User Driven 3D Reconstruction Environment

David Sedlacek and Jiri Zara

Czech Technical University in Prague, Faculty of Electrical Engineering
david.sedlacek@fel.cvut.cz

Abstract. An intuitive image-based 3D reconstruction tool based on inaccurate user strokes is presented in this paper. The combination of fast image segmentation method together with user knowledge about reconstructed scene forms a novel low-polygonal editor suitable for architecture reconstruction. The user interaction is minimized thanks to propagation of strokes among input photographs. The final model geometry is created by innovative algorithm. The input to the tool is a set of calibrated photographs together with a sparse pointcloud. The output is a structured low-poly 3D model.

1 Introduction

Inspired by well known 3D editors for low-polygonal image-based modelling (like ImageModeler, PhotoModeler) we propose a novel editor reducing amount of user interaction. We present a combination of state-of-the-art techniques and intuitive interaction methods for geometry construction. Our approach opens new ways for 3D reconstruction focused on low-poly output which is well suited for internet presentation. The user interaction brings also advantages for 3D reconstruction in low-textured or occluded areas where automatic methods often fail. This reconstruction tool is suitable for architecture reconstruction or other

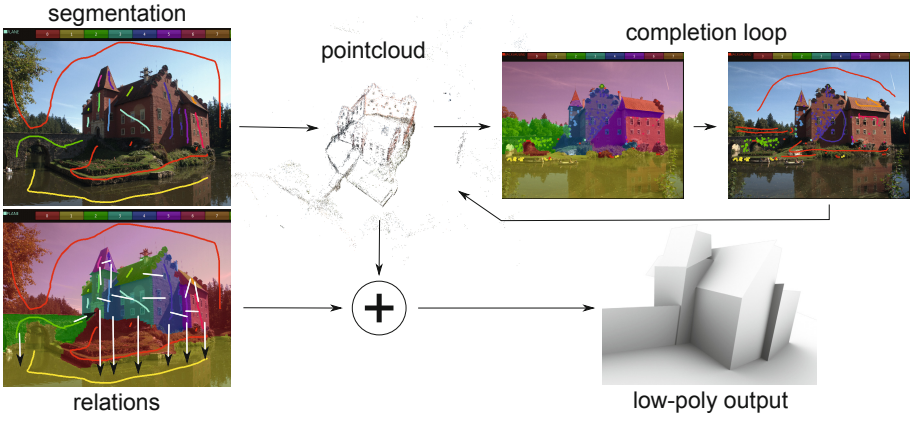


Fig. 1. Program workflow overview

G. Bebis et al. (Eds.): ISVC 2012, Part I, LNCS 7431, pp. 104–114, 2012.

areas where planar structures dominate. Calibrated photos (known camera positions) and sparse pointcloud (reconstructed during camera calibration phase) constitute the input of the method. We have successfully used both Bundler [1] and APERO [2] project as suitable calibration engines.

The core of our approach is geometry primitives fitting based on source photo segmentation, see Fig. 1. The structure of the paper corresponds to individual algorithm steps. The graph-cut image segmentation is driven by inaccurate user strokes (Section 3). The image segmentation is then used for labelling of sparse pointcloud points and is also propagated to close photographs (Section 4). Various geometry primitives are then fitted on labelled points (Section 5). Thanks to user-defined relations between adjacent geometrical structures (Section 6), final polygonal geometry is computed (Section 7).

