Chapter 4

Geometric Processing of Adaptive Triangular Meshes: Analysis Operations

This chapter presents a set of geometric processing operations applicable to $2^1/_2$D adaptive triangular meshes. As it was indicated in chapter 3, these meshes must be projectable onto the $z = 0$ reference plane, and they may approximate a digital image or any other kind of surface, such as terrain. Those meshes are simply referred to as *triangular meshes* throughout this chapter.

These operations have been classified as analysis operations since they involve the extraction of information from adaptive triangular meshes. The extracted information can then be used to perform a description, interpretation or understanding of the adaptive triangular meshes in some way. Section 4.1 introduces the proposed analysis operations. Section 4.2 describes a set of preprocessing operations: geometric transformations, thresholding, quantization and algebraic operations, region-of-interest selection and tools for the generation of synthetic triangular meshes. Section 4.3 describes an edge detection technique. Section 4.4 presents a segmentation technique. Section 4.5 describes a set of operations for

feature extraction from binary triangular meshes. Section 4.6 presents a histogram generation algorithm from adaptive triangular meshes. Finally, the contents of this chapter are summarized in Section 4.7.

4.1 Introduction

The interest of applying processing operations to digital images is twofold. On the one hand, image processing operations allow to improve the quality of the original images. On the other hand, those processing operations may simplify further computer vision tasks.

In chapter 3, it was shown how to obtain $2^1/_2$D adaptive triangular meshes starting with digital images. The advantage of utilizing triangular meshes is that they allow to represent the information contained in digital images in a compact form, by removing redundant information.

This chapter presents a set of techniques to perform typical image analysis operations upon $2^1/_2$D triangular meshes with two main purposes. First, by taking into account that the processed triangular meshes can represent any type of information, such as digital images or terrain surfaces, the typical operations applied in image processing upon gray level images can also be applied to triangular meshes in general. Additionally, by taking advantage that the processed triangular meshes are compact representations of digital images, the processing operations directly applied to those meshes can be more efficient than if they were applied to the original digital images.

The classification of analysis operations utilized in this chapter is based on the classification made by Jain (1989) and Umbaugh (1998). In the next section, a set of techniques to perform image preprocessing operations upon $2^1/_2$D triangular meshes is described.

4.2 Preprocessing Operations upon $2^1/_2$D Triangular Meshes

This section describes several image preprocessing operations applied to $2^1/_2$D triangular meshes. The described techniques allow the application of the following operations: geo-

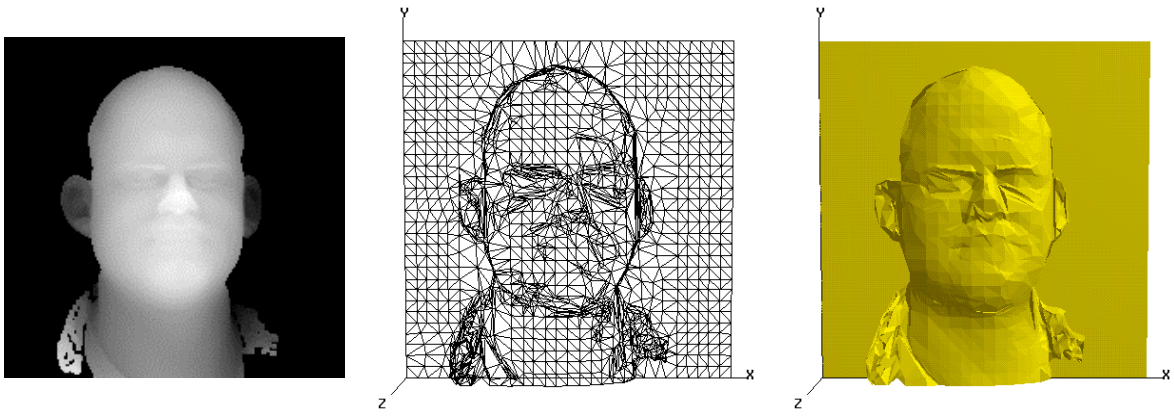

Figure 4.1. (*left*) Original range image with 40,000 pixels (200x200). (*middle*) Adaptive triangular mesh obtained with the algorithm described in Section 3.3.2 (1,634 sampled pixels). The proposed geometric transformations are applied upon this mesh. (*right*) Rendered adaptive triangular mesh.

metric transformations, thresholding, quantization, algebraic operations, selection of regions-of-interest and generation of synthetic triangular meshes.

The next section presents the following geometric transformations: rotation, translation, shrinking, stretching, scaling and deformation.

4.2.1 Geometric Transformations

The adaptive triangular meshes obtained with any of the two techniques proposed in Chapter 3 are representations of digital images at a higher level of abstraction. This may allow the application of many image processing operations more efficiently than if they were applied upon the individual pixels of the original images. For example, rotation, translation, scaling, shrinking, stretching and deformation operations are trivially implemented by applying geometric transformations to the 3D coordinates of the points that constitute the adaptive triangular meshes. Since those adaptive meshes contain a fraction of the original amount of pixels, these operations perform faster than their pixel-to-pixel counterparts. Figure 4.1(*middle*) shows an adaptive triangular mesh that will be used to illustrate the application of the proposed geometric transformations. This mesh was obtained by applying the algorithm described in Section 3.3.2.

4.2.1.1 Rotation and Translation of Triangular Meshes

The rotation of a triangular mesh can be implemented by rotating every point (x, y, z) of the mesh a previously defined angle. Hence, the new coordinates (x', y', z') of the points of the given triangular mesh are computed as follows:

$$\begin{aligned} x' &= x\cos\Phi + y\sin\Phi \\ y' &= -x\sin\Phi + y\cos\Phi \\ z' &= z \end{aligned} \tag{4.1}$$

where Φ is the specified rotation angle. Figure 4.2(*left*) shows the result obtained when a rotation of 45 degrees is applied to the triangular mesh shown in Figure 4.1(*middle*).

The translation of a triangular mesh consists of the translation of the (x, y, z) coordinates of its points:

$$\begin{aligned} x' &= x + T_x \\ y' &= y + T_y \\ z' &= z \end{aligned} \tag{4.2}$$

where (x', y', z') are the coordinates of the translated points and T_x and T_y are the specified translation factors. Figure 4.2(*middle*) shows the result obtained when a translation with $T_x = 70$ and $T_y = 100$ is applied to the points of an adaptive triangular mesh.

