Reconstruction of Relief Objects From Archeological Line Drawings

MICHAEL KOLOMENKIN and GEORGE LEIFMAN, Technion
ILAN SHIMSHONI, University of Haifa
AYELLET TAL, Technion

This article addresses the problem of automatic reconstruction of a 3D relief object from a line drawing. Our main application is reconstruction of archaeological artifacts based on line drawings. The problem is challenging due to five reasons: the small number of orthogonal views of the object, the sparsity of the strokes, their ambiguity, their large number, and their interrelations. We partition the reconstruction problem into two subproblems. First, we reconstruct the underlying smooth base of the object from the silhouette. Assuming that the variation of bases belonging to the same class of objects is relatively small, we create the base by modifying a similar base retrieved from a database. Second, we reconstruct the relief on top of the base. Our approach can reconstruct the relief from a complex drawing that consists of many interrelated strokes. Rather than viewing the interdependencies as a problem, we show how they can be exploited to automatically generate a good initial interpretation of the line drawing. Even though our algorithm is generic, its strength is demonstrated by the reconstruction of artifacts from manual drawings taken from real archaeological reports. These drawings are highly challenging, since artists created very complex and detailed descriptions of artifacts regardless of any considerations concerning their future use for shape reconstruction.

Categories and Subject Descriptors: J.2.10 [**Physical Sciences and Engineering**]: —*Archaeology*; I.2.10 [**Artificial Intelligence**]: Vision and Scene Understanding—*Modeling and recovery of physical attributes*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Physically based modeling*

General Terms: Algorithms

Additional Key Words and Phrases: 3D reconstruction, line drawing

ACM Reference Format:

Kolomenkin, M., Leifman, G., Shimshoni, I., and Tal, A. 2013. Reconstruction of relief objects from archeological line drawings. ACM J. Comput. Cult. Herit. 6, 1, Article 3 (March 2013), 19 pages.
DOI: http://dx.doi.org/10.1145/2442080,2442083

1. INTRODUCTION

Line drawings have been the standard method of archeological artifact documentation for many years. While many findings might have been lost or destroyed, their illustrations remain. Therefore, the

This research was supported in part by the Israel Science Foundation (ISF) 1420/12, Ollendorff Foundation, the P. and E. Nathan Research Fund, and the Argentinian Research Fund.

Authors' addresses: M. Kolomenkin, G. Leifman (corresponding author), and A. Tal, Department of Electrical Engineering, Technion, Haifa, Israel; email: gleifman@techunix.technion.ac.il; I. Shimshoni, Department of Information Systems, University of Haifa, Haifa, Israel.

DOI: http://dx.doi.org/10.1145/2442080,2442083

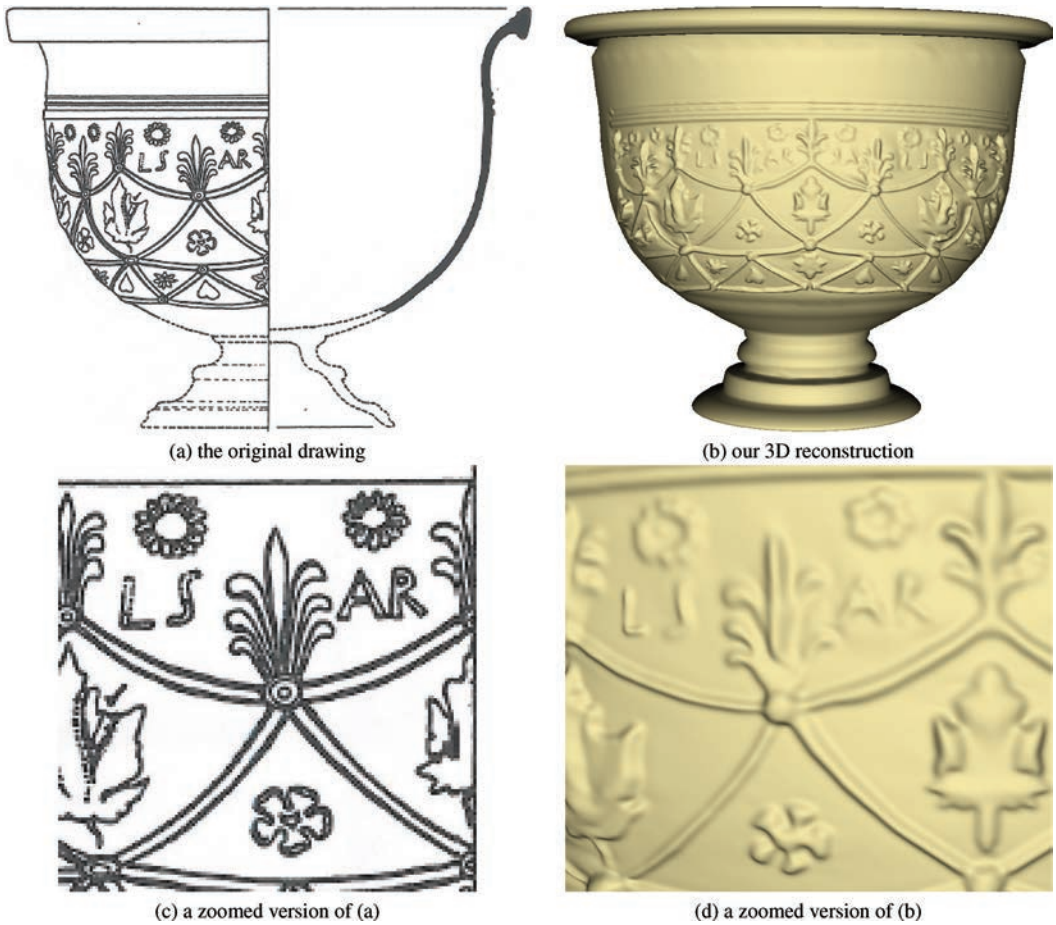


Fig. 1. Reconstruction from a manual drawing consisting of 571 curves. North Italian Sigillata. A cup by L. Sarius (name appears on the cup) 10 B.C–30 A.D [Brusic 1999], catalog number 273.

only existing inputs for reconstruction are the line drawings appearing in an archaeological report (Figure 1(a)). Usually, production of such drawings is derived from the analysis of an expert archaeologist. The archaeologist translates into a drawing his interpretation of the object. So our goal is not to recreate a digital replica of the object itself, but to recreate a missing object as it was perceived by the scientist who drew the archeological report. These inputs are usually highly complex. State-of-the-art algorithms for automatic reconstruction were not designed with such drawings in mind.

Line drawings are used to represent different kinds of findings, such as buildings, statues, and others. In this work, we focus on artifacts containing reliefs. They are important sources of information, since they are fingerprints of periods and cultures. In this article, we applied our algorithm to real examples from archeological reports of two types: Hellenistic and Roman relief pottery [Brusic 1999] and Cosa lamps [Fitch and Goldman 1994].

Automatic reconstruction of 3D relief objects from a line drawing can be divided into two subproblems—reconstruction of the base and reconstruction of the details (the relief) on top of the base. The main challenge of automatic base reconstruction is the fact that we are given only one or two views of the silhouette, which is insufficient. Given the base, automatic relief reconstruction from

a line drawing is a challenging task due to several reasons. First, the lines are usually sparse and thus, the object is not fully constrained by the input. Second, the line drawings are often ambiguous, since the lines may have different geometric meanings—they can indicate 3D discontinuities, surface creases, or 3D step edges. Third, the input may consist of a large number of strokes that need to be efficiently handled by the algorithm. Fourth, these strokes are interrelated. For instance, in the relief of Figure 1, the decorations are either protruded or indented as a whole, and a solution in which some of the lines indicate protrusions and others indentations is less likely.