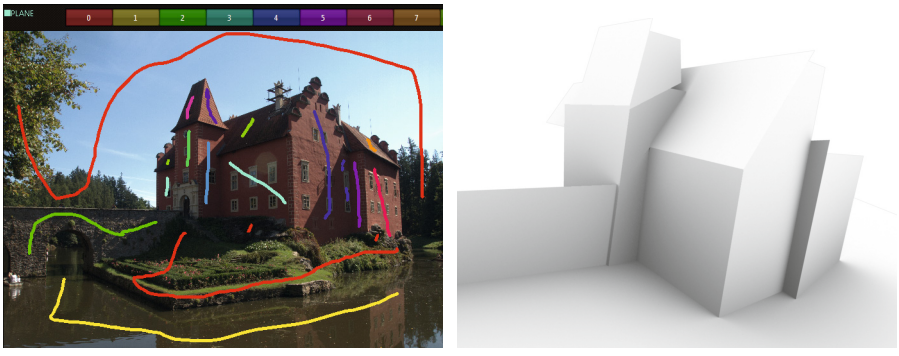
2 Related Work

The image-based 3D modeling is studied over last 15 years. Several systems were developed over this time, and all of them have common properties. The user works with a set of photographs and selects interesting parts of objects for the reconstruction.

One of the first work was Facade by Debevec at al. [3] which later gave rise to a commercial product called Canoma. The system provided several 3D primitives like prisms, cuboids and pyramids. The user placed the primitives into photographs and improved positions of the primitives in other views. Since the scene was continuously calibrated by user added objects, the system tended to be unstable. Later, other commercial products were inspired by Canoma software, like PhotoModeler by Eos Systems, ImageModeler by RealViz (now Autodesk) and the PhotoMatch component of Google SketchUP. The calibration of input photographs was shifted to user in those systems.

Other systems estimate the camera calibration using automatic methods based on SFM. Some programs have developed their own calibration core (VideoTrace by van den Hengel et al. [4], system for architectural modelling by Sinha et al. [5]). Other works benefit from external SFM calibration tools (like MatchMover by RealViz, Boujou by 2d3 or Bundler) and suppose calibrated scene as input (Habbecke et al. [6], Paczkowski at al. [7]).
Other works try to get as much information from one photo as possible [8–10]. The one photo calibration is based on finding vanishing points constraints and on the fact that all architectural buildings have parallel lines and the building blocks are build in Manhattan layout [11]. Vanishing points are used even in modelling from more photographs like Sinha et al. [5], Cipolla et al. [12] and Wilczkowiak et al. [13].

Sinha et al. [5] shows that the combination of unordered photo set, detected vanishing points and vanish lines and auto-calibration using SFM leads to user friendly free-polygonal modelling. The user draws lines to the image. The lines are snapped to the direction of vanishing lines and then the lines are extruded to the face in the direction of plane perpendicular to the line in user selected



(a) (b)

Fig. 2. User inputs and geometry output: (a) Sketch based segmentation interface; (b) Geometry reconstructed from pointcloud and informations defined by a user in the left photograph

direction. They use RANSAC algorithm for the best plane fitting constrained by vanishing points. The plane fitting algorithm found the best fitting plane in sparse pointcloud restricted by points lying within a face selected by user. They call their user modelling interface sketch-based but it is restricted to placing lines near the real line, in other cases the line is snapped to the wrong vanishing line. Sketch-based interfaces are also applied in works of El-hakim at al. [14] and Xiao at al. [15] where they provide tools for selection, marking, and other specific operations.

The sketch-based interface is mostly related to creation of new 3D models or annotations, see [16] for wide overview. Interesting work is combination of calibrated scene with creation of new architectural design in site, see Paczkowski at al. [7].

Our approach is similar to Sinha et al. [5] however we intensively utilize a sketch-based interface in order to define a higher logic and geometrical structure within 3D reconstructed scene.

3 Image Labelling

In our approach we benefit from user knowledge about the scene to be reconstructed. We suppose that the user recognizes basic scene primitives (a wall, a pillar, etc.) and the connections between them (adjacent or not). For this we have designed an intuitive user interface for photo labelling based on graph-cut algorithm, see Fig. 2a.

