Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online
CONFERENCE PAPER ARCHIVE

IFAC-PapersOnLine 49-15 (2016) 218–223

Optimizing Triangle Mesh Reconstructions of Planar Environments

Thomas Wiemann * **Kai Lingemann** ** **Joachim Hertzberg** ***

* *Osnabrück University, Germany (twiemann@uni-osnabrueck.de)*
** *DFKI Robotics Innovation Center, Osnabrück Branch, Germany (kai.lingemann@dfki.de)*
*** *Osnabrück University and DFKI Robotics Innovation Center, Osnabrück Branch, Germany (joachim.hertzberg@uni-osnabrueck.de)*

Abstract:
Automatic surface reconstruction from point cloud data is an active field of research in robotics, as polygonal representations are compact and geometrically precise. Standard meshing algorithms produce many redundant triangles. Therefore methods for optimization are required. In this paper we present and evaluate a mesh optimization algorithm for robotic applications that was specially designed to exploit the planar structure of typical indoor environments.

© 2016, IFAC (International Federation of Automatic Control) Hosting by Elsevier Ltd. All rights reserved.

Keywords: 3D Mapping, Surface Reconstruction, Mesh Optimization, Segmentation

1. INTRODUCTION

Reconstruction of polygonal models from point cloud data has recently drawn much attention in the robotics community (Wiemann et al., 2013; Karpathy et al., 2013; Günther et al., 2013). Especially KinectFusion (Izadi et al., 2011) and other methods that rely on RGB-D cameras (Steinbruecker et al., 2014) are of interest, since they allow large areas to be reconstructed using low cost sensors in near real time by exploiting the regular structure of the depth images of such cameras. Besides these RGB-D based methods, the Las Vegas Surface Reconstruction Toolkit (LVR) [1] (Wiemann, 2014; Rinnewitz et al., 2013) provides efficient methods for reconstruction from arbitrary point clouds with focus on high resolution laser scans. The reconstructed surfaces are usually represented as triangle meshes. Together with bitmap textures generated from color data, such meshes can be rendered efficiently.

Triangle meshes reduce the amount of data that is needed to represent an environment significantly compared to the raw point cloud data. However, basic reconstruction methods usually produce more triangles than needed for a memory efficient environment representation, especially in the presence of planar surfaces. In this paper we present a method to reduce the number of triangles in polygonal meshes of planar environments drastically without losing geometric precision. This method is evaluated in detail and compared with the state of the art.

2. RELATED WORK

Surface reconstruction from point cloud data is a classic field of research in computer graphics. Mostly Marching Cubes based methods (Lorensen and Cline, 1987) are used to compute triangle meshes. In order to use this method, an implicit mathematical representation of the scanned surfaces is needed. State of the art is the use of signed distance functions as introduced in Hoppe et al. (1992) for local approximation. Poisson Reconstruction (Kazhdan et al., 2006) tries to find a global representation by interpreting the search for an appropriate implicit function as a Poisson Problem that can be solved numerically. Besides Marching Cubes, algorithms for direct triangulation of point clouds like Power Crust (Amenta et al., 2001) or Alpha Shapes (H. Edelsbrunner and E.P. Mücke, 1994) can be used. A detailed evaluation of freely available reconstruction methods is given in (Wiemann et al., 2015).

Mesh optimization is usually done by iteratively removing triangles via edge collapses. In this process, the edge between two adjacent triangles is collapsed to a single new vertex, such that these two triangles are deleted and the area of the adjacent triangles is increased. To determine the edges that can be safely removed without altering the geometry too much, several metrics have been proposed. A simple and fast metric is presented in Melax (1998) in the context of mesh optimization in computer games. Garland and Heckbert (1997) use quadric error metrics to determine the local error and to compute a new vertex position in the edge collapse process. Hoppe's mesh optimization (Hoppe et al., 1992) is based on a global energy function that takes vertex count, convexity and squared distance towards the initial data points into account.

In this kind of optimization elements are removed from the mesh until a user given quality threshold is violated or a predefined mesh size is reached. Our approach is different from these methods, since it is based on planar optimization via region growing. The idea is to detect planar patches in the reconstruction and to compute an alternative triangulation of these areas using as few triangles as possible. In non-planar regions no triangles will be removed to maintain geometric accuracy, as described in the next section.

[1] `http://www.las-vegas.uni-osnabrueck.de`

2405-8963 © 2016, IFAC (International Federation of Automatic Control) Hosting by Elsevier Ltd. All rights reserved.
Peer review under responsibility of International Federation of Automatic Control.
10.1016/j.ifacol.2016.07.735

3. PLANAR OPTIMIZATION

The planar optimization algorithm is an extension of the mesh optimization procedure presented in Wiemann et al. (2012). Besides the region growing based approach presented in this paper, we added further processing steps to improve the quality of the optimized meshes, especially for reconstructions of environments that display mostly planar surfaces like typical robotic indoor scenarios. The procedure consists of the following steps: Iterative plane segmentation based on the region growing approach of Wiemann et al. (2012), optimization of the vertex positions between planar regions, contour extraction and re-triangulation. The details of the single steps are presented in the following sections. The first step in the optimization process is recapitulated from Wiemann et al. (2012) to make this paper self-contained.

