Chapter 3

Approximation and Generation of Digital Images Using Adaptive Triangular Meshes

This chapter presents two new techniques for approximating digital images with adaptive triangular meshes. The approximated images can represent gray-level images, range images or digital elevation maps. The first technique approximates a given image with an adaptive triangular mesh guaranteeing a maximum tolerance with respect to the original image. The second technique approximates an image with an adaptive triangular mesh keeping image discontinuities and avoiding optimization, although it does not ensure a maximum error. Additionally, this chapter presents two new techniques for the generation of digital images from triangular meshes. The first one samples the given adaptive triangular mesh uniformly at as many pixels as pixels the original image has. The second technique generates an image from an adaptive triangular mesh by taking advantage of graphics hardware acceleration.

This chapter is organized as follows. Section 3.1 introduces the proposed image approximation techniques. Section 3.2 gives some basic definitions that will be used throughout this dissertation. Section 3.3 describes two image approximation techniques by using adaptive triangular meshes. Section 3.4 describes two techniques for the generation of images from triangular meshes. The contents of this chapter are finally summarized in Section 3.5.

3.1 Introduction

Images are commonly transferred and stored in a compact form through well-known representations, such as *gif* and *jpeg*. Unfortunately, these representations are not well-suited for applying further processing operations directly in the compressed domain. Therefore, images codified in those formats must be uncompressed prior to being able to apply image processing operations upon them, no matter how big and redundant the images are. Nonetheless, some researchers have managed to apply various basic operations, such as arithmetic, scaling and feature extraction, upon such compressed representations (Section 2.7).

An alternative solution to the problem of compactly representing images consists of the utilization of geometric representations, such as triangular meshes. Those meshes allow to model large areas of pixels with basic geometric primitives. For example, a large white region can be represented by a few triangles instead of by tens of thousands of pixels. The meshes obtained in this way adapt to the features of the input images by concentrating points in areas of high curvature and by dispersing them over low variation regions. Geometric representations are applicable since the pixels of an image can be considered to be 3D points in a space in which coordinates x and y are functions of the rows and columns of the image, and coordinate z corresponds to the pixel's property (such as gray level, range or terrain elevation). Several algorithms have been proposed for approximating images by using triangular meshes (see Section 2.6.1 and Section 2.6.2).

Besides being able to model digital images, triangular meshes can also be used for simplifying and accelerating general image processing operations. The advantage of geometric representations is that further processing operations can be directly applied in the 3D geometric domain. For instance, scaling, translation and rotation operations can be simply implemented by applying affine transformations to the 3D coordinates of the vertices that constitute the meshes. An algorithm that uniformly samples the resulting meshes suffices to recover the corresponding images. Several fast techniques for processing images approximated by triangular meshes have been proposed in the literature, as described in Section 2.6.

The next section presents some definitions that will be used throughout this dissertation.

3.2 Basic Definitions

A *digital image* is a two dimensional array **I**, where each array element $\mathbf{I}(r, c)$, $r \in [0, R)$ and $c \in [0, C)$, which is referred to as a *pixel*, is a scalar that represents a certain physical magnitude (brightness, range, elevation, ...) measured by a sensor. R and C represent the number of rows and columns of the digital image respectively.

When the sensor utilized to acquire the digital image allows to obtain pixel values representing a certain gray level or intensity between black and white, the acquired digital image is known as a *gray-level image*. Alternatively, if the pixel values represent the distance from a point on the surface of a 3D object to a virtual plane referred to the sensor utilized to acquire the image, the digital image is known as a *range image* (also referred to as a *depth image*). Another important type of digital images, typically used in digital photogrammetry, is known as *digital elevation maps* (*DEM*). The pixel values in a DEM model represent terrain elevations from ground positions at regularly spaced horizontal intervals, which allow to model terrain.

Each pixel of a digital image can be considered to be a *point* of coordinates (x, y, z), defined in a 3D space in which coordinates x and y represent the pixel's column and row, and coordinate z represents the pixel value: $(x, y, z) = (c, r, \boldsymbol{I}(r, c))$. The maximum value that a pixel can take in a digital image will be referred to as Z_{MAX}. A digital image with all its pixels being equal to zero defines a plane in the aforementioned 3D space. This plane will be referred to as the *reference plane* of **I** (also referred to as the *xy* reference plane).

A *3D triangular mesh* is a piecewise linear surface consisting of triangular faces connected along their edges. Formally, a 3D triangular mesh M is a set $\{V, T\}$, where

$V = \{v_1, \ldots, v_m\}$, *m* being the number of vertices in the mesh, is a set of vertex positions that define the shape of the mesh in $\mathbf{R}^3$, and $T = \{t_1, \ldots, t_n\}$, *n* being the number of triangles in the mesh, is a set of triangles that define the mesh topology. Each vertex v_i is defined by three coordinates (x, y, z), and each triangle t_j is defined by three different vertices.

A *2$^1/_2$D triangular mesh* is a particular case of a 3D triangular mesh in which all the triangles are projectable without overlap onto the *xy* reference plane ($z = 0$).

An *approximating image* $\hat{\mathbf{I}}$ is a digital image generated from a 2$^1/_2$D triangular mesh, *M*, which approximates a given digital image **I** (the *approximated image*). Each element of the approximating image, $\hat{\mathbf{I}}(r, c)$, is obtained as the intersection between the triangular approximation, *M*, and a straight line orthogonal to the reference plane of **I** and passing through point $(c, r, 0)$. If this intersection does not occur, the value of $\hat{\mathbf{I}}(r, c)$ is set to zero. The latter takes place when the boundary of the projection of *M* over the reference plane is not a rectangle.

The *approximation error* $\xi(r, c)$ associated with each digital image pixel is defined as the difference between the approximated image point, $\mathbf{I}(r, c)$, and the approximating image point $\hat{\mathbf{I}}(r, c)$:

$$\xi(r, c) = \left|\mathbf{I}(r, c) - \hat{\mathbf{I}}(r, c)\right| \tag{3.1}$$

Figure 3.1 shows an example that illustrates the computation of the approximation error $\xi(r, c)$ corresponding to a pixel $\hat{\mathbf{I}}(r, c)$ of the triangular mesh *M*.

Considering the previous approximation error, the *Root Mean Square Error* (*RMSE*) between the approximated image **I** and the approximating image $\hat{\mathbf{I}}$ is obtained as:

$$RMSE = \sqrt{\frac{\sum_{r=0}^{R-1} \sum_{c=0}^{C-1} \xi^2(r, c)}{R \cdot C}} \tag{3.2}$$

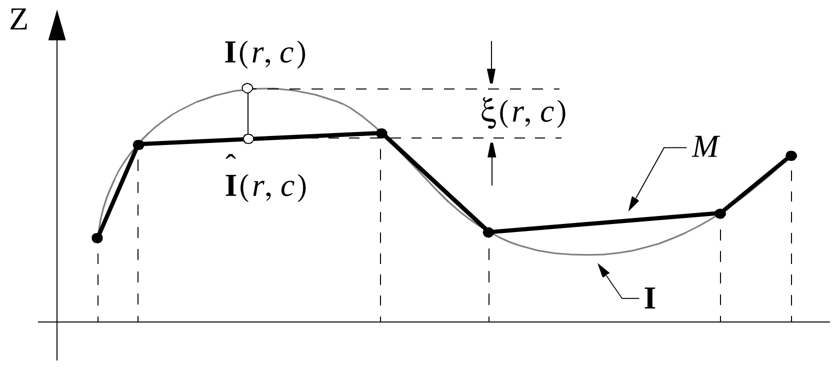

Figure 3.1. Computation of the approximation error $\xi(r, c)$. The thick polygonal line represents a 2D section of the triangular mesh M that approximates the original image $\mathbf{I}$.

3.3 Image Approximation with Adaptive Triangular Meshes

This section presents two new techniques to approximate digital images (images hereafter) with adaptive triangular meshes. The obtained adaptive triangular meshes are compact representations that model the original images with many fewer points. Hence, image processing operations applied upon those triangular meshes may perform faster than upon the original images.

