Single View Reconstruction of Piecewise Swept Surfaces

Avanish Kushal Steven M. Seitz
University of Washington, Seattle
{kushalav,seitz}@cs.washington.edu

Abstract

We present a novel approach for the single view reconstruction (SVR) of piecewise swept scenes, exploiting the regular structure present in these man-made scenes. The parallelism of lines and its extension to curves are used as cues within a novel sequential algorithm propagating information across faces and their boundaries. Using this approach we are able to model a wide variety of architectural scenes and man-made objects. We show results generated both automatically as well as using user interaction, within a unified framework.

1. Introduction

The reconstruction of 3D objects from 2D images has a long history in computer vision. Our objective in this paper is to reconstruct models of man made objects, particularly architectural scenes from a single image. These scenes can be characterised as being easily "outlined", i.e., their boundaries capture the information required to reconstruct them as shown in Figure 1. We build upon two directions of research — first, interactive modelling of planar surfaces [1, 2] and extend it to scenes with piecewise swept surfaces. Second, we extend recent techniques in automatic reconstruction of manhattan scenes [3] to this broader class.

We present a novel approach to the single view reconstruction (SVR) problem by solving for a dense orientation map, i.e., we compute the normal direction for every pixel in the image. This is done by a sequential algorithm that propagates normal information from faces to the lines or curves that outline them. In this way we are able to compute 3D line directions for image lines (or tangents directions for curves). Further, these computed directions can be propagated to other faces that are bounded by these lines or curves. This allows us to model a larger class of scenes, with lines and curves in arbitrary directions. This orientation map is integrated to get a per pixel depth map, in a manner similar to photometric stereo and shape from shading techniques [4, 5], and is then converted into a 3D texture mapped model for the object.

Many researchers have investigated 3D reconstruction from line drawings. Huffman [6] detected impossible objects by studying line intersections distinguishing between concave, convex, and occluding intersections. Kanade [7] reconstructed composites of shells and sheets as belonging to the special class of "origami world". Horry et.al [8] allowed piecewise planar reconstructions of paintings and photographs whereas subsequent systems [1, 2] generalized to scenes with multiple vanishing points or planes. However, the above systems are restricted to planar scenes and require users to specify parallel and orthogonal lines. Further they rely on computing 3D line directions from their corresponding vanishing points, information which might not be available. Using our approach we are able to extend these methods to scenes with lines and planar curves in arbitrary directions (even without their vanishing points), with minimal user interaction. These planar curves are present in abundance in urban scenes, e.g., arches and doorways. Often these curves are repeated and related by a translation in 3D. We show how such translated curves can be identified and subsequently used in computing the 3D tangent direction of each point on the curve.

There has been research in the single-view reconstruction of objects with free-form surfaces from their contours [9], as well as by specifying the normal directions at certain pixels [10]. Further, reconstruction of special classes of curved surfaces, such as surfaces of revolution and straight homogeneous generalized cylinders has been studied in [11]. Another recent approach looks at modelling reconstructed point clouds as generalized cylinders [12]. Our approach addresses a different class of curved objects — highly structured objects with faces whose normals can be determined by sweeping its contours, a generalization of piecewise planar scenes.

On the automatic side, [13, 14] used machine learning and databases of RGBD images to compute depth maps from a single view. Our work takes inspiration from the line sweeping approach used by Lee et al. [3] which showed that automatically detected line segments combined with prior knowledge about how interiors of buildings are structured can be successfully used to reconstruct the models in

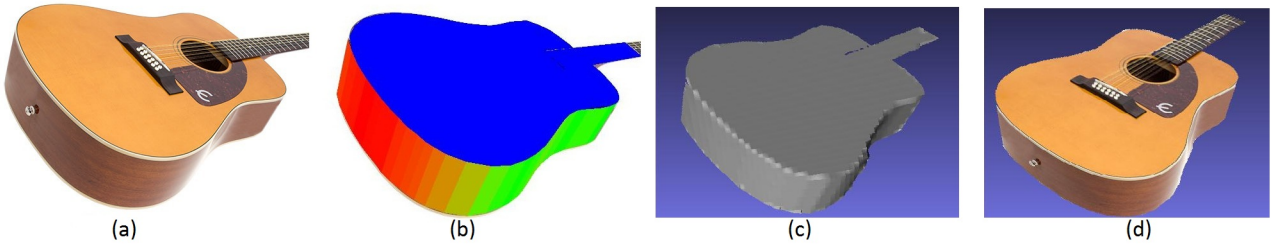

Figure 1. A reconstruction of a guitar (a) shows the input image, (b) the reconstructed orientation map, (c) the shaded depth map, and (d) the texture mapped model.

