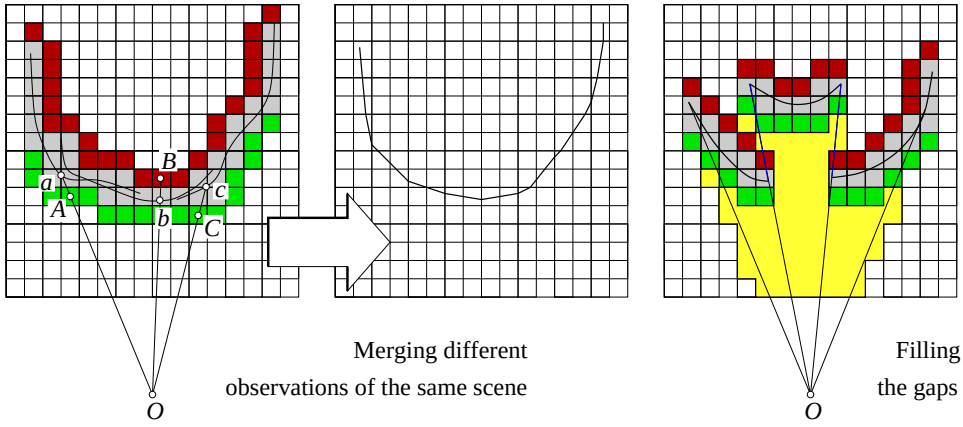

FIGURE 14.11: A 2D illustration of the Curless-Levoy method for fusing multiple range images. In the **left** part of the figure, three views observed by the same sensor located at the point O are merged by computing the zero set of a weighted average of the signed distances between voxel centers (e.g., points A, B, and C) and surface points (e.g., a, b, and c) along viewing rays. In general, distances to different sensors would be used instead. The light-gray area in the **right** part of the figure is the set of voxels marked as empty in the gap-filling part of the procedure.

scanner, as well as a physical model constructed from the geometric one via stereolithography (Curless and Levoy 1996).

14.4 OBJECT RECOGNITION

We now turn to actual object recognition from range images. The registration techniques introduced in the previous section will play a crucial role in the two algorithms discussed in this one.

14.4.1 Matching Piecewise-Planar Surfaces Using Interpretation Trees

The recognition algorithm proposed by Faugeras and Hebert (1986) is a recursive procedure exploiting rigidity constraints to efficiently search an interpretation tree for the path(s) corresponding to the best sequence(s) of matches. The basic procedure is given in pseudocode in Algorithm 14.3. To correctly handle occlusions (and the fact that, as noted earlier, a range finder will "see," at best, one half of the object facing it), at every stage of the search, the algorithm must consider the possibility that a model plane might not match any scene plane. This is done by always incorporating in the list of potential matches of a given plane a token "null" plane.

Selecting potential matches. The selection of potential matches for a given model plane is based on various criteria depending on the number of correspondences already established, with each new correspondence providing new geometric constraints and more stringent criteria. At the beginning of the search, we know only that a model plane with area A should be matched to scene planes with a com-

FIGURE 14.12: 3D Fax of a statuette of a Buddha. From left to right: photograph of the statuette; range image; integrated 3D model; model after hole filling; and physical model obtained via stereolithography. Courtesy of Marc Levoy. *Reprinted from "A Volumetric Method for Building Complex Models from Range Images," by B. Curless and M. Levoy, Proc. SIGGRAPH, (1996). © 1996 ACM, Inc. http://doi.acm.org/10.1145/237170.237269 Reprinted by permission..*

patible area, i.e., in the range $[\alpha A, \beta A]$. Reasonable values for the two thresholds might be 0.5 and 1.1, which allows for some discrepancy between the unoccluded areas, and also affords a degree of occlusion up to 50%.

After the first correspondence has been established, it is still too early to try and estimate the rigid transformation mapping the model onto the scene, but it is clear that the angle between the normals to any matching planes should be (roughly) equal to the angle θ between the normals to the first pair of planes, say those that lie in the interval $[\theta - \varepsilon, \theta + \varepsilon]$. The normals to the corresponding planes lie in a band of the Gauss sphere, and they can be retrieved efficiently by discretizing this sphere and associating to each cell a bucket that stores the scene planes whose normal falls into it (Figure 14.13).

A second pairing is sufficient to completely determine the rotation separating the model from its instance in the scene: this is geometrically clear (and will be confirmed analytically in the next section) since a pair of matching vectors constrains the rotation axis to lie in the plane bisecting these vectors. Two pairs of matching planes determine the axis of rotation as the intersection of the corresponding bisecting planes, and the rotation angle is readily computed from either of the matches. Given the rotation and a third model plane, one can predict the orientation of the normal to its possible matches in the scene, which can be recovered efficiently using once again the discrete Gauss sphere mentioned before. After three pairings have been found, the translation can also be estimated and used to predict the distance between the origin and any scene plane matching a fourth scene plane. The same is true for any further pairing.

The recursive function Match returns the best set of matching plane pairs found by recursively visiting the interpretation tree. It is initially called with an empty list of pairs and null values for the rotation and translation arguments rot and trans.
The auxiliary function Potential-Matches returns the subset of the planes in the scene that are compatible with the model plane Π and the current estimate of the rigid transformation mapping the model planes onto their scene matches (see text for details).
The auxiliary function Update-Registration-2 uses the matched plane pairs to update the current estimate of the rigid transformation.

```
Function Match(model, scene, pairs, rot, trans);
begin
bestpairs ← nil; bestscore ← 0;
for Π in model do
   for Π′ in Potential-Matches(scene, pairs, Π, rot, trans) do
     rot ← Update-Registration-2(pairs, Π, Π′, rot, trans);
     (score, newpairs) ← Match(model−Π, scene−Π′, pairs+(Π, Π′), rot, trans);
     if score>bestscore then bestscore ← score; bestpairs ← newpairs endif;
     endfor;
   endfor;
return bestpairs;
end.
```

Algorithm 14.3: The Plane-Matching Algorithm of Faugeras and Hebert (1986).