The first technique applies a refinement algorithm in order to approximate a given image with an adaptive triangular mesh. The proposed algorithm ensures a maximum RMS error or *tolerance* with respect to the original image. Several algorithms have been proposed in the literature to solve this problem, as it was described in Section 2.4. Those algorithms apply split and merge techniques that decide what triangles are split or merged. They are based on iterative optimization, in the sense that, at every iteration, they try to find the best triangle to be subdivided or the best triangles to be merged. Such an approach is CPU intensive.

Alternatively, the objective of the first proposed technique is the generation of an adaptive triangular mesh that approximates an image with a given tolerance, by applying an iterative algorithm that utilizes a non-optimization adaptive sampling technique at each iteration. This allows that the proposed technique converges faster than traditional mesh refinement algorithms.

The second technique proposed in this dissertation approximates images with discontinuity-preserving adaptive triangular meshes avoiding optimization techniques. In order to tackle this problem, different techniques have been proposed in the literature. García, Sappa and Basañez (1997b) propose an efficient algorithm for generating adaptive triangular meshes from range images without optimization. This technique samples a predefined number of pixels by considering the curvatures present in the given range image. A problem of this technique is that the discontinuities (contours) contained in the image are not specifically considered when the triangular mesh is generated. In this way, image contours are not sufficiently preserved.

Unlike the technique proposed by García, Sappa and Basañez (1997b), the second technique proposed in this dissertation explicitly models discontinuities. Hence, it is more suitable for approximating gray-level images, since it takes into account both image curvatures and discontinuities.

The adaptive triangular meshes generated with any of the proposed techniques model the given image with the constraint that the triangles do not overlap when they are projected onto the reference plane of the image. Hence, each image pixel has a unique value in the generated mesh.

The next section describes a technique for approximating digital images with adaptive triangular meshes ensuring a maximum RMS error.

3.3.1 Approximation of Digital Images with Bounded Error Adaptive Triangular Meshes

This section presents an iterative algorithm for approximating digital images with $2^1/_2$D adaptive triangular meshes, guaranteeing a maximum root-mean-square (RMS) error or tolerance with respect to the original image. At each iteration, the algorithm applies a non-iterative adaptive meshing technique. In this way, the proposed technique converges faster than traditional mesh refinement algorithms. This technique consists of four main stages that are summarized below.

First, an initial triangular approximation of the given image is obtained through the application of a non-iterative adaptive sampling technique proposed in (García, Sappa, Basañez, 1997a). The initial adaptive triangular mesh is then converted (*backprojected*) to an approximating image by sampling it regularly. Once the approximating image is generated, its RMS error with respect to the given image is obtained. If the RMS error is below a specified tolerance, the iterative process concludes. Otherwise, the regions of the approximating image whose error is above the tolerance (*error regions*) are detected. Afterwards, the same adaptive sampling process is applied upon every error region until the RMS error of the approximating image with respect to the original image is below the given tolerance. The previous steps are further described below.

3.3.1.1 Initial Adaptive Triangulation

Given an image $\mathbf{I}$ with R rows and C columns, an initial triangular approximation is obtained in two stages. In the first stage a predefined number of pixels is chosen from the image by applying a non-iterative adaptive sampling technique proposed in (García, Sappa, Basañez, 1997a). In the second stage, the chosen pixels are triangulated through a 2D Delaunay algorithm (Shewchuck, 1996a). These stages are described below.

Image Adaptive Sampling

First, a *curvature image* $\mathbf{K}$ is computed from the original image $\mathbf{I}$. $\mathbf{K}(r, c)$ gives, for every pixel $\mathbf{I}(r, c)$, an estimation of its curvature. $\mathbf{K}(r, c)$ is generated by merging two curvature estimations obtained along the horizontal, $\mathbf{K}_{rc}^{R}$, and vertical, $\mathbf{K}_{rc}^{C}$, directions of the image. Both $\mathbf{K}_{rc}^{R}$ and $\mathbf{K}_{rc}^{C}$ are calculated starting with two initial estimations proposed in (Yamada, Ishiguro, Uchikawa, 1993):

$$\begin{aligned} \mathbf{K}_{rc}^{C'} &= |\mathbf{I}(r, c-1) - 2\mathbf{I}(r, c) + \mathbf{I}(r, c+1)| \\ \mathbf{K}_{rc}^{R'} &= |\mathbf{I}(r-1, c) - 2\mathbf{I}(r, c) + \mathbf{I}(r+1, c)| \end{aligned} \tag{3.3}$$

Then, a threshold operator $(\alpha > 0)$ is applied

$$\mathbf{K}_{rc}^{C} = \begin{cases} \mathbf{K}_{rc}^{C'} & \mathbf{K}_{rc}^{C'} \leq \alpha \\ \alpha & \text{otherwise} \end{cases} \qquad \mathbf{K}_{rc}^{R} = \begin{cases} \mathbf{K}_{rc}^{R'} & \mathbf{K}_{rc}^{R'} \leq \alpha \\ \alpha & \text{otherwise} \end{cases} \tag{3.4}$$

The curvature estimation is finally calculated as the logical addition of the binary representation of the previous terms:

$$\mathbf{K}(r, c) = \mathbf{K}_{rc}^{R} \vee \mathbf{K}_{rc}^{C} \tag{3.5}$$

According to this formulation, $\mathbf{K}(r, c)$ is a scalar between zero and α. The larger that value is, the larger the curvature at $\mathbf{I}(r, c)$ will be. Parameter α has been set to 255 in order to store the curvature estimation as an 8-bit image while maintaining enough resolution. Considering that the background pixels must also be sampled, they have been given a constant curvature value to guarantee that background regions are sampled and also to prevent large concentrations of pixels near discontinuities.

Figure 3.2 shows an example of a real gray-level image and its associated curvature image computed according to this technique. White regions represent high values of $\mathbf{K}(r, c)$, which correspond to areas with high curvature. On the other hand, dark regions represent low values of $\mathbf{K}(r, c)$, which correspond to homogeneous regions.

After the curvature image has been computed, both the original image $\mathbf{I}$ and the curvature image $\mathbf{K}$ are divided into a predefined number of rectangular *tiles*. In particular, the images are divided into H horizontal and V vertical partitions.

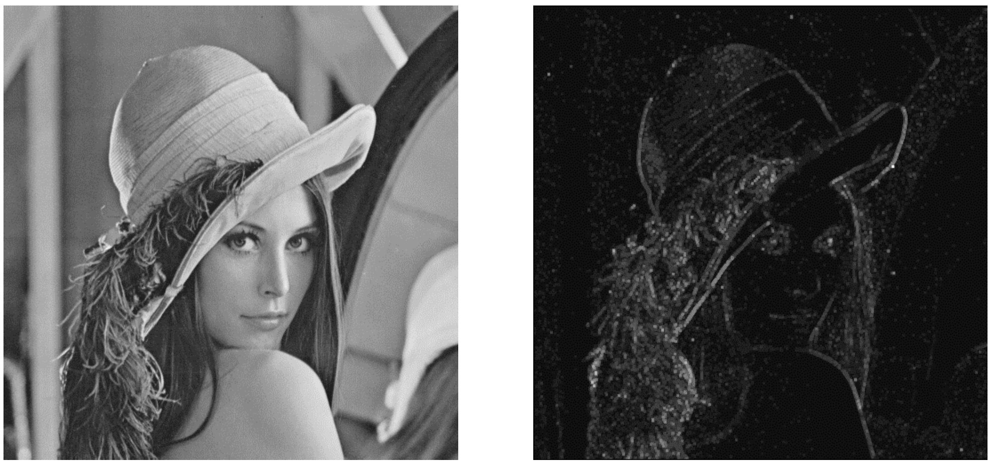

Figure 3.2. (*left*) Original gray-level image with 262,144 pixels (512x512). (*right*) Corresponding curvature image.