Similarly as Boykov at al. [17], we formulate labelling problem as an energy function minimization. As an input we use a gray-scale image P where each pixel $p \in P$ is connected to 4 neighbours (N). The goal is to find labelling l, i.e. to assign each pixel p a label in finite label set $L_p \in L$ while the energy function is minimized. We express the energy function as:

$$E(l) = \sum_{\{p,q\} \in N} V_{p,q}(L_p, L_q) + \sum_{p \in P} D_p(L_p) \tag{1}$$

Where $V_{p,q}$ is smoothness term representing the energy of intensity discontinuity between two neighbouring pixels and data term D_p is the energy of assigning the pixel to label. The $L_{p,q}$ stands for pixel labelling. We define the smoothness term as:

$$V_{p,q}(L_p, L_q) = 1 + (K * e^{\frac{-(I_q - I_p)^2}{\sigma^2}}) \tag{2}$$

Where $I_{p,q}$ are pixels intensities, constant K is scaling factor from float to integer values $\langle 0; K \rangle$ and constant σ controls exponent function behaviour. These constants are estimated based on image size, which is mentioned in following paragraphs. Similarly as Sykora at al. [18] avoid segmentation discontinuities using non-zero smoothness term we map the final values of smoothnes term to the $\langle 1; K \rangle$ interval (additive constant in eq. 2). The K and σ constants are set with respect to image size at the end of this section.

The data term in many segmentation methods is usually set to some image-based likelihood such as pattern or color similarity. Due to the fact that our segmentation completely relies on user inputs, the data term is set only to pixels selected by the user and does not take into account any specific image properties.

$$D_p(L_p) = \begin{cases} 0, & p \in L \\ \infty, & p \notin L \end{cases} \tag{3}$$

Since the smoothness term relies only on pixel intensity and not on the labelling, our energy function can be minimized by solving multiway cut problem on undirect graph [19]. Our graph $G = \{V, E\}$ has the same topology as described in [18], i.e. each image pixel correspond to one vertex P and is connected to 4 neighbours by edges E_p with capacity equal to smoothness term $w_{p,q} = V_{p,q}$ from eq. 2. The edges E_c between pixels P and terminals C has capacity $w_{p,c} = K - D_p$. Considering eq. 3 we can simplify terminal capacity to $w_{p,c} = K$. Recall that only pixels where user made hard strokes are connected to terminals.

Inspired by LazyBrush [18] we solve the multiway cut algorithm by greedy algorithm where one label is randomly chosen as S terminal and all others are connected to the T terminal. Thereafter max-flow/min-cut problem is solved using [17]. All pixels labelled as S are disconnected from graph and new label is selected as terminal S from the set of all remaining labels previously connected to T. This procedure is repeated until T is empty.

The image segmentation is used for labelling sparse pointcloud. Each 3D point of sparse pointcloud contains the list of cameras where it was visible and 2D point representing the image of 3D point in the corresponding camera. Thanks to this information, we are able to propagate the image segmentation to the 3D pointcloud which is then used for 3D reconstruction in section 5.

To achieve best results from calibration phase, we prefer to use photos in original size. Unfortunately the image size of several mega-pixels causes very long segmentation times (Table 1). That is why we internally down-scale images to the size

of 640px for longer edge. The computed segmentation is then up-scaled to original images. Thanks to low distribution of underlying pointcloud we do not need to use any of segmentation refinements in higher resolution, for example like [20]. Currently a faster single-core implementation of Boykov algorithm has been introduced by Jamriska et al. [21] but we have not integrated it yet.

Table 1. Processing times of graph-cut algorithm depending on image maximal edge size and labels count. Intel Core2 Quad, 2.5GHz, single core.

size	2 labels	5 labels	10 labels	15 labels
640px	402ms	723ms	921ms	1110ms
800px	800ms	1252ms	1650ms	2043ms
1024px	1342ms	2328ms	2970ms	3882ms

With respect to chosen image size, the edge capacity constants are set as: $K = 2 * (h + w)$ and $\sigma^2 = 32$. To get maximal interactivity during the labelling and reconstruction process, the image segmentation is performed immediately after each user stroke. If the user considers the delay after each stroke too long she can switch to on demand segmentation process where all strokes are given in advance followed by final segmentation evaluation.

