# A Human-Assisted System to Build 3-D Models From a Single Image

Alexandre R.J. François and Gérard G. Medioni

*University of Southern California, Los Angeles, CA 90089-0273*

*{afrancoi,medioni}@iris.usc.edu*

## Abstract

*We present a system at the junction between Computer Vision and Computer Graphics, to produce a 3-D model of an object as observed in a single image, with a minimum of high-level interaction from a user.*

*The input to our system is a single image. First, the user points, coarsely, at image features (edges) that are subsequently automatically and reproducibly extracted in real-time. The user then performs a high level labeling of the curves (e.g. limb edge, cross-section) and specifies relations between edges (e.g. symmetry, surface or part). NURBS are used as working representation of image edges. The objects described by the user specified, qualitative relationships are then reconstructed either as a set of connected parts modeled as Generalized Cylinders, or as a set of 3-D surfaces for 3-D bilateral symmetric objects. In both cases, the texture is also extracted from the image.*

## 1. Introduction

We present a system at the junction between Computer Vision and Computer Graphics, to produce a 3-D model of an object as observed in a *single* image, with a minimum of *high-level* interaction from a user.

A central issue in Multimedia applications is the mixing of real and virtual worlds. For example, augmented reality involves the incorporation of virtual objects into real images or video streams. Multimedia applications make extensive use of 3-D models, and require models of increasing quality. Today, 3-D modeling is a flourishing business, because building quality 3-D models is long and painful. Most 3-D models are created entirely manually, with the help of various available modeling tools. This requires art, creativity and time. An interesting easier alternative is modeling from images. Companies such as 3-D Construction Company [1] sell systems that allow the user to build a 3-D model from a set of images. The user specifies correspondences between the input images, used by the programs to infer the 3-D geometry of the scene. The texture is also extracted from the images. Tools for modeling can be seen as "intelligent" visual interface where the user performs most of the processing. We see a modeling environment as a Computer Vision system in which the processes are automatic, and the user is solicited to resolve process ambiguities. Interactivity is used in some Computer Vision systems, although it is often regarded as semi-failure. Interactive Vision systems have found much better appreciation in domains such as medical images analysis, where some interaction is a low price to pay to facilitate tedious image analysis tasks, such as boundary delineation. For example, user interaction is used for initializing contour extraction tasks that use snakes [9][14][15]. Another example of user-steered segmentation tool is Live-Wire [3], based on a graph representation of image gradients. User directed segmentation is also used in photogrammetry for building extraction from aerial images using modeling primitives [6].

In our system, the user provides limited but pertinent, high-level information to help feature extraction and grouping, so that robust Computer Vision algorithms can be used to perform the geometrical computations automatically. The general paradigm of our system is presented figure 1. The input to our system is a single image. First,
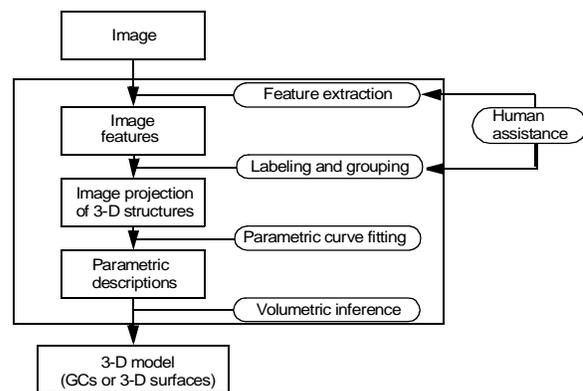


**Figure 1.  An interactive approach**

the user points, coarsely, at image features (edges) that are subsequently automatically extracted in real-time. Our interactive feature extractor is based on a recently introduced framework for feature saliency computation, called tensor voting [10][11]. We find this new framework offers unprecedented foundations for interactive (and possibly automatic) edge and junctions extraction. Both edge and junction information is handled in the same formalism and inferred from one single process. Unlike snakes, the voting process is not iterative, and unlike graph based

approaches, the saliency map is computed once and for all for the image.