Estimating the rigid transformation. Let us consider a plane Π defined by the equation $\boldsymbol{n} \cdot \boldsymbol{x} - d = 0$ in some fixed coordinate system. Here, $\boldsymbol{n}$ denotes the unit normal to the plane and d its (signed) distance from the origin. Under the rigid transformation defined by the rotation matrix $\mathcal{R}$ and the translation vector $\boldsymbol{t}$, a point $\boldsymbol{x}$ maps onto the point $\boldsymbol{x}' = \mathcal{R}\boldsymbol{x} + \boldsymbol{t}$, and Π maps onto the plane Π' whose equation is $\boldsymbol{n}' \cdot \boldsymbol{x}' - d' = 0$, with

$$\begin{cases} \boldsymbol{n}' = \mathcal{R}\boldsymbol{n}, \\ d' = \boldsymbol{n}' \cdot \boldsymbol{t} + d. \end{cases}$$

Thus, estimating the rigid transformation that maps n planes Π_i onto the matching planes Π'_i $(i = 1, \ldots, n)$ amounts to finding the rotation matrix $\mathcal{R}$ that minimizes the error

$$E_r = \sum_{i=1}^{n} ||\boldsymbol{n}'_i - \mathcal{R}\boldsymbol{n}_i||^2$$

and the translation vector $\boldsymbol{t}$ that minimizes

$$E_t = \sum_{i=1}^{n} (d'_i - d_i - \boldsymbol{n}'_i \cdot \boldsymbol{t})^2.$$

The rotation $\mathcal{R}$ minimizing E_r can be computed, exactly as in Section 14.4.1, by using the quaternion representation of matrices and solving an eigenvector problem.

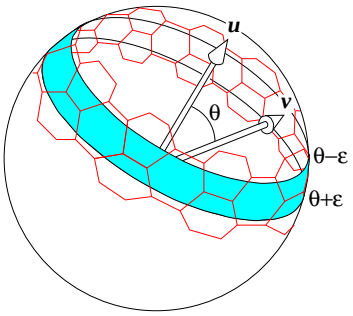

FIGURE 14.13: Finding all vectors $\boldsymbol{v}$ that make an angle in the $[\theta - \varepsilon, \theta + \varepsilon]$ range with a given vector $\boldsymbol{u}$. It should be noted that the unit sphere does not admit tesselations with an arbitrary level of detail by regular (spherical) polygons. The tesselation shown in the diagram is made of hexagons with unequal edge lengths. See, for example, (Horn 1986, Chap. 16) for a discussion of this problem and various tesselation schemes.

The translation vector $\boldsymbol{t}$ minimizing E_t is the solution of a (non-homogeneous) linear least-squares problem, whose solution can be found using the techniques presented in Chapter 22.

Results. Figure 14.14 shows recognition results obtained using a bin of Renault parts such as the one shown in Figure 14.8. The range image of the bin has been segmented into planar patches using the technique presented in Section 14.2.3. The matching algorithm is run three times on the scene, with patches matched during each run removed from the scene before the next iteration. As shown by the figure, the three instances of the part present in the bin are correctly identified, and the accuracy of the pose estimation process is attested by the reprojection into the range image of the model in the computed pose.

14.4.2 Matching Free-Form Surfaces Using Spin Images

As demonstrated in Section 14.2.2, differential geometry provides a powerful language for describing the shape of a surface *locally*, i.e., in a small neighborhood of each one of its points. On the other hand, the region-growing algorithm discussed in Section 14.2.3 is aimed at constructing a *globally* consistent surface description in terms of planar patches. We introduce in this section a *semi-local* surface representation, the spin image of Johnson and Hebert (1998, 1999), that captures the shape of a surface in a relatively large neighborhood of each one of its points. As will be shown in the rest of this section, the spin image is invariant under rigid transformations, and it affords an efficient algorithm for pointwise surface matching, thus completely bypassing segmentation in the recognition process.

Spin image definition. Let us assume as in Section 14.2.3 that the surface Σ of interest is given in the form of a triangular mesh. The (outward-pointing) surface normal at each vertex can be estimated by fitting a plane to this vertex and its neighbors, turning the triangulation into a net of *oriented points*. Given an oriented

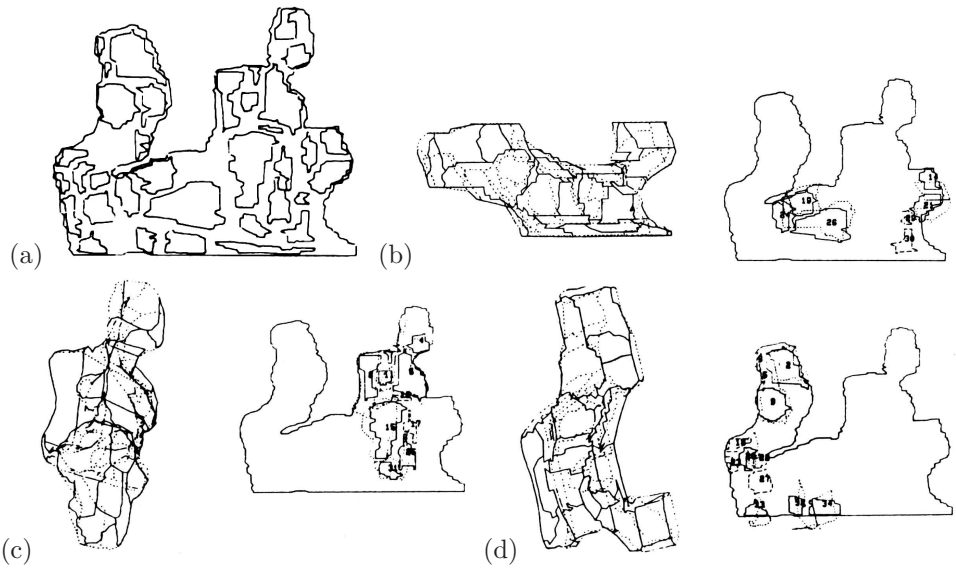

FIGURE 14.14: Recognition results: (a) a bin of parts, and (b)–(d) the three instances of the Renault part found in that bin. In each case, the model is shown both by itself in the position and orientation estimated by the algorithm, as well as superimposed (dotted lines) in this pose over the corresponding planes of the range image. *Reprinted from "The Representation, Recognition, and Locating of 3D Objects," by O.D. Faugeras and M. Hebert, International Journal of Robotics Research, 5(3):27–52, (1986).* © *1986 Sage Publications. Reprinted by permission of Sage Publications.*

point P, the spin coordinates of any other point Q can now be defined as the (non-negative) distance α separating Q from the (oriented) normal line in P and the (signed) distance β from the tangent plane to Q (Figure 14.15). Accordingly, the *spin map* $s_P : \Sigma \to \mathbb{R}^2$ associated with P is defined for any point Q on Σ as

$$s_P(Q) \stackrel{\text{def}}{=} (\underbrace{||\overrightarrow{PQ} \times \boldsymbol{n}||}_{\alpha}, \underbrace{\overrightarrow{PQ} \cdot \boldsymbol{n}}_{\beta}).$$

As shown by Figure 14.15, this mapping is not injective. This is not surprising because the spin map provides only a partial specification of a cylindrical coordinate system: the third coordinate that would normally record the angle between some reference vector in the tangent plane and the projection of $\overrightarrow{PQ}$ into this plane is missing. The principal directions are obvious choices for such a reference vector, but focusing on the spin coordinates avoids their computation, a process that is susceptible to noise since it involves second derivatives and may be ambiguous for (almost) planar or spherical patches.

