Chapter 06

Surface Rendering

6.1 INTRODUCTION

Surface rendering is one of the two major rendering modes for medical volume data. Instead of classifying volume data and mapping it directly to the viewport, surface rendering is based on an indirect surface mesh representation. This mesh is either generated by *extracting an isosurface* from the original volume data or by *transforming a segmentation result*. As we have discussed in § 4.6.2, surface renderings may not only visualize the final segmentation result, but also intermediate results that are interactively corrected, e.g., by dragging or bulging. In particular, in this use case, a fast update of the surface mesh is crucial.

Surface rendering of volume data was primarily motivated by two issues: it was much faster than volume rendering, since a mesh has a significantly lower memory footprint than a volume dataset and it provides clearly recognizable images with depth cues such as illumination. These original advantages have somehow lost their relevance when GPU rendering became more and more powerful and could be used to support all stages of direct volume rendering. Thus, even advanced volume rendering of typical medical datasets can be performed in real-time. However, surface rendering is available in any radiologic workstation and in most therapy planning systems, since users are familiar with this display mode. In the last years, three developments occurred that raised the importance of surface extraction and surface rendering:

- With the increased use of biophysical simulation, there is a need for surface meshes as a basis for volume grids that are used for simulations.
- Interactive 3D visualizations are used in web browsers and mobile settings. Although direct volume rendering is also feasible in such settings, surface renderings are prevailing due to the lower memory requirements.
- Surface mesh construction becomes increasingly important for 3D printing. For challenging treatment planning questions, relevant portions of the human anatomy are modeled and printed in 3D to enable an in-depth collaborative discussion of treatment options.

In this chapter, we focus on general requirements and solutions for surface rendering. We include the use of surface rendering in web browsers and in mobile settings. Mesh simplification is more essential in mobile use to cope with bandwidth problems. We postpone the additional requirements related to surface meshes used as input for volume grid generation and simulation to Chapter 19.

Organization We start this chapter with a brief discussion of the surface extraction process. In particular, we explain typical problems when contours in volume data are transformed into surfaces (§ 6.2). In this section, we also explain widespread data structures to represent surfaces. We go on and explain Marching Cubes—the most essential algorithm to carry out this transformation (§ 6.3). This section includes improvements with respect to quality and performance. In § 6.4, we describe how the methods described before can actually be used for generating high-quality surface renderings in unsegmented data. This includes preprocessing to reduce noise, adequate support in the selection of isovalues and the simultaneous visualization of several isosurfaces.

Visual Computing for Medicine, Second Edition. http://dx.doi.org/10.1016/B978-0-12-415873-3.00006-7

In advanced therapy planning, such as radiation treatment planning, the target objects are segmented. A relevant problem is thus the transformation of segmentation results into surface meshes (§ 6.5). In particular, binary segmentation results cause severe aliasing artifacts when they are visualized in a straightforward manner. Since the human visual system gets strongly distracted by aliasing artifacts, we have to discuss mesh smoothing and explain a number of methods that differ in computational effort, accuracy, and visual quality. We extend this discussion by introducing advanced mesh smoothing methods (§ 6.6). They are tailored to specific artifacts and consider various constraints to provide better trade-offs between accuracy and smoothness. Finally, we discuss surface rendering for mobile and web-based therapy planning and training (§ 6.7).

6.2 RECONSTRUCTION OF SURFACES FROM CONTOURS

In this section, we discuss the topology of surfaces, neighborhood relations as a special aspect of topology, and finally data structures that represent surface meshes.

6.2.1 TOPOLOGICAL PROBLEMS

The topology of surfaces is characterized by the number of connected components and by the so-called *genus*. The genus of a surface is the maximum number of non-intersecting cuts that leave the surface connected [Wood *et al.*, 2004]. A torus, for example, has genus 1, whereas a sphere has genus 0. Most anatomical surfaces, e.g., the brain, have a genus of 0. Thus, the computation of the genus in an extracted surface may reveal *topological noise*—excessive topology that needs to be detected and removed.

Segmentation often results in contours in the individual slices. The determination of surfaces from such contours is challenging, since three problems have to be solved [Meyers *et al.*, 1992]:

- the *correspondence problem*. Which of a potential number of contours in slice n corresponds to a contour in slice $n + 1$?
- the *tiling problem*. How should the vertices of contour C_n in slice n be connected to contour C_{n+1} in slice $n + 1$?
- the *branching problem*. If a contour C_n in slice n corresponds to C_{n+1_a} and C_{n+1_b} in slice $n + 1$, the question arises, where the contour actually splits.

Figure 6.1 shows several possible solutions of the correspondence problem in case that the number of contours in adjacent slices is different (contours may split or merge). While the correspondence problem is related to the branching problem, it may also occur in isolation, in particular if the contour curves change strongly between adjacent slices.

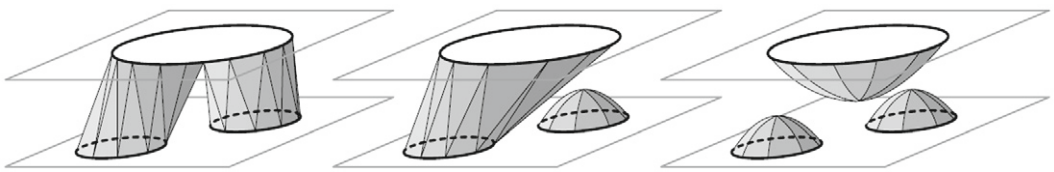

FIGURE 6.1 *Three possible solutions to the correspondence problem in case of a topology change (a bifurcation) (Courtesy of Ragnar Bade, University of Magdeburg).*

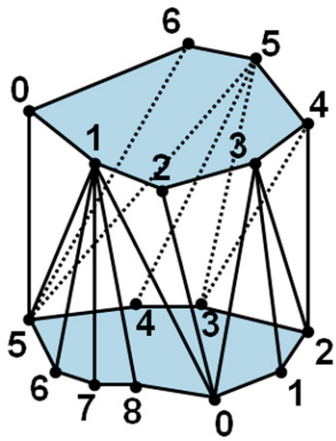

FIGURE 6.2 *The contour in slice n has seven vertices, whereas the contour in slice n + 1 has nine vertices. There are many possible triangulations. Favorable triangulations lead to compact objects with a low surface area and a high volume (Courtesy of Ragnar Bade, University of Magdeburg).*

The tiling problem is illustrated in Figure 6.2. Several authors discussed optimal triangulations between contours in adjacent slices. These include:

- maximization of the volume,
- minimization of the surface,
- preference for a connection that moves the barycenter in slice n close to the barycenter of slice $n + 1$.

Efficient solutions for this problem are based on search processes in graphs [Keppel, 1975, Fuchs *et al.*, 1977]. Thus, a *cost function*, a weighted combination of the three criteria mentioned above, is used to search for a path through all vertices of the two contours with minimum costs. We already discussed such minimum path search in image segmentation (recall § 4.4.1).

The topology of a surface may be well represented by the *reeb graph* [Carr *et al.*, 2000]. To construct this graph, an axis-aligned sweep through the volume is performed to determine the contours in each slice. Each contour in one slice represents a node in the reeb graph and edges represent the connection between contours in subsequent slices. Nodes with more than two neighbors reflect a branching. Cycles in the reeb graph indicate a structure with holes. Algorithms to clean the topology of surfaces frequently employ that graph. In a valid triangle mesh, no triangles overlap. Also there are no T-junctions or holes in a regular mesh.

6.2.2 NEIGHBORHOOD RELATIONS IN SURFACE MESHES

Most algorithms that process surface meshes operate on a local neighborhood of each vertex, e.g., to smooth or simplify the mesh. This local behavior enables efficient solutions. In the following, we employ V as a set of vertices of a surface mesh. There are primarily two different terms of neighborhood, e.g.:

- *topological neighborhood*, and
- *Euclidean neighborhood*.

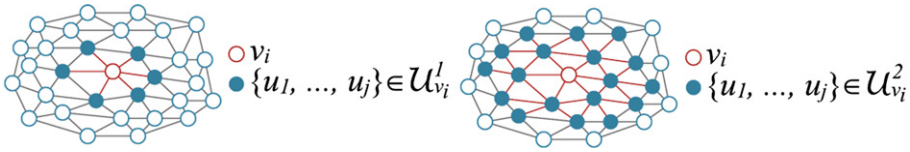

FIGURE 6.3 *Topological neighborhood in triangular meshes.* **Left:** *The 1-ring neighborhood (contains all vertices u_j directly connected with an edge to the current vertex v_i).* **Right:** *The 2-ring adds all vertices that are directly connected to the vertices of the 1-ring (Courtesy of Ragnar Bade, University of Magdeburg).*

The topological neighborhood of vertex $v_i \in V$ comprises all vertices $u_j \in V$ that are connected to v_i through a path of a certain length. The topological 1-neighborhood comprises exactly all u_i that share an edge with v_i, thus the connecting path has the length of 1. Similarly, the topological 2-neighborhood contains all vertices connected to v_i with a path consisting of one or two edges (Fig. 6.3). These neighborhoods are often shortly referred to as the 1-ring or 2-ring. The topological 2-neighborhood has significantly more vertices. Thus, even in a data structure that enables an efficient access to all adjacent vertices, any computation that involves this neighborhood is more compute-intense.

The *Euclidean neighborhood* of vertex v_i comprises all vertices $u_j \in V$ that have an (Euclidean) distance $\leq d$ to v_i, where d is a distance threshold. While the topological neighborhood may contain vertices that are rather distant, the Euclidean neighborhood has a circular (in 2D) or spherical (in 3D) shape, effectively limiting the distance. In practice, the 1-ring has a couple of advantages for many applications. The neighboring vertices u_j may be determined faster (without distance computations) and it always contains a certain number of elements that are usually needed, e.g., for robust mesh smoothing. However, Euclidean distances are also employed for some surface rendering problems in medicine, e.g., in vessel visualization with implicit surfaces (§ 11.6.2).

6.2.3 REPRESENTATION OF SURFACE MESHES

There are various data structures that represent surface meshes. We briefly describe a lean and compact data structure that is often used for rendering surface meshes. Moreover, we describe data structures that provide more information related to the topology of a triangle mesh and thus more direct access to information needed to simplify, smooth, or otherwise process a triangle mesh.

Shared Vertex Representations If surface meshes should only be rendered and not processed in any further way, a lean data structure with low memory consumption is often preferred. Shared vertex representations are optimal for this purpose.[1] It consists of a *vertex list* that comprises the coordinates of all vertices and of an *indexed face list*. The face list contains pointers to the vertex list, e.g., an entry (5, 6, 8) specifies that a polygon has the vertices with the indices 5, 6, and 8 in the vertex list. This indirection has several advantages over a direct list with all vertices of all polygons. In a regular triangle mesh, each vertex belongs to six triangles. Thus, the coordinates (3 float values with 4 bytes each) need to be redundantly stored six times, instead of one integer representing the index. Moreover, the indirect storage scheme explicitly represents that a certain vertex belongs to six triangles. If, for example, this vertex is modified, only one 3D vector, representing the coordinates, needs to be updated. If the mesh only consists of triangles, each triple of indices represents a triangle. In the more general form, an indexed face set may contain mixed polygons, e.g., triangles and quadrilaterals. In this case, a special entry, often -1, is used to indicate the

1 These data structures are also known as *IndexedFaceSet*, a name that corresponds to an Open Inventor class.

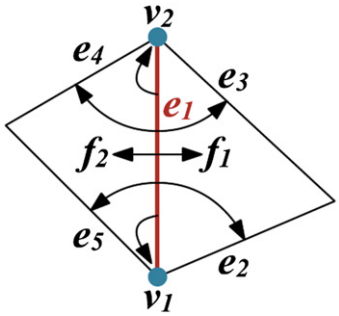

FIGURE 6.4 *Winged-edge data structure: an edge e_1 has pointers to its two vertices (v_1, v_2), to edges that share these vertices (e_2, e_3, e_4, e_5), and to the two faces that contain e_1 (Courtesy of Ragnar Bade, University of Magdeburg).*

separation between different polygon descriptions. Widespread toolkits for medical visualization, such as OpenInventor, provide classes for storing meshes in indexed face lists.

For many operations performed on polygonal meshes, an indexed face set does not provide direct support. As an example, it is often necessary to determine the two faces that share an edge in a mesh. In an indexed face set, this requires a search through the whole data structure. It would be more convenient, if such neighborhood relations are directly stored. Ideally, any adjacency relations between faces, edges, and vertices are explicitly represented. Since this would require compute-intense preprocessing and result in a large data structure, different subsets of these relations are represented in data structures, such as the winged-edge data structure [Baumgart, 1975] and the half-edge structure [Botsch *et al.*, 2002].

Winged-edge Data Structure The winged-edge data structure is centered on *edges*. In this data structure, each edge has a pointer to the two vertices and to the two adjacent faces. Moreover, an edge as part of an edge list has a pointer to its successor and its predecessor (see Fig. 6.4). For algorithms that iterate over vertices of a mesh and modify them according to their neighborhood (recall § 6.2.2), this data structure is not ideal, since neighbors of vertices cannot be directly accessed.

Half-edge Data Structure The half-edge structure splits each edge into two edges with the same coordinates but opposite direction. Each half-edge points to its opposite half-edge, one vertex (the so-called target vertex), the opposite and the previous half-edge, and the incident polygon. Half-edge-based structures are provided by CGAL[2] and OpenMesh.[3] For algorithms that require the insertion or deletion of elements, double-linked lists are provided, whereas array representations are favorable for algorithms that only modify existing mesh elements, since the access to array elements is faster. Thus, the algorithms that should be applied determine an appropriate data structure. Botsch *et al.* [2010] provide a good overview on mesh processing and related data structures.

6.3 MARCHING CUBES

Among the many available algorithms for the extraction of isosurfaces from volumetric image data [Hansen and Johnson, 2004] there is one classical algorithm: the Marching Cubes [Lorensen and Cline, 1987]. Essentially, the Marching Cubes algorithm examines each individual volume cell and generates a triangulation in case the isosurface intersects the cell. Thus, the process of extracting an isosurface from a

2 http://www.cgal.org.
3 http://www.openmesh.org.

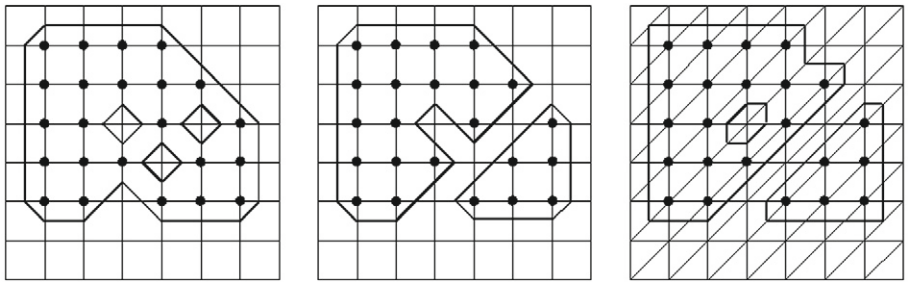

FIGURE 6.5 *Possible contours of a set of pixels with three ambiguous cells. From left to right: Marching Square contouring with case 4; contouring with the complementary case 4a; contouring after triangulation (Marching Triangles). All results are valid but they are topologically different, i.e., the number of holes and components is different (Courtesy of Dirk Bartz).*

volume is decomposed in individual cells and treated locally. A cell consists of four sample points in one plane and four in an adjacent plane. In a dataset with $N \times M \times O$ elements, there are $N-1 \times M-1 \times O-1$ cells to consider. The final surface is composed of the local contributions. The major innovation of Marching Cubes was the use of a lookup table—the case table—for every possible triangulation. This enabled a significantly faster triangulation of the specified isosurface. In the following, we will call the grid connections between pixels (and voxels in 3D) *cell edges*. Marching Cubes assumes that a contour is passing through a cell edge between two neighboring voxels with different states exactly once. Based on that assumption, it generates a contour crossing through the respective cell edges.

