Portugal. Math. (N.S.)
Vol. 75, Fasc. 3-4, 2018, 285–311
DOI 10.4171/PM/2020

Portugaliae Mathematica

Three-dimensional registration and shape reconstruction from depth data without matching: A PDE approach

Diogo A. Gomes, João Costeira and João Saúde

Abstract. The widespread availability of depth sensors like the Kinect camera makes it easy to gather three-dimensional (3D) data. However, accurately and efficiently merging large datasets collected from different views is still a core problem in computer vision. This question is particularly challenging if the relative positions of the views are not known, if there are few or no overlapping points, or if there are multiple objects. Here, we develop a method to reconstruct the 3D shapes of objects from depth data taken from different views whose relative positions are not known. Our method does not assume that common points in the views exist nor that the number of objects is known a priori. To reconstruct the shapes, we use partial differential equations (PDE) to compute upper and lower bounds for distance functions, which are solutions of the Eikonal equation constrained by the depth data. To combine various views, we minimize a function that measures the compatibility of relative positions. As we illustrate in several examples, we can reconstruct complex objects, even in the case where multiple views do not overlap, and, therefore, do not have points in common. We present several simulations to illustrate our method including multiple objects, non-convex objects, and complex shapes. Moreover, we present an application of our PDE approach to object classification from depth data.

Mathematics Subject Classification (primary; secondary): 65D18, 68U10; 94A08

Keywords: Computer Vision, Eikonal Equation, registration, reconstruction, 3D point cloud

1. Introduction

By watching their environment from several perspectives, humans effortlessly construct three-dimensional (3D) models of their surroundings by relating different views of partially visible complex objects. A significant amount of research in computer vision aims at giving computers the visual abilities of humans. The registration problem in computer vision consists of finding the relation (translation and rotation transformation) between different views of the same object. Here, we address this fundamental problem and develop a method to reconstruct the

3D shape of objects from depth data taken from various views. These data can be easily acquired using consumer products such as the Kinect camera. As we illustrate in several examples, our method reconstructs complex shapes, non-convex objects and multiple objects, and it does not need the views to overlap.

The reconstruction of 3D objects using visual data is a widely studied problem, see, for example, [3]. Here, we are interested in reconstruction of object shapes from depth maps. Due to the development of range or depth sensors, such as the Kinect camera, depth maps can easily be obtained. Before such devices were available, state-of-the-art reconstruction methods inferred depth relationships using intensity data. Earlier intensity-based methods were investigated in [11], [14], and [24]. However, these techniques lacked robustness and their results could be influenced by illumination, presence of shadows, or reflections. One advantage of depth sensors is that they are immune to such interferences because depth data depend only on the object's geometry.

Here, we use partial differential equations (PDE) to find the transformations relating 3D point clouds and to reconstruct a 3D model from depth data. We are given different views whose relative positions are unknown and corresponding depth data. The first step in our algorithm is to construct two distance functions for each view. The first one, the upper distance function, encodes the smallest possible body compatible with the data, a thin shell. The second one, the lower distance function, represents an object that occupies all the volume occluded by its visible surface. These distance functions solve the Eikonal equation [26]. The next step involves estimating the relative position of the views. We pick two views, estimate their relative position, merge them, and iterate this procedure until all views are merged. Given a rigid transformation relating two views, we consider the transformed distance functions and measure their mismatch. Our mismatch criterion compares the upper and lower distance functions from two views taking into account that the smallest possible body from one view is contained in the largest possible body from the other view. Finally, we minimize the mismatch and, thus, obtain the relative position of the views, after which the reconstruction is straightforward. To combine the lower distance functions, we intersect the two largest objects from each of the views and compute the distance to this body. Similarly, to combine the upper distance functions, we join the two smallest objects.

Our method is ''lossless'' because representing objects by their distance functions uses all information in the point cloud. Thus, our results depend only on the object's geometry. Another advantage of our algorithm is how it deals with occlusions: by merging different views, we can reconstruct parts of the object occluded in some of the images.

We end this introduction with an outline of the paper. We begin by reviewing previous work in Section 2. Next, in Section 3, we describe the building blocks of our method. We first address the theoretical foundations of the algorithm and dis-

cuss how to use depth data corresponding to different views to construct distance functions. Then, we describe how to combine different distance functions to compute the transformation mismatch between different views and how to merge them. Subsequently, we discuss the implementation of the algorithm, the numerical scheme, which uses a finite-differences fast-marching method. In Section 4, we present two-dimensional (2D) simulated experiments. We test our method with convex objects, non-convex objects, and a scene with two objects with partial occlusions. In Section 5, we use 3D point clouds from the Stanford scanning repository [13] to model a depth sensor and test our method. The objects chosen from the repository are intricate and nonconvex; thus occlusion phenomena occur. In Section 6 we use our method to automatically integrate real depth data acquired with a RGB-D camera, made available by the authors of [6]. Finally, we present an application to the partial-view clustering problem, where views from different objects must be clustered and classified.

2. Previous work

An earlier method to merge point clouds given by partial views of an object was proposed by Potmesil [21]. Starting with one of the point clouds, Potmesil looked at the surfaces that likely matched the original and chose the one with the best surface-segmented match. Then, by selecting points with specific features (such as maximum curvature), he estimated the relative position. Finally, by matching and merging surfaces iteratively, he obtained a 3D reconstruction. This method requires substantial pairwise overlapping of the views.

The Space Carving algorithm [12] recovers 3D shapes from images taken from multiple known viewpoints. These images determine constraints that the object satisfies. Next, starting with a volume containing the object, the algorithm checks each voxel (3D pixel) for consistency and removes (carves) all inconsistent voxels. Here, we work with distance functions rather than voxels, and we use depth data, not 2D images. Nevertheless, our algorithm behaves somewhat like a carving method because voxels at a positive distance from the body are carved out. However, our setting is more complex than the one in [12] because the relative positions of the views are unknown.

In [22], [23] the authors propose a method for registering 3D data points to a CAD (Computer-Aided-Design) model using local quadratic approximates to the squared distance function. In their method, for each data point from the 3D data, they first determine the closest point on the surface of the CAD model and determine the tangent plane at that point. Then, they use the squared distance to that tangent plane to that closest point on the CAD model surface. They minimize a quadratic objective function that uses the fact that the closest point to the tangent

plane in the surface of CAD model is along the normal of that point. After, as in ICP, they update the data points, by applying the determined transformation along the determined vector field, and iterate the procedure until some threshold is attained. Other variations and improvements of the previous described technique can be found in [15] and [9]. Unlike, our method, this procedure needs to have both the 3D data points and the CAD object model.

The Diffeomorphic Point Matching method [10] splits the 3D reconstruction problem into two problems. In the first, the Landmark Matching Problem, the point clouds have the same cardinality and their correspondences are known. In this case, the method selects a map that minimizes an energy functional to get a correspondence between the clouds. In the second problem, the Point Shape Matching Problem, the cardinality is distinct and the matches are unknown. To address this case, the points cluster and the solution to the first problem is used to find a one-to-one correspondence between the clusters.

The Iterative Closest Points (ICP) algorithm [29] aims at identifying the transformation that best aligns two datasets. The ICP algorithm consists of an iteration between two steps. First, an initial transformation is applied to a subset of a point cloud. Next, for each transformed point, the algorithm finds the closest point on the other point cloud. Finally, the transformation is refined using the correspondence between the points. This procedure is repeated until no further improvement is possible.

The partial matching of rigid shapes consists of merging distinct shapes with several similar parts. This happens, for example, if data acquisition is imperfect, there is occlusion, or there are scanning artifacts. To solve this problem, Bronstein and Bronstein [5] proposed the regularized partial matching algorithm. There, instead of using the ICP algorithm with rejection of dissimilar points, they introduce a multicriteria optimization problem to select two parts of two different objects that are regular, similar, and as large as possible. The solution to this problem is of Pareto-optimal type and uniqueness is not assured. Therefore, they look for a solution that maximizes a partial similitude measure. First, they find a rotation and translation that minimizes misalignment of two point clouds. Finally, for each point in both clouds, they find a membership function that minimizes a misalignment functional.

