

A Fast Texture Feature Extraction Method for Region-based Image Segmentation

Hui Zhang*, Jason E. Fritts, Sally A. Goldman
Dept. of Computer Science and Engineering, Washington University,
One Brookings Drive. St. Louis, MO, USA, 63130

ABSTRACT

Region-based image segmentation is one popular approach to image segmentation of generic images. In these methods, an image is partitioned into connected regions by grouping neighboring pixels of similar features, and adjacent regions are then merged with respect to the similarities between the features in these regions. To achieve fine-grain segmentation at the pixel level, we must be able to define features on a per-pixel basis. This is straightforward for color information, but not for texture. Typically texture feature extraction is very computationally intensive for individual pixels. In this paper, we propose a novel fast texture feature extraction method which takes advantage of the similarities between the neighboring pixels. The experiments demonstrate that our method can greatly increase the extraction speed while keeping the distortion within a reasonable range.

Keyword: image segmentation, region-based segmentation, texture, feature extraction, wavelet transform

1. INTRODUCTION

Image segmentation is a fundamental process in many image, video and computer vision applications. It decomposes an image into several constituent components, which ideally correspond to real-world objects. Region-based image segmentation is a popular approach to image segmentation, in which an image is partitioned into connected regions by grouping neighboring pixels of similar features, and adjacent regions are then merged under some criterion such as the homogeneity of features in neighboring regions. Features of interest often include color, texture, shape, etc.

To achieve fine-grain segmentation at the pixel level, we must be able to define features such as color and texture on a per-pixel basis. The method for exacting color information is straightforward, but texture feature extraction is very computationally intensive for individual pixels. In this paper, we propose a novel fast texture feature extraction method which takes advantage of the similarities between the neighboring pixels to estimate texture values. The experiments show that our method can greatly increase the extraction speed while keeping the distortion due to estimation error within a reasonable range.

The remainder of the paper is organized as follows: In Section 2, we define the texture features for individual pixels and describe the traditional extraction method. In Section 3, we describe our fast texture feature extraction method. Then in Section 4, we examine the speedup of our extraction method. The effectiveness of our texture feature extraction method is examined in Section 5, and Section 6 concludes the report and discusses future work.

2. PIXEL-LEVEL FEATURE EXTRACTION

Texture is one common feature used in image segmentation. It is often used in conjunction with color information to achieve better segmentation results than possible with just color alone. To enable fine-grained segmentation, we must be able to segment down to the level of individual pixels, so feature values must be extracted on a per-pixel basis. For color information, this is straightforward since the three color components defining the pixel's color intensity are usually used as its color features. For groups of pixels, the color features are represented by the average and standard deviation of the components' color values.

* huizhang@wustl.edu; phone 1 314 935-8561; fax 1 314 935-7302

For extracting texture feature information, there are two primary methodologies. The first class of methods applies a linear transform, filter, or filter bank globally to the image. The local energy of the filter responses represents the local texture feature values¹. Generally these methods have high computational complexity. The second class of methods divides the whole image into many small non-overlapping pixel blocks, and then applies some transform, such as a wavelet transform, to each block to get the local information². These methods extract texture features for a block of pixels. Both methodologies have the problem of generating texture information for each individual pixel.

To extract texture features for each pixel, we apply a window of some pre-determined size, $k*k$, to each pixel, as illustrated in Figure 1. The center of the window slides over every pixel and performs the wavelet transform[†] at each location to determine each pixel's texture feature.

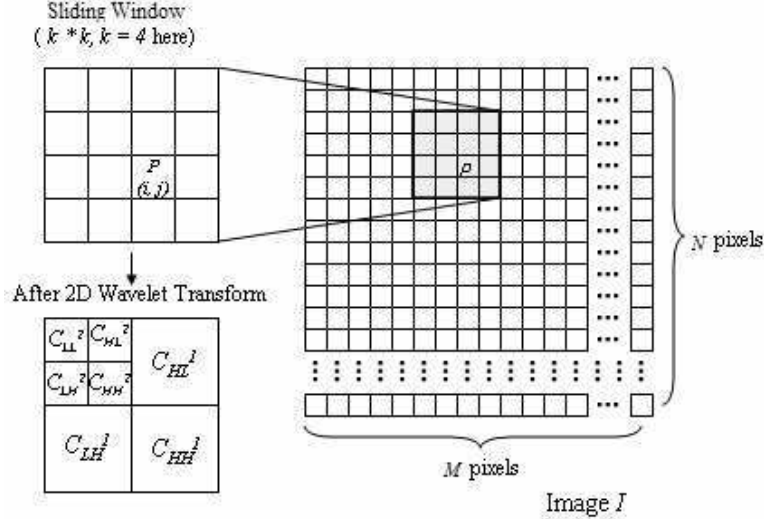


Figure 1: Texture Feature Extraction for pixel $p(i, j)$ (window size is $4*4$)

Throughout this paper we use the following notation. Let I be the whole image, M, N indicate the width and height of the image (as measured in pixels) respectively, and $p(i, j)$ be the pixel for which we want to extract the texture information, where $0 \leq i < M, 0 \leq j < N$. Let $k*k$ be the size of the transform window. Since the texture values are defined in three orientations^{3,4}, the horizontal, vertical and oblique directions, we use $V_i^h(p), V_i^v(p), V_i^o(p)$ to denote the texture in these directions, respectively. Assume $C_s^r(x, y)$ denotes the wavelet coefficient in sub-band s at position (x, y) , after the r^{th} level wavelet transform, where $s \in \{LL, HL, LH, HH\}, 0 \leq x < k, 0 \leq y < k$, and $r \leq \log_2 k$. Then the texture-orientation equations are defined as

$$V_i^h(p(i, j)) = \left(\sum_{r=1}^{\log_2 k} r * \left(\sqrt{\sum_{x=\frac{k}{2}^r}^{\frac{k}{2}^{r-1}-1} \sum_{y=0}^{\frac{k}{2}^{r-1}-1} (C_{HL}^r(x, y))^2} / k * 2^{-r} \right) \right) / \sum_{r=1}^{\log_2 k} r$$

$$V_i^v(p(i, j)) = \left(\sum_{r=1}^{\log_2 k} r * \left(\sqrt{\sum_{x=0}^{\frac{k}{2}^{r-1}-1} \sum_{y=\frac{k}{2}^r}^{\frac{k}{2}^{r-1}-1} (C_{LH}^r(x, y))^2} / k * 2^{-r} \right) \right) / \sum_{r=1}^{\log_2 k} r$$