Since Marching Cubes operates locally, inconsistencies of the triangulation resulting in holes may arise. In the following, we will first discuss the 2D version of Marching Cubes, the *Marching Squares* algorithm. Afterwards, we continue with Marching Cubes itself.

6.3.1 MARCHING SQUARES

The Marching Squares algorithm computes isolines from 2D images [Schroeder *et al.*, 2001]. In essence, Marching Squares examines all image cells of four pixels. For each cell, the pixels (or corners) are examined if they are greater or equal the isovalue or threshold. If that is the case, the respective pixel states are set or reset otherwise. In total, this generates $2^4(=16)$ possible combinations of set or reset pixels.

If we interpret the pixel states as digits of a number, this number may serve as index to a table that enumerates all 16 cases for the respective contour lines. Most of these cases are very similar. With rotations and inversion of the set/reset pixel states we can reduce the number of cases to five possible configurations.

Unfortunately, two cases have an ambiguous solution that cannot be decided without further information. In these cases, two pixel states are set and two are reset, and the two set pixels are diagonally located. Both set pixels can either be connected, or separated. Figure 6.5 (left, middle) shows the impact of the chosen case on the resulting contour. One remedy to avoid the ambiguity is to use a different cell type, e.g., a triangulation, which would generate a unique contour (Fig. 6.5, right). However, the transformation of quadrilateral cells to triangles is not unique. Thus, the ambiguity is just translated.

6.3.2 BASIC ALGORITHM

We will now look into the Marching Cubes algorithm [Lorensen and Cline, 1987]. The Marching Cubes algorithm solves five tasks to extract a surface from volume data:

1. determination of the case index of each cell,
2. determination of the intersected edges,

3 computation of intersections by means of linear interpolation,
4 triangulation of the intersections, and
5 computation of outward-pointing surface normals for illumination.

Marching Cubes processes each volume cell independently.

Determination of the Case Index and Intersected Edges The eight voxels of the volume cell have an assigned state, indicating whether their respective voxel value is greater, equal, or smaller than the specified isovalue τ. The state of each voxel is interpreted as a binary digit (1 for inside, and 0 for outside) and composed into an eight-bit number $index$. Figure 6.6 (left) shows which voxel state goes to which position.

After the case index for the cell is determined, the cell edges that are intersected by the isosurface can be looked up in the case table. In Figure 6.6 (right), six cell edges are intersected by the isosurface.

As an example, in case 9 ($9 = 1 * 2^0 + 1 * 2^3$), the states of voxel V_0 and voxel V_3 are set. We now have 256 possible configurations of voxel states, and hence we have 256 possible triangulations of a volume cell. However, not all of these configurations generate different triangulations. Most cases can be sorted into 15 equivalence classes,[4] taking into account rotation or mirroring (on a plane). Figure 6.7 illustrates all 15 classes, as listed by Lorensen and Cline [1987]. This reduction was more essential when Marching Cubes was invented than it is today: It enables to store the state of two cells in the eight bits of a byte, effectively reducing the storage consumption of the case table. Figure 6.7 also reveals that at most four triangles are generated as local contribution of a cell.

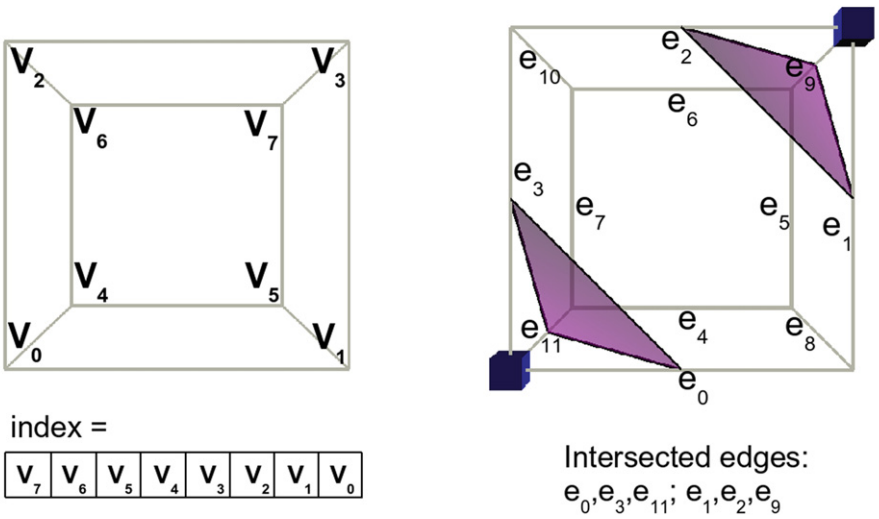

FIGURE 6.6 *Voxel and edge indexing of the Marching Cubes algorithm.* ***Left:*** *To compute the case table index, the voxel indices compose an eight-bit index, depending on if they are set or reset.* ***Right:*** *The case table contains a list of the intersected edges. In this example, edges e_0, e_3, e_{11} and edges e_1, e_2, e_9 are intersected. The blue cubes at the voxel positions indicate a set voxel state (Courtesy of Dirk Bartz).*

4 Some papers or lecture notes mention only 14 equivalence classes, but there are really 15 classes. Since only 14 classes actually contain a part of the isosurface, this might be the cause of confusion.

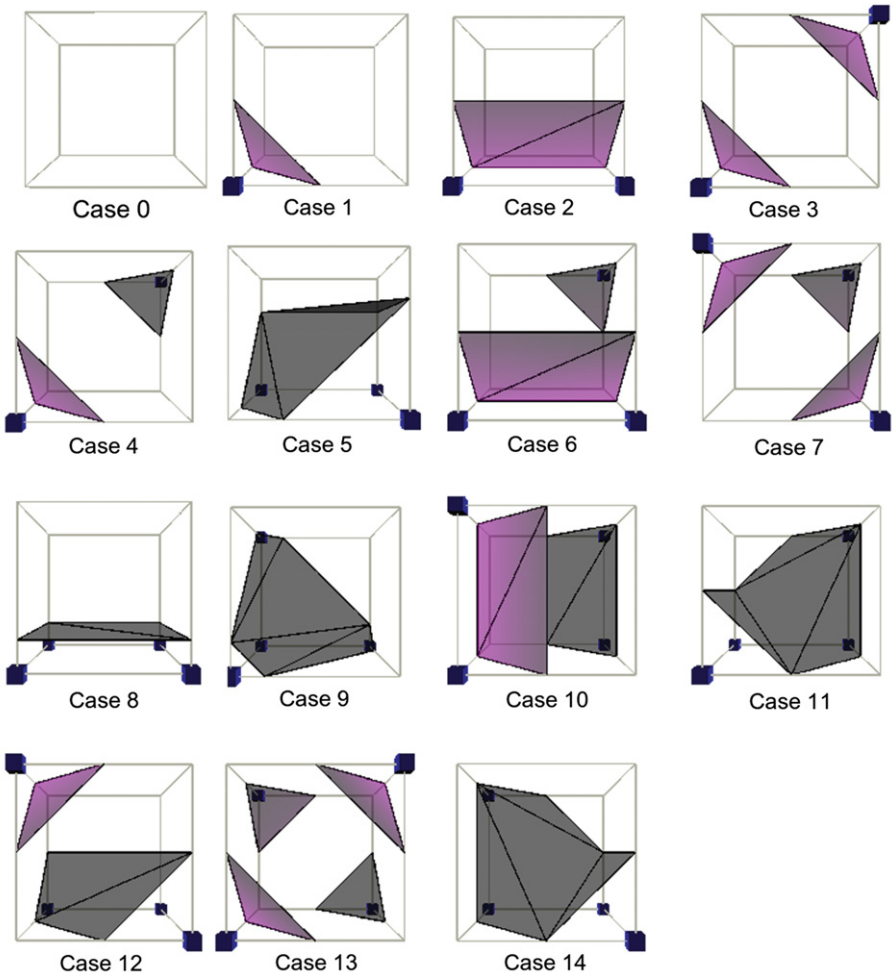

FIGURE 6.7 *Fifteen equivalence class cases of Marching Cubes. Not that the case numbers here do not represent bit states. Magenta triangles are front-facing, gray triangles are back-facing. The blue cubes at the voxel positions indicate a set voxel state (all other voxels have a reset state) (Courtesy of Dirk Bartz, University of Leipzig).*

Computation of Intersections Once the intersected cell edges are determined, the intersection itself is calculated by linear interpolation. The interpolation parameter t is calculated based on the isovalue τ and the voxel values V_j and V_{j+1} on both sides of the edge according to Equation 6.1. If we assume that X_j and X_{j+1} describe the coordinates of the respective voxels, we can compute the respective intersection X_e with the linear interpolation according to Equation 6.2.

$$t = \frac{\tau - V_j}{V_{j+1} - V_j} \tag{6.1}$$

$$X_e = X_j + t \cdot (X_{j+1} - X_j) \tag{6.2}$$

If the isovalue τ is closer to the voxel value V_{j+1}, the respective edge vertex moves closer to that voxel. Respectively, the edge vertex moves closer to the left voxel, if τ is closer to the voxel value V_j. If t is exactly zero or one, the surface would pass through a vertex of the cell. This causes problems in the subsequent triangulation, where a degenerated triangle would arise. Thus, t is often slightly modified to yield a valid triangulation.

Triangulation of the Intersections If we combine the interpolated intersections with the information how the edge intersections are composed into triangles—which is also stored in the case table—we can now determine the triangles.

Computation of Outward-pointing Surface Normals Finally, the surface normals need to be computed to enable a shaded visualization that clearly depicts the corresponding shape. Usually, surface normals are determined based on the orientation of the triangle. However, in the context of surface extraction, we might approximate the gradients of the scalar field by computing intensity differences. The use of the original data of the scalar field leads to a more precise definition of surface normals. This computation is performed for the three vertices of a triangle and as a simple solution it is averaged. The resulting gradient needs to be normalized for the use in an illumination model. The very same strategy is used for the integration of illumination in direct volume rendering.

6.3.3 DISCUSSION

In the following, we discuss some limitations of Marching Cubes and potential improvements. Marching Cubes, like any surface extraction method, is based on the *binary* decision whether portions of a cell belong to the surface or not. Thus, isosurfaces are appropriate for distinct surfaces, such as teeth, skin, and bones extracted from CT data (see Fig. 6.8). However, they are misleading if generated from noisy and inhomogeneous data or when the underlying phenomena are amorph with more fuzzy boundaries.

Topology Problems First, we discuss Marching Cubes in relation to the three problems explained in the transformation from contours to slices (recall § 6.2). Marching Cubes does not solve the *correspondence*

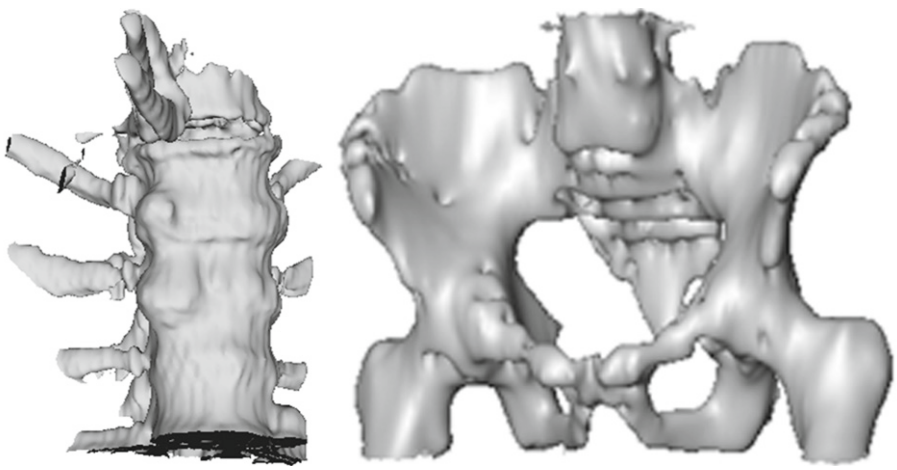

FIGURE 6.8 *Isosurfaces extracted from CT data with Marching Cubes.* ***Left:*** *A portion of the spine in the breast region.* ***Right:*** *Pelvis extracted from CT data (Courtesy of Wolf Spindler, Fraunhofer MEVIS, and Ragnar Bade, University of Magdeburg).*

FIGURE 6.9 *A Marching Cubes result of the segmentation of an elongated, partially strongly curved blood vessel (V. Jugularis). The contours in adjacent slices do not always overlap. Thus, Marching Cubes cannot generate a connected surface. The dataset has a rather low resolution between the slices (3 mm), compared to an in-plane resolution of 0.5 mm.*

problem. The contributions from adjacent slices to a surface are only connected if they overlap in these slices. This is very likely, in particular for compact large objects and for data with a low distance between the slices. However, if both requirements are not fulfilled, disconnected surface components arise, as Figure 6.9 illustrates.

The solution of the *tiling problem* is acceptable but not optimal. Any optimal solution to the correspondence problem and the tiling problem requires to consider the whole contour in two adjacent slices instead of only their local contributions in individual cells. Thus, Marching Cubes provides a trade-off between quality and speed and performs much faster than any of such optimization methods. The branching problem is not considered explicitly. This, however, is the least problem, since there are hardly any general assumptions to tackle this problem in a better way.

Accuracy and Smoothness The Marching Cubes computation of the surface is a reasonable trade-off between accuracy and speed. Instead of subdividing each cell and accurately determining how the surface proceeds inside, only the intersections of the surface with the cell edges are evaluated. Isosurfaces are C^0-continuous approximations that reflect the bendings of curved surfaces, such as anatomical shapes, only to a limited extent.

Cache Coherence After the computation of the triangulation within one volume cell, the algorithm *marches* to the next volume cell. Since voxels are typically indexed first in x, then in y and z, the next cell has an incremented x-position. However, this is often not the optimal addressing scheme. As in most algorithms for large volumetric datasets, only a part of the dataset can be stored in the caches of the CPU. Hence, an implementation of the Marching Cubes algorithm will benefit enormously if it adopts a *cache-sensitive* processing that takes into account how the voxels are organized in memory.

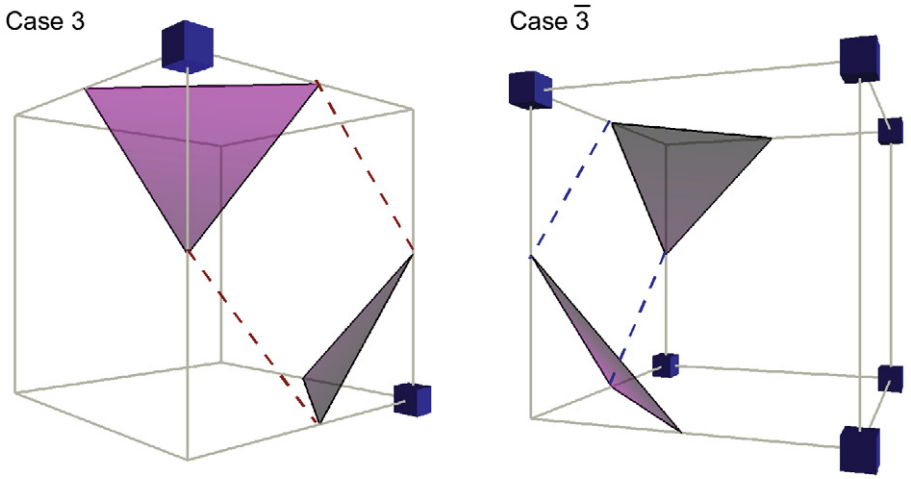

FIGURE 6.10 *Possible Marching Cubes triangulations for two neighboring volume cells. The left cell is triangulated to separate set voxels (case 3), the right neighboring cell is triangulated to connect set voxels (case $\overline{3}$). Both are valid triangulations, but they generate an inconsistent triangle interface with holes. The dashed lines indicate the triangle edges of the respective other cell (Courtesy of Dirk Bartz, University of Leipzig).*

Ambiguities in 3D Similar to the 2D case, Marching Cubes suffers from ambiguities. Depending on how intersections are triangulated, connected, or separated surfaces arise. The ambiguities may also cause inconsistencies, such as holes (see Fig. 6.10). Here, the very assumption that the isosurface extraction can be performed locally breaks down and the solution is not only ambiguous, as in the 2D case, but wrong (true isosurfaces are connected as long as they do not cross the border of the dataset). This problem (and others) of the Marching Cubes algorithm has been detected quite rapidly [Dürst, 1988] and a number of remedies have been suggested. An interesting idea from Wu and Sullivan [2003] is based on user input: the user specifies *connectivity priorities* and thus indicates whether the foreground structure or the background structure should be connected, if possible.