In [17] and [20], the authors use methods that estimate the motion and pose of a depth sensor to reconstruct 3D shapes. In [17], the algorithm merges depth data and constructs a 3D model in real time. To carry this out, Newcombe et al. track the camera's motion and, simultaneously, integrate depth data into a global model. First, they generate a vertex and a normal map from the depth data. Afterward, the pose of the sensor is estimated in real time using a multi-scale ICP alignment of current data and the previously predicted surface. Then, the current data are incorporated in the global model, which uses a truncated signed

distance function representation. Finally, they create a prediction of the surface. In this method, with a large displacement of the camera, drift can occur and reconstruction issues can arise. In [17], the merging of depth information relies on the continuous estimation of the camera's position and orientation using Simultaneous Localization and Mapping (SLAM) methods. To track the camera, these algorithms need depth data to overlap. In our approach, there is no camera tracking and overlap is not needed. State-of-the-art approaches like ICP and its variants, namely [17], assume or resort to other methods, like SLAM, so that they perform ICP on close/aligned enough 3D point clouds with common points. And then carry out local optimization to find the best possible rigid transformation. Our method is not directly comparable to those mentioned, since our approach, not only perform registration of 3D depth data (or point clouds) with common points but also successfully register 3D point clouds with no common points.

Recently, a probabilistic approach to the joint rigid registration of 3D point clouds was proposed in [8]. There, Evangelidis and Horaud assume that an arbitrary number of point clouds is generated from a single Gaussian Mixture Model (GMM). Hence, each point, after rotation and translation is produced from a Gaussian component. To register the point clouds, they estimate the rigid transformation (rotation matrices plus a translation vector) and the GMM parameters (mixing coefficients, mean vectors, and covariance matrices) with a maximum likelihood formulation using an expectation conditional maximization (ECM) algorithm.

3. The partial view matching problem

Our aim is to reconstruct 3D objects using depth data like the data that can be obtained using a Kinect camera. For that, we address the partial-view matching problem, which consists of finding the relation between incomplete views of an object, merge these views, and build a 3D reconstruction. Here, we are given two or more depth datasets obtained from distinct points of view. The correspondence between these, the rotation and translation relation, is not known a priori. Moreover, we do not assume that the object viewed is connected (thus, multiple objects are allowed) nor that there are common points in the different views. Next, we outline a PDE-based method to identify the correspondence between the views and to reconstruct the 3D model.

3.1. Depth data and the Eikonal equation. The first step in our 3D reconstruction is to use raw depth data to build an upper and lower bounds of the distance function. Then, we use those bounds to compute two solutions to the Eikonal equation. These solutions correspond to the distance to the largest and to the

smallest possible objects compatible with the depth data. Finally, we explain how to merge the data from different views, how to compute the relation between them, and how to reconstruct a 3D model of the object.

Depth data. A commonly used sensor in computer vision is the Kinect camera that, alongside with a standard RGB image, gives the depth of points of an object's surface. The depth is the distance obtained by ray-tracing from the sensor to the first point on the object's surface. If, for a cell on the sensor, there is no obstacle along a straight line departing from that cell, the depth is infinity (or, in practice, a large number). Consider a ray, R, with its origin at the sensor. A point, $x \in R$, is *non-occluded* if it lies between the sensor and the surface of the object. The remaining points in R are *occluded* points. We define two functions, $\psi^{\pm}$ in the following way:

- If x is a non-occluded point in a ray, R, then $\psi^{\pm}(x)$ is simply the distance along R to the surface of the object.
- If x is an occluded point, $\psi^{-}(x) = 0$, then $\psi^{+}(x) = +\infty$.

These two functions contain all information given by the depth data. Moreover, ψ^{-} is an upper bound to the distance function to the largest possible body compatible with the depth data, whereas ψ^{+} is an upper bound to the distance function to the smallest possible body.

The Eikonal equation and the distance function. Given an object, $\Omega \subset \mathbb{R}^3$, the *distance function* from a point, x, to Ω solves the Eikonal equation:

$$|Du| = 1, \qquad x \in \Omega^c,$$

where $u : \mathbb{R}^3 \to \mathbb{R}_0^+$ vanishes in Ω and Du is the gradient of u. Given a non-negative function, Ψ, the Eikonal obstacle problem is

$$\begin{cases} |Du(\mathbf{x})| = 1, \quad \text{on } u > 0 \\ 0 \leq u(x) \leq \Psi(x). \end{cases} \tag{1}$$

We use (1) to compute the upper and lower distance functions. The upper distance function, u^{+}, gives the smallest body possible from the depth data, using $\Psi = \psi^{+}$ as an upper bound. The lower distance function u^{-} corresponds to $\Psi = \psi^{-}$ and determines the largest possible body.

Merging two distance functions. Given two views, indexed by $i = 1, 2$, we compute the lower and upper distance functions, u_i^{-} and u_i^{+}. Then, we want to merge the information encoded by them. First, we assume that the relation between the two views is a known rigid motion map, T. This map consists of a rotation fol-

lowed by a translation. We fix the lower and upper distance functions for one view, u_1^- and u_1^+. Then, we pick the lower and upper functions of the second view, u_2^- and u_2^+, and compose them with T. To merge the lower functions, $u_1^-(\mathbf{x})$ and $u_2^-(T\mathbf{x})$, we start by computing their pointwise maximum, $u_{\max}^- = \max_{\mathbf{x}}\{u_1^-(\mathbf{x}), u_2^-(T\mathbf{x})\}$. The zero level sets of $u_1^-(\mathbf{x})$ and $u_2^-(T\mathbf{x})$ are the largest possible bodies compatible with each view. Hence, the zero level set of $u_{\max}^-$ is the intersection of these two bodies. In general, $u_{\max}^-$ is not a distance function; that is, it is only a solution to the Eikonal equation on the set $u_{\max}^- > 0$. However, it is an upper bound for the distance. Therefore, we compute the largest solution to the Eikonal equation that is below $u_{\max}^-$, which we call $\tilde{u}_T^-$. This is done by solving

$$\begin{cases} |Du_T(\mathbf{x})| = 1, & \text{on } u > 0 \\ 0 \le u_T \le u_{\max}^-. \end{cases}$$

To merge the upper distance functions, the process is simpler. We first rotate the distance function of the second view, and then we compute the pointwise minimum, $u_{\min}^+ = \min_{\mathbf{x}}\{u_1^+(\mathbf{x}), u_2^+(T\mathbf{x})\}$. The zero level sets of $u_1^+(\mathbf{x})$ and $u_2^+(T\mathbf{x})$ are the smallest possible bodies compatible with each view. Thus, the zero level set of $u_{\min}^+$ is the union of those two sets. Moreover, the resulting minimum is still a distance function. Therefore, $\tilde{u}_T^+ = u_{\min}^+$.

Error Function. To measure the mismatch between views, we observe that for the correct rigid motion, T, the lower distance function, $\tilde{u}_T^-(\mathbf{x})$, must be below its upper distance function, $\tilde{u}_T^+(\mathbf{x})$; that is, the smallest possible object corresponding to the views must be contained in the largest possible object. For a given transformation, T, we look for the region where this condition is not satisfied. Then, we integrate the squared difference of the upper and lower function over that region to get the *error function*,

$$E(T) = \int_A \left(\tilde{u}_T^-(\mathbf{x}) - \tilde{u}_T^+(\mathbf{x})\right)^2 d\mathbf{x}, \tag{2}$$

where $A = \{\mathbf{x} : \tilde{u}_T^- \ge \tilde{u}_T^+\}$. By minimizing $E(T)$, we obtain a rigid transformation that minimizes the mismatch between two views.

