4.2.4 Algebraic Operations upon $2^1/_2$D Triangular Meshes

This section describes a set of algorithms to apply algebraic operations to $2^1/_2$D triangular meshes. The algebraic operations have been classified into two basic categories: arithmetic and logic operations. The developed arithmetic operations include addition and subtraction, while the logic operations include: AND, NAND, OR, NOR, XOR and NOT. These operations are performed upon a pair of triangular meshes, except for the NOT logic operation, which only requires a single mesh.

The proposed techniques utilize an algorithm that allows to perform boolean operations between 2D polygons (Murta, 1999). A public implementation of that algorithm is available. It supports four boolean operations: *intersection* (*AND*), *difference*, *exclusive-or* (*XOR*) and *union* (*UNION*). The application of these operations to two simple polygons is shown in Figure 4.16. In each case, the resulting polygons after applying the respective boolean operation are displayed in dark. Similar algorithms can also be found in (Schutte, 1995; Zalik, Gombosi, Podgorelec, 1998).

4.2.4.1 Arithmetic Operations: Addition and Subtraction

The addition and subtraction operations described in this section are applied to a pair of triangular meshes. These operations combine the given meshes into a single mesh.

The proposed algorithm consists of three main stages. The first stage identifies the *intersection bounding box* between the two triangular meshes to be operated, such as in Figure 4.17(*top left*). Taking into account this bounding box, the algorithm labels the triangles of both meshes according to their position. The triangles that are totally inside the intersection bounding box are labeled as *interior*, while the triangles that are outside the intersection bounding box are labeled as *exterior*. The triangles that have a part inside and another part outside the intersection bounding box are labeled as *frontier*. Figure 4.17(*top right*) shows an example of the different types of triangles that are considered by the proposed algorithm.

The second stage of the algorithm intersects the triangles of both meshes that have been labeled as interior or frontier. This stage assumes that each triangle of a mesh represents a polygon that should be intersected with its corresponding intersection triangle

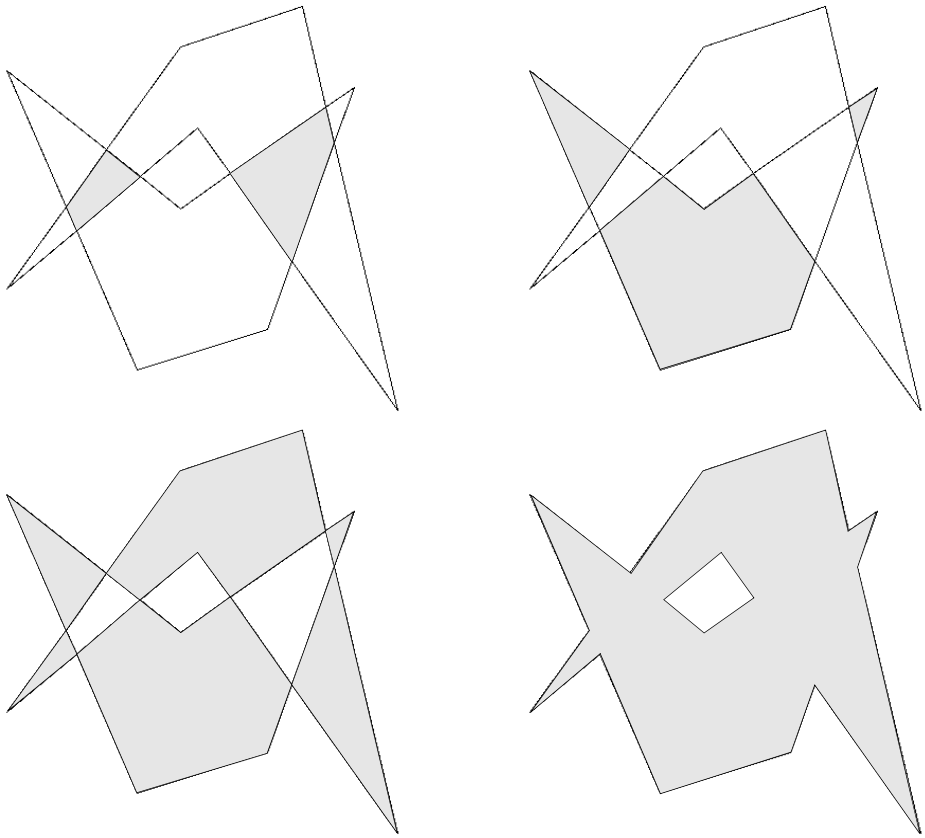

Figure 4.16. Polygons (dark regions) obtained after applying boolean operations between two 2D polygons by using the algorithm presented in (Murta, 1999): (*top left*) intersection, (*top right*) difference, (*bottom left*) exclusive-or and (*bottom right*) union.

(polygon) of the other mesh. The intersection between both triangles is performed through an algorithm that performs an AND boolean operation between two 2D polygons (Murta, 1999). After the AND operation has been applied, a set of *intersection points* is generated. Each intersection point must be added or subtracted (only its *z* coordinate), by computing the plane equations that correspond to the intersected triangles and by evaluating the (x, y) coordinates of the intersection point in those plane equations. The *z* coordinates obtained in this way are added or subtracted according to the specified mathematical operation. The segments that define the boundaries of the intersection regions are kept as constraints for the next triangulation process, Figure 4.17(*bottom left*).