Asymptotic Decider Nielson and Hamann [1991] suggested to interpolate a point on the face that is shared by both cells to provide information if the set voxels should be connected or separated. They exploit the fact that the contours on that face are hyperbolic and reconstruct the face interpolation point (or bilinear saddle point) with the asymptotes of the hyperbolas.

Case Table Refinement An alternative solution to the computation of additional interpolation points is the refinement of the original Marching Cubes case table. Chernyaev [1995] modified the case table to 33 cases to address the above-mentioned face and cell ambiguities. A similar approach was proposed by Cignoni *et al.* [2000], where the triangulation is progressively refined to address ambiguities. The probably most widely used solution is implemented in VTK [Schroeder *et al.*, 2001], where the full 255-case table[5] is used to avoid inconsistencies.

Marching Tetrahedra Similar to marching triangles in 2D, Shirley and Tuchman [1990] proposed the use of tetrahedra as cell type. In order to generate the respective tessellation of the Cartesian grid dataset, every

5 Only case 0 (no set voxel) and case 255 (eight set voxels) are combined into one case that does not contain an isosurface.

cuboid cell is decomposed in five tetrahedra with a consistent choice of primary and secondary diagonals. The new tetrahedron-based case table contains only four cases that do not introduce any ambiguity. On the downside, the number of generated triangles is approximately twice as large as for the Marching Cubes algorithm and the ambiguity is again translated to the step where the tetrahedrons are generated.

6.3.4 ADVANCED SURFACE EXTRACTION METHODS

Efficient Isosurface Extraction One problem of Marching Cubes is that considerable effort is spent on cells that do not contribute to the isosurface (often more than 90% of all cells). Thus, for more than 90% of the cells, either all vertices are inside the surface or all vertices are outside. This problem was aggravated in the last decades with a higher spatial resolution, further decreasing the portion of cells that contribute to an isosurface. As a remedy, hierarchical data structures might be employed to decide at a coarser level, which regions of the dataset can be excluded from detailed processing.

An *octree* is a hierarchical data structure that subdivides a volume dataset in all three dimensions in half, resulting in eight child blocks, the *octants*. The subdivision is continued until one octant represents one volume cell of eight voxels. If the volume dataset has a resolution of a power of two in each dimension, e.g., $512 \times 512 \times 256$, this process can be applied in a straightforward manner. Otherwise, a special treatment is needed.

A *kd-tree* is a slightly different data structure. Instead of subdividing the volume dataset in all three dimensions at the same time, only one dimension is chosen, generating a binary partition at every subdivision step. Note that the orientation of subdivision plane can be changed step by step. A similar data structure is the *BSP tree* (Binary Space Partitioning Tree), which uses arbitrary oblique subdivision planes.

Octrees, kd-trees, and BSP trees are hierarchical and recursive tree structures, where the root node represents the whole dataset. The depth of the tree is determined by the maximum subdivision level, which is usually the cell level where a block (or node) is represented by a volume cell of eight voxels.

All these data structures may be employed for accelerating isosurface extraction. Per node the minimum and maximum intensity value that occurs within the cells is represented. Therefore, such trees are called *MinMax trees*. For a particular isovalue i, the tree is traversed in a top-down manner and only nodes where i is between min and max need to be further considered. The traversal is performed until the level of the elementary cells that are treated like in the Marching Cubes algorithm. This very idea was introduced by Wilhelms and van Gelder [1992] using octrees. This algorithm was widely used, e.g., for multiresolution representation of isosurfaces and refined, e.g., in [Livnat *et al.*, 1996], where kd-trees have been employed.

Advanced topic: Precise marching cubes. The Marching Cubes algorithm does not deliver the isosurface with maximum accuracy. Trilinear interpolation with sampling points inside each cell delivers better results. However, the computational effort is prohibitive if this interpolation scheme would be applied to the whole dataset with a fine subdivision of each cell. Allamandri *et al.* [1998] and Cignoni *et al.* [2000] developed an algorithm that adaptively refines the surface representation in cases where the contours in adjacent slices change strongly. The resulting surfaces are at the same time more accurate and smoother. However, the computational effort, even with the adaptive scheme is large. For each refined cell, it is two orders of magnitude slower compared to Marching Cubes. The higher accuracy is also related to a larger number of triangles per surface mesh. Thus, not only surface extraction but also rendering is slower.

Advanced topic: Fuzzy isosurfaces. It may be desirable to create isosurfaces for structures with fuzzy surfaces. Instead of a precise threshold, an interval $[i_1; i_2]$ is defined to control the surface extraction. The surface corresponding to $(i_1+i_2)/2$ will be rendered opaque. Surfaces corresponding to values i in $[i_1, i_2]$ are rendered semitransparently, where the transparency increases according to $(i_1+i_2)/2-i$. This method was introduced by VOXAR in the 1990s.[6] It may also be used in combination with segmented medical image data, in particular if fuzzy segmentation or classification are used.

6.3.5 HARDWARE-ACCELERATED ISOSURFACE EXTRACTION

It would be highly desirable to perform surface extraction fast, even for larger datasets, so that the user can interactively modify an isovalue and can immediately see the resulting surface as feedback. Even with hierarchical data structures, this is difficult to achieve, since these data structures have to be rebuild at least partially when the isovalue is changed. There have been various attempts to perform Marching Cubes on the GPU, e.g., Pascucci [2004], Kipfer and Westermann [2005]. Unfortunately, these attempts suffered from the limited flexibility of the GPU at that time. There was no possibility to create geometry simply with fragment and vertex shaders. Thus, the number of triangles needs to be predetermined and a lot of unnecessary degenerated triangles were generated and had to be removed later. With the advent of shader model 3 and the geometry shader, more efficient isosurface extraction on the hardware was feasible.

As an example, we briefly describe the method by Dyken *et al.* [2008]. The cells of the 3D space are processed independently and in parallel to yield the output stream. It is based on a hierarchical data structure, referred to as *histogram pyramid*. The histogram pyramid is a stack of hierarchical 2D textures where the *base texture* has the full resolution and each element in the upper levels integrates 2×2 texels by summing up their values (in contrast to a mipmap where upper levels contain the average). The entries in the histogram pyramid represent the number of output elements to be generated, that is the number of triangles (a cell in 2D space contains up to two triangles). Thus, exactly the required geometry is created. For the sake of brevity, we refer the reader to [Dyken *et al.*, 2008] and restrict to a brief discussion of the results. The authors carried out a performance analysis, among others with medical data, e.g., a CT head dataset, and two MR brain datasets where the brain and the cerebral vasculature were extracted. The datasets had a resolution of $256 \times 256 \times 256$ each and the test was performed with various graphics cards. With the best GPU, frame rates of 53, 37, and 55 were observed. The differences result from the number of relevant cells contributing to the isosurfaces. A CUDA implementation was also provided and was only slightly slower (43, 34, 31 frames per second). Thus, even with a higher number of slices, as they occur, e.g., in the thorax region, isosurface extraction may be performed in less than a second.

6.4 SURFACE RENDERING OF UNSEGMENTED VOLUME DATA

In this section, we discuss practical problems and solutions related to surface renderings. First, we emphasize that prior smoothing of the image data favors the extraction of smooth isosurfaces. We also discuss the selection of isovalues. Medical doctors should be supported in this process to avoid a lengthy interaction and sub-optimal results. In particular for treatment planning, often several isosurfaces need to be extracted and displayed simultaneously. We discuss both, the efficient extraction of such isosurfaces and their visualization.

6 Voxar is now a part of Toshiba Medical Visualization Systems, http://www.tmvse.com/.

6.4.1 PREPROCESSING VOLUME DATA FOR VISUALIZATION

At the image data level, there are primarily two kinds of techniques that potentially favor subsequent surface extraction. If data are strongly anisotropic, severe aliasing artifacts may occur in extracted surfaces. The interpolation of intermediate slices reduces these problems at the expense of an extended memory consumption. Noisy data, e.g., CT data with a rather low radiation and a high spatial resolution, benefits from smoothing of the volume data. In principle, both techniques may be combined. However, since they both affect accuracy, this has to be performed with care, e.g., by analyzing the resulting errors. In the following, we discuss these methods.

Resampling Volume Data Interpolation of intermediate slices aims at a reduction of anisotropy, or, even better, at the generation of an isotropic volume with equal resolution in all spatial directions. It may be performed with different methods, resulting in different quality but also computational effort. Linear interpolation is the simplest method, whereas cubic B-spline interpolations, Hermite and Lanczos interpolations are more advanced. To yield isotropic data, it is usually not sufficient to interpolate in between existing slices, but to resample the data completely. Such resampling has a certain smoothing effect and thus might slightly degrade accuracy. Figure 6.11 compares surfaces extracted from anisotropic and from resampled isotropic data, where cubic B-splines were employed as resampling filter.

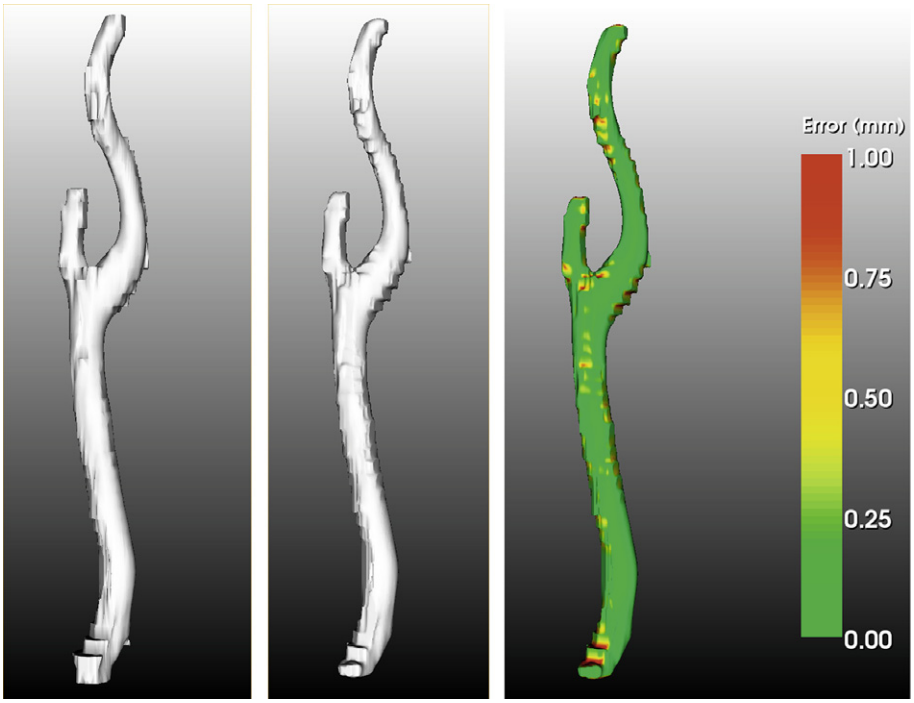

FIGURE 6.11 *Effect of the interpolation of intermediate slices on isosurface extraction.* ***Left:*** *The surface of the A. carotis is extracted from CT data with a typical resolution of* $0.453 \times 0.453 \times 3$ *mm.* ***Middle:*** *The data were resampled in the z-direction to an isotropic grid with cubic B-splines before surface extraction. Aliasing effects are slightly reduced but could not be removed.* ***Right:*** *The difference between both surfaces is analyzed and color-coded (Courtesy of Tobias Mönch, University of Magdeburg).*

Volume Smoothing We discussed volume smoothing in § 4.2.4 and learnt that there are simple methods that are independent from the actual data and replace intensity values based on a weighted average of surrounding values, such as the Gaussian filter. Advanced but also more compute-intensive algorithms work in an adaptive manner. They may preserve edges or adapt their behavior to other local properties of the data. We discussed some prominent examples, such as bilateral filtering and anisotropic diffusion. Figure 6.12 compares isosurface generation without smoothing, after smoothing with a small Gaussian filter kernel, and after smoothing with the non-linear anisotropic diffusion method. In Figure 6.13, the

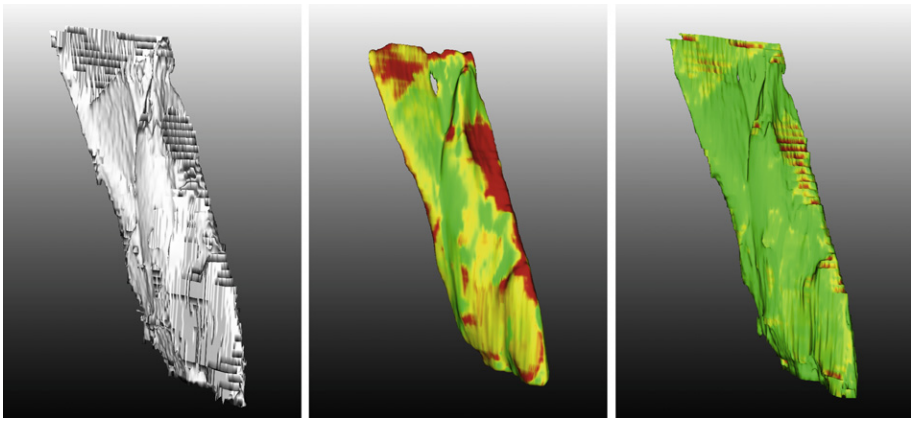

FIGURE 6.12 *Effect of volume smoothing on isosurface extraction. Color encodes error.* ***Left:*** *A noisy surface of a neck muscle extracted from the original data.* ***Middle:*** *Gaussian smoothing in a* $3 \times 3 \times 3$ *neighborhood prior to surface extraction leads to a surface with lower curvature.* ***Right:*** *Anisotropic diffusion reduces noise more effectively and thus leads to a further reduction in the curvature (Courtesy of Tobias Mönch, University of Magdeburg).*

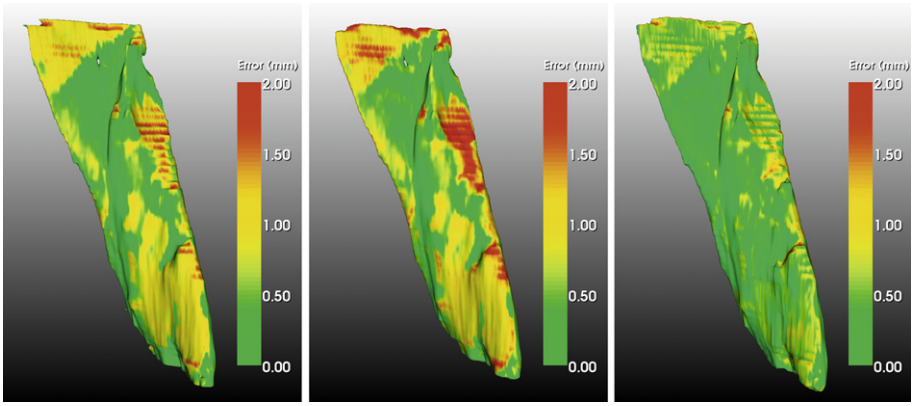

FIGURE 6.13 *Volume smoothing with larger filter kernels. The anisotropy in the data is considered by using anisotropic kernels.* ***Left:*** *Gaussian smoothing in a* $9 \times 9 \times 3$ *neighborhood and the error induced by smoothing.* ***Middle:*** *Gaussian smoothing in a* $9 \times 9 \times 5$ *neighborhood.* ***Right:*** *Anisotropic diffusion in a* $9 \times 9 \times 5$ *neighborhood with* $\sigma = 450$ *(Courtesy of Tobias Mönch, University of Magdeburg).*

same methods are employed but the kernels have a larger size. It becomes obvious that the diffusion filter preserves the original shape much better. In case of strongly anisotropic data with large distances between the slices, filter kernels should reflect that anisotropy, e.g., a $9 \times 9 \times 3$ kernel size is more appropriate than a $9 \times 9 \times 9$ kernel. As a consequence, moderate smoothing of CT and MRI data may be recommended as a preprocessing step prior to surface extraction. The specific choice of filters and parameters should consider the resolution of the data and the signal-to-noise ratio.