Object Reconstruction. Let T^* be a minimizer of (2). Then, the reconstruction of the objects from two different views is straightforward. The smallest possible object compatible with the data is given by the upper distance function, $\tilde{u}_{T^*}^+(\mathbf{x})$. More precisely, it is the points of the zero-level set of $\tilde{u}^+ : \{\mathbf{x} : \tilde{u}_{T^*}^+(\mathbf{x}) = 0\}$. We obtain the largest object in a similar way, as the zero level set of the lower distance function, $\{\mathbf{x} : \tilde{u}_{T^*}^-(\mathbf{x}) = 0\}$.

Integrating multiple views. To reconstruct an object, we may need to merge multiple views. The process described before can be extended by integrating multiple views iteratively. We merge the first two views and obtain upper and lower distance functions, $\tilde{u}_2^+$, $\tilde{u}_2^-$. Then, we proceed inductively by merging $\tilde{u}_n^+$, $\tilde{u}_n^-$ with u_{n+1}^+ and u_{n+1}^-, according to the method described before.

3.2. Implementation. Here, we discuss the numerical implementation of our method.

Discretized depth data. Consider a subset, $\Omega_i \subset \mathbb{R}^d$, containing the point cloud. Here, $d = 2, 3$ is the space dimension, and $i = \{1, \ldots, N\}$ are the different views of a given object or scene. We call Ω_i the *ambient space* associated with the i-th measurement. We assume Ω_i to be the product of d intervals, $\Omega_i = \Pi_{j=1}^d I_j$. For simplicity, we suppose that all I_j have the same length. The point cloud, Ψ^i, is gathered by a depth sensor, such as a Kinect camera, or given by synthetic data. We discretize the ambient space uniformly. In the 2D setting, this results in a square grid, whereas in the 3D case, it results in a cubic grid. Then, we proceed by inserting the depth values in that grid. In the 2D and 3D cases, we assume that the sensor is placed on an edge or a face of the square or cubic grid, respectively. Next, we fix a view, i, and let z be the length of the square or cubic grid. The discretization of the ambient space for this view is Ω_n^h, where n is the number of points in each dimension of the grid, and $h = \frac{z}{n}$ is the discretization step. In what follows, we discuss the 2D setting, $d = 2$, since the 3D case is analogous. For each view, we generate a depth matrix, $\psi \in \mathscr{M}^{2n}(\mathbb{R}_0^+)$, that assigns a depth value to each point of the discretization. Let Ψ, as before, denote the depth information simulated or retrieved by a depth sensor. The grid values for the *upper depth matrix* are given by

$$\begin{cases} \psi_{kl}^+ = |\Psi_k - lh|, & \text{for } l = 0, \ldots, n \\ \psi_{kl}^+ = \infty, & \Psi_{jk} > z, \end{cases} \tag{3}$$

where we assign the value ∞ (in practice, a large enough number) to points on a ray, R, with its origin at the sensor that does not intersect the object's surface. This procedure gives a depth grid on which the values of zero depth correspond to the visible surface of the object. Because we do not have information beyond that surface, we do not know if the object fulfills, partially or entirely, the space beyond those points. Thus, we consider a second matrix, the *lower depth matrix*, ψ^-, that assigns a zero depth to occluded points,

$$\begin{cases} \psi_{kl}^- = \Psi_k - lh, & \Psi_k - lh > 0 \\ \psi_{kl}^- = 0, & \Psi_k - lh \leq 0 \\ \psi_{kl}^- = \infty, & \Psi_k > z \end{cases} \tag{4}$$

for $l = 0, \ldots, n$. These two depth functions encode two limit situations. The first, given by (3), occurs if the object is only a thin shell. The second case, encoded in (4), corresponds to an object filling all the space beyond the visible surface. All possible objects that have surfaces consistent with the observations lie between these two cases.

Numerical implementation of the Eikonal equation. Now, we describe the monotonic scheme we use to compute u^+ and u^-. This scheme is closely related to the fast-marching method in [26]. We recall that $u^\pm$ solve the Eikonal obstacle problem (1) with the constraints given by $\psi^\pm$. Because both solve the same problem, in what follows, we omit the $\pm$ signs. Moreover, to simplify the discussion, we present the scheme for the 2D case; for 3D problems, the procedure is analogous. The coordinates of the grid points are (x_i, y_j), and we set $u_{ij} := u(x_i, y_j)$. We consider a nine-point stencil around any grid point. We use a monotonic numerical method as follows: we sort the $n \times n$ entries of the depth cube in increasing order. Next, we start with the point with a smaller depth value and proceed in increasing order. At each step, for a point (i, j), we consider its eight stencil neighbors. Then, we compute an auxiliary function, $\hat{u}$, given by

$$\begin{cases} \hat{u}_1 = \psi_{ij} \\ \hat{u}_p = \psi_{kl} + h, & kl = \{(i \pm 1, j), (i, j \pm 1)\},\ p = 2, \ldots, 5 \\ \hat{u}_p = \psi_{kl} + \sqrt{2}h, & kl = \{(i \pm 1, j \pm 1)\},\ p = 6, \ldots, 9. \end{cases}$$

The distance function is calculated as

$$u_{ij} = \min_{n=1,\ldots,P} \hat{u}_n,$$

where P is the number of points in the stencil, in this case $P = 9$. This procedure is repeated for $n \times n$ points in the discretization.

The preceding scheme is monotonically decreasing; any decrease in the neighbor's values leads to a decrease or to no change in the value of the solution at points (i, j). Thus, we can compute the distance function by visiting each point of the grid once.

Rigid transformations. Because we do not know the rigid body transformation that aligns both point clouds, we discretize the rotation angles, θ, and the translation, t. To select the correct rigid body transformation, we compare the resulting $\tilde{u}^-\big(T_i(\theta, t)\big)$ and $\tilde{u}^+\big(T_i(\theta, t)\big)$ using a mismatch criterion as before.

In our examples, we assume that the rigid transformation is a 2D rotation. The same procedure works for rigid motions, including translations and 3D rotations, at the expense of a higher computational cost.

Merging distance functions. Now, we explain how to merge two distance functions.

To merge two upper distance functions, we start by taking the pointwise minimum, which corresponds to the union of the shapes:

$$u^+_{\min,ij} = \min\{u^{1,+}_{ij}, u^{2,+}_{ij}\}.$$

As discussed in Section 3.1, the resulting function, $u^+_{\min}$, is a distance function. Note, in particular, that

$$\begin{aligned} u^+_{\min}(x_i) &= \min\{u^+_1(x_i), u^+_2(Tx_i)\} \\ &= \min\Big\{\min_{j\in I} u^+_1(x_j), \min_{j\in I} u^+_2(Tx_j)\Big\} \\ &= \min_{j\in I}\{\min\{u^+_1(x_j), u^+_2(Tx_j)\}\}, \end{aligned}$$

where I is the set of points in the stencil.

To merge two lower distance functions, we take the intersection of both objects, which corresponds to the pointwise maximum,

$$u^-_{\max,ij} = \max\{u^{1,-}_{ij}, u^{2,-}_{ij}\}.$$

This time, the resulting function may not be a distance function. We therefore look for the largest lower distance function, $\tilde{u}^-$, that is below $u^-_{\max,ij}$.

For this, we solve a discretized version of the following obstacle-type problem:

$$\begin{cases} |Du| = 1 \\ u \le \hat{u}^-_{\max}. \end{cases}$$

To compute the solution, we cannot use the previous scheme because we are looking for the largest lower distance function that is smaller than $u^-_{\max}$. We therefore use the following modified version:

$$\begin{cases} \bar{u}^-_p = \tilde{u}^-_{ij} \\ \bar{u}^-_p = \tilde{u}^-_{kl} - h, & kl = \{(i\pm 1, j), (i, j\pm 1)\} \\ \bar{u}^-_p = \tilde{u}^-_{kl} - \sqrt{2}h, & kl = \{(i\pm 1, j\pm 1)\}. \end{cases}$$

Then, the merged lower distance function is given by

$$\tilde{u}^-_{ij} = \max_{p=1,\ldots,P} \bar{u}^-_p,$$

where P is the number of points in the stencil.