The *spin image* associated with an oriented point is a histogram of the α, β coordinates in a neighborhood of this point. Concretely, the α, β plane is divided into a rectangular array of $\delta\alpha \times \delta\beta$ bins that accumulate the total surface area spanned by points with α, β values in that range.[3] As shown in Carmichael, Hubert,

[3]The corresponding point sets may actually be divided into several connected components.

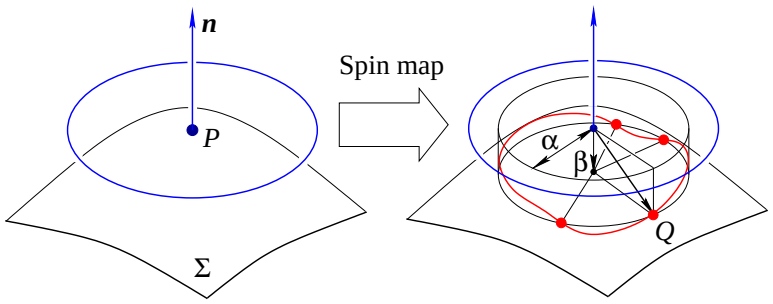

FIGURE 14.15: Definition of the spin map associated with a surface point P: the spin coordinates (α, β) of the point Q are respectively defined by the lengths of the projections of $\overrightarrow{PQ}$ onto the tangent plane and its surface normal. Note that there are three other points with the same (α, β) coordinates as Q in this example.

and Hebert (1999) and the problems, each triangle in the surface mesh maps onto a region of the α, β plane whose boundaries are hyperbola arcs. Its contribution to the spin image can thus be computed by assigning to each bin that this region traverses the area of the patch where the triangle intersects the annular region of $\mathbb{R}^3$ associated with the bin (Figure 14.16). The bins can be found efficiently using *scan conversion* (Foley *et al.* 1990), a process routinely used in computer graphics to find in optimal time the pixels traversed by a generalized polygon with straight or curved edges.

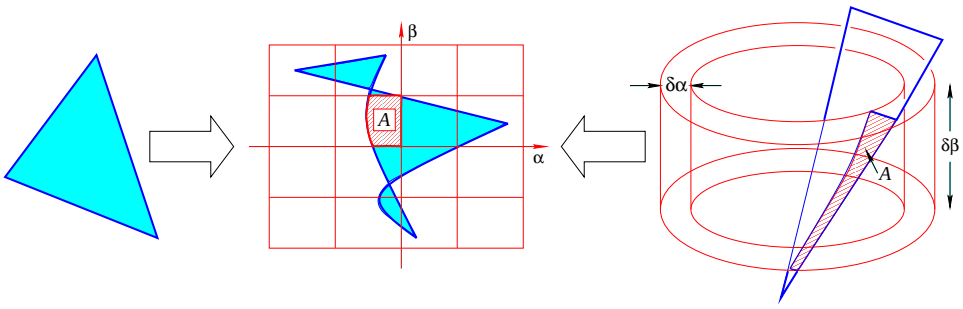

FIGURE 14.16: Spin image construction: the triangle shown in the left of the diagram maps onto a region with hyperbolic boundaries in the spin image; the value of each bin intersected by this region is incremented by the area of the portion of the triangle that intersects the annulus associated with the bin. After Carmichael *et al.* (1999, Figure 3).

Spin images are defined by several key parameters (Johnson and Hebert 1999). The first one is the support distance d that limits to a sphere of radius d centered in P the range of the *support points* used to construct the image. This sphere must be large enough to provide good descriptive power but small enough to support

For example, for small enough values of $\delta\alpha$ and $\delta\beta$, there are four connected components in the example shown in Figure 14.15, corresponding to small patches centered at the points having the same α, β coordinates as Q.

recognition in the presence of clutter and occlusion. In practice, an appropriate choice for d might be a tenth of the object's diameter; thus, as noted earlier, the spin image is indeed a semi-local description of the shape of a surface in an *extended* neighborhood of one of its points. Robustness to clutter can be improved by limiting the range of surface normals at the support points to a cone of half-angle θ centered in $\boldsymbol{n}$. As in the support distance case, choosing the right value for θ involves a trade-off between descriptive power and insensitivity to clutter; a value of 60° has empirically been shown to be satisfactory. The last parameter defining a spin image is its size (in pixels), or equivalently, given the support distance, its bin size (in meters), and it can be shown that an appropriate choice for the bin size is the average distance between mesh vertices in the model. Figure 14.17 shows the spin images associated with three oriented points on the surface of a rubber duck.

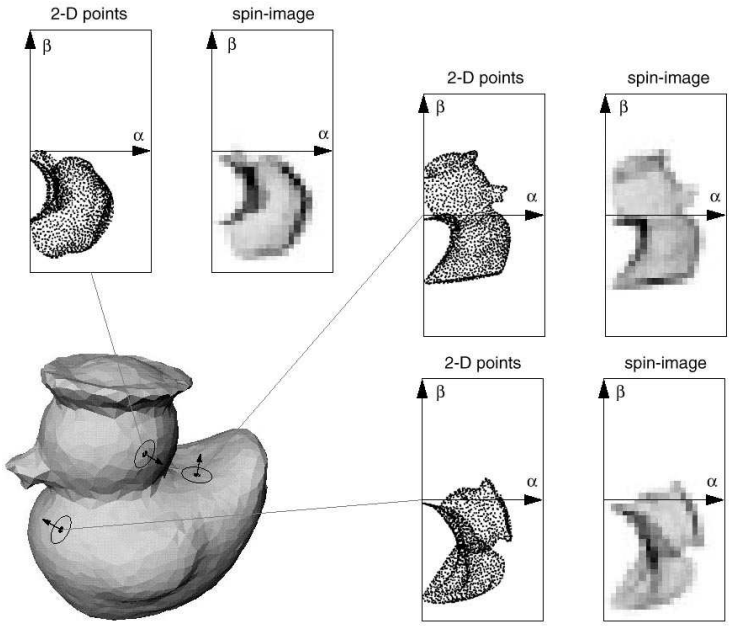

FIGURE 14.17: Three oriented points on the surface of a rubber duck and the corresponding spin images. The α, β coordinates of the mesh vertices are shown besides the actual spin images. *Reprinted from "Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes," by A.E. Johnson and M. Hebert, IEEE Transactions on Pattern Analysis and Machine Intelligence, 21(5):433–449, (1999).* © *1999 IEEE.*