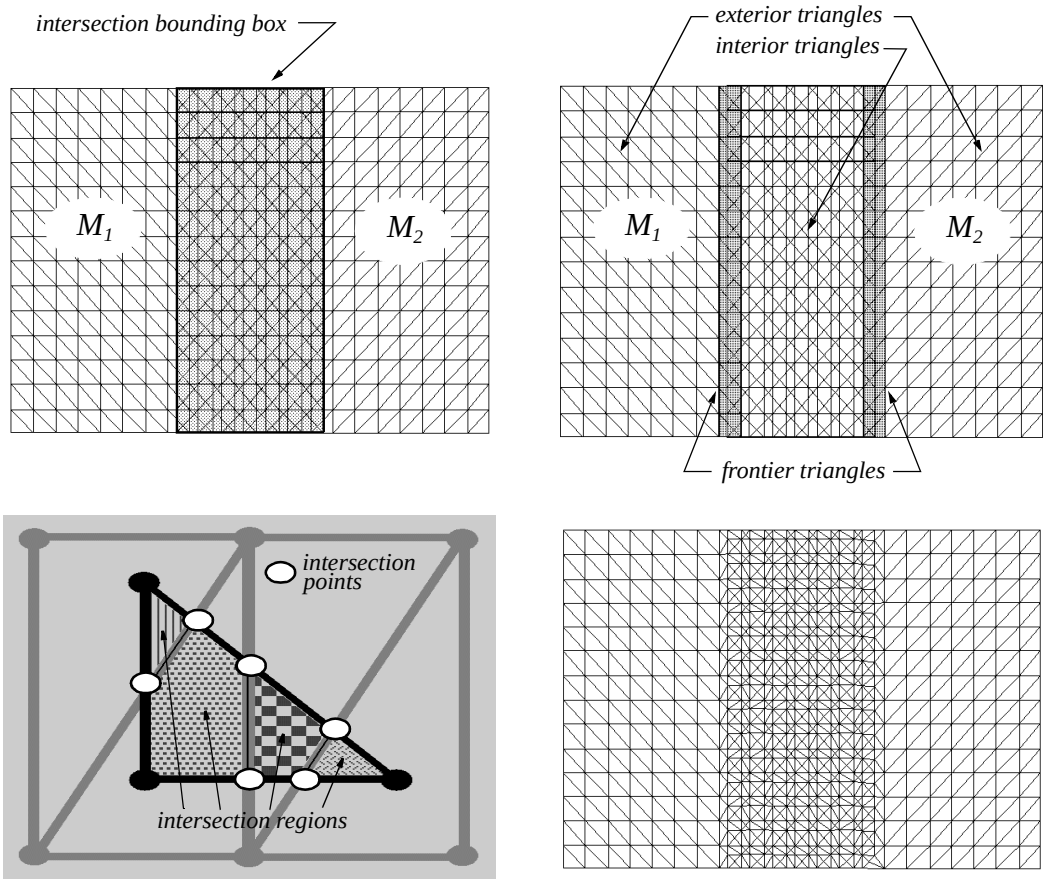

Figure 4.17. Illustration of the addition or subtraction process between two triangular meshes, M_1 and M_2. (*top left*) Detection of the intersection bounding box. (*top right*) Labeling of triangles according to their position in the intersection bounding box. (*bottom left*) Intersection regions and intersection points generated between intersected triangles. (*bottom right*) Resulting triangular mesh after applying the arithmetic operation.

The third stage of the algorithm generates the final triangular mesh. This stage uses as input data the vertices of the exterior triangles and the previously obtained intersection points. Additionally, the topology of the exterior triangles and the segments that define the boundaries of the found intersection regions are considered as constraints for the triangulation. Those data are triangulated by applying the 2D Delaunay triangulation algorithm described in (Shewchuk, 1996). Figure 4.17(*bottom right*) illustrates the final mesh obtained for the current example.

The different stages of this algorithm are further described below.

Determination of the Intersection Bounding Box

Given two $2^1/_2$D triangular meshes, M_1 and M_2, referred to the same global coordinate system, the first stage of the algorithm consists of determining the intersection bounding box between M_1 and M_2 on the *xy* reference plane. Afterwards, the algorithm labels the triangles of both M_1 and M_2 by taking into account the position of those triangles with respect to the bounding box. These steps are described below.

Several cases of intersection between the two input meshes may occur. Figure 4.18 shows those cases in considering different relative positions between them. The first step of this stage determines the intersection bounding box between M_1 and M_2 on the *xy* reference plane, such as it is shown in Figure 4.17(*top left*).

Afterwards, the algorithm labels the triangles of every mesh by taking into account whether their vertices are inside or outside the found intersection bounding box. Three labels are given: a triangle is labeled as interior if its three vertices are totally contained inside the intersection bounding box; if all the vertices are outside the intersection bounding box, the triangle is labeled as exterior; the triangle is labeled as a frontier if at least one of its vertices is interior and the others are exterior to the intersection bounding box. If one or more vertices of the triangle are on the border of the intersection bounding box and the other vertices are exterior or interior to it, the triangle is labeled either as exterior or interior, Figure 4.17(*top right*).

Generation of Intersection Regions

The aim of this stage is to obtain the regions generated after intersecting the triangles overlapped between M_1 and M_2. These regions are generated over the triangles that have been labeled either as interior or frontier as follows.

Consider two overlapped triangles, T_1 and T_2, belonging to M_1 and M_2 respectively. Both triangles are intersected by projecting them onto the *xy* reference plane, Figure 4.19(*left*), and then, by using an algorithm that allows to perform boolean operations

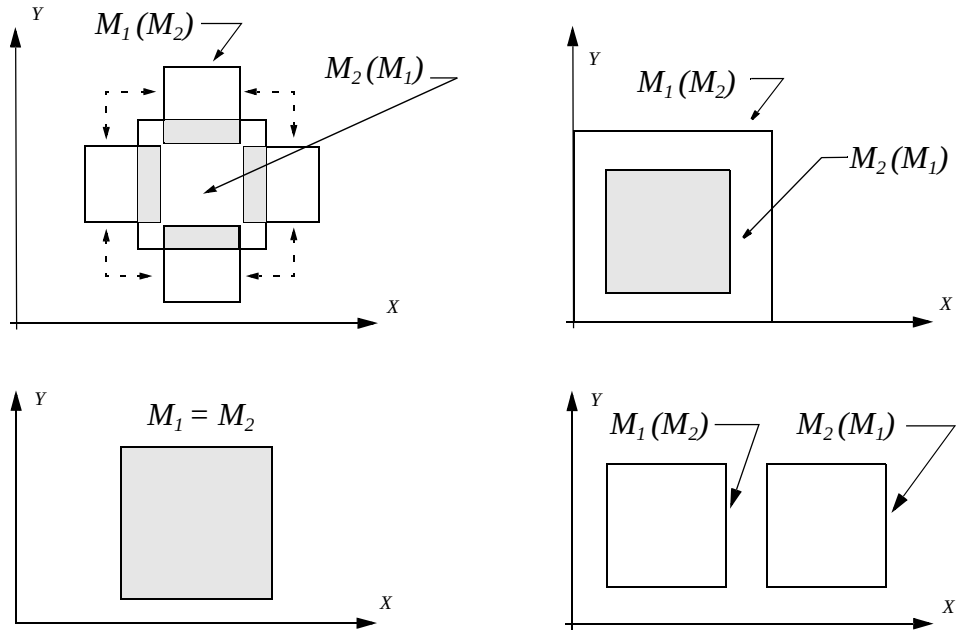

Figure 4.18. Determination of the intersection bounding box (shaded box) between two triangular meshes, M_1 and M_2. (*top left*) When M_1 and M_2 are partially intersected. (*top right*) When M_1 (M_2) is totally contained inside M_2 (M_1). (*bottom left*) When M_1 and M_2 are totally intersected and have the same bounding box size. (*bottom right*) When M_1 and M_2 are not intersected.

between 2D polygons (Murta, 1999). An AND operation is performed upon both projected triangles. Their intersection generates an intersection region that is defined on the *xy* plane, Figure 4.19(*right*). This region contains a set of intersection points and a set of segments that delimit the region boundaries. Those segments are considered as constraints in the next triangulation process.

Finally, the (x, y) coordinates of every obtained intersection point are evaluated in the plane equations corresponding to T_1 and T_2 in order to determine the *z* coordinates inside T_1 and T_2. The found *z* coordinates are added or subtracted according to the desired operation, Figure 4.19(*right*).

The previous steps are applied until all the overlapped triangles have been considered.

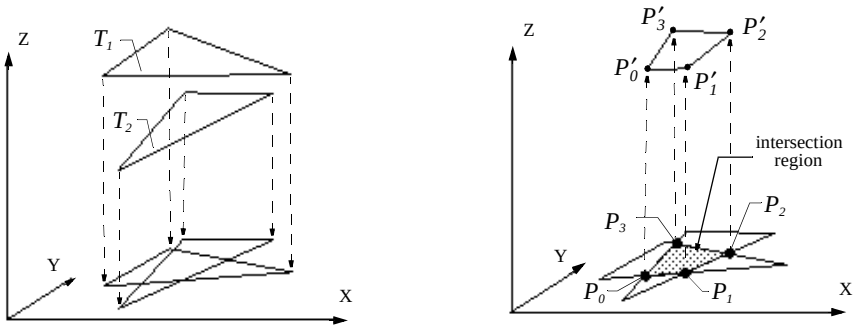

Figure 4.19. (*left*) Projection of T_1 and T_2 on the *xy* reference plane. (*right*) Intersection region between T_1 and T_2 on the *xy* reference plane. This intersection region is mapped to the 3D space by evaluating the (x, y) coordinates of every obtained intersection point in the plane equations corresponding to T_1 and T_2 in order to determine the *z* coordinates inside T_1 and T_2. The found *z* coordinates are added or subtracted according to the desired arithmetic operation.