Merging the views and reconstructing the 3D model. To determine the transformation angles between the views, we first compute the lower and upper distance functions of the first view, u_1^- and u_1^+, in its own frame and do the same for the second view in its own frame. Next, we rotate the distance functions of the second view, u_2^-, u_2^+, for each $\{\theta_i, i = 1, \ldots, N\}$, a discretization of the possible rotation angles. Then, we merge the upper distance function of the first view, u_1^+, with the rotated upper distance function, $u_2^+(\theta)$, and proceed similarly with the lower distance functions, u_1^- and $u_2^-(\theta)$. We perform this procedure for each $\theta \in \{\theta_i, i = 1, \ldots, N\}$.

As discussed in Section 3.1, the merged lower distance function, $\tilde{u}^-(\theta)$, must be below the merged upper distance function, $\tilde{u}^+(\theta)$. To measure the mismatch, for each θ_i, we determine the set of points where this does not hold:

$$A_i = \{\mathbf{x}_{kl} : \tilde{u}_{kl}^- > \tilde{u}_{kl}^+\},$$

where $i = 1, \ldots, N$ indexes a discretization of rotation angles.

We evaluate the error function (2) for each $i \in \{1, \ldots, N\}$ as

$$E(\theta_i) = \sum_{\mathbf{x}_{kl} \in A_i} \left(\tilde{u}^-(\mathbf{x}_{kl}) - \tilde{u}^+(\mathbf{x}_{kl}; \theta_i)\right)^2. \tag{5}$$

Then, the correct angles of rotation between the views correspond to the minimum of the components of E:

$$\theta^* := \operatorname{argmin} E(\theta_i).$$

The object reconstruction is then immediate; the smallest and largest possible objects are given by the zero-level sets of the upper and lower distance functions, respectively. To approximate the zero-level sets, we choose a small value, $\varepsilon > 0$, and find the points that satisfy $\tilde{u}^+(\mathbf{x}_{kl}, \theta^*) \leq \varepsilon$ and $\tilde{u}^-(\mathbf{x}_{kl}, \theta^*) \leq \varepsilon$.

3.3. Complexity. Now, we discuss the complexity of our algorithm. Let P be the number of points in the grid and let k be the number of discretizations of the rigid transformations. Then, the complexity of the algorithm is $kO(P \log P)$ as shown in Table 1. An advantage of our method is that the steps to solve the Eikonal equation and compute the rigid body transformations are fully parallelizable. For each view, we solve the Eikonal equation twice. To integrate two views, we apply k rigid-body transformations to one of the views and superpose them. This can be done in parallel for all k transformations.

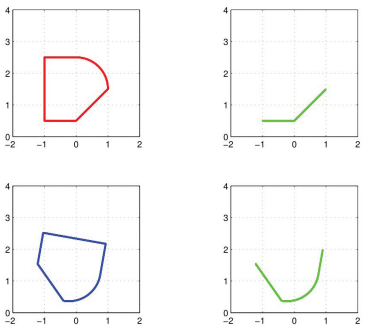
Table 1. Complexity of our algorithm for each view.

Step	Complexity
Compute upper/lower depth matrices	$O(P)$
Solve Eikonal equation (with FMM)	$O(P \log P)$
Merge upper distance functions	$O(P)$
Merge lower distance functions	$O(P \log P)$
Compute transformation, merging and reconstruction	$kO(P \log P)$
Total	$kO(P \log P)$

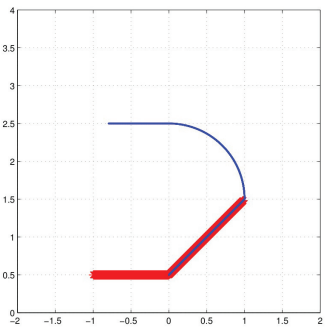
4. 2D simulations

To visualize how our algorithm works, we begin with the two-dimensional case (2D). We start with a single convex object. Then, we increase the complexity of the objects, and, finally, we deal with multiple objects. We only consider rotated views around the object/scene for simplicity; translations can be handled similarly at a higher computational cost.

4.1. Single convex object. We consider the 2D object, whose boundaries are drawn in red, shown in Figure 1. In blue, we show the rotation of the image by -100 degrees. We assume that the depth sensor is positioned at the bottom of the square and that it captures the depth information shown on the right-hand side of the views. In Figure 1b, we see both views superposed in the same frame.

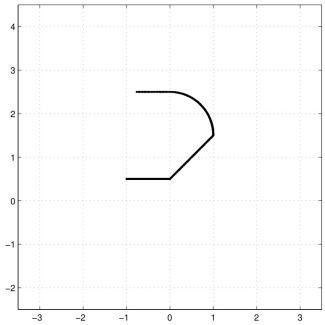


(a) The original object and two simulated views.

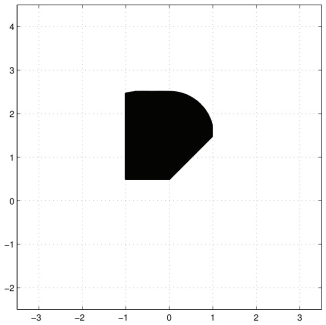


(b) Superposed views.

Figure 1. In the left figure, the red and blue plots are two different views of the same object. In blue, the object is rotated by -100 degrees relative to the red one. In green, we represent the depth data from the bottom to the top of the square. On the right hand side, both views are placed together in the same frame.



(a) Zero-level set for the upper distance function, $\tilde{u}^{+}_{T^*}$.



(b) Zero-level set for the lower distance function, $\tilde{u}^{-}_{T^*}$.

Figure 2. Two possible object reconstructions, using two different views.

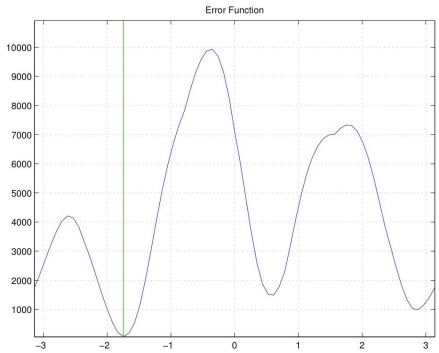
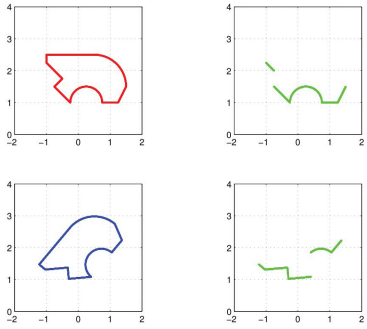


Figure 3. Cost function, $E(\theta)$, for $\theta \in [-\pi, \pi]$, in blue. In green, we see the true rotation value, -100 degrees.

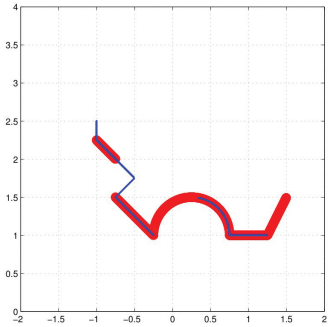
Using the algorithm from Section 3 and the information from each view (Figure 1a), we compute two possible reconstructions of the object. The first is given by the upper distance function, $\tilde{\boldsymbol{u}}^{+}_{T^*}$ (Figure 2a), and it is the smallest possible object compatible with both views. The second reconstruction (Figure 2b), uses the lower distance function, $\tilde{\boldsymbol{u}}^{-}_{T^*}$, and represents the largest object compatible with the data. The real object is between the largest and the smallest reconstruction.

In Figure 3, we show the error for each rotation of the second view. We see that the minimum cost is achieved when the rotation is -100 degrees.

