# A Non-hierarchical Procedure for Re-synthesis of Complex Textures

Paul Harrison

19 October 2000

**Abstract**

A procedure is described for synthesizing an image with the same texture as a given input image. To achieve this, the output image is built up by successively adding pixels selected from the input image. Pixels are chosen by searching the input image for patches that closely match pixels already present in the output image. It is shown that the accurate reproduction of features in the input texture depends on the order in which pixels are added to the output image. A procedure for selecting an ordering which transfers large complex features of the input to the output image is described. This procedure is capable of reproducing large features even if only the interactions of nearby pixels are considered. The procedure can be altered to allow specification of the placement of particular features in the output texture. Several applications of this are described.

## 1 Introduction

The texture of an image might be defined broadly as the interrelationships between pixels in that image. The ability to analyse and manipulate image texture has a number of interesting applications. The simplest application is to create a new image with the same texture but of different size and shape to a sample image. Seamless editing of images is also a possibility. For example an object could be removed from an image by synthesizing a new section of the background texture over the top of it. The texture of one image might even be applied to another image, for example to make a photograph look as if it had been painted. These applications all rely on the ability to re-synthesize a sample texture to fit a variety of constraints.

Because texture re-synthesis has a wide range of potential applications, many texture re-synthesis methods have been investigated. One approach taken in the literature is based on searching for specific features in textures. In these methods, the input image is decomposed into a set of features. Statistics about these features are collected, and used to synthesize a new image.

An example of this approach is described by Heeger and Bergen [11]. In this, a set of features known as a *steerable pyramid* is used to decompose an input image. A new image can then be synthesized by matching the histograms of these features. De Bonet [2] extended the use of steerable pyramids to consider interactions between different levels in the pyramid, yielding higher quality results.

Igehy and Pereira [12] have adapted De Bonet's technique to the removal of unwanted objects from an image. A portion of the image is re-synthesized, taking care to match the edges of this portion to the rest of the image. In this manner, objects can be convincingly removed from photographs.

A more advanced re-synthesis procedure has been described by Portilla and Simoncelli [14]: a textural image is decomposed into complex wavelets and a new image synthesized by matching a variety of statistics about these wavelets. These include the distribution of, and correlations between, wavelet parameters.

Portilla and Simoncelli apply this technique in a number of novel ways, including interpolating between two textures to produce a mixture of both, and applying the technique to resolution enhancement.

A problem with feature-based methods is that they can only recognize a set of features which have been specified in advance. While the results can be impressive, it is difficult to devise a generic feature set that can be used to describe all textures.

Another approach to the texture re-synthesis problem is to analyse and reproduce interactions between individual pixels. Methods based on this approach define a function describing a pixel in terms of its neighbours and a random element. This function is used to generate a new image pixel by pixel. A number of these pixel-based techniques of varying complexity have been developed.

An early example is Chellappa and Kashyap's autoregressive model [4], which expresses each pixel in terms of a random component plus a weighted sum of the pixel's neighbours. The model represents the texture with a very small number of parameters, and so can only synthesize a very limited range of textures.

Another pixel-based method is Monne and Massaloux's Markov chain model [13]. In this method, probability distributions of a pixel based on a neighbourhood of surrounding pixels are used to create a new random texture. The number of probability estimates needed in the probability distributions grows exponentially with the size of neighbourhood being considered. While this method is capable of producing complex textures, the number of probability estimates required to do so is large. For accurate estimates, a very large input texture is required.

Gagalowicz and De Ma [9] describe another variation based on interactions between pairs of nearby pixels. This method examines the histograms of pairs of pixels separated by a particular distance, in a particular direction. It tries to match the histograms of these pairs in the output image to those found in the input image. This has been applied to creating texture maps for three-dimensional objects [8].

Recently, a number of pixel-based techniques have been developed based on best-fit searching. These effectively piece together the output image from small parts of the input image. Each pixel is selected by searching the input image for a patch of pixels closely matching nearby pixels already present in the output image. Using this technique, textures can be synthesized in which the relationships between neighbouring pixels are extremely complex.

The earliest description of this technique is by Garber [10]. At the time it was too computationally expensive for practical use, involving as it does the searching of the entire input image to generate each pixel of the output. Because of improvements in computing hardware and the development of techniques for efficient searching, the method is now becoming more practical.

Efros and Leung [6] independently developed the best-fit technique, and have applied it to image extrapolation (generating plausible new details outside the border of the original image), and to reconstructing missing or damaged areas in an image. They report that for good results the size of the patch searched for in the input image should correspond to the size of the largest feature in the texture.

Wei and Levoy [15] present a refinement of the best-fit method capable of reproducing certain larger features. Their technique is a combination of feature-based and pixel-based methods. It is based on a pyramid of images of successively larger scales. To generate each pixel in the output pyramid, a region in the input pyramid similar to surrounding pixels in the current layer and the layers above is searched for. Wei and Levoy apply this to both two dimensional images and three dimensional textural animations with impressive results.

This paper presents a refinement of the best-fit method, based solely on interactions of closely neighbouring pixels. The major difficulty with this method lies in replicating large features in the texture, as no explicit model of these is ever created. The best-fit method is shown to be capable of reproducing features larger than the neighbourhood used for searching the input image, but only if care is taken in selecting the order in which pixels are added to the output image. Because it is not feature-based, the procedure described can replicate a great variety of large structures.