4 Propagation of Segmentation

As stated in previous section, the image segmentation is used only for finding labelling of sparse pointcloud. After processing the first photograph, the labelling of the whole model is not complete. Thus other photos are to be used to finish the labelling process. The automatic propagation of labelling is implemented to achieve faster reconstruction and also for the reason of user load reduction. The labelling suggestion is based on the following procedure.

For each feature point in image we get corresponding 3D point of pointcloud. If this 3D point is already labelled we transfer this labelling into next image. In related works the suggested labelling is transferred into segmentation graph as "Soft scribble". It can be interpreted as a connection of node to the corresponding terminal with much less capacity than used for the user input (hard scribble). The tests with real data shown successful propagation of labelling to new photo but brought problems in case of subsequent user corrections. When new user strokes were added the segmentation became less stable and disintegrated into small unconnected regions, requiring additional corrections, see Figure 3b. For this reason we decided to transfer labelling as new user strokes (hard scribbles) which can be easily deleted or over-painted by the user. New hard scribbles are generated as small circles with center at the position of image feature point, see Figure 3c.

5 Fitting of Geometry Primitives

The geometry primitives are fitted to the segmented pointcloud. Each label corresponds to one geometry primitive type (e.g. plane, sphere, box) and primitive

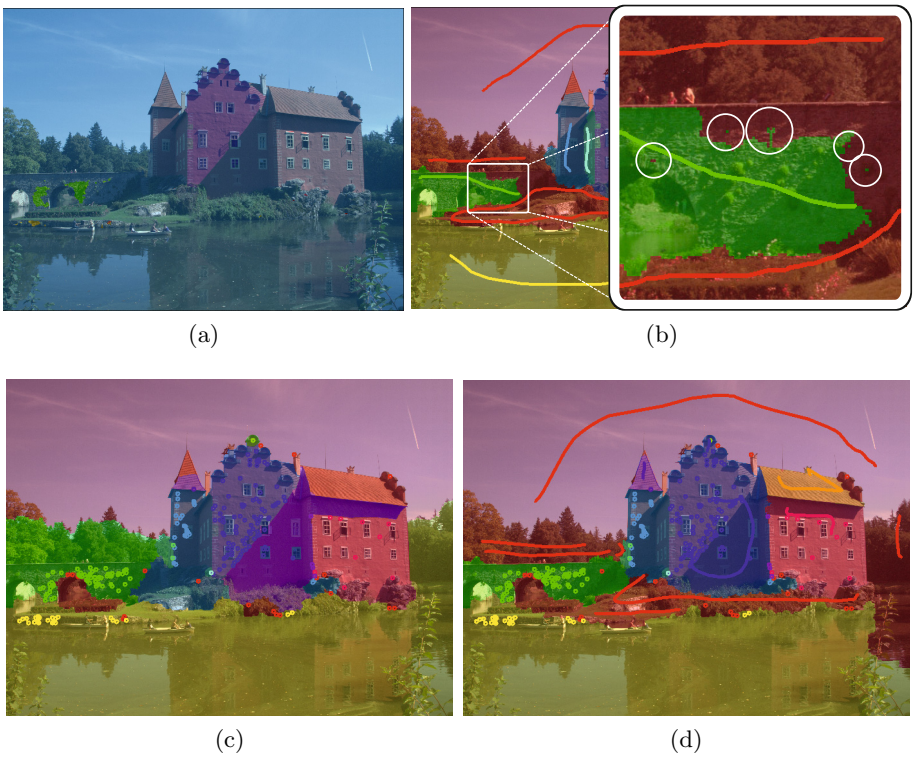


Fig. 3. The segmentation propagated from photo in Fig. 2a using different scribble types. (a) propagated segmentation using soft scribbles; (b) correction of soft scribbles segmentation, notice isolated parts in the closer view; (c) propagated segmentation using hard scribbles; (d) correction of hard scribbles segmentation.

occurrence. It means, that each primitive should be segmented with new label stroke of corresponding type. The RANSAC [22] algorithm is used for geometry model parameter estimation. Thanks to user points pre-selection there is a high probability that all points belongs to the given primitive. For this reason, we do not use any special candidates sampling strategy, for example as in [23], but all label points are sampled randomly.