4.2. Scenes with occlusions. The aim of this section is to demonstrate two important features of our method. The first is that our algorithm correctly handles

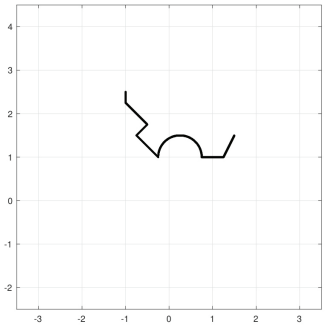


(a) In red we see the original object; in blue a rotation of 50 degrees, and in green the simulated views.

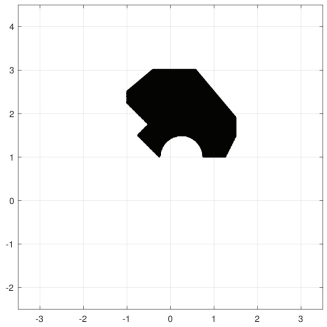


(b) Here, we see the both views superposed.

Figure 4. The red and blue plots are two different views of the same object. The blue one was rotated by 50 degrees. In green, we represent the depth data from the bottom to the top of the square. On the right-hand side, we show both views superposed.



(a) Zero-level set for the upper distance function, $\tilde{u}^{+}_{T^*}$.



(b) Zero-level set for the lower distance function, $\tilde{u}^{-}_{T^*}$.

Figure 5. Two possible object reconstructions, using two different views.

occlusions, as we show in the first experiment (Figure 4). The second is that it can handle views that do not overlap, but, nevertheless, it can reconstruct the object correctly (Figure 6).

We see that, even with occlusions, our algorithm successfully reconstructs the smallest possible object (Figure 5a) as well as the biggest possible object (Figure 5b) that satisfies the constraints determined by the views.

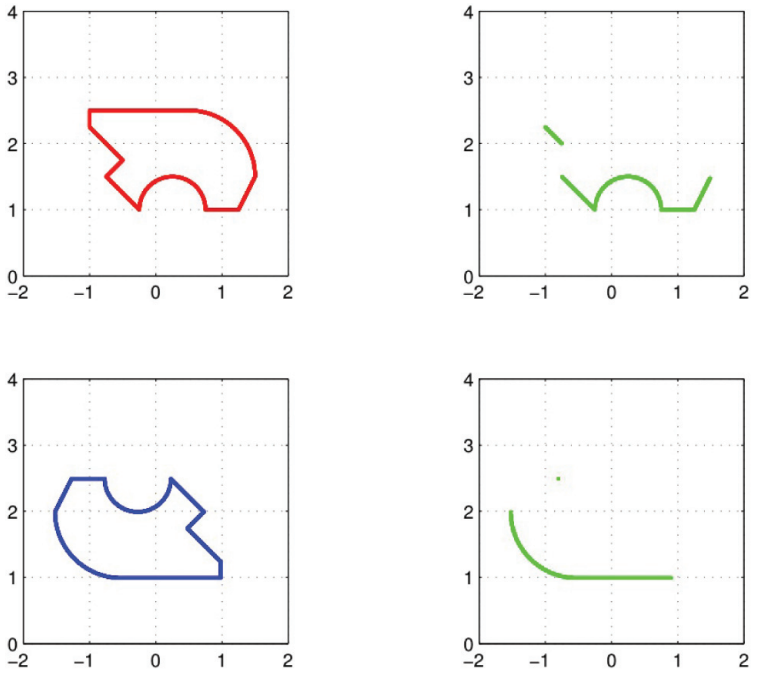
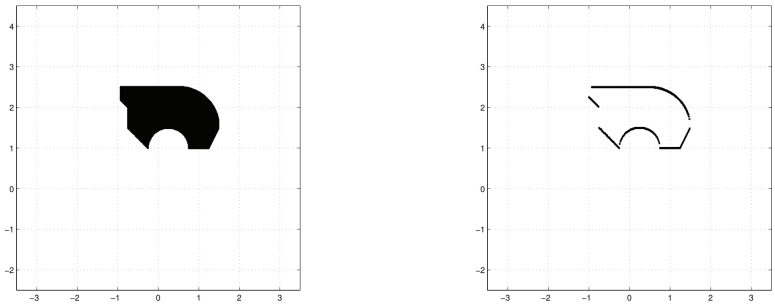


Figure 6. Both views of the object.



(a) Lower distance function. (b) Upper distance function

Figure 7. Object 2D reconstruction.

Now, we present an example where there is no overlap between the views. Yet, we reconstruct the 2D object correctly. In Figure 6, we show the object, its rotation, and the simulated captured views. In Figure 7, we see that our algorithm correctly computed the smallest and biggest possible objects compatible the views.

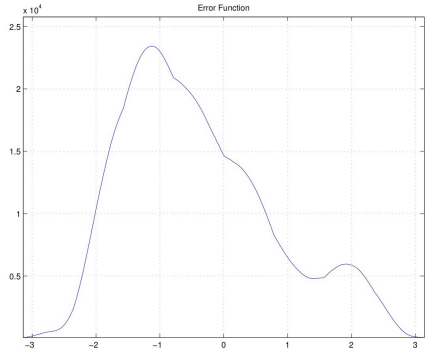
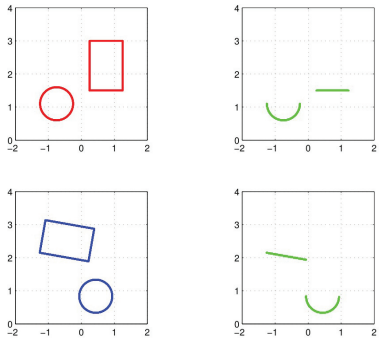
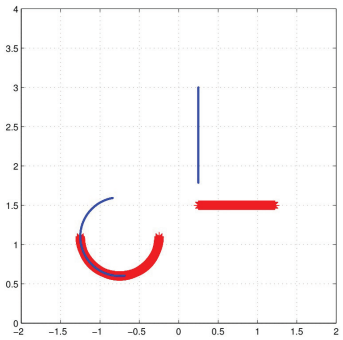


Figure 8. Cost Function.



(a) Objects, their rotations, and two different views.



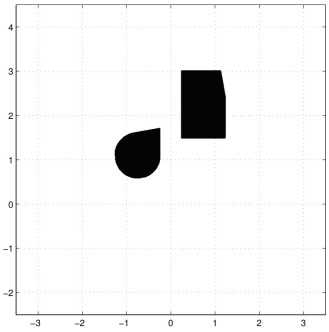
(b) Here we see the both views superposed.

Figure 9. On the left, of (a), the red and blue plots represent two different views of the same object. The blue one, shows, the object rotated by -80 degrees. On the right hand of (a), in green, we represent the depth data from the bottom to the top of the square. In (b) we show both views superposed.

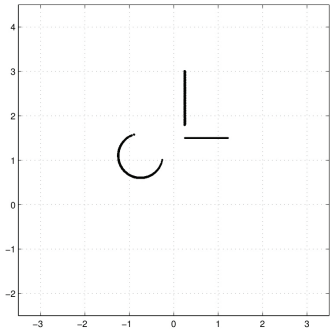
4.3. Two objects. As a last 2D example, we consider two objects (a circle and a rectangle). Two views are simulated as shown in Figure 9. Running our algorithm with the simulated data, we reconstruct the smallest and largest objects that are compatible with the views. We show our results in Figure 10.

5. 3D simulations

Now, we discuss 3D simulations. Both simulations, the Stanford Bunny and the Happy Buddha, use point clouds from the Stanford 3D scanning repository [13].



(a) Zero-level set of the lower distance function, $\tilde{u}^{-}_{T^*}$.

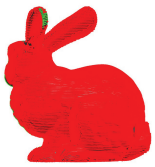


(b) Zero-level set of the upper distance function, $\tilde{u}^{+}_{T^*}$.

Figure 10. Reconstructions of two possible objects using the data from two different views.

(a) Both views.

(b) First view in green.



