Incremental Texture Mapping for Autonomous Driving

Miguel Oliveira *[a], Vítor Santos[b], Angel D. Sappa[c,d], Paulo Dias[b], A. Paulo Moreira[a]

[a] *INESC TEC - Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência R. Dr. Roberto Frias, 465, 4200 Porto, Portugal*
[b] *IEETA - Instituto de Engenharia Electrónica e Informática de Aveiro, Universidade de Aveiro, Campus Universitário de Santiago, 3810-193 Aveiro, Portugal*
[c] *Facultad de Ingeniería en Electricidad y Computación, Escuela Superior Politécnica del Litoral, ESPOL, Campus Gustavo Galindo, Km 30.5 vía Perimetral, P.O. Box 09-01-5863, Guayaquil, Ecuador*
[d] *Computer Vision Center, Campus UAB, 08193 Bellaterra, Barcelona Spain*

Abstract

Autonomous vehicles have a large number of on-board sensors, not only for providing coverage all around the vehicle, but also to ensure multi-modality in the observation of the scene. Because of this, it is not trivial to come up with a single, unique representation that feeds from the data given by all these sensors. We propose an algorithm which is capable of mapping texture collected from vision based sensors onto a geometric description of the scenario constructed from data provided by 3D sensors. The algorithm uses a constrained Delaunay triangulation to produce a mesh which is updated using a specially devised sequence of operations. These enforce a partial configuration of the mesh that avoids bad quality textures and ensures that there are no gaps in the texture. Results show that this algorithm is capable of producing fine quality textures.

*Corresponding author
Email addresses: m.riem.oliveira@gmail.com (Miguel Oliveira *), vitor@ua.pt (Vítor Santos), asappa@cvc.uab.es (Angel D. Sappa), paulo.dias@ua.pt (Paulo Dias), amoreira@fe.up.pt (A. Paulo Moreira)

Keywords: Scene reconstruction, Autonomous Driving, Texture Mapping.

1. Introduction

Autonomous vehicles often have a very large number of sensors mounted on-board. This is due to the need to observe the environment all around the vehicle, but in addition to that because there is the need to observe the scene with sensors of different nature. Mainly, sensors are divided into two groups: range sensors and vision based sensors. Sensors of the first group provide 3D measurements of the scene. On the other hand, vision based sensors collect photometric information of the scene. Due to the large number of sensors on-board these vehicles, it is not trivial to combine data from these sensors into a unique representation of the scene. Given that these sensors provide a continuous stream of data over time, and that they are displaced by the movement of the vehicle, then it follows that the representation of the scene must also be dynamic, in the sense that it must evolve to represent novel information collected at later stages of the mission. Note that, given a continuous throughput of images, the most recent image is not necessarily the best image to be used for texture mapping. For example, if the vehicle is moving away from an object, a camera on the rear side of the vehicle will produce images with decreasing quality. Rather, what is required is an algorithm that produces a scene representation at the early stages of a mission (because this might be immediately required for other tasks such as navigation, planning, etc.), but the later on is also capable of evaluating newly acquired images to assess whether or not these images are better than the previously used for mapping the texture. We refer to this as incremental texture mapping.

In [1], an algorithm for creating and incrementally updating a geometrical rep-

resentation of the scenario is presented. This work was later extended in [2]. The approach is based on *Geometric Polygonal Primitives* (GPP), and is shown to be capable of providing an accurate geometric description of the scenario. It uses data from range sensors only, and the geometric description changes to accommodate novel sensor data. In this paper we use the results given by the approach described in [2]. This means that we consider that there is, at all times, a geometric description of the scenario which is constantly evolving.

In this paper, we focus on how the vision based sensors can be used to enrich the description of the scenario. In other words, we propose to use the images from the cameras on-board the vehicle to produce texture, which may be added to the 3D description of the environment. Note that, as in the case of the range sensors, the vision based sensors also produce a continuous stream of information which must be integrated in order to create a unique photometric description of the scenario. In this paper, we propose an approach which is capable of incrementally updating texture mapped onto GPPs. The following lines show an example in which the need for incremental texture mapping becomes clear.

For testing and evaluation purposes, we use a data-set from the *Massachusetts Institute of Technology* (MIT) Team, taken from their participation in the DARPA Urban Challenge [3]. A small 40 seconds sequence was cropped from the MIT data-set. This sequence is referred to as the MIT sequence, and five key locations (*A* through *E* where marked in the sequence (see [2] for details). The approach described in [2] produces a description of the geometric structure of the environment observed by the vehicle's sensors. This description is given in the form of *Geometric Polygonal Primitives* (GPP), i.e., a list of polygons. Note that, as pointed out in [2] the geometric description of the scene is dynamic, since it may change

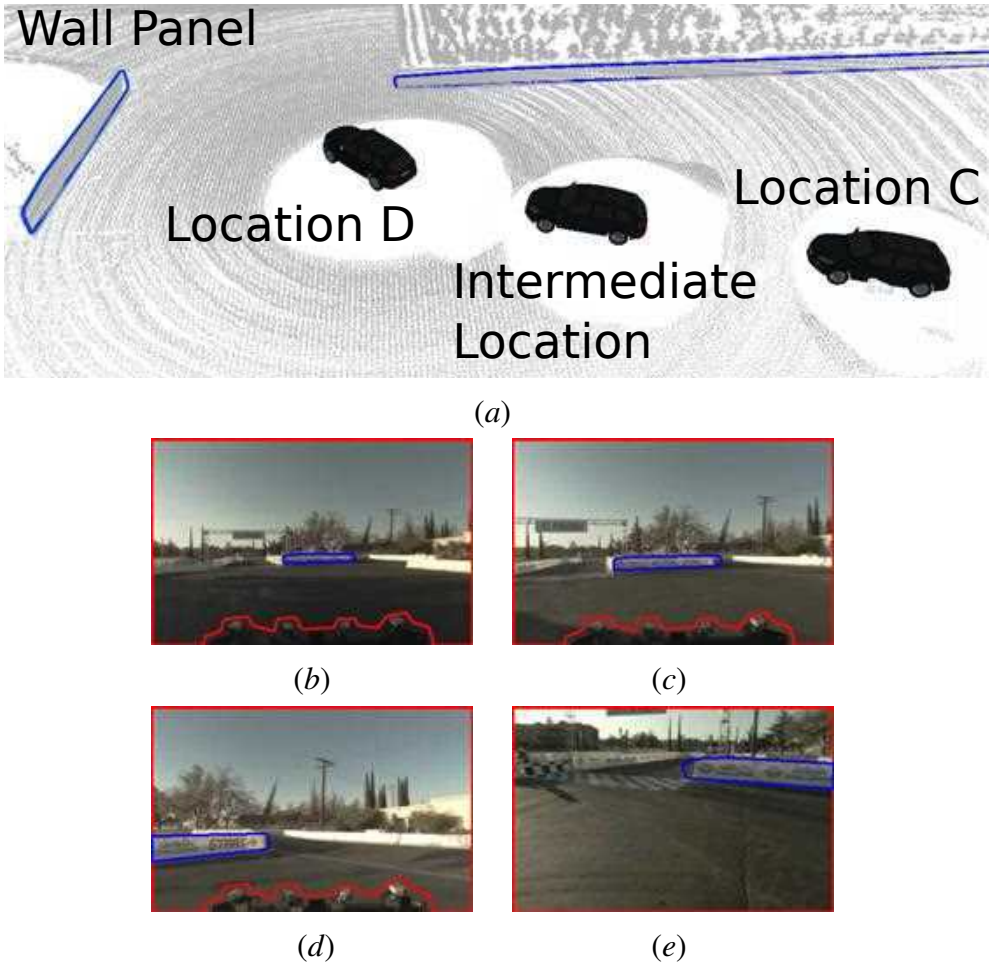


Figure 1: An example from the MIT data-set: three projections are collected over a period of time and mapped to a wall panel (GPP $k = 4$, in blue): (a) positions of the vehicle at the time each projection is collected; (b) image from front camera, at location C; (c), front camera, intermediate location; (d) front camera, location D; (e) left camera, location D.

whenever novel sensor information is collected.

An example is presented in Fig. 1 where the vehicle travels from location C to location D of the MIT sequence. Images are collected at three locations: location C at mission time t_0, location D at mission time t_2 and an intermediate location

between those two at mission time t_1 (Fig. 1 (*a*) shows the vehicle at each location). Consider a camera of index l, that produces an image which may virtually be projected to any GPP (i.e., to one of the polygons that constitute the geometric description of the scene), at any given mission time t. The term projection is defined as an image captured from a camera that can be used to map some texture to one of the polygons contained in the geometric description of the scene, and is denoted as $\mathbf{C}^{\{k,l,t\}}$. The data-set contains five color cameras (see [2, 4] for details). Without loss of generality, in this example only images from two cameras are used: front center ($l = 0$) and front left ($l = 3$), and only a single GPP (index $k = 4$) is employed, which corresponds to the wall panel in front of the vehicle (in blue, left side of Fig. 1 (*a*)). Note that, under the constraints defined above, i.e., $k = \{4\}$, $l = \{0,3\}$ and $t = \{t_0,t_1,t_2\}$, there are a total of six possible projections. However, two of these projections are empty, namely $\mathbf{C}^{\{k=4,l=3,t=t_0\}}$ and $\mathbf{C}^{\{k=4,l=3,t=t_1\}}$. This is because the left camera ($l = 3$) does not see the wall panel ($k = 4$) in the first two locations ($t = t_0$ and $t = t_1$). This can be observed in Fig. 1 (*a*), which shows that the vehicle turns right at location D, and only then the left side camera is pointed in the direction of the wall panel. The images from the remaining four projections are shown in Figs. 1 (*b*), (*c*), (*d*) and (*e*). As the vehicle approaches the wall panel, it collects images with higher resolution and better quality of that surface. Our goal is to study how a low resolution texture created when the vehicle was distant from the surface may evolve to a higher resolution texture once the vehicle comes closer to the panel. In other words, how can the texture of a surface be incrementally refined.

Note that we assume that an accurate localization is available at all times. In the case of the MIT dataset, localization is provided by an Applanix POS-LV

220 system [1], which includes a GPS, an inertial measurement unit and a wheel encoder. This is a very accurate system which publishes the 6 DOF pose of the vehicle at high frequencies (100Hz). Thus, it is possible to gather the pose of each of the onboard cameras at any point in time. Obviously, a less accurate ego motion estimation should influence the mapping of texture. However, a detailed analisys of the impact of other ego motion estimation systems is out of the scope of the current paper.

The remainder of the paper is organized as follows: related work is presented in section 2; the proposed approach is described in section 3 and, finally, results and conclusions are given in sections 4 and 5.

2. Related Work

Texture mapping is a technique for mapping a 2D image onto a 3D surface by transforming color data so that it conforms to the surface plot. It allows the application of texture such as tiles or wood grain, to a surface without performing the geometric modeling necessary to create a surface with these features, or, in other words, without computing the projection of every pixel in the image onto the surface. The color data can also be any image, such as a picture taken by a camera. Texture mapping is performed over convex polygons, most commonly on triangles. Let $\mathbf{X}_1$, $\mathbf{X}_2$ be the coordinates of the vertices 1 and 2 in 3D space. The coordinates $\mathbf{u}_1$, $\mathbf{u}_2$ of the pixels that correspond to those vertices in the image plane can be obtained using direct projection:

$$\mathbf{u}_i = \texttt{projection}(\mathbf{X}_i), \quad \forall i \in \{1,2\}. \tag{1}$$

[1] http://www.applanix.com/products/land/pos-lv.html

Let α be a parameter $0 < \alpha < 1$, that indicates how a given vertex is positioned along the $\overline{\mathbf{X}_1\mathbf{X}_2}$ line segment. Texture mapping interpolates the color value for any vertices along the line segment as follows:

$$\mathbf{u}_\alpha = (1-\alpha)\cdot\mathbf{u}_0 + \alpha\cdot\mathbf{u}_1, \tag{2}$$

which is of course a linear interpolation. When this kind of linear interpolation is used, the texture mapping is referred to as affine texture mapping. A linear interpolation works fine when the image plane and the projection plane are parallel. However, when this does not occur, the projection shows some artifacts that derive from the assumption that a linear interpolation can be used. This is a well documented problem, and is discussed in several works [5, 6]. The solution to this problem is called view dependent texture mapping, and it consists of making texture mapping account for the position of the vertexes in 3D space, rather than simply interpolating a 2D triangle. This achieves the correct visual effect, but it is slower to calculate. Instead of interpolating the texture coordinates directly, the coordinates are divided by their depth (relative to the viewer), and the reciprocal of the depth value is also interpolated and used to recover the perspective corrected coordinate. This correction operates so that in parts of the polygon that are closer to the viewer, the difference from pixel to pixel between texture coordinates is smaller (stretching the texture wider), and in parts that are farther away this difference is larger (compressing the texture). View dependent texture mapping can be formulated as:

$$\mathbf{u}_\alpha = \frac{(1-\alpha)\cdot\frac{\mathbf{u}_0}{w_0} + \alpha\cdot\frac{\mathbf{u}_1}{w_1}}{(1-\alpha)\cdot\frac{1}{w_0} + \alpha\cdot\frac{1}{w_1}}. \tag{3}$$

The solution proposed in eq. (3) is capable of producing accurate mapping for texture. View dependent texture mapping is significantly slower when compared

to affine texture mapping.

