Computers & Graphics 37 (2013) 137–147

Contents lists available at SciVerse ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Reconstructing shape boundaries with multimodal constraints ☆

Irene Reisner-Kollmann [a,b,*], Stefan Maierhofer [a], Werner Purgathofer [a,b]

[a] VRVis Research Center, Vienna, Austria
[b] Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria

ARTICLE INFO

Article history:
Received 25 October 2012
Received in revised form
28 December 2012
Accepted 2 January 2013
Available online 23 January 2013

Keywords:
Reconstruction
Shape primitives
Shape boundaries

ABSTRACT

Shape primitives are a valuable input for reconstructing 3D models from point clouds. In this paper we present a method for clipping simple shape primitives at reasonable boundaries. The shape primitives, e.g. planes or cylinders, are 2D manifolds which are automatically detected in unstructured point clouds. Shape boundaries are necessary for generating valid 3D models from multiple shape primitives, because shape primitives possibly have dimensions of infinite extent or they are only partially present in the scene. Hints for reasonable boundaries of shape primitives are indicated by different input sources and constraints. Point clouds and range images provide information where shape primitives coincide with measured surface points. Edge detectors offer cues for surface boundaries in color images. The set of shape primitives is analyzed for constraints such as intersections. Due to an iterative approach, intermediate results provide additional constraints such as coplanar boundary points over multiple shape primitives. We present a framework for extracting and optimizing shape boundaries based on the given input data and multiple constraints. Further, we provide a simple user interface for manually adding constraints in order to improve the results. Our approach generates structurally simple 3D models from shape primitives and point clouds. It is useful for reconstructing scenes containing man-made objects, such as buildings, interior scenes, or engineering objects. The application of multiple constraints enables the reconstruction of proper 3D models despite noisy or incomplete point clouds.

1. Introduction

Recent advances in scanning technologies allow the acquisition of point clouds from a large variety of scenes. Laser scanners are able to capture point clouds of real-world scenes, ranging from small objects to whole cities. Photogrammetric tools generate dense point clouds from a set of images [1], which works well for outdoor scenes. For interior scenes, Microsoft Kinect provides a cheap and simple possibility to generate point clouds [2]. Although capturing methods have different advantages and drawbacks, the generated point clouds usually share the same problems of noisy and missing data, e.g. due to a limited area of capturing viewpoints.

These artifacts make it very difficult to apply direct surface reconstruction methods [3,4], where the point cloud is approximated by a large number of triangles. It has been shown that shape priors are a valuable input for surface reconstruction [5,6]. Some scenes, such as engineering objects or interior rooms, can be represented to a large extent with only a few shape primitives. Reconstructing a point cloud with a set of primitive shapes provides advantages for many purposes. The resulting 3D model has a low number of vertices, it is well-defined and can be easily edited. The shapes provide further knowledge about the scene which can be used for automatically analyzing and editing the structure of a scene. For example, it is easier to separate an interior scene into walls and furniture based on primitive shapes than in a large set of triangles.

Many shapes, e.g. planes or cylinders, have an infinite extent in one or two dimensions. Also, surfaces do not always occupy the full extent along finite dimensions. Thus, generating a valid 3D model requires an additional step where the boundaries of the shapes are extracted. Most algorithms so far deal only with the detection of shapes, and create the final 3D model by intersecting nearby shapes [7], or combining all shapes to a closed model [8]. These approaches may fail if a scene is only partially reconstructed, contains large holes, or some shapes have been wrongly detected due to noise. In case of thin objects, such as table tops or doors, it is usually not possible to robustly estimate the thin sides just from a noisy point cloud. Especially in interior scenes, point clouds can only be captured from a limited space of viewpoints. Hence, only the front faces of an object are well defined by the point cloud.

In this paper we present a new algorithm for finding reasonable boundaries of shape primitives. We provide a generic optimization framework which works for a wide range of input

☆ This article was recommended for publication by Andrei Sharf.
* Corresponding author at: VRVis Research Center, Donau-City-Strasse 1, 1220 Wien, Austria.
E-mail address: irene@vrvis.at (I. Reisner-Kollmann).

0097-8493/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cag.2013.01.001

sources. The system allows to automatically create well-defined models from a set of shapes by incorporating high-level information. Additionally, the optimization framework is also open for user input in order to interactively edit the reconstructed model.

1.1. Input data

Our reconstruction system requires a point cloud as input, which can be obtained by different techniques such as laser scanners, range cameras or photogrammetry. Optionally, information from depth maps or color images can be included, provided that they are registered to the point cloud.

In our initialization step, shape primitives are fit to the point cloud with a RANSAC approach [5]. If the point cloud is represented by multiple range images, we use a graph-based initialization to handle the large amount of redundant data [9]. Optionally, the shapes are optimized with GlobFit [7] in order to have parallel directions as well as equal distances and angles.

The optimization framework uses a set of shape primitives together with the segmentation of the point cloud. It is necessary that the shape primitives are parameterized in a two-dimensional map. In our implementation we use planes, cylinders and cones, but the optimization framework and most of the proposed constraints can also be extended to other shapes, like spheres and tori (Fig. 1).

1.2. Related work

Many reconstruction algorithms assume that models can be completely reconstructed and shape boundaries are implicitly generated by creating closed meshes [8]. In this case, shape boundaries originate from intersections between neighboring shapes. This is the preferred approach for fully reconstructed data, but shapes may be wrongly connected in case of too much missing data.

A simple approach for finding boundaries for a single shape is to rasterize a point cloud in the 2D domain of a shape [5]. If color images are available, the uniform raster grid can be replaced by a grid formed from strong images lines [10]. While this extension can fill holes from missing data, it is mainly useful for buildings or other objects containing rectilinear structures.

Alpha shapes [11] are an extension of the convex hull for generating concave shapes. A global parameter defines the granularity of cavities. The l1-sparse reconstruction method [12] creates piecewise smooth boundaries that preserve sharp features. Given a set of oriented points, the algorithm starts with aligning nearby point orientations, which are then used to recover consistent positions. Chen and Chen [13] detect boundary points of a two-dimensional point set and cluster them to line segments. Similar to rasterization, artifacts from the original point cloud, such as missing data, will be transferred to the final shape boundaries.

Jenke et al. [14] propose an optimization algorithm for shape boundaries based on similar input data to ours, namely a point cloud segmented into shape primitives. Boundary points are extracted for each shape and iteratively optimized. The boundary points are attracted to nearby shapes while they keep a low distance to points assigned to their original shape and neighboring boundary points are smoothed. The algorithm generates sharp edges between nearby shapes and smooth boundaries where no neighboring shapes are available. In contrast to our algorithm, no additional constraints from other input data are included.

Some reconstruction algorithms have strict assumptions on the model, which simplifies the task for finding shape boundaries. Manhattan world reconstructions assume that scenes can be modeled with axis-aligned planes [15,16]. Furukawa et al. [17] use the Manhattan world assumption for reconstructing a scene from multiple images. Shape boundaries, i.e. intersections between planes, are optimized such that they are aligned with strong image edges. Sinha et al. [18] reconstruct piecewise-planar 3D models from images and provide a user interface for editing the polygonal outline of a planar shape. The modified vertices and edges are snapped to main directions given by vanishing points in the input images. Schindler and Bauer [19] reconstruct buildings from images by first creating a coarse piecewise-planar model, and then fitting pre-defined shapes for windows and doors. Arikan et al. [20] present an interactive optimization system for fitting planar polygons to point clouds and snapping neighboring polygon elements together. The user can sketch modifications on the model, which are exactly computed by the optimization system.