Notice that the rotation and translation operations can also be combined as follows:

$$\begin{aligned} x' &= (x + T_x)\cos\Phi + (y + T_y)\sin\Phi \\ y' &= -(x + T_x)\sin\Phi + (y + T_y)\cos\Phi \\ z' &= z \end{aligned} \tag{4.3}$$

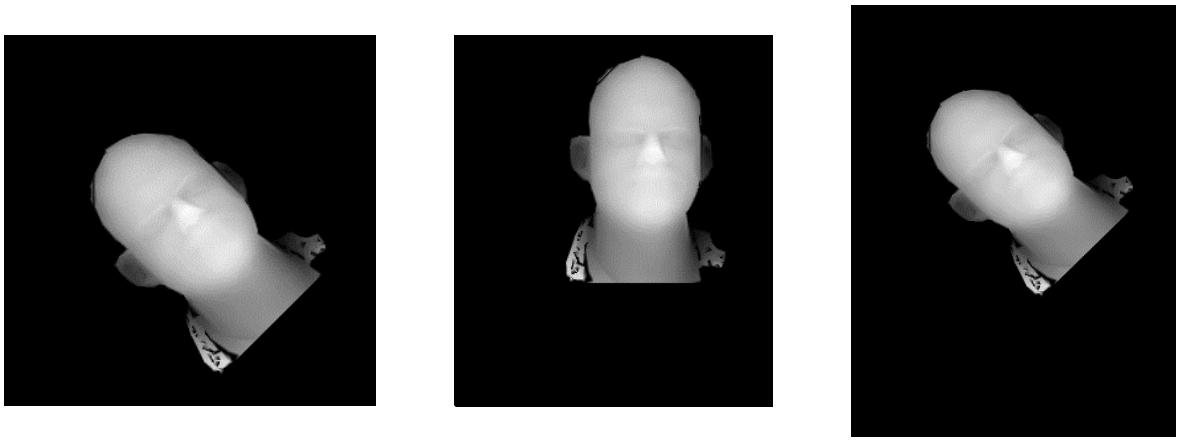

Figure 4.2. Approximating images obtained after applying geometric transformations to the original triangular mesh. (*left*) Rotation. (*middle*) Translation. (*right*) Combined operations: rotation and translation.

Figure 4.2(*right*) displays the final image after applying the same rotation and translation operations, Figure 4.2(*left*) and Figure 4.2(*middle*), but now combined into a single expression, by using Equation (4.3).

When an approximating image is generated from the triangular mesh obtained after a translation or rotation, some problems may arise. For instance, if a translation is applied, the dimensions of the mesh bounding box remain constant. In this way, the obtained approximating image by applying the z-buffering process (Section 3.4.2) would no visualize the effect of the translation applied to the mesh. To avoid this problem, a *visualization bounding box* has been defined. This new bounding box contains both the mesh bounding box and the effects produced after processing that mesh. The minimum coordinates (x_{min}, y_{min}) of the visualization bounding box are set to $(0, 0)$, while the maximum coordinates are defined according to the boundaries of the mesh. If any of the maximum coordinates are negative, they are translated back to the positive quadrant. A similar procedure is applied when the triangular mesh has been rotated. Figure 4.3 illustrates those problems. The pixels contained in the visualization bounding box that are located outside the limits of the resulting mesh are set to zero when they are visualized using the z-buffering technique presented in Section 3.4.2.

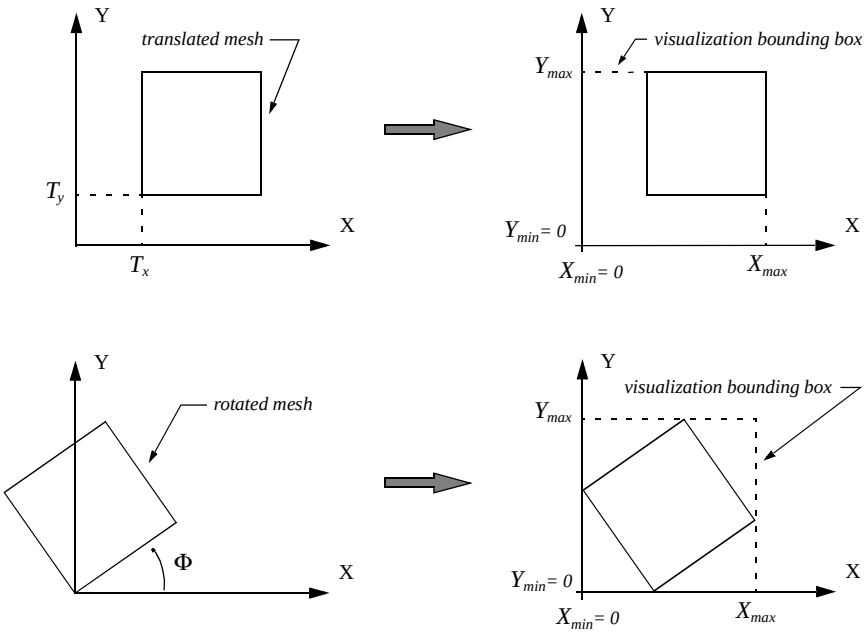

Figure 4.3. Definition of the visualization bounding box: (*top row*) after applying mesh translation and (*bottom row*) after applying mesh rotation.

4.2.1.2 Scaling, Shrinking and Stretching of Triangular Meshes

The scaling operation applied to a given triangular mesh is defined as:

$$\begin{aligned} x' &= x\,S_x \\ y' &= y\,S_y \\ z' &= z \end{aligned} \tag{4.4}$$

where S_x and S_y are the scale factors computed as: $S_x = (S(X_{max}+1)-1)/X_{max}$ and $S_y = (S(Y_{max}+1)-1)/Y_{max}$; where S is the scale factor specified by the user. Figure 4.4(*left*) displays the result obtained when a scale factor $S = 2$ is applied to the triangular mesh shown in Figure 4.1(*middle*).

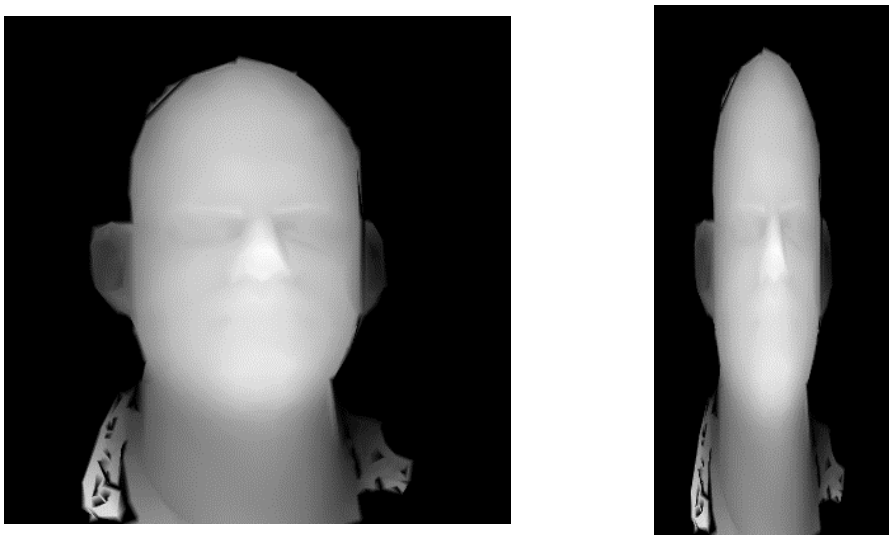

Figure 4.4. Approximating images obtained after applying: (*left*) *scaling* and (*right*) *stretching* and *shrinking* to the triangular mesh of Figure 4.1(*middle*).