3D. Flint et al. [15] instead computed the 3D model using dynamic programming to efficiently search from the possible Manhattan models. However these approaches are restricted to indoor Manhattan world scenes, where all the planes are oriented along one of the 3 mutually orthogonal directions, and additionally the scene has a ceiling, a roof and vertical walls joining them. We extend this sweeping approach of Lee et al. [3] to arbitrarily oriented lines, and compute a partial orientation map. This map is used within the central theme of identifying faces and their contours, following which 3D models can be computed automatically using the same framework as described for interactive modelling.

While our approach works for most piecewise swept scenes with minimal user interaction, automatic reconstruction is much more challenging as not all lines or curves defining the scene (orientation or depth discontinuities) are robustly detectable. Further, occlusion and surface edges complicate the problem. We show automatic results for this larger class of scenes, albeit for simpler scenes where most of the defining boundaries (orientation discontinuities) are clearly discernible.

2. Problem Definition

Consider the line drawing in Figure 2(a). Given this drawing, we can identify all the faces F, and the lines L in the scene. A face here refers to a closed region of the image bounded by lines (later generalized to curves). Each face $f \in F$ has an associated normal n_f. Each line $l \in L$ has an associated 3d direction d_l.

Our goal is to reconstruct each object in 3D as a texture mapped mesh. We first solve for the orientation map of the image, i.e., compute the per pixel normal. We use the known 3D line directions for some of the lines (using vanishing points) and propagate this information across the faces and contours to cover the entire object. In particular, if we know the direction of *two* lines outlining a face, its normal can be computed. Similarly for any line outlining a face with a known normal, we can compute its direction in 3D. These steps are repeated in an alternating manner to compute the orientation map for the entire object. The details of this algorithm are present in Section 3. We further generalize this approach to curves in Section 4, in particular to objects with faces whose normals can be computed by sweeping their bounding curves towards each other.

Having computed the per-pixel normals we integrate this normal map to compute the depth for each pixel in the image, up to a global scaling factor. Given the projection matrix, we compute the ray direction for each pixel along which the corresponding 3D point must lie. We solve for the depth along these rays such that the surface is locally perpendicular to the computed normals at each pixel. While posing it as such leads to a non-convex problem, it can be approximated as a linear problem under mild assumptions. Details are present in Section 5, where we deal with occlusions as well.

3. Algorithm

In this section we describe the algorithm that we used in compute the orientation map. The first step is to group the lines into clusters of parallel lines, using vanishing points. We use [16] to identify the dominant clusters $\mathbb{C}$ of intersecting lines. To assign a cluster we require a minimum of 3 lines to intersect at (or pass near to) a vanishing point. Having computed these clusters we assign labels to the lines indicating their cluster membership, i.e., a line l belongs to a cluster if it passes through the corresponding vanishing point. But not all lines are assigned to some cluster — some may be left unassigned, as shown in Figure 2(b). Here we have 3 clusters with member lines shown in red, green and blue. The remaining lines are un-clustered and shown in black. While there are multiple methods for computing the projection matrix from a single image as described in [17], we use [18] to compute the Projection Matrix $P = [KR|0]$, where the internal camera matrix K is assumed to have no skew and unit aspect ratio, from a set of 3 orthogonal vanishing points. We now describe the constraints these known line directions can impose on other lines and faces.

3.1. Line Direction from Vanishing Points

Given a vanishing point v we can compute the line direction d_l of all the lines l passing through this vanishing point

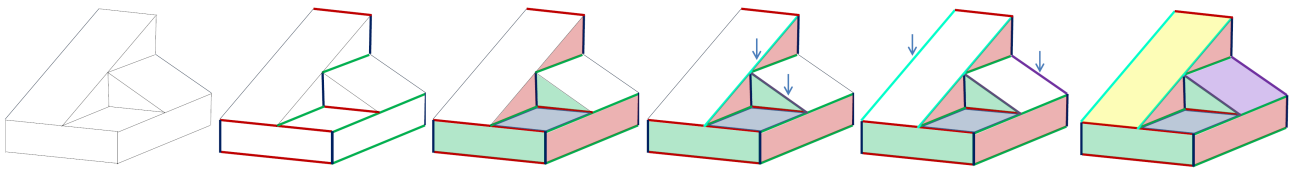

Figure 2. For a synthetic example, (a) shows the original image with the lines and faces. (b) shows the clusters of lines in red, green and blue, clustered using vanishing points. (c) shows the colored faces spanned by these lines whose normals can be computed. (d) shows how additional lines (cyan, purple) are computed as they line on faces with computed normals. (e) shows the line direction of the remaining lines that pass through the newly computed vanishing points is also known. (f) shows the completed labeling.

as $KRd_l = \lambda v$, which can be rewritten as

$$d_l \approx R^{-1}K^{-1}v, \tag{1}$$

where the $\approx$ implies equality to scale. Thus, we can compute the line direction corresponding to each cluster. Now we use these known line directions to compute orientations of faces that they outline (Section 3.2), and in turn use these face normals to compute the line directions for any line outlining them (Section 3.3). We alternate between these steps of computing face normals from line directions, as described below.