Matching spin images. One of the most important features of spin images is that they are (obviously) invariant under rigid transformations. Thus an image comparison technique such as correlation can in principle be used to match the spin images associated with oriented points in the scene and the object model. Things are not that simple, however: we already noted that the spin map is not injective; in general, it is not surjective either, and empty bins (or equivalently zero-valued pixels) may occur for values of α and β that do not correspond to physical surface

points (see the blank areas in Figure 14.17, for example). Occlusion may cause the appearance of zero pixels in the scene image, whereas clutter may introduce irrelevant non-empty bins. It is therefore reasonable to restrict the comparison of two spin images to their common nonzero pixels. In this context, Johnson and Hebert (1998) have shown that

$$S(\boldsymbol{I}, \boldsymbol{J}) \stackrel{\text{def}}{=} [\text{Arctanh}(C(\boldsymbol{I}, \boldsymbol{J}))]^2 - \frac{3}{N-3}$$

is an appropriate similarity measure for two spin images whose overlap regions contain N pixels and are represented by the vectors $\boldsymbol{I}$ and $\boldsymbol{J}$ of $\mathbb{R}^N$. In this formula, $C(\boldsymbol{I}, \boldsymbol{J})$ denotes the normalized correlation of the vectors $\boldsymbol{I}$ and $\boldsymbol{J}$, and Arctanh denotes the hyperbolic arc tangent function. Armed with this similarity measure, we can now outline a recognition algorithm that uses spin images to establish pointwise correspondences.

Off-line:

Compute the spin images associated with the oriented points of a surface model and store them into a table.

On-line:

1. Form correspondences between a set of spin images randomly selected in the scene and their best matches in the model table using the similarity measure S to rank-order the matches.
2. Filter and group correspondences using geometric consistency constraints, and compute the rigid transformations best aligning the matched scene and model features.
3. Verify the matches using the ICP algorithm.

Algorithm 14.4: Pointwise Matching of Free-Form Surfaces Using Spin Images, after Johnson and Hebert (1998, 1999).

The various stages of this algorithm are mostly straightforward. Let us note, however, that the filtering/grouping step relies on comparing the spin coordinates of model points relative to the other mesh vertices in their group with the spin coordinates of the corresponding scene points relative to their own group. Once consistent groups have been identified, an initial estimate of the rigid transformation aligning the scene and the model is computed from (oriented) point matches using the quaternion-based registration technique described in Section 14.3.2. Finally, consistent sets of correspondences are verified by iteratively spreading the matching process to their neighbors, updating along the way the rigid transformation that aligns the scene and the model.

Results. The matching algorithm presented in the previous section has been extensively tested in recognition tasks with cluttered indoor scenes that contain

both industrial parts and various toys (Johnson & Hebert 1998, 1999). It has also been used in outdoor navigation/mapping tasks with very large datasets covering thousands of squared meters of terrain (Carmichael *et al.* 1999). Figure 14.18 shows sample recognition results in the toy domain.

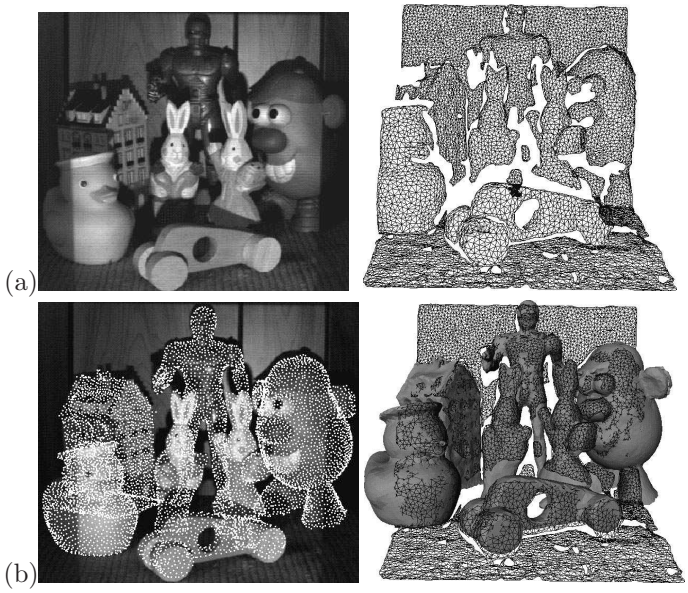

FIGURE 14.18: Spin-image recognition results: (a) a cluttered image of toys and the mesh constructed from the corresponding range image; (b) recognized objects overlaid on the original pictures. *Reprinted from "Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes," by A.E. Johnson and M. Hebert, IEEE Transactions on Pattern Analysis and Machine Intelligence, 21(5):433–449, (1999).* © *1999 IEEE.*

14.5 KINECT

Kinect is a video game technology developed by Microsoft for its Xbox 360 platform that allows its users to control games using natural body motions. It has three main components: a sensor that delivers accurate depth maps and color images at frame rate, an effective algorithm for estimating the pose (joint positions) of the players in every frame, and a tracking algorithm using this information to smoothly recover the parameters (joint angles) of a 3D kinematic model (skeleton) over time. This section discusses the pose estimation algorithm used by Kinect (Shotton *et al.* 2011), which relies on *random forests* to classify individual pixels from a single range image into one of a few predefined body parts, then uses a voting/averaging procedure to compute these parts' locations (joint positions) in 3D.

Kinect is a success story for computer vision, with several million units shipped as of 2011. Before getting into the details of its presentation, it may be worth examining some key elements that might explain (at least in part, and marketing and user-interface issues aside) some of this success:

1. The sensor, developed by Primesense,[4] delivers at 30Hz a depth map with VGA resolution (480×640 pixels) and a registered RGB image with UXGA resolution (1200×1600 pixels). The corresponding *Light Coding*$^{\text{TM}}$ technology uses a projected infrared pattern observed by a black-and-white camera and decoded on a dedicated chip. The two main features of this sensor is that it is *fast*, much faster than conventional range finders using mechanical scanning, and *cheap*—cheap enough, in fact, to ship as part of a mass-market video game package.
2. Range images are a lot easier to simulate realistically than ordinary photographs (no color, texture, or illumination variations). In turn, this means that it is easy to generate synthetic data for training accurate classifiers without overfitting.
3. Voting procedures are relatively robust to errors among individual voters. As shown later in this section, this explains that excellent pose estimation results can be achieved despite relatively large error rates (40%) at the individual pixel level.
4. Kinect's overall effectiveness and robustness are doubtless due in part to its tracking component, whose details are proprietary but, like any other approach to tracking (see Chapter 11), has temporal information at its disposal for smoothing the recovered skeleton parameters and recovering from joint detection errors.