In Section 3 we present possible constraints for our optimization framework. We use similar ideas as presented in previous work, such as intersection lines or rasterization. The main advantage of our algorithm is its flexibility in combining different constraints depending on input data and scene properties.

2. Optimization

The primary goal of our optimization framework is to find good surface boundaries for shape primitives, i.e. dividing shapes into *inside* and *outside* areas. This task is very similar to image segmentation techniques, where an image is divided into foreground and background. Inspired by popular image segmentation algorithms [21], we solve the problem of shape boundary optimization with graph cuts.

We parameterize the surface of a three-dimensional shape primitive in a two-dimensional space and uniformly rasterize it into a set of pixels I. The binary segmentation of the 2D space is obtained by minimizing the cost function defined in Eq. (1), containing a regional and a boundary term:

$$E = \sum_{p \in I} R(p) + \sum_{(p,q) \in N} B(p,q) \cdot \delta(A(p) \neq A(q)) \quad (1)$$

The result of the minimization is a labeling $A(p)$, where each pixel p is marked either *inside* or *outside*. $p \in I$ denote all pixels in the 2D shape parameterization, N contains all pairs of neighboring pixels under a standard 4-neighborhood system. Function δ denotes the Potts model, which returns 1 if the argument is true and 0 otherwise.

The regional term $R(p)$ provides penalties on labeling a pixel inside (R_{in}) or outside (R_{out}). The boundary term $B(p,q)$ puts a

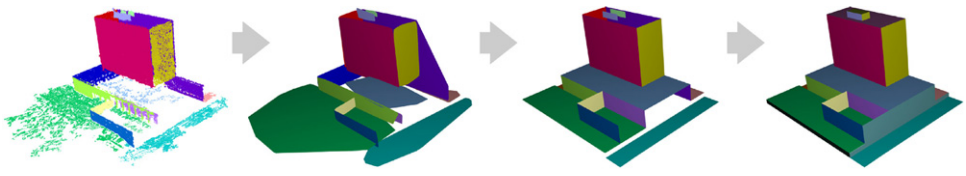

Fig. 1. Reconstructing a point cloud with shape primitives (from left to right): segmented input point cloud, detected shapes bounded by convex hull, optimized shape boundaries, additional shapes due to coplanarity.

penalty on a discontinuity between the pixels p and q, i.e. when they are assigned to different labels. Boundary constraints will set the boundary term to a low value, when the line between neighboring pixels is a good position for the shape boundary.

Performing the optimization in a discrete space allows the application of an efficient graph-cut algorithm [22]. However, discretization introduces the problem of rasterization artifacts. We successfully handle this problem by automatically snapping grid-aligned boundaries to input constraints (see Section 2.2) after the binary segmentation. The lower limit of the resolution is defined by the minimum size of features that should be included in the reconstruction. The upper limit depends on the maximum amount of time as an increased resolution leads to longer optimization times.

The optimization is applied only to a limited space of the shape defined by an oriented bounding box. The boundary will be wrongly clipped if the bounding box is selected too small, while a large bounding box will simply increase computational time. In our experiments, we found a good compromise by initializing it as the bounding box of the initial point cloud assigned to a shape, enlarged by 20%. After each optimization step, the bounding box is extended according to the new shape boundary if necessary. The size can also be manually increased in our interactive user interface.

Each shape is optimized separately, but many constraints are computed based on information from the whole model. For example, intersections are computed between pairs of shapes and each 3D intersection curve is projected onto two shapes. Thus, the shapes are optimized locally under global constraints. Some constraints depend on the current shape boundary, e.g. the coplanarity constraint tries to find dominant planes in all shape boundaries. Therefore, multiple optimization steps are executed, interleaved with an update of all constraints, as illustrated in Fig. 2.

2.1. Constraints

A wide range of input sources can be used as constraints for the boundary optimization. For example, the final polygon should contain all points used for estimating the shape, the boundary should match edges in images, or parallel lines are preferred (see Section 3 for detailed information). These constraints either influence the regional term $R(p)$ or the boundary term $B(p,q)$ of Eq. (1).

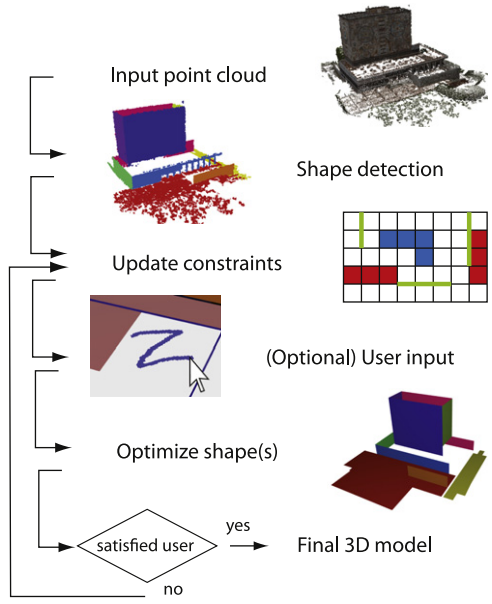

Fig. 2. Overview of our reconstruction pipeline: based on an input point cloud our system performs automatic shape detection and then shape boundaries are optimized iteratively until the user is satisfied with the results.

Regional constraints define which pixels should be labeled as inside respectively as outside. They provide two grayscale images, which correspond to the costs for labeling pixels as inside (R_{in}) or as outside (R_{out}).

Boundary constraints provide good locations for the shape boundary, i.e. for changing between inside and outside. These locations are stored as a set of line segments. The line segments are rasterized for the optimization, but their accuracy is independent from the shape resolution, which is essential for avoiding rasterization artifacts (see Section 2.2).

Multiple constraints can be used at the same time by summing up the individual costs. Using multiple constraints leads to

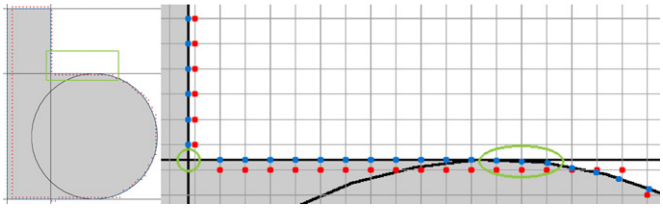

Fig. 3. Snapping the computed boundary to original constraint lines. The original boundary is located on the rasterization grid (red sample points). The final boundary is snapped to nearby constraint lines (blue sample points). On the left side (green circle) a corner point will be inserted, while at the right side (green ellipse) a smooth transition between the constraints is achieved. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

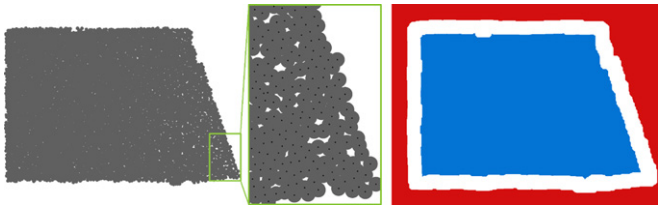

Fig. 4. Left: 3D points are projected onto the 2D parameterization of a shape. Neighboring points are connected due to the automatically computed point size as can be seen in the detailed view. Right: The costs based on the projected points are visualized. Blue pixels should be labeled inside (high value for R_{out}) and the red pixels outside (high value for R_{in}). The boundary will be located within the white area. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