3.2. Face Normals from Line Directions

The normal to a face f is computed as

$$n_f = d_{l_1} \times d_{l_2}, \tag{2}$$

if l_1 and l_2 are two lines with known directions that outline the face. Using Eq2, we can compute many of the faces spanned by the by the clustered lines as shown in Figure 2(c).

3.3. Line Directions from Face Normals

Given the orientation of a face f we can now compute the 3D line direction of any line l bounding f. We know that $d_l.n_f = 0$. Further, consider the plane π_l containing the camera center and passing through l in the image. We know that $d_l.n_{\pi_l} = 0$. Thus, we compute d_l as,

$$d_l = n_{\pi_l} \times n_f. \tag{3}$$

We compute the line direction of all such lines bounding these faces with known normals to identify new line directions as shown in Figure 2(d). At this stage the line direction for other lines that pass through this newly discovered vanishing points (corresponding to d_l), are also determined as shown in Figure 2(e).

We repeat this process until we cannot get any new normals or line directions. At this stage if we have computed $n_f, \forall f \in F$, we have successfully computed the orientation map for the figure. However, this procedure is not guaranteed to cover all the faces and lines, indeed it will model the part of the scene that is *reachable* from the initial set of lines. If a face f and its outlines l, are not connected to the initial set of lines, then the face f will not be covered. The final result can be seen in Figure 2(f). Algorithm 1 summarizes our approach. We show results of running the algorithm on real images in Figure 3. The left column shows the user markings — the lines shown in red, green and blue, are clustered using vanishing points and have known 3D direction. The 3D direction of the black lines and the yellow curves are unknown. The top result demonstrates the algorithm — the normal of the face in purple is computed in the third phase of the algorithm and the normal to the cyan face is determined further in the 5th phase of the algorithm. This process is shown in detail in the supplementary video. In the next section we extend this approach to planar curves to enable us to generate the remaining results in Figure 3.

Figure 3. The left column shows the input image with the user markings. The colors red, green and blue indicate the lines whose directions are known. The lines in black and curves in yellow are unknown. The right column shows the orientation maps.

Algorithm 1 $[n_f, d_l] = \text{Normals}(L, F)$

Compute the cluster of parallel lines and associated vanishing points vanishing points
Compute $P = [KR|0]$
repeat
  Compute $n_f = d_{l_1} X d_{l_2}$ for all faces f spanned by *two* known directions
  Compute $d_l = n_f X n_{\pi_l}$ for any line lying on a face with known normal
  Compute $d_l = R^{-1}K^{-1}v$ for any un-clustered line l if it passes through any of the newly detected vanishing points v
until No new face or line detected
RETURN $n_f \forall f \in F$, $d_l \forall l \in L$

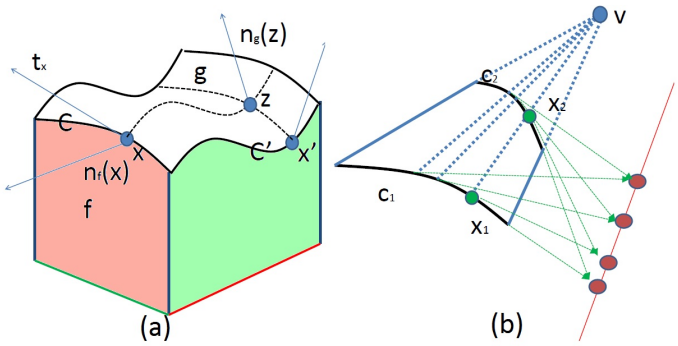

Figure 4. (a) curves c and c' lying on planes with known normals can be reconstructed in 3D, and this information can be further propagated to compute the normal at any point z on the face g. (b) shows a constraint on translated curves that determines corresponding points.

4. Extension to Curves

In the previous section we discussed faces bounded by lines, but the approach easily generalizes to curves. Curves are found in abundance in man made scenes, e.g., arches, doorways, etc. However, a face g spanned by a curve doesn't necessarily have a unique normal. Let $n_g(z)$ denote the normal to the face g at point z on it as shown in Figure 4(a). Given a curve c outlining the face f with known normals, consider an arbitrary point x on the curve with tangent direction t_x in the image. Let l be the the 3d tangent along the curve c at x. Then we compute the 3D tangent direction $d_l = n_{\Pi_l} \times n_f(x)$, as in the previous section. Computing the tangent direction at each point of the curve completely specifies the curve up to translation. Thus, similar to the discussion for lines, we compute the curve direction for any curve c in 3D, provided we know the normal to the face f that it bounds. This is shown in Figure 3 where the yellow curves lying on the frontal face of the Arch are so determined.

Once we have the tangent direction for each point on a curve, this information is further propagated to all other faces that this curve borders. In particular, consider the face g outlined by the curves c and c' and consider a point x' on the curve c'. Suppose we sweep x along the curve c' and x' along c shown by the 2 dotted curves in Figure 4(a). The normal at their intersection z is computed as the cross product of the (known) 3D tangent directions at x and x'. Similarly the normal at each point on the face g is computed to complete the orientation map. Thus we can compute the normal at any point of a face provided we know the curves that outline it. Using this, the normals on the underside of the Arch are computed as shown in Figure 3.