One may also argue that depth map features are more robust, or invariant to viewpoint changes, than those found in photographs. This is certainly true to some extent (see the spin images of Section 14.4.2). On the other hand, one may also argue that, in the context of video games, where the viewpoint does not vary much, the key advantage of these images might be that they readily provide occlusion boundary/silhouette information. Indeed, it is relatively easy to separate objects from background in range images, and all the data processed by the approach to pose estimation presented in the rest of this section is presegmented by a separate and effective background subtraction module.

14.5.1 Features

For efficiency reasons, Kinect uses very simple features that are related to spin images, but without the corresponding tangent plane computations. Instead, they simply measure depth differences in the neighborhood of each pixel. Concretely, let us denote by $z(\boldsymbol{p})$ the depth at pixel $\boldsymbol{p}$ in some range image. Given image displacements $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$, a very simple scalar feature can be computed as

$$f_{\boldsymbol{\lambda},\boldsymbol{\mu}}(\boldsymbol{p}) = z\left[\boldsymbol{p} + \frac{1}{z(\boldsymbol{p})}\boldsymbol{\lambda}\right] - z\left[\boldsymbol{p} + \frac{1}{z(\boldsymbol{p})}\boldsymbol{\mu}\right].$$

In turn, given some allowed range of displacements, one can associate with each pixel $\boldsymbol{p}$ the feature vector $\boldsymbol{x}(\boldsymbol{p})$ whose components are the D values of $f_{\boldsymbol{\mu},\boldsymbol{\mu}}(\boldsymbol{p})$ for all distinct unordered pairs $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ in that range.

[4] `http://en.wikipedia.org/wiki/PrimeSense`.

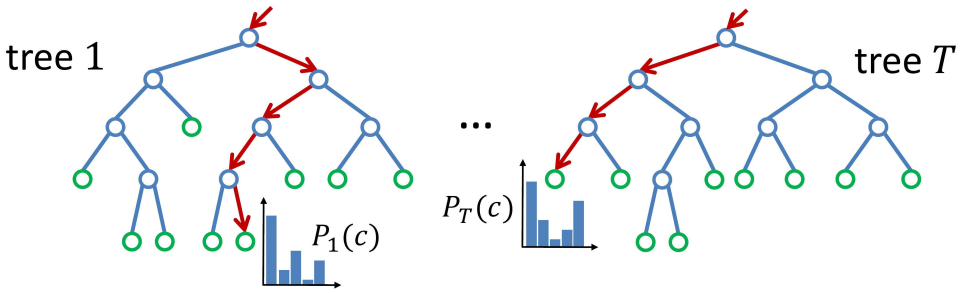

FIGURE 14.19: Random forests. See text for details. *Reprinted from "Real-Time Human Pose Recognition in Parts from Single Depth Images," by J. Shotton* et al.*, Proc. IEEE Conference on Computer Vision and Pattern Recognition, (2011).* © *2011 IEEE.*

As detailed in Section 14.5.3, these features are used to train an ensemble of simple *decision tree* classifiers, in the form of a *random forest*. After training, the feature $\boldsymbol{x}$ associated with each pixel of a new depth map is passed to every tree in the forest, where it is recursively redirected to the left or right descendants of the root according to simple binary tests until it reaches a leaf and is assigned some tree-dependent posterior probability of belonging to each body part (Figure 14.19). The overall class probability of the pixel is finally computed as an average of the tree probabilities. Before detailing this process, let us now present a bit more formally decision trees and random forests.

14.5.2 Technique: Decision Trees and Random Forests

Decision trees. Decision trees have long been used in machine learning and pattern recognition as efficient multi-label classifiers. Let us consider a classification problem with features $\boldsymbol{x} = (x_1, \ldots, x_D)^T$ in $\mathbb{R}^D$ and K different classes. A decision tree is a binary tree where every non-terminal node is associated with some coordinate x_d, with d in $\{1, \ldots, D\}$, and a threshold τ. A feature vector $\boldsymbol{x}$ is assigned to the node's left child if $x_d < \tau$, and to its right child otherwise. This recursive process eventually assigns any feature to some leaf in the tree.

Decision trees split the feature space into hyper-rectangular regions associated with their leaves. Given some labeled training data

$$\mathcal{D} = \{(\boldsymbol{x}_i, y_i),\ \boldsymbol{x}_i \in \mathbb{R}^D\ y_i \in \{1, \ldots, K\},\ i = 1, \ldots, N\},$$

they also classify any unlabeled feature by taking a majority vote among the labeled examples in $\mathcal{D}$ that have reached the same leaf.

Given some fixed tree structure—say, a balanced tree with depth L—training a decision tree amounts to selecting the feature space coordinates and the thresholds associated with its non-leaf nodes. This can be achieved by maximizing at every node the *information gain* associated with the corresponding coordinate x_d and threshold τ.

Intuitively, a decision tree should split any labeled data into subsets that are as homogeneous as possible, and ideally, all data reaching a leaf should have the same label. This can be formalized using the concept of *cross-entropy*. If the

number of points in $\mathcal{D}$ that belong to class k is N_k, its cross-entropy is defined as

$$E(\mathcal{D}) = -\sum_{k=1}^{K} p_k(\mathcal{D}) \log p_k(\mathcal{D}),$$

where $p_k(\mathcal{D}) = N_k/N$ is just the proportion of the points in class k. The cross-entropy reaches its maximum (positive) value of $\log K$ when the data is spread equally into all classes, and reaches its minimum value of zero when all the points belong to the same class. The goal is thus to decrease the cross-entropy as much as possible each time the data is split by a non-terminal node.

The information gain associated with some partition of the data $\mathcal{D}$ into left and right subsets $\mathcal{L}$ and $\mathcal{R}$ is the difference between the original cross-entropy and a weighted sum of the entropies associated with the partition, namely

$$G(\mathcal{D}, \mathcal{L}, \mathcal{R}) = E(\mathcal{D}) - \frac{|\mathcal{L}|}{|\mathcal{D}|} E(\mathcal{L}) - \frac{|\mathcal{R}|}{|\mathcal{D}|} E(\mathcal{R}).$$

Now, given some feature space coordinate x_d and threshold τ, let us define the corresponding left and right subsets of $\mathcal{D}$ as

$$\mathcal{L}_{d,\tau}(\mathcal{D}) = \{(\boldsymbol{x}, y) \in \mathcal{D},\ x_d < \tau\} \quad \text{and} \quad \mathcal{R}_{d,\tau}(\mathcal{D}) = \{(\boldsymbol{x}, y) \in \mathcal{D},\ x_d \geq \tau\}.$$

The information gain associated with d and τ can thus be defined as

$$G_{d,\tau}(\mathcal{D}) = G(\mathcal{D}, \mathcal{L}_{d,\tau}(\mathcal{D}), \mathcal{R}_{d,\tau}(\mathcal{D})),$$

and training a decision tree amounts to picking, for each of its non-terminal nodes, the values of d and τ that maximize $G_{d,\tau}$ for the corresponding subset of the labeled data. This procedure is described in Algorithm 14.5.