The result of this interactive phase is a set of curves and junctions. The user then performs a high level labeling of the curves (e.g. limb edge, cross-section) and specifies relations between edges (e.g. symmetry, surface or part). The 2-D image features are now understandable as projections of 3-D structures. Reconstruction of polyhedral objects from perfect data is a mathematically well understood problem [17]. The reconstruction of 3-D object models from real images however presents two difficult problems: the *segmentation* of the object features from the background (also called figure-ground problem), and the *3-D inference* itself. Solving these problems requires assumptions about the geometry of the observed object. Several description models capturing interesting projective properties of objects (projective quasi-invariants) have been introduced to help resolve segmentation and 3-D inference. Even so, dealing with real data is computationally expensive and the resulting systems lack robustness. Furthermore, the models used are limited to specific classes of objects. Zerroug's system [18] for example performs automatic detection, segmentation and 3-D reconstruction of sub-classes of Generalized Cylinders (GCs). The most difficult problem to solve in this kind of application is the segmentation problem. Once the object features are identified, reconstruction is an underconstrained geometry problem, solved using the projective properties of the kind of objects considered. We present two reconstruction paradigms, based on two different object representation models. The first one handles subclasses of GCs, and is largely derived from Zerroug's work. The underlying complex object model is a connection graph in which the nodes are parts modeled as GCs, and the edges describe the relations between the parts. Most of the projective properties used in the automatic system are useful for reconstruction, but they were primarily introduced for detection (symmetry properties): these properties are not all necessary to perform the reconstruction only, which allows us to somewhat relax the constraints on the kinds of GCs we are able to reconstruct. This approach however is still limited to a small class of objects, so we introduce another reconstruction approach, based on a more general property: *bilateral symmetry in 3-D*. The 3-D object model in this case is surface-based.

Both reconstruction paradigms involve geometrical computations, such as symmetry computation, that are more efficiently carried with parametric representations of the edges. While simple B-Splines are widely used in that purpose in Computer Vision [13][16], we choose to use more general Non-Uniform Rational B-Splines (NURBS). NURBS are used in Computer Graphics for the flexibility they provide in modeling complex curves or surfaces.

We describe the details of the interactive feature extraction in section 2, then we present a NURBS fitting algorithm in section 3, finally we describe the geometry of the 3-D reconstruction in section 4. Relevant results obtained on real images are presented in each section. We conclude this paper by a summary of our contribution in section 5.

## 2. Interactive feature extraction

We present here our interactive edge and junctions extractor. In this section, we consider an edge to be a non-oriented curve stored as a linked list of pixels.

The user specifies a point in the neighborhood of the edge, through a simple point-and-click interface. The closest edge point is then found, and the corresponding edge is automatically grown from that point, producing a curve object (linked list of pixels) and the eventual junction objects linked to the extremities of that edge. This extraction process runs in *real-time* (it is faster than the rendering of the extracted features), and ensures reproducibility of the extraction phase, since user input is used for feature identification only. This edge extractor is built upon a saliency computation engine developed by Lee and Medioni [10][11] and uses the formalism of tensor voting. We give a brief description of the information obtained from this engine, and then describe the interactive edge finding and automatic edge extraction processes.

### 2.1. Saliency computation

Starting from an edge image obtained by a classical edge-detector, the tensor voting mechanism produces a dense saliency tensor field [10][11]. In effect, we get, at each pixel of the image, a saliency tensor, which is a symmetric second order tensor that can be interpreted as a covariance matrix, characterized by three floating point values: the two eigenvalues $\lambda_1$ and $\lambda_2$ ($\lambda_1 \geq \lambda_2$) and the angle $\theta$ between the main axis and the image x-axis. This tensor can also be thought of as an ellipse, whose shape and size encode the tangent direction of a curve going through that pixel, its saliency and its uncertainty. The value $\lambda_1 - \lambda_2$ at each pixel constitutes a curve saliency map (see figure 2), in which salient curves are crest lines. The value of $\theta$ *at edge pixels* is the tangent direction of the curve passing through that pixel. We also use the value $(\lambda_1 \lambda_2)/(\lambda_1 + \lambda_2)^2$ to build a junction map, in which junctions (characterized by a strong relative value of $\lambda_2$ over $\lambda_1$) are local maxima.