4.1. Translated Curves : Generalized Parallel lines

Consider two curves c_1 and c_2 translated in space by an arbitrary vector as shown in Figure 4(b). Translated curves can be seen as a generalization of parallel lines. Suppose we knew the correspondences between points on the curve, which is equivalent to knowing the translation direction between the curves. Then, given two corresponding points x_1 on c_1 and x_2 on c_2, we compute where the tangents to the two curves at these points intersect. Tangents at x_1 and x_2 are parallel in 3D. Thus, their intersection point is a vanishing point, and we can compute the 3D direction of the tangents following the previous discussion. Using this, we can compute the tangent direction of the translated yellow curves in the bottom result in Figure 3.

To compute the correspondence, consider the collection of tangent intersection (vanishing) points, one from each pair of correspondences (one pair of corresponding points is shown in green and their vanishing point in red in Figure 4(b)). As these are planar curves, all the vanishing points must lie on the vanishing line (red line in figure) corresponding to the plane on which the curve lies. Thus, we can go through all the possible translational vectors as hypothesis candidates (e.g., line directions already in the image) and choose the one for which all the vanishing points lie close. Thus, we can specify a curve provided we can identify its translated copy. Figure 5 shows some more results for urban scenes (middle column shows the orientation maps) computed using this approach.

5. Reconstruction

Given the normal at every pixel p in the image, we wish to solve for the depth of each pixel, using a an approach