Generation of the Triangular Mesh

The final aim of this algorithm is the generation of a triangular mesh from the set of vertices that have been labeled as exterior and from the points obtained after applying the intersection process. The triangulation process utilizes the exterior triangles as constraints, as well as the segments that delimit the boundaries of the intersection regions obtained before. The resulting mesh is obtained by using the 2D constrained Delaunay triangulation proposed in (Shewchuk, 1996). Figure 4.17(*bottom right*) displays the final triangular mesh generated for the current example.

Experimental Results

Different adaptive triangular meshes have been processed by using the aforementioned arithmetic operations. Those meshes were generated with the algorithm described in Section 3.3.2. In particular, this section presents the results obtained after applying the addition operation upon two triangular meshes, and after performing an integration process upon a group of triangular meshes. In both cases, the CPU times were measured on a SGI Indigo II with a 175MHz R10000 processor.

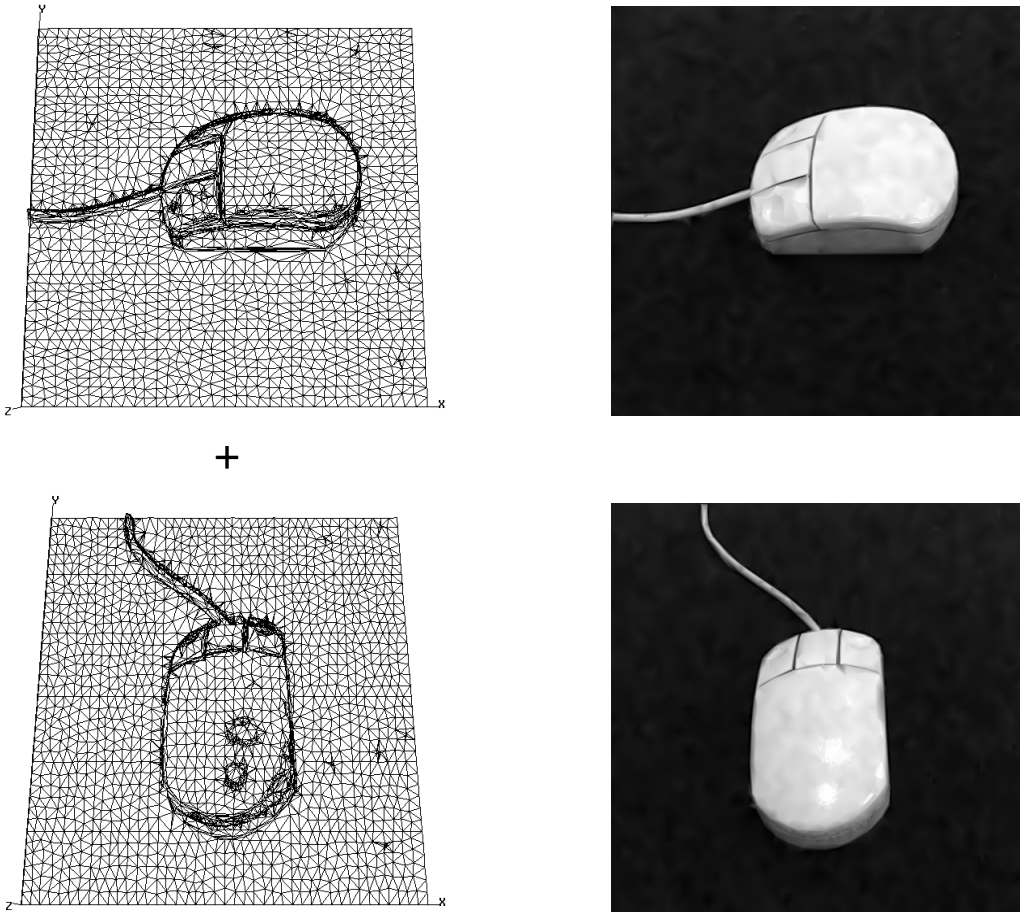

Figure 4.20. (*left column*) Input triangular meshes to be merged with the proposed addition operator: (*top*) 2,461 and (*bottom*) 2,624 points. (*right column*) Approximating images generated from the previous meshes. Both images contain 262,144 pixels (512x512).

Figure 4.20(*left column*) illustrates the two triangular meshes to be added. These meshes are approximations of 8-bit images. Their respective approximating images are shown in Figure 4.20(*right column*). Those input meshes contain 2,461 and 2,624 points respectively. The CPU time after applying the addition operation was 21.18 sec. The resulting mesh is displayed in Figure 4.21(*left*). This mesh contains 19,776 points. The same addition operation was applied to the corresponding 8-bit images by using CVIPtools (Umbaugh, 1998), a conventional image processing software. In this case, the addition

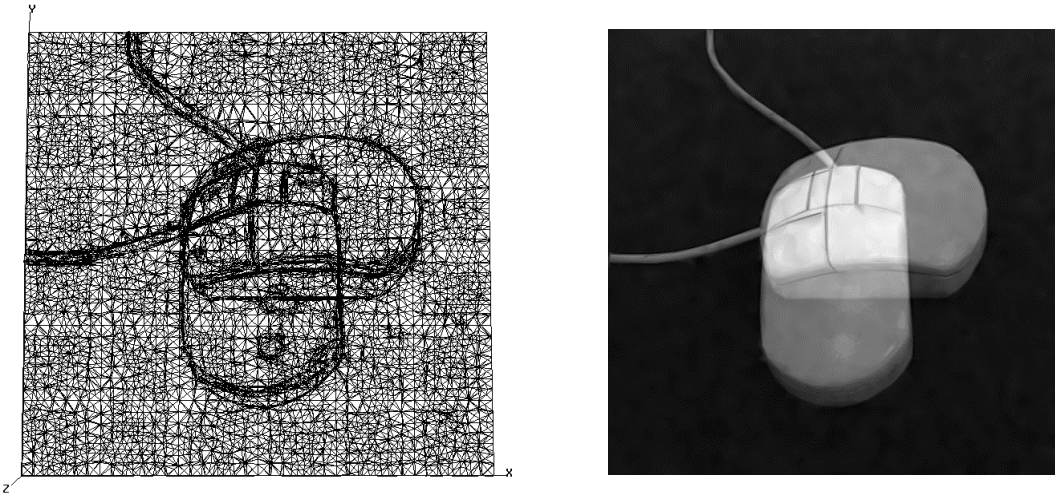

Figure 4.21. (*left*) Resulting triangular mesh obtained after applying the addition operation to the meshes shown in Figure 4.20(*left column*). This mesh contains 19,776 points. (*right*) Approximating image generated from the previous mesh.

operation with CVIPtools was performed much faster than with the proposed technique. For example, the CPU time with CVIPtools was 0.14 sec.

As with the image quantization algorithm, the arithmetic operations are much more costly in the geometric domain than in the image domain. Therefore, they are only useful when the input data are triangular meshes representing, for instance, terrain surfaces.