Since triangles are the atomic entities of texture mapping, triangulation methodologies are an important part of the process. In this scope, *Data Dependent Triangulation* (DDT) algorithms are of particular interest since they can produce triangulated meshes which are ideal for texture mapping. The goal of a DDT is, on the one hand, to obtain the best approximation possible, and on the other to reduce the number of triangles and in turn the memory load. Consequently, the number of vertices should be kept as small as possible to speed up processing and reduce memory load. The two variables that should be tuned to achieve a good approximation are then the position of the vertices and the connections between them. Even if we decide to fix the number of triangles and vertices, the possible combinations of the connections between vertices are usually very large. Hence, an exhaustive search of all possible combinations is not possible. Also, no assumptions should be made on the optimal shape or size of the triangles. One might tend to assume long, thin triangles are not adequate but in fact that depends on the nature of the image [7]. If the image contains high gradient long feature such as poles or trees, such triangles could be well suited to represent these regions. DDT algorithms can be divided into refinement, decimation, or modification approaches. In refinement approaches, the starting point for the algorithm is a very coarse triangular mesh that is then refined. The mesh is refined by inserting new vertices. Since the number of possible positions where vertices can be inserted is very high, authors make use of heuristics to limit the number of options. The greedy refinement algorithm proposed in [8] works by inserting vertices into a triangulated mesh. In every step, a new vertex is inserted at the position of the largest distance between the approximation and the data provided in the image.

In [9], the choice of which are the triangles to decimate is based on the high curvature of the data, and the positions where new vertices are to be inserted are locations with high proximity to the data. These methods have the drawback of tending to a local optima. Decimation approaches are the opposite of refinement meshes. The algorithms start from a very fine mesh and try to remove vertices and collapse triangles as they iterate. In [10] the initial triangulation is a full triangulation where each pixel is a vertex in the mesh. The algorithm then decimates the mesh by collapsing one of the edges of the mesh. The edge to collapse is the one that implicates less increase in the approximation error. Similar approaches were proposed in [11] and [12]. Finally, modification strategies start from a random arbitrary mesh and try to improve it by performing modification operations. These modification operations usually are edge swaps and the number of vertices in the initial mesh remains the same. It is the case of the algorithms proposed in [13] and [14]. Both propose different criteria for the selection of which are the edges that should be swapped.

Given a triangulated mesh of a surface and an image registered to that surface, it is possible to map the texture from the image onto the surface using classic texture mapping approaches. Figure 2 shows the triangulated meshes and textures produced using classical texture mapping, for each of the four projections displayed in Fig. 1. These mappings are computed independently for each location. As expected, textures that derive from projections taken closer to the surface (e.g., Fig. 2 (*c*) and (*d*)) have better quality when compared to projections taken far away from the surface (e.g., Fig. 2 (*a*) and (*b*)).

The question is how to create an unique triangulated mesh and texture and how it should be updated with local triangulated meshes from novel projections.

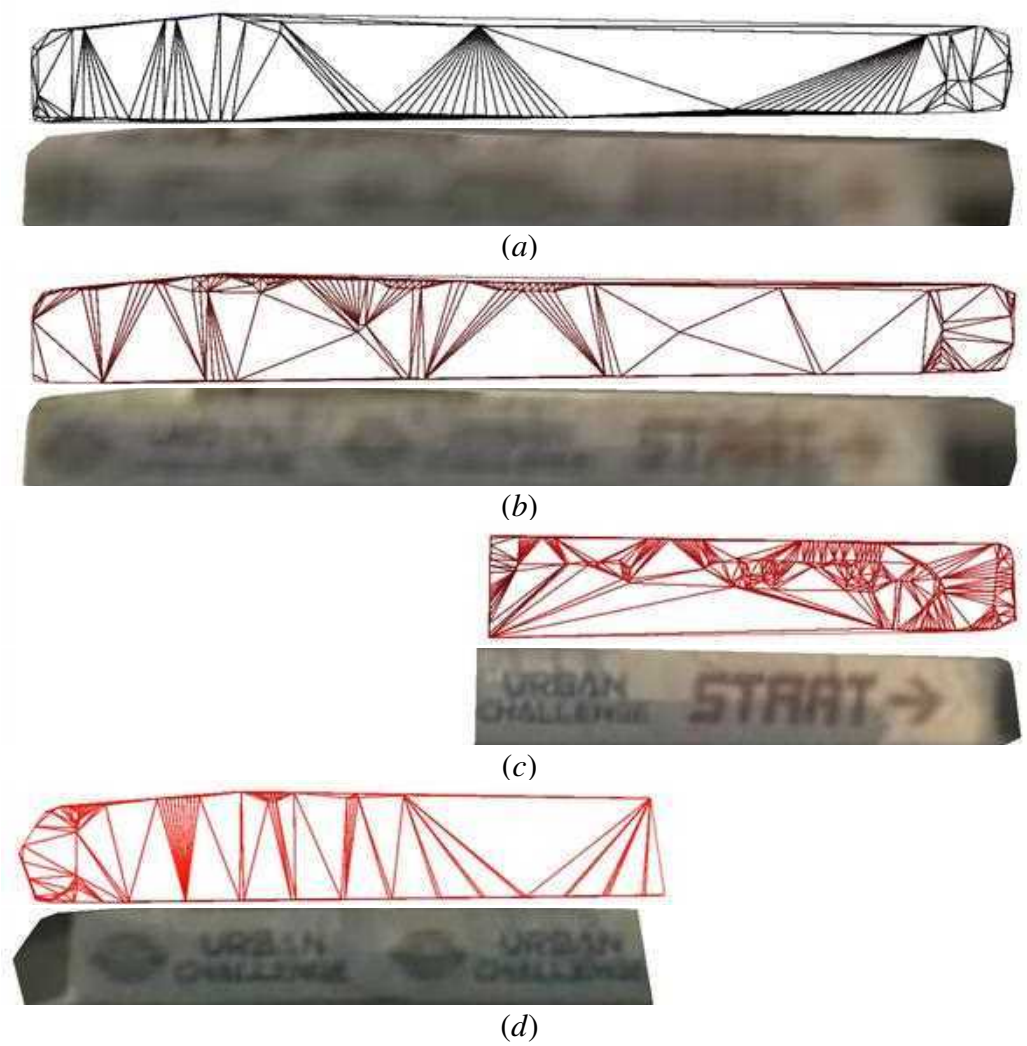


Figure 2: Triangulated meshes (*top*) and textures (*bottom*) created separately for each of the projections shown in the example of Fig. 1: (*a*) front camera, location *C*; (*b*) front camera, intermediate location; (*c*) front camera, location *D*; (*d*) left camera, location *D*.

Lets assume that there is a way of assessing the quality of each projection and in particular of each triangle in each mesh, so that it is possible to rank the triangles with respect to their quality. At first sight, several strategies can be used to fuse the textures, namely: **(1)** average the textures produced by local meshes; **(2)** insert

vertices from new triangles into the existing mesh and re triangulate, provided that these triangles yield better quality; **(3)** remove the triangles of the existing mesh that overlap the triangle (of better quality) to be inserted, and then insert the triangle.

Option **(1)** consists of averaging the textures provided by each local mesh. This could be achieved by setting the alpha channel of all local meshes so that they average out. Additionally, the average could be weighted by the quality of the triangles, although this was not tested in this work. The primitive would have several layers, each with a given local triangulated mesh belonging to each projection. Figure 3 (*a*) shows the results obtained using this strategy. Visually, results are not appealing. Another disadvantage concerns the need to store all local meshes, which is highly inefficient in terms of memory. As seen in Fig. 2, there are textures with much better quality than others. To average good textures with bad textures does not seem to make sense.

Option **(2)** proposes to address the problem by considering the vertices of the meshes only (rather than the triangles). Each vertex in the new mesh is added to the current mesh. This results in a super mesh containing all the vertices of the two previous meshes in which the triangles (the configuration of the mesh) are defined arbitrarily. The idea is to fuse using an additive strategy. Figure 3 (*b*) shows the results obtained using this approach. Again, results are not visually appealing.

In option (**3**), an alternative to averaging is considered: a winner takes all strategy. The idea is to select, for each region in the surface, a single triangle which will provide the texture. This selection can be done using the quality of each triangle. Note that, whenever a triangle to be inserted overlaps some triangles in the existing mesh, these triangles must first be removed so that the mesh preserves its

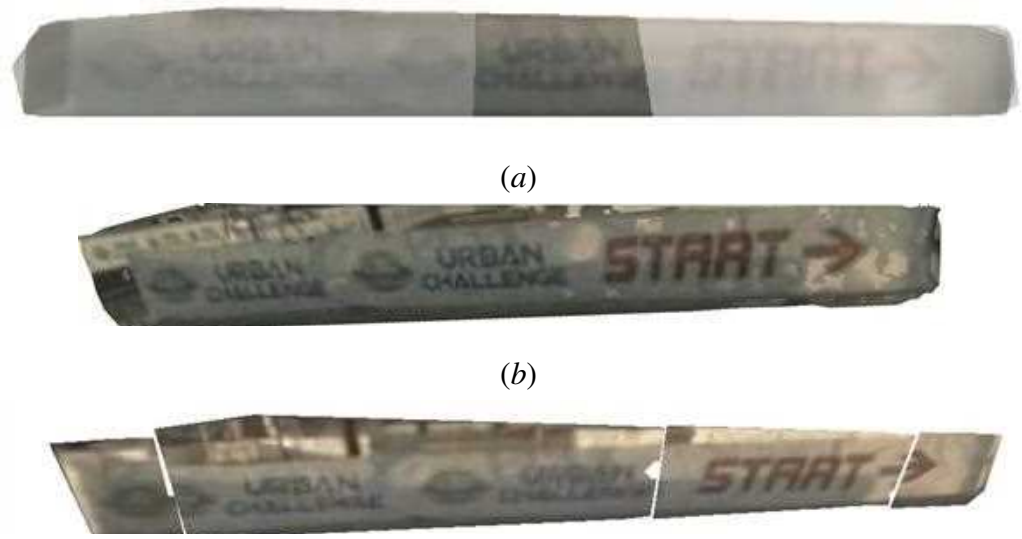


Figure 3: Textures obtained using different fusion strategies: (*a*) option (**1**), average textures from local meshes; (*b*) option (**2**), insert vertices from all local meshes; (*c*) option (**3**), insert triangles of better quality, removing overlapping triangles.

configuration and thus the quality of the texture mapping. Results obtained from this method are shown in Fig. 3 (*c*). Textures are visually appealing. Artifacts present in the average and the additive strategies are not visible. There is one problem however: removed triangles often overlap triangles to be inserted in just a portion of their area (partial overlap). When deleted, these triangles leave empty spaces where no texture is defined. This is visible in Fig. 3 (*c*).