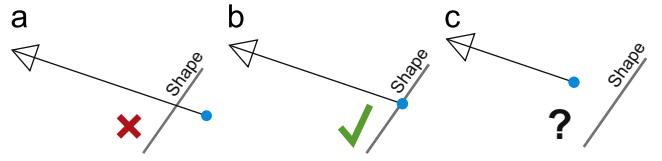

Fig. 5. A depth map provides one of three possible constraints on each shape pixel: (a) pixel should be labeled outside due to visibility of more distant point, (b) pixel should be labeled inside due to coincidence of depth value and shape position, (c) no constraint because the shape is behind another object. The gray line denotes the shape, the blue point denotes the measured point in the depth map, the triangle visualizes the position of the depth camera. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

conflicts when they vote for different labels. For this reason it is necessary to make some constraints more important than others. These conflicts can be easily solved by weighting the constraints with different impact factors which are editable by the user. The relative differences between impact factors are the important information, not the absolute values. For example, constraints from user input are always weighted higher than other constraints.

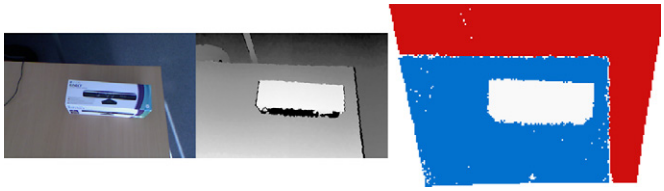

Fig. 6. Left: Input depth map and color image. Right: Visualization of costs for the plane representing the table top. The blue area is inside the plane surface while the red area is outside. For the white area no information is provided by the depth map. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

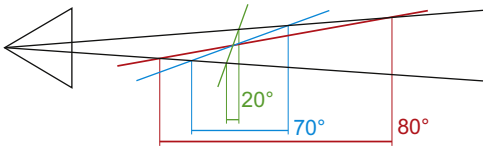

Fig. 7. For surfaces with normals that form an obtuse angle with the view direction of a range camera, no accurate measurements can be taken from the depth map. The corresponding pixels in the depth map would cover a large area on the surface, as can be compared between the red and green surfaces. We prune all measurements with normals having an angle larger than 70°. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

2.2. Rasterization artifacts

The pixel-based approach introduces the problem of rasterization artifacts. The polygon positions are aligned along the main directions of the rasterization grid and hence they are not exactly positioned on the constraint lines. For scenes with planar surfaces a lot of artifacts can be removed by aligning the local coordinate systems on planes with the major directions of the scene. A higher resolution would reduce the rasterization errors, but at the cost of higher computation times. We propose a constraint snapping algorithm for generating exact boundaries for all types of shapes which also works well for low resolutions (see Fig. 3), as follows:

The original line segments from all boundary constraints, which generally have a higher precision than the rasterization grid, are taken as input. The polygon created from the rasterization grid is sampled with the desired output tesselation. The sampling distance defines the precision of the final polygons and has to be smaller than the tesselation of curved objects. A graph cut optimization is applied for aligning the sample points with nearby constraint lines.

The cost for assigning a sample point p to a constraint line c is defined in Eq. (2). It is based on the distance between the point and line dist(p,c), the constraint line's weight w_c and the angle between the direction of the polygon at point p (dir_p) and the constraint line direction (dir_c). This has the effect that points are snapped to nearby lines with corresponding directions and that lines with higher weights are preferred. For sample points that are not snapped to a constraint line, we choose a constant cost value of 2.25. This means that constraint lines with highest weight and perfect direction have a maximum distance of 1.5 pixel for having a lower cost value than leaving a point on its original position:

$$E(p,c) = \mathrm{dist}(p,c)^2 + \exp\left(\frac{-w_c^2}{(w_{max}/4)^2}\right) + (1 - \langle dir_p, dir_c \rangle) \quad (2)$$

Neighboring sample points are preferably assigned to the same constraint line, especially to avoid oscillating between two nearby

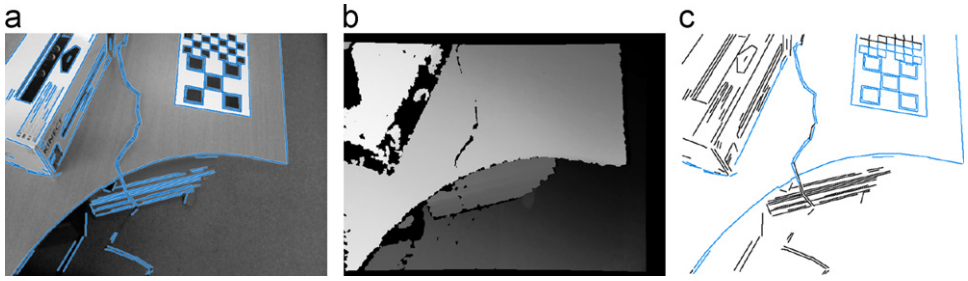

Fig. 8. These images illustrate, which line segments are extracted and accepted from one image for a planar shape representing the table top. (a) shows all detected lines in the color image, (b) is the depth image, (c) displays accepted line segments in blue. Line segments that originate from surfaces above or beneath the table top are not used as constraint. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

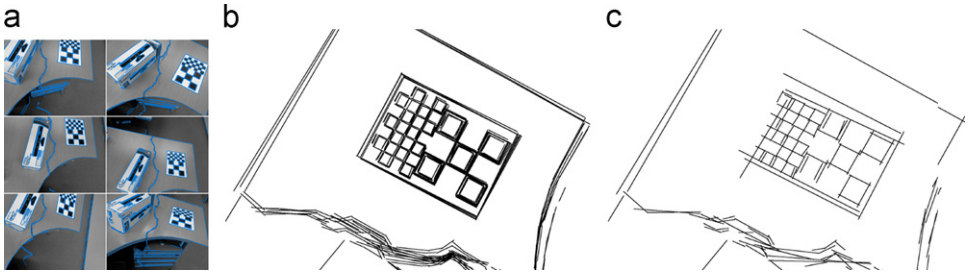

Fig. 9. Line segments from six input frames (a) have been projected onto a plane shape (b). They do not overlap exactly due to inaccuracies, therefore nearby line segments are merged (c).

constraint lines. Additionally, the transition between two different constraints should be located where the constraint lines are closest to each other. Eq. (3) shows the cost function for assigning point p_i to constraint c_k and point p_j to c_l. The function clp(p,c) returns the closest point on line c to point p:

$$E(p_i,p_j,c_k,c_l) = \|\mathrm{clp}(p_i,c_k)-\mathrm{clp}(p_i,c_l)\| + \|\mathrm{clp}(p_j,c_k)-\mathrm{clp}(p_j,c_l)\| \quad (3)$$

The optimization snaps point to nearby constraint lines and provide smooth transitions between different constraints. Corners are extracted from intersections between constraint lines or neighboring lines with sharp angles. The corners are inserted into the final polygon between the closest points on the according constraint lines.

3. Constraints

In this section we present some example constraints that can be used by our optimization framework. Constraints are either generated from input data such as point clouds or images or they are derived from the shapes themselves or from the current shape boundaries.

