Volume 0 (1981), Number 0 pp. 1–11

Peek-in-the-Pic: Flying Through Architectural Scenes From A Single Image

Amit Shesh and Baoquan Chen†

University of Minnesota at Twin Cities

Abstract

Many casually taken "tourist" photographs comprise of architectural objects like houses, buildings, etc. Reconstructing such 3D scenes captured in a single photograph is a very challenging problem. We propose a novel approach to reconstruct such architectural scenes with minimal and simple user interaction, with the goal of providing 3D navigational capability to an image rather than acquiring accurate geometric detail. Our system, ***Peek-in-the-Pic****, is based on a sketch-based geometry reconstruction paradigm. Given an image, the user simply traces out objects from it. Our system regards these as perspective line drawings, automatically completes them and reconstructs geometry from them. We make basic assumptions about the structure of traced objects and provide simple gestures for placing additional constraints. We also provide a simple sketching tool to progressively complete parts of the reconstructed buildings that are not visible in the image and cannot be automatically completed. Finally, we fill holes created in the original image when reconstructed buildings are removed from it, by automatic texture synthesis. Users can spend more time using interactive texture synthesis for further refining the image. Thus, instead of looking at flat images, a user can fly through them after some simple processing. Minimal manual work, ease of use and interactivity are the salient features of our approach.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Geometric Algorithms and Systems

1. Introduction

In the current age of digital photography, people take many casual photographs of places visited. Although these photographs serve as visual records, they do not create a sense of "being there". It would be much more exciting to experience the actual 3D scene by flying through it, rather than looking at a flat image. This paper addresses this goal.

Extracting geometry from images is a very challenging task. As a single method is unlikely to reconstruct various entities contained in photographs like buildings, vegetation, etc., most approaches concentrate on particular classes of entities (faces [LTM96], trees [RMMD04], etc.). We observe that many photographs largely capture buildings and other forms of architecture. Our system is designed to create geometry from such casually captured/created images.

Geometry captured in images can be acquired by *instantiation* (associating and aligning pre-defined building blocks with the image), or *reconstruction* (actually reconstructing the captured geometry). Instantiation may work well in many situations, but conceptually it offers a very technical user interaction, unlikely to be intuitive for the typical user. Reconstruction of geometry from images has been rigorously studied. If multiple images are available, photogrammetry and computational stereopsis [MB95, Sze94] can be effective. This problem is much more challenging if only one image for a scene is available (typical for the casual photographer). If very crude approximations of geometry suffice for an application, automatic (Photo Pop-up [HEH05]) and interactive methods (spidery-mesh interfaces [HiAA97]) can be employed to quickly construct them. For more complete geometry, ortho-rectification approaches based on homographies [LCZ99] can be useful. However these methods require significantly "technical" human intervention (specify-

† ashesh|baoquan@cs.umn.edu

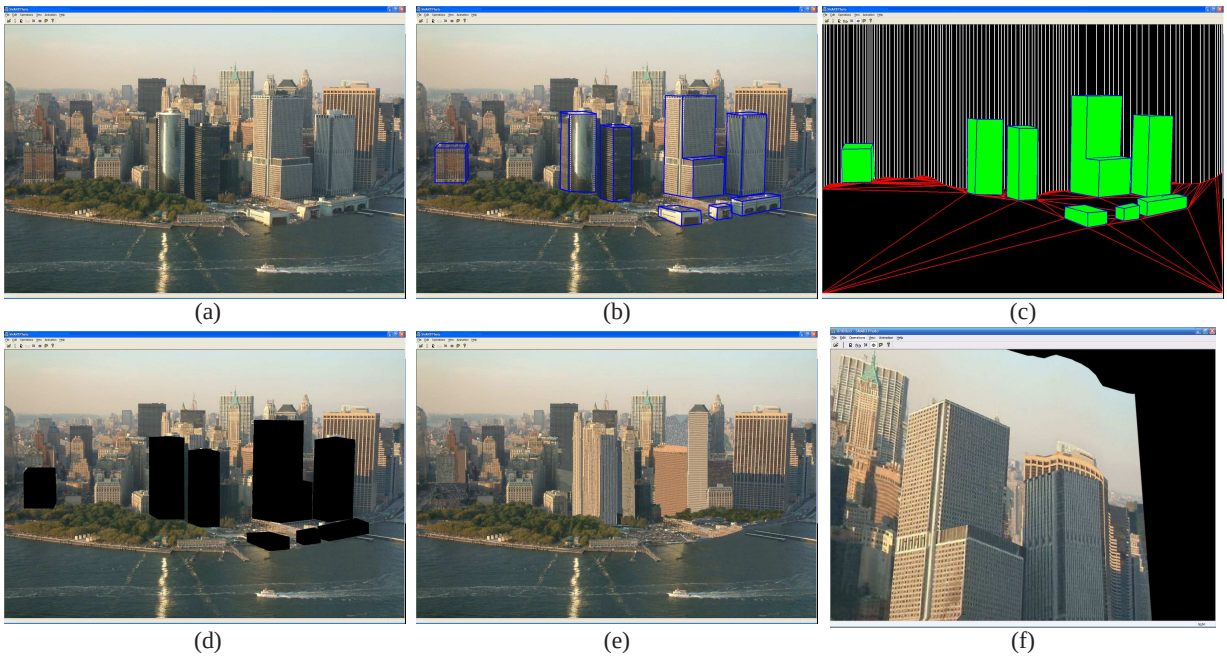


Figure 1: *Partial geometry reconstruction of lower Manhattan, New York City from a single image. (a) the original image. (b) all line drawings made by the user (figure shows all traced lines; actually, one building is traced and reconstructed at a time). (c) eight reconstructed buildings, with the ground relief and the background. (d) the original image with holes to be synthesized. (e) the synthesized image for background and ground geometry. (f) an alternate view of the city.*

ing relationships between planes, understanding the concept of vanishing points/lines, etc.) and in some cases may require carefully-planned photographs. We strive to devise a method that requires minimal *and* simple user interaction. Indeed, our system combines different disparate bodies of research in sketch-based geometry reconstruction, vision-based geometry reconstruction and image completion to provide a complete image-to-navigable-geometry tool. Many operations of our pipeline are fully automatic, with interactive alternatives to improve the automatic results.