None of the strategies discussed provides textures of sufficient quality. Thus the problem of incrementally updating the texture is not trivial. In the following sections, an approach is presented which is capable of generating higher quality textures.

3. Proposed Approach

Figure 4 shows a diagram which describes the functioning of the system. The following sections will describe these components in detail. First, a one-shot texture mapping based in DDT is presented in section 3.1. Here, we propose an algorithm based on the extraction of edges in the image and the construction of a constrained Delaunay triangulation, which is operated as a DDT and is very efficient. Then, the incremental texture mapping approach is presented in section 3.2. In this case, we propose a sequence of atomic operations to conduct the insertion of a new triangle in a triangulated mesh, which minimizes the changes in the configuration of the mesh.

3.1. One-shot texture mapping using Data Dependent Triangulation

In this paper, we propose an alternative solution to view dependent texture mapping. One reason for this is that the objective of this work is to develop a mechanism for mapping texture from images onto GPP (see [2]). Those geometric primitives consist of polygons, instead of the traditional triangles. The mapping of photometric properties can be performed by mapping triangles in image space to 3D space. These procedures are executed in *Graphical Processing Units* (GPUs), and programmed using OpenGL [15], Direct3D [16] or other graphics libraries. These libraries also have the functionalities of mapping convex polygons, but in fact these are mere high level functions that decompose the polygons in an arbitrary way into sets of triangles and then map texture onto those triangles. We argue that, if we control the process of triangulation in such a way that the edges in the images used in the projection are aligned with the edges of the triangles in 3D space, the distortion produced by linear texture mapping is not visible, and

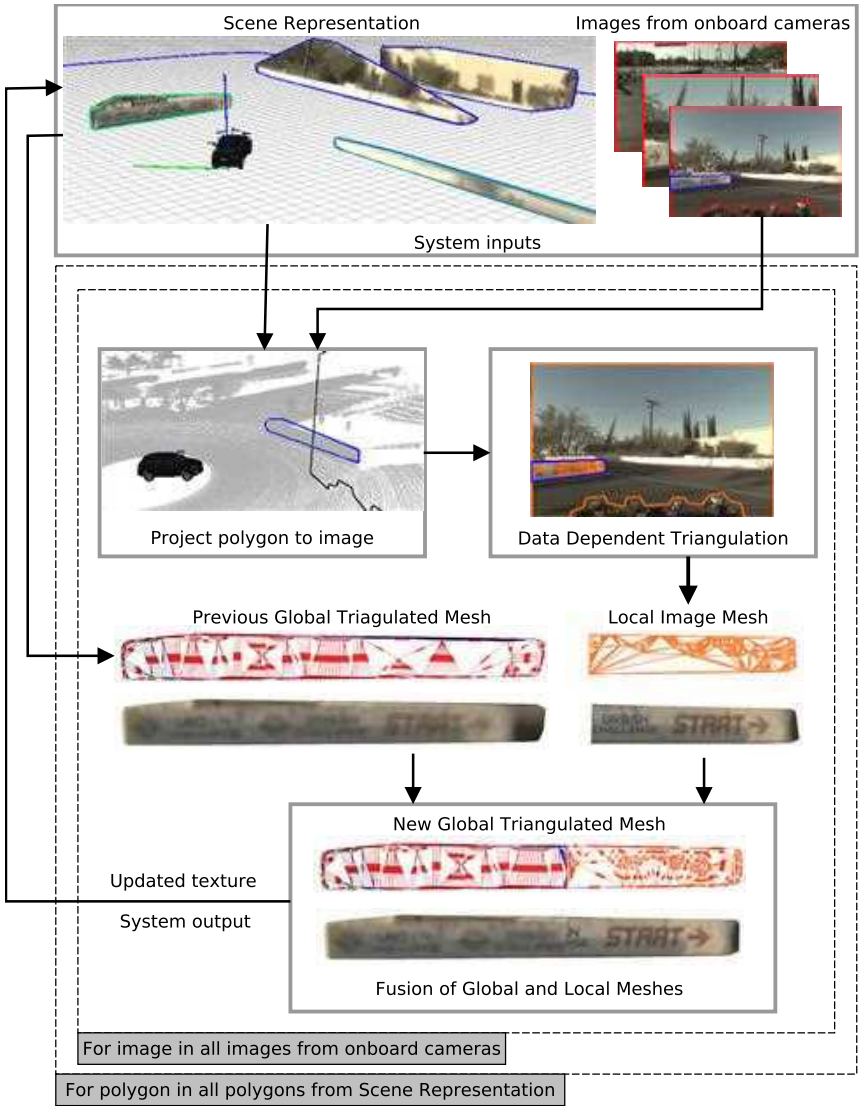


Figure 4: A diagram showing the main components of the proposed system.

thus, linear texture mapping may be used instead of view dependent triangulation, which is much slower. In other words, if the triangles are especially defined so that their faces represent smooth regions with constant color then, a linear texture mapping over these could in fact provide accurate projections. This procedure of

creating a triangulated mesh which accommodates some input data is called DDT [17], and the mapping of images using this technique will be referred to as DDT mapping as opposed to texture mapping. Unlike in standard texture mapping approaches, where the triangulation is executed in the 3D space, DDT triangulation operates in the image space, and only after those 2D triangles are mapped onto the 3D space.

Although there are many approaches in the literature to the data dependent triangulation problem, most of them are focused on the fact that such a triangulated mesh is capable of producing very good data compression ratios with respect to the real image, while still maintaining low approximation errors. Real time performance of the algorithms has seldom been debated, with authors reporting processing times of over three seconds for 512×512 images. The exception was the study conducted in [18], where DDT was parallelized, resulting in a significant speed up. We propose a simple procedure similar to [12]: edges are detected using a Hough lines detector [19] extended to obtain a description of line segments instead of lines (e.g., see [20] [21]). The triangulation is a Delaunay triangulation [22]: let the image be described by M line segments with starting points $\mathtt{s}_m$ and endpoints $\mathtt{e}_m$, where each detected line segment is defined as $\overline{\mathtt{s}_m\mathtt{e}_m}$. The Delaunay triangulation (**Delaunay**) receives the starting and endpoints as input to define the vertices of the triangulated mesh:

$$\mathtt{t} = \textbf{Delaunay}\big(\{\mathtt{s}_0, \mathtt{e}_0, \mathtt{s}_1, \mathtt{e}_1, \cdots, \mathtt{s}_{M-1}, \mathtt{e}_{M-1}\}\big), \tag{4}$$

where $\mathtt{t}$ is the resulting triangulated mesh. A constrained Delaunay triangulation is a generalization of the Delaunay triangulation where line segments may be imposed as belonging to the triangulated mesh (initially proposed by [23] for 2D spaces, later generalized to N dimensional spaces by [24]). A constrained Delau-

nay triangulation (**cDelaunay**) requires two inputs, a list of points and a list of line segments (also called constraints):

$$\mathtt{t} = \textbf{cDelaunay}\big(\{\mathtt{s}_0, \mathtt{e}_0, \cdots, \mathtt{s}_{M-1}, \mathtt{e}_{M-1}\}, \{\overline{\mathtt{s}_0\mathtt{e}_0}, \cdots, \overline{\mathtt{s}_{M-1}\mathtt{e}_{M-1}}\}\big). \qquad (5)$$

In brief, what we propose is a technique in which a constrained Delaunay triangulation is executed on the image space, having as input the line segments given by a line segment detection algorithm based on hough lines.

3.2. Incremental texture mapping

Section 3.1 described how a constrained Delaunay triangulation may be used to produce a data dependent triangulated mesh that conforms with edges previously detected in the image. Note that this is a one-camera, one-shot approach, since it does not consider how to map more than one image. In reality, there is always a large set of images available to use for texture mapping, either from multiple cameras or from a unique camera at different times. This section addresses this problem of merging multiple projections into a single representation. As described in section 3.1, a DDT triangulation is executed for each image used in a projection. Thus, there will be a triangulated mesh (a list of triangles) for each image (for each projection), to which we refer as local triangulated mesh. Local triangulated meshes for the example of Fig. 1 are shown in Fig. 2.

Let $\mathtt{M}$ be the global triangulated mesh, defined in $\mathbb{R}^2$, so that only one global mesh exists per each primitive. This global mesh should be updated when new projections are collected or, in other words, when novel local meshes are received, i.e. it should contain the result of the fusion of the several textures. A local triangulated mesh from projection index $j = \{k, l, t\}$ (i.e., form a given combination of k, l, t) is denoted as $\mathtt{T}^j$. Local triangulated meshes contain T^j number of triangles.

Individual triangles are denoted as $\mathtt{T}_i^j, \forall i \in \{0,1,\cdots,T^{\{k,l,t\}}\}$, when indicating the *ith* triangle of the local mesh j, or as $\mathtt{T}^j_{\{\mathtt{v}_1,\mathtt{v}_2,\mathtt{v}_3\}}$, in the case the vertices $\mathtt{v}_1$, $\mathtt{v}_2$ and $\mathtt{v}_3$ are specified. Likewise, triangles in the global mesh are notated as $\mathtt{M}_n, \forall n \in \{0,1,\cdots,N\}$, where N is the number of triangles in the global mesh. When the vertices of the triangles are specified, then the notation $\mathtt{M}_{\{\mathtt{V}_1,\mathtt{V}_2,\mathtt{V}_3\}}$ is used.

To continuously fuse local triangulated meshes from new projections onto the global mesh, we propose a mechanism which iterates all the triangles in the local projection mesh and decides whether they should be inserted in the global mesh by computing the benefit of this operation to the overall quality of the global mesh. At iteration i, triangle $\mathtt{T}_i^j$ from the local projection mesh is referred to as candidate triangle. First, the algorithm assesses if there is overlap between $(\mathtt{T}_i^j)$ and any of the existing triangles in the global mesh $\mathtt{M}_n, \forall n \in \{0,1,\cdots,N\}$. Let $\mathbf{intr}(A,B)$ be a function that tests intersection between triangles A and B. The test can be written as:

$$do\_intersect = \mathbf{intr}\left(\mathtt{T}^j_{\{\mathtt{V}_1,\mathtt{V}_2,\mathtt{V}_3\}}, \mathtt{M}_n\right), \forall n \in \{0,1,\cdots,N\}, \tag{6}$$

where N corresponds to the total number of triangles in the global mesh $\mathtt{M}$. The intersection of two triangles can result in an empty set, whenever there is no intersection, in a point, a line segment, or a polygon. There are several approaches to triangle triangle intersection tests, that provide fast and efficient algorithms [25] [26] [27]. Note that there is a distinction between overlap and intersection: what must be assessed is whether or not an insertion of the candidate triangle onto the global mesh will change its configuration. Thus, an overlap test is not the same as an intersection test, since there are some cases where the triangles do intersect but the mesh configuration is not altered. The overlap test is based on a set

of rules that analyse the return of the intersection function (**intr**, implementation from [28]), between candidate triangle $\mathtt{T}_i^j$ and global mesh triangle $\mathtt{M}_{\{\mathtt{V}_1,\mathtt{V}_2,\mathtt{V}_3\}}$. It returns *yes* if the triangles overlap or *no* otherwise. The algorithm is detailed in eqs. (7), (8) and (9):

$$\begin{cases} no & \Leftarrow \mathbf{intr}\left(\mathtt{T}_i^j, \mathtt{M}_{\{\mathtt{V}_1,\mathtt{V}_2,\mathtt{V}_3\}}\right) = \varnothing \\ yes & \Leftarrow \mathbf{intr}\left(\mathtt{T}_i^j, \mathtt{M}_{\{\mathtt{V}_1,\mathtt{V}_2,\mathtt{V}_3\}}\right) = \text{list of polygons} \\ Goto(8) & \Leftarrow \mathbf{intr}\left(\mathtt{T}_i^j, \mathtt{M}_{\{\mathtt{V}_1,\mathtt{V}_2,\mathtt{V}_3\}}\right) = \text{points } \mathbf{X} \\ Goto(9) & \Leftarrow \mathbf{intr}\left(\mathtt{T}_i^j, \mathtt{M}_{\{\mathtt{V}_1,\mathtt{V}_2,\mathtt{V}_3\}}\right) = \text{line segment } \mathbf{L} \end{cases} \tag{7}$$

where $\mathbf{X} = \{\mathtt{X}_0, \mathtt{X}_1, \cdots, \mathtt{X}_N\}$ and $\mathbf{L} = \{\overline{\mathtt{S}_0\mathtt{E}_0}, \cdots, \overline{\mathtt{S}_N\mathtt{E}_N}\}$;

$$\begin{cases} no & \Leftarrow \exists_{\mathtt{V}_g:\mathtt{V}_g=\mathtt{X}_o}, \forall_{\mathtt{V}_g\in\{\mathtt{V}_1,\mathtt{V}_2,\mathtt{V}_3\}}, \forall_{o\in\{0,1,\cdots,N\}} \\ yes & \Leftarrow otherwise \end{cases} \tag{8}$$

$$\begin{cases} no & \Leftarrow \exists_{\mathtt{V}_g:\mathtt{V}_g=\mathtt{S}_o} \wedge \exists_{\mathtt{V}_h:\mathtt{V}_h=\mathtt{E}_o}, \forall_{\mathtt{V}_g,\mathtt{V}_h\in\{\mathtt{V}_1,\mathtt{V}_2,\mathtt{V}_3\}}, \forall_{o\in\{0,1,\cdots,N\}} \\ yes & \Leftarrow otherwise \end{cases} \tag{9}$$