These maps, together with the tangent information, are used to interactively extract edges and their eventual terminal junctions. It is important to note that the saliency maps are computed once and for all for a given image.
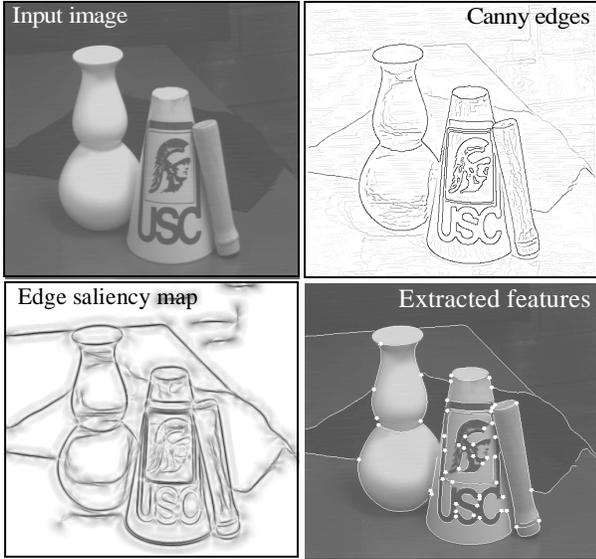
**Figure 2. Salient features extraction**

## 2.2. Edge finding

Once the user has indicated the edge to be extracted by specifying a point in the neighborhood of this edge, the first step is to find the closest edge point from the specified location. According to the saliency tensor theory, such a point is a local maximum of $\lambda_1$. The closest edge point is searched for by a simple gradient ascent on $\lambda_1$ starting from the specified location. Since the saliency gradient is significant only in the neighborhood of an edge, specifying a starting point far from any salient edge would yield undefined result. The edge point obtained is used as a seed for the automatic extraction of the complete edge.

## 2.3. Edge extraction

Starting from one seed edge point, the edge is "grown" on both sides using a simplified marching square algorithm. Considering the way feature information is represented in the saliency tensor, the edge corresponds to crest lines in the saliency map defined by $\lambda_1 - \lambda_2$. Since we also have the tangent direction at each pixel, extracting the crest is equivalent to extracting the zero-crossings of the function $\overrightarrow{\Delta\lambda} \cdot \vec{n}$, where $\vec{n}$ is the normal direction and $\overrightarrow{\Delta\lambda}$ is the gradient of $\lambda_1 - \lambda_2$. We developed an algorithm similar to a local isocontour marching process [12] to perform the zero-crossing extraction at pixel level.

Starting at the seed pixel, we compute the value taken by the dot product at each neighboring pixel, and test for a junction (high value in the junction map). If a junction is encountered, the edge-growing process stops. A zero-crossing occurs between two adjacent pixels whose values

are of opposite sign. When such a configuration occurs, we pick the pixel whose value has the smallest absolute value, since it is the closest to the actual zero-crossing. By examining each pair of adjacent pixels, we build a list of candidate pixels for the next pixel in the curve. The candidate yielding the smoothest curve is selected as the next pixel in the curve. This is repeated from this last pixel until either no candidate can be found to extend the curve, or a junction is encountered. The edge is grown separately in two directions from the seed, corresponding to the two possible orientations of the tangent direction. The resulting two curve segments are then merged to give one linked list of pixels. Edges and junctions extracted interactively with our system are presented in figure 2.

## 3. NURBS fitting

The interactive edge extraction procedure described in the previous section produces linked lists of pixels. A parametric representation of edges is necessary to perform efficiently the geometrical computations involved in the volumetric inference algorithms. B-Splines have been shown to be an efficient way to represent and manipulate curves in computer vision [13][16]. Among other advantages, they provide a compact, continuous and smooth representation of the curves, and thus allow to interpolate curve points and compute tangents at any location along the curve. We choose to use a more general type of B-Splines than the one described in the aforementioned papers. Our choice of working with NURBS, at the price of a more complicated fitting algorithm, is motivated by their wide use in Computer Graphics, and by projective properties of these curves [5]. We now describe our algorithm to fit a NURBS to a set of adjacent points (curve).