Additionally, shrinking and stretching operations can also be applied to the given triangular mesh if a new number of rows and columns is desired for the modified image. In this case, the coordinates (x', y', z') of the mesh points are computed as follows:

$$\begin{aligned} x' &= xS'_x \\ y' &= yS'_y \\ z' &= z \end{aligned} \tag{4.5}$$

where S'_x and S'_y are the shrinking or stretching factors computed as: $S'_x = (C_{new} - 1)/X_{max}$ and $S'_y = (R_{new} - 1)/Y_{max}$. C_{new} and R_{new} are the new number of columns and rows desired for the final image (shrunk or stretched image) respectively, and (X_{max}, Y_{max}) are the maximum coordinate values of the input triangular mesh. Figure 4.4(*right*) shows the result when the triangular mesh is scaled up to an image of 400x180 pixels (vertical stretching and horizontal shrinking).

4.2.1.3 Deformation of Triangular Meshes

Besides the basic functions described above, any mathematical function can be applied to the points of the mesh in order to obtain other types of visual effects. For example, a possible deformation is the generation of an elliptical bounded mesh from a quadrilateral bounded mesh. This kind of geometric manipulation directly works upon the positions of the points, maintaining the topology of the original triangular mesh. The procedure applied to obtain the proposed deformation is the following.

First, the x and y coordinates of the points of the original mesh are scaled and translated (Section 4.2.1.1 and Section 4.2.1.2) such that they are normalized into the interval $[-1, 1]$. Then, the positions of the points of the elliptical bounded mesh are obtained from the points of the previous normalized mesh as:

$$\begin{aligned} x' &= k_x\, r \cos\theta \\ y' &= k_y\, r \sin\theta \\ z' &= z \end{aligned} \tag{4.6}$$

where $r = x$ when either $-\pi/4 \le \theta < \pi/4$ or $3\pi/4 \le \theta < 5\pi/4$, and $r = y$ when $\pi/4 \le \theta < 3\pi/4$ or $5\pi/4 \le \theta < 7\pi/4$; θ is the angle of each normalized point with respect to the horizontal axis of a coordinate system located at the center of the normalized mesh, k_x and k_y are scalar factors used to generate a circular deformation ($k_x = k_y$) or an elliptical deformation ($k_x \neq k_y$). Two examples that illustrate the approximating images obtained after applying a circular deformation and an elliptical deformation are shown in Figure 4.6(*bottom left*) and Figure 4.6(*bottom right*) respectively.

4.2.1.4 Experimental Results

The proposed geometric transformations have been tested upon various adaptive triangular meshes of different size. Those meshes are representations of digital images that have been obtained with the algorithm described in Section 3.3.2. The CPU times to perform these operations were measured and compared with the times to perform similar operations with

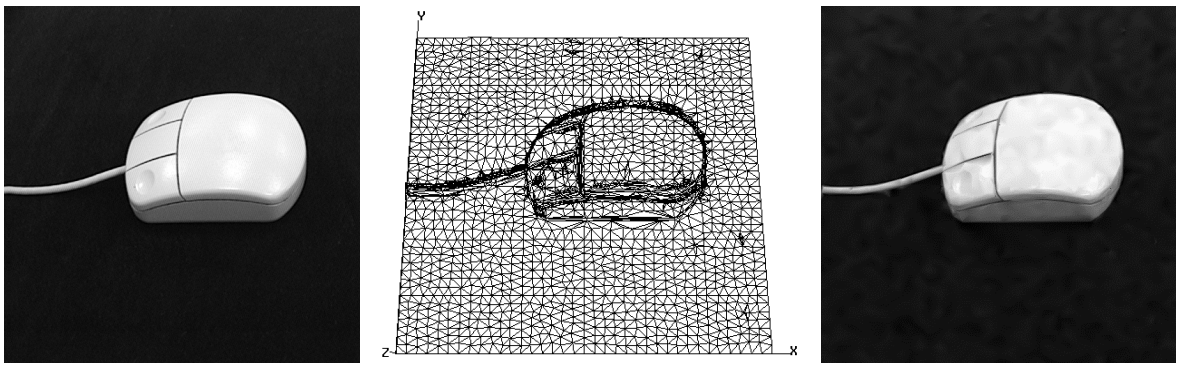

Figure 4.5. Triangular mesh utilized to perform the geometric transformation operations. (*left*) Original image with 262,144 pixels (512x512). (*middle*) Adaptive triangular mesh generated with the algorithm described in Section 3.3.2, which contains 2,461 sampled pixels. (*right*) Approximating image obtained from the previous mesh.

CVIPtools, a conventional image processing software which is publicly available (Umbaugh, 1998). These CPU times were measured on a SGI Indigo II with a 175MHz R10000 processor.

Figure 4.1(*middle*) and Figure 4.5(*middle*) show two examples of adaptive triangular meshes that contain 1,634 and 2,461 points respectively. Their corresponding RMS errors with respect to the original images, Figure 4.1(*left*) and Figure 4.5(*left*), are 9.92 and 7.26 respectively.

The CPU times to perform the rotation operation upon the images displayed in Figure 4.1(*left*) and Figure 4.5(*left*) with CVIPtools were 0.33 sec. and 2.26 sec., while the same operation in the geometric domain took 0.0043 sec. and 0.0056 sec. respectively. The translation operation took 0.05 sec. and 0.34 sec. with CVIPtools and 0.0003 sec. and 0.0004 sec. with the proposed technique. Finally, the scaling operation took 0.11 sec. and 0.34 sec. with CVIPtools, while it took 0.0004 sec. and 0.0006 sec. with the proposed technique in the geometric domain.

CVIPtools does not include any routines for producing deformations such as the circular and elliptical ones shown in Figure 4.6(*bottom*). Therefore, they should be implemented