Figure 5 (*a*) shows a global primitive mesh $\mathtt{M}$ that contains a single triangle (in blue), and a candidate triangle (in red). In each case, the geometries returned by the intersection function are as follows: an empty set (*d*), points (*a*) and (*e*), line segments (*b*) and (*f*), and polygons (*c*). If we consider the cases where the insertion of the candidate triangle (in red) does not change the configuration of the already existing global mesh (in this case, the initial global mesh is composed of a single triangle, in blue), we can say that in case (*a*), (*b*) and (*c*) the mesh would be altered, and that, in cases (*d*), (*e*) and (*f*) the mesh would remain unaltered. The overlap test returns a list $\mathbf{L}$ of indices of triangles from the global mesh which

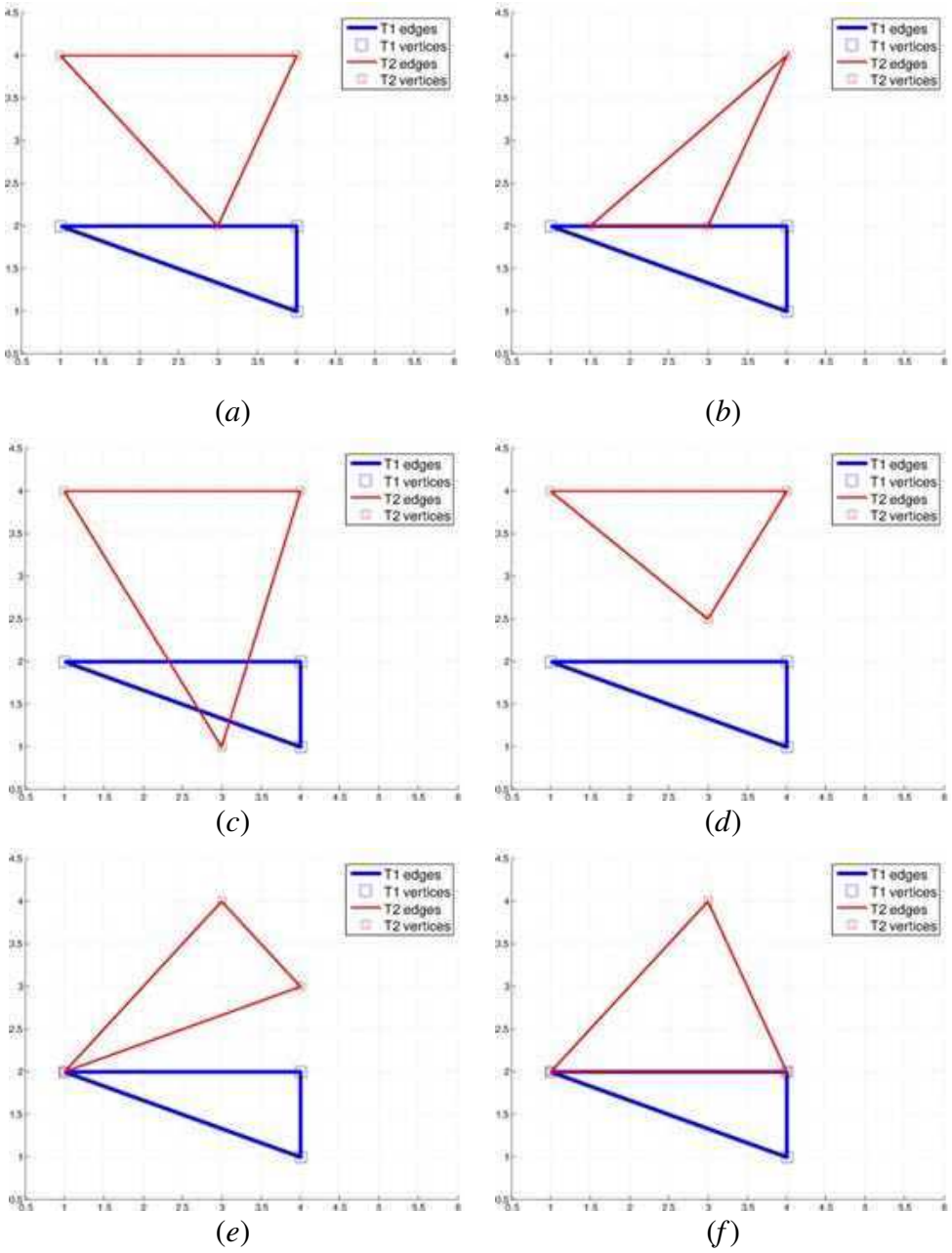


Figure 5: Triangle overlap test: (*a*) intersection returns points, overlap true; (*b*) intersection returns line segments, overlap true; (*c*) intersection returns polygons, overlap true; (*d*) intersection returns empty, overlap false; (*e*) intersection returns points, overlap false; (*f*) intersection returns line segments, overlap false;

overlap with the candidate triangle. From this, the benefit of inserting the candidate triangle in the global mesh is assessed. In this context, benefit is defined as an improvement in the quality of texture and estimated as follows:

$$\begin{cases} \textit{beneficial} \Leftarrow \mathbf{L} \textit{ is empty} \\ \textit{otherwise} \begin{cases} \textit{beneficial}, & \Leftarrow \mathbf{q}(\mathtt{T}_i^j) > \alpha \cdot max\big(\mathbf{q}(\mathtt{M}_g)\big),\ \forall g \in \mathbf{L} \\ \textit{not beneficial}, & \Leftarrow \textit{otherwise} \end{cases} \end{cases} \tag{10}$$

where $\alpha \geq 1$ is a user defined cost parameter, which defines how much better the quality of candidate triangle must be to any other triangle it overlaps, in order for the insertion to be considered beneficial, and $\mathbf{q}(\cdot)$ is an arbitrary function that returns the estimated quality of each triangle. In this work we define quality as directly proportional to the resolution of the texture. An image provides a texture of higher resolution when it is closer to the GPP. In addition to this, the focal distance of the camera should be taken into account. Thus, the quality of a triangle $\mathtt{T}^{j=\{k,l,t\}}$ is proposed as follows:

$$\mathbf{q}\left(\mathtt{T}^{\{k,l,t\}}\right) = \frac{f_l}{D_{\{k,l,t\}}}, \tag{11}$$

where f_l is the focal distance of camera index l and $D_{\{k,l,t\}}$ is the distance between the camera l and the GPP k computed at time t.

When the insertion of a candidate triangle is considered to be beneficial, the next step is to execute the insertion in the global mesh. The global primitive mesh is built as a constrained Delaunay triangulation. Hence, a description of the mesh contains a set of vertices, edges and constraints (implementation from [22] is used). After insertion, the candidate triangle should be preserved on the updated mesh (since it had a larger quality than any global mesh triangle it overlapped).

The configuration of all other non overlapped global mesh triangles should also be preserved. In order to comply with those objectives, the insertion of a candidate triangle is composed of a set of atomic operations executed in sequence. Figure 6 will be used to demonstrate why the proposed set of operations is required, by comparing it to other possibilities. Figure 6 (*a*) shows the initial situation: an existing mesh in blue must be altered by the insertion of a candidate triangle in red. The blue triangle has all three edges constrained (blue squares, in Fig. 6 (*a*)). Let **insert**$(\mathtt{V},\mathtt{M})$ be a function that inserts vertex $\mathtt{V}$ into mesh $\mathtt{M}$.

$$\mathtt{M}^* = \textbf{insert}\big(\mathtt{V},\mathtt{M}\big), \forall \mathtt{V} \in \{\mathtt{V}_a,\mathtt{V}_b,\mathtt{V}_c\}, \tag{12}$$

where $\mathtt{M}^*$ is the updated mesh. Figure 6 (*b*) shows the updated mesh after the insertion of the three vertices, indices $4,5$ and 6 (see vertices indices in the Fig. 6). The updated mesh does not preserve the configuration of the candidate triangle. In other words, there is no triangle with vertices $4,5,6$ in the updated mesh. The expression that asserts if the configuration of the candidate triangle is preserved can be stated as follows:

$$\begin{cases} \textit{preserve } \mathtt{T}^{j}_{\{\mathtt{V}_a,\mathtt{V}_b,\mathtt{V}_c\}}, \textit{if}\left(\exists \mathtt{M}^*_{\{\mathtt{V}_d,\mathtt{V}_e,\mathtt{V}_f\}} \in \mathtt{M}^*\right) : \mathtt{V}_d = \mathtt{V}_a \wedge \mathtt{V}_e = \mathtt{V}_b \wedge \mathtt{V}_f = \mathtt{V}_c \\ \textit{do not preserve } \mathtt{T}^{j}_{\{\mathtt{V}_a,\mathtt{V}_b,\mathtt{V}_c\}}, \textit{otherwise} \end{cases} \tag{13}$$

One of the reasons why the simple insertion of the vertices does not work is that the existing mesh had some constrained edges. After the mesh is updated, these constraints continue to exist (see squares on edges 1-2, 2-3, and 1-3 in Fig. 6 (*b*)). The configuration of the candidate triangle is not kept because no constraints over the edges of that triangle are set. Hence, the second alternative is to execute an additional operation on top of the insertion of vertices $\mathtt{V}_a$, $\mathtt{V}_b$ and