The arguments of the recursive procedure TrainDT for its first call are the tree root, 0, and the full dataset. Here, $Node.L$ and $Node.R$ respectively denote the left and right children of $Node$. The tree structure is assumed to be fixed, e.g., a balanced tree.

Procedure TrainDT($Node$,l,$\mathcal{D}$);

1. Find the pair (d, τ) maximizing $G_{d,\tau}(\mathcal{D})$;
2. If $l < L$ then
   - **(a)** TrainDT($Node.L$,$\mathcal{L}_{d,\tau}(\mathcal{D})$,$l+1$);
   - **(b)** TrainDT($Node.R$,$\mathcal{R}_{d,\tau}(\mathcal{D})$,$l+1$).

Algorithm 14.5: Training a Decision Tree.

For small feature space dimensions and labeled datasets, decision trees can be trained efficiently by exhaustively trying all splitting coordinates, and for each one of these, sorting all features. It is normally wise to grow a rather large decision

tree, and then prune it to balance its size with classification accuracy and avoid overfitting (*CART* procedure; see Breiman, Friedman, Ohlsen, and Stone [1984] for details).

As noted earlier, a decision tree classifies a feature vector $\boldsymbol{x}$ by taking a majority vote among the labeled training examples that have reached the same leaf. Alternatively, it is also possible to estimate the posterior probability $P(k|\boldsymbol{x})$ that $\boldsymbol{x}$ belongs to class k as the proportion of labeled samples with class k associated with that leaf.

A typical choice for D^* is $\sqrt{D}$. Adapted from Hastie *et al.* (2009).

1. For $b = 1$ to B do

  (a) Draw a bootstrap sample $\mathcal{D}^*$ from $\mathcal{D}$.

  (b) Grow a decision tree $\mathcal{T}_b$ for $\mathcal{D}^*$ using TrainDT modified such that, at each recursive step, $D^* \leq D$ out of the original D coordinates are picked randomly as splitting candidates;

2. Output the trees $\{\mathcal{T}_b,\ b = 1, \ldots, B\}$.

Algorithm 14.6: Training a Random Forest.

Random forests. A simple method for improving the classification accuracy of decision trees is *bagging* (or *bootstrap aggregation*): Given a dataset $\mathcal{D}$ consisting of N points, a *bootstrap sample* $\mathcal{D}^*$ is formed by randomly drawing N points with replacement from $\mathcal{D}$ (the same point can be drawn several times, and some points present in $\mathcal{D}$ might not appear in $\mathcal{D}^*$). Bagging consists of constructing B bootstrap samples, growing a decision tree for each one of them, and using a majority vote among the trees for classification. This process can be shown to reduce the variance of the prediction when the errors associated with the individual trees are uncorrelated. *Random forests* improve upon bagging by randomly selecting a subset of the input variables at each recursive step of the training process (Algorithm 14.6). The intended effect is to reduce the correlation between the constructed trees, thus reducing the variance of their mean prediction. In practice, as shown in Hastie, Tibshirani, and Friedman (2009) for example, random forests typically do not require pruning, and are easier to train and tune than boosting techniques, with very similar performance for many problems.

After training, a new feature is classified using a majority vote among the trees in the forest. As before, it is also possible to estimate the posterior probability $P(k|\boldsymbol{x})$ that $\boldsymbol{x}$ belongs to class k as the mean of the corresponding probabilities for each tree.

14.5.3 Labeling Pixels

The objective is to construct a classifier that assigns to every pixel in a range image one out of a few body parts, such as a person's face, left arm, etc. There are 10

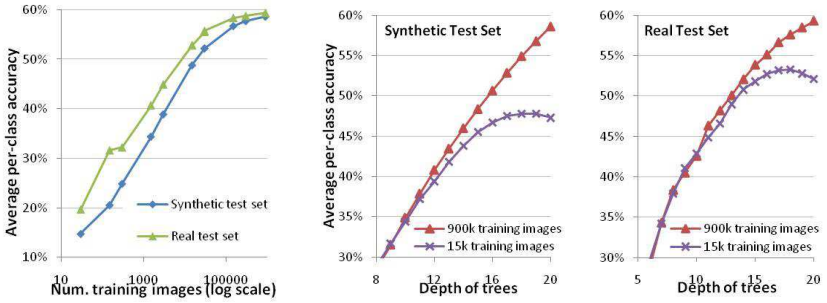

FIGURE 14.20: Effect of the number of (**left**) training images and (**center** and **right**) tree depth on classification accuracy on 5,000 synthetic depth images and 8,808 real hand-labeled ones. Figure courtesy of Jamie Shotton. *Reprinted from "Real-Time Human Pose Recognition in Parts from Single Depth Images," by J. Shotton* et al.*, Proc. IEEE Conference on Computer Vision and Pattern Recognition, (2011).* © *2011 IEEE.*

main body parts in Kinect (head, torso, two arms, two legs, two hands, and two feet), some of which are further divided into sub-parts, such as the upper/lower and left/right sides of a face, for a total of 31 parts. The classifier is trained as a random forest, using the features described in Section 14.5.1 and Algorithm 14.6, but replacing the bootstrap sample used for each tree by a random subset of the training data (2,000 random pixels from each one of hundreds of thousands of training images).

One of the main features of the training process is in fact this data: Its primary source is a set of several hundred *motion capture* sequences featuring actors engaged in typical video game activities such as driving, dancing, kicking, etc. After clustering close-by pictures and retaining one sample per cluster, a set of about 100K poses is obtained. The measured articulation parameters are transferred (*retargeted*) to 15 parametric mesh models of human beings with a variety of body shapes and sizes. Body parts defined manually in texture maps are also transferred to these models (Figure 14.21, top), which are then *skinned* by adding different types of clothing and hairstyle (Figure 14.21, center), and rendered from different viewpoints as both depth and label maps using classical computer graphics techniques (Figure 14.21, bottom).

Hundreds of thousands of labeled images can easily be created in this way. The experiments described in Shotton *et al.* (2011) typically use 2,000 pixels per image and per tree to train random forests made of three trees of depth 20, with 2,000 splitting coordinates and 50 thresholds per node. This takes about one day on a 1,000-core cluster for up to one million training images. Experiments with synthetic and real data show that increasing the size of the training sample improves the classification rate, and suggest that increasing tree depth also helps, at least for large datasets (Figure 14.20): The overfitting observed starting at depth 17 for small datasets of 15K images disappears for the largest datasets with 900K images. The best results are observed with 900K training images and trees of depth 20, with a pixelwise classification rate of about 60%.