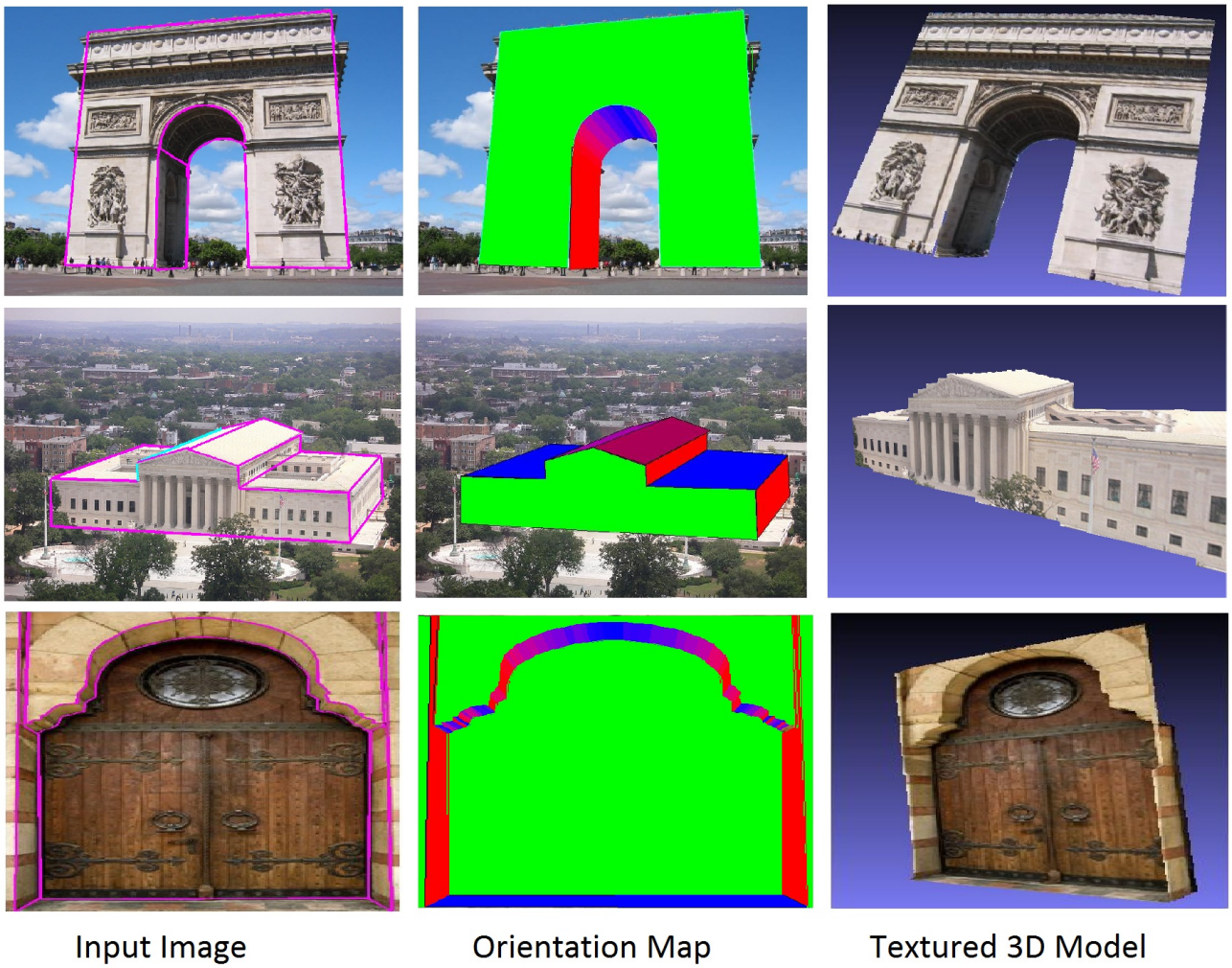

Figure 5. The figure shows the original image with the user markings (magenta shows the orientation discontinuities,and cyan show the depth discontinuities) in the left column, the orientation maps in the middle and the texture mapped results on the right.

similar to that used in photometric stereo [4, 5]. Given the projection matrix, we know R_p, the ray in 3D that projects onto the pixel p. Then $3d_p$, the 3D point corresponding to p, has one degree of freedom given by depth of the pixel z_p, and is computed as $3d_p = R_p \cdot z_p$.

We solve for the depth z_p along each ray such that the surface is locally perpendicular to the computed normal at each pixel. Let N be the set of pairs of pixels that are connected by a 4-neighborhood — for each pair of pixels $p, q \in N$, we compute the local surface direction as $(3d_p - 3d_q)$, and require that the normal at each point be perpendicular to this direction. For neighboring pixels across a depth discontinuity (occlusion), this requirement is dropped. It is worth noting that as long as a region of the image is not completely bounded by occluding edges, the depth for each pixel in the image is computed in a consistent manner. We setup the optimization problem as

$$\begin{aligned} \underset{z,3d}{\text{Minimize}} \quad & \sum_{p,q \in N-X} |(n_p + n_q) \cdot (3d_p - 3d_q)| \\ \text{subject to} \quad & 3d_p = R_p \cdot z_p, \ \forall p \\ & z_{min} >= 1. \end{aligned}$$

Here X is the set of neighboring pixels across a depth discontinuity. We set the minimum depth z_{min} to be greater than equal to 1 (an arbitrary constant, which fixes the global scale ambiguity). Note that we are making an approximation here — we would like to minimize the normalized dot product i.e., $\frac{(n_p+n_q)\cdot(3d_p-3d_q)}{||3d_p-3d_q||}$. However, posing it as such makes the problem non-convex. Ignoring the denominator (i.e., assuming that changes in depth are small between neighboring pixels) makes the problem linear. This assumption works well in practice; the effect will be significant only when where $||3d_p - 3d_q||$ is large, i.e., at depth discontinuities where normal integration fails anyways. Fur-

ther, this could be used as an initial guess for iterative non-convex methods, though we did not see the need for this in our experiments. Once we compute the 3d mesh, we map the image onto it to get a texture-mapped model. The right column in Figure 5 shows the results for some urban scenes. For the middle result, note that the inclined planes occludes the back horizontal face and there is a depth discontinuity (indicated in cyan). Please see the supplementary video for more results.

6. Automation

In the previous sections, line directions and face normals were propagated to compute orientation maps. There, we relied on the user to draw the boundaries; here we automate these steps. The first step is to apply image preprocessing to get a collection of lines. We apply Canny edge detection [19] to an image and then extract and cluster lines using the method described in [16]. The result using edge detection and clustering is shown in Figure 6(a). The colored lines (R, G, B) correspond to the known clusters. The black lines are the lines that are currently not clustered, and their directions are unknown. As can be seen, some lines of interest such as orientation or depth discontinuities are missing or incomplete. Further, spurious edges such as surface markings or background clutter make the problem challenging. Thus, we do not recover clean segmented faces and in general a face might have missing lines, or bounding lines that don't intersect at a corner.

To overcome the first challenge of defining faces (and their normals), we use the approach of Lee et al. [3]. They introduced a line sweeping approach where a detected line l having one of the Manhattan directions d_l is swept towards the vanishing point of another line $l^{'}$ and (Manhattan) direction $d_{l'}$ to sweep out an area which is likely to have the normal $d_l \times d_{l'}$.

We generalize their sweeping approach to handle non Manhattan lines, to carve out regions of the image where we can infer the orientation. Thus if we have $|\mathbb{C}|$ clusters of lines (each cluster corresponding to a 3D direction and a vanishing point) - we compute $\binom{|\mathbb{C}|}{2}$ regions. It is possible that some of these regions might be empty or may have the same normal, in which case they are merged. We briefly describe the approach as follows. Consider two clusters α and β, and let $l_\alpha \in \alpha$ and $l_\beta \in \beta$ be any two lines in these clusters as shown in Figure 8. We sweep l_α towards v_β, the vanishing point for β cluster, to sweep out a red region $O_{\alpha\beta}$ until we intersect a line from another cluster, l_γ. Similarly $O_{\beta\alpha}$ is computed (blue trapezoid). Then

$$R_{\alpha,\beta} = O_{\alpha\beta} \cap O_{\beta\alpha} \tag{4}$$

is the intersection of these regions (shaded in the figure), and is one face associated with the lines from clusters α and β. If any two faces with different normals ever intersect at any pixel (i.e., there is a conflict), then that pixel is removed from both the faces. The result of the sweeps is shown in Figure 6(b), where two faces (and their normals) have been identified and shown in red and blue.