### 3.1. Problem statement and notations

We consider the NURBS $P(u)$ of the form

$$P(u) = \sum_{k=0}^{N-1} B_{k,d}(u)p_k, \quad u_{d-1} \le u \le u_N \quad , 2 \le d \le N$$

Such a NURBS is defined by its order $d$, a set of $N$ control points $p_k$, $0 \le k < N$, and a knot vector $\begin{bmatrix} u_0 \ u_1 \ \cdots \ u_{N+d-1} \end{bmatrix}$. The fusion functions $B_{k,d}(u)$ are polynomials of degree $d-1$, defined by the Cox-deBoor recursion formulas [7]. We conducted our experiments with NURBS of order 3. The algorithm developed assumes a fixed, arbitrary order. However, since the degree of the blending polynomials increases with the order, the stability of the fitting process is expected to drop as the order becomes higher. Fitting a NURBS to a set of points comes down to three major steps:

- Estimate the minimum number of control points necessary to ensure a good approximation.
- Create a knot vector and parameterize the curve.
- Compute the position of the control points.

**3.1.1. Estimation of the number of control points.** The optimal number of control points is the minimum number of control points needed to get an approximation of the curve with a given maximum error. A good estimation of that number is the number of vertices of a polygonal approximation of the curve, obtained with a given maximum error. An insufficient number of control points will yield a poor approximation, whereas too large a number will deteriorate the compactness of the representation, and may cause stability problems in the fitting.

**3.1.2. Knot values and parameterization of the curve.**
The simplest way to fix the knot values is to define a uniform knot vector, eventually normalized but this has no effect on the remainder of the process. In the case of an open curve, the first and last points of the curve are used as control points. In order to force the NURBS through these points, the first and last $d$ knot values are set equal.

The points of the curve are considered as a sampling of the NURBS we are building. We must therefore associate to each point of the curve a value of the parameter $u$. One of the curve's extremities is chosen as the origin. We compute the parameter value for each curve point proportionally to its distance to the origin *along the curve*. Thus, since the knot vector is uniform (except maybe at the extremities), the speed defined by the parameterization along the curve decreases with the curvature. As a result, in a uniform sampling, the density of points will be higher in areas of large curvature, yielding a more precise approximation.

**3.1.3. Position of the control points.** Each point $c_i$ of the curve, of parameter $s_i$, ideally corresponds to the point $P(s_i)$ of the NURBS. Thus for each point, we have:

$$P(s_i) = \sum_{k=0}^{N-1} B_{k,d}(s_i)p_k = c_i$$

This set of equations obtained for all the points of the curve can be written as:

$$\left[ B_{0,d}(s_i) \dots B_{N-1,d}(s_i) \right] \left[ p_0^T \dots p_{N-1}^T \right]^T = c_i^T$$

This is a linear system whose unknowns are the coordinates of the control points. In general the number of curve points is much larger than the number of control points. In our experiments, with the order fixed to 3, a least square solution to this system gives a good NURBS approximation of the curve, as shown in figure 3 in the case of an open and a closed curve.
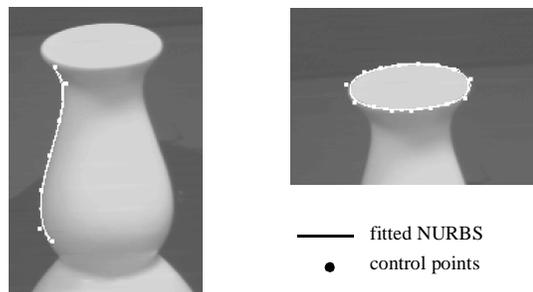


fitted NURBS
• control points

**Figure 3. NURBS fitting**

# 4. Volumetric inference

We present two 3-D reconstruction algorithms. The first one is derived from work on automatic segmentation and 3-D reconstruction of subclasses of Generalized Cylinders (GCs) [18]. The result of the reconstruction is a segmented volumetric model of the objects where parts are GCs. We developed the second method in the context of interactive segmentation, making use of bilateral symmetry in 3-D, in order to handle a less constrained class of objects. The objects is described in terms of 3-D surfaces. In both cases, the texture is extracted from the image. In the remainder of this paper, we assume orthographic projection. Given 2-D structures in the image plane *(x,y)*, our goal is to recover their depth and orientation with respect to the image plane, *i.e.* "inferring the $z$ coordinate".