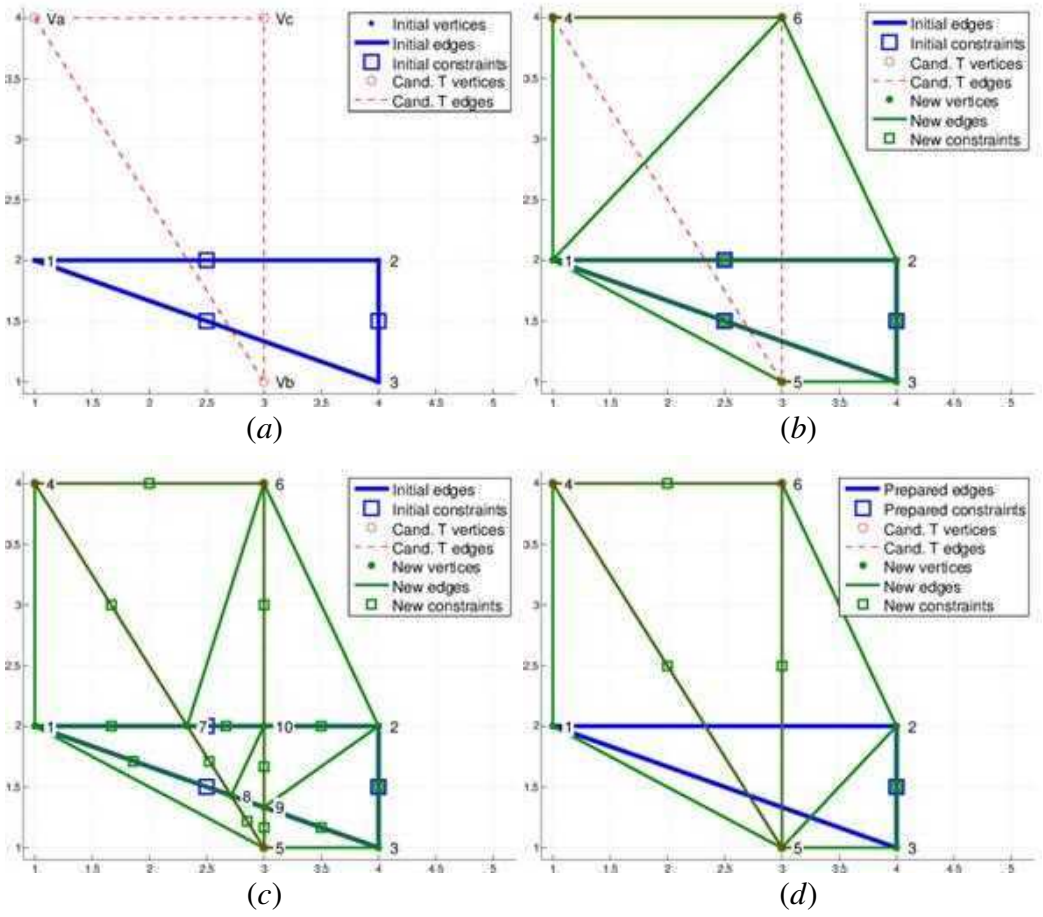


Figure 6: The insertion operation: (*a*) candidate triangle and initial mesh; (*b*) insertion of candidate triangle's vertices; (*c*) insertion of the candidate triangle's vertices and constraints; (*d*) preparation of the mesh followed by the insertion of the candidate triangle's vertices and constraints.

$\mathtt{V}_c$. Let **add_constraint**$(\mathtt{e},\mathtt{M})$ be a function that adds a constraint on edge $\mathtt{e}$. The operation can be expressed as follows:

$$\mathtt{M}^* = \textbf{add\_constraint}\big(\mathtt{e},\mathtt{M}\big), \forall\, \mathtt{e} \in \big\{\mathtt{V}_a\text{-}\mathtt{V}_b, \mathtt{V}_b\text{-}\mathtt{V}_c, \mathtt{V}_a\text{-}\mathtt{V}_c\big\}. \tag{14}$$

where V_a-V_b denotes the edge defined between vertices V_a and V_b.

Figure 6 (*c*) shows the updated mesh after this procedure is executed. Also in

this case the configuration of the candidate triangle is not preserved. The reason is that there are conflicting constraints inserted in the mesh. For example, initially, the global mesh had a constraint over edge 1-2 (see indices in Fig. 6). At the same time the constraint $\mathtt{V}_a$-$\mathtt{V}_b$ is inserted into the mesh. Since these two constraints intersect, a new vertex is created at the intersection point (vertex 7). Since a vertex is created at the intersection of the two initial constrained edges, four new edges are created (edges 4-7, 7-8, 1-7 and 7-10). All of these edges are constrained. From Fig. 6 (*c*), one can see that the overall result of this approach is that neither the candidate triangle nor the existing mesh is preserved. The reason is that contradictory (intersecting) constraints are inserted in the mesh. The solution is to remove the constraints from edges in the global mesh that intersect edges from the candidate triangle, prior to inserting the vertices and constraints of the candidate triangle. Let $\mathbf{E} = \{\mathtt{e}_0, \mathtt{e}_1, \cdots, \mathtt{e}_N, \}$ be the list of the global mesh constrained edges that intersect any of the candidate triangle's edges, and **remove_constraint**$(\mathtt{e}, \mathtt{M})$ a function that removes the constraint from edge $\mathtt{e}$ in the mesh $\mathtt{M}$. The prepared mesh $\mathtt{M}'$ is obtained as follows:

$$\mathtt{M}' = \textbf{remove\_constraint}\big(\mathtt{e}, \mathtt{M}\big), \forall\, \mathtt{e} \in \mathbf{E}, \tag{15}$$

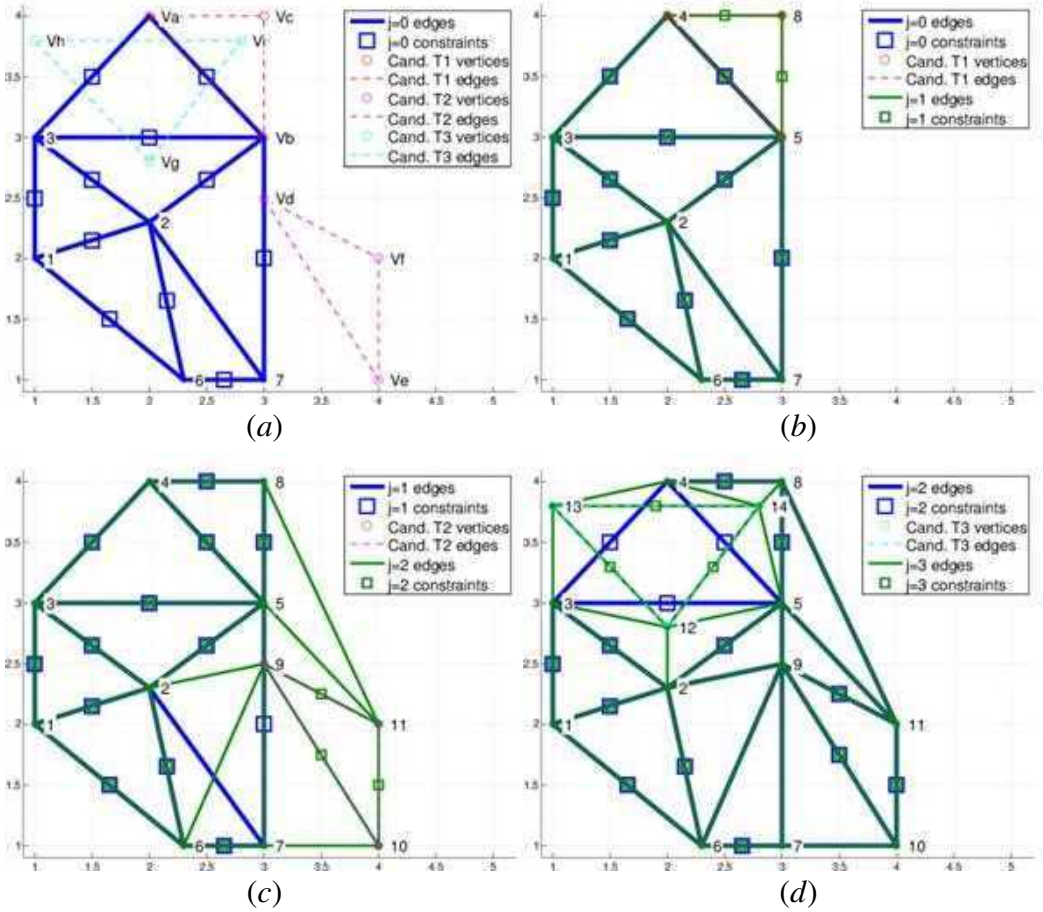
and after this, the operations described in eqs. (12) and (14) are executed. Fig. 6 (*d*) shows the results of this approach. The mesh preparation stage detected the following intersections (indices in Figs. 6 (*a*) and (*d*)): $\mathtt{V}_a$-$\mathtt{V}_b$ intersects with $\mathtt{V}_1$-$\mathtt{V}_2$, $\mathtt{V}_a$-$\mathtt{V}_b$ intersects with $\mathtt{V}_1$-$\mathtt{V}_3$, $\mathtt{V}_b$-$\mathtt{V}_c$ intersects with $\mathtt{V}_1$-$\mathtt{V}_2$ and $\mathtt{V}_b$-$\mathtt{V}_c$ intersects with $\mathtt{V}_1$-$\mathtt{V}_3$. As a result, the constraints of edges $\mathtt{V}_1$-$\mathtt{V}_2$ and $\mathtt{V}_2$-$\mathtt{V}_3$ are removed. Note that in Fig. 6 (*d*), the prepared mesh (not the initial global mesh) is shown in blue, and those constraints no longer appear. More important, the candidate triangle's configuration is preserved (triangle 4-5-6). In this particular

case, the initial configuration of the mesh is lost, since there was overlap between the candidate triangle and the initial mesh triangle.

We now show an example of continuous update of the global mesh: three new projections ($\mathbf{C}^{j=1}$, $\mathbf{C}^{j=2}$, $\mathbf{C}^{j=3}$) are available to update to the initial mesh $\mathtt{M}$. The projections are mapped sequentially, generating updated meshes $\mathtt{M}^{*}$, $\mathtt{M}^{**}$, etc. Each projection contains a single triangle to map to the global mesh. Triangles $\mathtt{T}^{1}_{\{\mathtt{V}_a,\mathtt{V}_b,\mathtt{V}_c\}}$, $\mathtt{T}^{2}_{\{\mathtt{V}_d,\mathtt{V}_e,\mathtt{V}_f\}}$ and $\mathtt{T}^{3}_{\{\mathtt{V}_g,\mathtt{V}_h,\mathtt{V}_i\}}$, correspond to projections $\mathbf{C}^{j=1}$, $\mathbf{C}^{j=2}$, $\mathbf{C}^{j=3}$, respectively. The quality of the triangles is such that the following holds:

$$\mathbf{q}\big(\mathtt{M}_n\big) < \mathbf{q}\big(\mathtt{T}^{1}_{\{\mathtt{V}_a,\mathtt{V}_b,\mathtt{V}_c\}}\big) < \mathbf{q}\big(\mathtt{T}^{2}_{\{\mathtt{V}_d,\mathtt{V}_e,\mathtt{V}_f\}}\big) < \mathbf{q}\big(\mathtt{T}^{3}_{\{\mathtt{V}_g,\mathtt{V}_h,\mathtt{V}_i\}}\big)\, \forall \mathtt{M}_n \in \mathtt{M}, \tag{16}$$

and the mesh update cost parameter is $\alpha = 1$, which means that there is no cost associated to the updating of the mesh (see eq. (10)). In other words, the insertion of all three candidate triangles is considered beneficial. The initial mesh is shown in Fig. 7 (*a*), along with the three candidate triangles. Figure 7 (*b*) shows the mesh after the insertion of the first candidate triangle, i.e., $\mathtt{M}^{*}$. Since there is no overlap, the candidate triangle is added to the mesh $\mathtt{M}^{*}_{\{4,5,8\}}$, and edges $\mathtt{M}^{*}_{\{4\text{-}5\}}$, $\mathtt{M}^{*}_{\{5\text{-}8\}}$, and $\mathtt{M}^{*}_{\{4\text{-}8\}}$ are constrained. Also, since there was no overlap detected, the initial configuration of the mesh is preserved. The result of the second insertion is shown in Fig. 7 (*c*). In this case, there is overlap between candidate triangle $\mathtt{T}^{2}_{\{\mathtt{V}_d,\mathtt{V}_e,\mathtt{V}_f\}}$ (seen in Fig. 7 (*a*)) and triangle $\mathtt{M}^{*}_{\{2,5,7\}}$ (seen in Fig. 7 (*b*)). An intersection between edges $\mathtt{V}_d$-$\mathtt{V}_e$ and edge $\mathtt{M}^{*}_{\{5\text{-}7\}}$ (seen in Fig. 7 (*b*)), is detected. As a result, the constraint from edge $\mathtt{M}^{*}_{\{5\text{-}7\}}$ is removed. The insertion results in a new triangle $\mathtt{M}^{**}_{\{9,10,11\}}$. Note also that the overlapping triangle $\mathtt{M}^{*}_{\{2,5,7\}}$ was not preserved, i.e., it does not exist in the new mesh $\mathtt{M}^{**}$. Finally, the third insertion detects that triangle $\mathtt{T}^{3}_{\{\mathtt{V}_g,\mathtt{V}_h,\mathtt{V}_i\}}$ overlaps triangles $\mathtt{M}^{**}_{\{3,4,5\}}$, $\mathtt{M}^{**}_{\{4,5,8\}}$ and $\mathtt{M}^{**}_{\{2,3,5\}}$. Edges $\mathtt{M}^{**}_{\{3\text{-}4\}}$, $\mathtt{M}^{**}_{\{4\text{-}5\}}$ and $\mathtt{M}^{**}_{\{3\text{-}5\}}$ intersect the edges of $\mathtt{T}^{3}_{\{\mathtt{V}_g,\mathtt{V}_h,\mathtt{V}_i\}}$ which is why their