6.4.2 SELECTION OF ISOVALUES

Isosurface visualization requires the proper choice of an isovalue. The isovalue may be chosen in a purely interactive manner without any guidance. This is usually not optimal, since even in CT data with standardized Hounsfield units, the selection of an optimal isovalue might be difficult. The specific density of anatomical structures, such as bones or contrast-enhanced vessels, differs from patient to patient. At the very least, an effective isosurface extraction is required to perform the selection effectively (recall § 6.3.5).

Histogram as Support for Isovalue Selection As a first level of support, the histogram of the dataset may be displayed and labeled to enable the user to select a certain value. In many cases, users are interested in an isovalue that represents a peak, a local maximum in the histogram or a value slightly below that peak to ensure that a certain structure is included in the isosurface (see Fig. 6.14).

Previews for Isovalue Selection A further support may be provided by rendering a few thumbnail images and enable the user to select from them. This *gallery concept* is widespread in radiology workstations. It may be

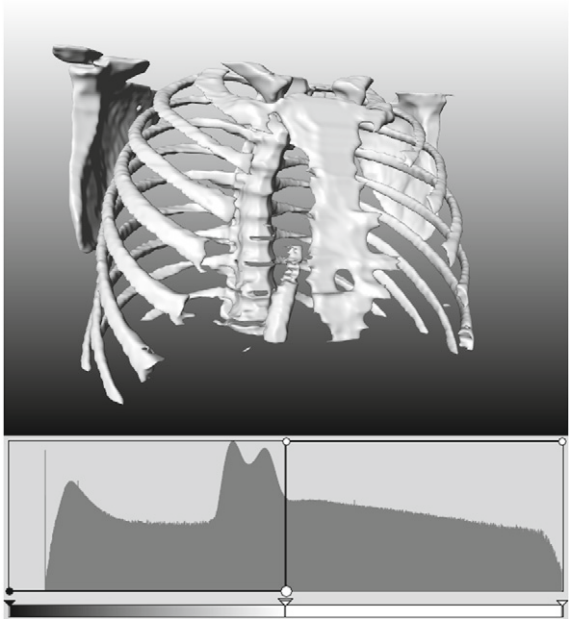

FIGURE 6.14 *A histogram of a CT thorax dataset is used to select isovalues for surface rendering. Here, a high isovalue (see the vertical line) is chosen to display skeletal structures. The first peak represents air, which fills the lungs. The other peaks represent lung tissue and other soft tissue structures (Courtesy of Tobias Mönch, University of Magdeburg).*

combined with some kind of intelligence. In a particular dataset, such as CT thorax data, there is a limited number of typical diagnostic goals for isosurface rendering, e.g., the bones only, or bones and soft tissue should be displayed. Thus, the thumbnails may employ a few isovalues and their presentation might be labeled with meaningful names.

6.4.3 MULTIPLE AND NESTED ISOSURFACES

Isosurface extraction approaches are often used to extract *one* surface. In particular for the visualization of multiple segmented anatomical structures, different objects need to be distinguished and rendered accordingly.

Extraction of Multiple Isosurfaces The brute force approach to extracting two (or more) isosurfaces is to apply Marching Cubes several times in sequence. There are, however, more efficient methods that visit each cell only once to decide whether they intersect one of the isosurfaces and to determine the related triangles in the same pass. As an Gerstner and Rumpf [1999] used a tetrahedral decomposition of hexahedral cells and built a hierarchical data structure to efficiently skip regions that do not contribute to any isosurface.

Visualization of Multiple Isosurfaces A basic distinction can be achieved by using different material parameters, e.g., different colors, for different objects (see Fig. 6.15). Unfortunately, some objects contain other objects and will hence occlude them. The optimal visualization of nested surfaces is a challenging issue. Obviously, a fully opaque rendering of the outer surface completely hides the internal surface. With an appropriate amount of transparency both are visible to a certain extent. Such a combined visualization enables an overview and thus an understanding of the location of inner surfaces within outer surfaces. Note that for the correct rendering all geometry primitives must be depth-sorted from back to front to allow the correct alpha blending in OpenGL or other graphics APIs (Application Programming Interfaces).

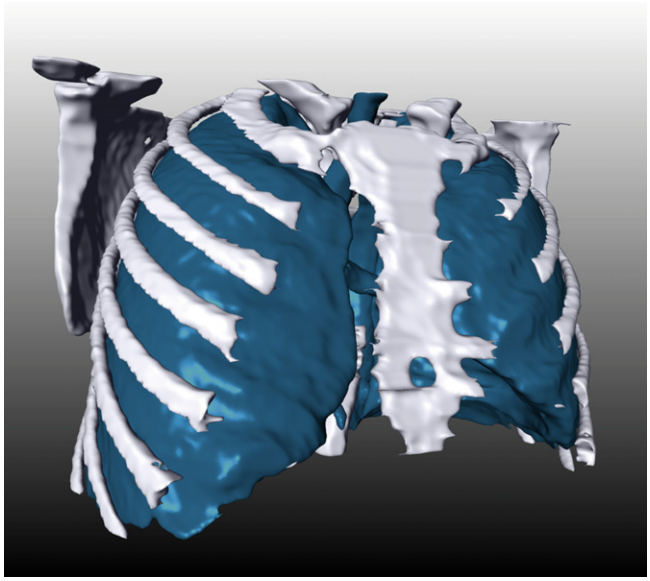

FIGURE 6.15 *Lung and ribs are displayed by means of two isovalues. Since the ribs do not occlude the lung strongly, both may be rendered opaque (Courtesy of Tobias Mönch, University of Magdeburg).*

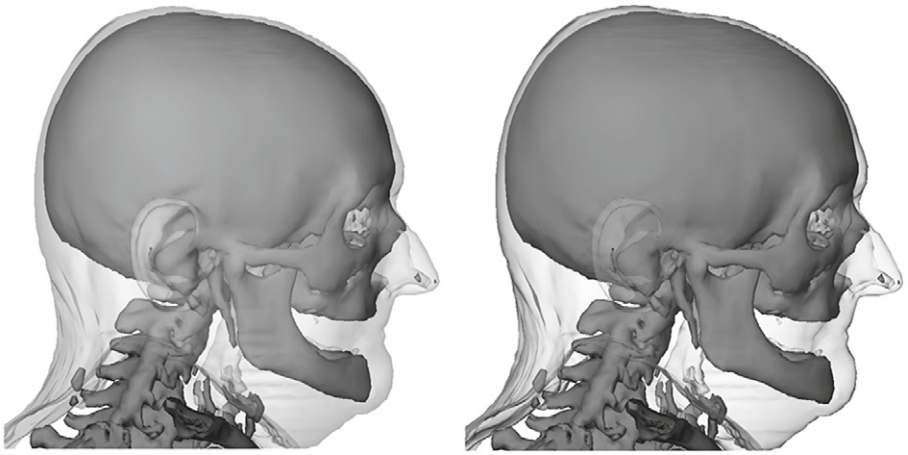

FIGURE 6.16 *Transparency-modulated Marching Cubes models [Stalling et al., 1999].* ***Left:*** *Constant transparency modulating increases the opacity for faces orthogonal to the viewing direction.* ***Right:*** *Oriented transparency modulating (Courtesy of Detlev Stalling, Zuse Institute Berlin).*

While this approach allows to see virtually all objects, it does not enable a good perception of the location of the internal object in relation to the outer object. In order to address this, the transparency can be adapted for each triangle. Stalling *et al.* [1999] presented such an approach where the transparency of a triangle is adapted based on its orientation toward the viewer. If it is parallel to the user, a small α-value is chosen, and a large α-value if the triangles are oriented more orthogonal to the user. This is a variation of a much older concept by Kay and Greenberg [1979], which assumes that a light ray will be more attenuated if it traverses a medium with a more acute angle. Figure 6.16 (left) shows an example of this technique, while Figure 6.16 (right) shows an example of constant transparency modulation. Overall, this transparency modulating results in an emphasis on the boundary of the semi-transparent objects.

6.4.4 ISOSURFACE TOPOLOGY SIMPLIFICATION

Connected Component Analysis A typical problem when isosurfaces are extracted from noisy image data, is that many small disconnected regions arise. Due to noise, the isovalue might be erroneously exceeded for just a few pixels. Often, the user is interested in one large connected component or at most a few components. Many disconnected components typically represent noise in the data. Thus, prior volume smoothing would lead to a strongly reduced number of components. To better support the user, surface extraction might be combined with a *connected component analysis* (CCA). This rather simple algorithm visits all cells of a dataset and starts a new component when a cell occurs that contributes to the isosurface. Other cells are labeled as background component. Once a new component is initialized, the CCA iteratively visits neighboring cells and labels these cells as parts of the initialized components if they also belong to the isosurface. That is basically the functionality of a region growing algorithm (recall § 4.3.4). In contrast to region growing, it does not stop after the first component is completed but searches for cells that are not labeled to find further components. The CCA stops when no more cells are unlabeled. In the CCA, the size of a component may easily be determined by incrementing a counter whenever a new cell is encountered that belongs to the component. Thus, the components may be sorted according to their size and, for example, to display only the largest component. Figure 6.17 illustrates how the CCA may be used to clean a visualization.

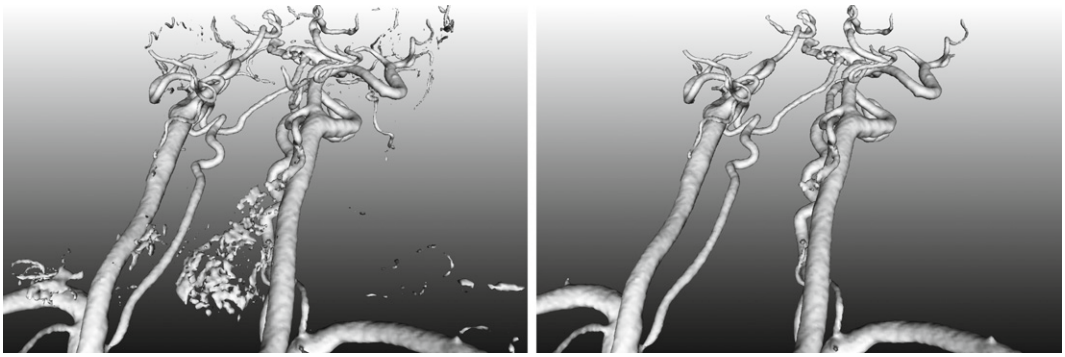

FIGURE 6.17 *An isosurface was created to display the neck and cerebral vasculature. As result of the noisy image data, many isolated components arise. With a reduction to the largest connected component, the relevant anatomy can be clearly displayed (Courtesy of Tobias Mönch, University of Magdeburg).*

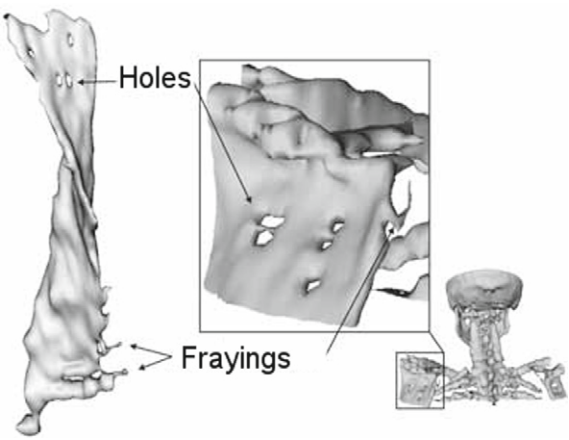

FIGURE 6.18 *Holes in a skeletal surface as a consequence of noisy data (Courtesy of Ragnar Bade, University of Magdeburg).*

Removing Excessive Topology Isosurface extraction frequently leads to holes. This might also happen in manually segmented data due to the lack of coherence in the slices. A number of methods have been developed that analyze the topology and simplify it if it is excessively complex. Figure 6.18 shows a medical surface model with erroneous holes in a muscle.

6.5 SURFACE RENDERING OF SEGMENTED VOLUME DATA

In the following, we consider the use of 3D binary segmentation results for isosurface visualization. The segmentation result is characterized by a *segmentation* or *label volume*. One value represents voxels belonging to the segmentation mask and a second value represents background voxels. Tomographic medical image data exhibit anisotropic voxels (slice distance is considerably larger than the in-plane resolution). A straightforward visualization of (binary) segmentation information represented in strongly anisotropic data leads to typical staircase artifacts and noisy surfaces (see Fig. 6.19, middle). For a correct and convenient perception

FIGURE 6.19 **Left:** *MR slice view.* **Middle:** *Due to the strong anisotropy, the Marching Cubes extraction yields a surface with strong edges.* **Right:** *A photo of that bone reveals that the edges represent artifacts from image acquisition and not features of the anatomy (Courtesy of Ragnar Bade, University of Magdeburg, photo: [Wikimedia Foundation Inc. 2005]).*

of shapes and spatial relations, the models should look naturally to resemble, e.g., the intraoperative experience of surgeons. The natural appearance refers to smoothness, since anatomical structures usually do not exhibit sharp edges. The problem can be tackled in two stages:

- at the image level, the segmentation information may be post-processed and smoothed, and
- the polygonal mesh may be processed and smoothed.

It is essential that these processes do not sacrifice accuracy. Thus, we consequently consider both improvements in smoothness and the side effects on accuracy.

6.5.1 PREPROCESSING

Also the surface visualization of segmented data benefits from preprocessing, similar to the preprocessing strategies discussed for the original data (recall § 6.4.1). The segmentation information may be used to enhance both resampling and volume smoothing.

Resampling Volume Data With segmented data, resampling may be strongly restricted to a ROI that just comprises the segmentation result. Thus, the resampling effort and the memory consumption may be strongly reduced. In addition, a more advanced interpolation technique may be applied. Instead of grey value interpolation, the shape of the contours representing the segmentation result in different slices should be interpolated. Thus, *shape-based interpolation* is frequently used [Raya and Udupa, 1990]. Shape-based interpolation takes two contours, represented as binary volumes, as input and generates a desired number of intermediate contours. Figure 6.20 compares surfaces extracted from segmentation in the original anisotropic data with surfaces extracted after shape-based interpolation. Please note that the aliasing effects are strongly reduced and the effect is stronger than by resampling that does not consider the shapes.