Taking this into account, a tile or *window* W_{vh}, $v \in [0, V-1], h \in [0, H-1]$, is defined by two 2D points: an upper-left corner (ulr_{vh}, ulc_{vh}) and a bottom-right corner (brr_{vh}, brc_{vh}). Considering integer division, their coordinates are computed as:

$$\begin{aligned} ulr_{vh} &= \frac{R}{V}v \qquad ulc_{vh} = \frac{C}{H}h \\ brr_{vh} &= \begin{cases} \frac{R}{V}(v+1) & v < V-1 \\ R-1 & v = V-1 \end{cases} \\ brc_{vh} &= \begin{cases} \frac{C}{H}(h+1) & h < H-1 \\ C-1 & h = H-1 \end{cases} \end{aligned} \tag{3.6}$$

Let $\mathbf{I}_{vh}$ be the image tile defined by a window W_{vh} when the latter is applied to the original image $\mathbf{I}$. Every pixel of the image tile $\mathbf{I}_{vh}$ is defined as follows:

$$\mathbf{I}_{vh}(r', c') = \mathbf{I}(r' + ulr_{vh}, c' + ulc_{vh}) \tag{3.7}$$

Both parameters (r', c') are local to the tile: $r' \in [0, R/V]$, $c' \in [0, C/H]$. Let also $\mathbf{K}_{vh}$ be the curvature image tile defined by the same window on the curvature image $\mathbf{K}$.

The objective at this point consists of choosing a predefined array of pixels with $\mathcal{R}$ rows and ζ columns from every image tile $\mathbf{I}_{vh}$. Taking advantage that each tile can be considered to be a small image with its corresponding curvature estimation, this process can be run in parallel by using high-performance computer architectures. Following the technique proposed in (García, Sappa, Basañez, 1997a; Sappa, 1999, pp. 44), $\mathcal{R}$x ζ pixels are sampled as follows.

A set of ζ pixels is selected from each row of pixels of the tile based on their curvature estimation. Let $\mathbf{I}_{vhr'}$ be a row of pixels of a given image tile ($\mathbf{I}_{vhr'}(c')$ is a pixel in that row, $c' \in [0, C/H]$) and let $\mathbf{K}_{vhr'}$ represent the curvatures corresponding to that row. By using the algorithm proposed in (García, 1995a), the curvature profile is mapped to an unnormalized probability density function that represents the probability of selecting each pixel from the image tile, such that pixels with high curvature will have a higher probability of being selected for the mesh. The discrete, unnormalized probability density function $f_{vhr'}(c')$ is defined by applying a transformation function to the curvature profile:

$$f_{vhr'}(c') = \mathcal{T}(\mathbf{K}_{vhr'}(c')) \tag{3.8}$$

The transformation function $\mathcal{T}$ determines the variation of pixel density with respect to the variation of curvature and has been defined as:

$$\mathcal{T}(x) = K_{\mathcal{T}}(x+1) \tag{3.9}$$

where $K_{\mathcal{T}}$ is a proportionality constant that determines the maximum value of the density function.

Next, a discrete, unnormalized probability distribution function $F_{vhr'}(c')$ is obtained as:

$$F_{vhr'}(c') = \sum_{i=0}^{c'} f_{vhr'}(i) - f_{vhr'}(0) \tag{3.10}$$

If the image space of $F_{vhr'}(c')$ is sampled at ζ uniformly distributed points, the application of the inverse distribution function $F^{-1}{}_{vhr'}(y)$ to those points leads to a set of ζ points that are adaptively distributed according to $f_{vhr'}(c')$. This principle is illustrated in Figure 3.3. In our case, since the probability distribution corresponds to the curvature estimation, the density of selected pixels will be correlated to the image curvature and, hence, to shape variations.

In order to obtain $F^{-1}_{vhr'}(y)$ given the set of ζ pixels y uniformly distributed between zero and the maximum $F_{vhr'}(c')$, a table keeping the values $F_{vhr'}(c')$ for all the $c' \in [0, C/H]$ is computed. Then, a single iteration traverses this table, extracting those positions c' such that $F_{vhr'}(c') = y$. A vector of horizontal sampled pixels $\mathbf{HS}_{vhr'}(j)$, $j \in [0, \zeta)$, keeping the different c's, is obtained in this way.

This process is repeated for every row r' of the image tile $\mathbf{I}_{vh}$, producing an $(R/V+1)$ x ζ array:

$$\mathbf{HS}_{vh}(r', j) = \mathbf{HS}_{vhr'}(j), \quad r' \in [0, R/V], \; j \in [0, \zeta) \tag{3.11}$$

For each value j, if we iterate over r', we obtain a collection of pixels $(r', \mathbf{HS}_{vh}(r', j))$ that determine a "vertical" curve in the original image. Going over all the different j values, we obtain a collection of vertical curves that tend to adapt to the shape of the underlying objects contained in the image, coming closer in areas of fast shape variation. An example that illustrates the previous process is shown in Figure 3.4. Figure 3.4(*right*) shows the set of vertical curves obtained by applying a tessellation with 30 tiles ($H = 5$ and $V = 6$), with 8

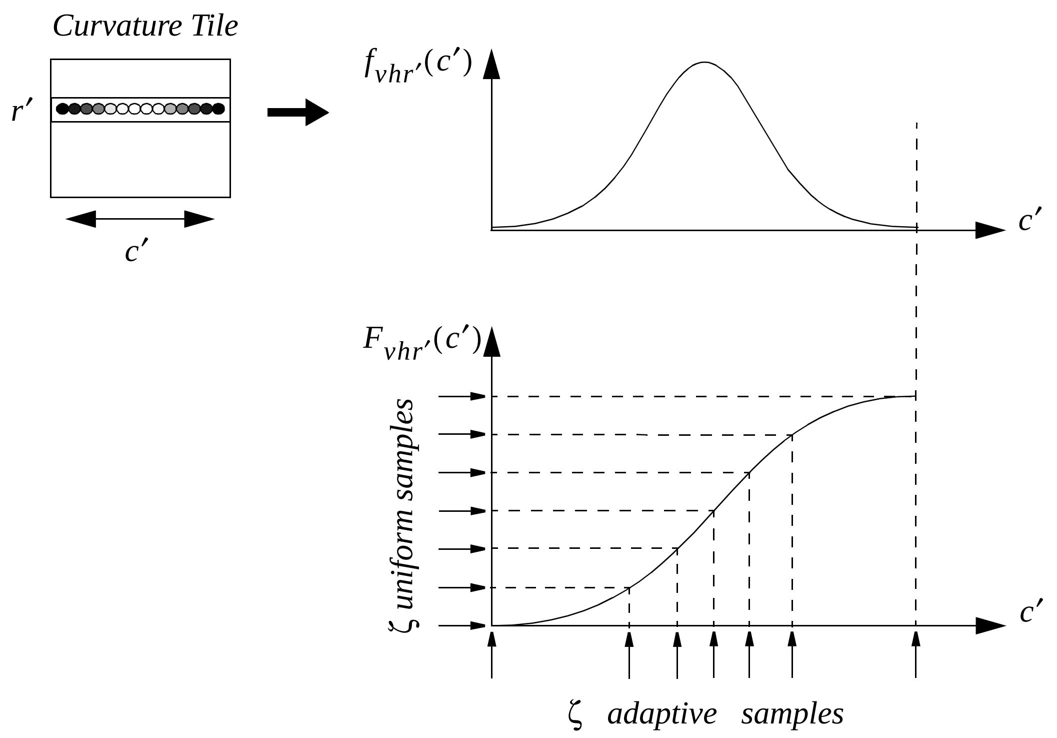

Figure 3.3. (*top left*) Curvatures associated with each row r' of an image tile. (*top right*) Unnormalized probability density function $f_{vhr'}(c')$ that represents the curvature associated with every pixel c' of row r'. (*bottom right*) Uniform sampling of the image space of the corresponding unnormalized probability distribution function $F_{vhr'}(c')$ gives a set of points whose density varies according to $f_{vhr'}(c')$.

columns per tile ($\zeta = 8$), to both the original and curvature images, Figure 3.4(*left*) and Figure 3.4(*middle*) respectively.