## 2 Re-synthesis procedure

The procedure described in this paper takes as input an image containing a sample of a texture and produces another image with the same texture. The input image is assumed to be a homogeneous texture, in the sense that statistics collected from one area of the input image will not differ significantly from those collected from another area (for sufficiently large areas).

The fundamental approach taken is to add pixels to the output image one at a time, starting with an empty image. To add a pixel to the output image at a particular location, the surrounding pixels which have already been added to the output image are examined. The closest match to these pixels is located in the input image, and the corresponding pixel from the input image is inserted in the output.

At this stage it is convenient to introduce some notation:

| | |
|---|---|
| $\Omega_I$ : | The set of pixel locations in the input image. |
| $I(s)$, $s \in \Omega_I$ : | The pixel in the input image at location $s$. |
| $\Omega_O$ : | The set of locations containing pixels in the output image. |
| $O(s)$, $s \in \Omega_O$ : | The pixel in the output image at location $s$. |
| $L(s)$, $s \in \Omega_O$ : | The location of the pixel in the input image that is in the output image at location $s$. |
| $\Omega_K$ : | The set of offsets considered when calculating the similarity of two texture patches. |
| $K(u)$, $u \in \Omega_K$ : | Weighting given to a particular offset $u$. |

Table 1: Table of symbols.

Colours are stored as RGB values. To compare individual pixels, the sum of the absolute values of the differences in each colour component was used:

$$d((r_1, g_1, b_1), (r_2, g_2, b_2)) = |r_2 - r_1| + |g_2 - g_1| + |b_2 - b_1| \tag{1}$$

To measure how closely patches from the input image match a patch in the output image, a distance function can be used. The distance function used in this paper is a weighted Manhattan (city block) distance function:

$$D(s, t) = \sum_{u \in \Omega_K, t+u \in \Omega_O} K(u) d(I(s+u), O(t+u)) \tag{2}$$

The Manhattan distance was chosen over the more commonly used Euclidean distance as it is more forgiving of outliers. I.e. a good match in most pixels in the patch will not be negated by a poor match in one or two pixels. The method used for selecting the weights $K(u)$ is described in section 2.2. Refinements to the distance function are discussed in sections 2.5 and 3.

For each location $t$ in the output image, a pixel $I(s)$ from the input image is chosen. The location $s$ in the input image is selected to minimize the distance $D(s, t)$ as given in equation 2.

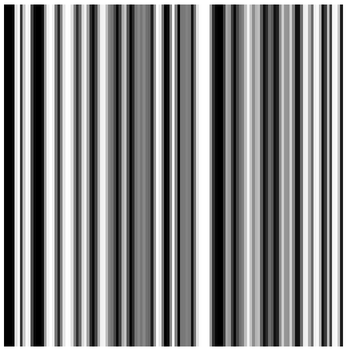


Figure 1: Barcode texture.

## 2.1 Ordering of pixel insertion

This section considers the order in which pixels are added to the output image. As will be demonstrated, certain features such as branching patterns are only reproduced in the output image if the placement of pixels proceeds in a particular order.

Consider the values a particular pixel in a textural image could take. The pixel value must be consistent with the values of all other pixels in the image. These other pixels may be said to constrain the values the selected pixel can take. Some pixels will provide more constraint than others. In general, close neighbours of a pixel will provide a greater constraint than pixels further away.

These constraints can be thought of in terms of information theory [1]. Without any prior knowledge, a pixel is equally likely to take any value, and this requires a certain number of bits to encode. If its neighbours impose constraints on the values it can take, it will require (on average) less bits to encode. The average number of bits required to encode some information is referred to as its *entropy* [1, pp. 55]. Entropy can therefore be used to measure the constraints imposed on a pixel.

A pixel in an image may be constrained more tightly by some neighbours than by others. Consider the barcode-like texture shown in figure 1. A pixel in this texture will be highly constrained by the pixels immediately above and below it, but very little by neighbours to its left and right.

A final point to note is that the constraint on a pixel provided by a distant pixel in some direction may also be provided by a closer pixel in the same direction. For example, in figure 1 the constraint on a pixel provided by another pixel some distance above it will also be provided by a pixel a small distance above it. However, for some textures, this may only apply in certain directions. In the barcode texture, this applies vertically but not horizontally.

The pixel selection procedure is based on a search for patches of pixels in the input image that match those already placed in the output. These patches must cover enough pixels to capture all the constraints on each location, or the texture will not be reproduced correctly. However, if the patch size is too large, there will be less chance of finding good matches in the input image. It is therefore desirable to choose an order to add pixels to the output image in which the constraints imposed on each location are captured by a small neighbourhood of pixels.

One order that can be used to add pixels is a raster scan. This is adequate for textures in which the constraint on pixels is the same from all directions. For some other textures, this works only if the raster scan proceeds in a particular direction. Figure 2 shows the result of applying raster scans that proceed in different directions to a picture of lichen with a complex branching pattern. As can be seen, the scan that proceeds from bottom to top (figure 2c) gives the best results.

The other raster scans in figure 2 have various problems. The top to bottom scan (figure 2b)

(a) 

(b)  (c) 

(d)  (e) 

$$
\text{(f)}\quad
\begin{array}{ccccc}
 & 1 & 1 & 1 & \\
1 & 1 & 1 & 1 & 1 \\
1 & 1 & \mathrm{P} & 1 & 1 \\
1 & 1 & 1 & 1 & 1 \\
 & 1 & 1 & 1 &
\end{array}
$$

Figure 2: Textures generated using different orders of pixel insertion.
(a) input image, (b) output from a raster scan from top-to-bottom and on each row, left-to-right, (c) output from a bottom-to-top left-to-right raster scan, (d) output from a left-to-right top-to-bottom raster scan, (e) output from a right-to-left top-to-bottom raster scan. (f) Weighting of the distance function $K(u)$ used about the center pixel (P).

produces anomalous dark blotches, and the lichen in the sideways scans (figures 2d and 2e) have branches on one side only.