The output of this algorithm step is not the final geometric representation of found primitives but only the algebraic parameters of geometry model (e.g. *a,b,c,d* parameters for plane or *center* and *radius* for sphere). The description of final geometry creation is described in section 7. Currently we support fitting of planes and spheres only, however our labelling framework is prepared for other geometry types as well.

6 Relations

In standard image-based modelling software (like ImageModeler, PhotoModeler, Sketchup), the reconstructed model shape is built bottom-up starting from the

smallest primitives (points) through edges up to faces and meshes. We introduce another approach (top-down), where the mesh geometry is defined by geometrical primitives and relations among them. In case of planes, a relation of two non-parallel planes defines an edge, similarly three planes form a point.

We recognize two relation types between planes, see Fig. 4. The first one is **cross** relation where two planes form an edge and both planes are split on two half-planes while only two of half-planes forms a mesh. This relation represents a mesh edge. We assume that this relation is enough if and only if all mesh planes are visible and they can be reconstructed. Unfortunately in real scenes a lot of mesh sides are hidden behind some bigger objects, for example a book lying on a table. The bottom side of the book is not visible but we need some reference for restriction of other four book sides. For this reason we introduce second relation - the **restriction**. The restriction relation naturally corresponds to the definition of two different objects and is interpreted like the first plane (book side) is split by the second one (table plane) while the second plane remains unchanged.

The relations are defined in the same interface as labelling by a stroke connecting areas with two different labels, see Fig. 4d.

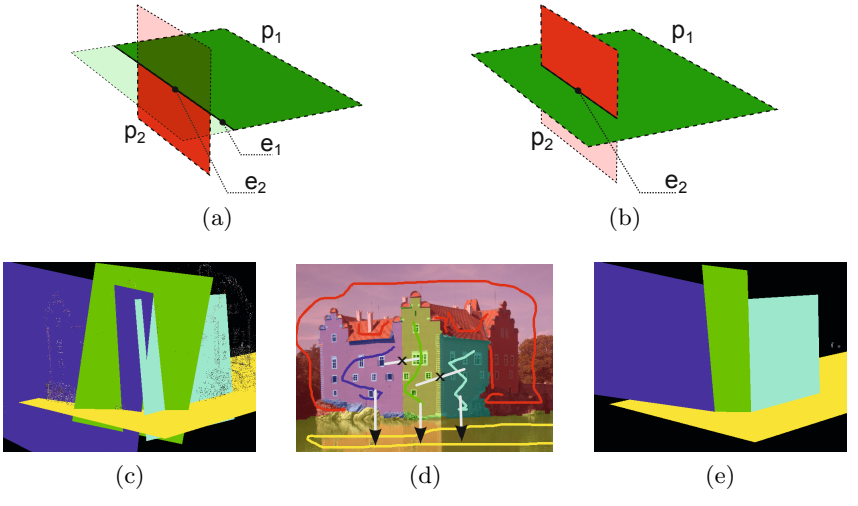


Fig. 4. Two types of relations shown on a schematic view and a real example. In schematic view two planes $p_{1,2}$ are split with respect to the relation type: the cross relation (a) and the restriction relation (b) (T-junction). The plane parts filled by solid color remain. In the real example (d), the cross relation is depicted by line with $\times$ letter in the middle, the restriction relation is depicted by an arrow. Stroke colors correspond to plane colors. Original planes (c) are unrestricted and they intersect mutually. In (e), three upper planes create edges defined by the cross relation. They are all restricted by the bottom yellow plane which is not cropped by other planes.