Each vertical curve obtained above corresponds to one of the columns of the tile being processed. In order to obtain the rows of the tile, each of these curves is adaptively sampled at $\mathcal{R}$ positions ($\mathcal{R}$ is the input parameter that indicates the number of rows per tile). The process is similar to the previous one, which leads to horizontal samples from rows of pixels extracted from the tile. The difference now is that a curvature profile is obtained from the positions of the pixels that belong to one of the curves, instead of from the positions corresponding to a horizontal row of pixels. Again, each tile is processed separately.

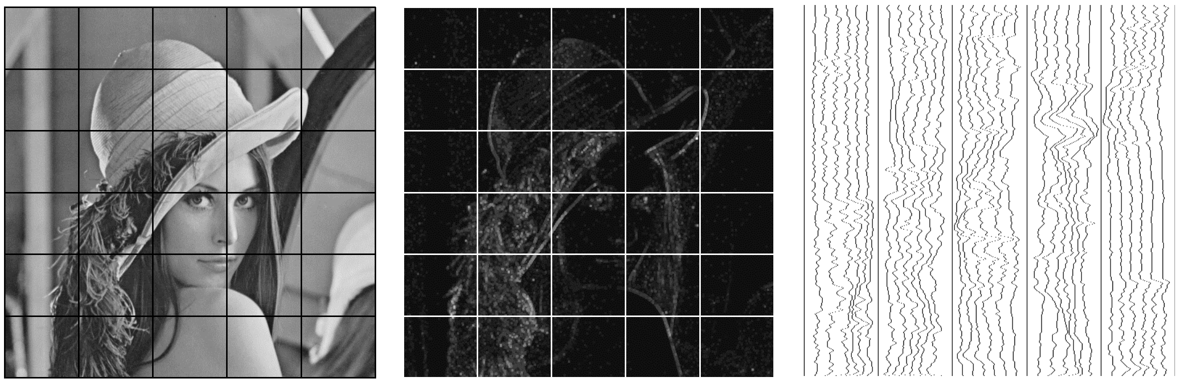

Figure 3.4. (*left*) Tiles superimposed on the original image. (*middle*) Tiles superimposed on the curvature image. (*right*) Vertical curves computed after adaptive horizontal sampling, whit $H = 5$, $V = 6$ and $\zeta = 8$.

Let $\mathbf{VC}_{vhj}(r') = \mathbf{HS}_{vh}(r', j)$, $r' \in [0, R/V]$, represent the vertical curve corresponding to a certain column j, $j \in [0, \zeta)$. The 2D positions of the pixels belonging to that curve are $(r', \mathbf{VC}_{vhj}(r'))$.

The unnormalized probability density function corresponding to the curvature profile associated with each curve is an adaptation of Equation (3.8): $f_{vhj}(r') = \mathcal{T}(\mathbf{K}_{vh}(r', \mathbf{VC}_{vhj}(r')))$. Similarly to Equation (3.10), an unnormalized probability distribution function $F_{vhj}(r')$ is computed. The image space of this distribution function is uniformly sampled at $\mathcal{R}$ positions y, and the inverse distribution function $F^{-1}_{vhj}(y)$ is applied to them in order to obtain a set of $\mathcal{R}$ pixels r' such that $F_{vhj}(r') = y$. A vector of vertical sampled pixels $\mathbf{VS}_{vhj}(i)$, $i \in [0, \mathcal{R})$, keeping the different r's, is obtained in this way. In the end, we obtain an $\mathcal{R}$ x ζ array of horizontal and vertical sampled pixels for every tile, whose elements are:

$$\mathbf{HVS}_{vh}(i, j) = (\mathbf{VS}_{vhj}(i), \mathbf{HS}_{vh}(\mathbf{VS}_{vhj}(i), j)) \tag{3.12}$$

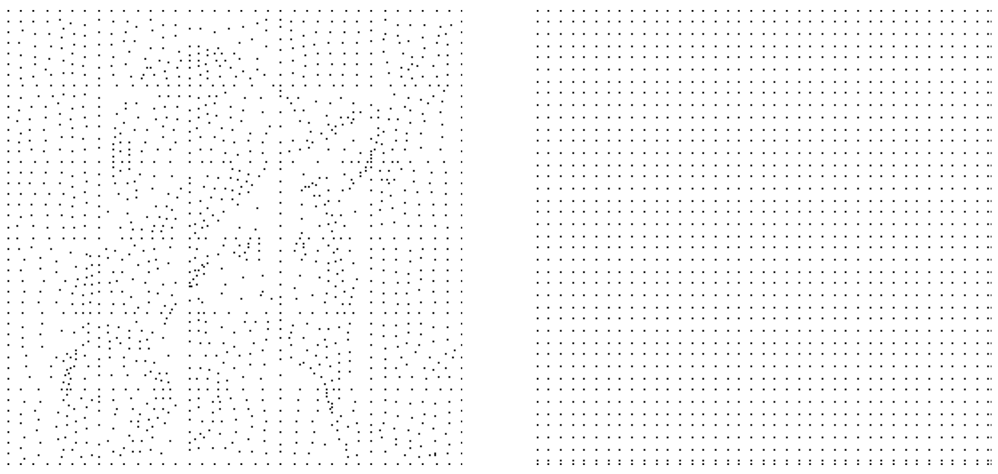

Figure 3.5. (*left*) Horizontal and vertical sampled pixels with $H = 5$, $V = 6$ and $\mathcal{R} = \zeta = 8$, by applying the adaptive sampling process: 1,548 pixels. (*right*) Similar number of pixels by applying uniform sampling: 1,521 pixels.

In summary, given an image tile $\mathbf{I}_{vh}$ and its associated curvature image $\mathbf{K}_{vh}$, $\mathbf{HVS}_{vh}(i, j)$ contains the 2D coordinates (r', c') of the pixel selected for each row i and column j of the given tile.

Since the boundaries of adjacent tiles are overlapped along one line of pixels and the pixels sampled in that line will coincide, the sampled pixels will be arranged as an array of $(\mathcal{R} - 1)V + 1$ rows and $(\zeta - 1)H + 1$ columns, with $(\mathcal{R}, \zeta)$ being the number of rows and columns per tile, and (V, H) being the number of vertical and horizontal partitions of the original digital image into tiles. Therefore, the elements of the 2D array **HVS**, which keeps, for every position, the (r, c) coordinates of the sampled pixels, are computed as:

$$\mathbf{HVS}(v(\mathcal{R} - 1) + i, h(\zeta - 1) + j) = \mathbf{HVS}_{vh}(i, j) + (ulr_{vh}, ulc_{vh}) \tag{3.13}$$

Figure 3.5(*left*) shows the set of vertical and horizontal sampled pixels obtained from all the tiles in which the original test image has been partitioned, considering $H = 5$, $V = 6$ and $\mathcal{R} = \zeta = 8$. A similar number of uniformly sampled pixels is shown in Figure 3.5(*right*). Notice that the sampled pixels in the adaptive distribution tend to concentrate in areas of