As with the interactive case, we wish to compute the line direction d_l for an un-clustered line l if it spans a face f with a known normal. To determine if a line belongs to a face known to spanned by lines in α and β, we sweep each unclustered line l, towards the vanishing direction v_α and v_β and check if we reach the face without intersecting any line from some other cluster l_γ, $\gamma \neq \alpha, \beta$. If so, the line l is deemed to bound the face f. The 3D line direction can then be computed as in the interactive case. This is shown in Figure 6(c) where the directions of the yellow and cyan lines are now inferred. Further, we check if any of the other unclustered lines pass through these newly found vanishing points. If so, they are added to the cluster, as shown in Figure 6(d), where the direction of two additional lines are determined and shown in color. We now repeat this sweeping process to determine new faces and lines until the algorithm terminates, giving us a partial orientation map shown in Figure Figure 6(e).

However, this orientation map contains holes and missing regions. We interpolate the orientation map solving for the missing normals via an optimization minimizing the difference between the normals, $|n_p - n_q|$ (L_1 norm) across neighboring pixels p, q, except for pairs of pixels across edges in the images. The completed orientation map is shown in Figure 6(f).

The sky region is removed by matting out the pixels (clustering the image in HSV space) to get an orientation map for the buildings as shown in Figure 6(g). Now we generate the texture mapped building model shown in Figure 6(h). The top result in Figure 7 shows another scene for which 4 clusters of lines are detected, shown in red, green, blue and yellow.

6.1. Extension to curves

The method of sweeping generalizes naturally to curves as well–to sweep a line along a curve, the curve is broken into tiny segments, and the line is swept towards the vanishing point of each segment to define a face. Similarly to check if a curve belongs to a face, each segment of the curve is swept towards the vanishing point of the bounding lines of the face and checked if it intersects the face, without intersecting any other line. While there are parametric approaches to detecting curves such as [20], we detect curves as large connected components in the edge image with smoothly varying image gradients. The bottom two results in Figure 7 show the results for scenes with curves. The first one computes the direction of the yellow curve by inferring that it lies on a face (shown in green) with a known