[†] Throughout this paper, we use Daubechies-4 wavelet transform.

$$V_i^o(p(i, j)) = \left(\sum_{r=1}^{\log_2 k} r * \left(\sqrt{\sum_{x=k/2^r}^k \sum_{y=k/2^r}^k (C_{HH}^r(x, y))^2} / k * 2^{-r} \right) \right) / \sum_{r=1}^{\log_2 k} r$$

This way, we get the texture feature for each pixel in three separate orientations. However, since this method performs a $k*k$ wavelet transform for each pixel instead of each $k*k$ pixel block, it requires $k*k$ times more computation than texture feature extraction on a single block. Texture feature extraction via this method is very computationally intensive for single-pixel texture feature extraction.

3. FAST FEATURE EXTRACTION METHOD

Usually the neighboring pixels in an image are not very distinct (i.e. they are highly correlated). Quite often in an image there are large regions of pixels with nearly the same color, such as the sky, or with uniform texture, such as walls, cloth, or sand. For typical pixel-level texture feature extraction, the texture values for each pixel are computed with the sliding window positioned such that the pixel is the center of the window. Clearly, for images of the type mentioned above, the neighboring pixels have the same (or nearly the same) texture features. In such cases, nearly identical results are generated by performing nearly identical computations. If we can determine beforehand which computations will result in nearly identical results, we can avoid these calculations, trading off decreased computational complexity with a small amount of distortion in the texture extraction results.

We propose a new hierarchical method to reduce the computation complexity and expedite texture feature extraction according to this principle. In our method, an image is divided into blocks of pixels of different granularities at the various levels of the hierarchy. The texture feature extraction hierarchy is represented by a quad-tree structure, in which a block at a higher level is divided into 4 sub-blocks at the next lower level. Starting from the highest level, we examine each block to see if unnecessary computations can be avoided. If a block at a given level of the hierarchy has solid color or uniform texture, we assign each pixel in this block the same texture feature values, which are equivalent to the texture features for the representative pixel[‡] of the block. Otherwise, we examine its sub-blocks in a similar way. Hence, each pixel will get its texture features in one of these blocks at a certain hierarchical level, either by copying the corresponding representative pixel's texture features, or by computing its own texture features if it is a representative pixel itself.

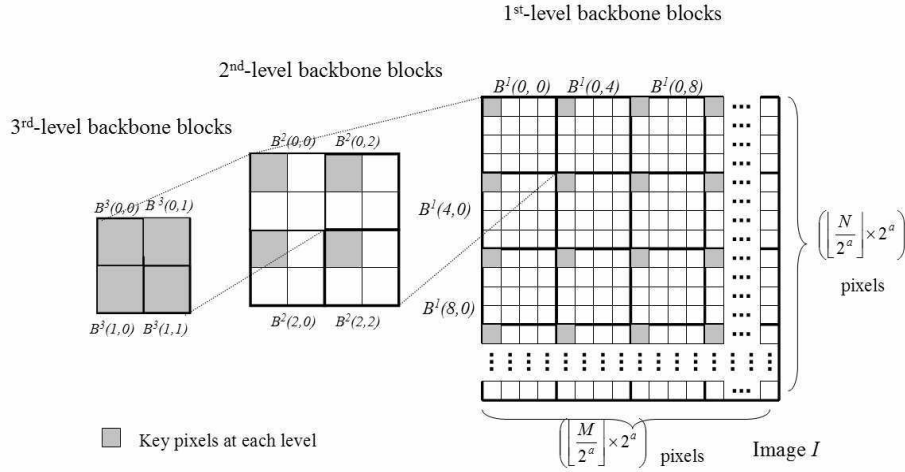


Figure 2: Backbone blocks and key pixels (out-of-block pixels are not included), when $a = 2$, $b = 4$

[‡] Since the texture features of the representative pixel are, by definition, the description of the texture within the block, the representative pixel of a block is actually the key pixel of the block, as defined below.

The whole image can be divided into many non-overlapping $b*b$ pixel blocks ($b = 2^a$, $a \in \mathbb{N}$). We call these blocks 1^{st} level backbone blocks, denoted B^1 . Each B^1 can be divided into 4 $b/2*b/2$ -sized 2^{nd} level backbone blocks, B^2 . Similarly, each B^2 can be divided into 4 $b/4*b/4$ -sized 3^{rd} level backbone blocks, and so on. Each block can be divided further and further until each backbone block contains only one pixel. These single-pixel backbone blocks are $(a+1)^{th}$ level backbone blocks.

For each backbone block B^s , where $1 \leq s \leq (a+1)$, the pixel at its upper-left corner is called its *key pixel*, denoted $P_{key}(B^s)$. If $p(i,j)$ is a key pixel of a s^{th} level backbone block, we call this backbone block $B^s(i, j)$, i.e. $P_{key}(B^s(i, j)) = p(i,j)$. Backbone blocks and key pixels are shown in Figure 2. Notice that the key pixels of larger backbone blocks are always the key pixels of smaller backbone blocks.

If the image dimensions, M and N , are not multiples of b , the size of the 1^{st} level backbone block, there will be some pixels that do not reside in 1^{st} level backbone blocks. We call these pixels *out-of-block pixels*, which we will examine a little later. Except for out-of-block pixels, each pixel resides in one of the 1^{st} level backbone blocks. Since each higher level backbone block is generated by dividing the next lower level block into 4 equal-sized sub-blocks, each pixel must also be in one backbone block at each level. For example, for a pixel $p(i,j)$, at the s^{th} level, where $1 \leq s \leq (a+1)$, it is in backbone block $B^s(\lfloor i/2^{s+a-1} \rfloor * 2^{s+a-1}, \lfloor j/2^{s+a-1} \rfloor * 2^{s+a-1})$, and the corresponding key pixel for this backbone block is $p(\lfloor i/2^{s+a-1} \rfloor * 2^{s+a-1}, \lfloor j/2^{s+a-1} \rfloor * 2^{s+a-1})$. Then, each pixel will get its texture features in one of these backbone blocks, either by copying the corresponding key pixel's texture features, or by computing its own texture features if it is a key pixel itself.