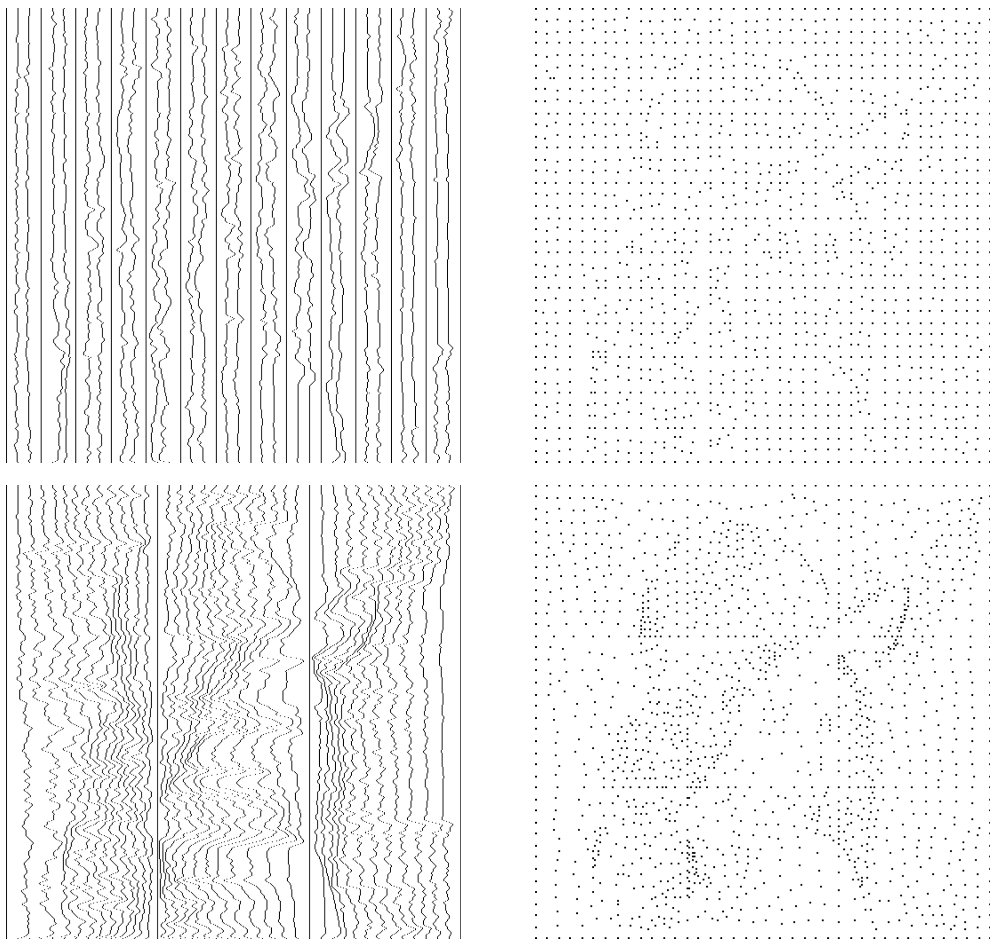

Figure 3.6. (*left column*) Vertical curves sampled by setting: (*top*) $H = V = 13$ and $\mathcal{R} = \zeta = 4$, and (*bottom*) $H = V = 3$ and $\mathcal{R} = \zeta = 14$. (*right column*) Sampled pixels from the previous vertical curves: 1,600 pixels in both examples.

high curvature, highlighting the shape of the objects contained in the image, whereas the chosen pixels are regularly sampled in the uniform distribution.

As mentioned above, the number of tiles (H and V) and the number of pixels per tile ($\mathcal{R}$ and ζ) are defined by the user. When a high number of tiles are considered, it leads to bad distributions of vertical curves, which produces uniformly-sampled pixels. When few tiles are considered, it leads to pixels concentrating in high curvature regions. Consequently, an intermediate number of tiles must be experimentally set. Figure 3.6 shows examples of these two cases.

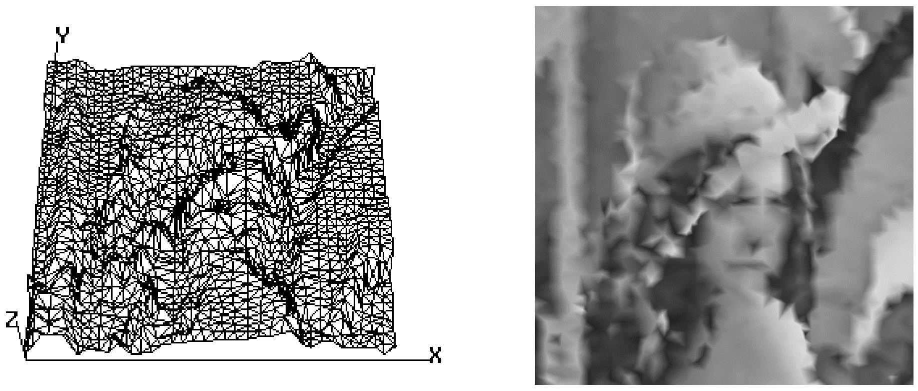

Figure 3.7. (*left*) Initial triangular mesh generated from the set of adaptively sampled pixels in Figure 3.5(*left*). (*right*) Approximating image obtained from the previous triangular mesh.

$2^1/_2$D Triangulation of Sampled Pixels

After the previous adaptive sampling, a set of $((R-1)V+1)$ x $((\zeta-1)H+1)$ pixels is selected from the original image. The objective now is to apply a $2^1/_2$D triangulation to the sampled pixels in order to obtain a triangular mesh that approximates the given image. This is done as follows.

As described above, each sampled pixel has an associated position (r, c) in the original image. The value stored in that location represents a property of the pixel, such as a gray level, range or terrain elevation. Thus, a 3D point (x, y, z) is defined from each sampled pixel by considering that both the *x* and *y* coordinates correspond to the pixel's column, c, and row, r, respectively, and the *z* coordinate corresponds to the pixel's property. A $2^1/_2$D triangulation is applied to those 3D points by projecting them onto the *xy* image reference plane. Then, a 2D Delaunay algorithm (Shewchuck, 1996a) is applied to their (x, y) coordinates. Finally, the $2^1/_2$D triangular mesh is obtained by linking the 3D points according to the resulting 2D triangulation. The result of this process is illustrated in Figure 3.7(*left*), which shows the initial adaptive triangular mesh that approximates the given image with 1,548 points.

3.3.1.2 Generation of the Approximating Image

Once the initial adaptive triangular mesh has been computed, it is necessary to obtain its corresponding approximating image in order to determine the accuracy with which that mesh approximates the original image. This is done by uniformly sampling the $2^1/_2$D triangular mesh at as many positions as pixels the original image has (*mesh backprojection*). A detailed description of two methods to obtain approximating images from $2^1/_2$D triangular meshes is presented in Section 3.4. The approximating image (with 262,144 pixels) corresponding to the initial adaptive triangular mesh shown in Figure 3.7(*left*) is presented in Figure 3.7(*right*).

3.3.1.3 Determination and Resampling of Error Regions

If the RMS error of the approximating image, computed according to Equation (3.2), is below the specified tolerance, the algorithm concludes and the adaptive triangular mesh obtained above is already the solution. Otherwise, the algorithm proceeds by identifying the regions of the approximating image whose error with respect to the original image is above the tolerance. This process is described below.

First, an *error image* is obtained by subtracting each pixel of the original image from the corresponding pixel of the approximating image, and by taking the absolute value, Equation (3.1). The result is shown in Figure 3.8(*left*). This error image is converted to a *binary image* by thresholding it with the given tolerance. Figure 3.8(*middle*) shows the binary image obtained for the current example.

All black regions in the binary image represent *error regions* that must be resampled. The binary image is labelled in order to determine the different error regions contained in it. Then, an *enclosing rectangle* for each separate error region is computed, such as it is illustrated in Figure 3.8(*right*). Very small enclosing rectangles (e.g., less than 3x3 pixels) are discarded. If the horizontal size of an enclosing rectangle is larger than the horizontal size of the initial tiles, that rectangle is horizontally subdivided into the minimum number of rectangles such that their corresponding horizontal sizes are less than or equal to the tile's horizontal size. The same procedure is applied to the vertical direction. Hence, the size of an