## 4.1. GC Reconstruction

GCs are defined by an axis (3-D curve), a cross-section (3-D curve) and a sweep function. The volume thus defined is obtained by sweeping the cross-section along the axis according to the sweep function.

In previous works, the 3-D inference problem has been addressed for constrained cases, e.g. for such subclasses of GCs as Straight Homogenous GCs and Planar Right Constant GCs. The interesting projective properties of these kinds of GCs were used to automatically detect, segment and infer parts presenting these properties. In our case, detection and segmentation are done by the user. We do not need to use all these properties to infer the 3-D of a GC part, which allows us to relax the constraints imposed.

We implemented the reconstruction for GCs with the following set of constraints:
- Planar axis
- Constant cross-section
- Cross-sections perpendicular to the axis (*a.k.a.* orthonality constraint)
- Two symmetric points on limb edges belong to the same cross-section

We also assume that the axis of symmetry in the image is the projection of the axis of symmetry in 3-D, which is a classical approximation.

These constraints are dictated by the geometry of the reconstruction, and will be justified in section 4.1.2. As an important part of the reconstruction process is symmetry computation between boundaries, we first describe our algorithm for symmetry computation.

**4.1.1. Symmetry computation.** Efficient ways to compute symmetries between B-Splines segments have been described in [16]. One of the main advantages of manipulating Spline representations of curves is the availability of the tangent at any point on the curve. Different kinds of symmetries have been used in computer vision. The skew symmetry and the parallel symmetry are two widely used examples. The strong geometrical constraints in the definitions of these symmetries have been very useful for automatic segmentation. Here, since segmentation is solved by the user during the feature extraction procedure, we can use the more general definition of Smooth Local Symmetry (SLS) [2] to compute symmetries between two edges.
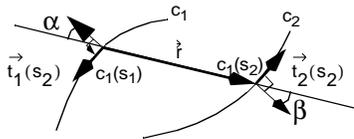
**Figure 4.  Smooth Local Symmetry definition**

With the notations defined in figure 4, the two points $c_1(s_1)$ and $c_2(s_2)$ are symmetrical if $\alpha = \beta$, i.e.
$$\vec{r}(c_1(s_1), c_2(s_2)) \cdot (\vec{t_2}(s_2), \vec{t_1}(s_1)) = 0 .$$

Our algorithm for symmetry computation is as follows: For each sampled point on each curve, we compute the set of points on the other curve with which it forms a symmetry as defined above. This computation reduces to a binary search of zero crossings of the function defined by the left-hand side of the previous equation, as described in [16]. Since SLS is a local property, we obtain a set of symmetry lines, some of which are incompatible. The set of symmetry lines centers defining the longest, smoothest valid axis is then extracted using a dynamic programming-style algorithm. Results are presented in figure 5.

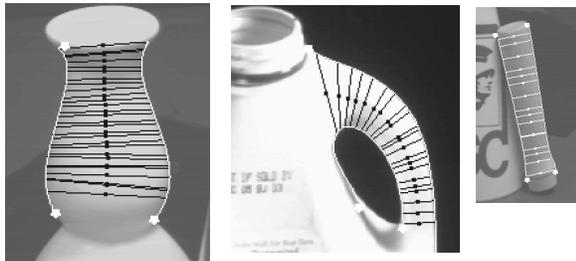**4.1.2. Volumetric inference.** The projection of a GC in the image produces three curves: two limb edges, and one cross-section. We suppose the user has labeled and grouped extracted edges accordingly.