If we let key pixel $p(i,j)$ be the upper-left pixel of a $k*k$ sliding window to which the wavelet transform is applied, we can associate the texture feature extracted by this transform window to $p(i,j)$. We use $V_{TK}^h(p(i,j))$, $V_{TK}^v(p(i,j))$, and $V_{TK}^o(p(i,j))$ to denote the texture features for key pixel $p(i,j)$, namely, the *key pixel texture features*, in the horizontal, vertical and oblique directions, respectively. Observe that this sliding window is the same as the sliding window for extracting texture feature for pixel $p(i+k/2, j+k/2)$ which is described in section 1, and shown in Figure 3, i.e. $V_{TK}^h(p(i,j)) = V_i^h(p(i+k/2, j+k/2))$, $V_{TK}^v(p(i,j)) = V_i^v(p(i+k/2, j+k/2))$, $V_{TK}^o(p(i,j)) = V_i^o(p(i+k/2, j+k/2))$. In this paper, we let $k = b = 2^a$.

So, $V_{TK}^h(p(i,j)) = V_i^h(p(i+2^{a-1}, j+2^{a-1}))$,
 $V_{TK}^v(p(i,j)) = V_i^v(p(i+2^{a-1}, j+2^{a-1}))$,
 $V_{TK}^o(p(i,j)) = V_i^o(p(i+2^{a-1}, j+2^{a-1}))$.

Note, this assignment causes a misalignment between the texture values and their associated pixels, so our algorithm realigns them after assigning all texture values, as discussed below.

For a backbone block B^s , if we let its key pixel $P_{key}(B^s)$ be the upper-left pixel of the sliding window for the wavelet transform, and let the size of the sliding window be 2^{s+a-1} , i.e. the sliding window covers the backbone block exactly, we can associate the texture feature extracted by this transform window to B^s . We use $V_{TB}^h(B^s)$, $V_{TB}^v(B^s)$, $V_{TB}^o(B^s)$ to denote those texture features, namely, the *backbone block texture features*, in the horizontal, vertical and oblique directions, respectively. Observe that this sliding window is the same as the sliding window for extracting key pixel texture feature for pixel $p(i, j)$, where the window size is 2^{s+a-1} , as shown in Figure 3. So,

$$\begin{aligned} V_{TB}^h(B^s) &= V_{TK}^h(P_{key}(B^s)) = V_i^h(p(i+2^{s+a-2}, j+2^{s+a-2})), \\ V_{TB}^v(B^s) &= V_{TK}^v(P_{key}(B^s)) = V_i^v(p(i+2^{s+a-2}, j+2^{s+a-2})), \\ V_{TB}^o(B^s) &= V_{TK}^o(P_{key}(B^s)) = V_i^o(p(i+2^{s+a-2}, j+2^{s+a-2})). \end{aligned}$$

Notice that for 1^{st} level backbone block $B^s(i,j)$, the backbone block texture features and the key pixel texture features for key pixel $p(i, j)$ are the same. Figure 3 shows how to extract the backbone block texture features for a backbone block and how to extract the key pixel texture features for its key pixel.

We use the sum of the three backbone block texture features as the *backbone block texture measure*, $M(B^s)$, to denote the overall texture in block B^s :

$$M(B^s) = \left| V_{TB}^h(B^s) \right| + \left| V_{TB}^v(B^s) \right| + \left| V_{TB}^o(B^s) \right|$$

And, for two backbone blocks B_1^s and B_2^s , we define the Manhattan distance between their key pixel texture features as the *backbone block texture distance*, $D(B_1^s, B_2^s)$, i.e.

$$D(B_1^s, B_2^s) = \left| V_{TK}^h(P_{key}(B_1^s)) - V_{TK}^h(P_{key}(B_2^s)) \right| + \left| V_{TK}^v(P_{key}(B_1^s)) - V_{TK}^v(P_{key}(B_2^s)) \right| \\ + \left| V_{TK}^o(P_{key}(B_1^s)) - V_{TK}^o(P_{key}(B_2^s)) \right|$$

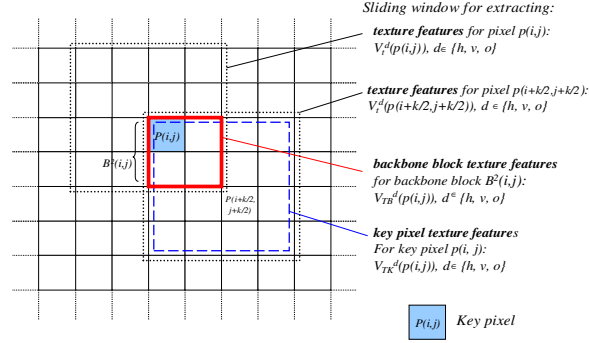


Figure 3: Sliding window for backbone block texture feature and key pixel texture feature extraction. Here $a = 2$, $b = k = 4$, and pixel P is the key pixel for $B^s(i, j)$.

Our fast texture feature extraction method takes advantage of the similarities between the neighboring pixels. First, we divide the whole image into non-overlapping 1st level backbone blocks, and apply the wavelet transform to each 1st-level backbone block. For each backbone block $B^s(i,j)$, we applied the following *Fast Texture Estimation Algorithm*:

1. *Solid Color Estimation*:

For a backbone block $B^s(i,j)$ and a given texture feature threshold T , compute its backbone block texture measure, $M(B^s)$.

If a backbone block's texture measure, $M(B^s)$, is less than the threshold, T , i.e. $M(B^s) \leq T$, it means that the pixels in this block have negligible variation in color, i.e. the texture in block B^s is negligible. So, we can say that every pixel in block B^s has the same texture feature and no further computation is needed to extract texture features for each pixel. Then go to step 4.

If $M(B^s) > T$, there are non-negligible texture values in B^s . Go to step 2.