In our system, user interaction is limited to simply tracing out buildings with lines; we treat these traces as 2D line drawings and reconstruct them. This unique treatment of reconstructing geometry from an image finds some common ground with reconstructing geometry from freehand sketches. But most of the work done in this area [LS96, SC04] applies only to orthographic drawings. *Peek-in-the-Pic* extends their approaches to perspective line drawings. It is designed such that the image ends up being used only as a guide to trace objects well. Without the context of image navigation, it functions as a general tool to reconstruct geometry from perspective sketches. This increases its scope of usage to many design-by-sketches applications.

This paper expands on a short conference paper [SC05], and discusses our approach, formulation and implementation of the problem in detail. The paper is organized as follows: Section 2 provides an overview of related work in two contexts, image-based modeling and sketch-based reconstruction. Section 3 provides an overview of our system. We introduce a simple camera calibration step in Section 4. Section 5.1 explains how various constraints are placed on an object to be reconstructed. Section 5.2 first summarizes reconstruction of geometry by optimization, and then provides details of how we re-formulate it for perspective line drawings. Section 6 explains how ground and background geometry is added to complete the 3D scene. Section 7 discusses how textures are obtained for the reconstructed geometry from the original image. We discuss results in Section 8 and conclude with a discussion in Section 9.

2. Related Work

2.1. Image-based Modeling

Image-based modeling reconstructs a scene from multiple photographs taken from various viewpoints by identifying correspondences between points in different images. If a sufficient number of photographs of a scene are available, reconstruction is a mathematically soluble, albeit semi-automatic problem. The *PhotoModeler* program† works on

† http://www.photomodeler.com

multiple images and concentrates on getting precise architectural measurements. The *ARBA3D* program[‡] uses only two images, but their absolute positions and correspondence between them have to be manually specified. Software like *SilverEye*[§] are limited to reconstructing geometry from satellite images. The *Canoma*[**] program works only on one image, but is based on *instantiation*, whose potential drawbacks have been discussed earlier.

Herman *et al.*'s work [HK87] attempts to reconstruct 3D models of large scenes using multiple aerial images. Debevec *et al.* [DTM96] interactively build an approximate model of the photographed scene that is then refined by locating image correspondences from multiple images. Reconstructing 3D models from single images, being mathematically insoluble, usually involves making assumptions about the apparent geometry in an image [Kan81], or attempting to fit simple 3D objects (cubes, wedges, etc.) of known topology to a line drawing obtained from the image [Rob63]. Seminal work by Huffman [Huf71] on the notion of labeling contours in an image to infer their geometric nature forms the basis of many approaches to reconstruct 3D geometry from 2D inputs [SG00, KH06, KC06]. Tour-Into-The-Picture [HiAA97] provides a spidery-mesh interface for the user to manually locate vanishing points of the image. Automatic Photo Pop-up [HEH05] automatically creates billboards and "folds" from a single image to create a "pop-up" effect. Both these approaches generate very crude approximations of geometry in the scene, which limits the types of scenes they can navigate and the freedom of navigation itself. Typical vision-based approaches [LCZ99, CDR99, CRZ00] perform reconstruction of a single image by calculating camera parameters, using vanishing lines and point correspondences on each plane. Such a "reconstruct-plane-by-plane" interface may be excessive for our application where accurate geometry is not desired. Also, such methods may not work correctly without a lot of interaction for planes that do not contain parallel edges from which vanishing lines can be computed (like pyramids). Oh *et al.* [OCDD01] regard reconstruction as a *Photoshop-like* operation termed *depth painting*. They segment the photograph into layers and assign depth coordinates to every pixel of every layer to produce convincing depth images, but at the cost of heavy and time-consuming user interaction. *Peek-in-the-Pic* works at the object level instead of the pixel level.

2.2. Reconstruction of Line Drawings

Reconstruction of geometry from freehand 2D sketches is another related area of research and is related to modeling from images in many ways. Although many sketching metaphors exist (Teddy [IMT99] for making rotund objects, SKETCH [ZHH96] for design by extrusion), approaches that use actual 2D projections to infer 3D geometry are more relevant to our work. Reconstruction of 3D models from line drawings is a very old research problem (Barrow *et al.* [BT81], Marill [Mar91], Leclerc *et al.* [LF92], Varley [Var03] are excellent examples). Lipson *et al.* [LS96] and Shesh *et al.* [SC04] formulate reconstruction as an optimization problem by evaluating the 2D input for various 2D image regularities like parallelism and orthogonality of lines and replicating their corresponding 3D properties in the geometry. Moreover while both reconstruct wire frame drawings (with hidden parts drawn), we reconstruct "line drawings" (with hidden parts unavailable). Most image regularities in many of these approaches are robust only if the underlying sketch is orthographic. We extend the optimization formulation to support perspective images and use gestures to provide geometric regularities that could otherwise be obtained implicitly from orthographic images. After the user "traces" out a building, we use this formulation to reconstruct its 3D geometry. Several other interesting approaches use a minimal set of lines (for example, only silhouettes and creases) to produce a plausible set of 3D curved surface models from them using simple mathematical constraints [PF06]. Kaplan *et al.* [KC06] propose an iterative user interaction process to progressively refine the constraint space and make the overall problem tractable.

3. Algorithm Overview

The general theme of *Peek-in-the-Pic* is "trace and reconstruct". Figure 1 illustrates our pipeline. An object is converted into a line drawing by tracing out its edges (Figure 1(b) shows all such tracings). Note that only the visible parts of an object can be drawn this way. They are consolidated to form a 2D graph and loops representing the faces of the object are determined. The graph is then analyzed to infer structural constraints on the object to be reconstructed. The user can also specify constraints via simple gestures (Section 5.1). The object is then reconstructed by solving an optimization problem (Section 5.2) that considers various structural constraints placed on it and camera parameters obtained in Section 4. Hidden parts of the object can be completed automatically or by interactive sketching (Section 5.3). The ground geometry is obtained after all desired objects have been reconstructed (Section 6, reconstructed geometry shown in Figure 1(c)). Holes in the image resulting from construction (Figure 1(d)) are filled automatically and can be refined using a simple interface(Section 7, Figure 1(e)). Finally, textures are mapped on all constructed geometry using the original and the synthesized images for navigation.