The proposed technique can also be used as a tool for integrating triangular meshes, such as it is shown in the following example. Figure 4.22 shows a group of triangular meshes that are representations of DEM's corresponding to the Balearic Islands. Those meshes contain 1,828 and 1,673 points, Figure 4.22(*top left*) and Figure 4.22(*top right*), and 1,654 and 1,307 points, Figure 4.22(*bottom left*) and Figure 4.22(*bottom right*). These meshes were integrated in pairs: First the two top meshes, then the two bottom meshes and finally the two resulting meshes obtained before.

The total CPU time after applying the whole integration process was 22.98 sec., giving

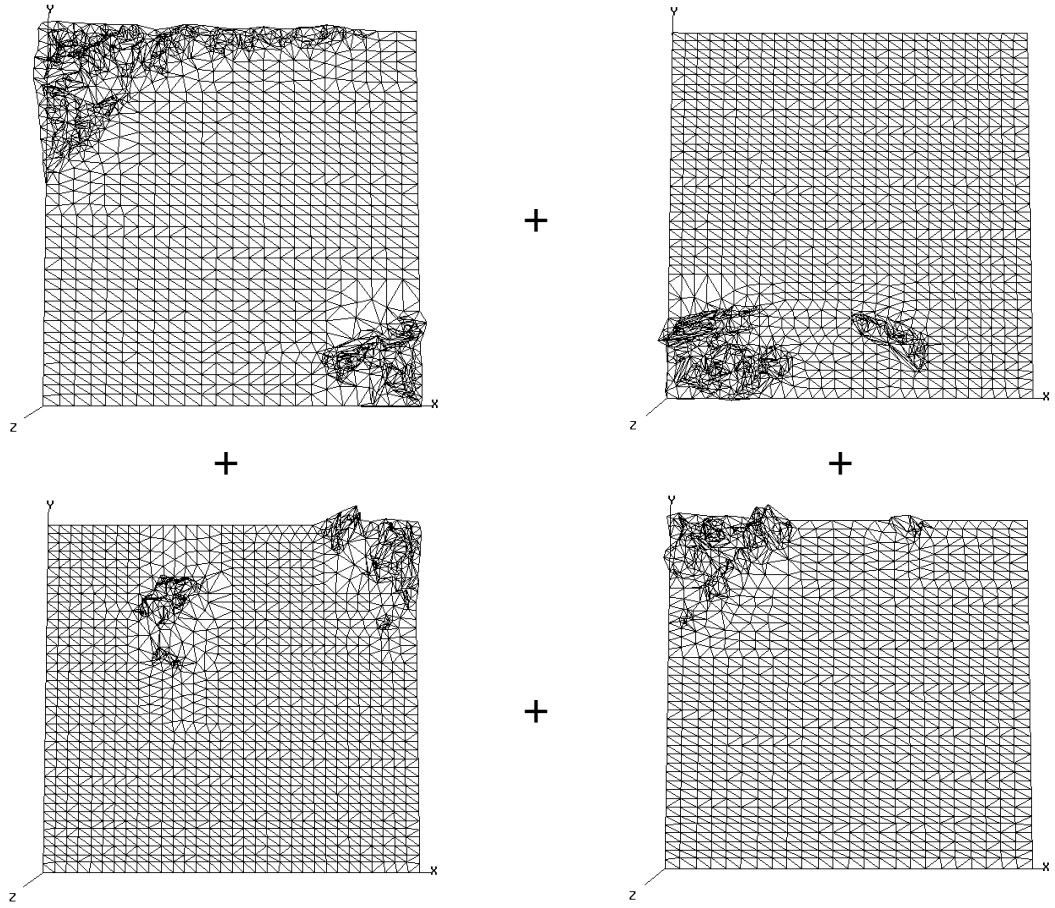

Figure 4.22. Group of triangular meshes to be integrated with the proposed addition operator. Those meshes contain: (*top left*) 1,828, (*top right*) 1,673, (*bottom left*) 1,654 and (*bottom right*) 1,307 points respectively. The given meshes are DEM representations of the Balearic Islands.

rise to a triangular mesh with 13,369 points. That mesh is displayed in Figure 4.23(*top*). The corresponding rendered triangular mesh is shown in Figure 4.23(*bottom*).

4.2.4.2 Logic Operations: AND, NAND, OR, NOR, XOR and NOT

Given two triangular meshes, M_1 and M_2, which have been obtained after applying the thresholding algorithm proposed in Section 4.2.2, this section describes an algorithm to perform AND, NAND, OR, NOR and XOR logic operations upon them, and an algorithm to perform the NOT logic operation, which only requires a single mesh.

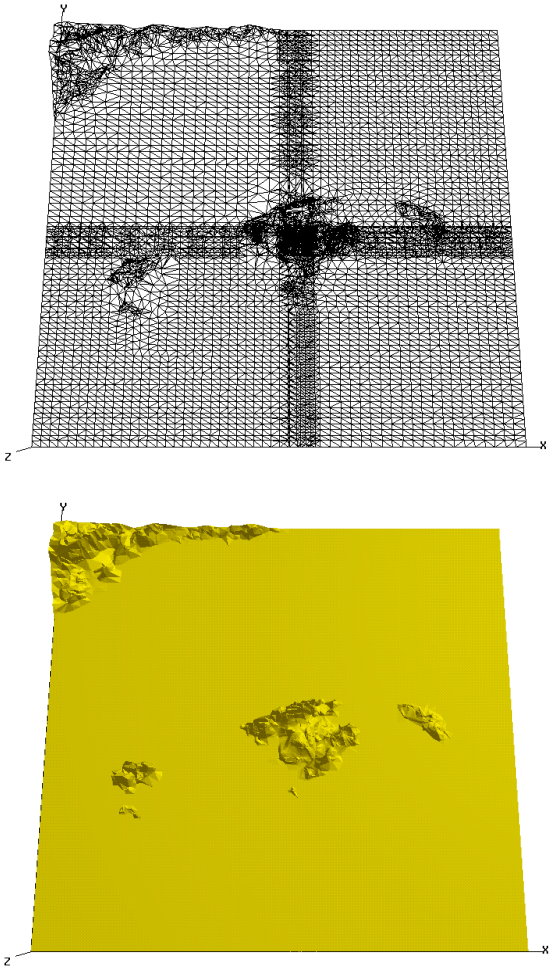

Figure 4.23. (*top*) Resulting triangular mesh obtained after applying the integration process to the previous triangular meshes. The final mesh contains 13,369 points. (*bottom*) Rendered triangular mesh.

The NOT logic algorithm simply consists of finding the negative version of a given mesh. In this way, it only modifies the position of the *z* coordinate of every mesh point. For example, if the given mesh represents an 8-bit gray level image, the new z' coordinate for every mesh point is calculated as $z' = Z_{MAX} - z$, where $z \in [0, Z_{max}]$, Z_{max} represents the maximum *z* coordinate corresponding to the points of the given mesh and Z_{MAX} is the maximum gray level value for 8-bit images. An example that illustrates this process is shown in Figure 4.24.