These artifacts can be explained by considering how the pixels constrain each other. The constraint imposed on a given pixel by the pixels immediately below it is almost the same as the constraint imposed by all pixels below it. This is not true of pixels above or to either side of that pixel. The bottom-to-top raster scan therefore produces a better result than the others tested.

To utilize these types of constraint, the re-synthesis procedure assigns a priority to each location in the output image. Highest priority is given to locations which are highly constrained by neighbouring pixels that have already been added. Then the following procedure is followed:

```
While there are still empty locations in the output image:
    Find the empty location with the highest priority
    Choose a pixel from the input image to place in that location
    Update priorities of empty neighbouring locations based on the
      new pixel value
```

A system for assigning priorities needs to be chosen. A simple prioritization scheme estimates the entropy of each unfilled location given its nearby neighbours, and gives higher priority to locations with lower entropy. The problem with this scheme is that there may be some areas of a texture in which nearby pixels tightly constrain each other, and others where there is less constraint and pixel values are more random. The areas with tight constraints will always be given higher priorities and therefore be disproportionately represented in the output image.

The approach taken in this paper is to define a normalized weighting $W(s,u)$. This weighting indicates the relative amount of information a pixel $I(s)$ gives about each of its neighbours $I(s+u)$. To ensure no part of the image is given an advantage over another part, this weighting is defined so that it sums to one for each pixel $I(s)$. The priority of each empty location in the output can then be defined as a sum of the weightings from neighbouring pixels.

## 2.2 Analysing interactions between neighbouring pixels

Weightings are needed to prioritize the order in which pixels are inserted. They are also needed in the distance function used to select these pixels. To create these weightings, interactions between neighbouring pixels in the input image are analysed. For purposes of computational simplicity, the effect of each neighbour on a location is assumed to be independent of the effects of other neighbours. The interactions between constraints of different neighbours on a location are neglected.

To compute the weightings, the amount of information every pixel provides about each of its near neighbours first needs to be estimated. The neighbours being considered are those whose offset from the central pixel is a member of the set of offsets $\Omega_K$ used to calculate the distance function $D(s,t)$ (equation 2).

This can be done by considering each offset $u \in \Omega_K$, in turn, for all pixels, rather than by considering each pixel in turn. For each offset, the number of bits of information $G(s,u)$ provided by each pixel in the input image $I(s)$ about the pixel offset from it $I(s+u)$ is estimated, as explained below.

Offset pixels $I(s+u)$ are grouped into sets based on the value of the corresponding pixel $I(s)$. Within each set, the pixels will have a distribution of values. These values are modelled as being normally distributed in each colour component (red, green and blue), and the parameters of these distributions are estimated. This is used to calculate the number of bits required to encode pixels of each set.

To distribute the pixels $I(s+u)$ into sets, they are classified by the most significant $m$ bits of the red, green and blue components of $I(s)$. The value of $m$ needs to be chosen carefully. If $m$ is too large, the pixels will be split into many small sets and it will not be possible to model the distribution of the pixels in these accurately. If $m$ is too small, the pixels will be split into a few large sets and most of the information from the pixels $I(s)$ will be discarded.

To balance these factors, the value of $m$ giving the best overall compression is chosen. This corresponds to the choice of $m$ which uses the most information from $I(s)$ to predict $I(s+u)$. For each possible $m$ the total number of bits to encode the entire input image (the sum of the number of bits to encode each set) can be computed and the $m$ giving the smallest size chosen. This avoids the need to specify $m$ as a parameter.

The Minimum Message Length (MML) technique [5] can be used to calculate the minimum number of bits required to encode a normal distribution. In MML, hypothetical messages conveying the data in a data set are considered. These messages consist of two parts: a hypothesis as to the nature of the data; and the data itself, concisely encoded with the aid of the hypothesis. The hypothesis producing the shortest such message is taken as the best hypothesis as to the nature of the data set. The length of this message is the minimum number of bits required to encode the data. In the case of normally distributed data, the message will consist of a hypothesis in the form of a mean and standard deviation, followed by the data encoded using this hypothesis.

The number of bits required to encode a message containing normally distributed data, to an accuracy of $\pm\frac{1}{2}$, is (from [5]):

$$Message\ Length = -\log_2 h(\mu,\sigma) + L + \frac{1}{2}\log_2 F(\mu,\sigma) - \frac{1}{2}\log_2 12 + \frac{1}{2\ln 2} \tag{3}$$

Here $h(\mu,\sigma)$ is the *prior*, the probability distribution representing our prior beliefs about the mean $\mu$ and standard deviation $\sigma$. $L$ is the number of bits required to encode the data given $\mu$ and $\sigma$. The quantity $F(\mu,\sigma)$, known as the *Fisher information*, describes how small changes to $\mu$ and $\sigma$ affect the message length.