Smoothing Segmentation Results for Visualization Isosurfaces from segmentation results are created by using an intermediate value (between the value for the segmentation mask and background) as isovalue. As an

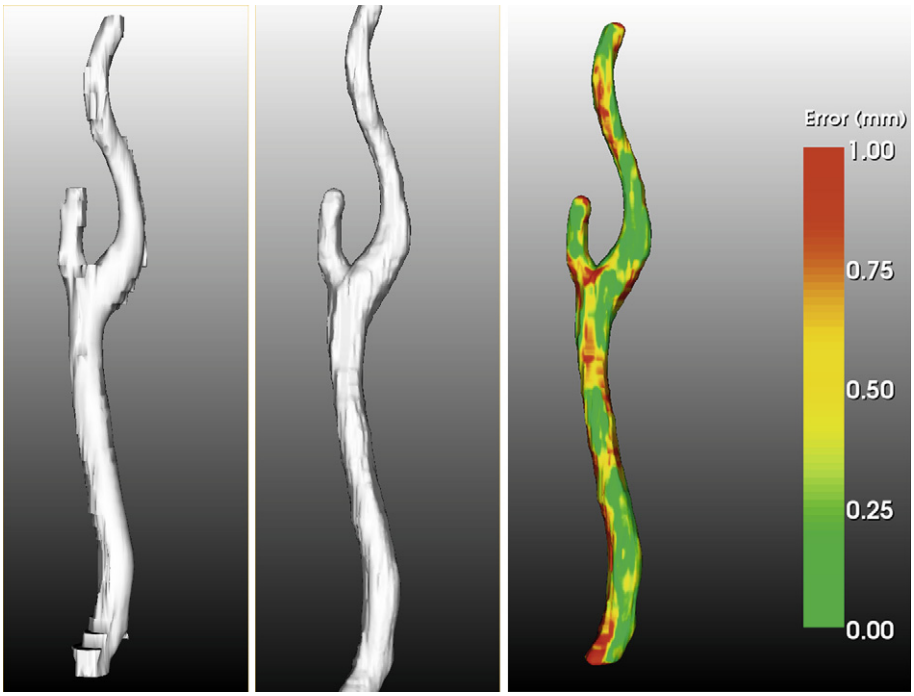

FIGURE 6.20 *Effect of shape-based interpolation on isosurface extraction.* **Left:** *The surface of the segmented A. carotis is extracted from CT data with a resolution of* $0.453 \times 0.453 \times 3$ *mm.* **Middle:** *Shape-based interpolation between the contours of the segmented A. carotis was applied prior to surface extraction. The data were resampled to an isotropic grid and cubic B-splines were applied for the interpolation.* **Right:** *The difference between both surfaces is analyzed and color-coded (Courtesy of Tobias Mönch, University of Magdeburg).*

example, 100 may denote voxels belonging to the segmentation mask, 0 represents background voxels, and 50 is used as isovalue. The simple isosurface generation suffers from strong aliasing artifacts, which become obvious as discontinuities in the surface normals. Therefore, it is useful to "smooth" the boundary in the segmentation result at the voxel level. Otherwise, intersections of an isosurface and a cell are always located at the center of an edge; never at any intermediate position. This explains why aliasing artifacts are significantly particularly severe when surfaces are extracted from *binary* volumes.

Fuzzy Boundaries Morphological operations may be employed to improve the appearance of visualizations that are based on binary segmentation results. In essence, they provide *fuzzy boundaries*. This is achieved by transforming the narrow region around the segmentation boundary such that the intensity values vary rather smoothly from background to foreground. This very idea was introduced by Lakare and Kaufman [2003]. They suggested to use the corresponding voxels in the original data to provide a visualization that considers both the segmentation and the acquired medical image data.

Smoothing Segmentation Results with Morphologic Operations In the following, we describe an efficient approach to smooth the segmentation result with morphological image processing [Neubauer *et al.*, 2004a].

We denote with v_1 the value of the segmentation result and with v_2 the value of the background voxels. τ represents the isovalue which is computed as $(v_1 + v_2)/2$. The method is based on morphologic

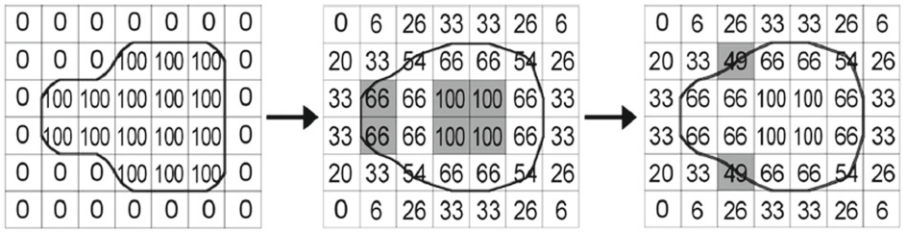

FIGURE 6.21 *Smoothing binary segmentation results (left) through morphological operators with* $v_1 = 0, v_2 = 100$, *and* $\tau = 50$. *The dark gray voxels in the left part (middle) would be removed by an erosion. This is avoided to preserve the object shape. In a last step, voxels that belong to the segmentation result and have a value below 50 are corrected (right). Instead of a binary volume, nine different values arise and lead to reduced discontinuities (Courtesy of André Neubauer, VRVis Wien).*

operations performed in a defined distance of the original object boundary and it is constrained to maintain the foreground-background classification.

Each voxel near the object boundary is assigned a new value v, with $v_1 \leq v \leq v_2$ using the following algorithm: A reference mask V_{ref} is created by eroding the segmentation result. After the erosion, all voxels of V_{ref} are assigned the value v according to Equation 6.3, which in essence moves the isosurface closer to the background value, thus farther away from the segmented object.

$$v = v_2 - (v_2 - v_1) * \frac{1}{3} \tag{6.3}$$

During erosion, the algorithm checks whether the shape has significantly changed such that small features on the boundary of the segmented object would be lost [Neubauer *et al.*, 2004b]. Significant changes are defined as removed voxels which are not adjacent to the original segmentation. These voxels are not removed (see Fig. 6.21).

After the modified erosion, two dilation operations are performed. The boundary voxels of the reference mask are tracked as reference voxels with the dilation front. Each voxel V that is added through dilation is therefore associated with a reference voxel V_{ref} and acquires the value

$$v = v_{ref} - (v_2 - v_1) * \frac{d}{3} \tag{6.4}$$

with d being the Euclidean distance from V to V_{ref} and v_{ref} being the value of V_{ref}. The method can be generalized to a wider filter region (with more erosion and dilatation steps). To preserve the segmentation result, it is desirable that all voxels which belong to the segmentation result yield a value above τ. This is achieved by a correction with a small positive δ in a final step. The resulting isosurface is at most one voxel away from the surface that would result from the original data.

This algorithm yields a near-linear value degradation from the reference mask outwards. It can be applied to smooth larger regions by eroding and dilating several times, see Figure 6.22. The algorithm retains the basic object shape and volume.

6.5.2 BASIC MESH SMOOTHING

In the following, we describe rather simple and general mesh smoothing techniques. In general, mesh smoothing does not change the number of vertices and the topology of a surface mesh. Instead, only the coordinates of vertices are changed. Thus, almost all mesh smoothing techniques are *topology preserving*. Before

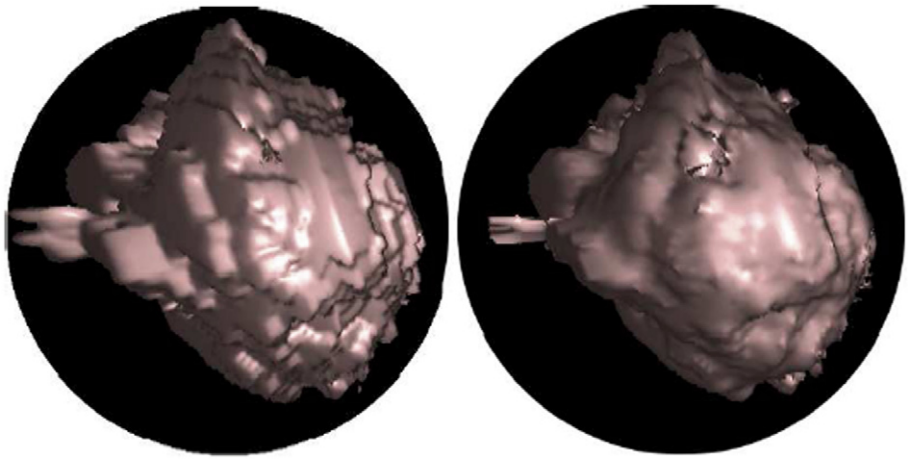

FIGURE 6.22 **Left:** *3D visualization based on the raw segmentation result.* **Right:** *The distracting interpolation artifacts are effectively reduced with smoothing based on a larger filter kernel. The circular images are from virtual endoscopy and resemble real endoscopic views (Courtesy of André Neubauer, VRVis Wien).*

we describe these techniques, we introduce the differential properties of curvature that are fundamental to mesh smoothing.

Curvature Efficient smoothing of a surface model reduces bumps and bulges. This leads to lower *curvature values*. Curvature characterizes how strongly the surface deviates from a plane. Curvature along a surface that lives in 3D space is measured along two directions: along the principal curvature direction and along an orthogonal direction. Based on these two measures κ_1 and κ_2, various curvature measures are defined, e.g.,

- mean curvature ($H = 1/2(\kappa_1 + \kappa_2)$), and
- Gaussian curvature ($K = \kappa_1 \cdot \kappa_2$).

Curvature measures are derived from 2nd order partial derivatives and thus can only be defined for smooth continuous surfaces with at least C^1 continuity. A polygonal mesh does not fulfill this requirement, actually it is locally flat and exhibits no curvature. For approximating curvature values, higher order polynomial surfaces, e.g., quadric or cubic surfaces, are optimally fitted to a polygon mesh. There are various approaches, differing in accuracy and computational effort, see e.g., [Razdan and Bae, 2005, Taubin, 1995b]. For our purposes, it is sufficient to state that curvature can be effectively approximated and used to evaluate smoothing algorithms. The chosen curvature value can be color-coded on the surface, and additionally histograms of curvature values may be displayed and analyzed. We discuss curvature estimation both related to curvature values and curvature directions in more detail in § 12.3, since it plays an essential role in illustrative visualization.

Laplacian Smoothing The Laplacian filter or operator is the simplest smoothing method, and it is widely available in visualization and geometric modeling toolkits (see Fig. 6.23). It is sometimes called *relaxation filter*. The term *relaxation* is related to the effect on the surface where the bending energy is reduced—the surface is relaxed. The filter is applied iteratively to a polygonal mesh and produces smooth surfaces after an appropriate number of iterations. In most implementations, it is applied to the 1-ring neighborhood,

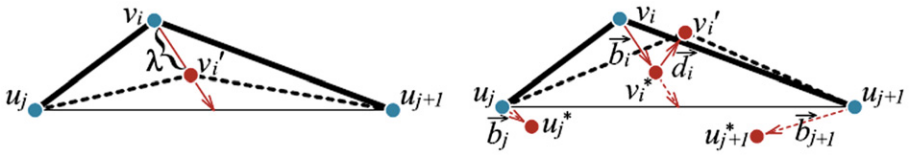

FIGURE 6.23 **Left:** *The Laplacian filter applied to a polygonal curve moves vertex* v_i *in the direction of its neighbors according to the weighting factor* λ. **Right:** *A correction after the Laplacian reduces the effect of the first smoothing step. The correction moves in a direction derived by the local normal vectors (Courtesy of Ragnar Bade, University of Magdeburg).*

since this is faster compared to the more extended neighborhoods. Because of this, it is also referred to as *umbrella operator*. This term relates to the fact that an umbrella has a central vertex, connected to a number of vertices in the surroundings. In each iteration, a vertex is moved in the direction of the geometric center of its neighbors. A weighting factor λ determines the influence of the original vertex position to its value after smoothing. The transformation of each vertex is performed according to Equation 6.5, where v_i is the previous position of a vertex, and v_i' is its new position and u_j are vertices of the neighborhood from v_i. Thus, the Laplacian operator is a discretized version of the Laplacian for continuous analytical functions, applicable to polygonal meshes. The number of iterations and the weighting factor λ are the two parameters that determine the amount of smoothing.

$$v_i' = v_i + \frac{\lambda}{m}\sum_{j=1}^{m}(u_j - v_i)v_i,\ u_j \in V,\ v_i \neq u_j,\ \forall u_j \in U_{vi} m = \|U_{vi}\| \tag{6.5}$$

Since all neighbors are treated equally in the smoothing process, the method leads to deformations of the surface when regions with significantly smaller edges are close to regions with large edges. In surface models, directly extracted from regular volume grids, this problem is not severe. If meshes are simplified and compressed, however, subsequent smoothing should properly weight the vertex contribution according to the distance from the current vertex.

An essential drawback of Laplacian smoothing is that smoothness is achieved at the expense of accuracy. Small details may be lost, since the method is not adaptive, similar to the Gaussian filter in image smoothing. Volumes typically shrink considerably. These problems are particularly relevant in medical applications where 3D visualization should provide faithful renditions suitable for treatment decisions. These drawbacks gave rise to a number of refinements that are discussed in the following.

Laplacian with Correction The Laplacian with an additional correction was introduced by Vollmer *et al.* [1999] to reduce volume shrinkage. In this approach, vertices are translated in roughly the original direction by a certain amount after Laplacian smoothing. The specific direction and the amount are derived from the previous position of that vertex and the averaged translation of its direct neighbors. This algorithm is rather time-consuming, since each step is more compute-intensive and more steps are required compared to the original Laplacian. However, more accurate results are achieved. A drawback is that the Laplacian with correction has four parameters (two in addition to the two of the Laplacian). Thus, users need guidance to adjust these parameters. Figure 6.24 illustrates the working principle of the Laplacian and the Laplacian with correction on a 2D polygonal curve.

The λ/μ Filter The λ/μ filter (also referred to as low-pass filter) combines a positively and a negatively weighted Laplacian filter that are applied alternately [Taubin, 1995a]. The result is similar to the Laplacian with correction and leads to more accurate results than the original Laplacian. In particular, volume

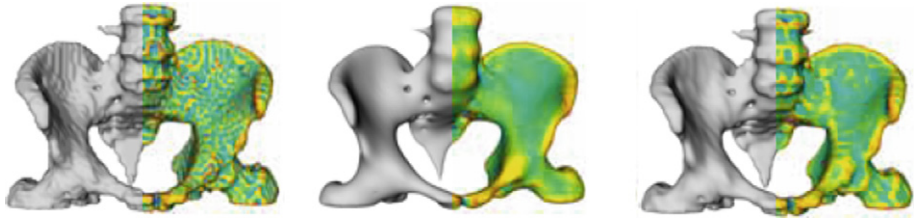

FIGURE 6.24 *Mesh smoothing and resulting maximum curvature in the right part (green represents zero and red represents maximum values).* ***Left:*** *Original surface of a pelvis extracted from binary thresholded CT data.* ***Middle:*** *After Laplacian smoothing with 20 iterations and* $\lambda = 0.7$. ***Right:*** *Laplacian smoothing with correction but identical number of iterations and weighting factor* λ. *The surface is not that smooth but represents the anatomy more faithfully (Courtesy of Ragnar Bade, University of Magdeburg).*

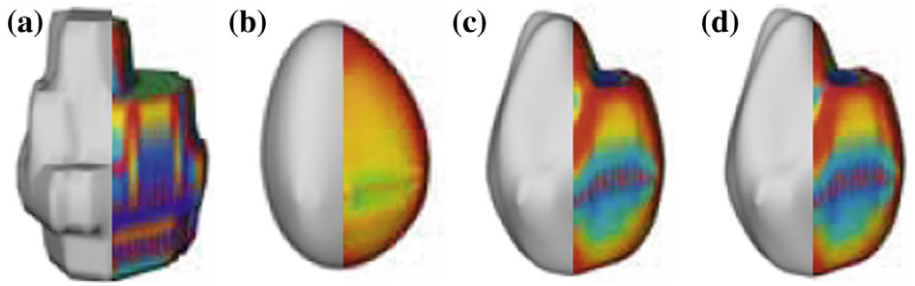

FIGURE 6.25 *Due to their small size lymph nodes are eligible for substantial volume shrinkage. Mesh smoothing and resulting maximum curvature (green represents zero, red represents maximum positive values, blue represents maximum negative curvature). (a) original surface, (b) after Laplacian smoothing with 50 iterations and* $\lambda = 0.9$, *(c) after Laplacian smoothing with correction, and (d) after applying the low-pass filter (Courtesy of Ragnar Bade, University of Magdeburg).*

shrinkage is strongly reduced. The λ/μ filter is particularly often employed to yield smooth and accurate surface meshes for grid generation in finite element simulation (§ 19.3.3). In Figure 6.25, the effect of the λ/μ filter is compared to other smoothing filters for a surface model of a lymph node.

The three filters described so far, benefit from a data structure that enables a fast traversal of vertices with immediate access to their 1-ring neighbors.