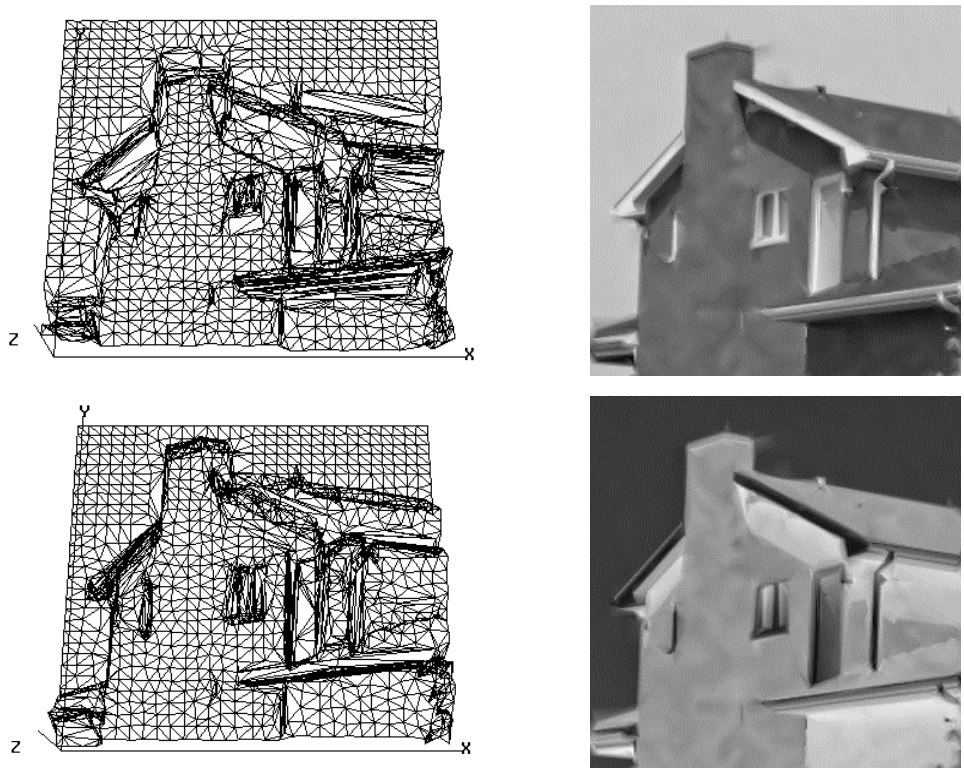

Figure 4.24. NOT logic operation. (*top row*) Input triangular mesh with 1,649 points, and the respective approximating image. The given mesh approximates a gray level image with 65,536 pixels. (*bottom row*) Mesh obtained after applying the NOT logic operation and its approximating image.

On the other hand, the algorithm for implementing the AND, NAND, OR, NOR and XOR logic operations in the geometric domain consists of two stages that are described below.

Logic Operations Among Polygons

This stage considers that the given meshes, M_1 and M_2, are representations of binary images that have been generated with the thresholding algorithm presented in Section 4.2.2. Furthermore, it considers that those meshes should be overlapped and that the *z* coordinate values of their points can only be either 0 or Z_{MAX}. The two examples of these meshes shown in Figure 4.25(*left column*) will be used to illustrate the application of the proposed logic operations.

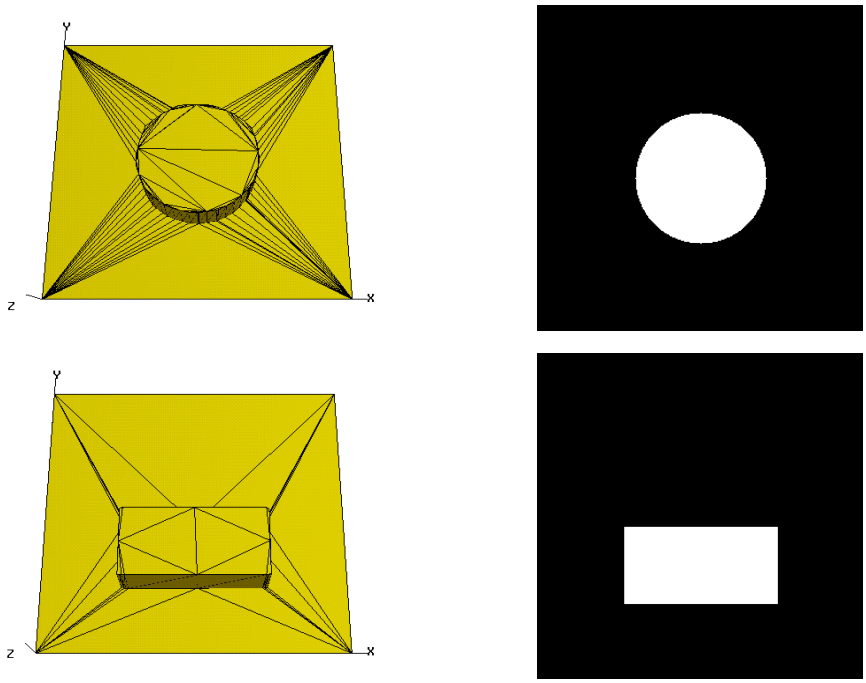

Figure 4.25. (*left column*) Input triangular meshes to which the proposed logic operations are applied. (*right column*) Approximating images generated from the previous triangular meshes.

Initially, the algorithm determines and labels the triangles that represent the vertical walls of the given meshes (*vertical triangles*). Those triangles are determined by computing the angle of the normal vector of every triangle with respect to the *xy* reference plane. If this angle is close to 0, the triangle is labeled as a vertical triangle. Figure 4.26(*left column*) shows an example of the vertical triangles obtained from the test meshes.

Afterwards, starting with the vertical triangles previously found, the algorithm obtains a set of closed polygons that represent the white areas of the given meshes. Each closed polygon is obtained by starting with a single segment (edge) of a vertical triangle. The *z* coordinates of the endpoints of this segment (the *initial segment*) are equal to Z_{MAX}. Starting with this initial segment, an adjacent segment belonging to another vertical triangle is linked to it by taking into account that two segments are adjacent if both of them share an endpoint. This last step is applied until the closed polygon is formed. The previous steps are

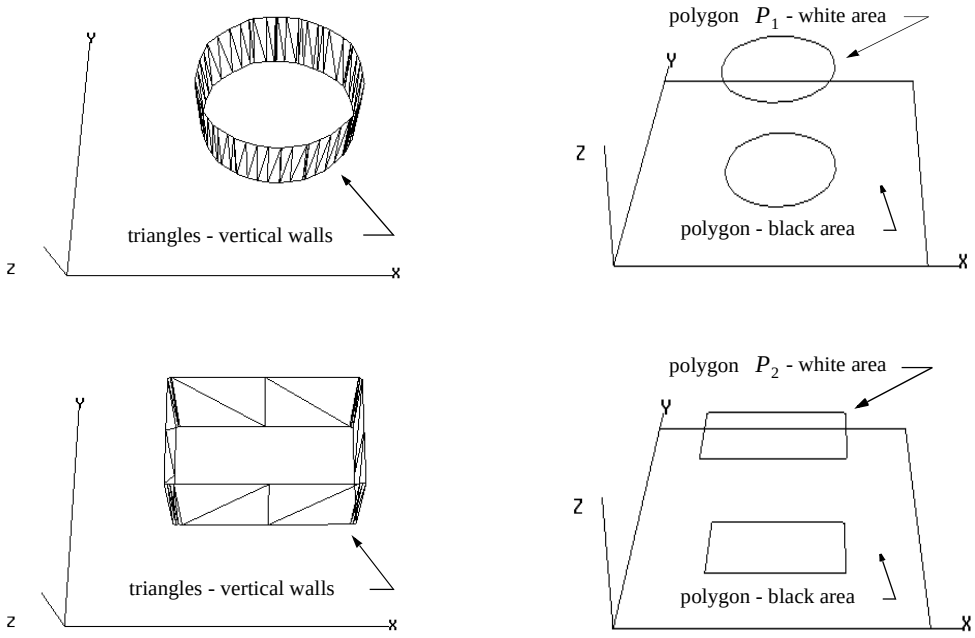

Figure 4.26. (*left column*) Triangles representing the vertical walls of the input meshes. (*right column*) Polygons representing the black and white areas of the input meshes.

iterated in order to generate the remaining closed polygons. When all the vertical triangles of both meshes, M_1 and M_2, have been analyzed, a set of polygons is generated for every mesh, P_1 and P_2. Figure 4.26(*right column*) shows the polygons obtained for the current example. They represent the white areas of the given meshes.