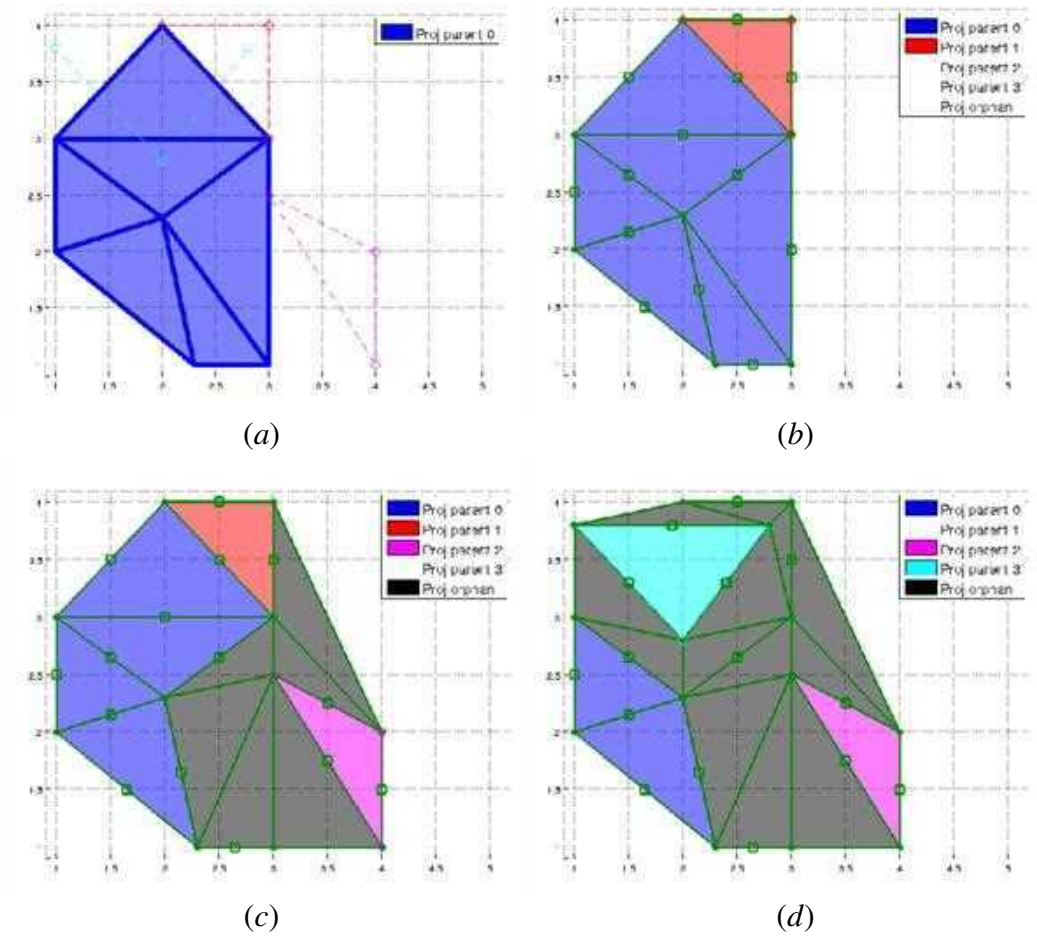
(*a*) (*b*)

(*c*) (*d*)

Figure 7: The insertion operation, example 1: (*a*) candidate triangles and initial mesh $\mathtt{M}$; (*b*) first insertion, mesh $\mathtt{M}^{*}$; (*c*) second insertion, mesh $\mathtt{M}^{**}$; (*d*) third insertion, mesh $\mathtt{M}^{***}$.

constraints are removed (actually, in this case they disappear after the candidate triangle is inserted).

The insertion of candidate triangles sometimes creates not only the candidate triangle itself, but also some additional triangles on the mesh. It is the case, for example, of triangle $\mathtt{M}^{**}_{\{7,9,10\}}$. We refer to this type of triangles as orphan triangles, meaning they have no parent projection. These are shown in grey color in Figure 8. Unlike triangles with parent projections, these triangles do not belong to any



(*a*) (*b*)

(*c*) (*d*)

Figure 8: The projection parent status of each triangle (same example as in Fig. 7): (*a*) candidate triangles and initial mesh $\mathtt{M}$; (*b*) first insertion, mesh $\mathtt{M}^{*}$; (*c*) second insertion, mesh $\mathtt{M}^{**}$; (*d*) third insertion, mesh $\mathtt{M}^{***}$.

projection and thus they do not derive from the DDT triangulation executed over an image of some projection. Because of this, there is no guarantee that these orphan triangles are compliant with edges in the projection images. For this reason, we propose that orphan triangles are set to have the quality 0. In summary, this approach for the update of a global primitive mesh consists of a set of procedures

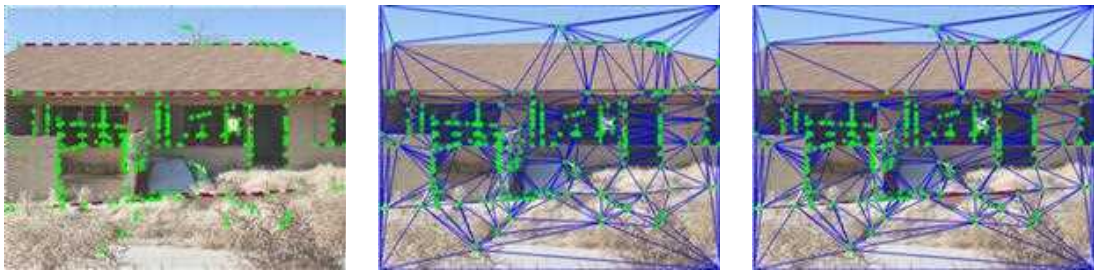
that are capable of updating the mesh whenever new, better quality triangles are available for insertion, but at the same time the mechanism is capable of filling the gaps left empty using orphan triangles.

4. Results

This section shows results both from one-shot texture mapping using DDTs, as well as results from the algorithm proposed to conduct incremental texture mapping.

4.1. One-shot Texture Mapping

Figure 9 (*a*) shows the detection of line segments in an image. Figure 9 (*b*) displays the result of a Delaunay triangulation with arbitrary configuration, e.g. computed by giving only the vertices as input (green dots in Fig. 9 (*a*)). Because the triangulated mesh has an arbitrary configuration, triangle often contain areas with multiple textures. This would cause problems when using affine texture mapping. Notice the large triangle that covers part of the roof of the building, as well as a portion of the sky. This triangle contains a significant change in color and thus



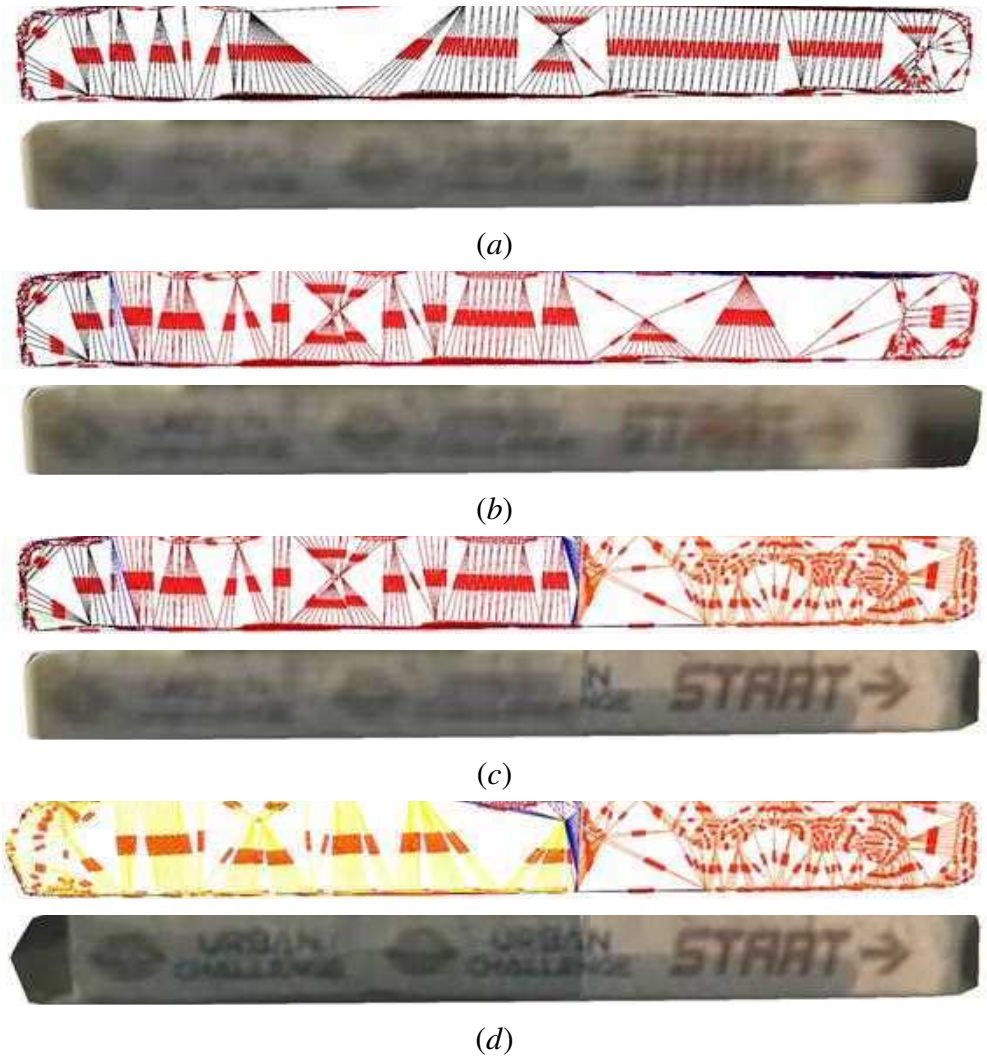
(*a*) (*b*) (*c*)

Figure 9: One-shot texture mapping: (*a*) image with line segments detected (red lines); (*b*) arbitrary Delaunay triangulation; (*c*) proposed approach, using a constrained Delaunay triangulation. Constrained edges marked in red.

its affine texture mapping would result inaccurate. Figure 9 (*c*) shows the result of the proposed DDT approach, where a constrained Delaunay triangulation is used. This triangulation is computed using as input the vertices as in the previous case but also the detected line segments (red lines in Fig. 9 (*a*), constrained edges also shown in (*c*) with red lines). In this case, the large triangle described above does not exist. In fact, there are no triangles which contain both sky and roof. Thus, we can argue that the proposed approach creates a mesh in which triangles contain smooth color transitions. The next section addresses the incremental update of these triangulation meshes.