We start by computing the symmetry lines between the two limb edges. The symmetry centers are considered to be the projection of the actual axis points. Since two symmetric points on limb edges belong to the same cross-section, and the cross-section is constant, a flat wireframe model of the GC is obtained by reporting the cross-section at each symmetry center, scaling and orienting the curve according to the symmetry line length and direction.

By fitting an ellipse to the cross-section, we can estimate its orientation in 3-D. The orientation is strictly correct if the cross-section is circular. If not, the recovered orientation is an acceptable approximation of the real orientation. We implemented Fittzgibbon's algorithm for direct least square fitting of ellipses [4]. The axis is assumed planar, and the cross-section perpendicular to the axis. Thus given the orientation of the cross-section in 3-D, we can compute the relative z coordinate of each axis point (symmetry centers). A NURBS can then be fitted to these 3-D points, to obtain an approximation of the 3-D axis of the GC. The 3-D of each cross-section is then computed by enforcing the orthogonality constraint.

The result is an intrinsic 3-D model of the GC part: cross-section, 3-D axis and sweep function. The 3-D information is up to an offset along the z axis (in orthographic projection, the distance of the object to the image plane is not recoverable).

We show, in figure 6, the projected inferred GC cross-sections superimposed on the part in the image, and a 3-D view of the reconstructed volume, with texture mapped, floating above the image plane. The user can then describe relations between different reconstructed parts to produce segmented 3-D models of objects.

### 4.2. Bilateral Symmetric Objects Reconstruction

Having relaxed the constraints as much as possible on the GCs we could reconstruct, we find that another reconstructing approach has to be taken to be able to handle more general types of objects. A central notion in 3-D inference is symmetry. Kanade observed that the ability to
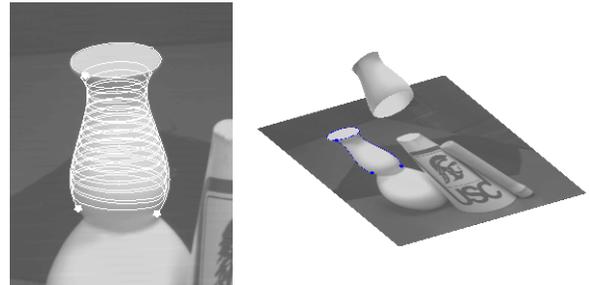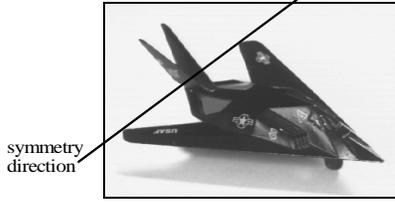
**Figure 5.  Symmetry axes and symmetry lines**

**Figure 6.  GC volumetric reconstruction**

**Figure 7. A bilateral symmetric object.**



**Figure 8. Computation of the z coordinate of two symmetrical points**

infer the 3-D of a volume observed in a picture relies on the assumption that "a skew symmetry depicts a real symmetry viewed from some unknown angle" [8]. A property that numerous objects share is indeed bilateral symmetry *in 3-D*. The GCs we considered so far are simply special cases of bilateral symmetric objects. Two *true edges* that are bilateral symmetric in 3-D project into two image edges that define a skew symmetry (assuming non accidental viewpoint). Given the projections of two perpendicular directions in an image (like the symmetry axis and the symmetry direction in a skew symmetry), the 3-D orientation of the plane defined by these two directions is not recoverable. Thus the use of 3-D bilateral symmetry is not well suited to automatic systems and has been mostly discarded as an underconstrained problem in Computer Vision. We show that it can be very useful in an interactive approach. We suppose that the skew symmetry direction is known, either given by the user or computed from the extracted edges.

**4.2.1. Inference of the z coordinate for a pair of symmetrical points.** A bilateral symmetry is defined by a symmetry plane. The symmetry direction in the image is the projection of the normal direction to the symmetry plane on the image plane. The symmetry axis direction is the direction of the intersection of the symmetry plane with the image plane. We know that given the symmetry direction and the symmetry axis direction, there is one extra degree of freedom to determine the 3-D orientation of the symmetry plane. Let us assume this direction is 45 degrees, and that the symmetry plane is at a fixed, arbitrary depth with respect to the image plane (see figure 8).