3.1 Plane Segmentation

The first step in our optimization procedure is to detect planes in the initial triangle mesh via region growing in a half edge representation. This data structure allows to find the neighbors of a triangle in constant time. Two adjacent triangles are assumed to belong to the same plane, if the angle between their normals is below a given threshold. To extract all planes, we start with some triangle and check if adjacent triangles have a similar normal, i.e., the angle between the normals is below a given threshold. For all neighboring triangles that fulfill this condition, the search is recursively continued until the border of a planar region is found. These border edges are stored in a list that represents the contour of the planar region. After all recursive searches from the starting triangle have stopped, a search with a new, previously unvisited triangle, is started as described in Algorithm 1.

Algorithm 1 The mesh simplification algorithm.

```
function SIMPLIFY
    for all faces do
        currentFace ← visited
        FUSE(currentNormal, currentFace, currentList)
        borderList ← currentList
        CREATEPOLYGON(borderList)
        currentList ← empty
    end for
end function

function FUSE(startNormal, currentFace, listOfBorders)
    currentFace ← visited
    for all neighbors of currentFace do
        angle ← startNormal · neighborNormal
        if angle < ϵ and neighbor not visited then
            FUSE(startNormal, neighbor, listOfBorders)
        else
            listOfBorders ← border edge to neighbor
        end if
    end for
end function
```

This region growing algorithm produces a list of clusters with more or less co-planar triangles. To flatten the regions and remove artifacts originating from sensor noise all triangle vertices of a cluster are projected into a common regression plane with a RANSAC approach. All triangles in the cluster now share the same normal. This may cause that the normal criterion between two neighboring clusters is now fulfilled. To further fuse such adjacent co-planar regions, we re-start the region growing algorithm for each cluster after the regression planes were calculated. This optimization is continued, until no new merges occur. This iterative plane optimization is especially effective for reconstructions from noisy data, as Fig. 1 demonstrates.

The displayed reconstruction is based on point clouds collected with a servo-tilted SICK LMS 200 laser scanner in an office building. Each detected plane is rendered in a different color. In the first step, due to noise in the servo movement of the used low cost 3D scanner, the ceiling plane is fragmented into several planar regions. After several iterations, neighboring planes are fused until nearly the complete ceiling is represented by a single plane.

3.2 Vertex Optimization of Plane Intersections

Usually meshing algorithms have significant problems in reconstructing sharp features. After plane segmentation it is possible to calculate the exact intersections between intersecting planes. If two planes P_1 and P_2 intersect, they lie on a straight line $S = \mathbf{P_0} + t\mathbf{d}$ with direction $\mathbf{d}$. This direction has to be perpendicular to the normals $\mathbf{n}_1$ and $\mathbf{n}_2$ of the adjacent planes. Thus $\mathbf{d} = \mathbf{n}_1 \times \mathbf{n}_2$. $\mathbf{P_0}$ has to be in a plane perpendicular to P_1 and P_2, like the one that is spanned by $\mathbf{n}_1$ and $\mathbf{n}_2$, therefore it follows that $\mathbf{P_0} = k_1\mathbf{n}_1 + k_2\mathbf{n}_2$. The parameters k_1 and k_2 can be determined be solving the following equation system

$$\mathbf{n_1} \cdot (k_1\mathbf{n_1} + k_2\mathbf{n_2}) = d_1$$
$$\mathbf{n_2} \cdot (k_1\mathbf{n_1} + k_2\mathbf{n_2}) = d_2$$

which can be solved directly:

$$k_1 = \big(d_1(\mathbf{n_2} \cdot \mathbf{n_2}) - d_2(\mathbf{n_1} \cdot \mathbf{n_2})\big)/t$$
$$k_2 = \big(d_2(\mathbf{n_1} \cdot \mathbf{n_1}) - d_1(\mathbf{n_2} \cdot \mathbf{n_2})\big)/t$$

with $t = (\mathbf{n_1} \cdot \mathbf{n_1})(\mathbf{n_2} \cdot \mathbf{n_2}) - (\mathbf{n_1} \cdot \mathbf{n_2})^2$. To optimize the representation of the intersections, all vertices of a plane's contour that are near that line are projected onto it. An example for such an projection is given in Fig. 2

3.3 Contour Extraction and Re-Triangulation

After the contours and intersections of the planes are extracted, edges that point into the same direction are fused to reduce the number of vertices needed to represent the contour. For example, in the initial Marching Cubes reconstructions the length of an edge is limited w.r.t. the resolution of the voxel grid that was used to compute the mesh, as can be witnessed in Fig. 2, where the edges lying on the optimized line are still sub-divided according to the used voxel size. To fuse such edges, we apply the well known Douglas-Peucker-Algorithm (Douglas and Peucker, 1973).

This algorithm generates the optimized contour recursively. Initially the first and last vertex of the contour are connected. Then the vertex with the largest distance to this line within an error tolerance is detected. This vertex sub-divides the line into two segments that then

Download English Version:

https://daneshyari.com/en/article/708733

Download Persian Version:

https://daneshyari.com/article/708733

Daneshyari.com