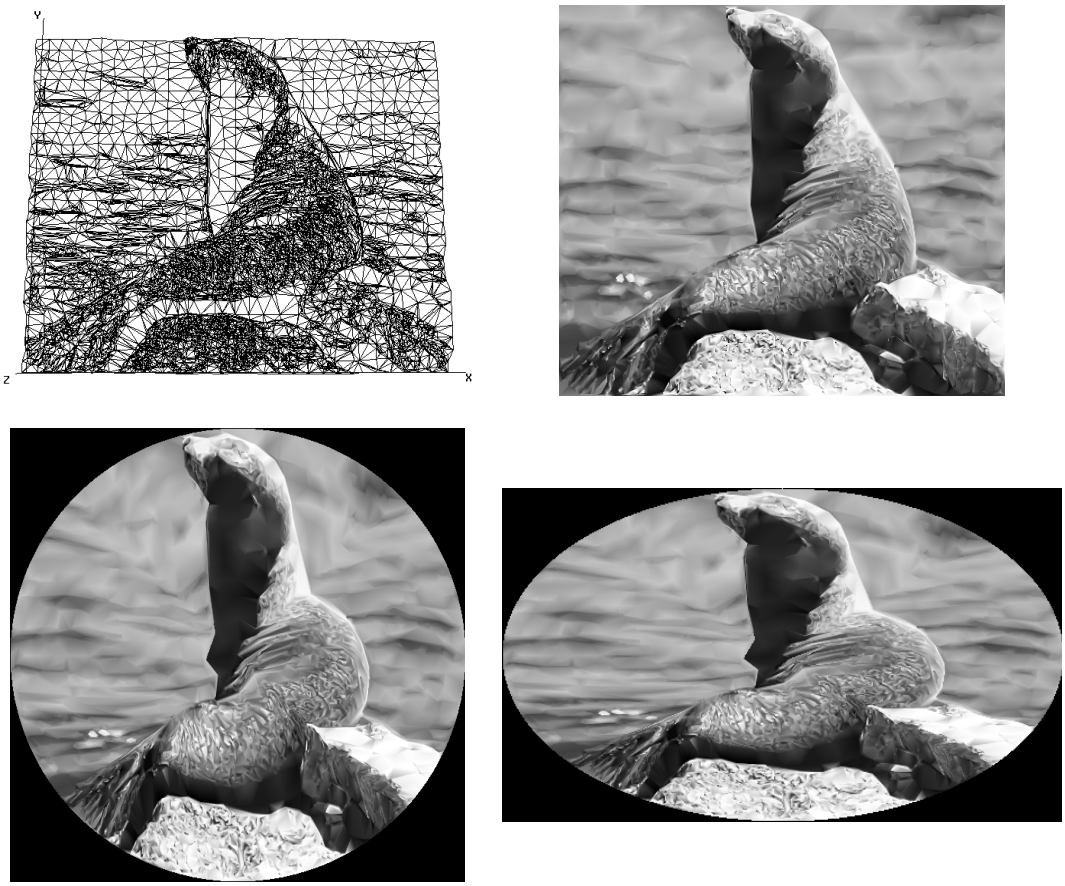

Figure 4.6. Deformation of triangular meshes. (*top left*) Given triangular mesh with 10,866 points. (*top right*) Approximating image generated from the previous triangular mesh (398x454 pixels). (*bottom*) Resulting images obtained after applying deformation operations upon the given triangular mesh: (*left*) circular and (*right*) elliptical deformation.

with a user program that would access the given image, pixel by pixel, with the subsequent time penalty. Similarly, all the image deformations typically found in Adobe's Photoshop-like image processing packages are easily implementable in the 3D geometric domain by trivial mesh deformations, requiring a fraction of the time utilized in the image domain.

Notice that the CPU times obtained with the proposed technique were much faster than the ones obtained with CVIPtools in all cases. The reason is that only a small percentage of points are processed in the geometric domain, while, in the image domain, all pixels must

be considered. Those times do not consider the mesh generation and image reconstruction stages. The reason is that these stages must only be applied once: to map the original image to the geometric domain and to map the resulting mesh back to image space. If several operations are performed (chained) in the geometric domain, the overhead of those two stages will become negligible.

4.2.2 Thresholding of Triangular Meshes

This section describes a technique for thresholding adaptive triangular meshes. The proposed technique generates triangular meshes that approximate the binary images that would have been obtained by thresholding the original images pixel by pixel. This operation is often used as a preprocessing step for the extraction of object features, such as area or perimeter, or for labeling the objects contained in the image.

The proposed geometric thresholding technique consists of two main stages. The first stage dissects the given adaptive triangular mesh with a plane parallel to the *xy* reference plane of the mesh, which intersects the *z* axis at a value that corresponds to the specified threshold. The result is a set of points that will be linked to produce *cross-sections.* The second stage of the algorithm triangulates the points that define the different cross-sections, using the segments between those points as constraints for the triangulation. Both stages are described below.

4.2.2.1 Single Dissection of a Triangular Mesh

This stage determines the regions of a triangular mesh which are above and below a given dissection plane. These regions will represent the areas in the binary image which are black and white respectively. The algorithm applied to obtain these regions is the following.

First, a plane, referred to as the *dissection plane*, parallel to the *xy* reference plane and intersecting the *z* axis at a value equal to a specified threshold is defined. This threshold will range between 0 and Z_{MAX}, where Z_{MAX} represents, for example, the maximum gray level in an 8-bit image. Figure 4.7(*top left*) illustrates a dissection plane superimposed over a triangular mesh, considering a threshold equal to 128. In this case, the given triangular mesh represents an 8-bit gray level image.

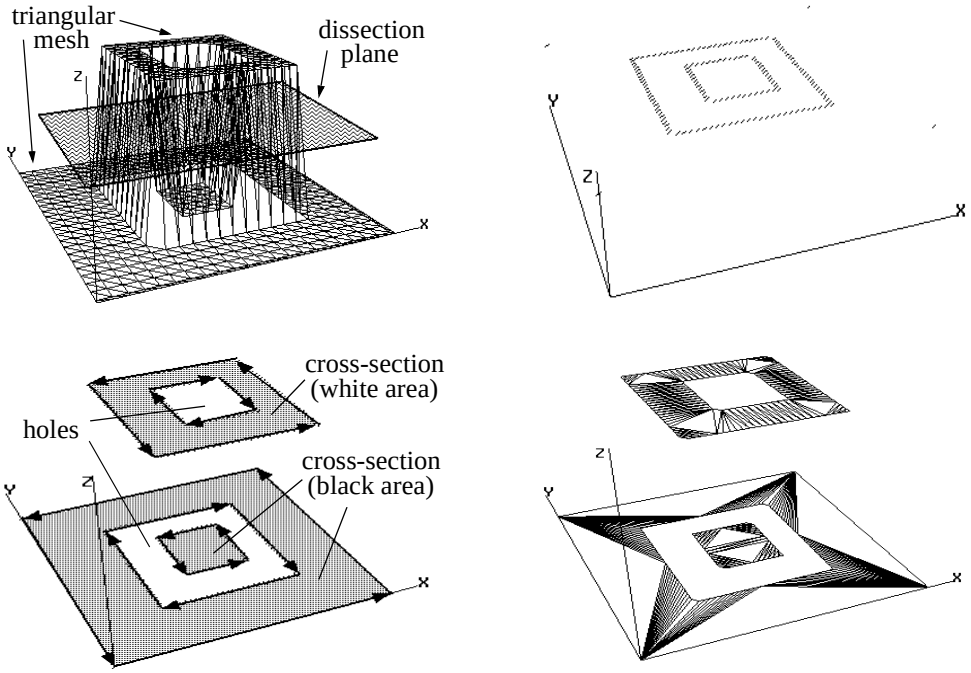

Figure 4.7. (*top left*) Dissection plane superimposed over a triangular mesh. (*top right*) Intersection points obtained after the dissection process. (*bottom left*) Cross-sections above and below the dissection plane, generated after linking the previously obtained intersection points. (*bottom right*) Triangular meshes generated from the previous cross-sections.

The next step intersects the triangular mesh with the previous dissection plane. Several cross-sections are generated as follows. First, the *z* coordinates of the points of the mesh that are contained in the dissection plane are increased in one unit. In this way, the dissection plane can only intersect each triangle at two *intersection points*. Figure 4.7(*top right*) shows the set of intersection points obtained for the example shown in Figure 4.7(*top left*).

If no intersection points are found after the dissection process, the whole triangular mesh is above or below the dissection plane. In the first case, the result is a white image, represented by a rectangular mesh formed by just two triangles, whose *z* coordinates are set to white (e.g., 255). In the second case, the resulting image is black, and it is represented by two triangles whose *z* coordinates are 0.