The number of bits required to encode $N$ normally distributed values $x_i$ to an accuracy of $\pm\frac{1}{2}$ is:

$$L = \frac{N}{2}\log_2(2\pi) + \frac{N}{2}\log_2(\sigma^2) + \frac{1}{2\ln(2)\sigma^2}\sum_{i=1}^{N}(x_i-\mu)^2 \tag{4}$$

The Fisher information for a normal distribution is:

$$F(\mu,\sigma) = \frac{N^2}{2\sigma^6} \tag{5}$$

Finally, reasonable prior beliefs about the mean and standard deviation need to be described. A commonly used prior for the normal distribution is $h(\mu,\sigma)\propto \frac{1}{\sigma^2}$ because this is independent of scale. If we assume the mean to be between zero and a maximum value $n$, and the standard deviation to be greater than or equal to one, this prior is

$$h(\mu,\sigma) = \frac{1}{n\sigma^2} \tag{6}$$

This gives the number of bits to encode each colour component in each set. These can be totalled, giving the number of bits to encode each set. The summation of the number of bits to encode each set gives the total number of bits to encode the image.

The number of bits from $I(s)$ used to classify $I(s+u)$, $m$, can then be chosen to minimize the total number of bits required to encode the image.

Once $m$ has been chosen and the pixels $I(s+u)$ placed in sets, the standard deviation of each colour component in each set can be used to estimate the entropy of pixels in that set. All the pixels in a set are assumed to come from the same distribution. Therefore, the standard deviations calculated from the sets can be used to describe individual pixels in those sets. This gives standard deviations for each pixel at each offset for each colour component: $\sigma_r(s,u)$, $\sigma_g(s,u)$, $\sigma_b(s,u)$.

These standard deviations can be used to calculate the entropy of each pixel $I(s+u)$ given $I(s)$. The average number of bits required to store a normally distributed variable with standard deviation $\sigma$, to an accuracy of $\pm\frac{1}{2}$, is:

$$\frac{1}{2}\log_2(2\pi) + \frac{1}{2\ln 2} + \log_2 \sigma \tag{7}$$

Summing this for each colour component gives the entropy of the pixel $I(s+u)$ given $I(s)$:

$$H(s,u) = \frac{3}{2}\log_2(2\pi) + \frac{3}{2\ln 2} + \log_2 \sigma_r(s,u)\sigma_g(s,u)\sigma_b(s,u) \tag{8}$$

Performing the same calculation without splitting the image into sets gives the entropy of pixels in the image independent of any of their neighbours, $H_{image}$.

The number of bits given by a pixel $I(s)$ about another pixel offset from it $I(s+u)$ can then be found by subtracting the entropy of $I(s+u)$ given $I(s)$ from the entropy of pixels in the image if nothing is known about their neighbours. Call this value $G(s,u)$, where:

$$G(s,u) = H_{image} - H(s,u) \tag{9}$$

### 2.2.1 Prioritization weighting

A normalized weighting $W(s,u)$ suitable for prioritizing the order in which pixels are added to the output image can be defined from $G(s,u)$ :

$$W(s,u) = \frac{G(s,u)}{\sum_{v \in \Omega_K} G(s,v)} \tag{10}$$

### 2.2.2 Distance function weighting

The distance function $D(s,t)$ (equation 2) used to select pixels also requires a set of weightings. These weightings specify the relative importance of each neighbour to a pixel being considered for selection. To make best use of the input image, these weightings should reflect the degree to which each neighbour constrains the value of the pixel.

The entropy of a normally distributed random variable, such as one of the color components of a pixel, is the logarithm to base two of its standard deviation, to within a constant (see equation 7). This means that for each time a constraint on a particular location halves the standard deviation of possible values of one color component of that location, one more bit of information about the location becomes known. In light of this relationship, the weighting given to a neighbour might be doubled for every bit it gives about a location.

The weighting system used in the distance function was chosen to satisfy this constraint:

$$K(u) = 2^{G(-u)} \tag{11}$$

where $G(u)$ is the average information given by a pixel about a particular neighbour:

$$G(u) = \frac{\sum_{s \in \Omega_I(u)} G(s,u)}{|\Omega_I(u)|} \tag{12}$$

where $\Omega_I(u)$ is the set of locations for which $G(s,u)$ is defined:

$$\Omega_I(u) = \{s : \ s \in \Omega_I \ and \ s + u \in \Omega_I\} \tag{13}$$

## 2.3 Efficient searching

Finding the closest matching patch in the input image is time consuming. It is an instance of the nearest-neighbour search problem, which occurs in a wide range of fields and has been well studied [7][16][3].

Two factors make the problem difficult in this instance. Firstly, the search space has a high number of dimensions (one for each color component of each pixel in the patch). Many nearest neighbour search algorithms perform poorly in high dimensions. Secondly, the shape of the patch being searched for varies between searches, as the patch only includes pixels that have already been added to the output image. This changes the nature of the distance function between searches.

A *kd-tree* [7] was found to be an effective way of decreasing the search time of the procedure.

## 2.4 Production of the output image

Each unfilled location in the output image is given a priority:

$$P(s) = \sum_{u \in \Omega_K, s-u \in \Omega_O} W(L(s-u), u) \tag{14}$$

Additionally, locations near the edge of the image have their priorities adjusted as if the positions beyond the edge were already filled. These locations act as starting positions for filling the output image.

Pixels are added to the output image until it is full. When selecting the location at which to add a pixel, the place with the highest priority is chosen. After choosing the pixel, the priorities of surrounding unfilled locations are updated appropriately.