(c) Second view in red.

Figure 11. Stanford Bunny and its simulated views.

First, we present an experiment using the Stanford Bunny point cloud. In this experiment, we reconstruct the object using two views rotated by 180 degrees (Figures 11a to 11c). The reconstructed 3D object is shown in Figure 12.

Next, we perform an experiment using the Happy Buddha point cloud. In Figures 13a to 13c, we see the two views of the Buddha used for the object's reconstruction.

After applying our algorithm and using the data from the partial views showed in Figure 13a, we get the largest and smallest possible objects that comply with the constraints. In Figure 14, we show the former and in Figure 15 the latter.

In these two examples, we obtain an excellent reconstruction of the objects even though the overlap of the views is minimal.

(a) Zero-level set of the lower distance function, $\tilde{u}^{-}_{T^*}$.

(b) Zero-level set of the upper distance function, $\tilde{u}^{+}_{T^*}$.

Figure 12. Two possible object reconstructions using two different distance functions.

(a) Both views.

(b) First view in green.

(c) Second view in red.

Figure 13. Happy Buddha and simulated views.

Figure 14. Reconstructed 3D object, lower distance function, $\tilde{u}^{-}_{T^*}$, from different perspectives.

Figure 15. Reconstructed object, upper distance function, $\tilde{u}^{+}_{T^*}$, from different perspectives.

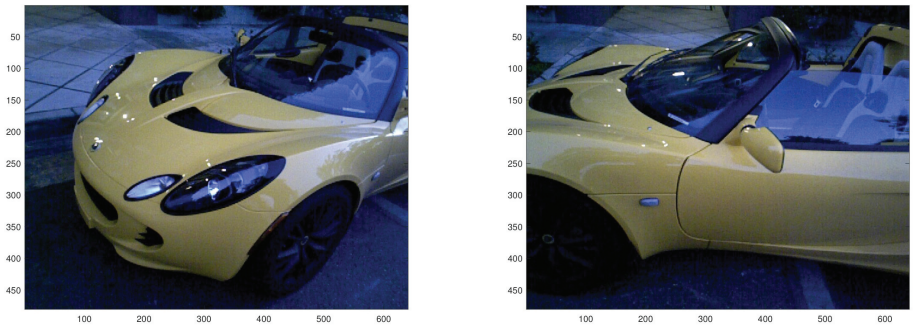
6. Merging partial views using real data

In this section we use our method to automatically fuse real depth data, that was obtained with a RGB-D camera. The real data we use was acquired by the authors of [6], and is publicly available in the website referred therein. In this experiment we automatically integrate two views of a car, see Figure 16 for the RGB pictures. Note that we show the pictures just for visualization, we do not use any RGB information to fuse the, corresponding, depth data.

We also tested the ICP using the same data and it failed to align the point-clouds correctly. We used the Matlab implementation of ICP (*pcregrigid*), with and without outlier rejection and did not get acceptable results.

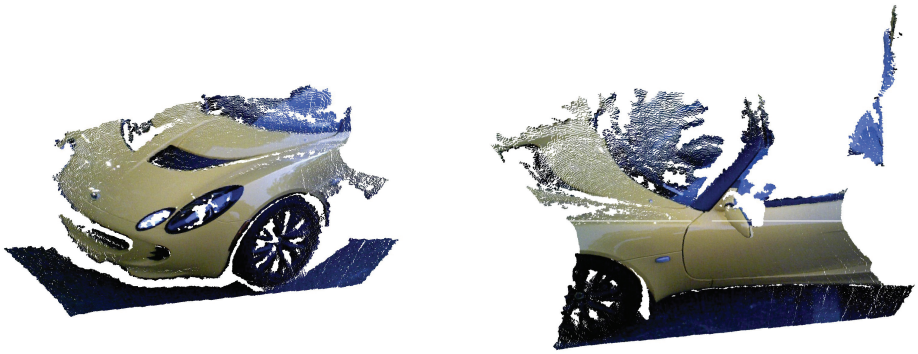
For both images we use the corresponding depth data that generate the 3D point clouds in Figure 17.

We fix the first partial view and use our method to automatically find the rigid transformation that correctly maps the second view to the referential of the first



(a) First picture. (b) Second picture.

Figure 16. Partial views of a car acquired with a RGB-D Camera.



(a) 3D point cloud for the first view. (b) 3D point cloud for the second view.

Figure 17. Point clouds for the partial views of the car acquired with a RGB-D camera.

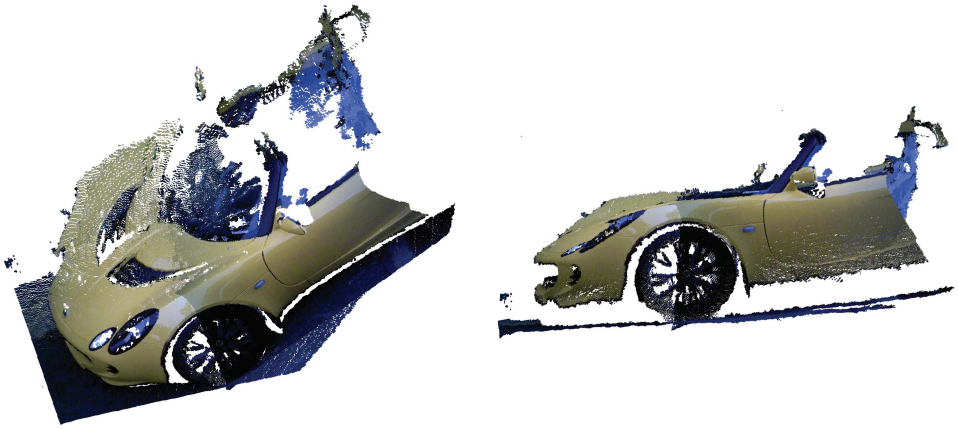


Figure 18. Different views for the automatically integrated 3D point cloud of a car using real depth data, acquired with a RGB-D camera.

one. This, fuses both depth maps and construct a 3D point cloud that correctly integrates the information from both views, see Figure 18.

To fuse the depth maps from both partial views, we discretize the space of rigid transformations, T, composed by a 3×3 rotation matrix, R and a translation vector t. We consider that the rotation angles $(\theta_x, \theta_y, \theta_z)$ take values in the interval $\left[-\frac{\pi}{4}, \frac{\pi}{4}\right]^3$. Regarding the discretization for the translation vector, we consider the interval $[-2, 2] \times [-0.5, 0.5] \times [-0.5, 0.5]$ (in meters). Regarding the ambient space Ω we considered the discretization of the cube $[-1.25, 1.25]^2 \times [0, 2]$ in a grid of $N = 100^3$ equally spaced points.

We first look for a coarse discretization of space of transformations, say $\tilde{N}_1 = 2^M$ for $M \in \mathbb{N}$ (we use $M = 18$, hence we divided in 8 equally spaced points each of the 6 intervals of possible rotations and translations). Then we chose half of the $\tilde{N}_1$ initial points, that held the lower error (2), and compute $\tilde{N}_2 = \frac{\tilde{N}_1}{2}$ additional points in the neighborhood of the previous ones, using a smaller step size than the previous one, say half of it. In the k step of the iteration we compute $\tilde{N}_k = \frac{\tilde{N}_1}{k}$ points. Repeat this step $M - 2$ times to find the best transformation. We used $M = 18$, and therefore search for about 500 thousand different transformations. We run our experiments on MATLAB 2016, using an Intel Xeon ES-2650 v2 processor with 16 cores and base frequency of 2.0 GHz. It took around two and a half hours to obtain the result.

We obtained the following solution: $(\theta_x, \theta_y, \theta_z) = (0.1571, 0.4712, 0.3142)$ radians, for the angles of rotation and $(t_x, t_y, t_z) = (-1.624, -0.333, 0.208)$ meters, for the translation vector.

In the same real data repository, [6], we use the depth data from the flower pot, that do not contain any common points see Figure 19.