2. *Uniform Texture Estimation*:

For a backbone block $B^s(i,j)$ and a given texture feature threshold T^{\S} , compute the backbone block texture distances with the right, below and diagonally right-below neighboring backbone block key pixels at the same level, $D(B^s(i, j), B^s(i, j+2^{a-s+1}))$, $D(B^s(i, j), B^s(i+2^{a-s+1}, j))$, and $D(B^s(i, j), B^s(i+2^{a-s+1}, j+2^{a-s+1}))$. Compute the required key pixel texture features, $V_{TK}^r(p)$, $r \in \{h, v, o\}$, if they have not been computed previously.

If all three backbone block texture distances are less than the threshold, i.e. $D(B^s(i, j), B^s(i, j+2^{a-s+1})) \leq T$ and $D(B^s(i, j), B^s(i+2^{a-s+1}, j)) \leq T$ and $D(B^s(i, j), B^s(i+2^{a-s+1}, j+2^{a-s+1})) \leq T$, it means the texture within block $B^s(i, j)$ is rather uniform. So, every pixel in block $B^s(i, j)$ can be assigned the same texture feature values as the key pixel $p(i,j)$, and no further computation is necessary. Then go to step 4.

Else, go to step 3.

3. *Hierarchy Control*:

If backbone block B^s is the $(a+1)$ th level backbone block, i.e. B^s only contains one pixel, no more texture extraction steps are needed for $B^s(i, j)$. Go to step 5.

^{\S} This threshold may be different from the texture feature threshold in step 1. In this paper, we use the same values just for simplicity.

Else, divide $B^s(i, j)$ into 4 $(s+1)^{\text{th}}$ level backbone blocks, namely, $B^{s+1}(i, j)$, $B^{s+1}(i+2^{a-s}, j)$, $B^{s+1}(i, j+2^{a-s})$, $B^{s+1}(i+2^{a-s}, j+2^{a-s})$. For each $(s+1)^{\text{th}}$ level backbone blocks, go to step 1, and run fast texture estimation algorithm for each of them.

4. *Texture Feature Assignment:*

Let each pixel in block $B^s(i, j)$ have the same texture feature values as the key pixel, i.e. $\forall p \in B^s(i, j)$, let $V_i^h(p) = V_{TK}^h(p(i, j))$, $V_i^v(p) = V_{TK}^v(p(i, j))$, $V_i^o(p) = V_{TK}^o(p(i, j))$. No more texture extraction steps are needed for $B^s(i, j)$. Go to step 5.

5. *Block Exit:*

Exit texture extraction for $B^s(i, j)$.

When extracting the texture features for a pixel, the pixel should be positioned at the center of the window, as shown in Figure 1 (and in Figure 3 again). In our fast extraction method, we instead associate the texture features to the pixel in the upper-left corner, i.e. $\forall d \in \{h, v, o\}$, $V_{TK}^d(p(i, j)) = V_i^d(p(i+k/2, j+k/2))$, as show in Figure 3. Therefore, after each pixel is assigned its texture values with the above algorithm, we have to make some modifications---let each pixel $x(i, j)$ have the texture value of $x(i-k/2, j-k/2)$. Before the shift, a pixel's color features and its texture features may be mis-aligned. After modification, every pixel will have its own local texture feature.

Observe that the fast texture estimation algorithm cannot assign texture feature values for the out-of-block pixels, but some of the out-of-block pixels may get their texture feature values in the alignment process described above. For the pixels that still don't have texture feature values after all those processing steps, we can either start with smaller sliding windows and extract texture features for each of them individually, or we can just use the texture features of the nearest key pixel as its texture features, as we did in this paper. If the width of the out-of-block pixels at the right margin of the image, and the height of out-of-block pixels at the lower margin of the image are both smaller than half of the size of the 1st level backbone block, i.e. $(M - \lfloor M/b \rfloor * b) \leq b/2$ and $(N - \lfloor N/b \rfloor * b) \leq b/2$, there are only $(M - \lfloor M/b \rfloor * b + N - \lfloor N/b \rfloor * b) * b/2$ above-mentioned pixels remaining; if the width of out-of-block pixels at the right margin is larger than half of the size of the 1st level backbone block, i.e. $(M - \lfloor M/b \rfloor * b) > b/2$, there are $(M - \lfloor M/b \rfloor * b - b/2) * (\lfloor M/b \rfloor * b)$ more remaining; if $(N - \lfloor N/b \rfloor * b) > b/2$, there are $(N - \lfloor N/b \rfloor * b - b/2) * (\lfloor N/b \rfloor * b)$ more remaining. These numbers are usually small. If $b = 4$, the largest possible number of these out of block pixels is $(M+N+7)$, which is only 1.002% of a 203*203 image. If $b = 8$, the largest possible number is $(3M+3N+23)$, which is only 3.011% of a 203*203 image. Hence, our methods for extracting texture features for them are reasonable.

With this algorithm, we can quickly extract the texture features for each pixel in the image. After we extract the color feature and texture feature information for every pixel, we can begin the grouping and the merging process to segment the whole image.

4. THE SPEEDUP OF THE FAST FEATURE EXTRACTION METHOD

To evaluate the performance of the fast texture feature extraction algorithm, we used it in our hierarchical image segmentation algorithm². This segmentation method assumes a hierarchical tree-like representation in which a node at a higher level represents a group of nodes at the next lower level, thus the root level represents the full image and each leaf node represents a small block of pixels or even a single pixel. A depth-first clustering algorithm is used to build the segmented hierarchical representation according to the similarities between the feature vectors of neighboring nodes at each level of the hierarchy. The feature vectors include color and texture information as described in Section 2 and 3.

Table 1 show the images which were segmented. Their sizes are included in the parentheses. The color and texture features for each pixel were extracted in the first stage of segmentation. The texture feature extraction times (in μsec) are illustrated in Figure 4.

As we can see in Figure 4, the texture feature extraction time decreases as texture threshold increases, as expected. The feature extraction times for "Dinosaur" and "Tower" are less than those of "Red Rose" and "Mountain", because they have larger portions of background with more homogeneous features, so less computation is needed.