7 Finding Output Geometry

The final geometry is constructed from outputs of fitting phase in combination with relations. The overall process is similar to modelling boundary representations of solids with boolean operations. The difference is that our primitives are not closed solids and that crossing planes do not determine inner or outer subspace unambiguously. To properly solve the problem we incorporate a knowledge of feature points belonging to fitted geometry. For the sake of simplicity we describe a situation in which planes are considered only.

Each plane defined by a user is initially treated as unbound and the mesh geometry is found as an intersection of related planes. The task is to find bounds for each plane what forms a polygon. Inspired by Bernstein at al. [24] we define convex polygon p as a couple of supporting plane s and counter clockwise ordered list of bounding planes $\{b_i\}_{i \in Z_n}$, see Fig. 5a. The polygon vertices are then given by intersection of three planes: $v_i = (s, b_{i-1}, b_i)$. We construct the final, possibly non-convex polygon from a set of small convex components, see Fig. 5b. All the details connected to polygons like numeric and geometric basis, and BSP operations are described in [24].

To make the process more robust, we artificially add several bounding planes in a far distance. This solves a problem of incompletely specified set of crossing planes.

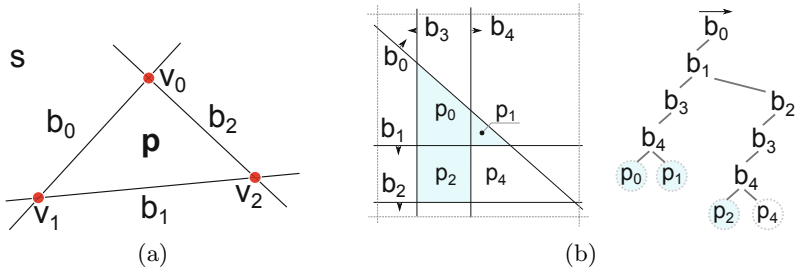


Fig. 5. Polygons: (a) convex polygon representation. Supporting plane s is bounded by three boundary planes b_i. Corresponding vertices are shown as dots; (b) Supporting plane is split by related planes into convex polygons. Artificial planes in far distance are illustrated as dashed lines and border polygons (i.e. adjacent to dashed lines) are not labelled for simplicity. The corresponding BSP (sub)tree on the right shows only the leaves containing inner convex polygons. The final polygon consists of three elementary polygons p_0, p_1, and p_2.

We find a final polygon defined by supporting plane s in two steps. Firstly we get a set of all convex polygons defined by plane s and all planes related to it as a result of BSP tree creation. Note that the elementary convex polygons are not treated as geometrical objects only but we keep a list of corresponding pointcloud points attached to them. This allows us to collect all contributing

elementary polygons by solving min-cut problem in the second step. The following algorithm describes the first step, its output (BSP tree and a set of convex polygons) is depicted in the Figure 5b.

Routine for finding all possible convex polygons

```
getPolygons (Plane s) {
   R = getRelatedPlanes(s); //get all s-related planes
   p = new Polygon(s); //create polygon bounded by distant planes
   BSP = new BSP(p); //BSP structure for polygon intersections
   for(Plane r : R) {
     BSP.addSplitPlane(r);
   }
   return BSP.getLeaves();
}
```

The second step is a selection of the subset of convex polygon participating in the final polygon. We are looking for binary labelling (IN, OUT) of elementary polygons based on information previously defined by a user. We state out the differences in relation to already described graph-cut algorithm in Section 3. The nodes of graph $G = (V, E)$ are now the leaves of the BSP tree P_i connected with neighbour leaves by edges E_p with capacity equal to $w_{p,q}$, see eq. 4. Some nodes are connected by edges with capacity $w_{p,s}$ and $w_{p,t}$ to S-source and T-sink terminals respectively, see Fig. 6b.