Using the same methodology as previously, we search on 500 thousand different translations and automatically register the views in two hours and a half (156 minutes), see Figure 20.

We obtained the reconstructions depicted in Figure 21 and 22. The pictures in Figure 21, correspond to the upper distance function reconstruction, obtained in the $N = 100^3$ grid, so it is not as good as using the computed transformation and merge the original datasets, as in Figure 20.

(a) First view. (b) Second view.

Figure 19. Real data acquired with a RGB-D camera.

Figure 20. Fused real data for the flower pot.

Figure 21. Reconstructed flower pot, using the upper distance function.

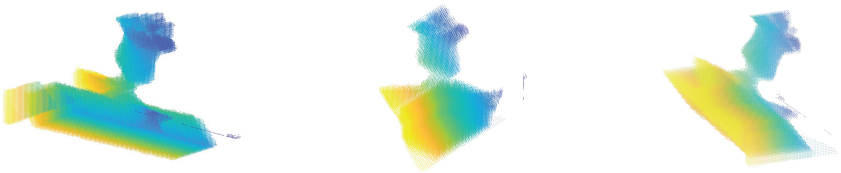


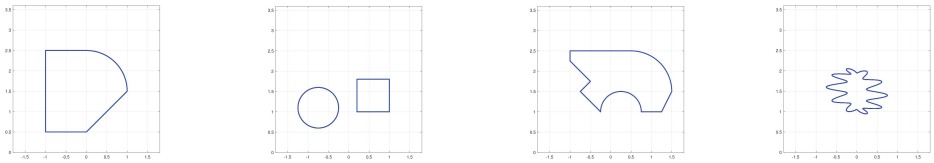
Figure 22. Reconstructed flower pot, using the lower distance function.

7. An application

Now, we apply our methods to the clustering problem of partial views. The problem is as follows: we are given N different views (or point clouds) from M different objects and we would like to cluster the views that belong to the same object. To perform this task, we compare all views by computing the minimum of $E(T)$ in (2). This procedure determines a similarity matrix, which is then used for clustering.

For illustration, consider the four 2D objects shown in Figure 23.

We simulate twelve views of each object as follows. For each object, the first view is taken from the bottom to the top. Then, we successively rotate each image by thirty degrees to obtain the new view, as shown in Figure 24.



(a) Object 1. (b) Object 2. (c) Object 3. (d) Object 4.

Figure 23. 2D objects.

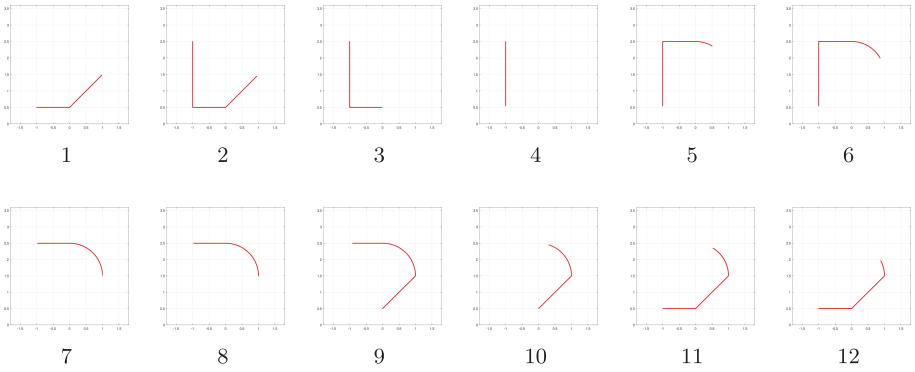


Figure 24. Several simulated views of the first object. Each view is rotated by 30 degrees, clockwise, from the previous one.

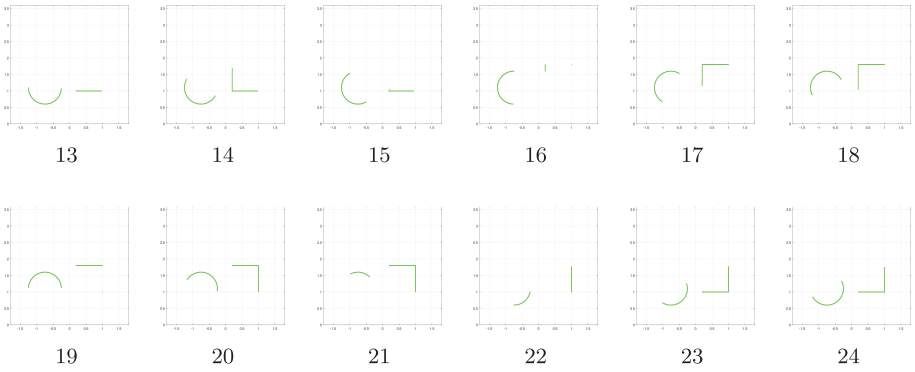


Figure 25. Several simulated views of the second object. Each view is rotated by 30 degrees, clockwise, from the previous one.

Given the views, the mismatch between views i and j is

$$E_{i,j} = \min_{\theta_m} \sum_{\mathbf{x}_{kl} \in A_m} \left(\tilde{\boldsymbol{u}}_i^-(\mathbf{x}_{kl}) - \tilde{\boldsymbol{u}}^+(\mathbf{x}_{kl}; \theta_m)\right)^2, \tag{6}$$

where $A_m = \{\mathbf{x}_{kl} : \tilde{\boldsymbol{u}}_{i,kl} > \tilde{\boldsymbol{u}}_{j,kl}\}$, and θ_m belongs to a discretization of $[0, 2\pi)$.

In Table 2, we show the twelve closest views to the view in the first column sorted by the value in (6). We choose to show the results for only the first view of each object due to space restrictions. The percentage of correct results is 89.8%. The numbers in red correspond to the views that does not belong to the correct object. In blue, we show the closest view. Of all 48 views, only in one case

View number	Closest views										
1	2	3	6	8	9	10	11	12	21	20	27
13	14	15	16	17	18	4	19	20	9	21	3
25	34	36	33	35	32	26	27	30	31	5	1
37	43	39	45	47	41	40	46	38	44	48	42

Table 2. Closest views to the view in the first column sorted in descending order of similitude. The numbers in blue indicate that the closest view belongs to the same object as the one in the first column; the views in red correspond to a distinct object from the one in the first column.

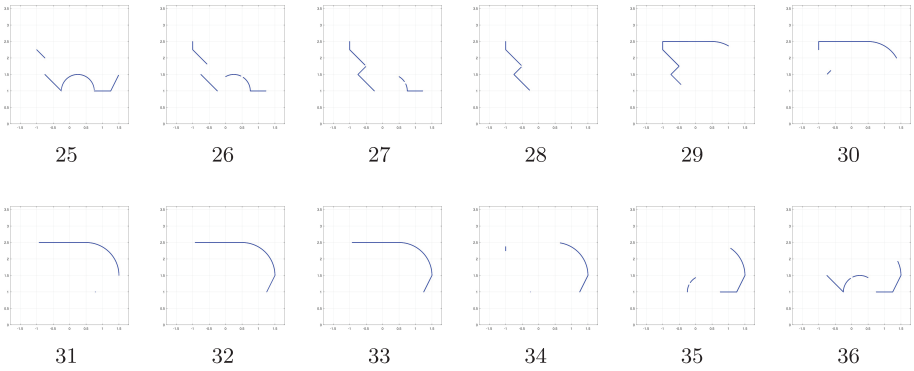


Figure 26. Several simulated views of the third object. Each view is rotated by 30 degrees, clockwise, from the previous one.

did the closest view not belong to the correct object: the closest view to view 28 in Figure 26 was view 37 in Figure 27. Hence, the percentage of correct results is 97.9%.