4. Camera Calibration

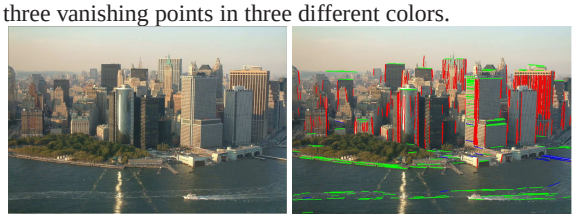
Camera parameters like focal length f of the lens and principal point p_0 are required to correct the perspective distortion

‡ http://www.arba3d.com
§ http://www.geotango.com/products/silvereye.htm
** http://www.canoma.com

in the image before it is reconstructed. Camera calibration by planting a known object in the scene [WT91] cannot be performed in our case as the photographs we strive to reconstruct are unplanned and casually taken. There are several existing methods to estimate camera parameters from a single image. We use the method by Cipolla *et al.* [CDR99] to calculate camera parameters assuming a simple camera. This method requires knowledge of the vanishing points of an image.

Automatic detection of vanishing points in an image has been a subject of research in computer vision [Rot00, MA84, EMJK94]. A common approach is to detect lines in the image and determine points in the image domain where large number of such lines intersect. Intersections can be determined by considering the domain to be the infinite image plane [Rot00] or projecting them as great circles onto a Gaussian sphere and clustering their intersections [EMJK94]. We follow the former approach and use a RANSAC algorithm [FB81] to determine likely vanishing points.

We begin by finding all edges in the image using a Canny edge filter [Can86] and linking them to find line segments in the image. We find the three vanishing points using the RANSAC algorithm as follows: we randomly choose two lines and determine their point of intersection v. We then count the number of lines in the remaining set that this point is close to (we set the threshold subtended by v and one of the end points of the line onto the other end point to be 5°). If this count is greater than that in the previous iteration, we select v as a candidate vanishing point. In the next iteration, we randomly pick another pair of lines and proceed similarly. After a sufficient number of iterations (one-third of the total number of detected lines in our implementation) we record the resulting point as a vanishing point and remove all lines that pass through or near this point from the set of lines. We repeat the algorithm twice to retrieve the remaining two vanishing points. If the randomly selected pair of lines is parallel to each other, the point is at infinity and we use their common direction to test the point against all remaining line segments. The figure below shows the lines used to get the three vanishing points in three different colors.



Previous work and our experiments indicate that this algorithm is prone to failures and errors in some cases: spurious edges, Canny thresholds and vanishing points approaching infinity leading to precision errors. When this occurs, we revert to a manual approach: the user select three pairs of parallel lines that are mutually perpendicular to each other. These pairs give the three vanishing points.

The three vanishing points constitute a triangle T whose sides are the vanishing lines. Then, p_0 is the orthocenter of T, while f is a function of ${\lambda_1}^2$,${\lambda_2}^2$,${\lambda_3}^2$, where the λ_i's are the areas of triangles subdivided by p_0 in T. A perspective matrix P is constructed from these two parameters, that is then used during optimization. We refer the reader to [CDR99] for further details.

5. Reconstruction of Object Geometry From Perspective Line Drawings

The user now traces the visible edges of a building to be reconstructed from the image, forming a perspective line drawing (since photographs are perspective by nature)[††]. All lines are consolidated into a 2D graph G of vertices and edges. Clustering [SL97] is used for this consolidation. (Visible) faces of the object are then determined using the modified Dijkstra algorithm proposed by Shesh *et al.* [SC04]. All vertices of the graph that are on the ground are determined by starting at the lowest vertex of the graph (obviously on the ground). A breadth-first search of the graph is initiated from this vertex. Any edge from the current vertex that makes a small enough angle (say Θ) with the horizontal is assumed to lie on the ground. In our current implementation, we use $\Theta = 35^\circ$. In case the program does not select these vertices correctly, the user can manually specify them.

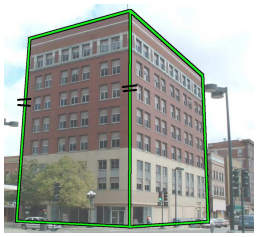
5.1. Constraint Specification

Once a 2D graph representing the traced object is compiled, constraints are imposed on its 3D structure that are used during optimization. Although similar constraints have been used to refine crudely approximated 3D geometry from photographs [CFD02], the aim of our constraints is to approximate 3D geometry from 2D inputs. Techniques explained in [LS96, SC04] cannot be directly applied to perspective images because these images cannot provide the same image regularity cues. For example, two parallel 3D lines are almost never parallel in perspective projection. We assume some constraints *implicitly* and also allow the user to explicitly specify them.

Some general observations can be made about architectural objects: they are attached to the ground, the walls touching the ground *most likely* rise vertically upwards, many edges tend to be parallel or perpendicular to each other, etc. These observations can be *implicitly* used as constraints on the geometry of the object. As the line drawing is in perspective projection, only a few constraints can be obtained from image regularities. The user can specify more

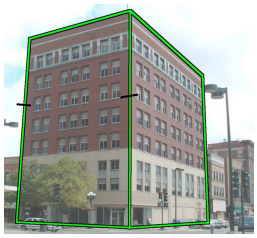
†† Our experiments indicated that detecting *only* the visible edges of a building is much more difficult to achieve through automatic edge detection than detecting all straight-line edges, as was needed to automatically infer the vanishing points. This is largely because of a large number of "spurious" edges produced by textures.

constraints *explicitly* through gestures. The three vanishing points (and hence the vanishing lines) obtained during camera calibration are used to derive more parallel lines. In any face, if two edges intersect at or near a vanishing line, they are assumed to be parallel to each other. It is also assumed that all edges that have exactly one vertex on the ground are perpendicular to the edges on the ground. This observation is not true for all objects–a notable example being a pyramid. But as these constraints are used as penalties in our optimization, they are not hard constraints. In fact, Figure 5(d-f) show the construction of the pyramidal head of a tower.

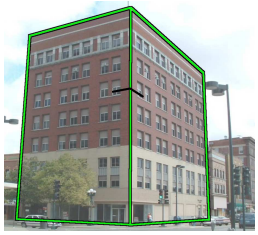


Making two edges parallel to each other

Making two edges perpendicular to each other



Making two edges congruent to each other



Making two faces perpendicular to each other

Additional geometric constraints can be specified by the use of gestures. We currently support four types of constraints. These constraints can also be specified *after* the object has been reconstructed; the reconstruction algorithm is rerun in this case. Constraints assumed implicitly or imposed by the user are used to derive other constraints. For example, $(e_1 \| e_2) \wedge (e_2 \| e_3) \Rightarrow (e_1 \| e_3)$.