## 2.5 An extension to remove unwanted repeating patterns

The distance function described in equation 2 is entirely deterministic. Performing a search for the closest match to a particular patch of pixels will always yield the same result. This has the unfortunate effect that unwanted repeating patterns are sometimes produced in the output image. The problem can be avoided by introducing a small amount of randomness into the distance function. To do this, two uniformly random functions are defined. These are $A(s)$ for all locations $s$ in the input image, and $B(t)$ for all locations $t$ in the output image. Then, for some small $\varepsilon$, the distance function can be redefined as

$$D(s,t) = \varepsilon \, |A(s) - B(t)| + \sum_{u \in \Omega_K, t+u \in \Omega_O} K(u) d(I(s+u), O(t+u)) \tag{15}$$

The parameter $\varepsilon$ may be adjusted to vary the randomness of the output texture.

## 2.6 Results of the synthesis procedure

Example results from the procedure are shown in figure 3. A seven by seven neighbourhood of pixels ($\Omega_K = [-3,3] \times [-3,3]$) was used to generate these images.

An implementation of the procedure can be obtained from `http://www.csse.monash.edu.au/~pfh/resynthesizer/`

One interesting feature of the procedure is that parts of the input image are occasionally replicated in the output. This can be seen as advantageous, as some of the more complex features not entirely described by interactions of near neighbours are replicated in the output. If this is undesirable it can be avoided by increasing the random component of the distance function (equation 15).

Less successful results are shown in figure 4. In particular the procedure is very sensitive to features that only appear at the edge of the input image. In this case the output tends to contain artifacts such as repeating patterns.

This can sometimes be avoided by altering the input slightly. For example, suppose the texture was a photograph taken under non-homogeneous lighting, but was otherwise homogeneous. Removing the lowest frequencies of the photograph's Fourier transform would remove the differences in lighting, making the texture homogeneous. The re-synthesis procedure would then produce better results.

Also, as can be seen in figure 4c, the procedure does not recognize regularly spaced components of a texture such as tiles.

# 3 Specification of the placement of texture regions

Some textures contain regions with properties that differ from one to another. For example, a sky texture might have regions of cloud and regions of blue sky, or a painting might have light and dark areas. If the synthesis method described above is used, these regions will be placed randomly in the output image.

A simple extension to the distance function used to select pixels gives a means of specifying the layout of different regions in the output image. This extension requires the definition of two new images, one which maps regions in the input, $J(s)$, and one which specifies corresponding regions in the output, $P(t)$. It also requires weightings, $L(u)$, which represent the relative importance of pixels from the input map in determining the value of their surrounding pixels in the input image. These are chosen using the same method as that used to choose $K(u)$. Then a new distance function is defined:

$$\begin{aligned} D(s,t) = \quad & \varepsilon\,|A(s) - B(t)| + \\ & \textstyle\sum_{u \in \Omega_K, t+u \in \Omega_O} \quad \{K(u) d(I(s+u), O(t+u)) + \\ & \qquad\qquad L(u) d(J(s+u), P(t+u))\} \end{aligned} \tag{16}$$

## 3.1 Results of the extended synthesis procedure

An example of the use of the extended synthesis procedure is shown in figure 5. The texture used was an aerial photograph (figure 5b). A map of this texture was created showing regions of cloud and land (figure 5a). Another map, of a checkerboard pattern, was used as the output map (figure 5c). The result of applying the procedure to these three images is shown in figure 5d.

Another example is shown in figure 6. Here, examples of crosshatching were used as the texture (figure 6a). The texture image was used as its own map. For the output map, a photograph of a sunset was used (figure 6b). The result (figure 6c) is a sunset rendered in a crosshatched style.

