $\boldsymbol{\alpha}$, $\boldsymbol{\beta_1}$ and $\boldsymbol{\beta_2}$ show a great impact to the outcome. In general, if $\boldsymbol{\alpha}$ is less than 70, the average PSNR values at a higher level is about the same as the single resolution decomposition. In table 2, we give the results of 2 runs, with $\boldsymbol{\alpha}$= 80. The value of $\boldsymbol{\beta_2}$ should be higher than $\boldsymbol{\beta_1}$. We chose $\boldsymbol{\beta_2}$ = 95 through our analysis. This means that unless the percentage of damaged pixels in a pixel block is higher than 95, the mean color of an outside big block should not be used. The value of $\boldsymbol{\beta_1}$ is critical. If $\boldsymbol{\beta_1}$ is less than 60, the result is not as good as expected. Table 2 also shows that both the PSNR values and the area of good image by our multi-resolution algorithm are better than the single resolution approach in general.

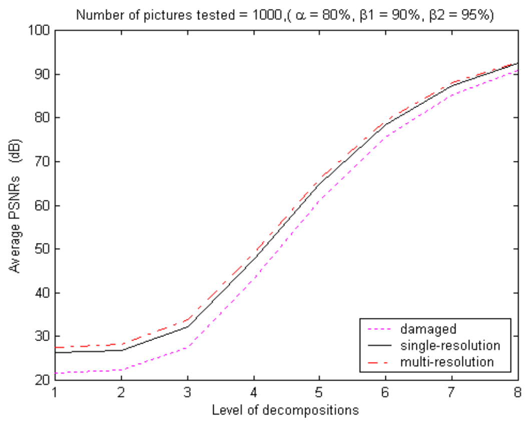
(Original Picture)


(Damaged Picture)


(Single Resolution Inpainting)

(Multi-Resolution Inpainting)
Figure 5: A sample test set

The values of PSNR become high as the level number increases. This is due to the fact that, the percentages of noise in a picture is relatively low. In some tests, if the entire picture is covered by noise, the PSNR values increase slowly when the level number increases.



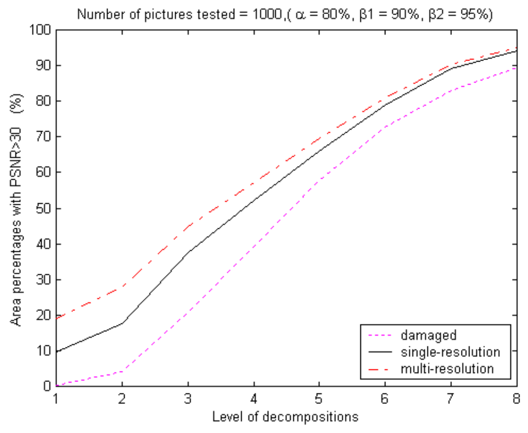


Figure 6: Results from 1000 pictures

Figure 6 shows the result from 1000 24-bit BMP images. The overall performance of multi-resolution image inpainting is better than the single resolution approach, if a set of parameters is carefully chosen (i.e., α= 80, β_1= 90, and β_2= 95). One of the important contributions of multi-resolution image inpainting is the prevention of error propagation, which is encapsulated inside a block. However, the disadvantage is that the multi-resolution approach does not look at the picture from a global view. Discontinuity occurs due to block subdivision. We are working on a dynamic resolution scheme to cope with this drawback.

6. Conclusions

We are currently working on transferring our technology to the industry. A few issues still need to be resolved. A friendly interface will allow users to mark the portion of damaged pictures, before the system can inpaint and print the pictures. The prototype only takes 24-bit BMP images for now. We also need to incorporate more off-the-shelf graphics formats. Also, it is possible to separate the damaged picture into multiple layers. Artists draw pictures in multiple layers. Usually, the background in a Chinese painting is in a light color. The foreground is painted in a darker color. Separation of layers can be done by a classification mechanism based on color histogram. Different inpainting mechanism can be used in different layer. And, the results can be combined. We believe that, the contribution of inpainting methods have both academic and commercial values.

References

[1] M. Bertalmio, G. Sapiro, V. Caselles and C. Ballester, "Image Inpainting," in Proceedings of the ACM SIGGRAPH Conference on Computer Graphics, 2000, SIGGRAPH 2000, 2000, pp.417-424.

[2] M. Bertalmio, A. Bertozzi, G. Sapiro, "Navier-Stokes, Fluid-Dynamics and Image and Video Inpainting," in Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2001, pp. I355-I362.

[3] T. F. Chan and J. Shen, "Nontexture Inpainting by Curvature-Driven Diffusions," Journal of Visual Communication and Image Representation, V. 12, N. 4, 2001, pp. 436-449.

[4] Manuel M. Oliveira, Brian Bowen, Richard McKenna, Yu-Sung Chang, "Fast Digital Image Inpainting," in Proceedings of the International Conference on Visualization, Imaging and Image Processing (VIIP 2001), 2001, pp. 261-266.

[5] C.-H. Huang; J.-L. Wu, "Inpainting attacks against visible watermarking schemes," in Proceedings of Spie, the International Society for Optical Engineering, 2001, no. 4314, pp. 376-384.

Table 2: Test Results of 2 sets of parameters using 1000 pictures

	Average PSNR values (dB)				Area percentages with PSNR>30 (%)			
Level	Damaged	Single	Multiple		Damaged	Single	Multiple	
			α:80,β_1:80	α:80, β_1:90			α:80,β_1:80	α:80, β_1:90
1	21.0908	26.8535	27.9091	27.9103	0	7	17	17
2	21.9556	27.3046	28.4243	28.4252	3	18	30.25	30.25
3	26.4713	32.2096	33.6134	33.6144	18.82	39.6	46.47	46.47
4	42.5164	47.7414	49.2164	49.2175	37.62	53.99	59.23	59.23
5	60.5305	64.8264	65.9877	65.9892	56.32	66.98	70.64	70.64
6	75.702	78.8266	79.6501	79.6509	72.35	79.09	81.03	81.03
7	85.2987	87.5305	88.1085	88.1092	82.98	88.62	89.65	89.66
8	90.79	92.403	92.8222	92.823	89.2	93.5	94.18	94.18