5.2. Optimization for Geometry Reconstruction

Given a 2D graph G of the traced object and structural constraints, we reconstruct 3D geometry that represents the object's projection in the photograph. Specifically, we "inflate" G by assigning suitable depth to each vertex.

This problem finds some common ground with that of reconstructing 3D geometry from 2D sketches. Sketch reconstruction by inflation was used earlier by Leclerc *et al.* [LF92] and subsequently by Lipson *et al.* [LS96] and Shesh *et al.* [PMC03, SC04]. Various constraints like parallelism and perpendicularity of edges and faces are set up by evaluating 2D image regularities in the sketched image and Z-coordinates are given to each vertex of the graph to satisfy these constraints. However, all these methods assume orthographic drawings with image regularities that provide most of the constraints used by their optimization process. Most of the assumed image regularities do not exist for perspective line drawings. We extend their work by following a similar framework to reconstruct *perspective* line drawings through constraints explained in the previous section. All geometric constraints are incorporated into the compliance function as penalties.

It must be noted that inflation of the graph G by assigning a Z-coordinate to each vertex does not result in the actual correct geometry, as G originally represents a distorted projection of the object. Therefore, we use the perspective matrix P (Section 4) to undistort a candidate 3D graph *before* evaluating all characteristics. The resulting 3D graph projects onto the region occupied by it in the image.

The compliance function that the optimization attempts to minimize is of the form:

$$f = w_i * f_i$$

where $w = [w_i]$ is a weighting factor and f_i are various terms calculated as below. In practice, we use a weighting vector of $(\frac{1}{3}, \frac{2}{3})$ which was obtained empirically. The following notation is used henceforth:

v_i	:	i^{th} vertex of the graph G
f_i	:	i^{th} face in the graph G
$\vec{v_i}$	:	3D vector representing edge e_i
$\hat{v_i}$	:	Normalized 3D vector representing edge e_i
$\|\vec{v_i}\|$	:	Magnitude of vector $\vec{v_i}$
$\hat{n_i}$	:	Unit normal vector of face f_i
$n(G)$	:	Number of vertices in the graph G
$e(G)$	:	Number of edges in the graph G
$f(G)$	:	Number of faces in the graph G

The various terms f_i used are as follows:

1. Face Planarity
   This constraint ensures that all faces of the objects are planar. A plane is fit on all vertices on each face f_i and the sum of distances of each vertex from its fit plane comprises this term.
   $f_1 = \sum_{i=1}^{f(G)} \sum_{v_j \in face\ F_i} |a_i * x_j + b_i * y_j + c_i * z_j + d_i|$
2. Geometry Constraints
   This set of terms is used to evaluate all the geometric constraints compiled earlier.
   a. Parallelism of edges
      For all pairs e_i and e_j of edges (total $n_{parallel}$) that are supposed to be parallel to each other,
      $t_1 = \frac{\sum_{e_i \| e_j} (1 - |\hat{v_i} \cdot \hat{v_j}|)}{n_{parallel}}$
   b. Perpendicularity of edges
      For all pairs e_i and e_j of edges (total n_{perp}) that are supposed to be perpendicular to each other,
      $t_2 = \frac{\sum_{e_i \perp e_j} |\hat{v_i} \cdot \hat{v_j}|}{n_{perp}}$

c. Congruence of edges
For all pairs e_i and e_j of edges (total n_{cong}) that are supposed to be equal in length to each other,

$$t_3 = \frac{\sum_{e_i=e_j} \frac{abs(\|\vec{v_i}\| - \|\vec{v_j}\|)}{max(\|\vec{v_i}\|, \|\vec{v_j}\|)}}{n_{cong}}$$

d. Perpendicularity of faces
For all pairs f_i and f_j of faces (total n_{fperp}) that are supposed to be perpendicular to each other,

$$t_4 = \frac{\sum_{f_i \perp f_j} |\hat{n_i} \cdot \hat{n_j}|}{n_{fperp}}$$

e. Edge-face perpendicularity
For all pairs e_i and f_j (total n_{efperp})of such that edge e_i is perpendicular to face f_j,

$$t_5 = \frac{\sum_{e_i \| f_j} |\hat{v_i} \cdot \hat{n_j}|}{n_{efperp}}$$

$$f_2 = 0.2 * \sum_{i=1}^{5} t_i$$

We use Brent's minimization [PTVF02] to solve the above optimization problem, as it offers a good tradeoff between speed and accuracy. We use the layered method of Shesh *et al.* [SC04] for a good initial guess: all vertices on the silhouette form a middle layer, visible vertices not on the silhouette form the front layer and hidden vertices not on the silhouette form the back layer. Although this initial guess works well for closed objects, it tends to fail in case of incomplete objects composed of facades, like Figure 5(j-l). In that case we resort to the trivial initial guess (all vertices with the same Z-coordinate) and rely more on implied and gestured hints to reconstruct the 3D model.

5.3. Completing Object Geometry

The user can trace only those parts of the objects that are visible in the image. In order to complete the building geometry, hidden edges must be estimated. In general, this is a difficult problem because there can theoretically be an infinite number of configurations for hidden geometry. However, in case of architectural buildings, a reasonable hidden topology can be estimated automatically.

If we assume that all buildings are trihedral, this problem can be solved automatically. Our solution is similar to that offered by Varley *et al.* [VM00], but for perspective line drawings. We first detect vertices that have degree two (i.e they have an edge missing). Then, we estimate the direction of the missing edge by pairing the visible edges at these vertices with the known vanishing points. Then, we iteratively select two incomplete vertices, and determine the intersection of the two rays along their respective hidden edge directions. In order to prune incorrect pairs of vertices, we constrain all these intersections to lie within the convex hull of the building (implicitly sketched by the user). Thus we keep eliminating vertices, until we are left with only two vertices (that we simply connect to each other). Below, the figures in the top row show the input sketched by the user (in blue) and the automatically completed line drawing (in green).