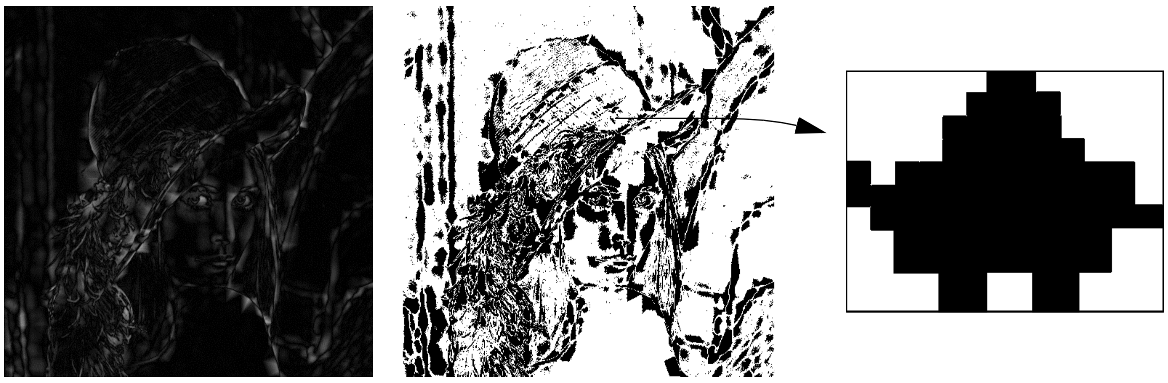

Figure 3.8. (*left*) Error image obtained by applying Equation (3.1) to the original gray-level image, Figure 3.2(*left*), and the approximating image, Figure 3.7(*right*). (*middle*) Binary image obtained from the error image by applying a threshold equal to the given tolerance (12). (*right*) Enclosing rectangle corresponding to a certain error region.

enclosing rectangle is guaranteed to be lower or equal to the size of a tile. Figure 3.9(*left*) illustrates the enclosing rectangles corresponding to all the separate error regions superimposed on the binary image.

Each enclosing rectangle obtained above delimits a region that must be resampled. This resampling process consists of applying the same adaptive sampling technique described in Section 3.3.1.1, although with each enclosing rectangle being now considered to be a single tile. The number of pixels adaptively sampled over each tile is not predefined as it was before, but computed as *k* times the number of pixels previously sampled over it, with *k* being a real larger than one. For instance if $k = 1.5$, the result will be a fifty per cent increase of pixel density at each enclosing rectangle. The old pixels sampled in each enclosing rectangle are substituted for the new ones.

The new set of sampled pixels is the result of merging the pixels resampled in the enclosing rectangles and the old sampled pixels that did not belong to any enclosing rectangle. Figure 3.9(*right*) shows the new adaptive pixel distribution obtained when the previous process is applied to the found enclosing rectangles.

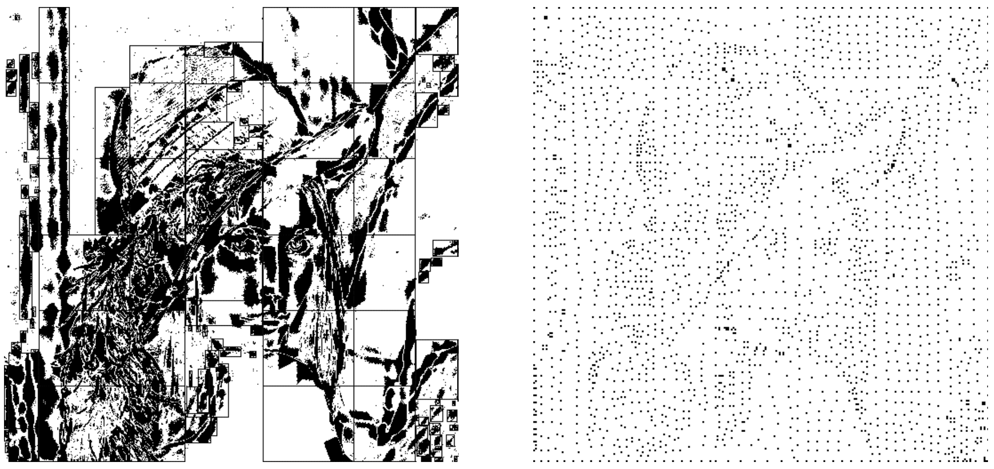

Figure 3.9. (*left*) Enclosing rectangles for detected error regions in the initial approximating image. (*right*) New adaptive distribution of pixels obtained when the adaptive sampling process is applied to the enclosing rectangles, and the new pixels are merged to the old pixels that did not belong to any enclosing rectangle (2,645 pixels).

3.3.1.4 Delaunay Retriangulation

The new resampled pixels and the old ones that did not belong to any enclosing rectangle are retriangulated by applying the 2D Delaunay algorithm (Shewchuck, 1996a). At this point, the image approximation algorithm proceeds to refine the obtained triangular mesh by iterating from the approximating image generation step (Section 3.3.1.2) until the RMS error of the approximating image is below the given tolerance. Figure 3.10(*top middle*) shows the adaptive triangular mesh obtained from the final set of sampled pixels displayed in Figure 3.10(*top left*). This mesh was generated after 4 iterations, given a tolerance equal to 12. The total CPU time was 10.3 sec. on a SGI Indigo II with a 175MHz R10000 processor. The approximating image obtained from the previous mesh is shown in Figure 3.10(*top right*). The results when a uniform sampling is applied to the original image, considering a similar number of pixels, are shown in Figure 3.10(*bottom*). Notice that the uniform sampling technique misses details (contours) which are captured by the proposed adaptive sampling technique.

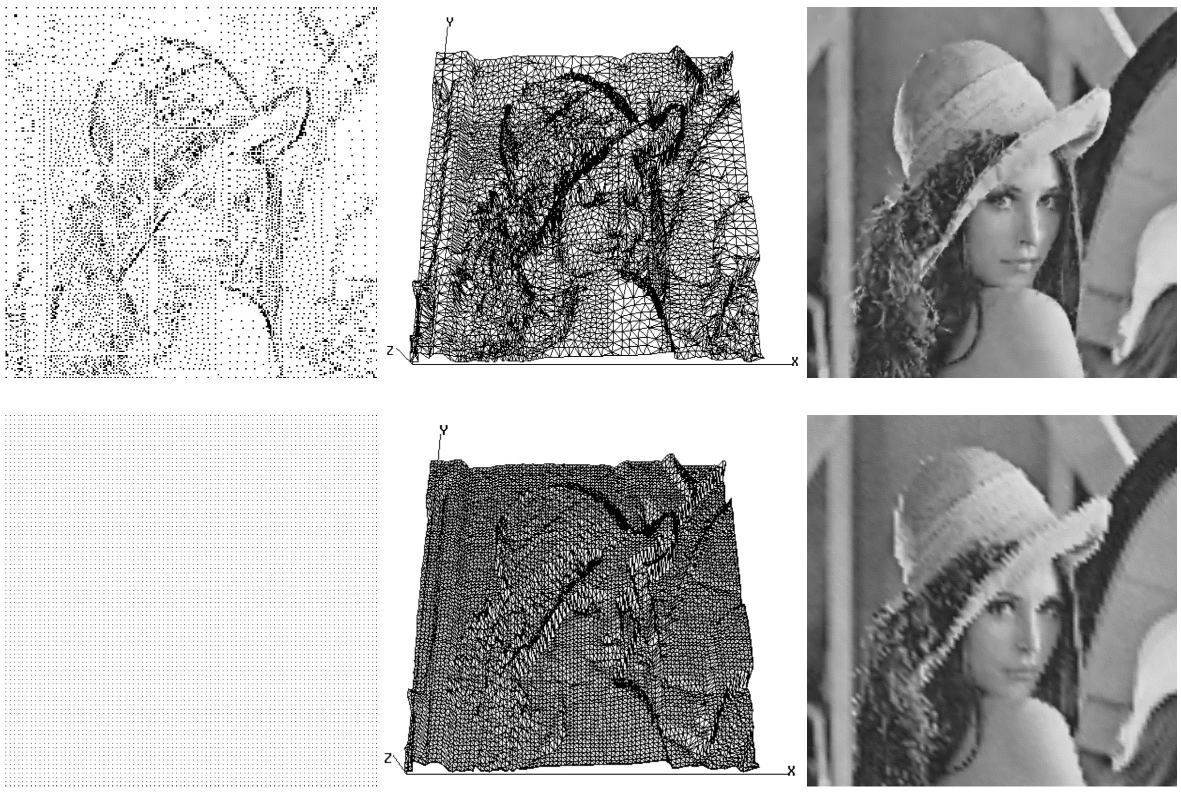

Figure 3.10. (*top left*) Final set of pixels obtained with the proposed adaptive sampling technique, 7,772 pixels. (*top middle*) Adaptive triangular mesh generated from the previous points. (*top right*) Approximating image obtained from the previous triangular mesh, RMS = 11.2. (*bottom left*) Similar number of pixels obtained by applying the uniform sampling technique, 7,921 pixels. (*bottom middle*) Triangular mesh generated from the previous points. (*bottom right*) Approximating image obtained from the previous triangular mesh, RMS = 14.2.

