Mesh from Depth Images Using GR²T

Mairead Grogan & Rozenn Dahyot
School of Computer Science and Statistics
Trinity College Dublin
Dublin, Ireland
mgrogan@tcd.ie, Rozenn.Dahyot@tcd.ie
www.scss.tcd.ie/~mgrogan/IMVIP.html

Abstract

This paper proposes an algorithm for inferring a 3D mesh using the robust cost function proposed by Ruttle et al. [12]. Our contribution is in proposing a new algorithm for inference that is very suitable for parallel architecture. The cost function also provides a goodness of fit for each element of the mesh which is correlated to the distance to the ground truth, hence providing informative feedback to users.

Keywords: 3D reconstruction, Depth images, Generalised Relaxed Radon Transform.

1 Introduction

To capture the 3D shape of an object using low cost hardware opens interesting perspectives for non-specialist users for archiving, reproducing or displaying objects. Several solutions have already been proposed. For instance Autodesk [1] reconstructs a 3D object from multiple colour images with offline processing on the cloud. Using depth images recorded by the Kinect sensor, Microsoft has proposed a real-time algorithm called Kinect Fusion for computing a 3D mesh of an environment [9] on desktop computers. Project Tango [7] pushes this further by reconstructing a 3D scene in real-time on a mobile device using an integrated depth sensor. However the success of all algorithms depends on the recorded data available for inferring the 3D scene. It is therefore important to give feedback about the estimated mesh to help the user improve the results by, for instance, recording more data of under-exposed areas of the scene. In this paper, we propose to use the cost function recently proposed by Ruttle et al. [12] for 3D reconstruction from depth images. This cost function corresponds to a probability density function over the 3D space allowing us to directly give a measure of confidence for each vertex and edge on the inferred mesh, giving feedback about the local quality of the mesh. This cost function is also designed to be robust [5] to noise, and any new recorded data point contributes to the overall cost function in an additive and localised fashion. After a brief review (section 2), we propose a new algorithm in section 3 that is highly parallelizable. Section 4 presents quantitative and qualitative 3D reconstructions obtained using our approach with comparison to ground truth and Ruttle et al.’s reconstructions. Conclusions and future work are discussed in section 5.

2 State of the Art

Reconstructing a 3D mesh from RGB-D cameras is an area of intense research in computer vision. In the past, algorithms have mostly consisted of three steps - denoising the depth images from several camera views, converting them to 3D point clouds and aligning the point clouds to recreate the surface [3, 9]. Izadi et al. proposed a real time 3D reconstruction algorithm (KinectFusion) using depth information [9]. An iterative pipeline is implemented which processes each depth image consecutively and uses a volumetric surface representation to generate
a mesh of the scene. As a preprocessing step, a bilateral filter is applied to the raw depth data in order to reduce noise. This step ignores the uncertainty associated with the depth information and pixel positions as well as resulting in a loss of important information. A vertex and normal map of the scene from the first depth map are then computed using the connectivity of the pixels in the depth image.

For each consecutive depth map the pose of the camera is estimated and the depth information is fused with a volumetric truncated signed distance function (TSDF) [4] representing the scene. This representation gives a signed value to each voxel in the scene, depending on how far it is from the surface of the object. There is no measure of confidence associated with the vertices on the object’s surface and the distances given to each voxel are not calculated using a robust objective function, but using a weighted distance measure. In order to render the surface in the scene a per pixel ray cast is performed. Each pixel’s corresponding ray is calculated and marched starting from it’s minimum depth value until the surface interface is found. This fully parallel mapping algorithm takes full advantage of GPU processing hardware and scales naturally with processing and memory resources.

Recently, methods have been proposed which generate a density function from 3D depth images or point clouds [12, 13]. To find points on the surface these density functions are then explored using either gradient ascent algorithms or marching cubes. Ruttle et al. [12] proposed to accurately infer the 3D shape of an object captured by a depth camera from multiple view points. The Generalised Relaxed Radon Transform (GR2T) [5] is used here to merge all depth images in a robust kernel density estimate that models the surface of an object in the 3D space. The kernel is tailored to capture the uncertainty associated with each pixel in the depth images and the resulting cost function is defined for a 3D location $\Theta$ as:

$$\bar{\Pi}(\Theta) = \frac{1}{C} \sum_{c=1}^{C} \frac{1}{N_c} \sum_{i=1}^{N_c} p_i(F(x_c^{(i)}, \Theta, \Psi_c)),$$

where $C$ is the number of recorded camera views (with known camera parameters $\Psi_c, \forall c = 1, \cdots, C$) and $N_c$ is the number of pixels in the image generated by camera $c$, $x_c^{(i)}$ is a triplet of values recorded by camera $c$ corresponding to the 2D location and the depth value of the pixel $i$. This function accounts for uncertainties in the observations via the probability density function $p_i$ that is chosen Gaussian. $F$ is a link function associated with the pin-hole camera model connecting a 3D position to its projection in an image plane [8, 12]. Suitable values must also be chosen for the parameters $h_1$, $h_2$ and $h_3$, which account for noise in the pixel and depth values.