The segment between each pair of intersection points $\{P_1, P_2\}$ is considered to be a constraint for the subsequent triangulation phase. In order to determine the orientation of this segment in a consistent manner, a plane that passes through the midpoint of the segment and whose normal is the cross product between the normal of the triangle and the normal of the dissection plane is determined. If P_1 is in the positive half-space defined by that plane, the segment is oriented as (P_1, P_2). Otherwise, the segment is oriented as (P_2, P_1).

The intersection points found in this way and their corresponding segments are projected both upwards and downwards. Specifically, two new points are generated for each intersection point. These points have the same *x* and *y* coordinates as the intersection point but different *z* values: one of them is set to white (e.g., 255) and the other to black (e.g., 0). The segments that are projected upwards (*top segments*) keep their original orientation, while the segments projected downwards (*bottom segments*) invert their orientation, Figure 4.7(*bottom left*).

Correlative top segments (the endpoint of a segment coincides with the starting point of the other) are linked forming polylines. The same process is applied to the bottom segments. Each polyline defines a closed region. A polyline will be open when its endpoints lie at any of the four sides of the triangular mesh. If both endpoints lie at the same side, the polyline is closed by linking its endpoints. If the endpoints lie at consecutive sides, the polyline is closed by linking both endpoints to the corresponding corner of the triangular mesh. This implies that the four corners of the triangular mesh are also projected upwards and downwards, similarly to the intersection points, Figure 4.7(*top right*). If the endpoints lie at opposite sides, those endpoints are linked to the two corners that produce a closed, counter-clockwise polyline.

Each closed polyline obtained above defines a cross-section. Thus, top and bottom cross-sections are formed, each corresponding to black and white regions of the binary image. Figure 4.7 (*bottom left*) shows the top and bottom cross-sections obtained after the application of this process.

4.2.2.2 Triangular Mesh Generation

The top and bottom cross-sections obtained above are triangulated separately, using their respective polylines as constraints. A public implementation of the 2D constrained Delaunay triangulation algorithm presented in (Shewchuk, 1996) is utilized. The triangulation is applied to the *x* and *y* coordinates of the points that belong to those polylines.

By construction, clockwise polylines define holes in the final triangulation, while counter-clockwise polylines define solid areas. It is possible that several polylines with alternative orientations are inside each other, giving rise to an alternation of solid and empty areas, Figure 4.7(*bottom left*). It is also possible that two counter-clockwise polylines intersect. In that case only the area belonging to the intersection is solid. The algorithm presented in (Shewchuk, 1996) is able to deal with both situations.

The triangular meshes generated for the current example are shown in Figure 4.7(*bottom right*). The integration of the top and bottom meshes is simply done by triangulating the endpoints of each top segment of a polyline with their corresponding points in the bottom segment. Those triangles define vertical walls in the final triangular mesh.

In order to obtain a digital image from the previously computed triangular mesh and, in general, from a $2^1/_2$D triangular mesh, any of the image generation algorithms described in Section 3.4 can be applied. Basically, the triangular mesh is uniformly sampled at as many positions as pixels the final approximating image has. The gray level of each pixel is the *z* value of its corresponding sample.

Two binary images generated with the proposed algorithm are shown in Figure 4.8(*middle column*). The corresponding binary images obtained with a conventional thresholding algorithm (Umbaugh, 1998) are shown in Figure 4.8(*right column*). The initial adaptive triangular meshes were obtained with the algorithm presented in Section 3.3.2. Figure 4.9 shows the original gray level images, their adaptive triangular meshes and the approximating images corresponding to those meshes.

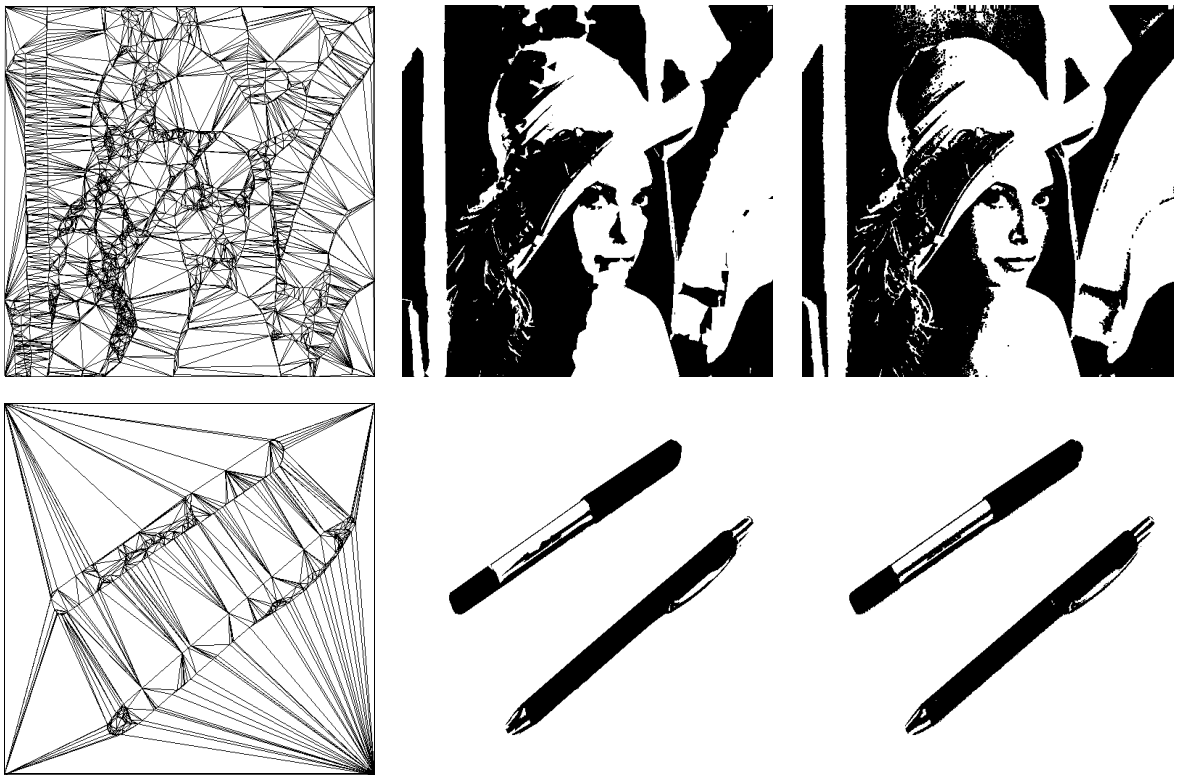

Figure 4.8. (*left column*) Triangular meshes obtained by applying the proposed algorithm: (*top*) 8,502 points (*bottom*) 1,134 points. (*middle column*) Approximating images obtained from the previous triangular meshes. (*right column*) Binary images generated with a conventional image processing algorithm. Threshold set to 116.