Related work. Reconstruction of a 3D object from a line drawing is a fundamental problem in computer vision; see Cooper [2008] for a survey. Approaches to reconstruction from a line drawing typically consist of two steps: line labeling and the reconstruction itself. Line labeling focuses on finding a set of consistent labels given a set of lines [Clowes 1971; Huffman 1977; Liu et al. 2007a; Shimshoni and Ponce 1997; Waltz 1975]. A line is assumed to indicate depth or orientation discontinuity of an object. A label states whether the line represents a concave or convex edge, or an occlusion.

Reconstruction algorithms build a 3D object from the labeled lines. Usually, the algorithms assume that the object consists of planar faces [Malik 1987; Sugihara 1982]. They find a set of consistently oriented faces that generate a feasible object. Recent algorithms are also able to handle drawings composed of arcs and not only straight lines [Chen et al. 2008; Wang et al. 2009].

These algorithms are less suitable for relief objects because of two reasons. First, the algorithms handle only specific types of lines—lines representing depth or orientation discontinuities. Relief objects, however, usually do not have such discontinuities. Instead, they are described by general lines representing 3D step edges. Second, the algorithms are designed to reconstruct CAD-like objects and do not effectively handle highly curved objects.

A related problem was addressed in computer graphics, denoted by *sketch-based modeling*. Most of the work modeled general objects [Igarashi et al. 1999; Karpenko and Hughes 2006; Nealen et al. 2007; and Gingold et al. 2009]. The goal is to generate the smoothest-possible object constrained by the given line. The techniques provide intuitive interfaces and generate visually pleasing results. However, the underlying smoothness assumption results in smooth, blob-like objects, which cannot accurately convey the details of reliefs.

Techniques that focus on relief editing can produce accurate surfaces of various types [Gingold and Zorin 2008; Kerautret et al. 2007; Wu et al. 2007; Joshi and Carr 2008]. However, it might be difficult to employ them on drawings consisting of a large number of curves. First, they require the user to enter manually the parameters of each curve. This is a tedious and time consuming process. Second, they assume that the user input is consistent. This assumption makes the editing of multiple interrelated curves very challenging.

Our approach. We propose an approach that is able to reconstruct a relief from a complex drawing that consists of many interrelated strokes, such as the one in Figure 1. The algorithm manages to compute good results automatically, by setting, for each curve, its interpretation, that is, its shape parameters. Yet, the user is allowed to fine tune the interpretation of the drawing, if required.

The approach is derived from the observation that a relief object can be represented as a composition of a smooth base surface and a height function (relief) defined over that base [Kolomenkin et al. 2009; Liu et al. 2007b]. Therefore, our reconstruction task consists of two subtasks: base estimation and relief reconstruction.

For the base estimation, we assume that the variation of bases that belong to the same class of objects is relatively small and the base can be constructed as a modification of one of the bases in a database. Given the drawing, a database of 3D models is searched for models having similar silhouettes. The most similar model is deformed so that its silhouettes best match the drawings. The deformation is

performed by solving a linear optimization problem in which the silhouettes provide the boundary conditions.

As for the second subtask—relief reconstruction—we assume that the base is given. The algorithm is based on two key ideas. First, the interdependencies between the strokes of the line drawing can be exploited to automatically generate a good initial interpretation of the line drawing. Second, given an interpretation, it is possible to reconstruct a consistent surface.

For each idea, we provide a novel algorithm that solves the corresponding problem. To interpret the detailed line drawing, we show that our problem can be represented as a problem of topological ordering of a graph and solved efficiently. Given the base and the line drawing interpretation, the surface is reconstructed by posing it as a pair of linear optimization problems, which can be readily solved.

The contribution of this article is three-fold. First, the major contribution is an efficient algorithm for reconstruction of reliefs from line drawings. The method is able to handle highly complex real drawings (Sections 2–4). Second, we introduce an algorithm for automatic estimation of a base from the outline of the given line drawing (Section 5). Both algorithms are realized in an interactive system (Section 6). Last but not least, the method makes a significant step towards solving an important problem in archaeology (Section 7)—a domain that has recently attracted a lot of attention in computer vision and computer graphics [Brown et al. 2008; Koller et al. 2006; Kolomenkin et al. 2009; Rushmeier 2005].

A preliminary version of this aritcle was presented at the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) [Kolomenkin et al. 2011a].

2. RELIEF RECONSTRUCTION—GENERAL APPROACH

Given a line drawing of a relief object, our goal is to reconstruct the surface as shown in Figure 1. The base is estimated from the line drawing, as will be explained in Section 5. The algorithm then reconstructs the relief on the base regardless of its shape. This is the focus of the current section. In the following, we present the problem definition and outline our algorithm.

Problem definition. For most relief objects, the details can be described as a *height function* defined on a surface termed *the base*.

The lines of drawings of relief objects typically indicate changes of the height function. This height function is usually smooth far from the lines and constant along them. Its gradient is strongest near the lines in the direction perpendicular to the lines. Hence, the value of the height function in the line's neighborhood depends only on the distance from the line.

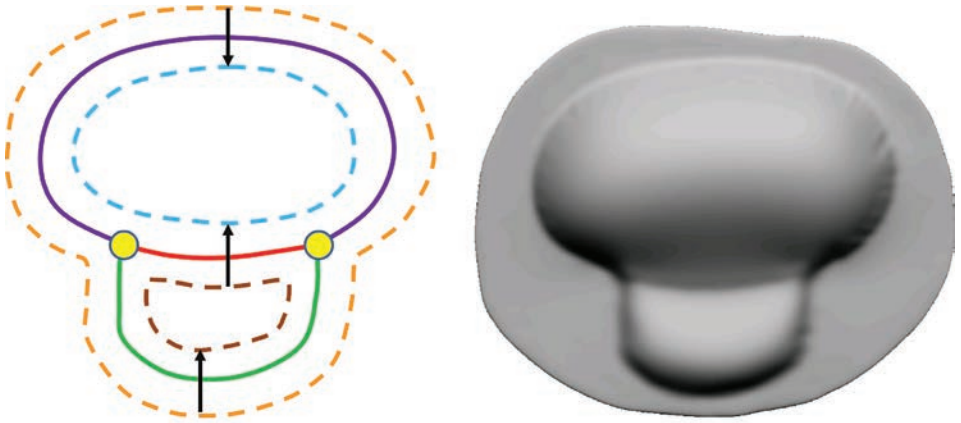
A drawing consists of *drawing curves*, *junctions*, and *margins* (Figure 2(a)). We define the *drawing curves*, or simply the *curves*, as the lines of the given drawing. They indicate visually-meaningful locations on the relief. The curves may be connected by *junctions*, but cannot cross them. *Margins* are the borders of a curve's neighborhood.