4.2. Incremental Texture Mapping

To show the results of incremental texture mapping, we recover the example of section 1 (see Fig. 1): the vehicle approaches a wall panel, which has the word *START* written on it and collects four images in sequence (color coded black-red-orange-yellow in Fig. 10). The global mesh is created with the first image and then updated three times. The global triangulated mesh at each iteration is shown in Figs. 10 (*top*). Textures for each of these cases are show in Figs. 10 (*bottom*). Projection $\mathbf{C}^{\{k=4,l=1,t=t_1\}}$ is used to create the global mesh. Thus, the global mesh is composed only of triangles with parent projection $\mathbf{C}^{\{k=4,l=front\,center,t=t_1\}}$ (black triangles in Fig. 10 (*a*)). Then, a new projection $\mathbf{C}^{\{k=4,l=1,t=t_2\}}$ becomes available. The global mesh is updated (Fig. 10 (*b*)), and now contains a majority of triangles from $\mathbf{C}^{\{k=4,l=1,t=t_2\}}$ (red triangles). Then projection $\mathbf{C}^{\{k=4,l=1,t=t_3\}}$ is mapped. Since only a right side portion of the primitive is seen, orange triangles can be observed on the right side of Fig. 10 (*c*)), while the left side retains red colored triangles from previous projections. Orphan triangles (in blue) are generated to fill the gaps that appear between the triangles with parent projections. Finally,



(*a*)

(*b*)

(*c*)

(*d*)

Figure 10: The evolution of the global primitive mesh (*top*) as well as the texture (*bottom*): (*a*) time $t = t_1$; (*b*) time $t = t_2$; (*c*) time $t = t_3$, insertion of front center camera; (*d*) time $t = t_3$, insertion of front left camera.

projection $\mathbf{C}^{\{k=4,l=3,t=t_4\}}$ is used. This image views only the left portion of the wall panel. As such, we can see yellow triangles on the left side of Fig. 10 (*d*)). This example shows how the proposed mechanism is capable of creating and maintaining a global triangulated mesh which is used for enhancing the texture

mapped onto the GPPs whenever new (and better) images are collected.

4.3. Projection of a Single Camera onto the Ground Plane

This section shows three examples of how the global primitive mesh evolves when using a single camera to map a single primitive. We consider a similar scene to the one presented in Fig. 1. Throughout the three time instants $t = t_1$, $t = t_2$ and $t = t_3$, the vehicle is moving forward. From t_1 to t_2 the vehicle drives straight, and from t_2 to t_3 the vehicle turns slightly to the right. In this case the primitive that represents the ground plane is used for texture mapping ($k = 0$). As a consequence, there is always a portion of the images from the projections that view the ground. In other words, at all instants any of the cameras view a portion of the ground, since they are pointed downwards. We will consider three different cases, each generating a unique scene representation: In the first case only the front center camera ($l = 1$) is used for projection. Hence there will be three projections: $\mathbf{C}^{\{k=0,l=1,t=t_1\}}$, $\mathbf{C}^{\{k=0,l=1,t=t_2\}}$ and $\mathbf{C}^{\{k=0,l=1,t=t_3\}}$; In the second case only the rear center camera ($l = 4$) is used for projection. Hence there will be three projections: $\mathbf{C}^{\{k=0,l=4,t=t_1\}}$, $\mathbf{C}^{\{k=0,l=4,t=t_2\}}$ and $\mathbf{C}^{\{k=0,l=4,t=t_3\}}$; In the third case only the front left camera ($l = 3$) is used for projection. Hence there will be three projections: $\mathbf{C}^{\{k=0,l=3,t=t_1\}}$, $\mathbf{C}^{\{k=0,l=3,t=t_2\}}$ and $\mathbf{C}^{\{k=0,l=3,t=t_3\}}$.

The final global primitive meshes (those obtained after inserting projections at times t_1,t_2 and t_3) for each case are displayed in Fig. 11 (*left column*). Figure 11 (*right column*) shows the percentage of triangles of the global mesh that belong to each projection, as a function of the mission time. Note that the final position of the vehicle (which is the same for all cases) is depicted in the images, and bear in mind that, during this sequence, the vehicle moves forward from the right to the left. The triangles of the global primitive mesh are shown in colors, where each

color corresponds to a particular projection.

Fig. 11 (*first row, left*) shows the distribution of triangles according to the parent projection. In this case, the images are provided by the front center camera. As the vehicle moves forward, the ground in front of the vehicle that has been previously mapped by previous projections is now visible in images at a closer range. This leads to the effect that more recent projections tend to override older projections, i.e., red color (t_2) overrides black color (t_1), and yellow color (t_3) overides the other two. Figure 11 (*first row, right*) shows that at time t_1, only triangles from the first projection (black) and orphan triangles (blue) exist. Then, at time t_2, the triangles from the second projection (red) are added to the global mesh. As a consequence, the percentage of triangles from the first projection (black) decreases. At time t_3, the third projection again takes the major slice of percentage with respect to the previous two projections. In front facing cameras, when the vehicle is moving forward, more recent projections tend to contribute with a larger portion of the total triangles in the global mesh.

The second case is shown in 11 (*second row, left*). Here, since the camera is facing the rear side of the vehicle, the opposite phenomena occurs: the vehicle is moving away from the ground behind it, and thus older projections were taken at closer distances to the ground. As a consequence, the red color (projection at t_2) overrides the yellow color (projection at t_3), and the black color (projection at t_1, the oldest one) overrides all others. This is observable in Fig. 11 (*second row, right*), where the first projection (black) is, at all times, the one with the largest percentage of the triangles.

Figure 11 (*third row, left*) shows the third case. Here, since the camera is facing the left side of the vehicle, a hybrid phenomena takes place. For each projection,

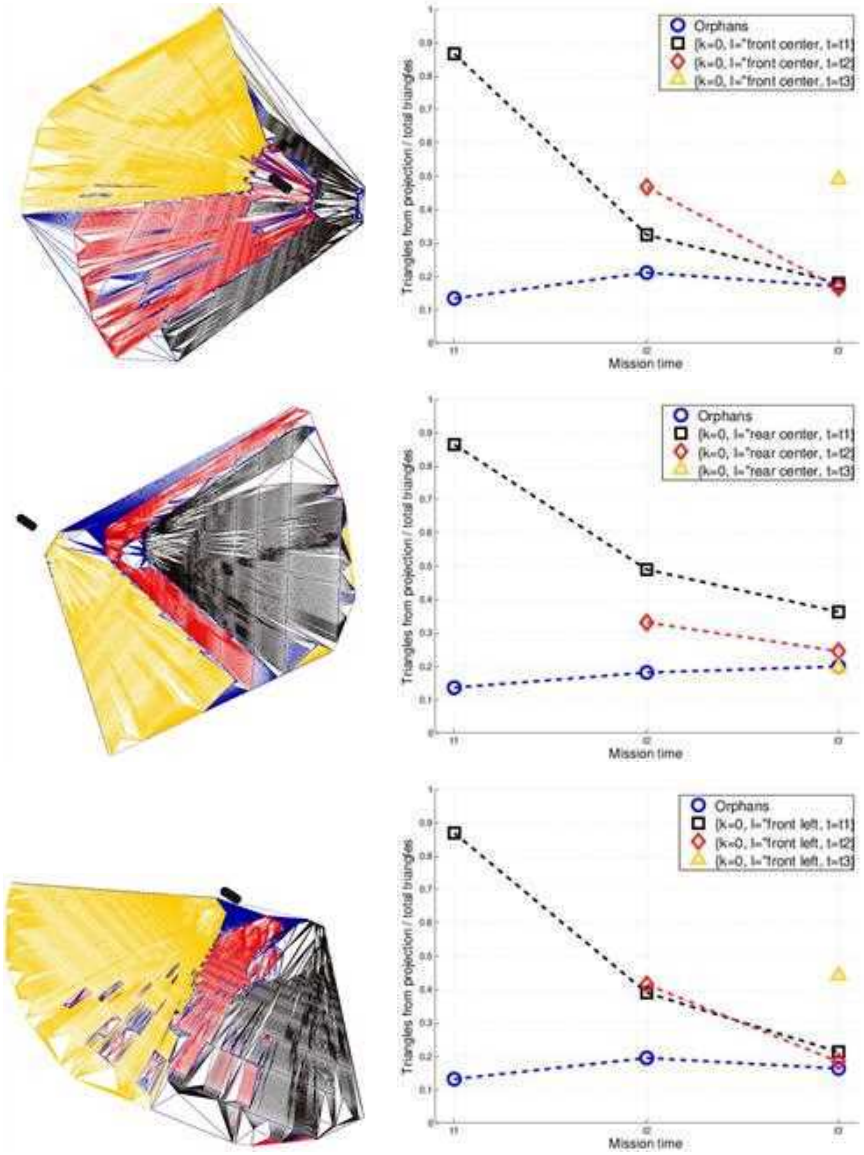


Figure 11: Mapping of a single camera to the ground plane: (*first row*) front center camera; (*second row*) rear center camera; (*third row*) front left camera; (*left column*) global primitive meshes; (*right column*) contribution of each projection to the total number of triangles in the globel mesh. Colors denote each of the projections, i.e., black is time t_1, red is time t_2 and yellow is time t_3. Blue triangles are orphan triangles.

there is always a portion of the triangles, i.e., those that map the ground directly in front of the camera for that particular instant, that have a higher projection quality when compared to others. Figure 11 (*third row, right*) also shows this tendency: the percentage of projections tends to be the same for all projections, which is why the second projection (red) when first mapped at time t_2 achieves approximately the same percentage of triangles as the first projection (black). They continue to have similar percentages also at time t_3. At time t_3, the third projection (yellow) obtained a higher value of percentage because the vehicle as turned slightly to the right and the left camera faced an area of the ground that was not previously mapped by any of the previous projections.

4.4. Projection of Multiple Cameras onto the Ground Plane

The examples given in section 4.3 have shown that the proposed algorithm is capable of handling multiple projections, correctly determining which are the best quality projections to map onto the global mesh. Nonetheless, those examples were simplified since only one camera was considered to provide projections in each case. In this section, the five cameras onboard the *Talos* are used to provide projection to be mapped onto the ground plane. The same sequence is used: the vehicle is moving forward and three time instants are used to generate projections. Each time instant t_1, t_2 and t_3 generates five projections, one for each camera. Figure 12 (*a*) shows the state of the global mesh after time t_1. Five projections are contained in the mesh. At time t_2, the global mesh incorporates many of the projections that are computed at this time (Fig. 12 (*b*)). The same occurs at time t_3 (Fig. 12 (*c*)). Note that these images are not exactly the same as those in Figure 11, because in that case only the final global projection mesh was shown for three different examples. Here, we show the state of a single global projection mesh at

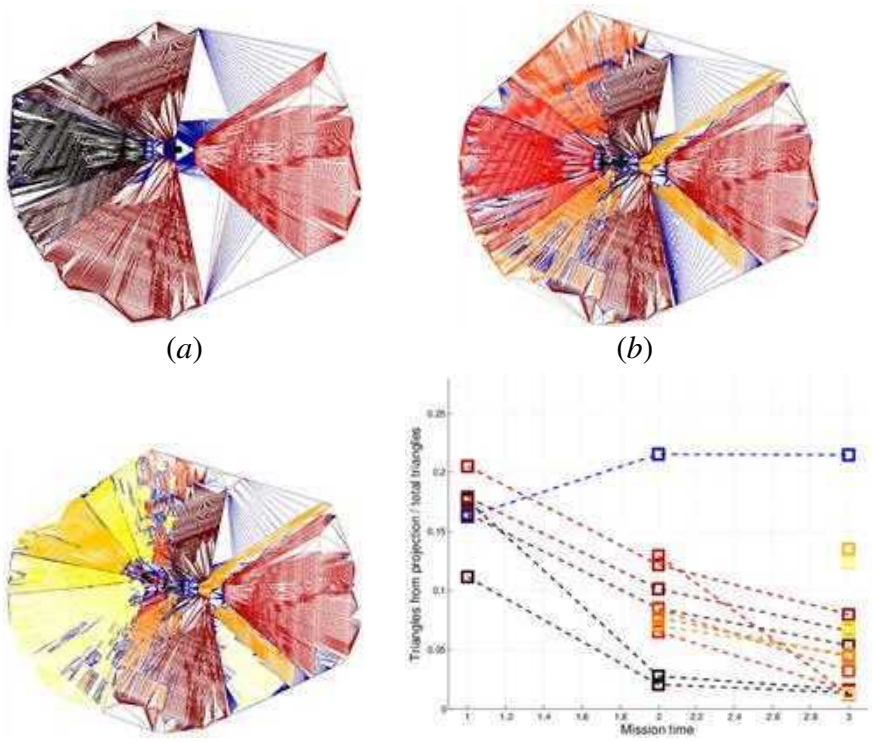


Figure 12: Distribution of triangles according to projection, for an example with five cameras. Three time instants (15 projections in total) are considered; (*a*) t_1; (*b*) t_2; (*c*) t_3; (*d*) percentage of triangles by parent projection. Projections are colored with a black to yellow color coding, denoting oldest to newest projections. Blue color denotes orphan triangles.

times t_1, t_2 and t_3. Thus, in this case it is possible to see how the mesh evolved as more projections became available. The resulting mesh is an intricate mosaic of triangles coming from several projections. At time t_1, the area of projection from the rear center camera was not connected to the areas of projection of the other cameras. Note that the red triangles in Fig. 12 (*a*) are not connected to any triangle with a parent projection, only to orphan triangles. This unmapped region corresponds to the ground that was bellow the vehicle at time t_1. Obviously,

there is no coverage from the cameras for that area, so the system handles this by defining orphan triangles (blue) to cover that area. At time t_2, the vehicle has moved forward, and the uncovered ground is now visible from the rear camera. Hence, the areas mapped by the rear cameras connect to the areas mapped by the other cameras, as seen in Fig. 12 (*b*). At time t_3, since the vehicle has turned to the right, the rear camera now views a different portion of the ground that had not been captured by any other camera. Note, in Fig. 12 (*c*), how the triangles of the rear camera (the brightest yellow at the bottom right side) map a region that was not seen before and was previously covered only by orphan (blue) triangles.