Mean Curvature Flow Smoothing with the *mean curvature flow* was introduced by Desbrun *et al.* [1999] to perform smoothing independently from the triangulation. With this approach, smoothing is applied in the direction of the local surface normal and the amount of smoothing is determined by the *mean curvature*. With the mean curvature flow the new position of v_i' is computed according to Equation 6.6.

$$v_i' = v_i + \frac{\lambda}{G} \sum_{j=1}^{m} (\cot \alpha_{ij} \cot \beta_{ij})(u_j - v_i) v_i \in V,\ u_j \neq u_j,\ \forall u_j \in U_{v_i} m = \|U_{v_i}\| \qquad (6.6)$$

Smoothing with the mean curvature flow preserves the shapes better compared to standard Laplacian smoothing, but it does not preserve the volume well.

Mean and Median Smoothing of the Surface Normals Discontinuities of surface normals are pronounced by lighting models and thus clearly visible. If similar methods, that were discussed so far for vertex

coordinates, are applied to surface normals, and if vertices need to be only slightly translated according to the modified normals, volume shrinkage might be reduced. This idea was realized by several groups, in particular by Yagou *et al.* [2002a,b], Tasdizen *et al.* [2002].

Mean filtering is performed by first computing the average normal $\vec{m}$ for a triangle based on the normals of all adjacent faces and then adapting the vertices of the triangle with a weighting factor λ considering vertices of the 1-ring neighborhood. Median filtering of the surface normals is inspired by the median filter used for noise reduction in images and requires a proper sorting of the normals in the neighborhood. The normal $\vec{m}$ is then replaced by the median in this sorted sequence of normals.

6.5.3 INTERACTIVE REAL-TIME MESH SMOOTHING

Mesh smoothing, like mesh extraction, is an operation that should be performed fast enough to enable real-time adjustment of parameters. As an example, Laplacian smoothing is controlled by two parameters, namely the weighting factor λ and the number of iterations. With a fast implementation, the user might adjust these parameters by mouse movements in x and y direction, similar to brightness and contrast adjustments in the reading of radiological image data. For larger meshes, real-time behavior can only be achieved by exploiting the GPU or by parallelizing the code to be executed at multiple cores. Mönch *et al.* [2012] described such an interactive mesh smoothing process.

To implement Laplacian smoothing efficiently, two aspects are crucial:

- Data structures must be suitable for stream processing.
- Synchronization should be minimized.

In the following, we describe such an interactive mesh smoothing approach introduced by Mönch *et al.* [2012]. Proper data management is crucial to achieve high performance on throughput processors. While generic mesh data structures (recall § 6.2.3) are often optimized for editing of the topology, the performance suffers from too many memory indirections. The data structure described in the following ensures that the processors' cache line mechanism is utilized well, as required data is contiguously stored in memory. The processing benefits from prefetching the coordinates of the next vertices. To cope with the memory latency, modern CPUs can analyze the instructions and execute them out-of-order to lower the impact of memory fetches. SIMD (single instruction multiple destinations) allows wider memory fetches and operations per clock. NVIDIA's SIMT (single instruction multiple threads) model, established with CUDA, widened the data processing further running thousands of threads in parallel.

ALGORITHM 1 Pseudocode of Laplacian smoothing

```
1:  for i = 0 → iterations do
2:      for all v ∈ Vertices do
3:          v_s.Initialize()
4:          for all v_n ∈ v.GetNeighbors() do
5:              v_s ← v_s + v_n
6:          end for
7:          v_s ← v_s/v.NumNeighbors()
8:          v_s ← v + α · (v_s − v)
9:      end for
10: end for
```

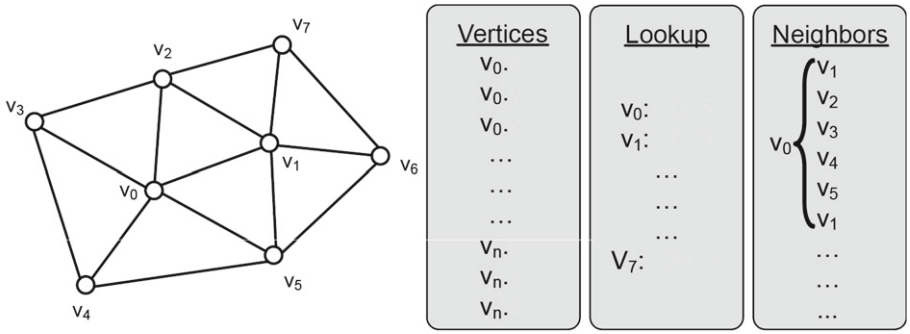

FIGURE 6.26 *The data for hardware-accelerated mesh smoothing is organized in three arrays: vertex coordinates, a lookup table holding information on the number of neighboring vertices, and their location in the list of neighboring vertices (From: [Mönch et al., 2012]).*

We consider the pseudocode of Laplacian smoothing (Alg. 1), since a number of other smoothing techniques have the same algorithmic structure. The algorithm (see Alg. 1) consists of an outer loop (line 1), which cannot be parallelized, since it requires the results of a former cycle (line 7). The second loop (line 2) can be executed free of synchronizations within one iteration. For the Laplace+HC- and the low-pass filter, an additional correction step has to be considered.

The data structure for fast CPU and GPU-based mesh smoothing is depicted in Figure 6.26. In this data structure, each vertex holds information on its neighboring vertices. Thus, an array for the vertex coordinates and a second array for the indices of neighboring vertices is created. For each vertex, the list of neighboring vertices is finalized by adding the first neighbor to the list again. This duplicate neighbor index simplifies the code for average vertex normal generation from the adjacent triangles. Through counterclockwise sorting of the list the 1-ring neighborhood may be created quickly. Furthermore, a lookup table contains neighborhood information, such as the number of neighboring vertices and the location (offset) of the indices of neighboring vertices in the neighbor index list.

CPU Implementation with OpenMP CPU-based smoothing filters can easily be extended by OpenMP which is supported by a variety of compilers, e.g., Microsoft Visual C/C++ 2008 [Mönch et al., 2012]. Especially the iteration over the vertices within one smoothing cycle can be parallelized. After computing the new vertex positions, the pointers to the input and output arrays need to be switched, such that the subsequent smoothing iteration continues with the updated vertex positions as new input. The required normal update is carried out likewise.

OpenGL Implementation Smoothing through OpenGL involves the usage of shaders. Vertex shaders are primarily designed to output positions and they execute code for *one* vertex without natively providing further geometry information, such as the location of neighboring vertices. To provide access to the previously described data structure (see Fig. 6.26), vertex buffer objects (VBOs) and texture buffer objects (TBOs) are used. OpenGL provides a buffer-centric data model, in which the programmer can manage memory on the device and bind it to different functionalities of the rendering pipeline. VBOs allow to upload vertices and related vertex attributes to the graphics card. Up to 16 attributes (4D vectors) may be attached to a vertex. This is sufficient to store the 1-ring neighborhood. For mesh smoothing, this includes the vertex's own position and the neighborhood lookup data. Within the vertex shader, only the attributes of the current vertex can be accessed. Since mesh smoothing requires information on the location of neighboring vertices, TBOs are employed which allow arbitrary access to any location within bound

buffer objects from a shader. Thus, it is necessary to bind the buffer holding all the vertex coordinates, and the buffer with the indices of neighboring vertices as textures. For storing the smoothed vertices, a transform feedback buffer (XBO), that has the same size as the vertex buffer, is used. This setup is shown in Figure 6.27 and allows to implement the inner smoothing loop inside a vertex shader.

To perform the next outer loop cycle, the buffers are switched such that the modified vertex positions become the input data (see Fig. 6.28). For that, the buffer containing the updated vertex data is bound as VBO and TBO, whereas the former vertex data buffer is bound as XBO to store the new vertex positions. Thus, this is equivalent to the pointer switching in the CPU method. The buffers used for smoothing

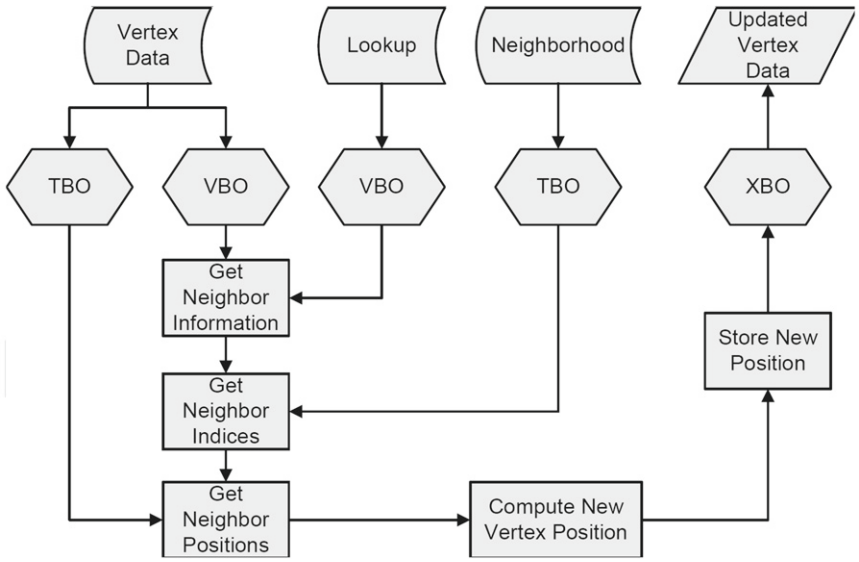

FIGURE 6.27 *The interaction between the different buffer objects required for accessing neighborhood information and output of the results (From: [Mönch et al., 2012]).*

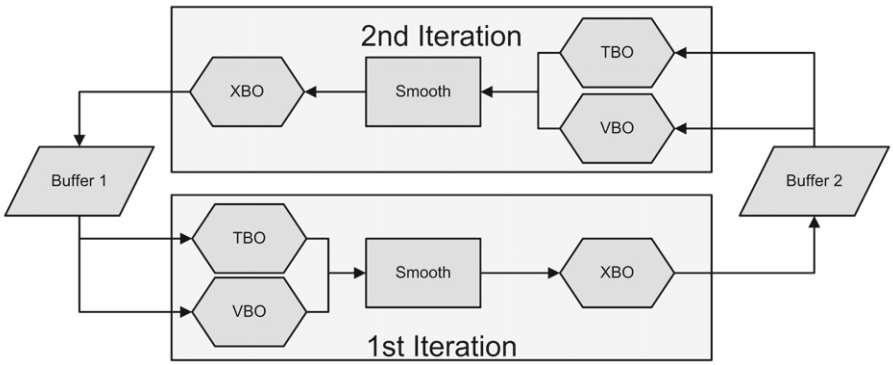

FIGURE 6.28 *The input and output buffers are switched after each smoothing cycle by binding them as VBO/TBO and XBO (From: [Mönch et al., 2012]).*

computations can subsequently be used for rendering. The VBOs for vertex data and lookup, as well as the neighborhood TBO can again be employed to gain all neighborhood information and finally compute the updated vertex normals. Since the data is already located in graphics card memory, it can directly be used for visualization. As a result, the OpenGL approach is advantageous for smoothing with an immediate visual feedback. The execution of GLSL code depends on the employed OpenGL extensions. For the described setup, at least OpenGL 3 is required.

6.5.4 EVALUATION OF SMOOTHING APPROACHES

Before we discuss the appropriateness of the basic smoothing methods for medical surface models, it is necessary to elaborate on evaluation criteria that go beyond a subjective visual evaluation.

A first objective criterion is the *distance* between a surface considered to be correct and a smoothed surface. Often, it is reasonable to consider the Marching Cubes result as correct, although the simple linear interpolation introduces a slight error. We can thus compare a smoothed surface with the original surface by computing the Hausdorff distance between them. We got to know this measure in the validation of image segmentation. The Hausdorff distance is a kind of worst-case approximation of the distance between the surfaces; the average distance is usually significantly lower. A second widely used measure is *volume preservation*, that measures how strongly the volume after smoothing differs from the original volume.

Comparison of Basic Mesh Smoothing Filters Bade *et al.* [2006] compared the effects of basic mesh smoothing filters to the visualization of medical surface models. A set of six surface models was selected, including elongated and branching anatomical structures, large compact structures, such as organs, and small compact structures, such as lymph nodes. Different types of anatomical structures were considered, since optimal approaches or optimal parameters might differ depending on the geometry and size of the considered objects. For all smoothing filters, different parameters were selected to understand at which levels optimal trade-offs between accuracy and smoothness are achieved. The 1-ring and 2-ring neighborhood were employed (see Fig. 6.29).

The comparison focused on the measurements described above. The whole analysis was quite complex, since six shapes, two neighborhoods, three smoothing methods, four different iteration numbers, and six smoothing factors were considered, resulting in 864 measurements. For smoothing compact objects (e.g., organs, lymph nodes), the low-pass filter and the Laplace+HC with a weighting factor λ between 0.5 and

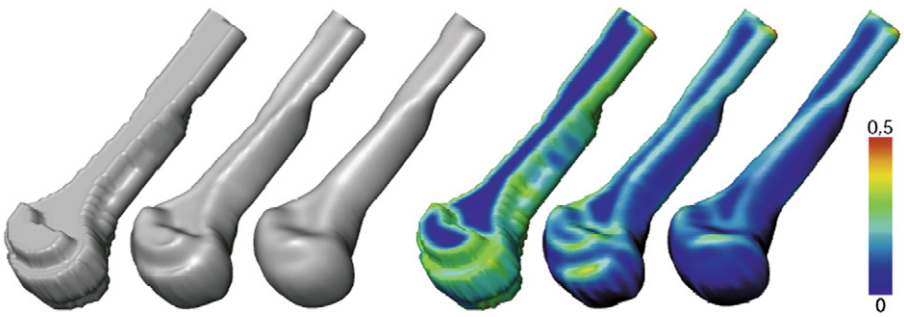

FIGURE 6.29 *Curvature is used to assess the effect of smoothing. From left to right: the Marching Cubes surface generated from the binary segmentation of a bone, Laplacian smoothing with 1-ring neighborhood, Laplacian smoothing with 2-ring neighborhood. The same three models are shown with color-coded curvature. The strong reduction of high-frequency noise leads to significantly reduced curvature (Courtesy of Ragnar Bade, University of Magdeburg).*

0.9 and with 20 to 50 iterations turned out to be equally appropriate. For flat objects, smoothing with the low-pass and Laplace+HC filter yielded the best results. However, the low-pass filter slightly better preserves the volume and shape. A weighting factor between 0.5 and 0.7 and 20 iterations turned out to be appropriate. For elongated and branching vascular structures, partially represented with a few voxels in their cross sections, all methods performed badly. This gives rise to specific surface extraction methods for vascular and other branching structures (see § 11.5).

6.6 ADVANCED MESH SMOOTHING

While the methods described above treat the whole mesh equally, advanced methods are adaptive and modify the smoothing process depending on a local analysis. This enables them to better preserve features or to provide error bounds. Such adaptive methods, similar to the global methods, are inspired by image processing techniques, such as anisotropic diffusion. These methods are referred to as *feature-sensitive* and are widespread in processing engineering data, such as CAD models[7] [Kobbelt *et al.*, 2001]. However, if these methods are applied to surface models derived from medical image data, they identify staircase artifacts as features to preserve. Only recently, adaptive methods specific for medical image data have been developed.

6.6.1 CONSTRAINED MESH SMOOTHING

Dedicated attempts have been made to wrap binary volume data with smooth surfaces. This principle is closely related to the segmentation with snakes (recall § 4.5.1), where energy minimizing surfaces are fitted to the image data and attracted to features, such as edges. Gibson [1998] applied the principles of energy-minimization, to generate smooth surfaces from binary medical volume data. Initially, the centers of each boundary cell of a binary volume are extracted and connected. In an iterative energy minimization process, each vertex is modified but never leaves a cube region with a size that corresponds to the resolution of the data (Fig. 6.30). In particular, the "terraces" in binary segmented data can be handled appropriately by their method of fitting a surface around the binary segmentation result. The iterative relaxation process is constrained by a voxel environment—created by the volume cells that contain at least one boundary voxel of the binary segmentation and at least one background voxel. This concept is referred to as *Constrained elastic surface nets* (CESN). Such constraint may even be used to give a guarantee on the preciseness of the surface.