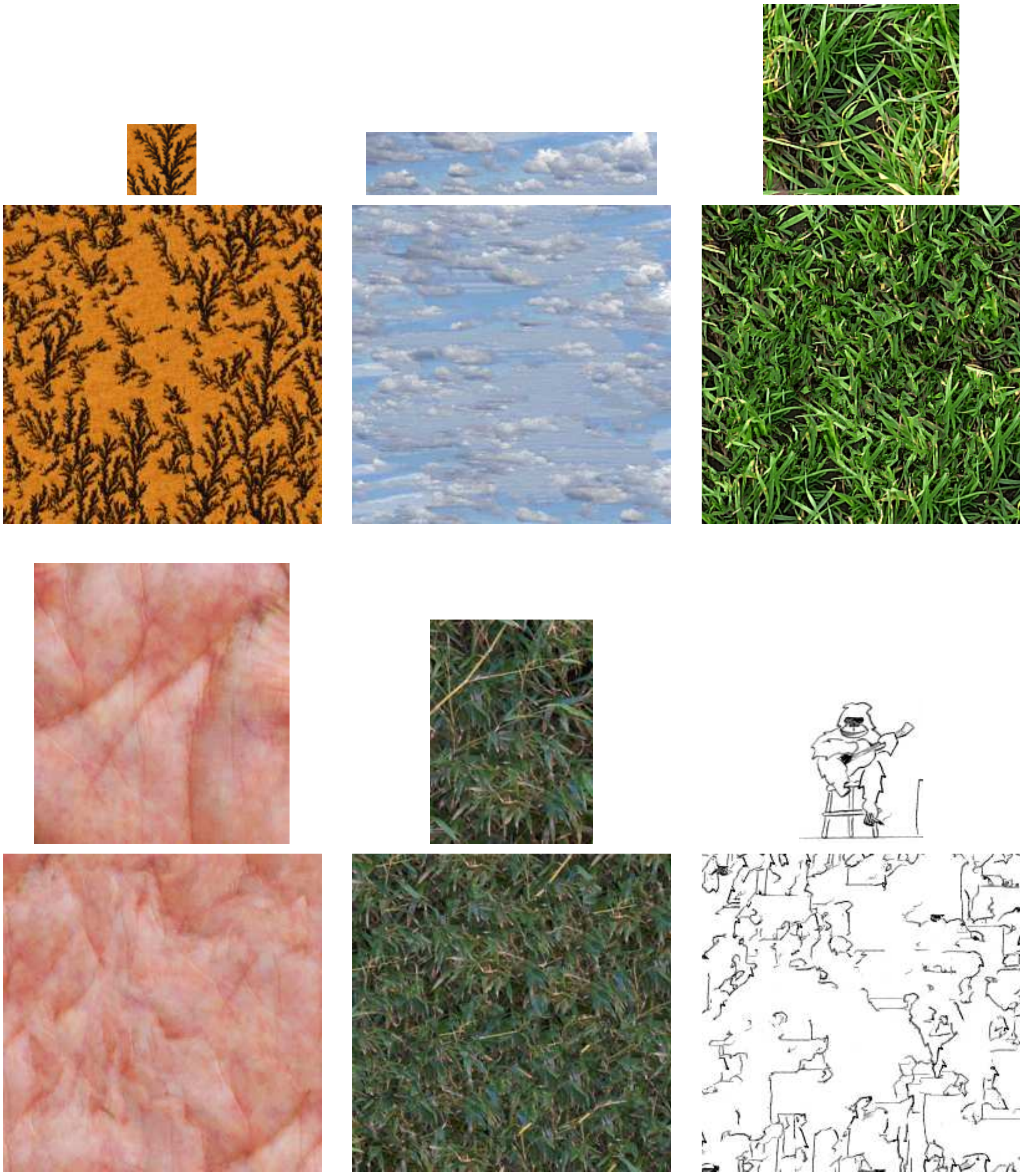


Figure 3: Sample results.
The output of the procedure is shown below each input image.

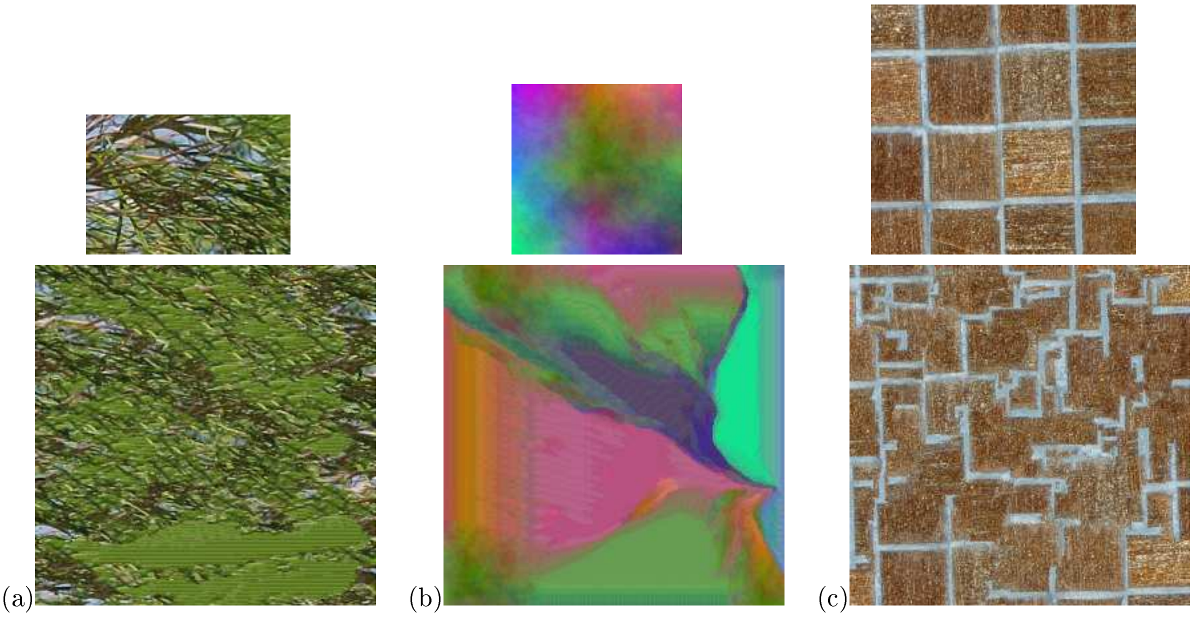



Figure 4: Some failures.