Generally, the input data for detecting shapes is included into the optimization. This is either a point cloud (Section 3.1) or a set of oriented depth maps (Section 3.2), which additionally provide information on occlusions over point clouds. Intersection constraints (Section 3.3) are important for generating closed meshes where the individual shape boundaries exactly meet each other. Optionally, edges from color images can be included, when images are available and contain visible object boundaries (Section 3.4). We present two constraints, which analyze the current shape boundaries for coplanar segments (Section 3.5) respectively for dominant two-dimensional shapes (Section 3.6). These constraints are often combined with manual inputs (Section 3.7), where the user pushes the boundary towards the favored shape which is then exactly computed by other constraints.

In our system individual constraints provide costs in the range of zero to one. This guiding principle facilitates the task of assigning different impact factors to constraints in order to favor important constraints over others (see Section 2.1).

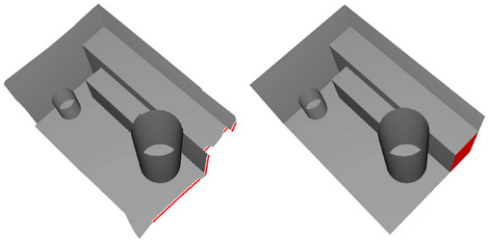

Fig. 10. Left: Input model with coplanar sample points highlighted in red. Right: Result after optimization with coplanarity constraint. The detected plane has also been added to the model and is shown in red. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

3.1. Point cloud

In our reconstruction system, shapes are initialized from a segmented point cloud, i.e. a subset of points is assigned to each shape. An obvious constraint is that these points are located inside the shape boundary. The points are projected onto the two-dimensional bitmap of the shape. The point size is automatically computed by the average point density such that neighboring points touch each other (see Fig. 4).

Small holes in the initial bitmap are removed by applying a morphological close-operation. The border is not well-defined by the projected points and will usually be optimized by boundary constraints. Therefore, the bitmap is eroded by a few pixels to relax the point cloud constraint at the boundary. The default

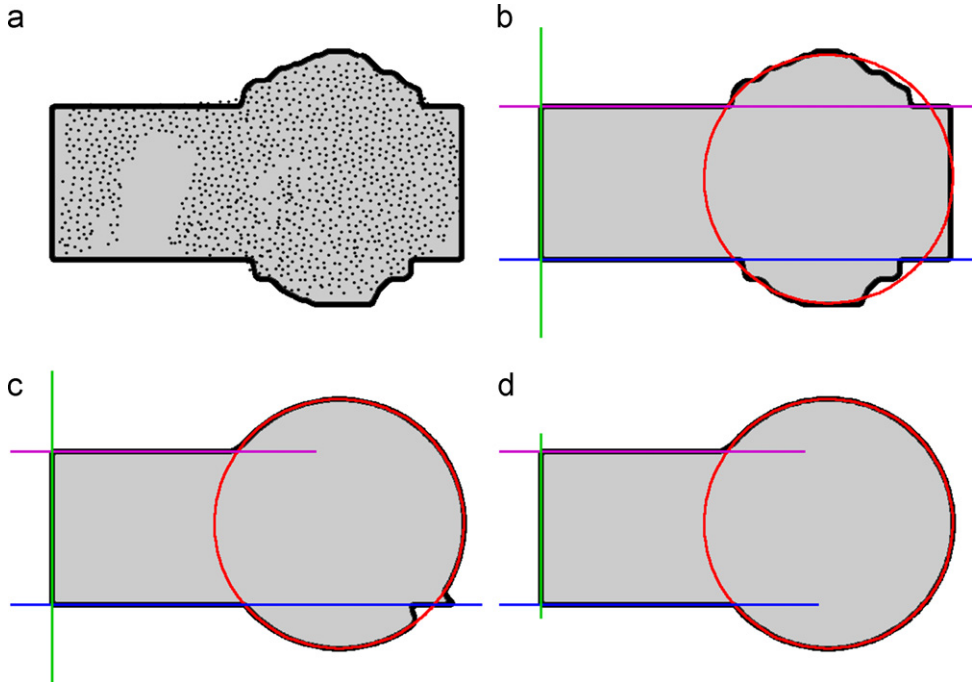

Fig. 11. (a) shows the input point cloud and the boundaries of the table in Fig. 18 after the initial optimization step according to the point cloud and intersection lines. In (b) the colored lines are constraints calculated from the initial boundaries. Next, in (c), a new boundary is computed from the point cloud using the constraints from (b). After another constraint calculation and optimization step, (d) shows an improved result. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

value is set to three pixels, but can be adapted by the user. If the assigned point set has a low confidence, e.g. due to large noise, better results can be achieved with a larger erosion. On the other hand, if the final reconstruction should stick close to the assigned point set, a smaller erosion is chosen. In the final bitmap, all pixels which should be labeled inside the shape have the value one. Thus, the values can be directly used as cost R_{out}.

The cost for labeling pixels inside R_{in} is on the other hand not clearly defined, because parts of the surface might be missing in the point cloud, e.g. due to occlusion. Nevertheless, it is necessary to define parts as being outside of the shape. Otherwise, the shape will be extended to the whole bounding box as long as no other constraints are defined.

We take the inverse of the initial bitmap and erode it by the same amount of pixels as defined above. This bitmap is used as R_{in}, but it is weighted much lower. In our experiments we use a constant value of 0.1 for labeling these pixels as inside. This value is sufficient for restricting the shape boundary when no other constraints are used, but it does not have much influence in case of more confident constraints.

3.2. Depth maps

Multiple depth maps provide the same information as point clouds when they are available together with their camera information. Additionally, it is possible to compute where a shape has been occluded by other geometry. Conversely, it can be determined, where a shape is not valid because it would have been visible in any camera. The maximum cost value is used when these areas are labeled as inside.

Each depth map provides three possibilities for each pixel in the shape parameterization, which are illustrated in Figs. 5 and 6. The pixel has to be outside of the surface, when the corresponding 3D position is inside the depth camera's view frustum, but a more distant point has been captured by the depth camera. R_{in} is set to the maximum value 1 for this pixel. The pixel is inside the surface when the corresponding 3D position approximately coincides with the measured value in the depth map. R_{out} is set to the maximum value 1 for this pixel. No constraint can be created when the shape position is occluded by a closer point in the depth map or when it is not projected onto a depth map pixel with a valid measurement. Both regional terms, R_{in} and R_{out}, are set to zero in this case to indicate there is no constraint.

When multiple depth maps are available, constraints are created as soon as they occur in one depth map. Care has to be taken for areas that are nearly parallel to the view direction of the depth map. These pixels cover a large area on the shapes and thus do not provide accurate measurements (see Fig. 7). Therefore we apply a simple pruning step and consider only depth map pixels whose normals form a maximum angle of 70° with the camera direction. Usually, only a few pixels are affected by these pruning step which can be compensated by more accurate measurements from other range images.

3.3. Shape intersections

Intersections with nearby shapes provide very strong cues for surface boundaries. Intersection lines are computed for each pair of shapes and projected into 2D space. The lines have to fall at least in one of the shapes' bounding boxes in order to be accepted. The intersection lines are weighted inversely proportional to the distance between the shapes such that neighboring shapes will be preferred.

The usage of shape intersections is necessary for generating closed models. Additionally, more distant shapes also provide useful information, e.g. when parts of a shape are missing in the point cloud.