Later, Bade *et al.* [2007] used a diamond-shaped region to restrict the movement of each vertex and enables a slightly better precision (see Fig. 6.31). For segmentation results of general anatomical shapes constrained smoothing techniques can be recommended. Problems occur if structures are at least partially

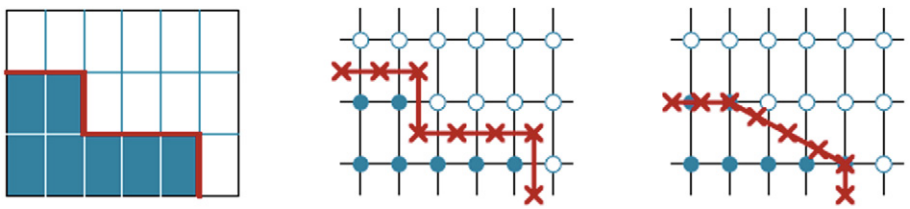

FIGURE 6.30 *Constrained elastic surface nets: The vertices of the red surface are iteratively modified and reduce even staircasing artifacts without leaving a voxel-sized cubical region (Courtesy of Ragnar Bade, University of Magdeburg).*

7 CAD here denotes computer-aided design, whereas it denotes more frequently in the book computer-aided detection.

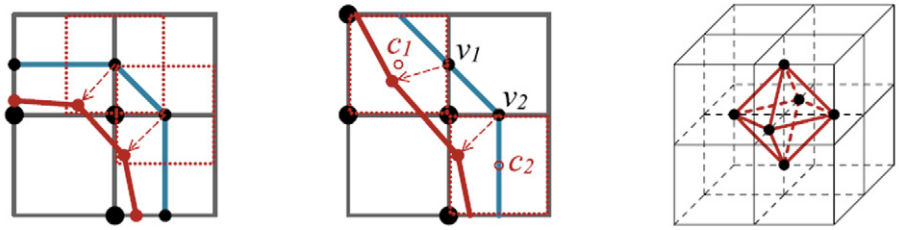

FIGURE 6.31 *Diamond-constraint smoothing: The movement of vertices is restricted to a 3D diamond-shaped region centered around their initial coordinate (Courtesy of Ragnar Bade, University of Magdeburg).*

very thin, since they may degenerate to a line. With CESN and related techniques there is no guarantee that the topology of the surface can be preserved.

6.6.2 CONTEXT-AWARE SMOOTHING

In a series of publications, MÖNCH and colleagues identified features to preserve in medical image data and designed methods to modify existing smoothing techniques that restrict smoothing to certain regions or reduce the amount of smoothing to preserve features. These methods are summarized as *context-aware* smoothing [Mönch *et al.*, 2011a].

Distance-aware smoothing. Since surface visualizations are frequently in therapy planning, an essential aspect is to preserve such distances. That means that interactive or automatic measures[8] after smoothing should yield reliable results. Thus, smoothing needs to be adapted in regions of a surface where another surface is located close. Mönch *et al.* [2010a] introduced this concept and referred to it as *distance-aware smoothing*. Otherwise, more aggressive smoothing is appropriate. Figure 6.32 illustrates distance-aware smoothing. This strategy leads to visualizations where the amount of detail varies according to the amount of smoothing. From a perceptual point of view, the viewer is directed towards the important regions with more detail.

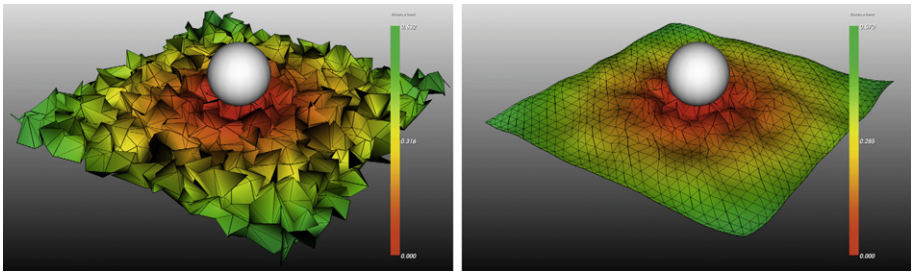

FIGURE 6.32 *Principle of distance-aware smoothing. The noisy plane (left) is strongly smoothed in regions that are far from the sphere (right). In regions close to the sphere, however, smoothing is strongly reduced. The plane geometry is color-coded according to the distance, with red representing low distances (Courtesy of Tobias Mönch, University of Magdeburg).*

8 Measurement techniques are discussed in Chapter 10.

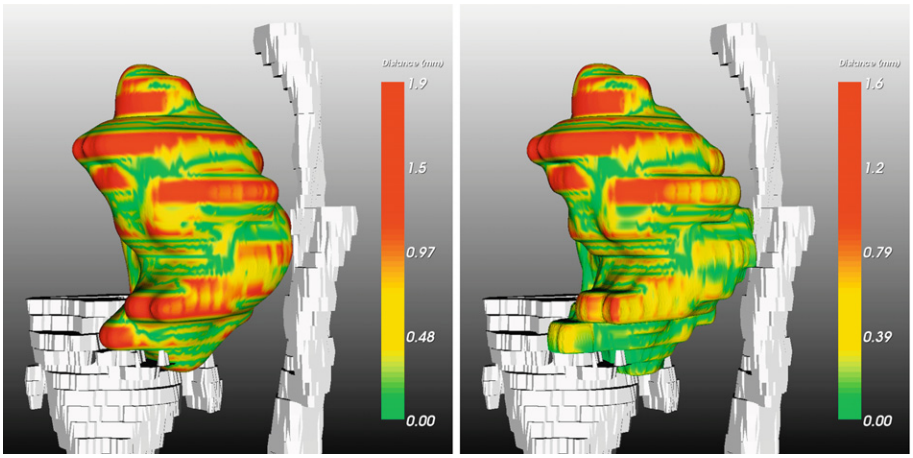

FIGURE 6.33 *Distance-aware smoothing preserves the distance between a tumor and an adjacent vascular structure.* **Left:** *Uniform Laplacian smoothing.* **Right:** *Distance-aware smoothing. The damping factor included in the smoothing process is linearly adapted to the distance to adjacent structures (Courtesy of Tobias Mönch, University of Magdeburg).*

To implement this strategy, the minimum distance $dist_{min}$ between each pair of objects is incorporated a smoothing damping factor $damp$ that may be combined with any existing smoothing technique. When the distance is below a threshold, distance-aware smoothing is enabled. The damping factor $damp$ is 1 (maximum damping) in regions with a distance of $dist_{min}$ and decreased in other regions. $damp$ may be decreased either linearly or exponentially until 0. Figure 6.33 presents an example with linear scaling.

Advanced topic: Staircase-aware smoothing. To provide smooth visualizations without strong side-effects, Mönch *et al.* [2010b] described a method that detects staircases from anisotropic tomographical data. This method explicitly considers a model assumption derived from the properties of medical imaging data. The slicing direction, the in-plane resolution and the slice distance are extracted from the DICOM data and used to search for staircases. Smoothing is modified with weights that represent the detected staircases. As Figure 6.34 indicates, the same amount of smoothness is achieved with better accuracy. This *staircase-aware* smoothing takes longer, since a preprocess is necessary to identify the staircases and the smoothing process is slightly more complex. The modifications due to distance computations and due to staircases may be combined.

Staircase-aware smoothing is related to an approach presented by Xu *et al.* [2006]. They analyzed a mesh to classify vertices in *noise vertices* (probably due to noise or reconstruction errors) and *feature vertices*, focusing smoothing on the noise vertices. Since this approach was also inspired by and tested with contours from tomographical data, the methods are closely related. However, Xu *et al.* [2006] also refine meshes adaptively, whereas Mönch *et al.* [2010b] leave the mesh complexity constant. The method from Xu *et al.* [2006] is less specialized, not explicitly searching for staircases.

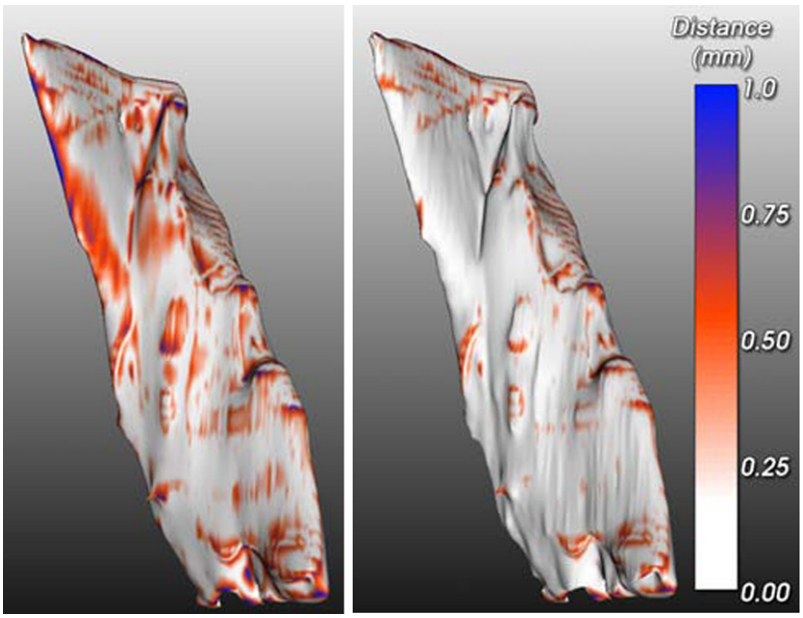

FIGURE 6.34 *A muscle model, where staircases were detected to scale the smoothing factor appropriately.* **Left:** *Uniform smoothing leads to larger displacements along the surface.* **Right:** *The distance to the original surface before smoothing is color-coded. Only close to the staircases, notable differences exist* (From: [Mönch *et al.*, 2011a]).

6.6.3 EXTRACTING SURFACES FROM LABEL VOLUMES

Label volumes contain multiple segmentation results. If a label volume should be used, e.g., to generate surfaces from all inner organs, there are cells (consisting of the eight adjacent vertices considered in Marching Cubes) that represent more than two different values. Thus, surface extraction is more complex than the definition of just one material transition. Moreover, any post-processing, such as smoothing operations, must consider the special problems of multiple materials to avoid that the surfaces intersect later. Hege *et al.* [1997] introduced "a generalized Marching Cubes"—an approach to extract smooth surfaces from non-binary *label volumes*. They extended the original Marching Cubes and considered the different topological states if three, four, or even more materials exist in one cell. They could show that in a cell with three materials 44 additional cases occur. In cells with four different materials, another 66 topological different states are possible. Together with the 15 cases from a binary situation, 59 and 125 cases respectively have to be considered. For three materials, the configurations are carefully illustrated. However, it is not sufficient to consider just more cases. The boundaries between different materials in the inner of the cell needs to be taken into consideration, leading to vertices not only along at edges, but also at faces and in the internal part of a cell. Thus, as a consequence of this complex setup, a lookup-table with a manageable size (below 1 MByte) is only feasible for up to three materials. According to Hege *et al.* [1997] penetrating surfaces have to be avoided.

Later, Wu and Sullivan [2003] described the multimaterial Marching Cubes algorithm that provides an alternative to the "Generalized Marching Cubes." They renounce on the idea of using a case table to prepare a triangulation for every possible case. Similar to Hege *et al.* [1997], in case that the four vertices that constitute a face of the cells share more than two materials, additional face center points are introduced. Figure 6.35 illustrates some possible configurations and the surfaces defined in this case.

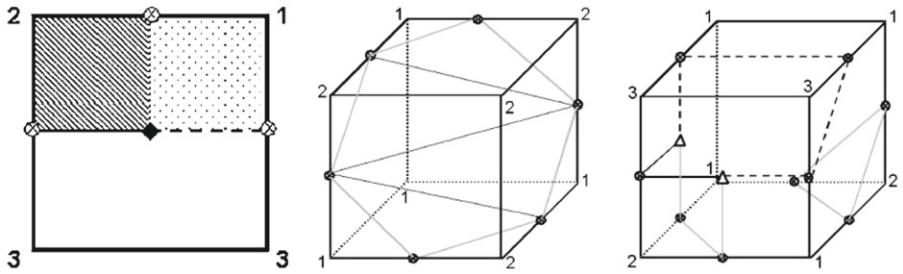

FIGURE 6.35 *Multimaterial Marching Cubes.* **Left:** *A 2D example with three materials where a central point is inserted to generate consistent isolines.* **Middle:** *A 3D example with three materials where no face centers need to be inserted.* **Right:** *A 3D configuration with three materials where two face centers were introduced for surface extraction (Inspired by [Wu and Sullivan, 2003]).*

Surface extraction is quite complex in cells where more than two materials exist. In the large majority, where only two materials are present, the standard Marching Cubes extraction may be performed. With this strategy, all surfaces can be generated in one pass and with a similar performance like the classical Marching Cubes. A drawback is that the number of generated triangles is large.

6.6.4 EVALUATION OF ADVANCED MESH SMOOTHING

In this subsection, we build on the evaluation criteria discussed in § 6.5.4. These criteria are, of course, also essential for the advanced smoothing approaches. However, they need to be adapted to the context-specific character of advanced methods. As an example, for the evaluation of distance-aware smoothing, the distance evaluation needs to be performed at a finer level. It is essential that distances to critical structures are preserved accurately, whereas global distance measures or volumetric overlap are less relevant. Consequently, distance-aware smoothing was primarily evaluated with respect to its effect on minimal distance computations. Similarly, staircase-aware smoothing should be evaluated with respect to its local effect close to the identified artifacts. Advanced smoothing techniques combine a feature detection step and a sophisticated smoothing approach considering these features. Obviously, the computational demands are increased compared to basic methods. Thus, a substantial performance analysis should also be part of the evaluation.

6.7 MESH SIMPLIFICATION AND WEB-BASED SURFACE RENDERING

A promising usage context of 3D medical visualization is its use directly in the web browser. While web-based interactive 3D renderings are feasible in principle since the 1990s, only recently convenient use is possible. On the one hand, with new standards such as HMTL 5, 3D renderings may be used without the need to install any plugin—an improvement that is crucial in hospitals, where it is often not allowed to install additional software. On the other hand, the bandwidth of current internet connections enables interactive 3D visualizations of at least moderately sized surface meshes. Surface renderings are based on polygonal meshes that have a much lower storage footprint than medical volume data. Thus, in the foreseeable future, mesh-based surface rendering is the better option for web-based medical visualization. In medicine, two application types are prevailing:

- interactive surgical planning, and
- interactive medical education

In the following, we discuss web-based surface rendering for these applications.

6.7.1 MESH SIMPLIFICATION

For web-based surface rendering, the size of surface models is crucial. Several strategies are used to cope with the related bandwidth problems. The most general strategy is to transform a surface mesh M in a low resolution mesh M' that is as close as possible to the original mesh but encompasses a significantly lower number of vertices and polygons. A good trade-off between accuracy and size of the mesh can only be achieved with adaptive methods that remove only geometrical details not contributing significantly to the shape. Flat regions with no or only low curvature may be strongly reduced, whereas regions with high curvature can only be reduced mildly.

The selection of a difference metric is crucial. In theory, it should reflect how much a model is visually changing. In practice, however, this is difficult to measure. Hence, a number of specific difference metrics are used, such as the geometric distance in object space (e.g., the Hausdorff distance), the pixel distance in screen space, or the differences in the attribute space (e.g., normals, colors, or textures).

Mesh simplification may take as input

- a certain target number of polygons,
- a certain percentage of reduction in size, or
- an error that should not be exceeded.

Vertices and/or edges are classified with respect to their relevance by assessing the error that would be introduced by removing them. Based on this assessment, geometrical details are removed until the target size or percentage is reached. It is essential that, after removing a vertex, a local *remeshing* provides a local triangulation in that area. Edge collapse is a simplification operation and means that one vertex v_s is moved to an adjacent vertex v_t and then deleted. Remeshing means that after removal of a vertex and the associated edges the affected part of the mesh is transformed in a triangle net (see Fig. 6.36). This is always possible. Among the possible triangulations, one is searched for where the triangles are as equi-lateral as possible.