Once both sets of polygons, P_1 and P_2, have been obtained, the next step consists of applying the required logical operation to them. In this way, the algorithm initially projects P_1 and P_2 onto the *xy* reference plane ($z = 0$), such as illustrated in Figure 4.27(*left*). Then, depending on the specified logical operation, the algorithm proceeds as follows:

AND and NAND Operations

In this case the AND operation proposed in (Murta, 1999) is applied as many times as intersections between the projected polygons P_1 and P_2 there are. The resulting intersection regions are mapped to the 3D space by changing the *z* coordinate value of the intersection

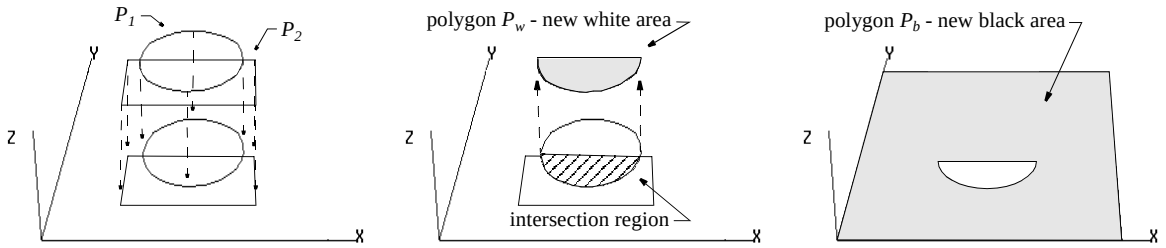

Figure 4.27. (*left*) Polygons that represent the white areas are projected onto the *xy* reference plane. (*middle*) Generation of polygons P_w which represent the new white areas in the final mesh. (*right*) Generation of polygons P_b which represent the new black areas.

points to Z_{MAX}. Thus, these regions define a set of polygons P_w, which represent the new white areas in the final mesh. Figure 4.27(*middle*) shows an example of this process.

Finally, the algorithm generates the respective polygons P_b that define the new black areas. In the first place, the bounding box between M_1 and M_2 is generated. Then, the difference operation proposed in (Murta, 1999) between that bounding box and the polygons P_w is applied onto the *xy* reference plane. The regions obtained after the difference operation constitute the polygons P_b that define the black areas. The *z* coordinate of all the points that belong to P_b is set to 0. Figure 4.27(*right*) shows the polygons P_b obtained for the current example.

If the NAND logical operation is required, the NOT logical operator is applied to the points (*z* coordinate) that constitute the resulting polygons P_w and P_b.

OR and NOR Operations

In this case, the algorithm performs a procedure similar to the one applied for the AND and NAND operations. The only difference is that instead of applying the AND operation proposed in (Murta, 1999) to the projected polygons P_1 and P_2, the UNION operation (Murta, 1999) between P_1 and P_2 is applied. Afterwards, the procedure is the same as before. Hence,

polygons P_b and P_w, which represent the new black and white areas of the final mesh, are obtained.

Similarly to the NAND logical operation, if the NOR logical operation is required, the NOT logical operator is applied to the points (z coordinate) that constitute the resulting polygons P_w and P_b.

XOR Operation

In order to implement the XOR operation, the algorithm applies a procedure similar to the one utilized for the AND and NAND logical operations. The only difference is that, instead of applying the AND operation (Murta, 1999) to the projected polygons P_1 and P_2, the XOR operation (Murta, 1999) is applied.

Triangular Mesh Generation

This stage is responsible for generating the final triangular mesh that represents the result after applying the specified logical operation to the input meshes. It utilizes the triangulation algorithm proposed in (Shewchuk, 1996) and consists of three steps. First, the points that belong to the polygons P_w, which represent the new white areas, are triangulated. This triangulation considers the segments of P_w as constraints.

The second step is similar to the previous one, but this time the points of the polygons P_b, which represent the new black areas, are triangulated, keeping the segments of those polygons as constraints. Finally, the resulting triangular mesh is obtained by integrating the previous meshes. This is done by merging those meshes through the segments that form the polygons P_w and P_b, creating thus new vertical triangles.

Three different meshes obtained after applying the AND, OR and XOR logical operators upon the two meshes shown in Figure 4.25(*left column*) are displayed in Figure 4.28(*top row*). The respective approximating images are also shown in Figure 4.28(*bottom row*).

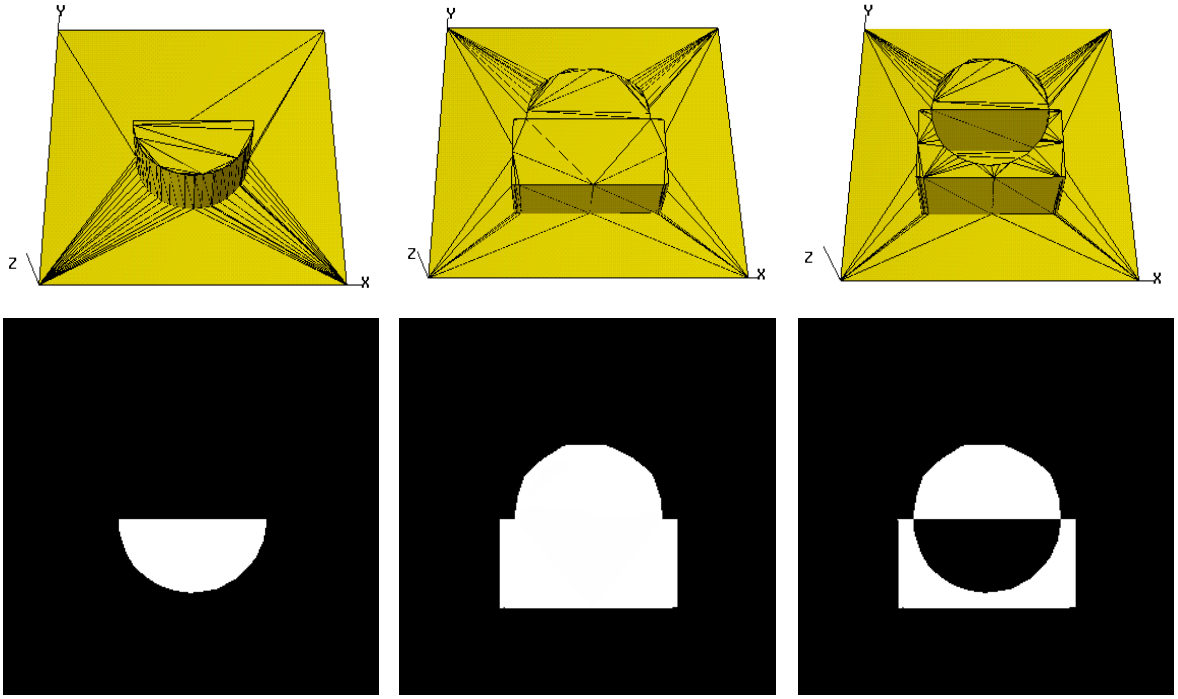

Figure 4.28. (*top row*) Resulting triangular meshes obtained after applying the: (*left*) AND, (*middle*) OR and (*right*) XOR logic operation. (*bottom row*) Approximating images obtained from the previous triangular meshes.