To extract a surface mesh using the cost function $\bar{\Pi}(\Theta)$, Ruttle et al. proposed a two stage process [12]. First maxima of the cost function are extracted. These are then connected in a second step by finding vertices and edges that connect them by following the ridge created by the object’s surface in the cost function $\bar{\Pi}(\Theta)$. Both algorithms correspond to gradient ascent algorithms, the first is initialised by several positions in the 3D space to converge to several local maxima of $\bar{\Pi}(\Theta)$, while the second algorithm is initialised with the output of the first algorithm. This approach is simplified further by considering 2D slices in the 3D space, and performing the optimisation in parallel in these 2D manifolds.

This surface exploration technique is time consuming and inefficient. As an alternative to this two stage process, we propose an algorithm constraining the solution to a 1D manifold (a ray in the 3D space) to estimate a vertex on the surface of the object. This process is performed for each pixel in the depth images independently and the resulting approach is highly parallelizable. Connectivity between vertices is inferred automatically using pixel neighborhood information from the depth images.

## 3 Mesh Inference from $\bar{\Pi}(\Theta)$

A mesh is a discrete representation of a continuous coloured 3D surface and is made up of a number of different elements. In our approach, we used the .ply format, which can be used to
store a variety of mesh properties including vertices, edges, faces, vertex colour, edge colour and vertex normals. We focus on efficiently inferring the vertices, faces and edges of the mesh as well as creating informative colour information which indicates the likelihood value of a particular vertex and edge.

In order to explore the density $\Pi_k(\Theta)$ and determine which 3D points are most likely to be on the object’s surface we propose generating a separate mesh for each camera view by casting a ray from the camera centre through each pixel in the image into 3D space, as shown in Figure 1a. This ray is then marched, starting from the calculated depth value, until the maximum likelihood value $\hat{\Theta}$ is found.

![Image](image.png)

(a) Ray based optimisation of $\Pi_k(\Theta)$.

(b) Generating the edges of the mesh.

Figure 1: Ray based strategy for Mesh from Depth images.

In order to calculate $\hat{\Theta}$, the point of maximum likelihood on the ray passing through the camera centre $C_{\Psi_c} \in \mathbb{R}^3$ and pixel $(x_1, x_2)$, we maximize the following:

$$\hat{\Theta} = \arg\max_\beta \Pi_k(\Theta)$$ \hspace{1cm} (2)

subject to the constraint that

$$\Theta = C_{\Psi_c} + \beta \vec{n}, \beta \in \mathbb{R},$$ \hspace{1cm} (3)

where $\vec{n}$ is the direction of the ray. As we can express $\Theta$ in terms of known parameters $C_{\Psi_c}$ and $\vec{n}$, and a one dimensional latent variable $\beta$, we have reduced our latent space from three dimensions to one dimension. This greatly reduces the computational cost of our optimisation problem. $W$ and $H$ represent the width and height of the image in pixels and $f_x$ represent the focal length in the horizontal axis. We define the horizontal field of view to be

$$\phi = \arctan\left(\frac{W/2}{f_x}\right).$$ \hspace{1cm} (4)

The distance between the camera and the projection plane $d$ is given by the formula $d = \frac{1}{\tan(\phi)}$. We let $a = \frac{W}{f}$ and define $\vec{n} = (n_1, n_2, n_3)$ to be:

$$\begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix} = \left( R(\Psi) \right)^{-1} \begin{pmatrix} \psi_4 \\ \psi_5 \\ \psi_6 \end{pmatrix} \begin{pmatrix} \frac{2x}{W} \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} \frac{2y}{W} \\ \frac{2z}{W} \end{pmatrix} \begin{pmatrix} 0 \\ 2 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ d \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix},$$ \hspace{1cm} (5)

where $R(\Psi)$ is the camera rotation matrix and $(\psi_4, \psi_5, \psi_6)$ is the camera translation vector [14, 6]. Given an observed depth value of $x_3$ at the pixel $(x_1, x_2)$, the point on the ray which is a distance $x_3$ from the camera centre is given by $\beta^{(0)} = \frac{a x_3}{\gamma}$ (initial guess) where

$$\gamma = \frac{\vec{n} \cdot C_{\Psi_c}}{\|\vec{n} \times C_{\Psi_c}\|}.$$ \hspace{1cm} (6)
A Newton Raphson gradient ascent algorithm is implemented to iteratively update the position on the ray until the point with maximum likelihood is found. This process is repeated for each pixel in the image, and generates a point cloud of the portion of the surface visible from camera \( c \). This is repeated for all \( C \) cameras, generating \( C \) point clouds. Our approach is highly parallelizable as each ray can be computed and marched independent of the other rays.