3.4. Images

Color images provide valuable information about the location of surface boundaries. For this it is necessary that an object is clearly distinguishable from its background, e.g. by luminance, color or texture. Unfortunately, such changes often occur inside an object too, which makes it hard to extract edges. Therefore we apply some additional conditions to avoid cluttering from too many image edges.

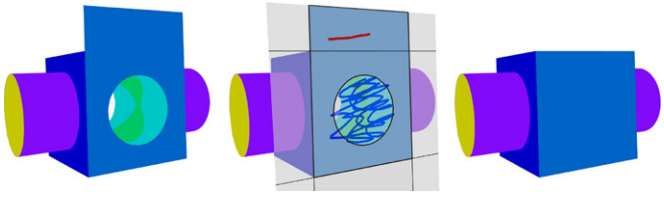

Fig. 12. User input: red pixels mark areas as outside, and blue as inside. The right image shows the result after another optimization step. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

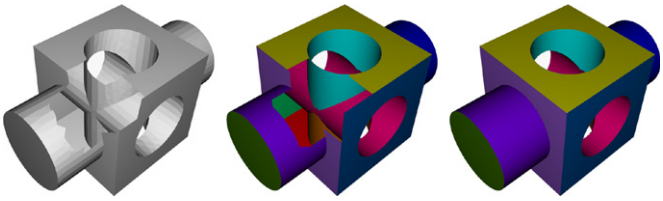

Fig. 13. Reconstruction of block shape with missing data. Left: original mesh used for sampling. Middle: reconstruction with equal impact factors for point cloud and intersections. Right: high impact factor for intersections.

Table 1
The table shows the computation times (in seconds or minutes) for shape detection and fitting (T_{pre}), initial automatic boundary optimization (T_{init}), and complete user input (T_{ui}). Additionally, the times for updating constraints (T_{cstr}) and optimizing shapes (T_{opt}) are given. Finally, some statistics on the input and output data are given, namely the number of Kinect RGB-D images (N_{frames}) or the number of input points (N_{points}), as well as the number of shape primitives before and after user input (N_{shapes}), the average number of raster points per shape primitive (N_{cells}), and the average number of optimization steps per shape primitive (N_{steps}).

Dataset	T_{pre} (s)	T_{init} (s)	T_{ui} (m)	T_{cstr} (s)	T_{opt} (s)	N_{frames}	N_{points}	N_{shapes}	N_{cells}	N_{steps}
Kitchen 1 (Fig. 16)	26	1.3	6.3	36.2	5.9	8		12/17	21 763	5.4
Kitchen 2 (Fig. 17)	57	7.6	12.7	105.2	10.8	13		16/26	32 765	6.2
Bar table (Fig. 18)	51	1.3	3.1	4.5	1.9	5		7/9	8937	6.1
Church (Fig. 19)	53	20.1	12.9	124.6	15.2		29 600	18/25	36 139	6.5
Lans le Villard (Fig. 19)	178	27.9	16.0	230.6	22.3		44 187	27/32	43 758	5.6
Ballroom (Fig. 19)	528	36.3	26.9	247.2	50.5		61 699	44/40	44 978	5.5

We use a state-of-the-art line segment detector [23] for detecting line segments in all input color images. The restriction to line segments is useful as we focus on interior scenes and other man-made objects where mainly straight line segments occur. Nevertheless, other shape boundaries are approximated by concatenating multiple lines, provided their visual cue is strong enough (e.g. the table in Fig. 8(a)).

Another restriction is based on depth maps registered with the color images, which are available when an RGB-D sensor was used for capturing. For each shape, only line segments which are located on the shape's surface are used as constraint. Line segments where the associated depth values do not meet the shape are removed (see Fig. 8). Lines along surface boundaries are usually located along large depth discontinuities, but usually they are slightly off due to registration or rasterization errors. Therefore, it is important to look for corresponding depth values in a slightly increased test area. In our system we empirically set this offset to 1.5 pixels which works well for depth maps captured with Microsoft Kinect.

Line segments are detected in all available color images and projected onto the 2D parameterization of a shape. Due to inaccuracies in camera pose estimation, shape detection and line extraction, lines are usually projected with small offsets next to each other. Therefore, we merge nearby lines with a line segment clustering algorithm [24], as can be seen in Fig. 9.

3.5. Coplanarity

Coplanarity provides boundary constraints in areas where points from multiple shape boundaries are located approximately on the same plane. This plane provides a joint limit in one direction for multiple shapes. Such limits are valuable for incomplete models, e.g. to restrict walls of an interior room where the ceiling has not been scanned. Furthermore, the detected planes are often useful for extending the existing model.

The coplanarity constraint is computed based on the current set of shape boundaries and is updated after each optimization step. We restrict the search to planes perpendicular to global directions of the model, e.g. to plane normals or cylinder axes. This restriction is based on the assumption that shapes in man-made objects are often positioned with orthogonal angles to each other [7]. Otherwise, undesired constraint lines would affect the optimization because planes are detected in unrelated shape boundaries. Sample points are extracted together with their line directions from all current 3D shape boundaries. For each global direction, all sample points with a perpendicular direction are selected. The points are projected to one-dimensional ray coordinates along the global direction. The ray coordinates are clustered in order to find dominant planes.

The clusters are accepted only if the points originate from at least two different shapes and if the points are not approximately collinear. Clusters which are consistent with already existing planes in the model are also removed. The final planes are moved to the median value of all ray coordinates. This ensures that the plane is snapped to dominant boundaries, since averaging over all ray coordinates would be influenced too much by outliers. Fig. 10 shows an example for optimizing a model with coplanarity constraints.

3.6. 2D shapes

Similar to finding shapes in 3D, local boundaries can be enhanced by approximating them with two-dimensional shapes such as