Given the projections of two points symmetrical with respect to the symmetry plane, a very simple geometric construct allows to compute their z coordinate. First, we observe that the line joining the two points in 3-D has to be normal to the symmetry plane. The points have to be on their respective orthographic projection lines, and at equal distance from the symmetry plane. Choosing an orientation of 45 degrees for the symmetry plane yields the following construct: if point O is the intersection of the symmetry plane with the symmetry line joining the projections of the two points, then the z coordinate of each point is the distance between O and the projection of the other point.
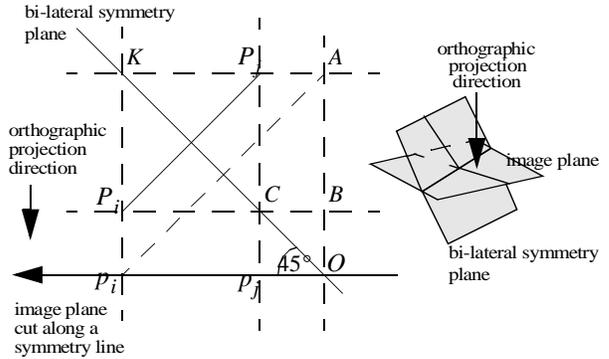
All pairs of points that are symmetrical with respect to the same symmetry plane, and whose 3-D position with respect to that plane is computed as described above, will be in correct relative position with respect to each other.

**4.2.2. 3-D edges computation.** Given the symmetry direction, and the knowledge that two edges are symmetrical, we first have to establish point to point correspondence between the two edges. For a given point on one of the edges, we consider the line passing through this point and parallel to the symmetry direction. We then compute the intersection of this line with the symmetric edge. This can be done by looking for zero crossings of the function defined by $s \times l$ where $s$ is the symmetry direction and $l$ is the vector joining the current point to a point in the symmetric edge. If there is no intersection, the $z$ coordinate of the point cannot be computed. If there are more than one intersections, tangent information or curve smoothness can be used to resolve the ambiguity.

The method described in the previous section can then be used to compute the $z$ coordinate of each pair of symmetric points in the edges.

**4.2.3. 3-D surface model inference.** At this stage, 3-D has been computed for most points of symmetric edges. We assume the user has grouped edges belonging to the same surface. A plane is fitted to the points of the edges belonging to a same surface, using only the points for which the z coordinate could be computed. Our current implementation deals only with planar surfaces, but the algorithm can be extended to more complex surfaces. Once the surface has been estimated, the $z$ coordinate of the edge points that could not be computed previously is fixed by forcing these points onto the surface. These points had no visible symmetric points in the image, but they belong to symmetric edges, and therefore should have symmetric points that were occluded in the image. These points are computed and added to the corresponding edges. The result is a set of complete 3-D edges. The sur-

**Figure 9. Three-D surface model.**

face information is then recomputed to fit the new boundaries, and the corresponding texture is extracted from the image. Textured models build with our system are presented figure 9.

The only parameters arbitrarily fixed in the 3-D inference algorithm are the depth and the orientation of the symmetry plane with respect to the image plane. The former is a free parameter that fixes the relative depth of the scene with respect to the image plane, unrecoverable under orthographic projection. The latter can be linked to an aspect ratio of the model reconstructed. Intuitively, varying the orientation of the symmetry plane changes the direction and the length of the line joining two recovered symmetrical points. Only the length is an intrinsic parameter of the model. Therefore the angle that the symmetry plane makes with the image plane is directly linked to an aspect ratio of the model along and across the symmetry plane. This aspect ratio can be easily fixed by the user after the model has been extracted, either by direct manipulation or by specifying a metric.

## 5. Conclusion

We have presented a human assisted system to reconstruct 3-D objects as observed in a single image.