Using this method, points are found at regular intervals along the surface and each point has a corresponding pixel in the image. We use the connectivity of the pixels to create the edges of the mesh. We consider four pixels \( i, i+1, j \) and \( j+1 \), as seen in Figure 1b, which make up a \( 2 \times 2 \) square in the image. We also consider the 3D points \( x_i, x_{i+1}, x_j \) and \( x_{j+1} \) that were found by tracing the ray through each of these pixels. We create four edges \([x_i, x_{i+1}], [x_j, x_{j+1}], [x_i, x_j], [x_{i+1}, x_{j+1}]\) between these points since their corresponding pixels are connected in a horizontal or vertical direction. Then, in order to ensure that the faces of the mesh are triangular in shape, we also create an edge between points \( x_i \) and \( x_{j+1} \). This is a very simple meshing algorithm which is easy to implement and eliminates the need to cluster the data or calculate vertex neighbourhoods as in other meshing algorithms [11, 2, 10].

We set the colour value of each vertex \( \Theta \) in the .ply mesh according to its likelihood value \( \text{lik}(\Theta) \). For each edge in the mesh, the barycentre \( B \) of the edge is calculated. The colour value of the edge is then set according to the value of \( \text{lik}(B) \). This can be seen in Figure 2. This allows the user to see which vertices and edges have a high or low likelihood, and which regions of the object may have been poorly scanned.

4 Experimental Results

Our approach was first applied to the ground truth Stanford Bunny mesh (size \( 10 \times 13 \times 13 \) in cm). Autodesk 3DS max was used to generate 12 depth images of the bunny, with no noise added to the depth values (apart for the digitisation process in creating the projected depth images). The camera parameters were assumed to be known. We set the pixel bandwidth to \( h_1 = h_2 = 2 \) and depth bandwidth to \( h_3 = .001 \). For each camera view and corresponding depth image, a mesh was generated using our method. Figure 2 presents four meshes: the colour of each vertex in a mesh represents the likelihood that the vertex is on the bunny surface. Blue vertices have a low likelihood value and many appear at the edge of the mesh as their rays do not intersect with the bunny. Vertices with a low probability can be easily removed from the mesh by thresholding the likelihood values (second row of Figure 2). These colour values also illustrate which regions of the object have been poorly scanned, allowing the user to scan them in order to ensure that a more reliable mesh is generated.

The average time taken to converge to the point with highest likelihood on a given ray is \( .5162 \) seconds, with a standard deviation of \( .7080 \) seconds (non optimised Matlab code on a single core). The average number of iterations needed per ray is 71.2106. Our algorithm is highly parallelizable (as each ray can be marched independently) and optimising it to perform on the GPU would result in considerable speedup.

In Figure 3 we compare our reconstructed results to those obtained by Ruttle et al. in [12]. The colours of each vertex in these meshes represent the distance to the closest point on the ground truth Stanford Bunny mesh. The algorithm proposed by Ruttle et al. performs well apart from concave regions such as the neck and between the ears. Their meshing algorithm creates edges between points on different ears, and between points on the head and back. The red vertices on the bunny in Figure 3 (f) and (h) represent these meshing errors. Our results (top row of Figure 3) show that our meshing algorithm has eliminated these errors as it only considers vertices and edges with a high likelihood value.

We computed the average distance between the reconstructed meshes and the ground truth bunny mesh. The average distance is 0.000527m for our algorithm. This can be compared to 0.000711m obtained with Ruttle et al.’s algorithm [12]. We also investigated the correlation between the likelihood value \( \text{lik}(\Theta) \) of a vertex \( \Theta \), and the distance between \( \Theta \) and the ground truth Stanford Bunny mesh. We have found that a threshold on the likelihood values can easily
Figure 2: Meshes from 4 camera views (visualisation with Meshlab meshlab.sourceforge.net): with all vertices and edges shown with their probability (top) and when low probability vertices are deleted (bottom). The Max value on the colour scale refers to the maximum value of lik(Θ) found on each mesh.

Figure 3: Visual comparison with ground truth (using Cloud Compare software www.cloudcompare.org): our method (top row), and Ruttle et al. [12] (bottom row).

be set to keep 73.65% of vertices, amongst which only 3% are vertices far from the ground truth (or 97% of vertices with high likelihood are close to the ground truth). This indicates that lik(Θ) is a good measure of confidence for each element of the mesh. It can provide users with feedback to improve areas of the surface that have low likelihood and are therefore very likely to be far from the ground truth. Because this ground truth mesh is not available in general, lik(Θ) can be used as a good substitute.

We have run several experiments with noisy depth images. Similarly to Ruttle et al., the cost function is robust to noise and our algorithm is not affected by this noise on the depth values.
5 Conclusion

We have proposed another algorithm for optimisation of the robust cost function proposed by Ruttle et al. [12]. This new approach allows us to infer a mesh of vertices and edges for each camera view including a measure of uncertainty for each element in the mesh. Future work will focus on stitching together the meshes created for each camera view so that a single mesh is generated. Considering the confidence value associated with each vertex will ensure those vertices with a higher likelihood will be given preference.

Acknowledgements

This work has been supported by a Ussher scholarship from Trinity College Dublin (Ireland), and partially supported by EU FP7-PEOPLE-2013-IAPP GRAISearch grant (612334).

References