Thus, in contrast to mesh smoothing, mesh simplification often alters the topology of a surface mesh. Garland and Heckbert [1997], Hoppe *et al.* [1993], and Lindstrom and Turk [1998] described powerful mesh simplification algorithms.

Progressive Meshes Instead of just producing *one* strongly reduced mesh, it might be reasonable to generate a sequence of reduced meshes. If the user opens a corresponding website, a strongly reduced base mesh is shown first. This base mesh M_0 may not be reduced further without changing the topology. If the user does not manipulate the visualization, refined versions are loaded. Instead of providing a number of complete meshes, it is reasonable to store a base mesh and the local differences to more refined meshes

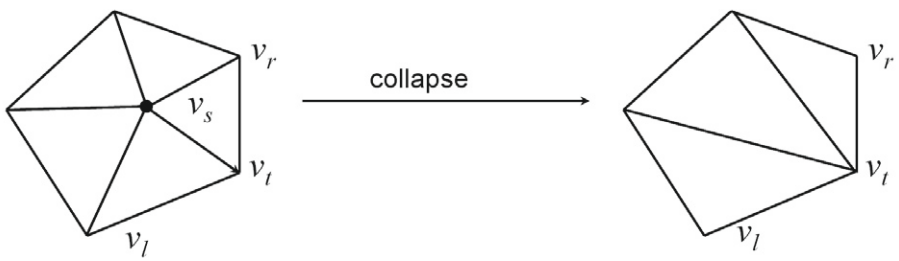

FIGURE 6.36 *After an edge collapse, the resulting loop of edges is triangulated such that no degenerated triangles arise.*

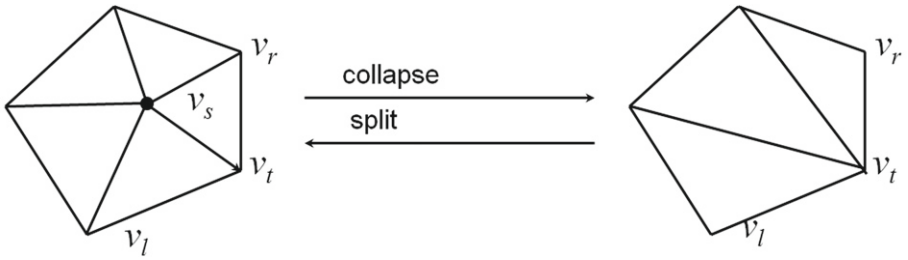

FIGURE 6.37 *Representation of a collapse/split. For the collapse of an edge, its left and right neighbor as well as the remaining vertex are stored. The coordinates of the removed vertex are also stored to be able to restore the edge in a split operation.*

$(M_1, M_2, \ldots, M_n)$. This is feasible if there are invertible transformations to reduce the mesh. This idea is known as *Progressive meshes* [Hoppe, 1996]. Figure 6.37 illustrates the invertible edge collapse operation employed for generating reduced meshes from the original mesh M_n.

The chosen simplification approach depends largely on the required accuracy for the simplified model. In particular the requirements for the boundary between the different objects are important. If adjacent objects are simplified separately, cracks between the polygonal surfaces may appear and need to be re-triangulated.

There are a number of surveys on the variety of mesh simplification algorithms, error metrics, and applications [Garland, 1999, Luebcke *et al.*, 2004].

6.7.2 WEB-BASED SURGICAL PLANNING

For regular clinical use, the development of Acrobat 3D PDF is promising. PDF documents are widely used for the exchange of documents among physicians. If the familiar experience of using PDF documents can be combined with the functionality to interactively explore a 3D model of the relevant patient anatomy,

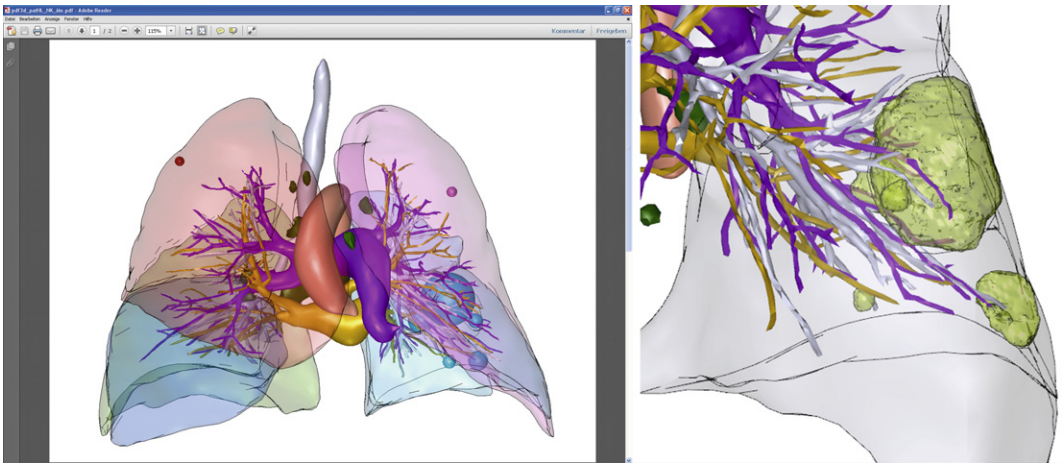

FIGURE 6.38 *Acrobat 3D technology is employed to provide interactive 3D visualizations for lung surgery planning. The close-up view (right) reveals that enough detail is available for specific treatment planning questions. The original surface model (1.463 K vertices and 2.289 K edges) was compressed by 90%. The PDF document is still rather large (9.9 MByte) (Courtesy of Steven Birr, University of Magdeburg).*

it is more likely that these functions are actually used. Figure 6.38 shows an example of a documentation and planning system for thorax surgery, presented by Birr *et al.* [2011b].

The use of Acrobat 3D PDF for surgery planning was pioneered by MeVis Distant Services in 2009 and is regularly performed for liver surgery planning in case of liver tumors and for planning transplantation surgery. They also take care of reducing the geometric models and provide sets of carefully defined 3D models that depict the vascular anatomy, the risk associated with surgery and a number of resection proposals. In Figure 6.39, the hepatic veins are shown and color-coded with respect to their distance to the tumor—the analysis of affected vascular structures is one essential step in surgery planning.

6.7.3 WEB-BASED MEDICAL EDUCATION

In § 5.9 we discussed the mobile use of medical (image) data. Surface renderings prevail in mobile settings, because they can be easily used in any web-browser. Computer-based training applications increasingly support web and mobile usage, and as a consequence, such support is expected by medical users. Figure 6.40 presents an example where 3D rendering in the web browser is employed for medical education.

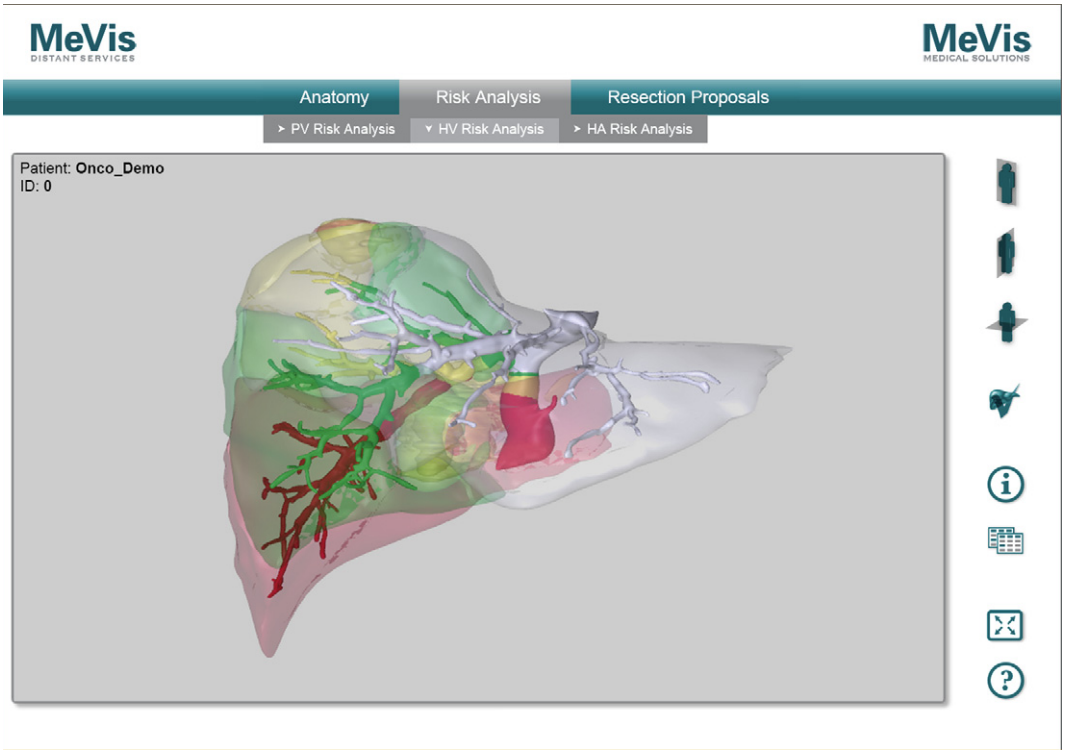

FIGURE 6.39 *Acrobat 3D technology is used for liver surgery planning. The liver and the hepatic vein (one of three vascular systems in the liver). The color of the vascular branches indicates their distance to a tumor (red represents regions in the 2 mm margin around the tumor). A set of 10 3D models, sharing large parts of the geometry are provided to support the complete surgical planning process with appropriate 3D models (10.6 MByte) (Courtesy of MeVis Distant Services).*

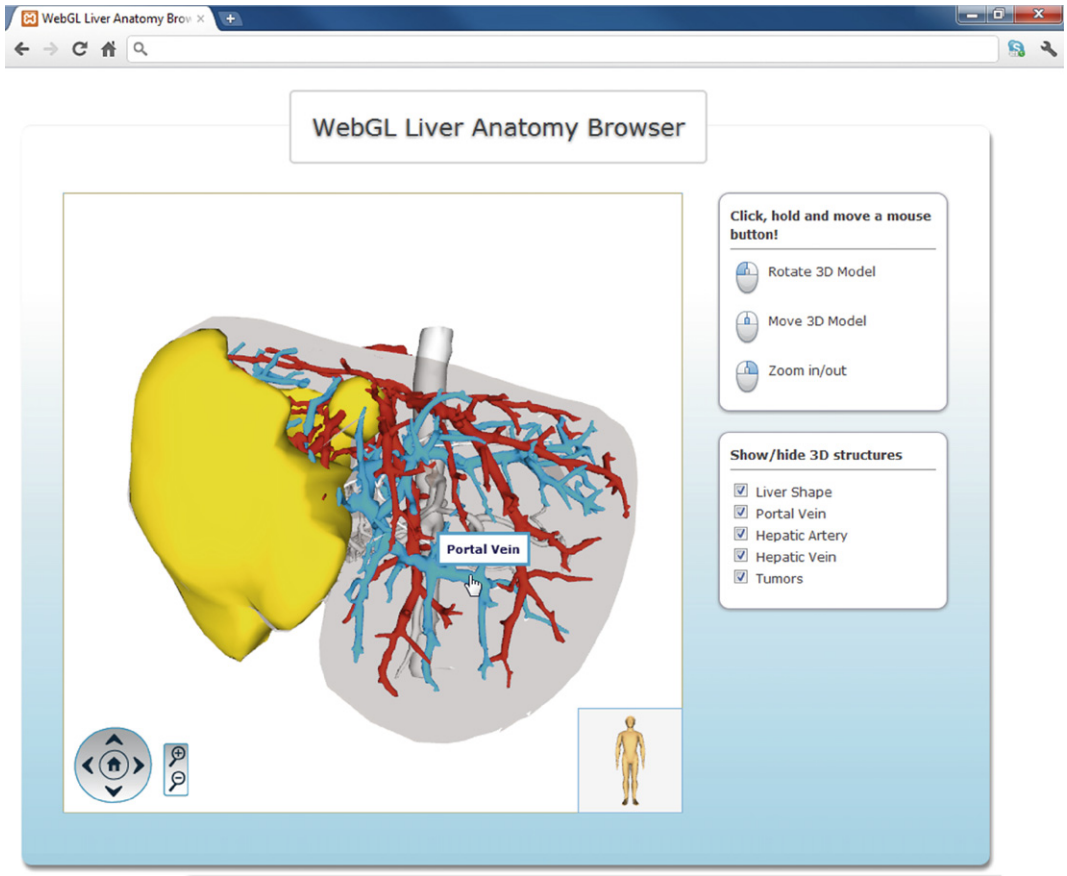

FIGURE 6.40 *The liver anatomy of a patient with a large tumor (yellow) is presented as surface mesh in the web browser. The 3D visualization is the basis for a training system, where medical doctors should assess the vascular anatomy and the resectability of the patient. Web GL is the underlying technology. The surface meshes have an overall size of 26 K vertices and 52 K faces resulting from a strong simplification of the original models (Courtesy of Steven Birr, University of Magdeburg).*

6.8 CONCLUDING REMARKS

In this chapter, we learned about the visualization of medical volume data by means of surface meshes. We discussed the extraction of surfaces from contours, the internal representation of meshes and the modification of the initially extracted meshes, e.g., to reduce their geometric complexity and to remove noise. Marching Cubes turned out to be a useful basic algorithm for surface extraction. Faster surface extraction techniques, more precise isosurface extraction and solutions for the ambiguity problem were discussed. Mesh smoothing was discussed in considerable detail to reflect on different usage scenarios. In particular, the surface extraction from binary and strongly anisotropic volume data causes severe aliasing artifacts which give rise to smoothing techniques. Curvature measures enable an objective discussion of the achieved effects on smoothness. Accuracy was evaluated, e.g., with respect to volume shrinkage and surface distances to original unaltered surfaces. The relation between noise reduction at the image data

level and the resulting surfaces was described. We discussed methods also with respect to performance issues that are particularly important when surface meshes are used in web-based environments. While our focus in this chapter was on the visual exploration on surface meshes for standard visualization, surface extraction is also the basis for illustrative rendering (Chap. 12) and for biophysical simulations (Chap. 19).

Surface Meshes for 3D Printing We have not discussed the emerging use case of generating medical surface models for 3D printing. Physical models of anatomical structures have a great potential for surgical planning and medical education. 3D printers have strongly increased in their capabilities, e.g., to employ different materials and colors, and even transparency and their price has been reduced considerably making the technology much more available. Schmauss *et al.* [2012], e.g., describe, how physical 3D models are used for planning heart valve replacement and Mönch *et al.* [2011b] describe the generation of physical vascular surface models with complex pathologies. Also for planning stent implantation [Armillotta *et al.*, 2007] and for educational purposes physical models are essential [Knox *et al.*, 2005]. In all these cases, the geometry is represented with surface meshes that should be smooth and accurate.

In addition, some specific requirements have to be considered, e.g., adjacent parts of a complex geometry should not be too close to each other to avoid that they merge in the model generation process. Also, depending on the quality of the 3D printer, very thin branches are not stable enough to be constructed. These specific requirements often need special editing with mesh processing software.

FURTHER READING

We discussed data structures for meshes only briefly. For an in-depth discussion, we refer to [Botsch *et al.*, 2002, Sieger and Botsch, 2011] and to a recent book on mesh processing [Botsch *et al.*, 2010]. Mesh simplification was also only briefly discussed. Cignoni *et al.* [1998] provides an overview.

Mesh smoothing may also be performed based on subdivision. These approaches add vertices and edges to rough portions of a mesh to represent it at a finer scale and thus provide a smooth appearance in regions where the original resolution was insufficient to capture curved regions appropriately [Xu *et al.*, 2006]. Theisel [2002] showed how exact contours of 3D regular volumes may be determined by triangular rational cubic Bezier patches. Curvature estimation is an essential aspect of mesh processing. Discrete differential geometry operators, in most cases tailored to triangular nets, are used for this purpose [Meyer *et al.*, 2002, Desbrun *et al.*, 2006].