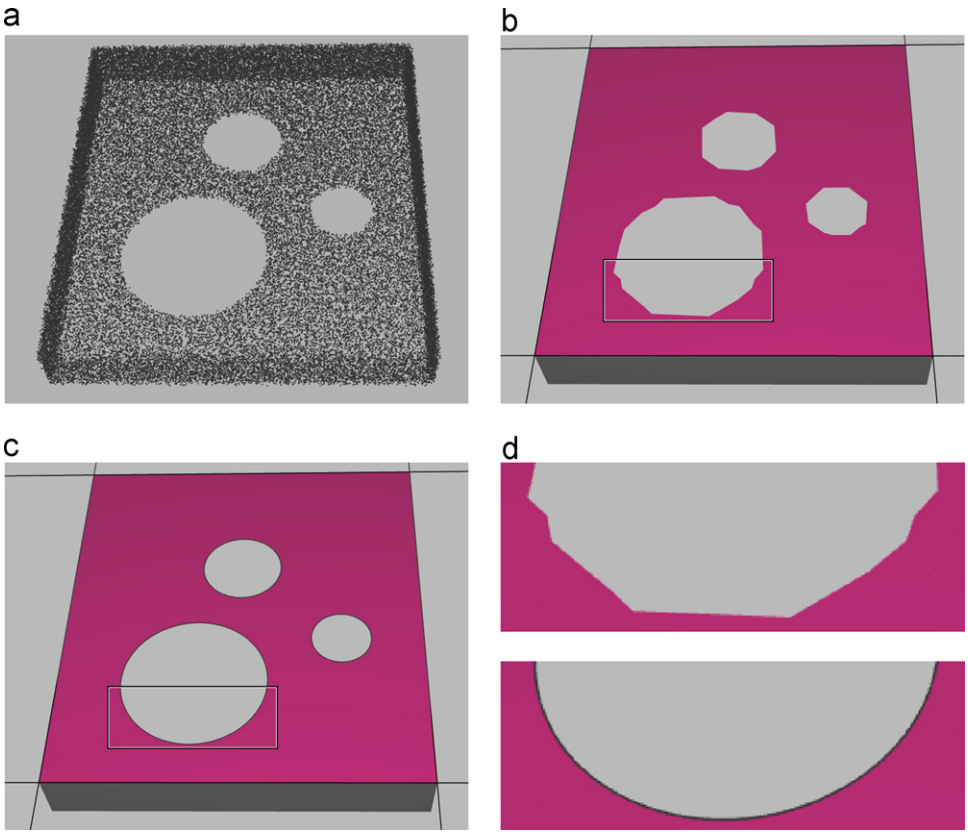

Fig. 14. A synthetic dataset of a table with three holes has been reconstructed: (a) sampled point cloud, (b) initial result without 2D shapes constraint, (c) final result based on detected 2D shapes, (d) detailed view of initial and final results.

circles or lines. Using two-dimensional shapes generally simplifies the existing polygon. Noisy curves are straightened and small features are removed, while at the same time sharp corners between different shapes are generated. Fig. 11 shows an example, where a circle and multiple lines are detected in the existing boundary and used as constraint for the next optimization step. As can be seen, the detection of 2D shapes usually needs some iterations of alternating constraint updates and shape optimizations.

The detection of 2D shapes is based on current boundaries, as we already have seen with the coplanarity constraints. Points and directions are sampled for a RANSAC-based detection of 2D shapes. Candidate shapes are generated and tested against all sample points. The candidate shape which is supported by the largest number of points is accepted and further shapes are detected in the remaining sample points.

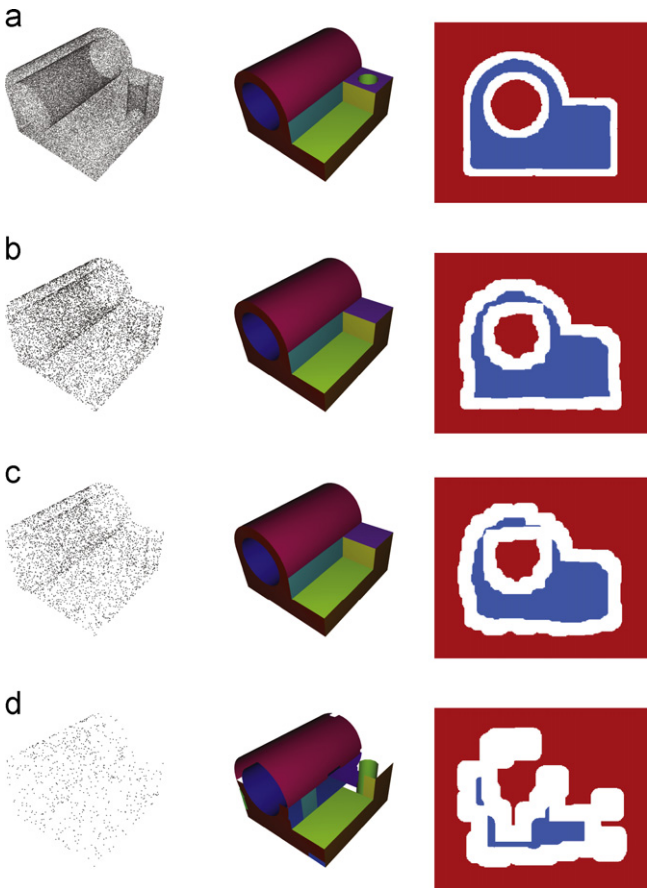

Fig. 15. Reconstruction of the joint model with decreasing point density (100%, 10%, 5% and 1%). The images show from left to right: input point set, reconstructed model, point cloud constraint. Note that the shape primitives were not automatically fit in this example, in order to compare only the boundary optimization. (a) 69 544 points, 12 correct shape boundaries. (b) 6954 points, 10 correct shape boundaries. (c) 3477 points, 8 correct shape boundaries. (d) 695 points, 2 correct shape boundaries.

3.7. User input

Fully automatic reconstruction systems not always deliver the result desired by a user. Especially, if large areas of the original scene are missing in the captured data, it is generally not possible to reconstruct these missing parts automatically. Therefore it is essential to provide a possibility for manually editing the reconstruction results and including additional information and creativity provided by a human user.

In our reconstruction system, the user can add optimization constraints by explicitly defining pixels as *inside* or *outside*. Similarly, it is possible to manually define good boundary positions. These user constraints are weighted with a very high impact factor so that they overrule other constraints.

We provide a user interface, where the user can directly draw constraints on shape surfaces in 3D (see Fig. 12). This is very easy to use because no knowledge about 3D modeling is needed. In the following optimization step, areas will be filled or deleted according to this new input and surface boundaries will snap to appropriate boundary constraints.

Some other constraints (coplanarity, 2D shapes) depend on the current shape boundary. A good strategy is therefore to push the shape boundary approximately to the desired location with user-defined constraints. Afterwards, the constraints are updated in order to get exact borders from other relations.

We have included the optimization system into a complete reconstruction application, which provides the whole pipeline for loading point clouds, detecting shape primitives, and reconstructing

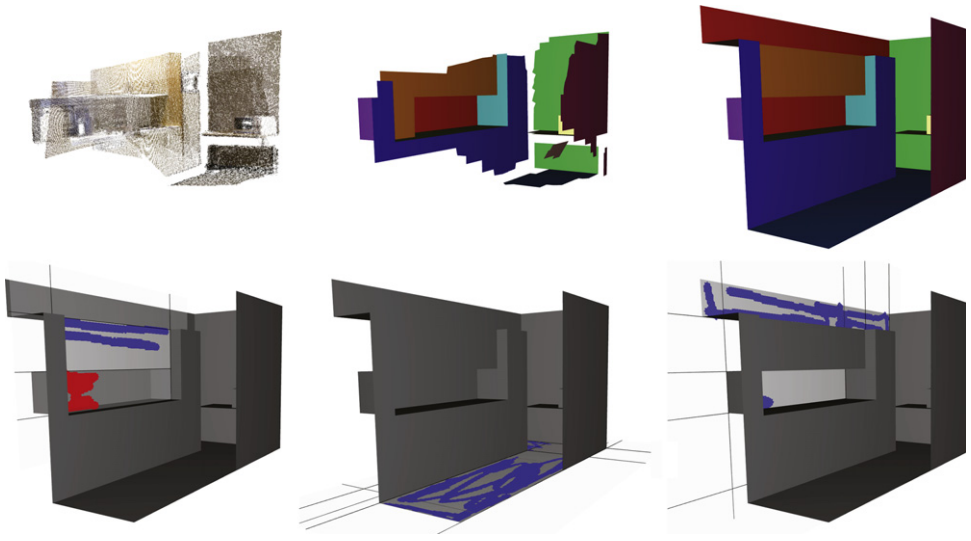

Fig. 16. This point cloud from eight Kinect range images has been reconstructed with 17 shapes. The top row shows the input point cloud, the initially reconstructed and the final model after some user inputs. The second rows show the user input for some shapes.