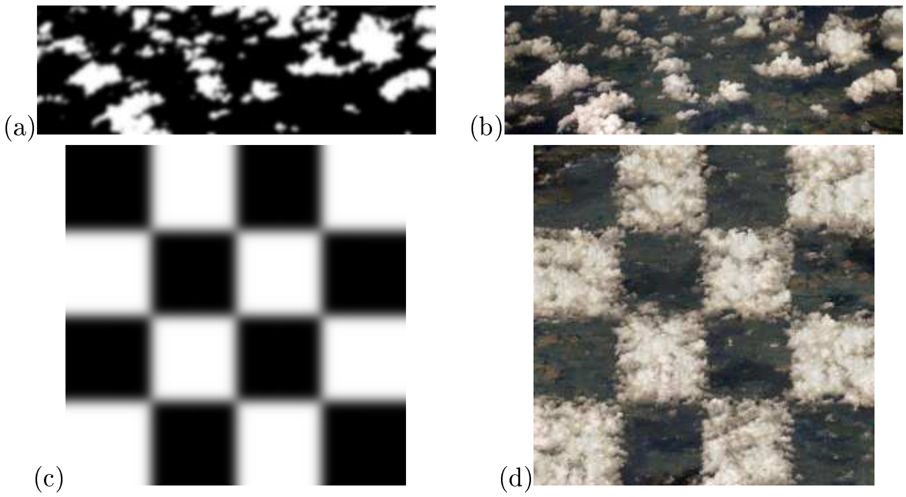
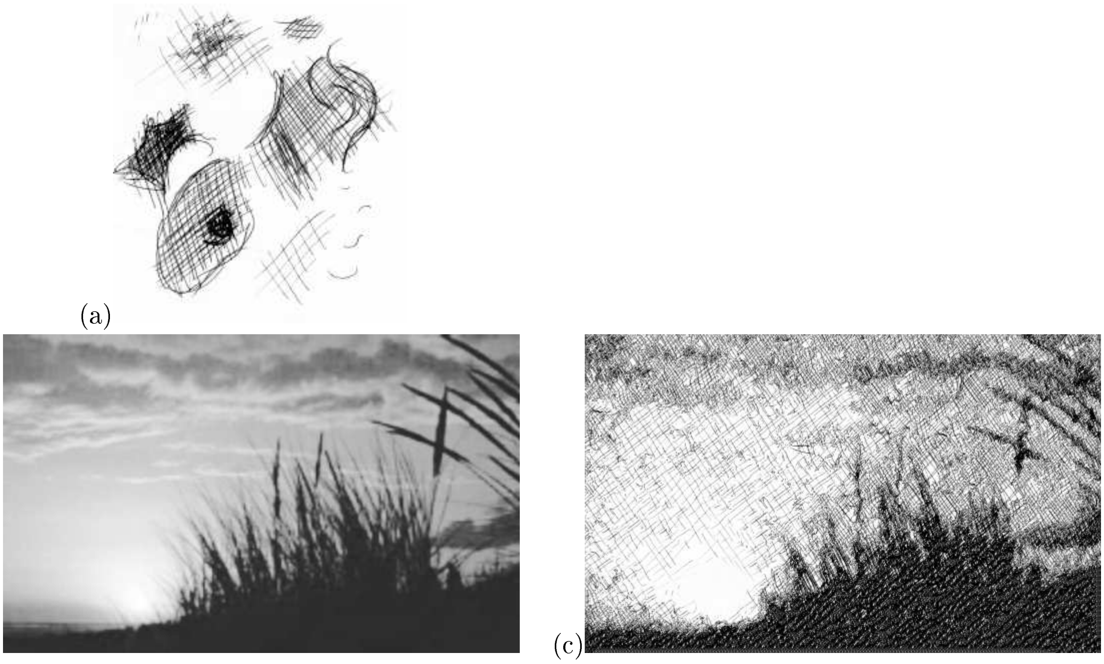



Figure 5: Constraining a cloud texture to a checkerboard pattern.
(a) input map, (b) input texture, (c) output map, (d) output of the extended synthesis procedure.

(a)



(b) (c)

Figure 6: Applying a crosshatched drawing style to an image.
(a) image for both input map and texture, (b) output map (photo courtesy of Lynne Shandley), (c) output of the extended synthesis procedure.

Texture re-synthesis has been previously applied to the removal of objects on a homogeneous background by synthesizing a new section of that background [12]. The ability to constrain placement of texture regions allows synthesis of a new background to replace an object, even if the background is non-homogeneous. An example of this is shown in figure 7. Figure 7a shows a picture of a donkey standing in a field. In this picture, the field is not homogeneous because of perspective expansion. A feature map of the image (figure 7b) was constructed with parts of the image equidistant from the camera having the same value. A new section of the background was then synthesized to replace the donkey, using the rest of the background as input texture. The new section was constrained to join with the existing background and to follow the feature map. The result is shown in figure 7c.