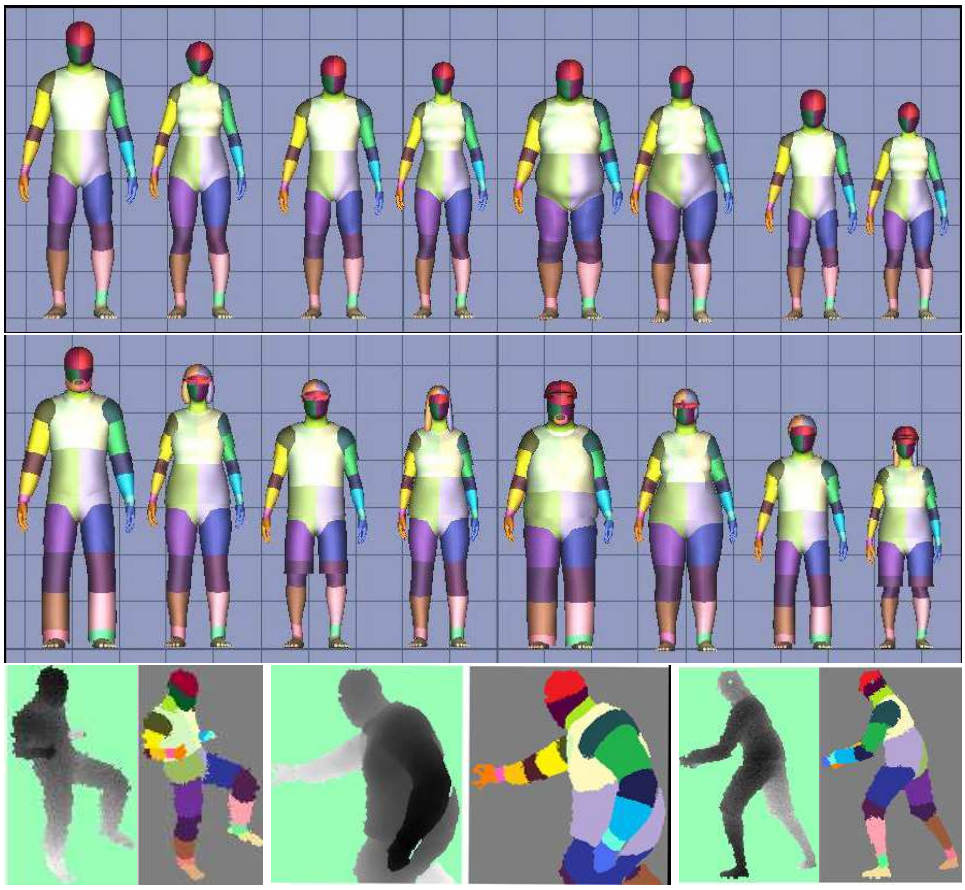

FIGURE 14.21: Data generation process with, from top to bottom: sample models generated by retargeting the motion capture data on meshes corresponding to different body types; models after skinning using different types of clothing and hairstyle; and rendered depth maps together with their labels. *Reprinted from "Real-Time Human Pose Recognition in Parts from Single Depth Images: Supplementary Material," by J. Shotton* et al., *Proc. IEEE Conference on Computer Vision and Pattern Recognition, (2011).* © *2011 IEEE.*

14.5.4 Computing Joint Positions

The classifier described in the previous section assigns to each pixel some body part, but this process does not directly provide the joint positions because there is no underlying kinematic model. Instead, the position of each body part k could (for example) be estimated as some weighted average of the positions of the 3D points corresponding to pixels labeled k, or using some voting scheme. To improve robustness, it is also possible to use mean shifts to estimate the mode of the following 3D density distribution:

$$f_k(\boldsymbol{X}) \propto \sum_{i=1}^{N} P(k|\boldsymbol{x}_i) A(\boldsymbol{p}_i) \exp[-\frac{1}{\sigma_k^2} ||\boldsymbol{X} - \boldsymbol{X}_i||^2],$$

where "$\propto$" stands for "is proportional to," $\boldsymbol{X}_i$ denotes the position of the 3D point associated with pixel $\boldsymbol{p}_i$, and $A(\boldsymbol{p}_i)$ is the area in world units of a pixel at depth $z(\boldsymbol{p}_i)$, proportional to $z(\boldsymbol{p}_i)^2$, so as to make the contribution of each pixel invariant to the distance between the sensor and the user. Each mode of this distribution is assigned the weighted sum of the probability scores of all pixels reaching it during the mean shift optimization process, and the joint is considered to be detected when the confidence of the highest mode is above some threshold. Since modes tend to lie on the front surface of the body, the final joint estimate is obtained by pushing back the maximal mode by a learned depth amount.

Figure 14.22 shows several results obtained on real data. Quantitative results can be obtained by measuring the per-joint precision, measured by counting the proportion of proposals within 0.1m of the true joint positions in hand-labeled depth maps. Experiments using the same synthetic and real data as before shows that the average per-joint precision over all joints and all test images is 0.914 for the real data, and 0.731 for the synthetic one, which is much more challenging due to a great variability in pose and body shape. In realistic game scenarios, the precision of the recovered joint parameters is good enough to drive a tracking system that smoothly and very robustly recovers the parameters of a 3D kinematic model (skeleton) over time, which can in turn be used to effectively control a video game with natural body motions.

14.6 NOTES

Excellent surveys of active range finding techniques can be found in Jarvis (1983), Nitzan (1988), Besl (1989), and Hebert (2000). The model-based approach to edge detection presented in Section 14.2.2 is only one of the many techniques that have been proposed for segmenting range pictures using notions from differential geometry (Fan, Medioni, & Nevatia 1987; Besl & Jain 1988). An alternative to the computational molecules used to smooth a range image in that section is provided by anisotropic diffusion, where the amount of smoothing at each point depends on the value of the gradient (Perona and Malik 1990*c*). The method for segmenting surfaces into (almost) planar patches presented in Section 14.2.3 is easily extended to quadric patches (see Faugeras and Hebert [1986] and the problems). Extensions to higher-order surface primitives is more problematic, in part because surface fitting is more difficult in that case. There is a vast amount of literature on the