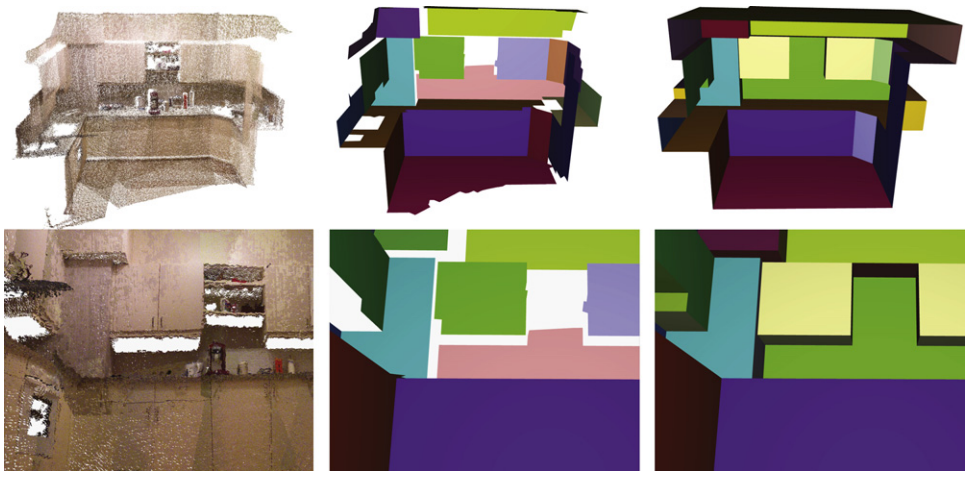

Fig. 17. A point cloud from 13 range images has been reconstructed with 26 shapes. Both rows show the input point cloud, the initially reconstructed and the final model after some user inputs from different viewpoints.

the shape boundaries. The application contains the following features for direct user interaction:

- Draw constraints on shape surfaces for defining areas as inside or outside, and for defining boundary edges.
- Select appropriate constraints and adjust impact factors as well as individual constraint parameters.
- Optimize shape boundaries for a single shape primitive or for the whole model.
- Create new shape primitives based on a set of selected points.
- Delete wrongly detected shape primitives.
- Create additional shapes based on coplanarity constraints.

4. Evaluation

We have tested our framework on a variety of synthetic and real-world datasets. The real-world datasets have either been captured as multiple depth maps with Microsoft Kinect (Figs. 16–18), created with photogrammetric reconstruction from multiple images (Fig. 19), or acquired with a laser scanner (Fig. 19). Table 1 summarizes the performance statistics and the necessary amount of user input for each dataset.

4.1. Synthetic datasets

Fig. 13 shows how our algorithm can deal with missing data. If the boundaries are optimized under equally weighted constraints for point clouds and intersections, the reconstructed model is approximately consistent with the input mesh. On the other hand, if the impact factor for intersections is largely increased by a factor of 20, the shapes extend to their nearest intersection lines and a closed model is generated. Note that this solution can only be applied when a closed model is desired. Another solution in this case is user input, where the missing areas can be filled just with a few simple brushing operations.

The application of 2D shape constraints has been evaluated as shown in Fig. 14. A table top with three circular holes has been uniformly sampled. Fig. 14(b) shows the result when the boundaries are optimized only according to the point cloud and intersections. The holes have been correctly preserved as the surface is densely sampled without large artifacts or missing data. But the surface boundary is not clean because it is just interpolated between the raster cells. After applying the 2D shape constraint, all three circles are correctly detected and the next optimization step produces a surface boundary with exact circles.

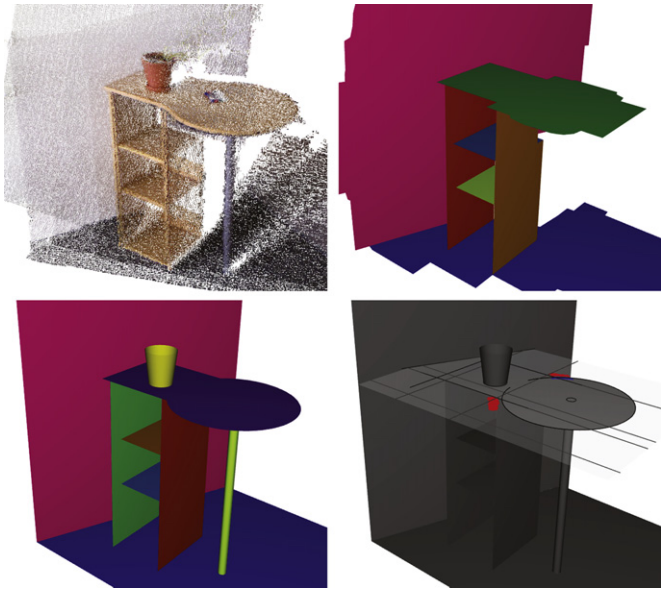

Fig. 18. Reconstruction of a bar table. The table leg and the flower pot have been inserted by fitting the shapes to the according points. The right picture shows how the boundary is restricted by the 2D shape constraint.

Fig. 15 shows how our algorithm performs with low point densities. The models are reconstructed solely with intersections and the point cloud constraint. The same parameters have been used for all input point sets. As can be seen, good results are obtained also in case of very few points. Only in the last case, where only 1% of the original point set is used, there is not enough information for extracting correct shape boundaries.

4.2. Real datasets

Figs. 16 and 17 show the results for two datasets containing a kitchen. The models have been optimized using constraints from the point cloud, intersections and coplanarity as well as user input. In both scenes, large parts are missing in the point cloud. While these parts are still missing in the initially reconstructed point cloud, they can be easily filled in by the user. In the model of Fig. 17 many additional planes have been inserted based on the

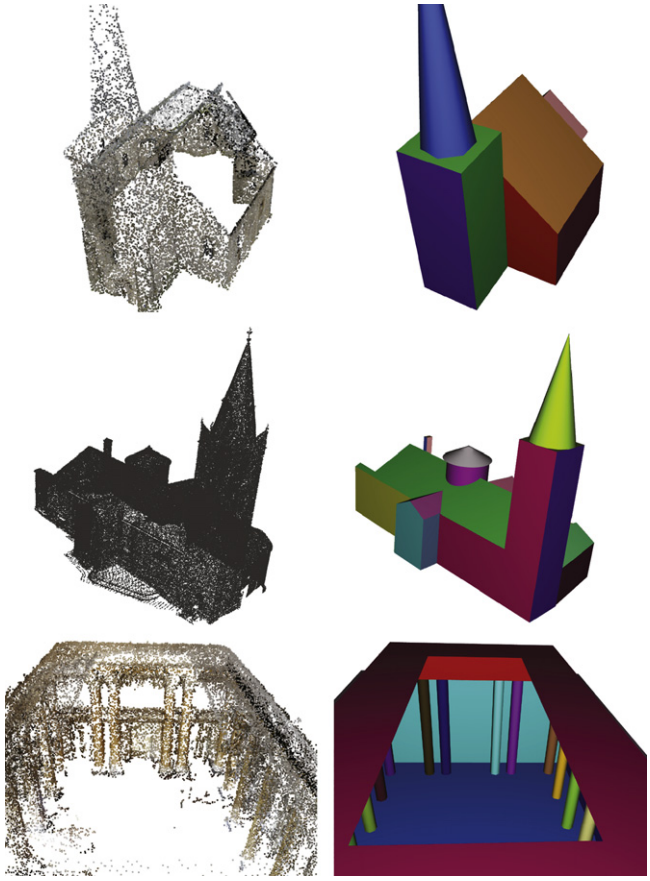

Fig. 19. Reconstruction of several datasets (from top to bottom): church (photogrammetry), church of Lans le Villard (laser scan), ballroom (photogrammetry).

coplanarity constraint, e.g. the top and bottom sides of cupboards. In this example it can also be seen, how the coplanarity constraints limits all shapes to the open front side.