We have developed an interactive edge and junction detector based on saliency maps obtained by tensor voting. This detector requires the user to point to an area of interest used to initialize the edge extraction process. Saliency information is computed once and for all for a given image, and subsequent interactive feature extraction takes place in real-time. The feature extraction process, while requiring user input, ensures consistency and reproducibility of the resulting data. The user also specifies high level relationships between selected image features, which allow the system to interpret image features as projections of 3-D structures. Image features are fitted with parametric representations useful in reconstruction algorithms. We have developed an algorithm for fitting a NURBS to a curve. We have presented an algorithm for inferring volumetric information for some constrained GCs. We have also developed an algorithm to produce 3-D surface models of bilateral symmetric objects. The texture extracted from the image allows to produce realistic models, that can be used in multimedia applications.

## 6. References

[1] 3-D Construction Company, www.3dconstruction.com

[2] Brady M. and Asada H., "Smooth Local Symmetries and Their Implementation". In *International Journal of Robotics Research*, vol. 3, no. 3, pp. 36-61, 1984.

[3] Falcao A.X. *et al.*, "User-Steered Image Segmentation Paradigms: Live-Wire and Live-Lane". Tech. Report MIPG213, Dept. of Radiology, University of Pennsylvania, June 1995.

[4] Fitzgibbon A.W., Pilu M. and Fisher R.B., "Direct Least Squares Fitting of Ellipses". In *Proc. ICPR'96* (A76.2).

[5] François A. and Medioni G., "NURBS Descriptions of Image Features and Regions". IRIS Technical Report, University of Southern California, in preparation.

[6] Gülch, E., "Application of Semi-Automatic Building Acquisition". In *Proc. Ascona Wkshop 1997 "Automatic Extraction of Man-Made Objects from Aerial and Space Images"*, May 1997.

[7] Hearn D. and Baker M.P., *Computer Graphics*, 2nd ed., Prentice Hall, Englewood Cliffs, NJ, 1994.

[8] Kanade T., "Recovery of the Three-Dimensional Shape of an Object from a Single View". In *Artificial Intelligence*, vol. 17, 1981, pp. 409-460.

[9] Kass M., Witkin A.P. and Terzopoulos D., "Snakes: Active Contour Models". In *International Journal of Computer Vision*, vol. 1, no. 4, January 1988, pp. 321-331.

[10] Lee M.S. and Medioni G., "Inferred Descriptions in Terms of Curves, Regions and Junctions from Sparse, Noisy Binary Data". In *Proc. International Workshop on Visual Form*, Capri, Italy, May 1997, pp. 350-367.

[11] Lee M.S. and Medioni G., "Grouping ., -, ->, O-, into Regions, Curves and Junctions". In *Proc. IEEE Wkshop on Perceptual Organization*, Santa Barbara, CA, Jun. 1998.

[12] Lorensen W.E. and Cline H.E., "Marching Cubes: A High Resolution 3-D Surface Reconstruction Algorithm". In *Computer Graphics*, vol.21, no.4, 1987.

[13] Menet S., Saint-Marc P. and Medioni G., "B-Snakes: Implementation and Application to Stereo". *DARPA90*(720-726).

[14] Neuenschwander W., Fua P., Szekely G. and Kubler O., "Initializing Snakes". In *Proc. CVPR'94*, pp. 658-663.

[15] Neuenschwander W., Fua P., Iverson L., Szekely G. and Kubler O., "Ziplock Snakes". In *International Journal of Computer Vision*, vol. 25, no.3, December 1997, pp.191-201.

[16] Saint-Marc P., Rom H. and Medioni G., "B-Spline Contour Representation and Symmetry Detection". In *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 15, no. 11, November 1993, pp. 1191-1197.

[17] Sugihara K., *Machine Interpretation of Line Drawings*. MIT Press, Cambridge, MA, 1986.

[18] Zerroug M. and Nevatia R., "Three-Dimensional Descriptions Based on the Analysis of the Invariant and Quasi-Invariant Properties of Some Curved-Axis Generalized Cylinders". In *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 18, no. 3, March 1996, pp. 237-253.