When the texture threshold T is rather large, say 100,000 in our experiment, the backbone block texture measure and the backbone block texture distance will always be smaller than the threshold. So, no texture feature extraction beyond the level of the 1st-level backbone blocks is necessary, and the extraction time becomes approximately equivalent to block-level texture feature extraction times. Conversely, when the texture threshold T is 0, the texture features have minimum distortion. However, the computation time will be much greater, though potentially less than computing texture features for all individual pixels, since there may be backbone blocks in which there is no texture variation over a wide area in the image.

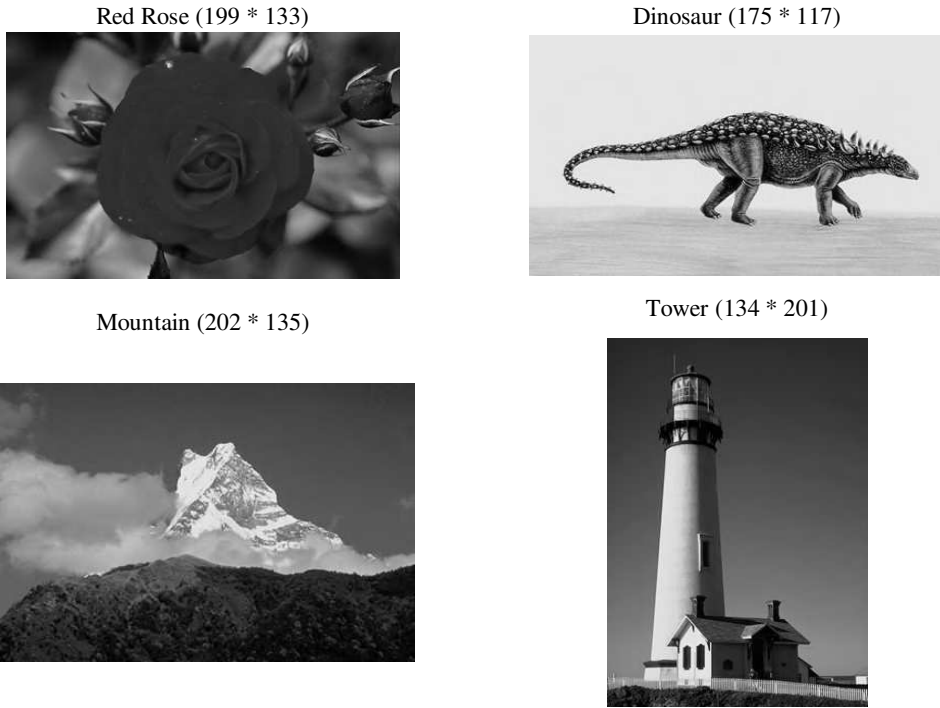


Table 1: The images used in our experiments and their respective size

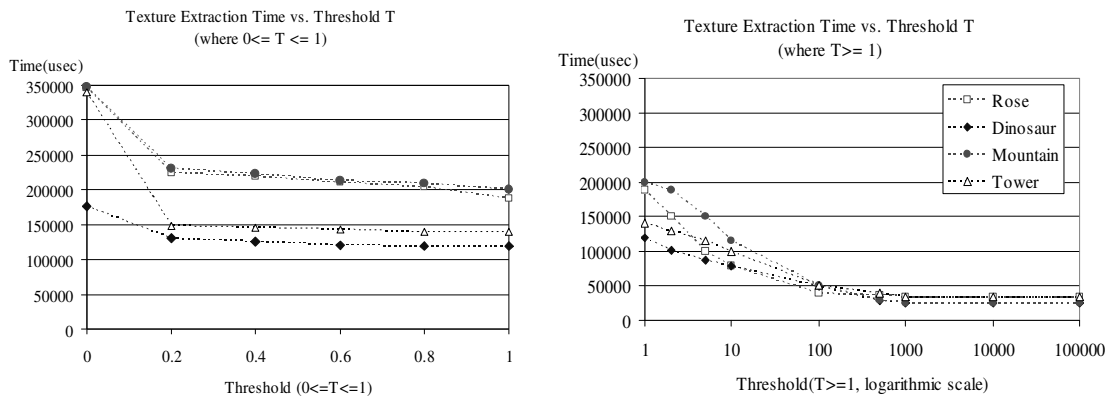


Figure 4: The texture feature extraction time ($\mu sec.$) vs. the texture threshold. Notice that the x-axis is on logarithmic scale when threshold $T \geq 1$.

Table 2 shows the results comparing the computation time for texture feature extraction using the fast texture feature extraction algorithm versus extracting texture feature at the backbone level or pixel level. In this experiment the block size is 4*4 pixels. Normalizing the extraction time for block-level feature extraction to 1, the time for pixel-level (without the fast texture feature extraction) is 16. However, using our fast texture feature extraction method, the normalized extraction times for threshold $T = 0$ are 8.12~11.87. If we further relax the condition on distortion by increasing the texture threshold, the normalized time decreases significantly to 4.67~7.82 when the threshold T is 0.2. When threshold is 1, the normalized time further decreases to 4.30~6.76. The “Dinosaur” image has the lowest computation time when the threshold is 0, because it has a large background with homogenous texture. The “Tower” image has relatively large regions of the background that vary only marginally, which is why its normalized computation time decreases remarkably when the threshold is increased to 0.2.

Image	Pixel-level time	Threshold=0	Threshold=0.2	Threshold=1	Block-level extract time
Red Rose	16	11.87	7.82	6.46	1
Dinosaur	16	8.12	4.67	4.30	1
Mountain	16	11.39	7.42	6.76	1
Tower	16	10.74	5.86	5.21	1

Table 2: The normalized extraction time

As can be seen in Figure 4, the computation time curve for extracting texture features was similar for all four images. The extraction time decreases as the texture threshold increases, especially when the threshold is less than 1. The times decreased particularly dramatically for thresholds less than 0.2. The reduction in extraction time can be seen in Table 3. When the texture threshold is 0.2, the extraction time decreases by 27.79%~56.5%, depending on the content of the images. When the threshold is 1, the extraction time further decreases by 35.84%~59.92%. The “Tower” image, in particular, has a wide area of the background that varies only slightly, so its extraction time decreases quickly at texture thresholds below 0.2.