Fig. 18 shows the reconstruction of a bar table. The result of the automatic pipeline, including point cloud, intersection and coplanarity constraints, produced good results for most of the shapes. The wall and the floor needed some manual input in order to generate nice outlines, because the point cloud was only partially visible and there were no limitations by other shapes. For the table top we applied the 2D shape constraint in order to retrieve an exact circular boundary. Details about the 2D shape fitting can be seen in Fig. 11. The shapes for the table leg and the flower pot were not discovered by the automatic shape detection algorithm, but our user interface provides the possibility to select points from the input point cloud and fit a shape primitive to it.

Additional examples that have been generated with our reconstruction system can be seen in Fig. 19. The input datasets are point clouds, either generated by photogrammetric reconstruction [1] or captured with a laser scanner.

As can be seen in Table 1, the Ballroom dataset contains a large number of shape primitives, which can still be handled by our system. Because each shape is optimized individually, the optimization step of our algorithm scales linearly with the number of shape primitives provided they have equal resolutions. Currently, our system is limited by the computations needed for constraint updates which are defined over all shape primitives of the scene (e.g. intersections). This clearly limits the number of shape primitives and therefore makes processing more complex scenes tedious. However, reconstructing more complex scenes would be possible if the computation of constraints is restricted to parts of the scene, e.g. by computing intersections only between nearby shape primitives.

5. Conclusions

We have presented a novel method for extracting and optimizing reasonable boundaries for shape primitives. The optimization framework supports many input sources and a wide range of high-level constraints. We have shown that the automatic algorithm provides a good initial reconstruction in many cases, despite noisy or incomplete point clouds. For difficult cases we provide a simple user interface which allows the user to push the shape boundaries towards preferred locations.

While we only employed three types of shape primitives, namely planes, cylinders and cones, the algorithm can easily be extended to other 2D manifolds. The only requirement is that the shape can be parameterized in a two-dimensional grid. Also the range of constraints can be further extended. For example, constraints could be inferred from approximate symmetries.

The main limitation of our system is that it depends on the quality of the detected shapes. In the future, we would like to use similar constraints also for generating new shapes or adapting existing ones.

Acknowledgments

This work has been partially funded by ZIT - Die Technologieagentur der Stadt Wien, Call IKT Wien 2010, Project Facsimile. Irene Reisner-Kollmann has been supported by the Doctoral College on Computational Perception at the Vienna University of Technology.

Appendix A. Supplementary material

Supplementary data associated with this article can be found in the online version of http://dx.doi.org/10.1016/j.cag.2013.01.001.

References

[1] Furukawa Y, Ponce J. Accurate, dense, and robust multiview stereopsis. IEEE Trans Pattern Anal Mach Intell 2010;32(8):1362–76.
[2] Henry P, Krainin M, Herbst E, Ren X, Fox D. Rgb-d mapping: using depth cameras for dense 3D modeling of indoor environments. In: International symposium on experimental robotic (ISER), 2010.
[3] Kazhdan M, Bolitho M, Hoppe H. Poisson surface reconstruction. In: Eurographics symposium on geometry processing, 2006.
[4] Alliez P, Cohen-Steiner D, Tong Y, Desbrun M. Voronoi-based variational reconstruction of unoriented point sets. In: Eurographics symposium on geometry processing, 2007. p. 39–48.
[5] Schnabel R, Wahl R, Klein R. Efficient RANSAC for point-cloud shape detection. Comput Graph Forum 2007;26(2):214–26.
[6] Gal R, Shamir A, Hassner T, Pauly M, Cohen-Or D. Surface reconstruction using local shape priors. In: Eurographics symposium on geometry processing, 2007. p. 253–62.
[7] Li Y, Wu X, Chrysathou Y, Sharf A, Cohen-Or D, Mitra NJ. Globfit: consistently fitting primitives by discovering global relations. ACM Trans Graph 2011;30.
[8] Schnabel R, Degener P, Klein R. Completion and reconstruction with primitive shapes. Comput Graph Forum 2009;28(2):503–12.
[9] Reisner I, Maierhofer S. Segmenting multiple range images with primitive shapes. In: International conference on systems, signals and image processing (IWSSIP), 2012. p. 320–3.
[10] Reisner-Kollmann I, Luksch C, Schwärzler M. Reconstructing buildings as textured low poly meshes from point clouds and images. In: Eurographics 2011—short papers, 2011. p. 17–20.
[11] Edelsbrunner H, Kirkpatrick D, Seidel R. On the shape of a set of points in the plane. IEEE Trans Inf Theory 1983;29:551–9.
[12] Avron H, Sharf A, Greif C, Cohen-Or D. l1-Sparse reconstruction of sharp point set surfaces. ACM Trans Graph 2010;29:135:1–12.

[13] Chen J, Chen B. Architectural modeling from sparsely scanned range data. Int J Comput Vision 2008;78:223–36.
[14] Jenke P, Krückeberg B, Straßer W. Surface reconstruction from fitted shape primitives. In: Vision, modeling and visualization, 2008. p. 31–40.
[15] Vanegas C, Aliaga D, Benes B. Building reconstruction using manhattan-world grammars. In: IEEE conference on computer vision and pattern recognition (CVPR), 2010. p. 358–65.
[16] Nan L, Sharf A, Zhang H, Cohen-Or D, Chen B. Smartboxes for interactive urban reconstruction. ACM Trans Graph 2010;29(4):93:1–10.
[17] Furukawa Y, Curless B, Seitz SM, Szeliski R. Manhattan-world stereo. In: IEEE conference on computer vision and pattern recognition (CVPR), 2009. p. 1422–9.
[18] Sinha SN, Steedly D, Szeliski R, Agrawala M, Pollefeys M. Interactive 3D architectural modeling from unordered photo collections. ACM Trans Graph 2008;27:159:1–10.
[19] Schindler K, Bauer J. A model-based method for building reconstruction. In: IEEE workshop on higher-level knowledge in 3D modeling and motion analysis, 2003. p. 74–82.
[20] Arikan M, Schwärzler M, Flöry S, Wimmer M, Maierhofer S. O-Snap: optimization-based snapping for modeling Architecture. ACM Trans Graph 2013;32(6):1–15.
[21] Boykov YY, Jolly MP. Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. In: IEEE international conference on computer vision (ICCV), 2001. p. 105–12.
[22] Boykov YY, Veksler O, Zabih R. Fast approximate energy minimization via graph cuts. IEEE Trans Pattern Anal Mach Intell 2001;32(11):1222–39.
[23] von Gioi RG, Jakubowicz J, Morel JM, Randall G. LSD: a fast line segment detector with a false detection control. IEEE Trans Pattern Anal Mach Intell 2010;32:722–32.
[24] Nacken P. A metric for line segments. IEEE Trans Pattern Anal Mach Intell 1993;15(12):1312–8.