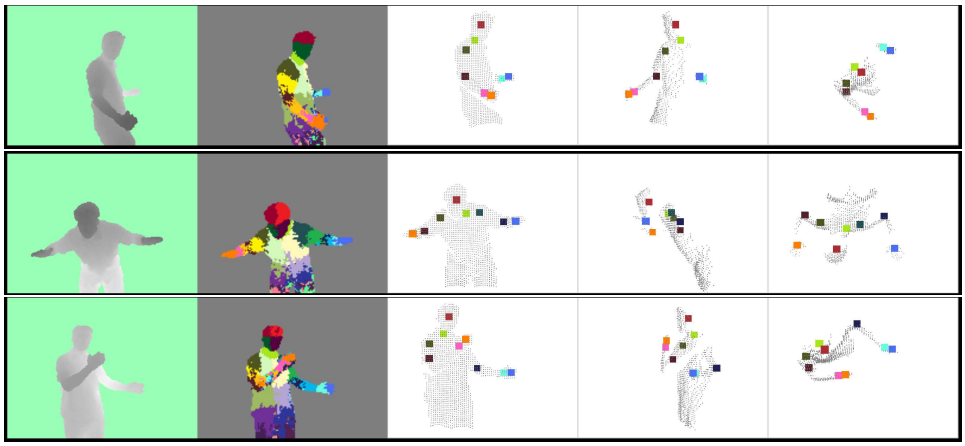

FIGURE 14.22: Sample results with, from left to right, the input depth map, the color-coded classification of pixel into body parts, and renderings of the recovered joint positions from three different viewpoints. *Reprinted from "Real-Time Human Pose Recognition in Parts from Single Depth Images: Supplementary Material," by J. Shotton* et al.*, Proc. IEEE Conference on Computer Vision and Pattern Recognition, (2011). © 2011 IEEE.*

latter problem, using superquadrics (Pentland 1986; Bajcsy & Solina 1987; Gross & Boult 1988) and algebraic surfaces (Taubin, Cukierman, Sullivan, Ponce, & Kriegman 1994; Keren, Cooper, & Subrahmonia 1994; Sullivan, Sandford, & Ponce 1994) for example.

Different variants of the ICP algorithm presented in Section 14.3.2 and Besl and McKay (1992) have been developed over the years, including robust ones capable of handling missing data and/or outliers (Zhang 1994; Wheeler & Ikeuchi 1995), and they have been applied to a number of global registration problems (Shum, Ikeuchi, & Reddy 1995; Curless & Levoy 1996).

Alternatives to the Curless and Levoy (1996) approach to the fusion of multiple range images include the Delaunay triangulation algorithm of Boissonnat (1984), the zippered polygonal meshes of Turk and Levoy (1994), and the crust technique of Amenta *et al.* (1998). The quaternion-based approach to the estimation of rigid transformations described in this chapter was developed independently by Faugeras and Hebert (1986) and Horn (1987*a*). The recognition technique discussed in Section 14.4.1 is closely related to other algorithms using interpretation trees to control the combinatorial cost of feature matching in the two- and three-dimensional cases (Gaston & Lozano-Pérez 1984; Ayache & Faugeras 1986; Grimson & Lozano-Pérez 1987; Huttenlocher & Ullman 1987).

The spin images discussed in Section 14.4.2 have been used to establish pointwise correspondences between range images and surface models. Related approaches to this problem include the structural indexing method of Stein and Medioni (1992) and the point signatures proposed by Chua and Jarvis (1996). The original algorithm described in Section 14.4.2 has been extended in various directions: a scene can now be matched simultaneously to several models using principal component analysis (Johnson and Hebert 1999), and learning techniques are used

to prune false matches in cluttered scenes (Carmichael *et al.* 1999).

Kinect's pose estimation algorithm is detailed in Shotton *et al.* (2011). Decision trees date back to the 1960s, and classical treatments can be found in (Breiman *et al.* 1984; Quinlan 1993). The bootstrap was introduced in Efron (1979), bagging was proposed in Breiman (1996), and random forests in (Amit & Geman 1997; Breiman 2001). See Hastie *et al.* (2009) for a synthesis of these techniques.

PROBLEMS

14.1. Use Equation (14.1) to show that a necessary and sufficient condition for the coordinate curves of a parameterized surface to be principal directions is that $f = F = 0$.

14.2. Show that the lines of curvature of a surface of revolution are its meridians and parallels.

14.3. Step model: compute $z_\sigma(x) = G_\sigma * z(x)$, where $z(x)$ is given by (14.2). Show that z''_σ is given by Equation (14.3). Conclude that $\kappa''_\sigma/\kappa'_\sigma = -2\delta/h$ in the point x_σ where z''_σ and κ_σ vanish.

14.4. Roof model: show that κ_σ is given by Equation (14.4).

14.5. The Rodrigues formula. Consider a rotation $\mathcal{R}$ of angle θ about the axis $\boldsymbol{u}$ (a unit vector). Show that $\mathcal{R}\boldsymbol{x} = \cos\theta\boldsymbol{x} + \sin\theta\,\boldsymbol{u}\times\boldsymbol{x} + (1-\cos\theta)(\boldsymbol{u}\cdot\boldsymbol{x})\boldsymbol{u}$.
Hint: A rotation does not change the projection of a vector $\boldsymbol{x}$ onto the direction $\boldsymbol{u}$ of its axis and applies a planar rotation of angle θ to the projection of $\boldsymbol{x}$ into the plane orthogonal to $\boldsymbol{u}$.

14.6. Use the Rodrigues formula to show that the quaternion $\mathfrak{q} = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}\boldsymbol{u}$ represents the rotation $\mathcal{R}$ of angle θ about the unit vector $\boldsymbol{u}$ in the sense of Equation (14.5).

14.7. Show that the rotation matrix $\mathcal{R}$ associated with a given unit quaternion $\mathfrak{q} = a + \boldsymbol{\alpha}$ with $\boldsymbol{\alpha} = (b, c, d)^T$ is given by Equation (14.6).

14.8. Show that the matrix $\mathcal{A}_i$ constructed in Section 14.3.2 is equal to

$$\mathcal{A}_i = \begin{pmatrix} 0 & \boldsymbol{y}_i^T - \boldsymbol{y}_i'^T \\ \boldsymbol{y}'_i - \boldsymbol{y}_i & [\boldsymbol{y}_i + \boldsymbol{y}'_i]_\times \end{pmatrix}.$$

14.9. As mentioned earlier, the ICP method can be extended to various types of geometric models. We consider here the case of polyhedral models and piecewise parametric patches.

(a) Sketch a method for computing the point Q in a polygon that is closest to some point P.

(b) Sketch a method for computing the point Q in the parametric patch $\boldsymbol{x} : I \times J \to \mathbb{R}^3$ that is closest to some point P. Hint: use Newton iterations.

14.10. Develop a linear least-squares method for fitting a quadric surface to a set of points under the constraint that the quadratic form has unit Frobenius form.

14.11. Show that a surface triangle maps onto a patch with hyperbolic edges in α, β space.

PROGRAMMING EXERCISES

14.12. Implement molecule-based smoothing and the computation of principal directions and curvatures.

14.13. Implement the region-growing approach to plane segmentation described in this chapter.