Experimental Results

The proposed logical operators have been evaluated by processing several adaptive triangular meshes generated with the thresholding algorithm described in Section 4.2.2. This section presents the results obtained after applying the NOT operator to the triangular mesh shown in Figure 4.24(*top left*), and after applying the AND, OR and XOR operations to the triangular meshes shown in Figure 4.25(*left column*). The obtained CPU times were measured and compared to the times corresponding to the application of the same operations with CVIPtools (Umbaugh, 1998). The CPU times were measured on a SGI Indigo II with a 175MHz R10000 processor.

The CPU time for the proposed NOT operator was 0.0001 sec., while the same operation applied with CVIPtools took 0.01 sec. The proposed technique is more efficient since

only a small percentage of points are processed in the geometric domain, while, in the image domain, all the pixels that constitute the image must be considered. Therefore, the NOT operator is always more efficient in the geometric domain than by sequentially processing all the image pixels.

On the other hand, when the AND, OR and XOR operators were applied upon the given triangular meshes, the CPU times in the geometric domain were higher than with CVIPtools. For example, the CPU times corresponding to the AND, OR and XOR operations in the geometric domain were 0.12 sec., 0.11 sec. and 0.11 sec., while in the image domain were 0.01 sec. in all cases. The reason is that the number of operations performed in the geometric domain is very superior to the number of operations performed in the image domain, although a small percentage of points is only processed in the geometric domain.

4.2.5 Selection of Regions-of-Interest from Triangular Meshes

This section presents an algorithm to select a region-of-interest (*ROI*) from a given $2^1/_2$D triangular mesh. The selected region-of-interest is specified by the user and is defined by a list of points that determine the segments of a closed polygon within the mesh. The proposed algorithm consists of two stages that are described below.

4.2.5.1 Dissection of a Region-of-Interest in a Triangular Mesh

The first stage dissects the given triangular mesh with a set of segments that define a closed polygon representing the region-of-interest. The result of this dissection is a set of intersection points that are obtained as follows.

First, each segment of the specified ROI generates a vertical dissection plane. This dissection plane contains that segment and is orthogonal to the *xy* reference plane. Figure 4.29(*left*) shows the segments of a predefined ROI as thickened lines.

The next step dissects the given triangular mesh by using the previously obtained orthogonal planes. This dissection process is similar to the one applied in Section 4.2.2.1 in order to threshold triangular meshes, but, in this case, the number of dissections depends on the number of dissection planes that represent the ROI. Furthermore, the intersection points

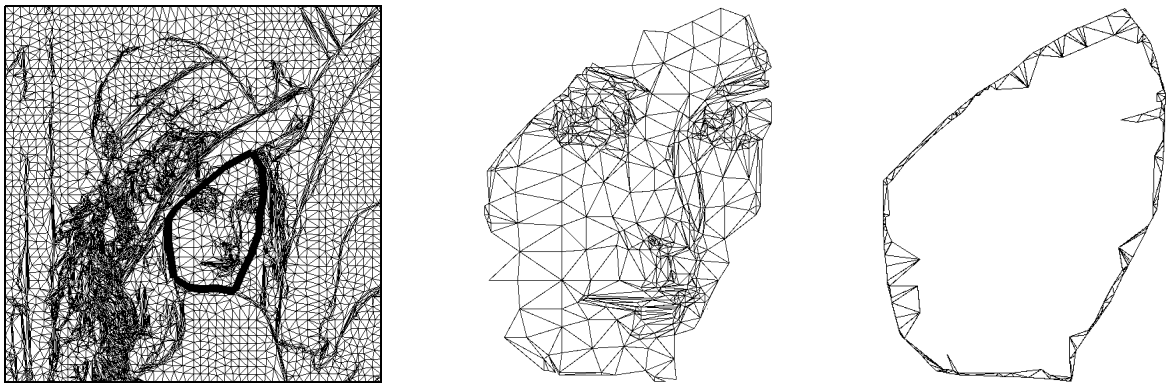

Figure 4.29. (*left*) Selected ROI on the input triangular mesh. The given mesh contains 7,492 points and approximates a gray level image with 262,144 pixels. (*middle*) Interior triangular mesh defined inside the selected ROI. (*right*) Exterior triangular mesh generated from the intersection points and the points that delimit the boundaries of the previous interior triangular mesh.

between the triangular mesh and the orthogonal planes do not have to be projected onto any plane.

4.2.5.2 Generation of the Final Triangular Mesh

Once the intersection points have been generated, the next step consists of obtaining the resulting triangular mesh. This mesh is generated in three steps.

First, the triangles of the given mesh that are contained in the selected ROI are preserved. The topology of those triangles generates an *interior triangular mesh* inside the ROI. The second step generates an *exterior triangular mesh* by triangulating the obtained intersection points and the points that delimit the boundaries of the interior triangular mesh. The segments that define the specified ROI as well as the edges that delimit the boundaries of the interior triangular mesh are kept as constraints in the triangulation. Figure 4.29(*middle*) and Figure 4.29(*right*) show both the interior and exterior triangular meshes generated for the current example.

Finally, the resulting triangular mesh is obtained by merging the previously obtained exterior and interior triangular meshes, such as shown in Figure 4.30(*left*).

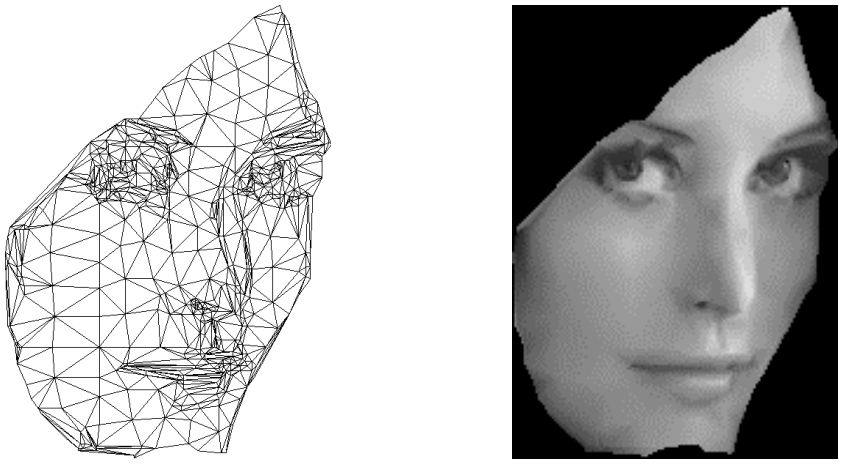

Figure 4.30. (*left*) Resulting triangular mesh obtained after linking the interior and exterior triangular meshes. (*right*) Approximating image generated from the previous triangular mesh.

The approximating image generated from the resulting triangular mesh is obtained through any of the algorithms presented in Section 3.4. Figure 4.30(*right*) shows the approximating image generated from the previous triangular mesh.