Image	Threshold=0	Threshold=0.2	Time Reduction (%)	Threshold=1	Time Reduction (%)
Red Rose	345281	227493	34.11%	187829	45.6%
Dinosaur	174558	126051	27.79%	111997	35.84%
Mountain	339837	221667	34.77%	202035	40.55%
Tower	339092	147526	56.5%	135897	59.92%

(The percentage is the reduced time when threshold = 0.2 and 1 over the time when threshold = 0)

Table 3: The reduction of extraction time when texture threshold = 0.2 and 1 (Unit: $\mu sec.$)

From the analysis above, we can see that this fast texture feature extraction method greatly decreases the computational complexity and extraction time for pixel-level texture feature extraction.

5. THE FEATURE EXTRACTION EFFECTIVENESS OF THIS METHOD

Performing fast texture feature extraction with a threshold does introduce some distortion, effectively allowing the algorithm to trade off distortion with extraction time. To examine the degree of distortion introduced by our method, we examined the difference between the wavelet coefficients generated by the fast extraction method and those generated by the traditional pixel-level wavelet transform, across a range of thresholds.

5.1 The Distortion Percentage of Fast Extraction Method

First, we shall define the Distortion Percentage. Suppose the image to be segmented is of size $M*N$. We use $V_s^T(i, j)$ to denote the wavelet coefficient of the pixel at position (i, j) (where $0 \leq i < M$, $0 \leq j < N$) in sub-band s ($s \in \{HL, LH, HH\}$), extracted with the threshold T . Then, the Distortion Percentage DP_s^T for some sub-band s and extraction threshold T is:

$$DP_s^T = \frac{\left(\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |V_s^T(i, j) - V_s^0(i, j)| \right) / (M * N)}{\left(\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} V_s^0(i, j) \right) / (M * N)} = \frac{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |V_s^T(i, j) - V_s^0(i, j)|}{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} V_s^T(i, j)}$$

Low Distortion Percentages mean that the extracted wavelet coefficients are close to the original coefficients generated by traditional pixel-level texture feature extraction. A Distortion Percentage of 0 indicates they exactly match. Conversely, high Distortion Percentages indicate significant variation from the original wavelet coefficients.

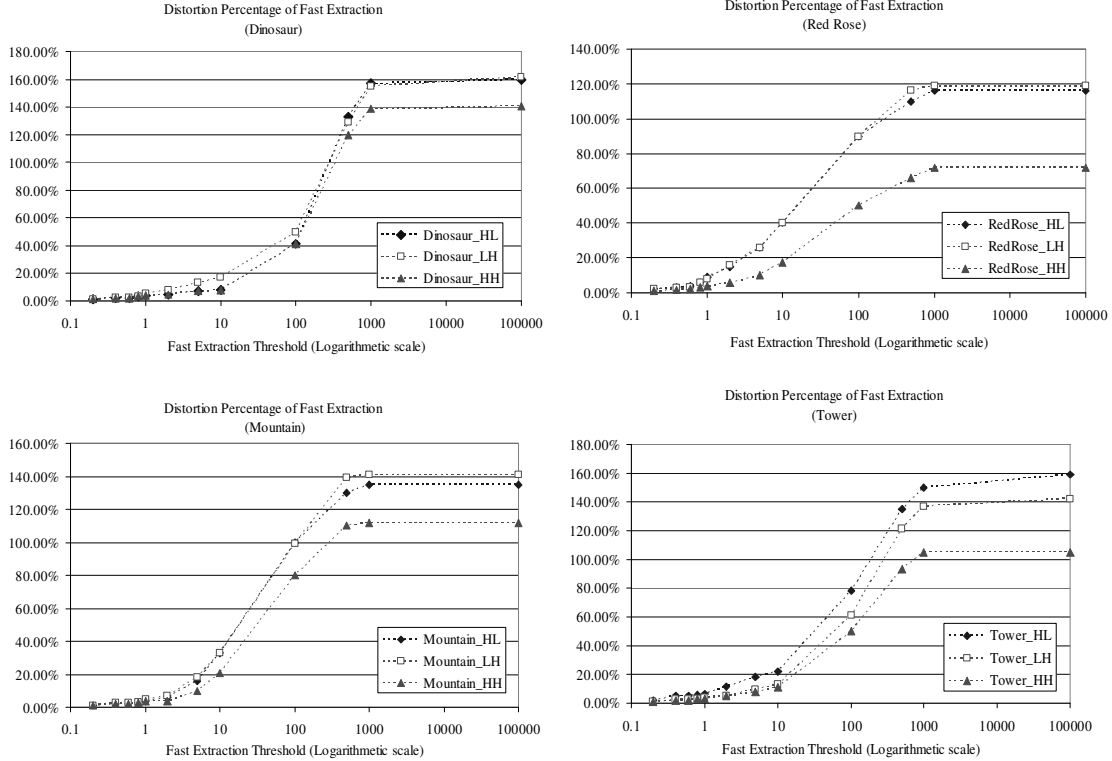


Figure 5: The Distortion Percentages of Fast Extraction method for “Dinosaur”, “Red Rose”, Mountain” and “Tower”. Notice that the x-axes are on logarithmic scale.

Figure 5 shows the Distortion Percentage of fast extraction method for “Dinosaur”, “Red Rose”, “Mountain” and “Tower” respectively. For all sub-bands, if the extraction threshold is in [0, 1], the Distortion Percentages are usually very low, less than 10%; for thresholds in the range (1, 5), the Distortion Percentages are generally below 20%. As we can see from Figure 5 and Table 3, when the threshold is 0.2, the Distortion Percentage is 0.6%~3.7%, while the reduction of computation time is 27.8%~56.5%; when the threshold is 1, the Distortion Percentage increases to 2.2%~7.7%, while the reduction in computation time is 35.8%~59.9%. So, fast texture extraction can be used to speed up the feature extraction process with only a small cost in feature distortion.