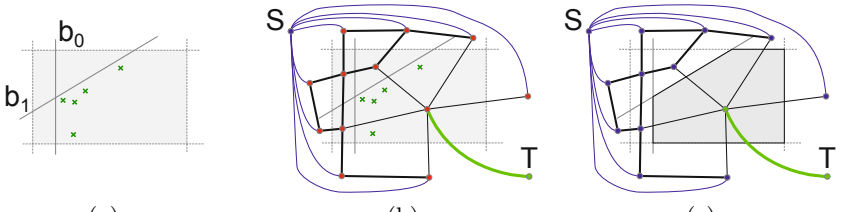


Fig. 6. Graph-cut: (a) An input plane bounded by distant planes (gray rectangle with dashed boundary), two related planes and few plane points. (b) Input plane extended by graph for solving min-cut problem. (c) Labelled graph with depicted final polygon. Point out that minimal-cut correspond to polygon boundary edges. The gray filled polygon is labelled IN, the other three white filled polygons are labelled OUT.

$$\begin{aligned} w_{p,q} &= 1 + (K * e^{\frac{-(Q_{points} - P_{points})^2}{\sigma^2}}) \\ w_{p,s} &= K, \qquad\qquad if(P_{polygons} = 0) \\ w_{p,t} &= K * P_{points}, \quad if(P_{points}/points > 0.6) \end{aligned} \tag{4}$$

The terminals are connected to nodes only if the given condition is satisfied. The symbols $P_{points}, P_{polygons}$ stand for a number of pointcloud points or polygons

in current node respectively. The symbol *points* is an overall number of points contained in the supporting plane. Notice that only the tree leaves representing the space out of artificial bounding planes have zero polygons. The constants K and σ^2 were experimentally set to $K = 100$ and $\sigma^2 = 1000$. The output of this algorithm is a set of elementary polygons contained in nodes labelled as T (i.e. IN) and forming the final mesh geometry, see Fig. 6c.

8 Conclusion

We have presented a novel interactive 3D reconstruction tool for calibrated photographs. It is composed of several well established techniques extended by new ideas and algorithms. Especially the algorithm presented in Section 7 that is able to construct non-convex polygons from imperfectly specified input planes using graph-cut algorithm represents a promising way.

The reconstruction is driven by inaccurate strokes and is well prepared for use on touch screens. The output is a structured low-poly model suited for online presentations.

Acknowledgment. This work has been partially supported by the Grant Agency of the Czech Republic under research program P202/11/1883, and the Grant agency of the CTU Prague, grant No. SGS10/291/OHK3/3T/13.

References

1. Snavely, N., Seitz, S.M., Szeliski, R.: Photo tourism: exploring photo collections in 3d. In: SIGGRAPH 2006: ACM SIGGRAPH 2006 Papers, pp. 835–846. ACM, New York (2006)
2. Deseilligny, M.P., Clery, I.: Apero, an open source bundle adjusment software for automatic calibration and orientation of set of images. In: Proceedings of the ISPRS Symposium, 3DARCH11 2011 (2011)
3. Debevec, P.E., Taylor, C.J., Malik, J.: Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In: SIGGRAPH 1996: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, pp. 11–20. ACM, New York (1996)
4. van den Hengel, A., Dick, A., Thormählen, T., Ward, B., Torr, P.H.S.: Videotrace: rapid interactive scene modelling from video. In: SIGGRAPH 2007: ACM SIGGRAPH 2007 Papers, p. 86. ACM, New York (2007)
5. Sinha, S.N., Steedly, D., Szeliski, R., Agrawala, M., Pollefeys, M.: Interactive 3d architectural modeling from unordered photo collections. In: SIGGRAPH Asia 2008: ACM SIGGRAPH Asia 2008 Papers, pp. 1–10. ACM, New York (2008)
6. Habbecke, M., Kobbelt, L.: An intuitive interface for interactive high quality image-based modeling. Comput. Graph. Forum 28, 1765–1772 (2009)
7. Paczkowski, P., Kim, M.H., Morvan, Y., Dorsey, J., Rushmeier, H., O'Sullivan, C.: Insitu: sketching architectural designs in context. ACM Trans. Graph. 30, 182:1–182:10 (2011)