A drawing defines *the relief* (the height function). Outside the margins, the relief is assumed to be smooth. Inside the margins the relief is approximated by a step edge of height h and width w (Figure 3). Height h can be positive or negative, representing the direction of the edge. The shape of the relief defined by step edge $s(x)$ is $p(u)$:

$$p(u) = h \cdot s(u/w), \tag{1}$$

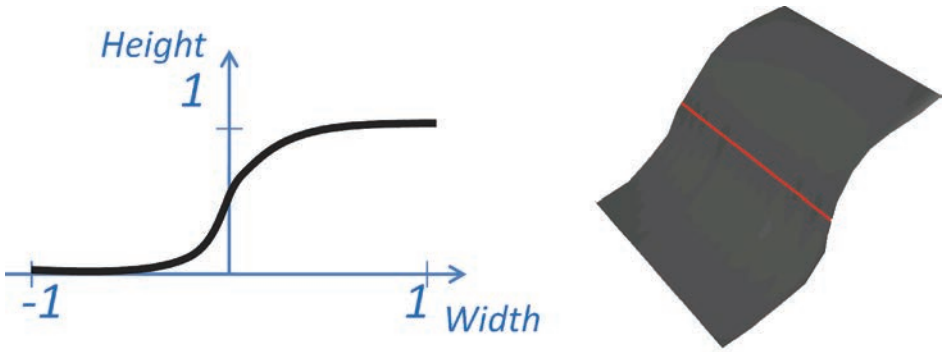
where $u \in [-w, w]$ is the width parameter of the step edge.

The sought-after surface is a combination of the base B and the relief. If we look at a cross section perpendicular to the curve, $B(u)$ is defined at a point on B at a signed distance u from the curve. Thus,



(a) Drawing data types (b) The corresponding relief

Fig. 2. Notations. Curves are solid lines, junctions are yellow circles, margins are dashed lines, and step edge directions are arrows. Each curve and margin is drawn in a different color.



(a) (b)

Fig. 3. A step edge approximation. (a) A cross section of a normalized step edge. (b) A 3D view of a normalized step edge.

the surface S within the curve's margins on the cross section can be written as

$$S(u) = p(u) + B(u) = h \cdot s(u/w) + B(u). \tag{2}$$

See Figure 4 for an illustration.

We can now rephrase our goal. Given a line drawing, we want to compute the relief object, such that near the curves, the shape of the cross section will match the 3D step edge p, whereas elsewhere its shape will be smooth and similar to the base B. Specifically, we need to compute the height h for the step edge of every curve in a consistent manner.

We assume that the margin width w and the step edge's shape $s(x)$ are constant for all the curves. In our system the margin width is set to 0.05% of the diagonal of the object's bounding box. The step edge shape is the shape of an ideal step edge smoothed with a Gaussian of standard deviation equal to the average edge length. Both w and the shape of the edge $s(x)$ can be later modified by the user.

Algorithm overview. In a preprocessing step, the curves are extracted from the image of the drawing, using the ridge detection algorithm of Steger [1998]. The algorithm first smoothes the image by a

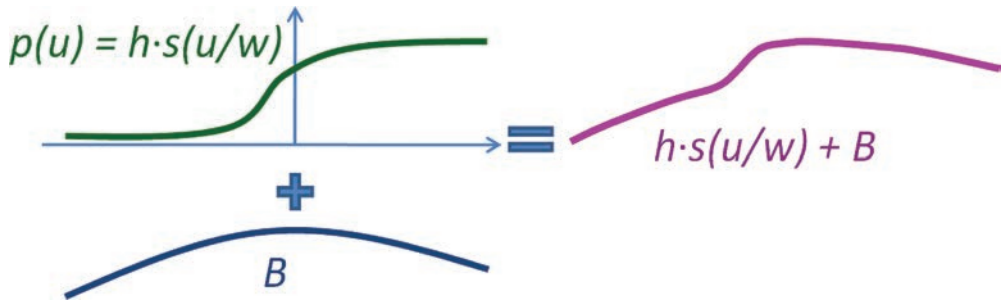


Fig. 4. The relief in a curve's neighborhood is a combination of the step edge $p(u)$ and the base B.

Gaussian filter of standard deviation equal to 2.0 pixels. Then it marks as potential ridges all pixels that satisfy the following criteria.

—Low gradient. The derivative of the image should be close to zero on the ridge.

—The first eigenvalue of the Hessian matrix is high and the second eigenvalue of the Hessian matrix is low. The second derivative of the image should be high in the direction perpendicular to the ridge and low along the ridge.

—The pixel intensity is low. We are searching for dark curves only.

After all the ridges have been marked, the algorithm applies nonminimal suppression to make the ridges thinner. Finally it links all ridge pixels and detects junctions. Small nonconnected ridges are removed. Then, junctions are detected and margins are generated. These elements serve as the input to our algorithm. Since the input was drawn manually, we assume it does not contain noise and that all the curves represent real features. We only remove short lines that are often used for shading effects.

Initially, an automatic interpretation of the line drawing is computed, by calculating consistent height values for the step edges (Section 3). Given the curves and the step edges, the surface is reconstructed (Section 4).

3. LINE DRAWING INTERPRETATION

Interpreting the line drawing requires setting the height of the step edge of each curve. Doing this manually for a complex drawing composed of dozens, or even hundreds of curves is a cumbersome process. Moreover, the set of assigned heights has to be consistent. Consistency refers to the requirement that two different paths between points should result in the same height difference (Figure 5). The solution to this problem should address the challenges of interpretation consistency combined with the large number of strokes.

The key idea of our method for computing the heights of the step edges is to reduce this problem to the problem of a constrained topological ordering of a graph. A similar reduction is proposed in Sỳkora et al. [2010] for adding depth to cartoons. In the following, we first describe the reduction and then the algorithm for solving the corresponding graph problem.

Reduction to a graph problem. Given a drawing (Figure 6(a)), we initially represent it by an undirected graph $G = \{V, E\}$ as follows (Figure 6(b)). The margins are the nodes of the graph. There exists an edge between two nodes wherever a curve lies between the corresponding margins.

Our goal is to direct the graph and to find for each directed edge, a positive integer weight. The weight corresponds to the height difference between the source and the target nodes of the edge. A directed edge indicates that the margin representing the source node is lower than the margin

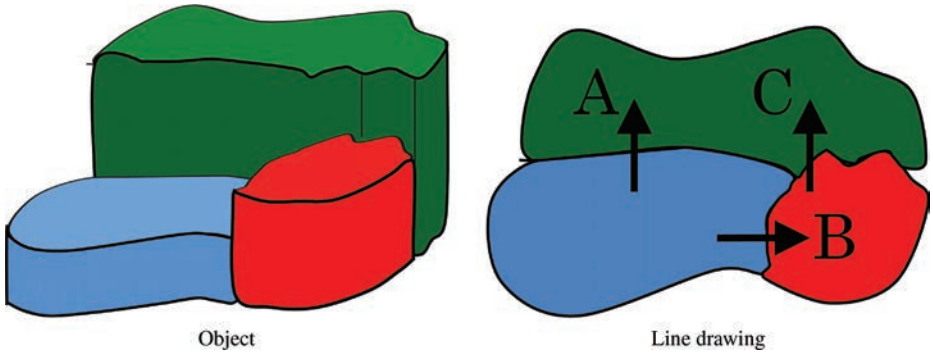


Fig. 5. Height consistency. The height of step edge **A** should be equal to the sum of the heights of step edges **B** and **C**.

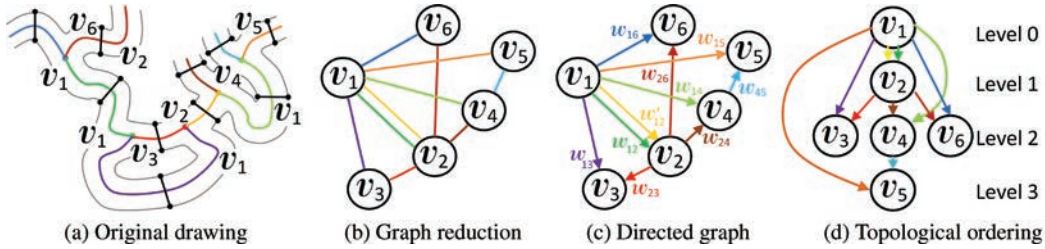


Fig. 6. Graph representation. (a) A drawing in which each curve is colored differently. The margins are denoted by v_i. The black segments represent the step edges. (b) The corresponding graph. Each step edge is colored consistently with the curve it crosses. (c) Given the undirected graph, our algorithm directs it and finds its weights w. (d) The topological levels of the graph. The weight of an edge is equal to the difference between the levels of its nodes.

representing the target node. The weights should be consistent, i.e., all the paths between two nodes should have the same weight. Let us denote the height of node $v \in V$ by h_v and the weight of the edge e_{km} from node k to m by w_{km}. We aim at generating a weighted directed graph that satisfies the following constraint.