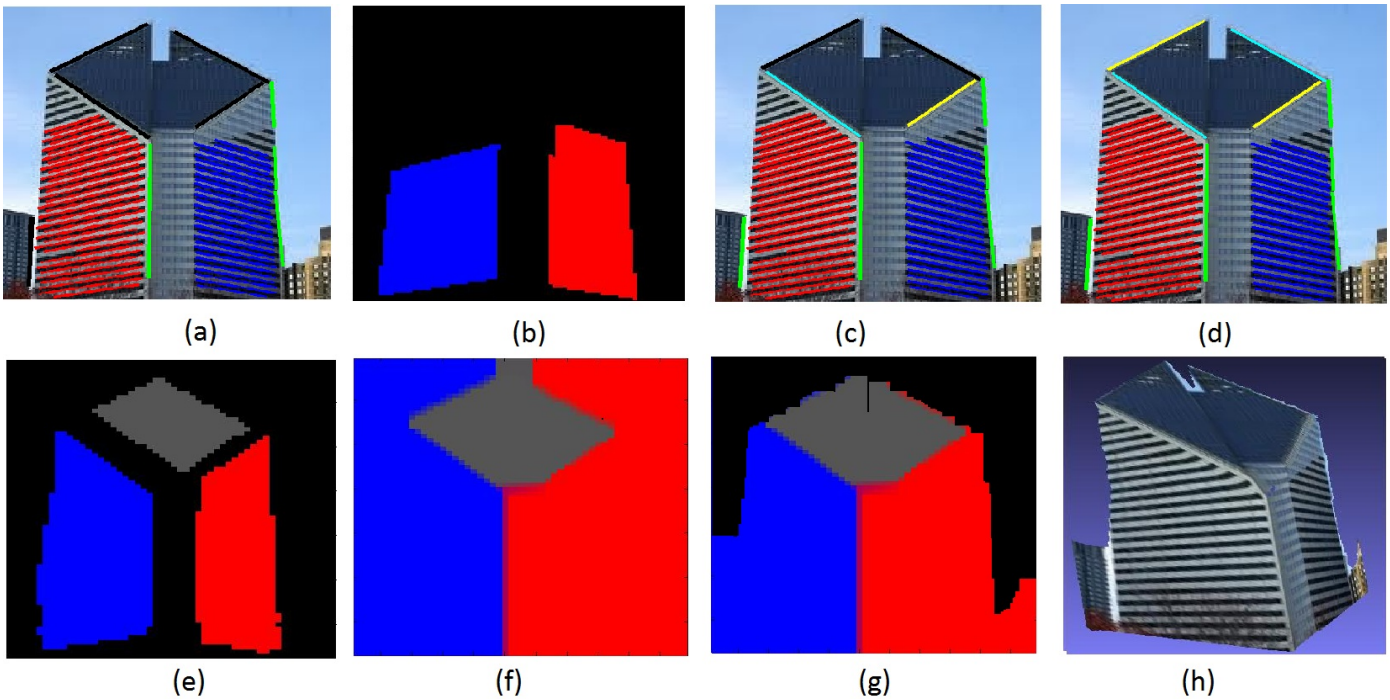

Figure 6. The figure (a) shows the original image with the detected and clustered lines in colour (unclustered are shown in black), (b) shows the result of the first sweep, (c) shows the algorithm after the directions for the un-clustered lines belonging to the face have been computed, (d) shows the result after remaining lines have been assigned to the clusters, (e) result at the termination of the algorithm, (f) shows the smoothed orientation map, (g) shows the orientation map after removing the sky pixels (h) shows the texture mapped result

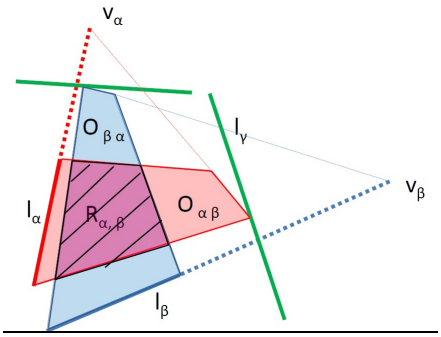

Figure 8. The figure shows how we sweep lines belonging to one cluster towards the other, and the process of associating faces with lines

normal. The next one computes the curve direction by identifying them as translated curves (note how the curves are rectified and horizontal in the reconstruction). While the results for the automatic case are for buildings where the sky can easily be segmented, the scenes have lines and curves in arbitrary directions (non-Manhattan directions), a problem that has not been previously studied.

7. Conclusions

We present a novel approach to the SVR problem, centred around the notion of faces and their bounding lines and curves, that enables propagating information across them. This allows us to model a more general class of piecewise swept surfaces than was previously possible from a single photo. While allowing for lines and curves in arbitrary directions, a limitation of this work is the reliance on the detection of these lines and curves. In the interactive setting, the system requires users to mark out occlusion boundaries; it may be possible to automate this step, e.g. [6]

For future work, it would be interesting to study the degrees of freedom present in objects not fully determined by just drawing their boundaries, e.g., a pyramid has one degree of freedom in its height. Further, our approach currently makes hard decisions about normals and line directions – a probabilistic approach here might be able to resolve conflicts better. Another direction would be to use symmetry, present in abundance in man made scenes, to further constrain the solution.

Acknowledgements: This work was supported in part by National Science Foundation grant IIS-0963657, the University of Washington Animation Research Labs, and Google.

References

[1] A. Criminisi, *Accurate Visual Metrology from Single and Multiple Uncalibrated Images*. Springer, 2001. 1

[2] P. Sturm and S. J. Maybank, "A method for interactive 3d reconstruction of piecewise planar objects from single im-