4.2.2.3 Experimental Results

The proposed algorithm has been tested upon a set of 18 triangular meshes that represent gray level images of different size, and also compared to a conventional image processing software (CVIPtools) presented in (Umbaugh, 1998). The tested meshes were obtained with the algorithm described in Section 3.3.2. Figure 4.9(*middle column*) shows two of the utilized adaptive triangular meshes.

The CPU times corresponding to the thresholding operation with the proposed algorithm and CVIPtools are displayed in Figure 4.10, considering the two given triangular meshes. These algorithms were tested with thresholds ranging between 0 and 255. The CPU times were measured on a SGI Indigo II with a 175MHz R10000 processor.

Figure 4.9. (*left column*) Original images with 1,048,576 pixels (1024x1024, Lenna) and 262,144 pixels (512x512, pens). (*middle column*) Adaptive triangular meshes generated with the technique presented in Section 3.3.2: (*top*) 9,497 points (*bottom*) 1,354 points. (*right column*) Approximating images obtained through z-buffering (Section 3.4.2). RMS Error: (*top*) 10.8, (*bottom*) 6.8, considering 256 gray levels.

Experimental results show that this thresholding operation may be more efficient in the geometric domain than in the image domain. The actual times depend on the number of triangles intersected by the dissection plane, which, in turn, depends on the contents of the image, the resolution of the triangular mesh (i.e., its approximation error) and the chosen threshold. Experiments have shown that, in general, an effective speed-up is guaranteed when the original images have more than a hundred times more pixels than points their corresponding triangular meshes have. This is the case of the examples shown in Figure 4.8 and Figure 4.9.

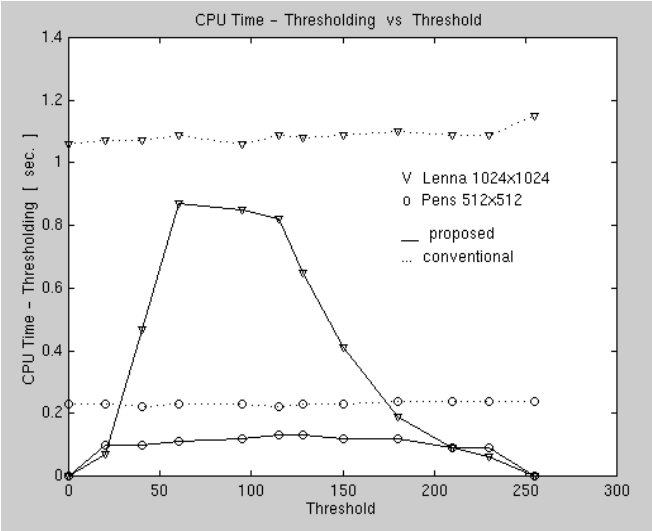

Figure 4.10. CPU times for the thresholding operations, using both the proposed and conventional algorithms with different thresholds.

The previous CPU times do not consider the image-to-mesh and mesh-to-image stages. The reason is that these stages must only be applied once. If many operations are performed (chained) in the geometric domain, the overhead of those two stages will become negligible.

4.2.3 Quantization of Triangular Meshes

Triangular mesh quantization refers to the process of defining a set of possible *z* coordinate values that a point of a given $2^1/_2$D triangular mesh may take. This process consists of mapping predefined ranges of *z* values of the given triangular mesh to single *z* values. Within the image processing field, this operation is referred to as *image quantization*.

The proposed quantization technique applies two main stages for mapping every predefined range of z values to a single *z* value. The first stage dissects the triangular mesh with as many horizontal planes as desired *z* values. Those planes are parallel to the *xy* reference plane of the given mesh and intersect the *z* axis at values that correspond to specified thresholds, Figure 4.11(*left*). Each *parallel plane* generates a mesh dissection, which is obtained

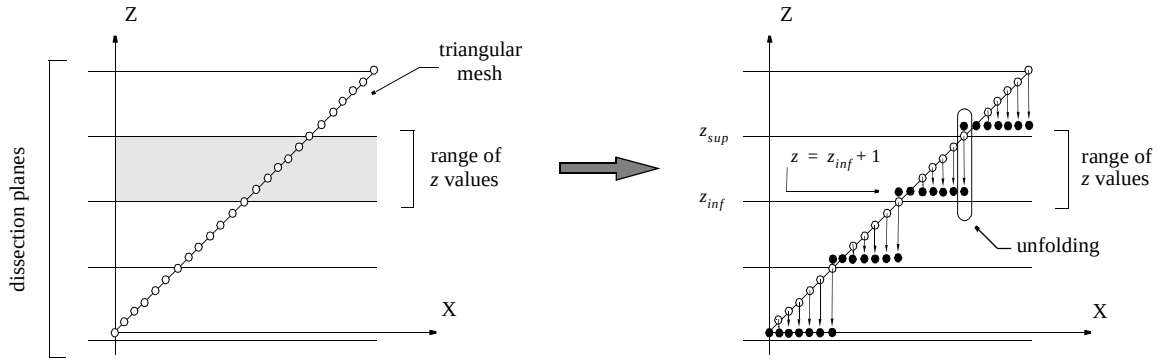

Figure 4.11. Illustration in the 2D space (*zx* plane) of the process of multiple dissection of a triangular mesh. (*left*) Parallel planes dissecting the given triangular mesh. (*right*) The range of *z* values comprised between two consecutive planes is projected to coordinate $z = z_{inf} + 1$, where z_{inf} corresponds to the *z* height of the inferior plane. Furthermore, the intersection points are unfolded.

by applying the dissection algorithm proposed in the thresholding operation described in Section 4.2.2.

The range of *z* values comprised between two consecutive parallel dissection planes is mapped to a single *z* value that corresponds to coordinate $z = z_{inf} + 1$, where z_{inf} is the height of the inferior dissection plane, Figure 4.11(*right*). Furthermore, each intersection point obtained after applying the mesh dissection process is unfolded into two points, by projecting every intersection point both upwards and downwards. Those points have the same (x, y) coordinates as the intersection point, but different *z* coordinates: one of them is set to coordinate $z = z_{inf} + 1$, where z_{inf} is the *z* coordinate of the inferior plane and the other is set to coordinate $z = z_{sup} + 1$, where z_{sup} is the *z* coordinate of the superior plane. Figure 4.11(*right*) illustrates an example of this process.

The second stage of the algorithm retriangulates the triangles that have been dissected with the previous process. In this stage, the final quantized triangular mesh is generated. Figure 4.12(*middle*) illustrates an example of the resulting triangular mesh after applying the second stage of the proposed algorithm. Both stages are described in more detail below.