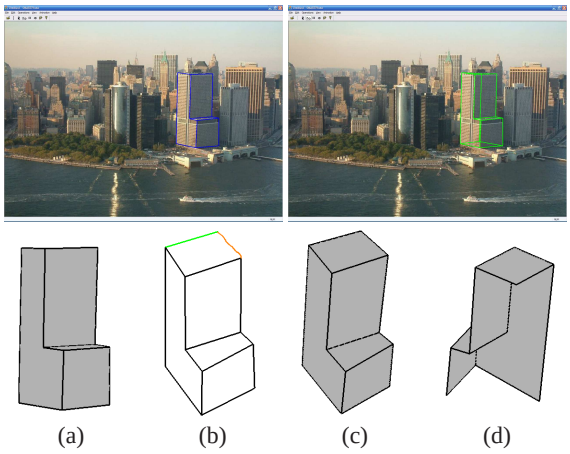


Figure 2: *Geometry completion. First row: automatic geometry completion. If the objects are assumed to be trihedral, the simplest hidden geometry can be automatically estimated using a variant of the technique by Varley* et al. *[VM00]. Second row: interactive geometry completion. (a) an incomplete building in its original position in the scene. (b) it is rotated and a missing face is drawn. (c) as the face is completed, it is reconstructed. (d) the augmented object can be rotated to complete other missing faces similarly.*

In case the trihedral assumption does not hold for a building, the user can resort to completing its geometry manually. In this case, although the user could "guess" the invisible edges while tracing an object, our initial experiments indicated that such guesses can affect the overall geometry reconstruction adversely. This is mainly because invisible edges also define vertices on the ground, and so guessing them can adversely affect the resulting ground geometry. Also, it is difficult to draw accurately in perspective and it may be easier to specify these edges from a different view.

The figures in the bottom row above provide an example of the L-shaped building in Figure 1(c). The object is rotated and completed progressively. The user sketches strokes for missing edges. When the program detects a new face formed by them, it reconstructs its 3D geometry. The user can then rotate the partially completed object and continue sketching.

It may be observed that this functionality of progressively constructing an object can be used independently as a design tool. In fact, the input image is used only as a guide for setting up the camera (Section 4), tracing out buildings (Section 5) and texturing the reconstructed buildings (Section 7). Thus, geometry can be progressively reconstructed from perspective sketches once a perspective camera is set up. This is an extension of the work done in [SC04, LS96] and can be used as a tool for reconstruction of sketches.

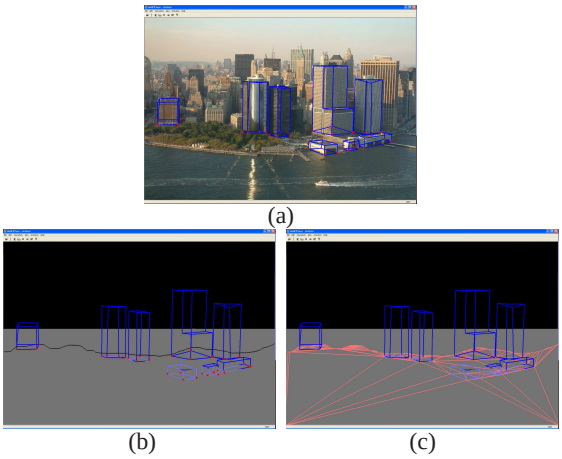


Figure 3: *Determination of ground and background geometry. (a) the image with reconstructed objects in wireframe; the vertices marked in red lie on the ground. (b) a least-squares plane is constructed from these vertices as shown in grey. As it is an approximate plane, it passes through some objects (shown in light blue). A horizon curve is sketched by the user. (c) points on the horizon line are projected on the plane and are used with the red vertices to get the ground geometry shown as an orange mesh. The background relief is raised from the horizon curve as shown in Figure 1(c).*

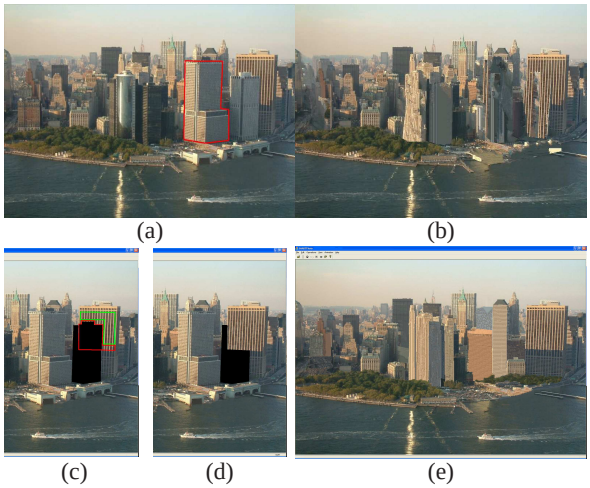


Figure 4: *Hole-filling by texture synthesis. Top row: automatic texture synthesis (a) when a building is sketched by the user, its outline is used as the region for synthesis. The bounding box of this region is scaled up and used as the source region for the texture synthesis. (b) Holes from Figure 1(d) filled using this method. Note that although this image looks patchy, the reconstructed buildings overlap these regions for most view points. Row 2: interactive texture synthesis (c) the source and target regions are specified in green and red respectively. (d) the synthesized output for (a). (e) Holes from Figure 1(d) filled using this interactive method.*

6. Ground and Background Geometry

Once all the buildings have been constructed, the ground geometry must be determined to complete the environment. All the edges of the buildings that should be on the ground are heuristically determined as discussed in Section 5. Figure 3(a) shows these vertices in red. Let the set of such vertices be denoted by *S*. A least-squares plane P (shown in grey in Figure 3(b)) is fit through points in *S*. It can be seen that *P* may pass through some objects.

The user then sketches out a pseudo-horizon curve (intersection of the ground and the background, see Figure 3(b)). This curve should be drawn just under the buildings in the background, so that the image does not "fold" in the middle of a building. The curve is then projected on *P* to obtain its 3D coordinates. These projected points are added to *S*, along with two points on the bottom corners of the screen. All points in *S* are triangulated in *P*. These form the ground relief (shown in orange in Figure 3(c)). Points of the pseudo-horizon curve are raised up to form the background, as seen in Figure 1(c). This completes the geometry of the 3D scene.

7. Image Completion

As a building is constructed, it has to be removed from the original image, creating a hole (Figure 1(d)). Such textures are usually filled manually by cloning (such as in [OCDD01]), a user-intensive and cumbersome process. Many image completion methods exist in computer graphics and vision literature. Many texture synthesis algorithms exist that complete a given "target region" of an image from source regions contained within the same or different images. Recent research has even led to real-time [LLX*01] and controllable [LH05] texture synthesis. However, texture synthesis primarily works well for amorphous images (flowers, stones, fuzzy background scenery, etc.) and its matching techniques tend to break down when synthesizing well-defined geometric structures like buildings. Image inpainting [BSCB00] works well for filling small holes in images, but it is not as effective on large missing regions. Although automatic, both texture synthesis and image inpainting are too slow for the size of images in this paper, mitigating most advantages of the automation.