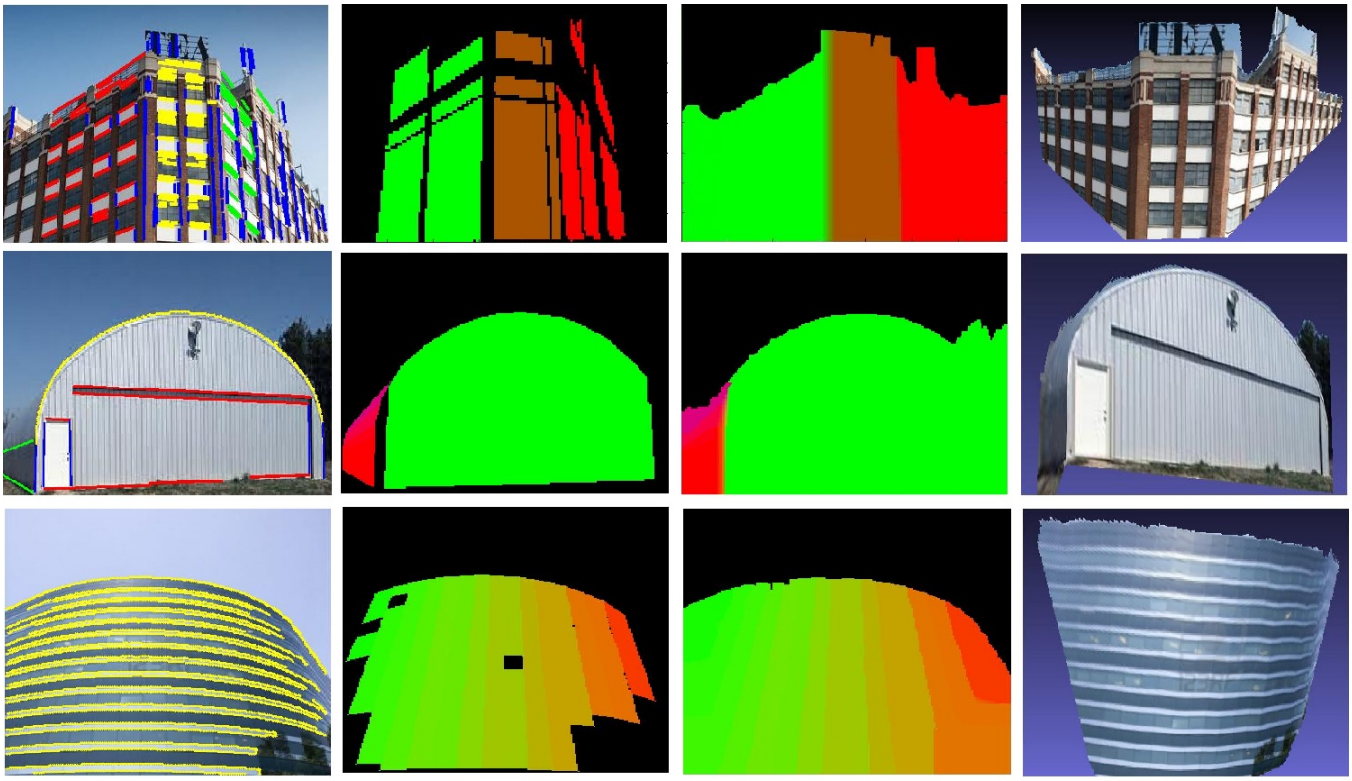

Figure 7. The figure (a) show the original image with the detected line and curves. The clustered lines are shown in colour, the unclustered in black. The curves are shown in yellow. (b) shows the orientation map at end the algorithm (f) shows the orientation map after smoothening and matting out the sky pixels (g) shows the texture mapped result

ages," ser. BMVC'99, pp. 119–126. 1

[3] D. Lee, M. Herbert, and T. Kanade, "Geometric reasoning for single image structure recovery," ser. CVPR'09, pp. 2136–2143. 1, 2, 6

[4] P. Kovesi, "Shapelets correlated with surface normals produce surfaces," ser. ICCV'05. 1, 5

[5] E. Prados and O. D. Faugeras, "Perspective shape from shading and viscosity solutions," ser. ICCV'03, pp. 826–831. 1, 5

[6] D. Huffman, "Impossible objects as nonsense sentences." in *Machine Intelligence 6*, 1971. 1, 7

[7] T. Kanade, "A theory of origami world." in *Artificial Intelligence 13*, 1980. 1

[8] Y. Horry, K. Anjyo, and A. K., "Tour into the picture: Using a spidery mesh interface to make animation from a single image," in *SIGGRAPH '97*. 1

[9] M. Prasad, A. Zisserman, and A. Fitzgibbon, "Single view reconstruction of curved surfaces," ser. CVPR'06, pp. 1345–1354. 1

[10] L. Zhang, G. Dugas-Phocion, J.-S. Samson, and S. M. Seitz, "Single-view modelling of free-form scenes," *Journal of Visualization and Computer Animation*, vol. 13, no. 4, 2002. 1

[11] Z. A. Utcke, S., "Projective reconstruction of surfaces of revolution." in *Lecture Notes in Computer Science*, 2003. 1

[12] C. Wu, S. Agarwal, B. Curless, and S. M. Seitz, "Schematic surface reconstruction," ser. CVPR'12, pp. 1498–1505. 1

[13] A. Saxena, M. Sun, and A. Y. Ng, "Make3d: Learning 3d scene structure from a single still image," *IEEE Trans. PAMI 31*, pp. 824–840, 2009. 1

[14] K. Karsch, C. Liu, and S. B. Kang, "Depth extraction from video using non-parametric sampling," ser. ECCV'12. 1

[15] A. Flint, C. Mei, D. Murray, and I. Reid, "A dynamic programming approach to reconstructing building interiors," ser. ECCV'10, pp. 394–407. 2

[16] J. Kosecká and W. Zhang, "Video compass." ser. ECCV'02, pp. 476–490. 2, 6

[17] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer-Verlag, 2010. 2

[18] T. D. R. Cipolla and D. Robertson, "Camera calibration from vanishing points in images of architectural scenes," ser. BMVC'99, pp. 38.1–38.10. 2

[19] J. Canny, "A computational approach to edge detection," in *IEEE Trans. PAMI*, 1986. 6

[20] D. Ballard, "Generalizing the hough transform to detect arbitrary shapes," in *Pattern Recognition Vol. 13*, 1981. 6