(1) Positive integer weight: $w_{km} > 0$, $w_{km} \in \mathbf{Z}$.
(2) Height consistency: $h_k + w_{km} = h_m$.

Note that due to transitivity, Constraint 2 guarantees that all the paths between two nodes have equal weights.

If the user provides additional information, such as specific directions or weights, they should also be satisfied. The additional constraints are indicated as follows.

(3) Respect the given directions of some edges e_{km}.
(4) Respect the given weights w_{km} of some edges e_{km}.

Constraint 3 explicitly specifies the direction of some edges of the graph, that is, which of the two nodes has a lower height. This constraint is used to make the reliefs consistently higher or lower than the base.

Our algorithm employs the following heuristic to determine this constraint automatically. Nodes representing the base margins are identified as nodes that are adjacent to many edges. Hence, we set the directions such that they point away from these nodes. As a result, the reliefs are consistently higher than the base surface.

Constraint 4 sets the weight of some edges of the graph. It can be employed to make certain parts higher or lower than those computed by the automatic algorithm. This is not mandatory and is omitted by default.

We first, explain how the graph is directed and then how the edge weights are computed.

(1) *Directing the graph.* Given an undirected graph (Figure 6(b)), we aim at generating a directed graph that will allow us to later assign consistent weights according to the second constraint (Figure 6(c)). This will specify the direction of every step edge, that is, specify its lower and its higher ends.

We observe that a basic requirement of the sought-after solution is that the resulting directed graph should be acyclic (DAG). This is so since as all the weights are positive, a cycle in the graph would be of a positive weight w_{cycle}, contradicting Constraint 2 that for a node v on the cycle $h_v + w_{cycle} = h_v$.

Any undirected graph may be made into a DAG by choosing a total linear order for its vertices and orienting every edge from the earlier endpoint in the order, to the later endpoint. Our algorithm chooses an order that is consistent with Constraint 3. The direction of unconstrained edges is chosen so as to produce a DAG.

(2) *Assigning weights to the edges.* Given a DAG, the goal is to compute positive weights of the edges, such that Constraints 1–3 hold. The weights correspond to the heights of the step edges.

Initially, the graph is partitioned into topological levels, similarly to the way its done in topological sorting (Figure 6(d)). In this partition, level 0 includes only nodes for which there are no incoming edges and recursively, level i includes only nodes for which the incoming edges come from nodes at level $i - 1$ or smaller, and at least one of them comes from level $i - 1$.

We assign the weights of the nodes at level i to i and assign the edge weights w_{km} to the difference between the weight of node k and that of node m. This setting satisfies the constraints.

4. FROM RELIEF INTERPRETATION TO RECONSTRUCTION

Given a base, a set of curves, and the heights of their step edges that were computed in Section 3, the goal is to reconstruct the relief surface. The output of the algorithm is a height function defined for every point on the base, yielding the resulting surface.

The algorithm should address two of the challenges specified in the introduction, namely the sparsity of the input curves and the interactions between close curves. The sparsity challenge is approached by producing a smooth interpolation in regions in which curves do not exist. The interaction challenge requires that the reconstructed surface will not contain undesired artifacts due to interactions between close curves.

To achieve these goals, we require that the reconstructed height function satisfy two constraints. Locally, the relief in the curve's neighborhood is defined up to a constant using only the step edges of the margins. The constant is important since a relief can be defined either on the base or on another relief, yet the height function is measured relative to the base. Globally, the relief should be the smoothest function that coincides with the relief obtained locally for every curve. During the reconstruction, the algorithm should compute for each curve its constant. Hereafter, we describe our algorithm for realizing these two requirements.

(1) *Computing the relief locally.* Given a curve and its step edge, the goal is to compute its relief locally, independently of other curves. Let $\mathbf{p} = (\nu, \upsilon)$ be a point in the curve's neighborhood (defined by the step edge's width), where υ is the arc-length parameter along the curve of its closest point on the curve and ν is the signed distance from it to $\mathbf{p}$ (0 for a point on the curve). The relief $r(\mathbf{p}) = r(\nu, \upsilon)$ at point $\mathbf{p}$ is set to

$$r(\mathbf{p}) = p(\nu, \upsilon) + C_{\mathbf{p}}, \tag{3}$$

where $p(\mathbf{p})$ is the corresponding step edge and $C_{\mathbf{p}}$ is the height of the step edge with respect to the base.

(2) *Computing the relief model.* Given a base and the height function $r(\mathbf{p})$ computed locally for each curve, our goal is to compute the global height function (relief) R on the whole surface. This function should coincide with r in the neighborhoods of the curves (boundary conditions) and be smooth elsewhere.

The key idea is to reconstruct the surface in two linear steps. First, the smoothest Laplacian of the desired function R is estimated, such that it satisfies the boundary conditions. Then, it is used to calculate R. The linearity of these stages allows us to deal with complex line drawings efficiently.

Computing the Laplacian of the Relief. To formulate the smoothness requirement, we demand that the Laplacian of the Laplacian of the relief is zero. Intuitively, it is roughly equivalent to the requirement that the mean curvature of the surface changes linearly. Note that if we required only a zero Laplacian, the reconstructed surface would be planar in between the curves, which is undesirable. Consider, for example, Figure 2. The nonplanar reconstruction of the surface in between the curves in Figure 2(b) is the result of this requirement.

Let us denote by Δ the Laplacian operator on a scalar function. We are seeking the Laplacian L of R, which is a scalar function defined on the base surface. L should be equal to the Laplacian of r in the neighborhoods of the curves and its Laplacian $\Delta(L)$ should be zero elsewhere. Formally, L is the solution of the following system of linear equations.

We assume that the surface is represented by a triangulated mesh. Let $\mathbf{p}$ be a vertex of the mesh. We search for L that satisfies

$$\begin{aligned} L(\mathbf{p}) &= \Delta r(\mathbf{p}), \quad \mathbf{p} \in \text{curve's neighborhood}, \\ \Delta L(\mathbf{p}) &= 0, \quad \text{elsewhere.} \end{aligned} \tag{4}$$

Assume that $N(\mathbf{p})$ is the set of the neighbors of $\mathbf{p}$, A is the area of the Voronoi cell of $\mathbf{p}$, and γ_j and δ_j are the angles opposite the edge $[\mathbf{p}, \mathbf{p}_j]$ of the triangles adjacent to this edge.

The Laplacian Δ of a scalar function f (either r or L) at point $\mathbf{p}$ on a mesh is calculated as [Meyer et al. 2002]

$$\Delta f(\mathbf{p}) = \frac{1}{2A} \sum_{j \in N(\mathbf{p})} (\cot(\gamma_j) + \cot(\delta_j))(f(\mathbf{p}) - f(\mathbf{p}_j)), \tag{5}$$

where $N(\mathbf{p})$ is the set of neighbors of $\mathbf{p}$ on the mesh, A is the Voronoi area of $\mathbf{p}$, and γ_j and δ_j are angles opposite to $\mathbf{pp}_j$. See Figure 7 for clarification of the notations. Obviously, Equation (5) is linear in the values of f. Hence, any linear equation solver can be utilized to calculate f.