3.3.1.5 Experimental Results

The proposed algorithm has been tested upon various real images. All CPU time were measured on a SGI Indigo II with a 175MHz R10000 processor.

Figure 3.11 shows three of the test images. The left image, *Cotopaxi volcano*, was initially partitioned into 30 tiles (H = 5 and V = 6), and 64 pixels ($\mathcal{R} = \zeta = 8$) were adaptively sampled over each tile. The given RMS error was set to 12. Taking into account the previous initial parameters, 9,546 pixels were finally sampled with the proposed technique in 5 iterations, Figure 3.12(*left column, top*). The adaptive triangular mesh generated from the

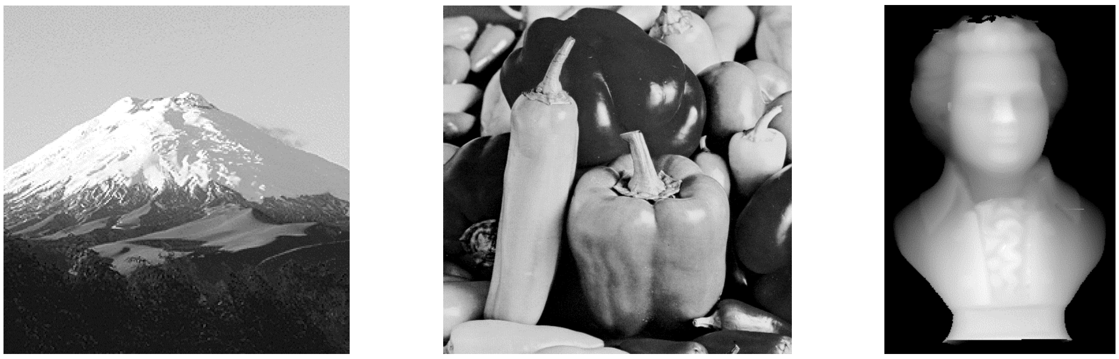

Figure 3.11. Some test images. (*left*) Gray-level image (*Cotopaxi volcano*) with 262,144 pixels (512x512). (*middle*) Gray-level image (*Peppers*) with 262,144 pixels (512x512). (*right*) Range image (*Bust*) with 104,937 pixels (399x263).

previous points is shown in Figure 3.12(*middle column*, *top*). The total CPU time was 11.81 sec. The approximating image obtained from the mesh shown in Figure 3.12(*middle column, top*) is displayed in Figure 3.12(*right column, top*). The RMS error of the previous approximating image is 11.48. The middle image (*Peppers*), Figure 3.11(*middle*), was partitioned into $H = 5$ and $V = 6$ tiles, and $\mathcal{R} = \zeta = 8$ pixels were adaptively sampled over each tile. The maximum RMS error was set to 14. 7,644 pixels were finally sampled in 4 iterations, Figure 3.12(*left column, middle*). The total CPU time was 7.86 sec. The approximating image obtained from the mesh shown in Figure 3.12(*middle column, middle*) is displayed in Figure 3.12(*right column, middle*). The RMS error of the approximating image is 13.45. Finally, a range image (*Bust*), Figure 3.11(*right*), was partitioned into $H = 6$ and $V = 4$ tiles, and $\mathcal{R} = \zeta = 8$ pixels were adaptively sampled over each tile. The maximum RMS error was set to 4. A total of 4,187 pixels were finally sampled after 7 iterations, Figure 3.12(*left column, bottom*). The total CPU time was 4.98 sec. The approximating image obtained from the adaptive triangular mesh shown in Figure 3.12(*middle column, bottom*) is presented in Figure 3.12(*right column, bottom*). Its RMS error is 3.89.

The triangular meshes obtained with the proposed technique do not have to store the mesh topology, since they can be directly recovered by triangulating the sampled pixels. In this way, the generated triangular meshes are kept in a compact representation by only sav-

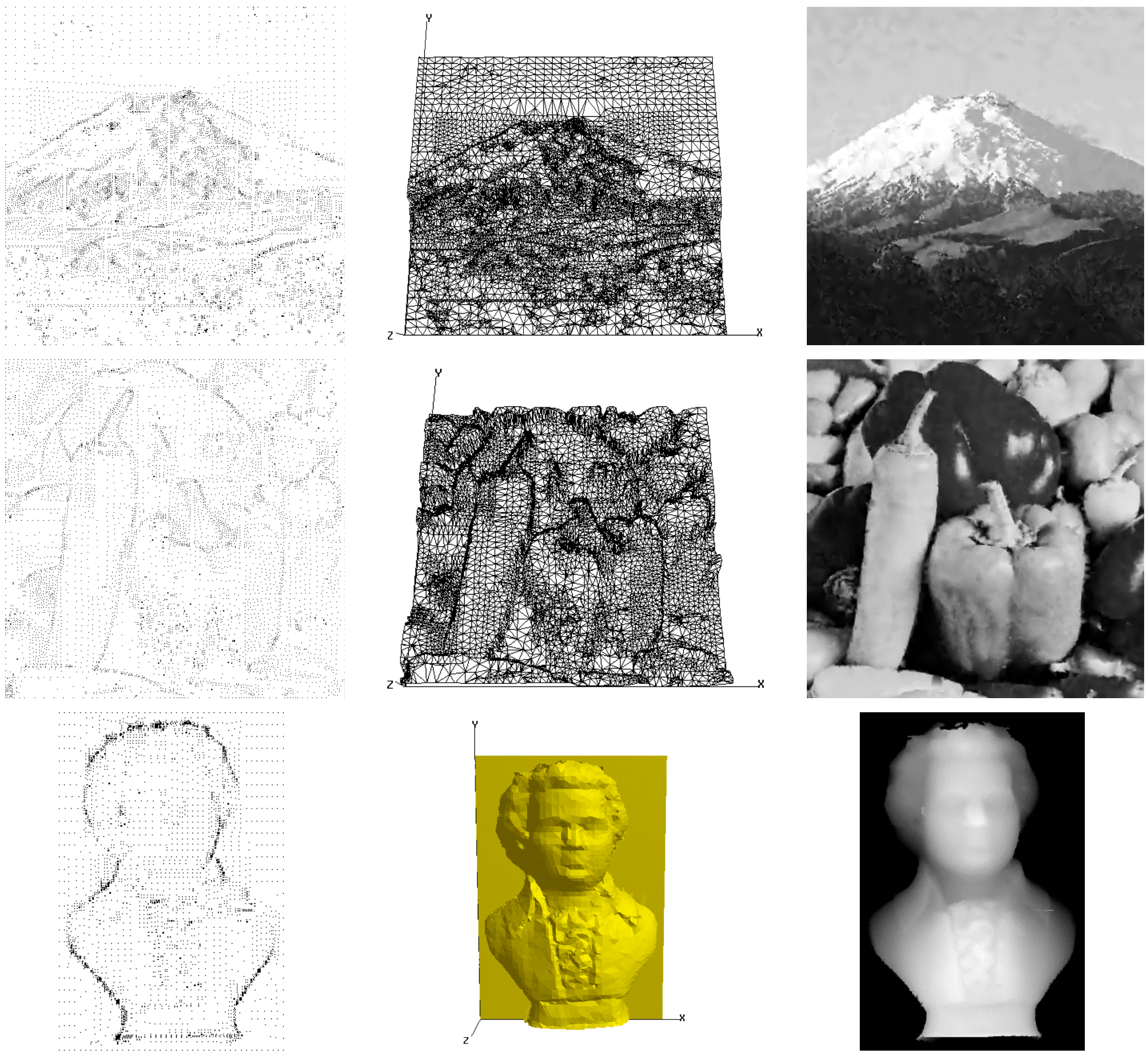

Figure 3.12. (*left column*) Final set of sampled points: (*top*) 9,546, (*middle*) 7,644 and (*bottom*) 4,187 points. (*middle column*) Adaptive triangular meshes generated from the previous points. (*right column*) Approximating images with RMS errors: (*top*) 11.5, (*middle*) 13.4 and (*bottom*) 3.9.

ing the coordinates of each sampled vertex: $(c, r, \boldsymbol{I}(r, c))$. For example, for 512x512x256 images, 4 bytes per vertex are necessary. After downloading the vertices into memory, they are triangulated by applying a 2D Delaunay algorithm. Considering the previous compact representation, the compression ratios corresponding to the meshes shown in Figure 3.12(*middle column*) are: (*top*) 6.9:1, (*middle*) 8.6:1 and (*bottom*) 6.3:1.