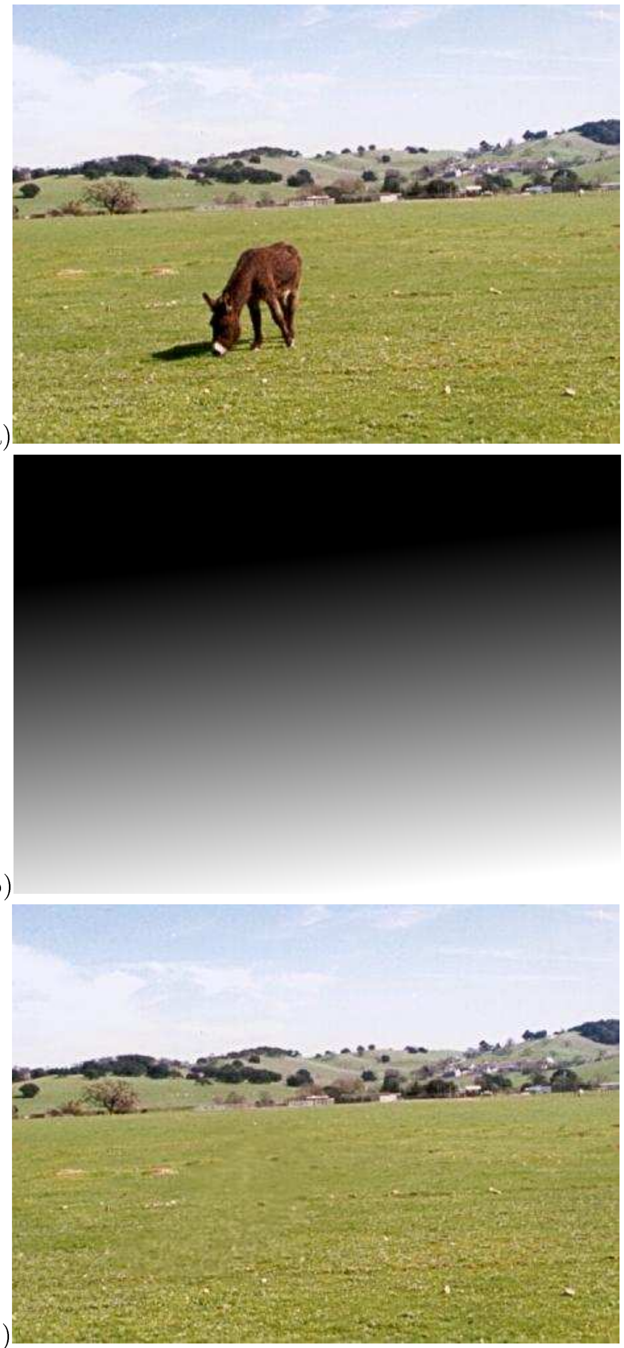
# 4 Conclusion and discussion

A procedure has been devised for synthesizing an image of arbitrary size with the same texture as that of a sample input image. The procedure is capable of reproducing large features from this input image, even though it only examines interactions between pixels that are close neighbours. Significantly, the procedure does not have any preconceived notion of what features to expect in the input texture.

A simple extension to this procedure allows feature placement in the new image to be constrained. It is effectively a generic filter, in that when given an example of a change to one image it can reproduce the same change in another image.

Texture re-synthesis techniques could conceivably be used to create more intelligent versions of many tools found in modern image manipulation programs. For example:

- An eraser tool that extends the surrounding texture to cover the item being erased.



(a) (b) (c)

Figure 7: Example of object removal.
(a) original image (from [12]), (b) map, (c) image with object removed.

- A paintbrush tool that can simulate any type of painting technique, given sample brush-strokes.
- An image enlargement tool that extrapolates an image's existing texture to a finer scale.

Future work on the procedure described here could refine the distance function to consider matches that have similar structure even if they are slightly lighter or darker or have a different hue. It could also be extended to work on non-flat geometries, to allow synthesis of textures covering three-dimensional objects. It might also be applied to three dimensions to manipulate animations or solid textures, or to one dimension as a sound synthesis technique.

## 5 Acknowledgements

I thank my P.h.D. supervisor, Dr. Alan Dorin, for his patience and for many helpful suggestions. Without his input, this paper would have been incomprehensible. I also thank Homan Igehy, Lucas Pereira and Lynne Shandley for the use of several images in this paper.

## References

[1] R. E. Blahut. *Principles and Practice of Information Theory.* Addison-Wesley, 1987.

[2] J. S. D. Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *Proceedings of SIGGRAPH 1997*, pages 361–368, 1997.

[3] S. Brin. Near neighbour search in large metric spaces. In *Proceedings of the 21st International Conference on Very Large Data Bases*, pages 574–584, 1995.

[4] R. Chellappa. Texture synthesis using 2-d noncausal autoregressive models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 33(1):194–203, Feb. 1985.

[5] D. L. Dowe and G. E. Farr. An introduction to MML inference. Unpublished, contact {dld, gfarr}@cs.monash.edu.au, 1997.

[6] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the 7th IEEE International Conference on Computer Vision*, pages 1033–1038, 1999.

[7] J. H. Friedman. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, Sept. 1977.

[8] A. Gagalowicz. Texture modelling applications. *The Visual Computer*, 3:186–200, 1987.

[9] A. Gagalowicz and S. D. Ma. Sequential synthesis of natural textures. In *Selected Papers on Digital Image Processing*, pages 184–210. SPIE Optical Engineering Press, 1990.

[10] D. Garber. *Computational Models for Texture Analysis and Texture Synthesis.* PhD thesis, University of Southern California, 1981.

[11] D. J. Heeger and J. R. Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of SIGGRAPH 1995*, pages 229–238, 1995.

[12] H. Igehy and L. Pereira. Image replacement through texture synthesis. In *Proceedings of the 1997 IEEE International Conference on Image Processing*, 1997. Available: `http://graphics.stanford.edu/papers/texture_replace/` (Accessed: 2000, September 7).

[13] J. Monne, F. Schmitt, and D. Massaloux. Bidimensional texture synthesis by Markov chains. *Computer Graphics and Image Processing*, 17:1–23, 1981.

[14] J. Portilla and E. P. Simoncelli. Texture modeling and synthesis using joing statistics of complex wavelet coefficients. In *Proceedings of the IEEE Workshop on Statistical and Computational Theories of Vision*, 1999. Available: `http://www.cis.ohio-state.edu/~szhu/SCTV99.html` (Accessed: 2000, September 7).

[15] L. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of SIGGRAPH 2000*, 2000. Available: `http://graphics.stanford.edu/projects/texture/` (Accessed: 2000, September 7).

[16] P. N. Yianilos. Data structures and algorithms for nearest neighbour search in general metric spaces. In *Proceedings of the 4th Annual ACM Symposium on Discrete Algorithms*, pages 311–321, 1993.