Finally, we assume that the number of objects is given. With this extra information, we perform spectral clustering [18] to identify which views belong to each four objects. For that, we construct the adjacency matrix, A, as follows

$$\begin{cases} A_{ij} = \left(\frac{E_{ij}}{\|E\|}\right)^{-1}, & i \neq j \\ A_{ii} = 1. \end{cases}$$

With the spectral clustering, we correctly identified the correspondence of all but one view. The algorithm mistakenly identified view 28 in Figure 26 as belonging to the object 4 in Figure 23. Thus, we achieved a success rate of 97.9%.

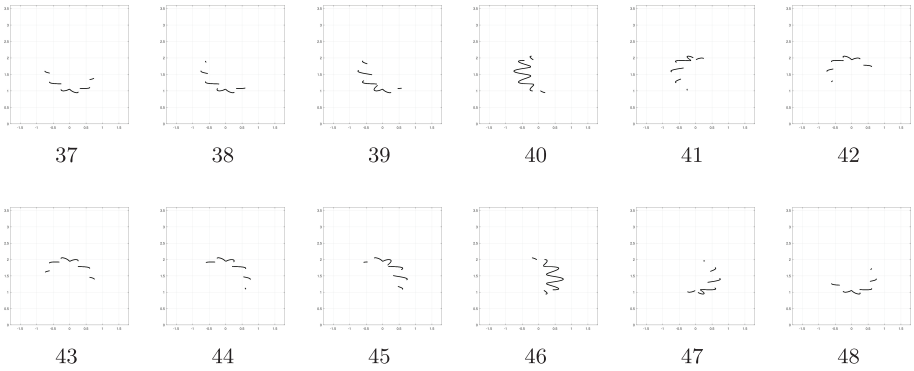


Figure 27. Several simulated views of the third object. Each view is rotated by 30 degrees, clockwise, from the previous one.

8. Concluding remarks

Our algorithm can reconstruct complex shapes from 2D and 3D depth data. Moreover, extensions such as the one discussed in Section 7 may prove relevant to object recognition and our methods may allow computer vision systems to learn object shapes autonomously. Recognition using intensity images is sensitive to factors external to the object; such factors include illumination, light source, and reflection. In contrast, recognition using range data is not sensitive to these factors. Furthermore, there are no smoothness requirements on the object surface. Finally, it may be possible to combine our methods with other approaches. For example, the recognition method proposed in [4] uses invariant surface characteristics such as the Gaussian and mean curvatures. To find a match, these quantities are compared to a library of objects. If several matches are found for the same object, a depth map is used for further verification. This last step can be carried out by our algorithm.

Acknowledgments. D. Gomes was partially supported by baseline and start-up funds from King Abdullah University of Science and Technology (KAUST). J. Saúde was partially supported by the Portuguese Foundation for Science and Technology through the Carnegie Mellon Portugal Program under the Grant SFRH/BD/52162/2013. The authors contributed equally to this work.

References

[1] M. Agrawal, A. Mittal and L. Davis. Multi-view reconstruction of static and dynamic scenes. In *Handbook of Mathematical Models in Computer Vision*, pages 405–422. Springer, 2006.

[2] Paul Besl and Ramesh Jain. Intrinsic and extrinsic surface characteristics. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Francisco, California*, pages 226–233, 1985.

[3] Paul J. Besl and Ramesh C. Jain. Three-dimensional object recognition. *ACM Computing Surveys (CSUR)*, 17(1):75–145, 1985.

[4] Paul J. Besl and Ramesh C. Jain. Invariant surface characteristics for 3d object recognition in range images. *Computer Vision, Graphics, and Image Processing*, 33(1):33–80, 1986.

[5] Alexander M. Bronstein and Michael M. Bronstein. Regularized partial matching of rigid shapes. In *European Conference on Computer Vision*, pages 143–154. Springer, 2008.

[6] Sungjoon Choi, Qian-Yi Zhou, Stephen Miller, and Vladlen Koltun. A large dataset of object scans. arXiv preprint arXiv:1602.02481, 2016.

[7] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 303–312. ACM, 1996.

[8] Georgios Evangelidis and Radu Horaud. Joint registration of multiple point sets. arXiv preprint arXiv:1609.01466, 2016.

[9] Natasha Gelfand, Niloy J. Mitra, Leonidas J. Guibas, and Helmut Pottmann. Robust global registration. In *Symposium on Geometry Processing*, volume 2, page 5, 2005.

[10] Hongyu Guo, Anand Rangarajan, and S. Joshi. Diffeomorphic point matching. In *Handbook of Mathematical Models in Computer Vision*, pages 205–219. Springer, 2006.

[11] Berthold K. P. Horn and Michael J. Brooks. *Shape from shading*. MIT Press, 1989.

[12] Kiriakos N. Kutulakos and Steven M. Seitz. A theory of shape by space carving. *International Journal of Computer Vision*, 38(3):199–218, 2000.

[13] Marc Levoy, J. Gerth, B. Curless, and K. Pull. The Stanford 3d scanning repository. http://www.graphics.stanford.edu/data/3dscanrep, 2005.

[14] Pierre-Louis Lions, Elisabeth Rouy, and A. Tourin. Shape-from-shading, viscosity solutions and edges. *Numerische Mathematik*, 64(1):323–353, 1993.

[15] Niloy J. Mitra, Natasha Gelfand, Helmut Pottmann, and Leonidas Guibas. Registration of point cloud data from a geometric optimization perspective. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 22–31. ACM, 2004.

[16] James Mure-Dubois and Heinz Hügli. Real-time scattering compensation for time-of-flight camera. In *Proceedings of ICVS*, 2007.

[17] Richard A. Newcombe, Andrew J. Davison, Shahram Izadi, Pushmeet Kohli, Otmar Hilliges, Jamie Shotton, David Molyneaux, Steve Hodges, David Kim, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011, pages 127–136. IEEE, 2011.

[18] Andrew Y. Ng, Michael I. Jordan, Yair Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems*, 2:849–856, 2002.

[19] Stanley Osher and James A. Sethian. Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79(1):12–49, 1988.

[20] Marc Pollefeys. 3d from image sequences: Calibration, motion and shape recovery. In *Handbook of Mathematical Models in Computer Vision*, pages 389–403. Springer, 2006.

[21] Michael Potmesil. Generating octree models of 3d objects from their silhouettes in a sequence of images. *Computer Vision, Graphics, and Image Processing*, 40(1):1–29, 1987.

[22] Helmut Pottmann and Michael Hofer. Geometry of the squared distance function to curves and surfaces. In *Visualization and Mathematics III*, pages 221–242. Springer, 2003.

[23] Helmut Pottmann, Stefan Leopoldseder, and Michael Hofer. Registration without ICP. *Computer Vision and Image Understanding*, 95(1):54–71, 2004.

[24] Emmanuel Prados and Olivier Faugeras. Shape from shading. In *Handbook of Mathematical Models in Computer Vision*, pages 375–388. Springer, 2006.

[25] Elisabeth Rouy and Agnès Tourin. A viscosity solutions approach to shape-from-shading. *SIAM Journal on Numerical Analysis*, 29(3):867–884, 1992.

[26] J. A. Sethian. *Level set methods and fast marching methods: Evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science.* Volume 3 of Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, Cambridge, second edition, 1999.

[27] James A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996.

[28] James Albert Sethian. *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*, volume 3. Cambridge university press, 1999.

[29] C. Stewart. Uncertainty-driven, point-based image registration. In *Handbook of Mathematical Models in Computer Vision*, pages 221–235. Springer, 2006.

Received September 13, 2018; revision received February 5, 2019

D. A. Gomes, CEMSE Division, King Abdullah University of Science and Technology (KAUST), Thuwal, Jeddah 23955-6900, Saudi Arabia

E-mail: diogo.gomes@kaust.edu.sa

J. Costeira, LARSyS, Instituto Superior Técnico, Ave. Rovisco Pais 1, 1049-001 Lisboa, Portugal

E-mail: jpc@isr.tecnico.ulisboa.pt

J. Saúde, ECE Department, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213, USA

E-mail: jsaude@cmu.edu