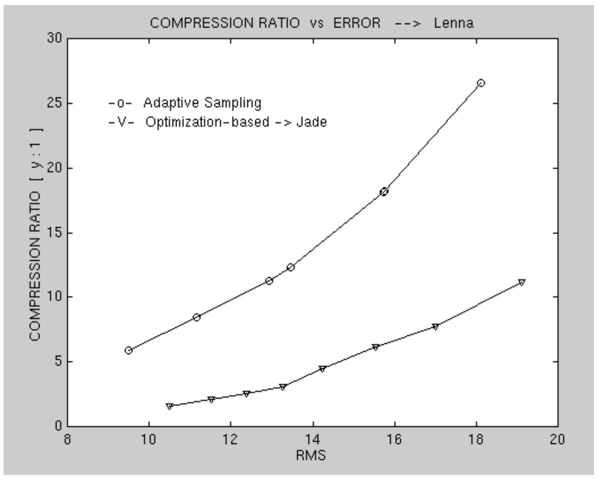

Figure 3.13. Compression ratio vs. RMS error for a gray-level image (Lenna) with 512x512 pixels, by considering a fine-to-coarse optimization technique (JADE) and the proposed adaptive sampling technique.

The proposed algorithm has been compared with a mesh refinement algorithm based on iterative optimization (Ciampalini et at., 1997). An implementation of the latter, called JADE (*Just Another Decimator*), is publicly available. JADE starts with a dense triangular mesh, which in our case contains all the pixels of the given image, and obtains a decimated mesh with a predefined number of vertices that minimizes the approximation error. The triangular meshes generated with JADE (and with optimization-based algorithms in general) can not be recovered from the vertices themselves. Hence, each triangle must also be saved as three integers containing indices to its vertices. In the optimistic case that the number of sampled vertices is lower than 65,536, short integers (2 bytes) can be utilized.

Figure 3.13 displays compression ratios versus RMS errors for Lenna, considering both JADE and the proposed technique. The proposed technique was run for various tolerances. JADE was run to decimate the original images until it produced the same number of vertices as the proposed technique. In all cases, the RMS error with the proposed technique was slightly lower than with JADE. Moreover, the proposed technique was up to two orders of magnitude faster than JADE. For instance, the approximation of Lenna for tolerance 12 took 10.3 sec. with the proposed technique and 2,626 sec. with JADE. Figure 3.14 shows

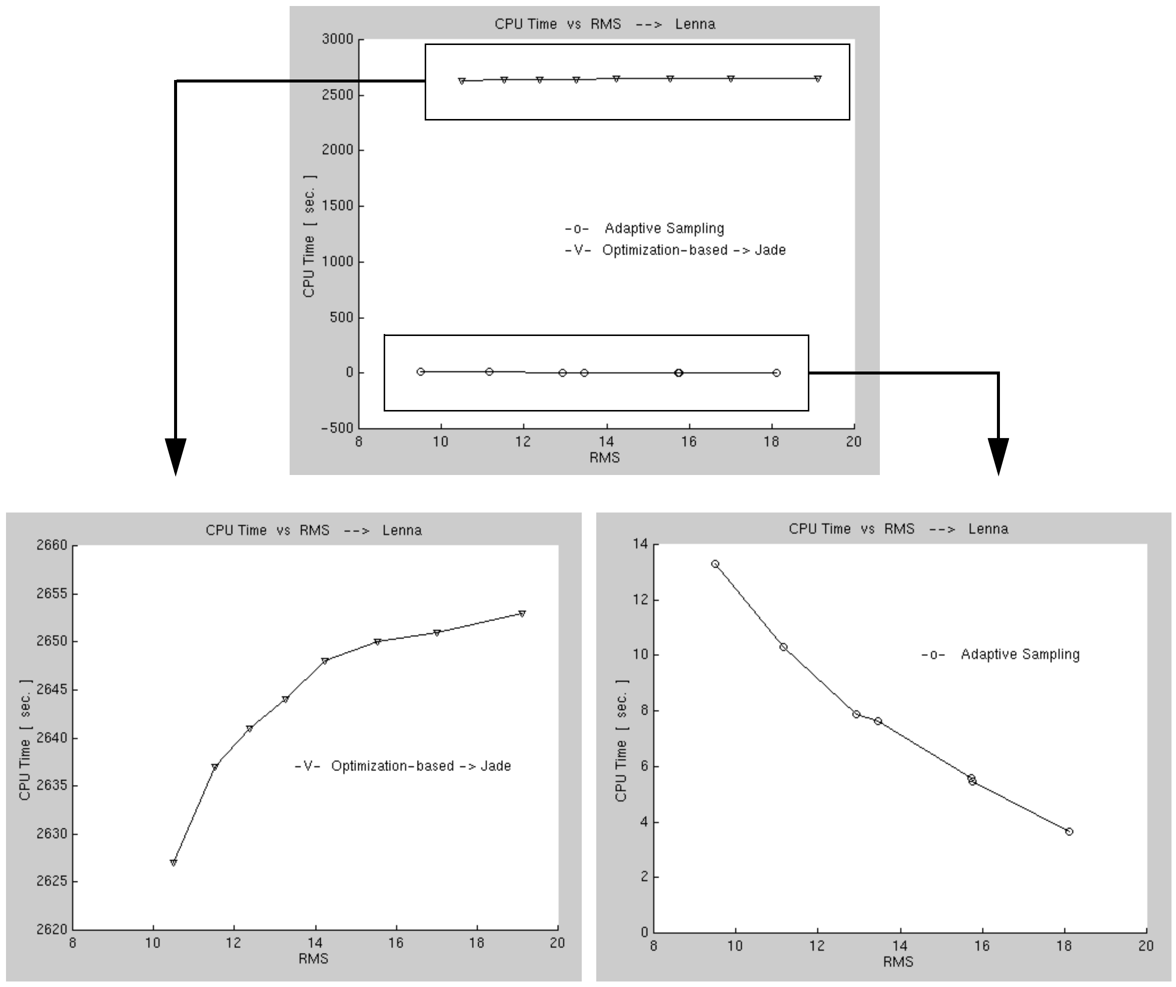

Figure 3.14. (*top*) CPU time vs. RMS error for Figure 3.13(*left*), by considering a fine-to-coarse optimization technique (JADE) and the proposed adaptive sampling technique. (*bottom left*) CPU time vs. RMS error with JADE. (*bottom right*) CPU time vs. RMS error with the proposed technique.

the CPU time versus RMS error considering both approaches. Notice that, when an adaptive triangular mesh with a low RMS error is required, the decimation technique behaves better than the proposed technique. This occurs because the final triangular mesh is similar to the original mesh, the latter being the starting point of the decimation algorithm. On the other hand, when a coarse adaptive triangular mesh is required, the proposed technique is more suitable than the decimation approaches. In this case, the solution is closer to the coarse mesh utilized as the starting point of the proposed technique.