8. Jiang, N., Tan, P., Cheong, L.F.: Symmetric architecture modeling with a single image. In: SIGGRAPH Asia 2009: ACM SIGGRAPH Asia 2009 Papers, pp. 1–8. ACM, New York (2009)
9. Guillaume, L.Z., Zhang, L., Dugas-phocion, G.: Single view modeling of free-form scenes. In: Proc. of CVPR, pp. 990–997 (2002)
10. Zheng, Y., Chen, X., Cheng, M.M., Zhou, K., Hu, S.M., Mitra, N.J.: Interactive images: Cuboid proxies for smart image manipulation. ACM Transactions on Graphics 31 (2012)
11. Duan, W., Allinson, N.M.: Vanishing points detection and line grouping for complex building facade. In: Proc. of WSCG 2010 (2010)
12. Cipolla, R., Robertson, D.: 3d models of architectural scenes from uncalibrated images and vanishing points. In: Proceedings. International Conference on Image Analysis and Processing, pp. 824–829 (1999)
13. Wilczkowiak, M., Sturm, P., Boyer, E.: Using geometric constraints through parallelepipeds for calibration and 3d modeling. IEEE Transactions on Pattern Analysis and Machine Intelligence 27, 194–207 (2005)
14. El-hakim, S., Whiting, E., Gonzo, L.: 3d modelling with reusable and integrated building blocks. In: 7th Conference on Optical 3-D Measurement Techniques, pp. 3–5 (2005)
15. Xiao, J., Fang, T., Tan, P., Zhao, P., Ofek, E., Quan, L.: Image-based façade modeling. In: SIGGRAPH Asia 2008: ACM SIGGRAPH Asia 2008 Papers, pp. 1–10. ACM, New York (2008)
16. Jorge, J., Samavati, F.F. (eds.): Sketch-based Interfaces and Modeling, 1st edn. Springer (2011)
17. Boykov, Y., Veksler, O., Zabih, R.: Fast approximate energy minimization via graph cuts. IEEE Trans. Pattern Anal. Mach. Intell. 23, 1222–1239 (2001)
18. Sýkora, D., Dingliana, J., Collins, S.: Lazybrush: Flexible painting tool for hand-drawn cartoons. In: Computer Graphics Forum (Proceedings of Eurographics 2009), vol. 28, pp. 599–608 (2009)
19. Boykov, Y., Veksler, O., Zabih, R.: Markov random fields with efficient approximations. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 1998, p. 648. IEEE Computer Society Press, Washington, DC (1998)
20. Lombaert, H., Sun, Y., Grady, L., Xu, C.: A multilevel banded graph cuts method for fast image segmentation. In: Tenth IEEE International Conference on Computer Vision, ICCV 2005, vol. 1, pp. 259–265 (2005)
21. Jamriška, O., Sýkora, D., Hornung, A.: Cache-efficient graph cuts on structured grids. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (2012)
22. Fischler, M.A., Bolles, R.C.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM 24, 381–395 (1981)
23. Schnabel, R., Wahl, R., Klein, R.: Efficient ransac for point-cloud shape detection. Computer Graphics Forum 26, 214–226 (2007)
24. Bernstein, G., Fussell, D.: Fast, exact, linear booleans. In: Proceedings of the Symposium on Geometry Processing, SGP 2009, Aire-la-Ville, Switzerland, Eurographics Association, pp. 1269–1278 (2009)