To solve the system of Equations (4), we need to know the values of r. Recall however, that by Equation (3), r is known only up to constants $C_{\mathbf{p}}$. The trick used here is that we can compute the Laplacian of r without knowing them, since these constants add a linear component to the function and the Laplacian is independent of the linear components.

Computing the Relief. Given the Laplacian of the relief, we calculate the relief (height function) R on the whole base. In the neighborhoods of the curves, R should coincide with r, whereas elsewhere the Laplacian ΔR should be equal to the computed Laplacian L. Hence, R is the solution of the following system of linear equations.

$$\begin{aligned} R(\mathbf{p}) &= r(\mathbf{p}), \quad \mathbf{p} \in \text{A curve's neighborhood}, \\ \Delta R(\mathbf{p}) &= L(\mathbf{p}), \quad \text{elsewhere.} \end{aligned} \tag{6}$$

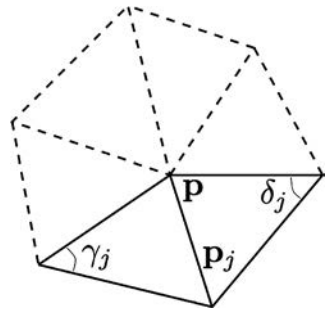


Fig. 7. Notations of Equation (5). The Laplacian Δ of a scalar function f at point $\mathbf{p}$ on a mesh is a linear combination of the values of f on the neighbors $\mathbf{p}_j$ of $\mathbf{p}$.

In this system of equations, the unknowns are the values of R at every vertex $\mathbf{p}$ and the constants $C_{\mathbf{p}}$. This system of equations is similar to that of Equation (4) and thus it is solved in a similar fashion. A similar approach is used for mesh fairing in Schneider and Kobbelt [2001] and Nealen et al. [2007].

Dealing with curve interactions. The problem of curve interaction manifests itself when a point belongs to several curve neighborhoods. Such a point will therefore appear in several equations in the systems of Equations (4) and (6). Each such equation is weighted according to the relative distance of the point from the corresponding curve. There are several ways to express the requirement that the surface near the curve coincides with the profile. It was determined empirically that the method we use avoids creating spurious artifacts on the object in cases of curve interactions.

5. BASE ESTIMATION

This section addresses the generation of the base model, which will then be given as an input to the reconstruction algorithm. One way to generate the base is to use a professional modeling tool or a sketch-based tool, such as Teddy [Igarashi et al. 1999] or FiberMesh [Nealen et al. 2007]. While these systems produce pretty results, they require some expertise, they are interactive, and they can produce relatively simple models.

We, however, would like to estimate the base from the drawing itself. We propose to find a similar base in a database of models, which enables us to reconstruct objects having highly complex bases when the database contains similar objects. However, since the database is not guaranteed to contain a model that accurately fits the line drawing's outline, the retrieved base has to be modified accordingly. Therefore, our algorithm consists of two stages. First, given the drawings, it finds the most similar model in the database. Then, the model is deformed so as to obtain a base whose orthographic projections are very close to the drawings, while preserving the shape of the matched model. Figure 8 summarizes the steps of our base estimation algorithm. Given a drawing (Figure 8(a)) and a database (Figure 8(b)), the most similar model is retrieved (Figure 8(c)). The result of the deformation is shown in Figure 8(d).

Reconstruction of a 3D object based on two or three orthographic views, as given in the archeological report, is an ambiguous task. For most artifacts this input is insufficient to produce an accurate base for a 3D model. We believe that our approach of deforming the most similar model from the database resolves the ambiguity in most of the cases when the variability of the base models is not very large.

(1) *Database search.* We use a database of nearly 2000 objects, which consists of three parts:

—1850 models from different classes, for more details see [Leifman et al. 2005];

—11 scanned models of archaeological artifacts from the Technion's CG&M Lab repository;

—35 models of lamps and vases created using various modeling tools.

Fig. 8. Base estimation algorithm. Given a line drawing (a) and a database (b), our algorithm first retrieves the most similar model from the database (c) and then deforms it to better suit the line drawing (d).

In the future we plan to enlarge the database of scanned models of archaeological artifacts.

During preprocessing, the models in the database are normalized to align along the principal directions [Elad et al. 2001]. For each orthographic view (three views) of each object in the database, the silhouette is extracted and sampled. Then a feature vector is calculated as follows. The silhouette is enclosed within a predefined shape (e.g., a circle, a rectangle, etc.) and the feature vector includes the distances from the points on the silhouette to the closest points on the predefined shape. Essentially, this vector defines a deformation between 2D shapes—the silhouette and the predefined shape. In our implementation we always use a circle as a predefined shape.

Given a drawing, the boundary of the drawing (the silhouette of the drawn object) is extracted. The drawing is manually divided into separate images, where each image contains a single view. Silhouette extraction of each image is performed using the active contour model [Kass et al. 1988]. A feature vector of the boundary is computed as described in the preceding. It is compared to those stored in the database using the $L2$ distance. If the drawing contains multiple views, the similarity scores of the views are averaged. The most similar model is returned.

Other methods were proposed for solving this and used for modeling [Lee and Funkhouser 2008; Shin and Igarashi 2007; Yang et al. 2005]. We found that our matching criterion minimizes the required deformation that has to be applied in the subsequent stage.

(2) *Model deformation.* It is seldom the case that the database contains a base model that perfectly fits the drawing. Therefore, we deform the retrieved model to better match the drawing. The key idea of our algorithm is to move the points on the orthographic projections to their matched points on the drawing and move the other points of the model so as to preserve its shape.

The input to our deformation algorithm is a 3D model and 2D drawings and the output is a deformed 3D model. Let $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_n, \mathbf{v}_i \in \mathbb{R}^3\}$ be the set of the vertices of the input model.

The algorithm, which is illustrated in Figure 9, consists of three steps. First, we use orthographic projection to find the model's silhouettes corresponding to the directions of the silhouettes in the drawing. Each silhouette of the model consists of edges and vertices. For a given silhouette, we denote the set of the silhouette vertices by $V_s \subset V$.

Second, we compute for each vertex $v_i \in V_s$, its desired location in 3D, denoted by V_s'. We find for each vertex in V_s, the closest point on the drawing's boundary. Then, the coordinates of each vertex in V_s' are defined such that their projection equals the matched vertex in the drawing, while maintaining the original depth of the vertex. This step is performed for the points of all the silhouettes.

Third, we formulate our problem as an optimization problem whose solution yields the deformed model $\overline{V}$. We require that in this model, every vertex in V_s moves close to its corresponding vertex in V_s', while preserving the shape as much as possible. This is done by minimizing the difference between the Laplacian coordinates of the given model and those of the deformed model.

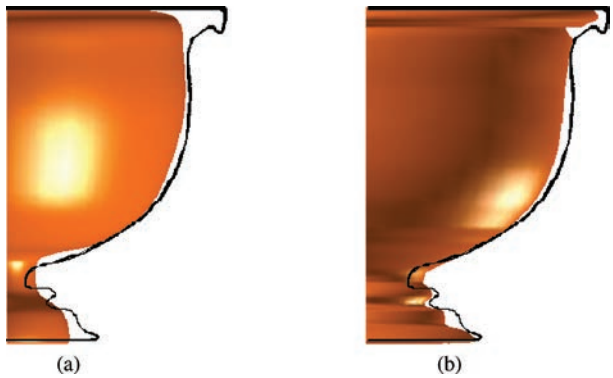


Fig. 9. Model deformation. (a) The desired locations V_s' (the black contour) are computed for all the vertices in V_s (the orange outline). (b) The vertices of V_s move closer to their corresponding vertices in V_s' (the black contour), while preserving the shape, resulting in $\overline{V}$ (the orange outline).