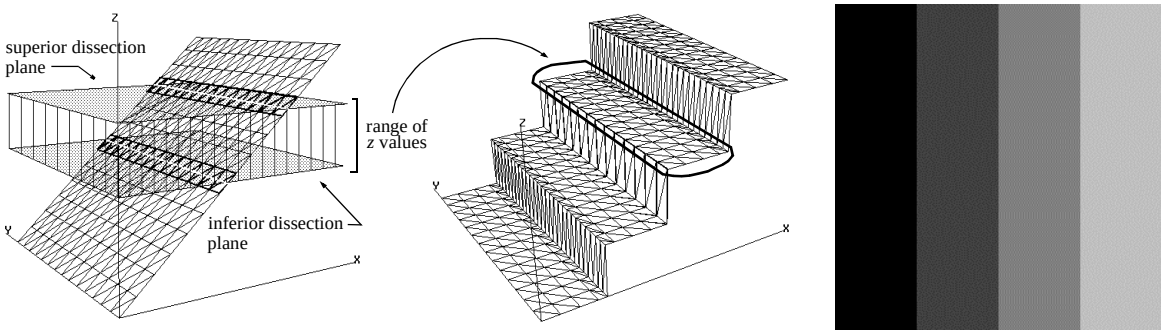

Figure 4.12. Illustration of the triangular mesh quantization process in the 3D space. (*left*) Inferior and superior plane dissecting a range of *z* values in the triangular mesh. The black triangles are labeled as dissected, while the others are labeled as non-dissected. (*middle*) Triangular mesh generated after applying the quantization process by using four dissection planes. (*right*) Approximating image obtained from the previous quantized triangular mesh (four *z* values are generated).

4.2.3.1 Multiple Dissection of a Triangular Mesh

Given a triangular mesh *M*, a multiple dissection process is applied upon *M*. This process dissects the given mesh at as many horizontal planes as possible *z* values, *nz*, are desired for the final quantized mesh. Each mesh dissection is performed with a plane parallel to the *xy* reference plane of *M*, by using the dissection algorithm proposed for the thresholding operation (Section 4.2.2). These dissection planes intersect the *z* axis at a value defined as follows:

$$z_i = i \quad \frac{Z_{MAX} + 1}{nz} - 1 \tag{4.7}$$

where $i \in [0, nz]$ and Z_{MAX} corresponds to the maximum value that can take the *z* coordinate of the mesh points. Considering that the given triangular mesh *M* represents, for example, an 8-bit gray level image, the maximum number of *z* values, *nz*, will be 256. The value of *nz* must be specified by the user.

Once the intersection values with the *z* axis, z_i, for every dissection plane have been determined, the triangular mesh is dissected by using those planes one after the other.

In Equation (4.7), the first plane intersecting the *z* axis at $z_0 = -1$ is not considered as a dissection plane, but as the first *inferior dissection plane*. When the following dissection plane is determined, it is considered as the *superior dissection plane*. That superior plane dissects the triangular mesh by applying the algorithm proposed in Section 4.2.2 (thresholding operation) in order to obtain the corresponding *intersection points*. In order to accelerate the next stage of the algorithm, only the triangles that are dissected by the superior plane are labeled as dissected, while the others are labeled as non-dissected. Figure 4.12(*left*) illustrates the previous steps. The range of *z* values comprised between the inferior and superior plane, $[z_0, z_1)$, corresponds to the *z* coordinates that should be mapped to a single *z* value. Those values are finally set to $z = z_0 + 1$. Finally, each intersection point is unfolded in two points by projecting it to $z = z_0 + 1$ and $z = z_1 + 1$, while keeping the same *x* and *y* coordinates.

When a new dissection plane is computed, the previous superior dissection plane is considered to be the inferior dissection plane, while the new dissection plane becomes the new superior dissection plane.

The next stage of the algorithm triangulates the regions defined by the triangles labeled as dissected.

4.2.3.2 Triangular Mesh Generation

Taking advantage that the points comprised between each pair of dissection planes are maintained, the retriangulation process is only applied to the dissected triangles, keeping the topology of the non-dissected triangles. This retriangulation process is applied as follows.

Consider a triangle t_j with vertices $\{v_0, v_1, v_2\}$, which has been dissected by a horizontal plane. Furthermore, consider that points p_0 and p_1 are the intersection points

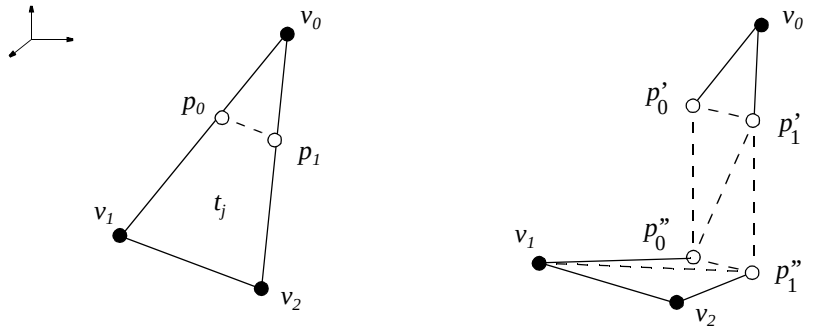

Figure 4.13. Illustration of the retriangulation process. (*left*) Triangle t_j is dissected at points p_0 and p_1 by a horizontal plane. (*right*) Retriangulation of triangle t_j when it is dissected by a superior dissection plane.

corresponding to segments $\{v_0, v_1\}$ and $\{v_0, v_2\}$ respectively, such as illustrated in Figure 4.13(*left*).

A *hash table* is generated in order to store and retrieve all the information about the intersection points. Hash tables are data structures that allow the storage of large amounts of records identifiable by a certain numeric or alphanumeric key, guaranteeing that records are retrieved in constant asymptotic time. In our case, the hash table is defined by a vector with *B* entries, where *B* is the prime number closest to the total number of triangles contained in the given mesh. In this way, the information associated with intersection point p_0 is stored at the entry determined by the following hash function:

$$h(i_0, i_1) = \left(i_0 + i_1 \frac{B-1}{3} \right) / B \tag{4.8}$$

where $\{i_0, i_1\}$ are the identifiers of the two vertices $\{v_0, v_1\}$ respectively.

The intersection points stored at each entry of the hash table keep: *(1)* the identifiers, $\{i_0, i_1\}$, of segment $\{v_0, v_1\}$, *(2)* the coordinates (x, y, z) of intersection point p_0 and *(3)* their opposite point identifiers $\{i_0, i_2\}$ corresponding to segment $\{v_0, v_2\}$. Taking

advantage of the data kept in the hash table, it is very simple to determine the sequence of intersection points to be retriangulated.

The triangulation process is applied by only retriangulating the dissected triangles. First of all, two new points are generated for every intersection point. Hence, the intersection points p_0 and p_1 in triangle t_j generate four new points: $\{p_0', p_0''\}$ and $\{p_1', p_1''\}$, which respectively lie at the inferior and superior dissection planes. Afterwards, two "vertical" triangles are generated from those new points: $\{p_0', p_0'', p_1'\}$ and $\{p_1', p_0'', p_1''\}$, such as displayed in Figure 4.13(*right*). The previous process is applied for every dissected triangle.

Once the mesh generation stage has been performed, a new dissection plane is computed and the previous two stages are performed again. This process concludes when all the dissection triangles have been retriangulated.

The approximating image corresponding to the final triangular mesh is obtained by applying any of the image generation algorithms described in Section 3.4. An example that illustrates the approximating image obtained from the triangular mesh in Figure 4.12(*middle*) is shown in Figure 4.12(*right*).

4.2.3.3 Experimental Results

This section presents experimental results obtained with the proposed quantization algorithm, which was tested upon various adaptive triangular meshes of different size. The utilized triangular meshes were generated with the algorithm described in Section 3.3.2. The CPU times were measured on a SGI Indigo II with a 175MHz R10000 processor.