From our initial experiments with cloning, we observed that buildings tend to be filled up using regions immediately surrounding them, not from distant image regions. Intuitively a building must be replaced by the background it occludes, which can be approximated from the image regions adjacent to the building. We use this observation to define the source region for a given hole. We calculate the bounding box of the hole to be filled, and scale it by 25%. We use the region in this scaled bounding box (minus the hole) as the source for the texture synthesis process. We use

the texture synthesis algorithm proposed in [EL99,LLX*01]. As the hole is automatically derived from the user's sketch, the synthesis is done with no extra input. Row 1 of Figure 4 shows how this method produces a synthesized image. Each hole was filled in less than 30 seconds.

Although the image in Row 1 contains very obvious patches and discontinuities, the reconstructed buildings occlude large parts of these patches from most view points, and thus they are not as disturbing. However, because of the limitations of texture synthesis discussed earlier, the results produced may sometimes be too disturbing. In these cases, the user can resort to an interactive process. We take inspiration from the paint-by-numbers interface [Hae90] by augmenting this process with a simple interface to interactively synthesize these holes (see Row 2 of Figure 4). The user marks the source and target regions in the image with poly-lines to start texture synthesis. As the user marks small regions at a time, this synthesis is fast.

8. Implementation and Results

All results in this paper were produced on a desktop system with a 2.6 GHz Pentium Xeon processor and 1 GB RAM, with an external tablet device (Wacom Cintiq PL-550).

Figure 5(c) shows a part of Manhattan, New York reconstructed using Figure 5(a)[‡‡] as input. Figure 5(b) shows how the constructed 3D environment looks from a distant view point. Eight buildings have been reconstructed in this example. The total time from (a) to (c) took about 15 minutes (including interaction), while the image completion took another 10 minutes. The actual optimization time is 2-3 seconds per building.

Figure 5(d) shows a drawing of Foshay Tower in Minneapolis, MN (USA)[§§] from the 1930s. Figure 5(f) shows an alternate view obtained by reconstruction (note the correctly reconstructed pyramidal top of the tower). Three buildings were reconstructed in this view. The total time taken for geometry reconstruction was 5 minutes (including interaction), while the actual optimization time is 2-3 seconds per object.

Figure 5(g) shows a drawing of the 1305 Church of Aston-Cantlow, Warwickshire, England[¶¶]. Figure 5(i) shows a zoom and change of angle towards the church. The image synthesis in this case was done by cloning, as the background is fairly homogeneous. Please see the accompanying video for a visual illustration of our pipeline and fly-throughs of these reconstructed scenes.

Figure 5(j) shows a photograph from Liebowitz *et al.* [LCZ99] (used with permission from the authors). This photograph is challenging because of some radial camera distortion and because it breaks the layered initial guess assumption of Shesh *et al.* [SC04] (the corner between the walls and ground is further away from the viewer than the vertices on the silhouette, instead of being nearer to it). The models in (k) and (l) were produced in two steps: first reconstructing the two walls and then reconstructing the roof and ground (1 second each).

9. Discussion and Future Work

Peek-in-the-Pic creates "navigable" 3D models from a single image by amalgamating camera calibration techniques and work done in the area of sketch reconstruction. Since it takes mostly "tracing" operations from the user and works on general photographs taken casually, it is suitable for non-technical users as well. *Peek-in-the-Pic* works best for modern architectural buildings that have polyhedral shapes. Reconstructing more involved architecture like buildings with ancient carvings is more difficult. However, as the final geometry is textured, artifacts are not noticeable even if a building with details is approximated by simpler geometry, unless one zooms in closely near a building.

In many ways, *Peek-in-the-Pic* can be seen as proof that by compromising on the accuracy of the acquired 3D model, one can create a system that works with a much simpler user interface that expects a lower degree of expertise from the user. Thus it lies in the middle of the spectrum of image-based modeling techniques, ranging from the completely automatic but often unsuccessful [HEH05] to the more accurate but also user-intensive [LCZ99], both in terms of the quality and extent of the user interaction and the quality of the resulting 3D models. Although working towards common goals, it is difficult to quantitatively compare the user interactions involved in such systems and *Peek-in-the-Pic* because the nature of user interactions tends to be very different.

As our method is based on inflation-based reconstruction, it shares some of its limitations. As the traced line drawing becomes more complex, the probability of convergence to a visually incorrect local minimum increases. In many cases this can be remedied by specifying constraints after reconstruction, but we acknowledge that this approach may not always succeed. The user may also successfully construct a building part-by-part (Figure 5(j-l)), as is common in case of inflation-based sketch reconstruction like SMART-PAPER [SC04]. However such an approach often follows an earlier failed attempt to construct the whole building at once and thus represents a limitation in the context of *Peek-in-the-Pic*. We believe an interesting extension would be to devise implicit gestures built into the tracing process itself, that may aid the reconstruction process (specifically in the initial guess estimation that affects the optimization the most). For example, users could provide hints about concavity/convexity of an edge of the building by tracing it us-

‡‡ Source:http://www.pilotlist.org/balades/manhattan/manhattan.html

§§ Source:http://www.minneapolishistory.com/marriott3.htm

¶¶ Source:http://www.holoweb.net/ liam/pictures/oldbooks/OldEngland/pages/1305-Church-of-Aston-Cantlow/