Specifically, we define the Laplacian coordinates $L(v_i)$ as

$$L(v_i) = v_i - \frac{1}{|N_i|} \sum_{j \in N_i} v_j, \tag{7}$$

where N_i is the set of v_i's neighbors. As indicated in Sorkine et al. [2004], the Laplacian coordinates are an intrinsic representation of a surface, and thus their preservation leads to preservation of the local surface structure. Moreover, they are a linear function of the object coordinates, which allows efficient computation within the optimization process.

Combining our two requirements—similarity to the line drawing and shape preservation—into a single optimization framework yields:

$$\overline{V} = \arg\min \left(\sum_{v_i' \in V_s'} (\overline{v}_i - v_i')^2 + \alpha \sum_{v_j \in V} (L(\overline{v}_j) - L(v_j))^2 \right). \tag{8}$$

The parameter α controls the importance of preserving the shape with respect to the importance of matching the drawing boundaries. In our experiments, $\alpha = 1$ yielded the best results. This optimization problem is solved using a sparse least-squares solver (we use sparse Cholesky factorization). Note that due to the shape preservation requirement, not all the vertices of V_s move to their corresponding locations, V_s', as illustrated in Figure 9(b).

Our base estimation algorithm is very robust and produces reasonable models even when the object found in the database is highly dissimilar from the required result. Figure 10(a) demonstrates this fact, by starting the deformation from a cylinder and yielding a goblet for the drawing in Figure 8(a). Naturally, the more similar the retrieved object is, the better the quality of the produced base as shown in Figure 10(b).

6. SYSTEM

We built an interactive system that realizes our algorithm. Figure 11 portrays the pipeline of the system. Initially, the algorithm estimates the underlying base from the drawing, as described in Section 5. Next, given the drawing and the base, the relief is reconstructed (Sections 2–4). Finally, the user may fine-tune the reconstructed surface.

Fig. 10. Comparison of the resulting deformed object, when the retrieved database model is a cylinder (a) or a more suitable one (b), which is shown in Figure 8(c).

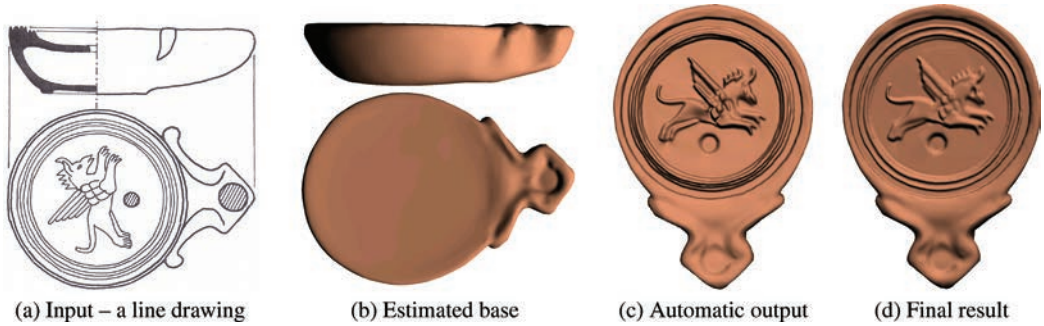


Fig. 11. Algorithm stages. Given a line drawing (a), our algorithm first estimates the base (b), and then reconstructs the relief (c). The user can then modify the profiles or add new ones, in order to fine-tune the result (d). Here the user changed the relative heights of the wing's feathers and the shape of the armor on the back of the Pegasus. He also switched the direction of the curve of the eye, transforming it from a protrusion to an indentation. (Roman triangular nozzle lamp of a griffin, 25/20 B.C. [Fitch and Goldman 1994] catalog number 378).

User interaction. The automatically reconstructed surface is usually satisfactory. However, in some cases, it is desirable to give the user the ability to fine-tune the resulting surface. It may happen when the user has information regarding the geometry of the surface that our algorithm cannot deduce from the drawing, or when our assumption that the surface can be accurately portrayed by a step edge in the line's neighborhood, is not suitable. Therefore, we provide the user with three interaction options, which are easily executed with a single mouse click.

(1) A curve's cross section can be depicted with any free-form profile instead of a step edge. Figure 12 portrays several possible profiles.
(2) The weight and the direction of the step edge (or any other profile) can be set by the user (Constraints 3 and 4 of the interpretation algorithm in Section 3).
(3) The user can set an arbitrary number of profiles per curve, enabling the shape of the surface to change along the curve.

Implementation and running times. Our algorithm consists of three parts: the base estimation, the line drawing interpretation, and the relief reconstruction. The heaviest one, is the relief reconstruction, which was therefore implemented on the GPU. Hereafter, we discuss the implementation and the running times for each part on a 2.4GHz Intel Core 2 Duo processor with 2GB of memory.

The first two parts of the algorithm were implemented in C++. During the generation of the base surface, searching a database that contains almost 2000 models takes 0.3 seconds. The deformation

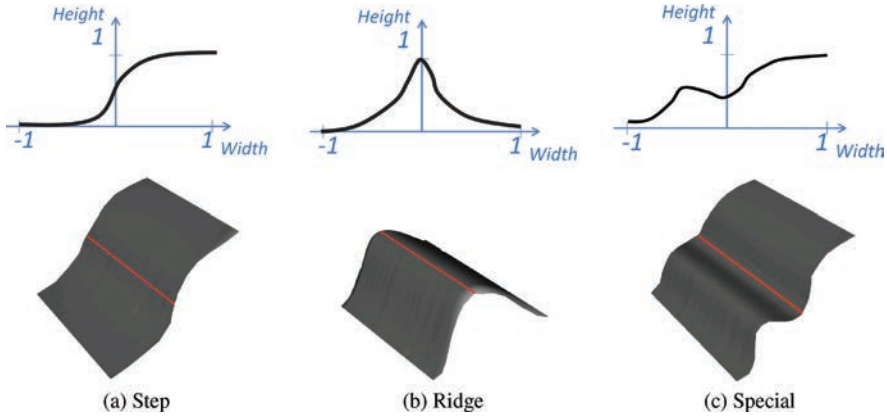


Fig. 12. Examples of profile shapes. Top: profiles; bottom: corresponding 3D reliefs with the line-drawing curves in red.

Fig. 13. Reconstruction of a Hellenistic relief large light-brown krater (Found in V. Mrdakovica-Croatia, [Brusic 1999] catalog number 1.) The drawing consists of 1300 tightly interconnected lines.

takes less than a second for a 10K-face model and 9 seconds for a 100K model. The running time of the line drawing interpretation is negligible, since it has a linear asymptotic complexity.

Relief reconstruction, however, may provide a challenge for large models, since the large and sparse linear systems in Equations (4) and (6) need to be solved by iterative optimization methods. To overcome this problem, we took two measures. First, we employed the results of the line drawing interpretation to initialize the optimization procedure. Second, we implemented the Biconjugate Gradient optimization algorithm on the GPU using the cusp library [CUDA]. The running times of the algorithm are in the range of 15 seconds for an 160K-face model (Figure 18) to 200 seconds for a model with almost 2M faces (Figure 13).

Table I summarizes the running times.