4.2.5.3 Experimental Results

The CPU times to perform the ROI operation were measured and compared to the performance of CVIPtools (Umbaugh, 1998). Several adaptive triangular meshes were processed in the geometric domain, while in the image domain, the respective approximated images were utilized. The processed triangular meshes were generated with the algorithm presented in Section 3.3.2. The CPU times were measured on a SGI Indigo II with a 175MHz R10000 processor.

Two types of regions-of-interest were considered with the proposed technique. The first type defines a rectangular region, while the second type defines a polygonal region. When the selected ROI was defined as a rectangular region, the CPU times with the proposed technique were much larger than with CVIPtools.

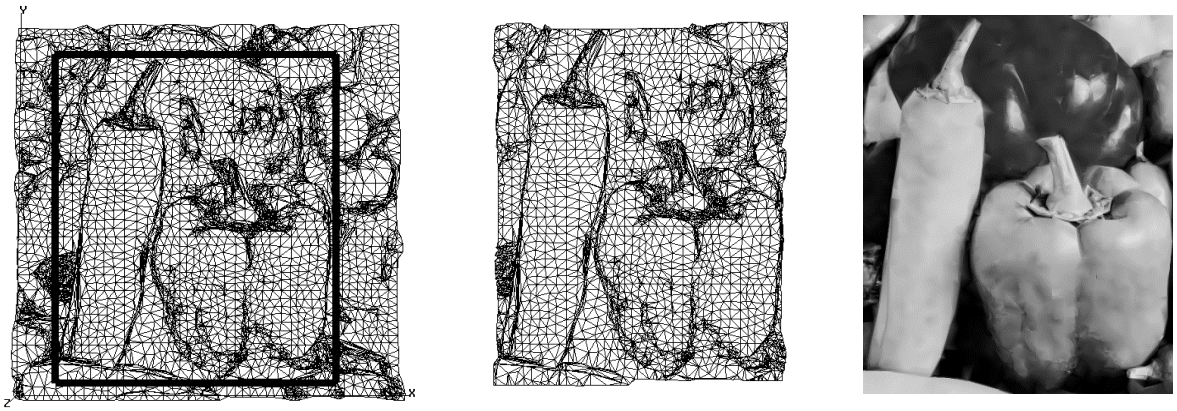

Figure 4.31. (*left*) Rectangular ROI superimposed over the given triangular mesh. (*middle*) Resulting triangular mesh obtained after applying the desired ROI by means of the proposed technique. (*right*) Approximating image generated from the previous triangular mesh.

For example, Figure 4.31(*middle*) illustrates the selected ROI applied to the triangular mesh shown in Figure 4.31(*left*). The CPU time with the proposed technique was 0.17 sec., while with CVIPtools it was 0.01 sec. The reason is that, with a conventional algorithm, it is not necessary to perform any operations to select a ROI. The user simply defines the coordinates of the bounding box that contains the region, and the algorithm simply copies this area to a new image. However, with the proposed algorithm, the triangular mesh must be dissected and, furthermore, the number of dissections depends on the number of triangles that are dissected with the segments that define the desired ROI.

Figure 4.32 shows a ROI that defines a polygonal region. The CPU time to process the given triangular mesh with this ROI was 0.78 sec. CVIPtools does not have the possibility of defining non-rectangular ROIs.

4.2.6 Generation of Synthetic Triangular Meshes

This section presents a set of tools to generate simple synthetic triangular meshes from user specified data. Four types of triangular meshes with different boundaries have been defined: rectangles, circles, ellipses and closed polygons. The triangulation of the data that define the

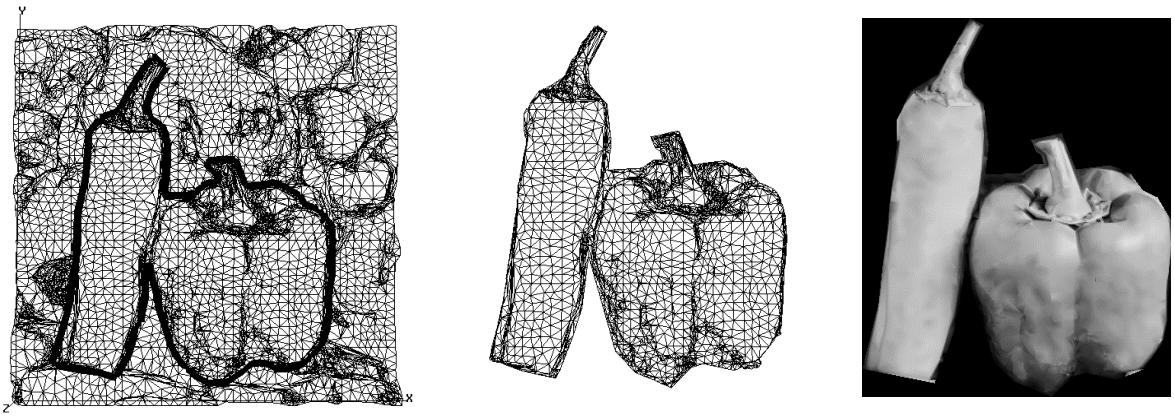

Figure 4.32. (*left*) Desired polygonal ROI superimposed over the given triangular mesh. (*middle*) Resulting triangular mesh obtained after selecting the desired ROI with the proposed technique. (*right*) Approximating image generated from the previous triangular mesh.

specified boundary is obtained by applying the 2D constrained Delaunay algorithm proposed in (Shewchuk, 1996), by using that boundary as a constraint for the triangulation. These meshes are generated as follows:

Rectangular Bounded Mesh

The user defines the minimum and maximum coordinates for the desired rectangular bounded triangular mesh: (X_{min}, Y_{min}) and (X_{max}, Y_{max}) respectively. Furthermore, the range of *z* values in which the mesh is defined is also specified by defining a value Z_{min} for the points with coordinate X_{min} and a value Z_{max} for those points with coordinate X_{max}. If $Z_{min} = Z_{max}$ the obtained mesh defines a plane parallel to the *xy* reference plane. On the other hand, if these values are different, the generated mesh defines a ramp.

Circular or Elliptical Bounded Mesh

This function receives the coordinates of the center (x_c, y_c) of the circle or ellipse, and the value of the radius *r* or the biggest and smallest axis, a_b and a_s. The *z* coordinate of all the mesh points is set to a given constant, Z_{max}. Those points are computed with the following equation:

$$\begin{aligned} x' &= R_x \cos\beta + x_c \\ y' &= R_y \sin\beta + y_c \\ z' &= Z_{max} \end{aligned} \tag{4.9}$$

where $R_x = R_y = r$ for the circular bounded mesh or $R_x = a_b/2$ and $R_y = a_s/2$ for the elliptical bounded mesh; β is the angle for every point of the required mesh, computed as: $\beta = 2\pi n/s$, where s represents the number of points that are sampled on the circle or ellipse and $n \in [1, s]$. It is advisable that $s \geq 15$ in order that the generated mesh approximates the circular or elliptical bounded mesh in an acceptable way.

Free-Form Bounded Mesh

This algorithm generates a triangular mesh bounded by a polygonal contour. The algorithm receives as input data a list of points that define a closed polygon. This polygon is approximated by a cubic spline. This curve is evaluated at new points that define a new closed polyline whose points are then triangulated. The segments of that polyline are utilized as constraints. A z coordinate value, Z_{max}, for the vertices of the resulting triangular mesh is predefined by the user.

Several synthetic triangular meshes generated with the functions proposed in this section are illustrated in Figure 4.33.