Figure 4.14(*middle*) shows an adaptive triangular mesh that represents a digital elevation map (DEM), which was generated by converting a DEM file to an 8-bit image and then by approximating it with a triangular mesh through the algorithm described in Section 3.3.2.

Figure 4.15(*left column*) illustrates the triangular mesh obtained after applying the proposed quantization algorithm, considering 8 possible *z* coordinate values. The CPU time was 0.93 sec. For 32 possible *z* coordinate values, the CPU time was 6.65 sec. The corresponding triangular mesh is displayed in Figure 4.15(*right column*). In both cases, the

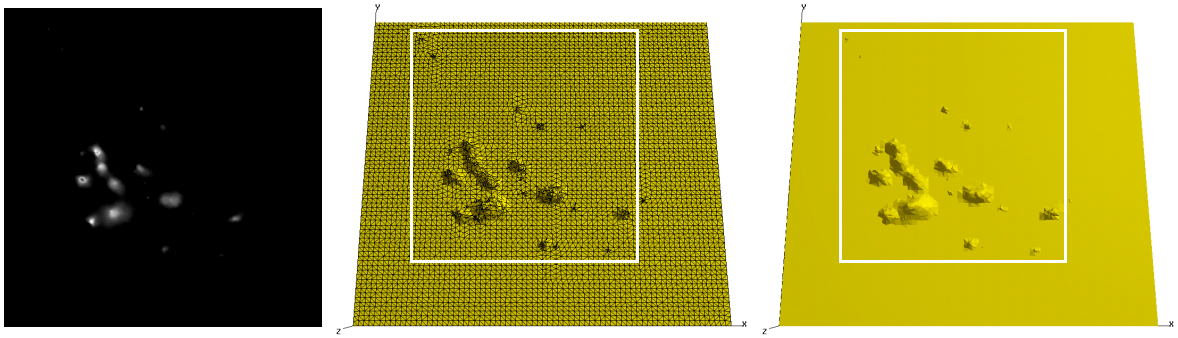

Figure 4.14. (*left*) DEM of the Galapagos Islands converted to an 8-bit image with 262,144 pixels (512x512). (*middle*) Adaptive triangular mesh obtained with the algorithm described in Section 3.3.2 (5,808 sampled pixels). (*right*) Rendered triangular mesh.

approximating images generated from those meshes are shown in Figure 4.15(*bottom row*). The same operations were applied to the original 8-bit image by using CVIPtools (Umbaugh, 1998). In this case, both quantization operations with CVIPtools were performed faster than with the proposed technique. For example, the CPU time with CVIPtools was 0.03 sec. for both 8 and 32 gray levels.

The reason for the poor performance of the geometric technique in this case is the high cost of the multiple dissection and retriangulation process. Therefore, the mesh quantization technique is not advantageous for accelerating digital image processing, although it is useful for being applied upon data originally described by $2^1/_2$D triangular meshes, such as terrain surfaces.

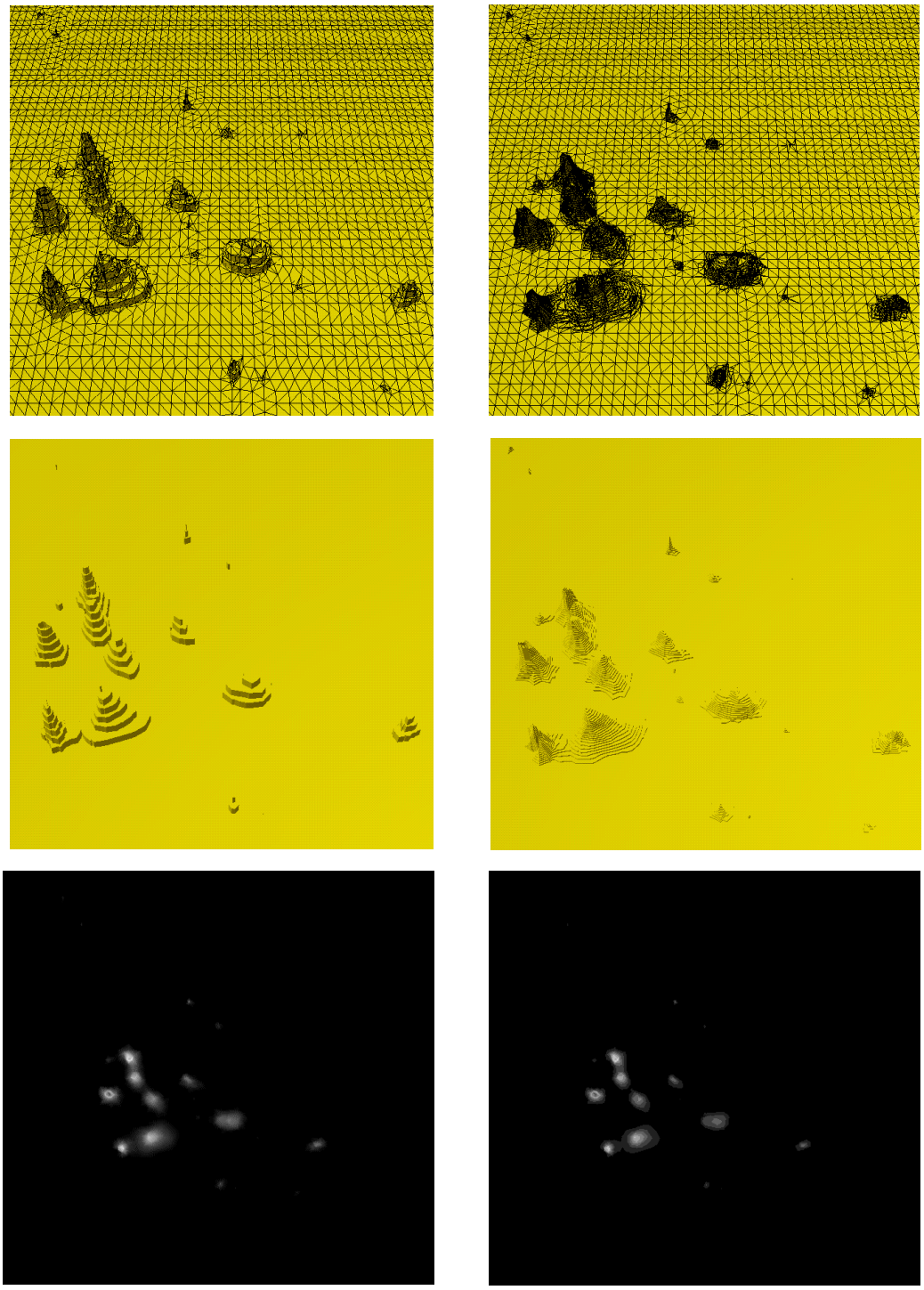

Figure 4.15. Adaptive triangular meshes obtained after applying the proposed quantization process: (*left column*) with 8 possible intensities and (*right column*) with 32 possible intensities. The bottom row shows the approximating images corresponding to the quantized triangular meshes.