Table I. Time Required to Preprocess and Edit each Model

Fig.	Num. of curves	Preprocessing	Automatic	Manual
1	571	5 min	2 min	
11	56	5 min	1 min	1.5 min
13	1300	15 min	3.5 min	
14	39	5 min	30 sec	
15	41	5 min	32 sec	
16	69	5 min	1 min	
17	46	5 min	40 sec	2 min
18	65	5 min	45 sec	2 min

The time for manual editing is shown only where manual editing was applied.

Fig. 14. Reconstruction of a Roman triangular nozzle lamp of Eros, A.D. 50-ca 100. [Fitch and Goldman 1994] catalog number 504.

7. RESULTS

We will now present several additional examples of reconstruction of reliefs of archaeological artifacts, demonstrating the ability of our algorithm to deal with complex line drawings. All the archaeological drawings appearing in this section were taken from Brusic [1999] and Fitch and Goldman [1994].

Figures 1 and 13 show the automatic reconstruction of intricate reliefs of a cup and a krater. Though these drawings consist of 571 & 1300 tightly interconnected lines, the reconstruction achieves visually pleasing results. The previously described speedup procedure enables the reconstruction of these objects in a reasonable time.

Figure 1(c) zooms in on the fine details of the reconstructed relief, showing for example, that the reliefs are indeed of different heights. Note that manual reconstruction techniques, such as Wu et al. [2007], would require the user to provide parameters for each of the 571 or 1300 lines one after the other, which would be extremely labor intensive.

Figure 14 demonstrates the reconstruction of a Roman oil lamp. The drawing contain 39 lines. It can be seen that our automatic reconstruction is quite good. Figure 15 demonstrates the automatic reconstruction of the base surface and the relief from the drawing of a a triangular heat shield of

Fig. 15. Reconstruction of a Roman Triangular heat shield of a palmette flanked at the base by a pair of dolphins A.D. 50-ca 100. [Fitch and Goldman 1994] catalog number 723.

Fig. 16. Reconstruction of a Hellenistic relief large bowl (Found in Resnik, Siculi [Brusic 1999] catalog number A197).

the oil lamp. Figure 16 demonstrates the automatic reconstruction of the base surface and the relief from the drawing of a Hellenistic relief large bowl. Figure 17 shows the reconstruction of a Roman triangular nozzle lamp of a horse. In this case prominent shading [Kolomenkin et al. 2011b] was used to enhance the 3D features of the reconstructed object.

Figure 18 demonstrates the user's fine tuning. Here, the user added extra step edges to two of the curves and modified the height of two other step edges. These changes enable height changes along a curve and improve the quality of the bird's tail and wing. The drawing also includes ridges (the bird's legs) and valleys (the bird's eye and the centers of the leaves), which are not supported by the automatic algorithm and were specified by the user.

As opposed to step edges, ridges and valleys have only local influence over the height of the relief. Adding a ridge (or a valley) changes the height only within the margins of the ridge. Thus, these curves are easier to deal with and are omitted from our line interpretation algorithm. They are, however, dealt with in the relief reconstruction stage.

Limitations. Our method for base estimation cannot change the topology of the models found in the database. For example, it fails to produce an accurate reconstruction of bases for oil lamps with

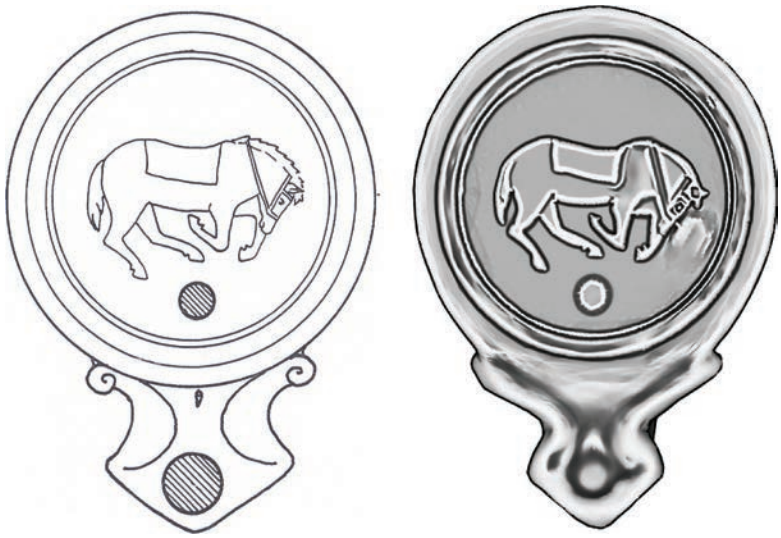


Fig. 17. Roman triangular nozzle lamp of a horse, A.D 50-ca. 100. [Fitch and Goldman 1994] catalog number 483.

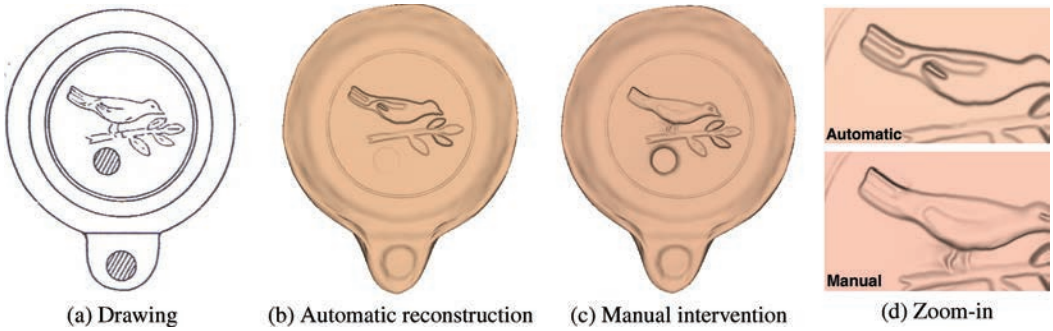


Fig. 18. An example of manual fine tuning of the results. The original drawing (a) includes ridges (the bird's legs) and valleys (the bird's eye and leaves). Automatic reconstruction (b) is enhanced by several simple manual operations to produce (c). Zoom-in (d) on the automatic and the manual reconstructions reveals that the automatically-obtained surface is less accurate. (Roman fat lamp of a bird on a spray of leaves, 25/20 B.C. - A.D. 40/45. [Fitch and Goldman 1994] catalog number 775)

a handle, when the most similar lamp from the database does not have a handle. Fortunately, in our experiments in most of the cases the database search returned a model with a handle.

A second limitation concerns our relief construction method. As seen in Figure 16, reliefs that are only partially visible (those that intersect the silhouettes) cannot be reconstructed. This is so, since obviously, the invisible part of the relief cannot be utilized.

8. CONCLUSIONS

In this article, we addressed the problem of automatic reconstruction of a relief object from a line drawing. Based on the observation that relief objects are composed of a smooth base and relief details, we focused on two subproblems: estimation of a smooth base from the silhouette and reconstruction of the relief on top of the base.

We propose a data-driven algorithm for generating the base from the outline of a given line drawing. The base is constructed irrespective of the details, by utilizing database search and deforming the retrieved most-similar object. This allows us to deal with objects that have complex bases.

While reconstructing reliefs from complex line drawings, we identified four challenges that have to be tackled: the sparsity of the lines, the ambiguity of the line drawing, the large number of strokes comprising the line drawing, and the interactions between close curves. A novel algorithm was proposed that addresses these challenges. It consists of two parts. First, the line drawing is interpreted, solving the challenges of ambiguity and input size, without requiring the user to manually specify the complex line drawing interpretation. Second, given the base and the interpretation, the relief object is reconstructed, addressing the challenges of sparsity and close-curve interaction.

The algorithm was implemented and tested on real complex archaeological illustrations. This lets the archaeologists reconstruct the shapes of artifacts for which, in many cases, the line drawings are the only remaining evidence.