5.2 The Fast Extraction Signal Noise Ratio

The other measure that we used to evaluate the amount of distortion is the *Fast Extraction Signal Noise Ratio (FESNR)*. Again, for an image with size $M*N$, we use $V_s^T(i, j)$ to denote the wavelet coefficient of the pixel at position (i, j) ($0 <= i < M, 0 <= j < N$) in sub-band s ($s \in \{HL, LH, HH\}$), extracted with the threshold T . Then, the Fast Extraction Signal Noise Ratio, $FESNR_s^T$ for sub-band s and texture threshold T is:

$$FESNR_s^T = \frac{\left(\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (V_s^T(i, j))^2 \right) / (M * N)}{\left(\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (V_s^T(i, j) - V_s^0(i, j))^2 \right) / (M * N)} = \frac{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (V_s^T(i, j))^2}{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (V_s^T(i, j) - V_s^0(i, j))^2}$$

Here, higher *FESNR* means the extracted coefficients are closer to the original coefficients generated at pixel level.

Figure 6 shows the *FESNRs* for “Dinosaur”, “Red Rose”, Mountain” and “Tower”. When the fast extraction threshold is less than 1, the *FESNR* is usually very high, and with increasing of threshold, the *FESNR* decreases slightly. Above the extraction threshold 1, the *FESNR* decreases much more quickly. From these results, we can see that it is best to use a fast texture extraction threshold in the range $[0, 1]$. It greatly increases the extraction speed, but with only a small amount of distortion.

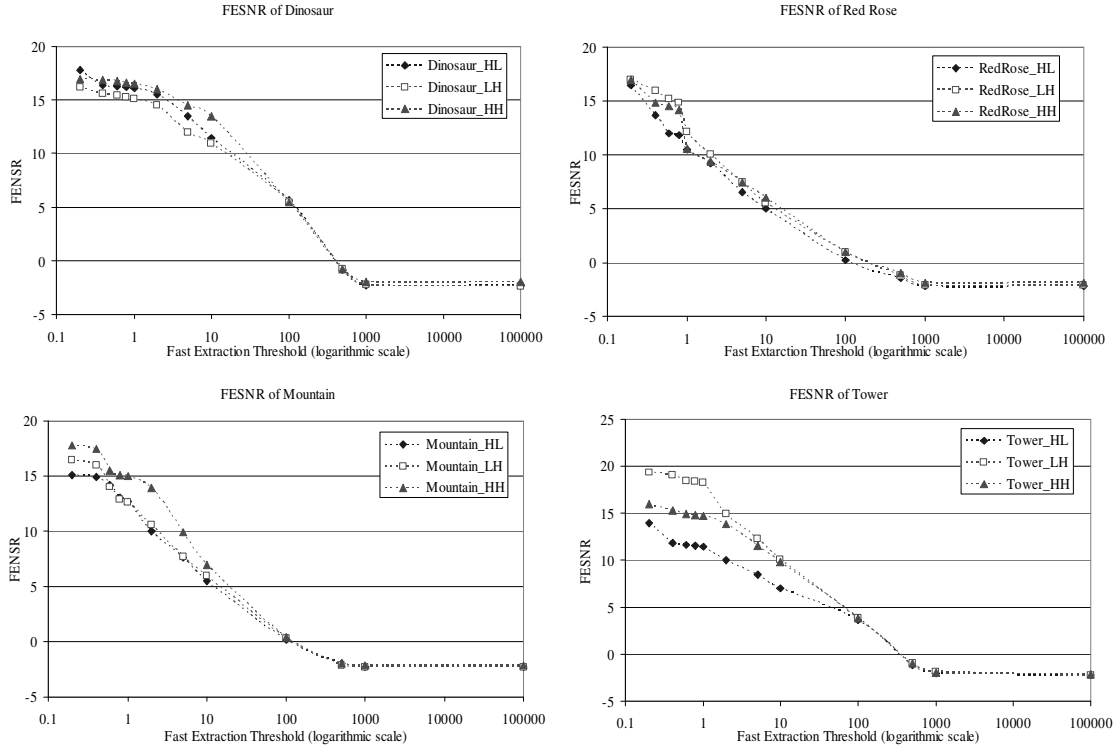


Figure 6: The *FESNRs* for “Dinosaur”, “Red Rose”, Mountain” and “Tower”. Notice that the x -axes are on logarithmic scale.

5.3 The Final Segmentation with Fast Extraction Results

While the above results examine the distortion in the texture feature coefficients directly, we are also interested in how they affect segmentation performances. We once again use our hierarchical segmentation method to generate the

segmented image, and then evaluate the segmentation results with several objective segmentation evaluation methods, including the F function proposed by Liu and Yang⁵, the Q functions proposed by Borsotti et. al⁶, and an entropy-based function E proposed by Zhang, Fritts and Goldman⁷. A full discussion and examination of these methods was recently performed by Zhang et. al⁷. For all these evaluation functions, segmentations that result in small values are considered more favorable (i.e. better segmentations) than those with high values.

The F , Q and E for “Red Rose” and “Tower” are shown in Table 4 and 5. For all results shown here only the fast feature extraction threshold is varied. The number of regions is set at 10 to minimize the bias from a large number of regions. Although the results from these evaluation functions are not always consistent with each other, they generally showed that the segmentations with fast feature extractions do not necessarily mean inferior results. During the clustering process of our image segmentation algorithm, the regions are compared and merged according to the similarities in their texture and color feature values. The merging of feature values blurs the impact that was brought about by large texture thresholds here.