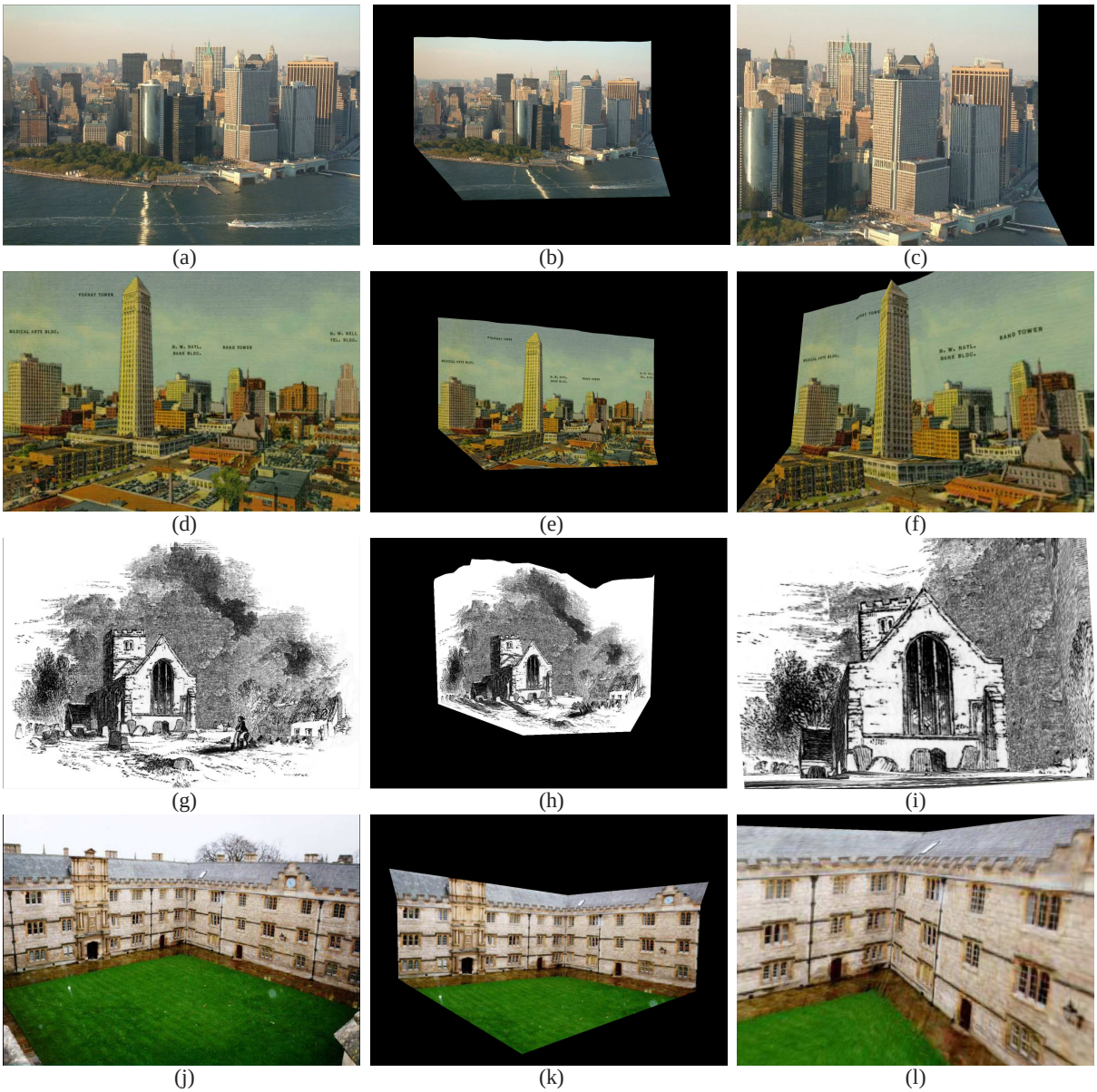


Figure 5: *Results. (a) photograph of Manhattan, NY (b) view from a distant point. (c) a unique view of the reconstructed 3D scene. (d) painting of 1930s downtown Minneapolis, MN. (e) view from a distant point. (f) a unique view of the reconstructed 3D scene. (g) painting of 1305 church (h) view from a distant point. (i) a unique view of the reconstructed 3D scene. (j) the original photograph of Merton College, Oxford, taken from [LCZ99]. (k) view from a distant point. (l) a unique close-up view of the 3D reconstructed scene.*

ing a different line style. As lines are being traced using a simple clicking interface, we believe such gestures would be more tolerable than having to sketch different line styles as in SMARTPAPER. Alternatively such hints may be built as post-process gestures. However a more formal user study would be needed to ascertain which gestures, if any, are acceptable and meaningful to the general user so as to not disturb the intuitiveness of the user interface that we have strived to maintain.

Our optimization framework currently uses no domain-based knowledge about the geometry that it attempts to reconstruct. We believe our approach can be combined with learning-based approaches [LS02, HEH05] to learn local

characteristics of typical buildings and their projections in images that will aid in the reconstruction process, and also to fix parameters like Θ. However some user interaction is unavoidable for any method to work for general cases.

Finally, although tracing out buildings is simple and easy, it is still the most time-consuming task in *Peek-in-the-Pic*. Automating this process by devising smart edge-detection methods fails because of strong edges produced by textures. Extending methods like *Lazy snapping* [LSTS04] to segment buildings automatically is worthy of further study.

References

[BSCB00] BERTALMIO M., SAPIRO G., CASELLES V., BALLESTER C.: Image inpainting. In *Proc. SIGGRAPH* (2000), pp. 417–424. 7

[BT81] BARROW H., TENNENBAUM J.: Interpreting line drawings as three-dimensional surfaces. *Artificial Intelligence 17* (1981), 75–116. 3

[Can86] CANNY J.: A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 8* (nov 1986), 679–698. 4

[CDR99] CIPOLLA R., DRUMMOND T., ROBERTSON D. P.: Camera calibration from vanishing points in image of architectural scenes. In *Proc. BMVC* (1999). 3, 4

[CFD02] CANTZLER H., FISHER R. B., DEVY M.: Improving architectural 3d reconstruction by plane and edge constraining. In *BMVC* (2002). 4

[CRZ00] CRIMINISI A., REID I. D., ZISSERMAN A.: Single view metrology. *International Journal of Computer Vision 40*, 2 (2000), 123–148. 3

[DTM96] DEBEVEC P. E., TAYLOR C. J., MALIK J.: Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. *Proc. SIGGRAPH* (1996), 11–20. 3

[EL99] EFROS A. A., LEUNG T. K.: Texture synthesis by non-parametric sampling. *IEEE Intl. Conf. Computer Vision* (September 1999), 1033–1038. 8

[EMJK94] E.LUTTON, MAITRE H., J.LOPEZ-KRAHE: Contribution to the determination of vanishing points using hough transform. *IEEE Trans. Pattern Analysis and Mach. Int. (PAMI) 16*, 4 (1994), 430–438. 4

[FB81] FISHER M., BOLLES R.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Comm. of the ACM 24* (1981), 381–395. 4

[Hae90] HAEBERLI P.: Paint by numbers:abstract image representations. *Proc. SIGGRAPH* (1990), 207–214. 8