ACKNOWLEDGMENTS

We thank the Laboratory of Computer Graphics & Multimedia at the Technion and the Zinman Institute of Archaeology, University of Haifa, for the scanned models of the archaeological artifacts.

REFERENCES

BROWN, B., TOLER-FRANKLIN, C., NEHAB, D., BURNS, M., DOBKIN, D., VLACHOPOULOS, A., DOUMAS, C., RUSINKIEWICZ, S., AND WEYRIC, T. 2008. A system for high-volume acquisition and matching of fresco fragments: Reassembling Theran wall paintings. *ACM Trans. Graph. 27*, 3, 1–9.

BRUSIC, Z. 1999. Hellenistic and Roman relief pottery in Liburnia (North-East Adriatic, Croatia). British Archaeological reports (BAR) International Series 817.

CHEN, X., KANG, S., XU, Y., DORSEY, J., AND SHUM, H. 2008. Sketching reality: Realistic interpretation of architectural designs. *ACM Trans. Graph. 27*, 2, 370–381.

CLOWES, M. B. 1971. On seeing things. *Artif. Intell. 2*, 1, 79–116.

COOPER, M. 2008. *Line Drawing Interpretation*. Springer-Verlag New York Inc.

CUDA. Cusp-CUDA based sparse linear algebra library. http://code.google.com/p/cusp-library/.

ELAD, M., TAL, A., AND AR, S. 2001. Content Based Retrieval of VRML objects—An iterative and interactive approach. *EG Multimedia, September,* 97–108.

FITCH, C. AND GOLDMAN, N. 1994. *Cosa, the Lamps.* Vol. 39. University of Michigan Press.

GINGOLD, Y., IGARASHI, T., AND ZORIN, D. 2009. Structured annotations for 2D-to-3D modeling. *ACM Trans. Graph. 28,* 5, 148.

GINGOLD, Y. AND ZORIN, D. 2008. Shading-based surface editing. *ACM Trans. Graph. 27*, 3, 95–101.

HUFFMAN, D. 1977. Impossible objects as nonsense sentences. *Comput. Meth. Image Anal.* 338–347.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3D freeform design. *ACM Trans. Graph. 19*, 3, 409 – 416.

JOSHI, P. AND CARR, N. 2008. Repoussé: Automatic inflation of 2D artwork. In *Proceedings of the Eurographics Workshop on Sketch-Based Interfaces and Modeling*.

KARPENKO, O. AND HUGHES, J. 2006. SmoothSketch: 3D free-form shapes from complex sketches. In *Proceedings of ACM SIGGRAPH 25*, 3. 589–598.

KASS, M., WITKIN, A., AND TERZOPOULOS, D. 1988. Snakes: Active contour models. *Intern. J. Comput. Vis. 1*, 4, 321–331.

KERAUTRET, B., GRANIER, X., AND BRAQUELAIRE, A. 2007. Intuitive shape modeling by shading design. In *Proceedings of the International Symposium on Smart Graphics*. 163–174.

KOLLER, D., TRIMBLE, J., NAJBJERG, T., GELFAND, N., AND LEVOY, M. 2006. Fragments of the city: Stanford's digital forma urbis romae project. In *Proceedings of the 3rd Williams Symposium On Classical Architecture*. Vol. 61. 237–252.

KOLOMENKIN, M., LEIFMAN, G., SHIMSHONI, I., AND TAL, A. 2011a. Reconstruction of relief objects from line drawings. In *Proceedings of the IEEE Conference on CVPR*. 993–1000.

KOLOMENKIN, M., SHIMSHONI, I., AND TAL, A. 2009. On edge detection on surfaces. In*Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2767–2774.

KOLOMENKIN, M., SHIMSHONI, I., AND TAL, A. 2011b. Prominent field for shape processing and analysis of archaeological artifacts. *Int. J. Comput. Vis. 94*, 1, 89–100.

LEE, S. AND FUNKHOUSER, T. 2008. Sketch-based search and composition of 3D models. In *Proceedings of the EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling*.

LEIFMAN, G., MEIR, R., AND TAL, A. 2005. Semantic-oriented 3D shape retrieval using relevance feedback. *Vis. Comput. 21*, 8–10, 865–875.

LIU, J., CAO, L., LI, Z., AND TANG, X. 2007a. Plane-based optimization for 3D object reconstruction from single line drawings. *IEEE Trans. Pattern Anal. Mach. Intell. 30*, 2, 315–327.

LIU, S., MARTIN, R., LANGBEIN, F., AND ROSIN, P. 2007b. Background surface estimation for reverse engineering of reliefs. *Int. J. CAD/CAM 7*.

MALIK, J. 1987. Interpreting line drawings of curved objects. *Int. J. Comput. Vis. 1*, 1, 73–103.

MEYER, M., DESBRUN, M., SCHRODER, P., AND BARR, A. H. 2002. Discrete differential-geometry operators for triangulated 2-manifolds. *Vis. Math. 3*, 7, 34–57.

NEALEN, A., IGARASHI, T., SORKINE, O., AND ALEXA, M. 2007. Fibermesh: Designing freeform surfaces with 3D curves. *ACM Trans. Graph. 26*, 3, 41–51.

RUSHMEIER, H. 2005. Eternal Egypt: Experiences and research directions. In *Proceedings of Modeling and Visualization of Cultural Heritage*. 22–27.

SCHNEIDER, R. AND KOBBELT, L. 2001. Geometric fairing of irregular meshes for free-form surface design. *Comput. Aided Geom. Design 18*, 4, 359–379.

SHIMSHONI, I. AND PONCE, J. 1997. Recovering the shape of polyhedra using line-drawing analysis and complex reflectance models. *Comput. Vis. Image Understand 65*, 2, 296–310.

SHIN, H. AND IGARSHI, T. 2007. Magic canvas: Interactive design of a 3-D scene prototype from freehand sketches. In *Proceedings of the Graphics Interface Conference*. 63–70.

SORKINE, O., COHEN-OR, D., LIPMAN, Y., ALEXA, M., RÖSSL, C., AND SEIDEL, H. 2004. Laplacian surface editing. In *Proceedings of the Eurographics Symposium on Geometry Processing*. 184–194.

STEGER, C. 1998. An unbiased detector of curvilinear structures. *IEEE Trans. Pattern Anal. Mach. Intell. 20*, 2, 113–125.

SUGIHARA, K. 1982. Mathematical structures of line drawings of polyhedrons-toward man-machine communication by means of line drawings. *IEEE Trans. Pattern Anal. Mach. Intell. 3*, 5, 458–469.

SYKORA, D., SEDLACEK, D., JINCHAO, S., DINGLIANA, J., AND COLLINS, S. 2010. Adding depth to cartoons using sparse depth (in) equalities. *Comput. Graph. Forum 29*, 2, 615–623.

WALTZ, D. 1975. Understanding line drawings of scenes with shadows. In Patrick Winston (Ed), *Psychology of Computer Vision*, 19–91, McGraw Hill.

WANG, Y., CHEN, Y., LIU, L., AND TANG, X. 2009. 3D reconstruction of curved objects from single 2D line drawings. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1834–1841.

WU, T., TANG, C., BROWN, M., AND SHUM, H. 2007. ShapePalettes: Interactive normal transfer via sketching. *ACM Trans. Graph. 26*, 3, 44–52.

YANG, C., SHARON, D., AND VAN DE PANNE, M. 2005. Sketch-based modeling of parameterized objects. In *Proceedings of the Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM)*.

Received April 2012; revised July 2012; accepted September 2012