	0	0.2	0.4	0.6	0.8	1
F	0.1240 (5)	0.1376 (6)	0.1205 (4)	0.1023 (1)	0.1057 (2)	0.1176 (3)
Q	0.7324 (2)	0.8507 (5)	0.7553 (3)	0.7887 (4)	0.8986 (6)	0.7142 (1)
E	8.1702 (1)	8.1892 (5)	8.1889 (4)	8.1920 (6)	8.1733 (2)	8.1747 (3)

Table 4: The objective evaluation values and ranks (in parentheses) for “Red Rose” while the texture threshold varies

	0	0.2	0.4	0.6	0.8	1
F	0.1102 (6)	0.0097 (3)	0.0089 (1)	0.0091 (2)	0.1057 (5)	0.1009 (4)
Q	0.6391 (6)	0.5743 (4)	0.5323 (2)	0.5008 (1)	0.5875 (5)	0.5741(3)
E	8.0527 (1)	8.0866 (6)	8.0665 (2)	8.0850 (5)	8.0843 (4)	8.0798 (3)

Table 5: The objective evaluation values and ranks (in parentheses) for “Tower” while the texture threshold varies

Figure 7 shows the actual segmentation results for “Red Rose” image when the fast texture feature extraction threshold T varies from 0 to 1. Comparing these segmentation results with the original image in Table 1, we can see that using higher fast feature extraction threshold does not lead to inferior segmentation results. The segmentation result with threshold $T = 0.4$ actually is better than the result with threshold 0.2; and using threshold $T = 1$ creates better segmentation result than using threshold 0.6 and 0.8. Overall, the differences between the segmentation results across thresholds in the range 0 to 1 are minor.

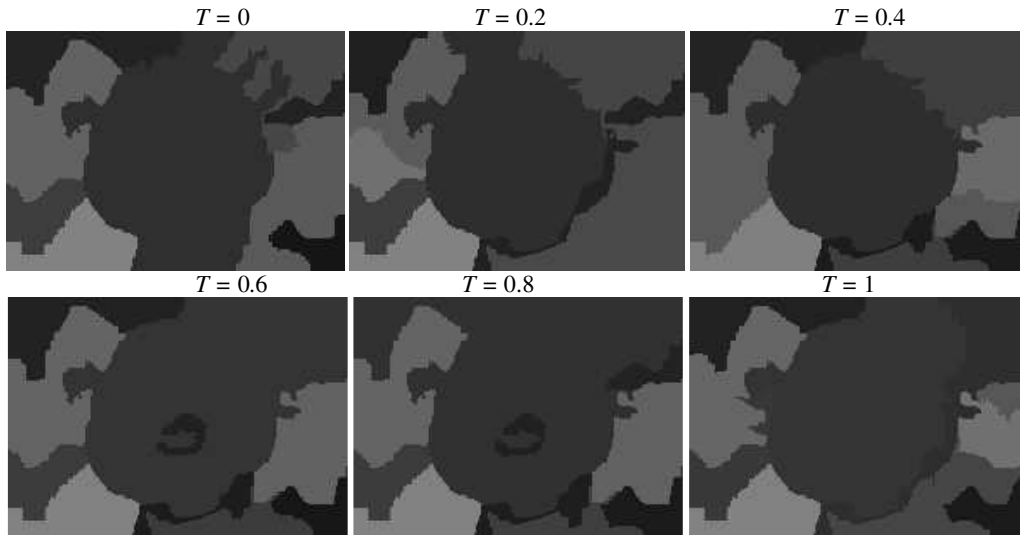


Figure 7: The segmentation results of “Red Rose” when fast texture extraction threshold varies from 0 to 1

So, for the whole segmentation system, using fast feature extraction can speed up the process, and it may at the same time help improve the segmentation results, which makes it a good alternative to the current feature extraction method.

6. CONCLUSION AND FUTURE WORK

In image segmentation, pixel-level feature extraction is required for a good segmentation performance. However, traditional texture feature extraction is very computationally intensive for individual pixels. In this paper, we propose a novel fast texture feature extraction method which takes advantage of the similarities between neighboring pixels. By carefully choosing the texture feature extraction threshold T , we can greatly increase the extraction speed while keeping the distortion within a reasonable range. The texture feature extraction threshold T is user-defined, so we can balance the extraction speed and the distortion according to specific situations.

However, there are still some problems. Even if we let the texture feature extraction threshold T be 0, for most images we cannot get exactly the same result as using traditional pixel-level feature extraction. Since the sliding window for key pixel texture feature is fixed to be $b*b$, whereas the sliding window for backbone block pixel feature is related to the size of the backbone block, using the backbone block texture measure, $M(B^s)$, to measure the texture, but using the key pixel texture feature to describe all pixels when $M(B^s)$ is small, will inevitably result in distortion when there is texture in B^s 's lower level backbone blocks. One solution to this problem is to use variable-sized sliding window for the key pixel texture feature, but this leads to inconsistent feature accuracy. Or, we could remove the first step in the fast texture feature extraction algorithm, which solves this problem, but results in slower feature extraction.

The other problem is that while we can measure the overall distortion from fast texture feature extraction, and its overall impact on the final segmentation, we still don't know how it affects the segmentation locally. And, since the objective evaluation methods are generally not as capable of comparing segmentations with minor differences, a more appropriate evaluation method to fully justify the advantages and disadvantages of the fast extraction method with respect to segmentation accuracy is still needed.

REFERENCES

- ¹ Trygve Randen, and John Hakon Husoy, "Filtering for Texture Classification: A Comparative Study", *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 21, no. 4, pp. 291-310, April 1999
- ² E. Saber and A. M. Tekalp, "Integration of Color, Shape, and Texture for Automatic Image Annotation and Retrieval," *Journal of Electronic Imaging (special issue)*, vol. 7, no. 3, pp. 684-700, July 1998
- ³ M. Unser, "Texture Classification and Segmentation Using Wavelet Frames", *IEEE Trans. on Image Processing*, Vol. 4, No. 11, Nov. 1995, 1549-1560.
- ⁴ James Wang, Jia Li, and Gio Wiederhold, "SIMPLIcity: Semantics-Sensitive Integrated Matching for Picture Libraries", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 9, pp. 947-963, Sept. 2001
- ⁵ Jianqing Liu and Yee-Hong Yang, "Multi-resolution color Image segmentation", *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 16(7), pp. 689-700, 1994
- ⁶ M. Borsotti, P. Campadelli, and R. Achettini, "Quantitative evaluation of color image segmentation results", *Pattern recognition letters*, 19, pp.741-747, 1998
- ⁷ Hui Zhang, Jason Fritts, and Sally Goldman, "An Entropy-based Objective Evaluation Method for Image Segmentation", *Storage and Retrieval Methods and Applications for Multimedia. Proceedings of the SPIE*, vol. 5307, pp. 38-49, 2003.