[HEH05] HOIEM D., EFROS A. A., HEBERT M.: Automatic photo pop-up. *Proc. SIGGRAPH* (2005), 577–584. 1, 3, 8, 9

[HiAA97] HORRY Y., ICHI ANJYO K., ARAI K.: Tour into the picture: using a spidery mesh interface to make animation from a single image. *Proc. SIGGRAPH* (1997), 225–232. 1, 3

[HK87] HERMAN M., KANADE T.: The 3D MOSAIC scene understanding system: incremental reconstruction of 3d scenes for complex images. *Readings in computer vision: issues, problems, principles, and paradigms* (1987), 471–482. 3

[Huf71] HUFFMAN D.: Impossible objects as nonsense sentences. *Machine Intelligence 6* (1971), 295–303. 3

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: a sketching interface for 3d freeform design. *Proc. SIGGRAPH* (1999), 409–416. 3

[Kan81] KANADE T.: Recovery of the three-dimensional shape of an object from a single view. *Artificial Intelligence 17* (1981), 409–460. 3

[KC06] KAPLAN M., COHEN E.: Producing models from drawings of curved surfaces. In *Proc. SBIM* (2006), pp. 51–58. 3

[KH06] KARPENKO O. A., HUGHES J. F.: SmoothSketch: 3d free-form shapes from complex sketches. *ACM Transactions on Graphics 25*, 3 (2006), 589–598. 3

[LCZ99] LIEBOWITZ D., CRIMINISI A., ZISSERMAN A.: Creating architectural models from images. In *Proc. Eurographics* (1999), pp. 39–50. 1, 3, 8, 9

[LF92] LECLERC Y., FISCHLER M.: An optimization-based approach to the interpretation of single line drawings as 3d wire frames. *IJCV 9*, 2 (November 1992), 113–136. 3, 5

[LH05] LEFEBVRE S., HOPPE H.: Parallel controllable texture synthesis. *Proc. SIGGRAPH 24*, 3 (2005), 777–786. 7

[LLX*01] LIANG L., LIU C., XU Y.-Q., GUO B., SHUM H.-Y.: Real-time texture synthesis by patch-based sampling. *ACM Trans. Graphics. 20*, 3 (2001), 127–150. 7, 8

[LS96] LIPSON H., SHPITALNI M.: Optimization-based reconstruction of a 3d object from a single freehand line drawing. *J. Computer Aided Design 28*, 8 (1996), 651–663. 2, 3, 4, 5, 6

[LS02] LIPSON H., SHPITALNI M.: Correlation-based reconstruction of a 3d object from a single freehand sketch. *AAAI Spring Symposium on Sketch Understanding* (2002), 99–104. 9

[LSTS04] LI Y., SUN J., TANG C.-K., SHUM H.-Y.: Lazy snapping. *Proc. SIGGRAPH* (2004), 303–308. 10

[LTM96] LENGAGNE R., TAREL J., MONGA O.: From 2d images to 3d face geometry. pp. 301–306. 1

[MA84] M.J.MAGEE, AGGARWAL J.: Determining vanishing points from perspective images. *Proc. CVGIP 26*, 2 (1984), 256–267. 4

[Mar91] MARILL T.: Emulating the human interpretation of line drawings as three-dimensional objects. *IJCV 6*, 2 (1991), 147–161. 3

[MB95] MCMILLAN L., BISHOP G.: Plenoptic modeling:an image-based rendering system. *Proc. SIGGRAPH* (1995), 39–46. 1

[OCDD01] OH B. M., CHEN M., DORSEY J., DURAND F.: Image-based modeling and photo editing. *Proc. SIGGRAPH* (2001), 433–442. 3, 7

[PF06] PRASAD M., FITZGIBBON A.: Single view reconstruction of curved surfaces. In *Proc. CVPR* (2006), pp. 1345–1354. 3

[PMC03] PIQUER A., MARTIN R., COMPANY P.: Using skewed mirror symmetry for optimisation-based 3d line-drawing recognition. *Proc. 5th IAPR Intl. WS on Graphics Recognition* (2003), 182–193. 5

[PTVF02] PRESS W., TEUKOLSKY S., VETTERLING W., FLANNERY B.: *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press, New York, NY, 2002. 6

[RMMD04] RECHE-MARTINEZ A., MARTIN I., DRETTAKIS G.: Volumetric reconstruction and interactive rendering of trees from photographs. *Proc. SIGGRAPH* (2004), 720–727. 1

[Rob63] ROBERTS L.: *Machine Perception of 3-D Solids*. PhD thesis, MIT, 1963. 3

[Rot00] ROTHER C.: A new approach for vanishing point detection in architectural environments. In *BMVC* (2000), pp. 382–391. 4

[SC04] SHESH A., CHEN B.: SMARTPAPER–an interactive and user-friendly sketching system. *Computer Graphics Forum (Proc. Eurographics) 23*, 3 (2004), 301–310. 2, 3, 4, 5, 6, 8

[SC05] SHESH A., CHEN B.: Peek-in-the-pic: Architectural scene navigation from a single picture using line drawing cues. *Proc. Pacific Graphics (Short Paper)* (2005). 2

[SG00] SCHWEIKARDT E., GROSS M. D.: Digital clay: Deriving digital models from freehand sketches. *Automation in Construction 9* (2000), 107–115. 3

[SL97] SHPITALNI M., LIPSON H.: Classification of sketch strokes and corner detection using conic sections and adaptive clustering. *ASME J. Mechanical Design 119*, 2 (1997), 131–135. 4

[Sze94] SZELISKI R.: Image mosaicing for tele-reality applications. *WACV94* (1994), 44–53. 1

[Var03] VARLEY P.: *Automatic Creation of Boundary-Representation Models from Single Line Drawings*. PhD thesis, 2003. 3

[VM00] VARLEY P., MARTIN R.: Constructing boundary representation solid models from a two-dimensional sketch–topology of hidden parts. *Proc. First UK-Korea Wksp. Geometric Modeling and Computer Graphics* (2000), 129–144. 6

[WT91] WANG L.-L., TSAI W.-H.: Camera calibration by vanishing lines for 3-d computer vision. *IEEE Trans. Pattern Analysis and Machine Intelligence 13*, 4 (Apr. 1991), 370–376. 4

[ZHH96] ZELEZNIK R. C., HERNDON K. P., HUGHES J. F.: SKETCH: an interface for sketching 3d scenes. *Proc. SIGGRAPH* (1996), 163–170. 3