Figure 12(*d*) shows the percentage of triangles of each projection as a function of the mission time. As each time instant, only newly acquired projections are used to update the mesh. Hence, triangles from previous iterations, if removed, will not be retested for insertion. That means each triangles is tested for insertion a single time. If a triangle is removed, it will never again be reinserted. This can be observed in the Figure, since none of the projections increases the percentage of triangles it contains. Figure 13 shows the fifteen images used to compute these representations.

As the vehicle moves and turns around, more and more of the ground that had not been viewed before is covered by new projections. This is a clear example of why integrating several projections over time is advantageous. A composite photometric description of the environment can be obtained that was impossible to compute without the capability of integrating multiple projections over time in an incremental fashion.

The incremental texture mapping of an entire scenario can be observed in `https://youtu.be/UG8WMCDxx8A`. The scenario is composed of the en-

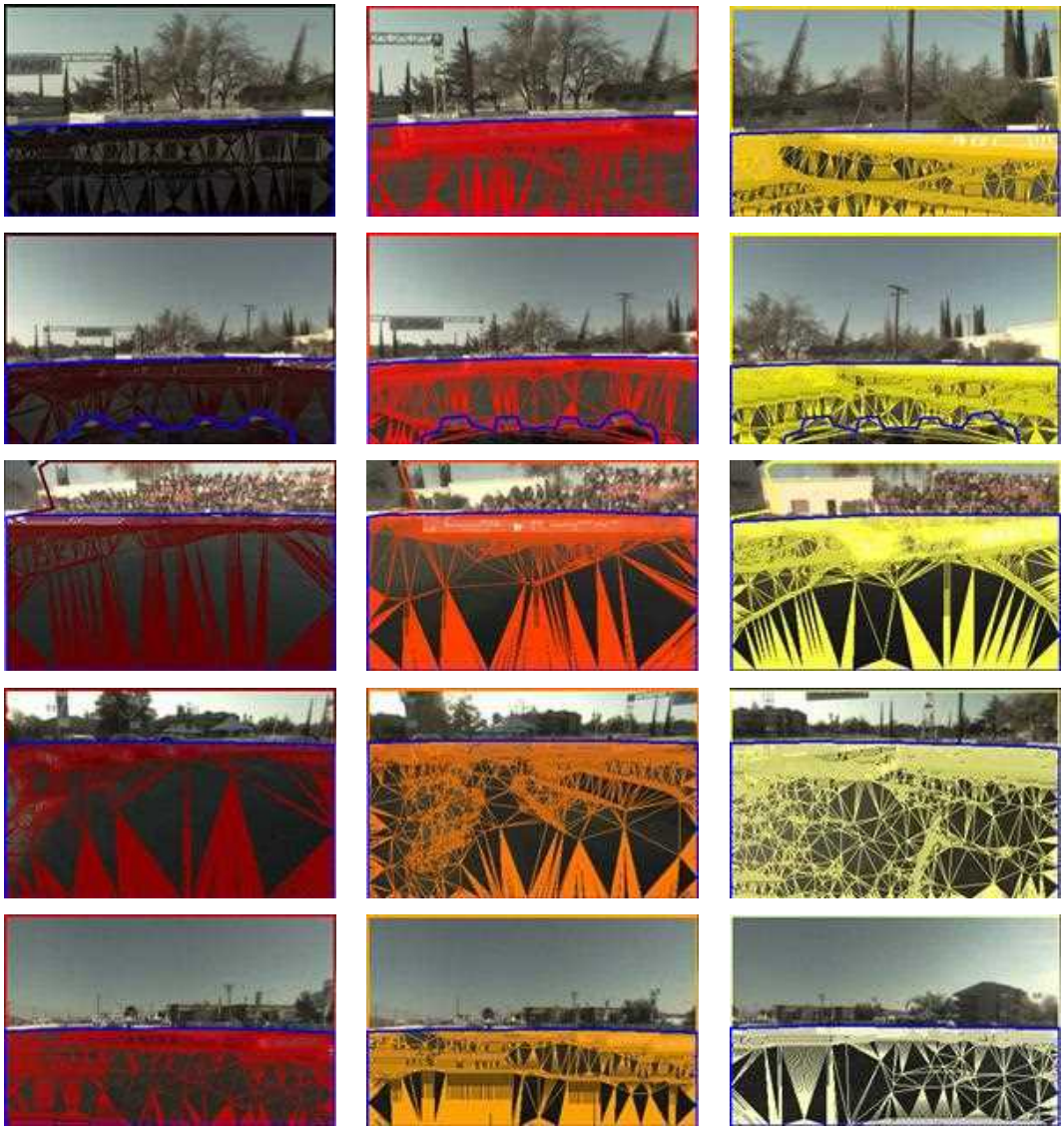


Figure 13: Images and local triangulated meshes for all projections shown in Fig. 12: First row front center teleobjective camera; Second row front center camera; Third row, front right camera; Fourth row, front left camera; Fifth row, rear center camera; Left column: time t_1, (Fig. 11 (*a*)); Middle column: time t_2, (Fig. 11 (*b*)); Right column: time t_3, (Fig. 11 (*c*));

tire MIT sequence. All five cameras onboard the *Talos* vehicle are used as input to the texture mapping. Geometric primitives are represented in the environment by the blue-green polygons. A blue to green colormap is used to color the primitives according to their index, the more recently detected the primitive, the closer to green it is. Photometry is represented by the texture mapped onto the primitives. Note that at each of the time instants new projections will update the global meshes of the detected polygonal primitives. Hence the scenario representation will evolve photometrically over time. Furthermore, also the geometric representation will evolve over time (see [2] for details). For a better visualization, the primitive that represents the ground plane is not textured in the video.

5. Conclusions

This paper addressed the problem of how to create and update a triangulated mesh. These meshes are used for texture mapping surfaces in 3D, and the input are images collected from cameras mounted on-board a vehicle. The geometric structure onto which texture is mapped is described in detail in [2] and given as a list of polygons. Because the atomic entities of the 3D structure are defined as polygons (rather than triangles), it is possible to perform a triangulation of the convex hull of that polygon, as opposed to having an arbitrary triangulation. This triangulation is computed in the image space, and is defined as a constrained Delaunay triangulation. This makes it possible to impose line segments as constrained edges in the triangulation, which creates triangles with smooth color transitions. This, in turn, makes it possible to use affine texture mapping.

Incremental texture mapping is done by creating and updating a global triangulated mesh per geometric primitive. The update of this mesh is done using

meshes created from projections. In this paper, we have proposed a sequence of operations which are used for inserting triangles from the projection mesh into the global triangulation mesh. This procedure ensures that the inserted triangles maintain their configuration as well as the existing triangles which do not overlap the inserted triangles. Furthermore, the proposed algorithm fills the gaps in the mesh where there are to triangles with parent projections with orphan triangles. Using this mechanism, the holes that could exist between textures of different projections are replaced by orphan triangles where texture is interpolated, resulting in a better overall quality of the texture.

To the best of our knowledge, this is the first approach in this field capable of fusing images continuously and in an incremental fashion in order to generate a single texture of good quality.

Acknowledgments

This work has been supported by the "Fundação para a Ciência e Tecnologia" (Portuguese Foundation for Science and Technology) under grant agreements SFRH/BD/43203/2008 and SFRH/BPD/109651/2015 and National Funds within project UID/EEA/50014/2013. This work was also financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme within project «POCI-01-0145-FEDER-006961». A. Sappa has been partially supported by: the Spanish Government under Project TIN2014-56919-C3-2-R and the PROMETEO Project of the "Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación de la República del Ecuador.

References

[1] M. Oliveira, V. Santos, A. D. Sappa, P. Dias, Robot 2015: Second Iberian Robotics Conference: Advances in Robotics, Volume 1, Springer International Publishing, Cham, 2016, Ch. Scene Representations for Autonomous Driving: An Approach Based on Polygonal Primitives, pp. 503–515.

[2] M. Oliveira, V. Santos, A. Sappa, P. Dias, A. P. Moreira, Incremental scenario representations for autonomous driving using geometric polygonal primitives, Robotics and Autonomous Systems (submitted January 2016, under review).

[3] A. S. Huang, M. Antone, E. Olson, L. Fletcher, D. Moore, S. Teller, J. Leonard, A High-rate, Heterogeneous Data Set from the DARPA Urban Challenge, International Journal of Robotics Research 29 (13) (2011) 1595–1601.

[4] A. Huang, E. Olson, D. Moore, Lcm: Lightweight communications and marshalling, in: Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, 2010, pp. 4057 –4062.

[5] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, P. Haeberli, Fast shadows and lighting effects using texture mapping, SIGGRAPH Comput. Graph. 26 (2) (1992) 249–252.

[6] P. Debevec, Y. Yu, G. Boshokov, Efficient view-dependent image-based rendering with projective texture-mapping, Tech. rep. (1998).

[7] S. Rippa, Long and thin triangles can be good for linear interpolation, SIAM Journal on Numerical Analysis 29 (1) (1992) pp. 257–270.

[8] M. Garland, P. S. Heckbert, Fast polygonal approximation of terrains and height fields, Tech. Rep. CMU-CS-95-181 (September 1995).

[9] R. Schätzl, H. Hagen, J. C. Barnes, B. Hamann, K. I. Joy, Data-dependent triangulation in the plane with adaptive knot placement, in: Geometric Modelling, 2001, pp. 309–321.

[10] H. Hoppe, Progressive meshes, in: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, SIGGRAPH '96, 1996, pp. 99–108.

[11] L. Demaret, N. Dyn, A. Iske, Image compression by linear splines over adaptive triangulations, Signal Process. 86 (7) (2006) 1604–1616.

[12] A. D. Sappa, M. A. García, Coarse-to-fine approximation of range images with bounded error adaptive triangular meshes, J. Electronic Imaging 16 (2) (2007) 23010.

[13] N. Dyn, D. Levin, S. Rippa, Boundary correction for piecewise linear interpolation defined over data-dependent triangulations, J. Comput. Appl. Math. 39 (2) (1992) 179–192.

[14] L. L. Schumaker, Computing optimal triangulations using simulated annealing, Computer Aided Geometric Design 10 (3-4) (1993) 329–345.

[15] Opengl, D. Shreiner, M. Woo, J. Neider, T. Davis, OpenGL(R) Programming Guide : The Official Guide to Learning OpenGL(R), Version 2 (5th Edition), Addison-Wesley Professional, 2005.

[16] D. Blythe, The direct3d 10 system, ACM Trans. Graph. 25 (3) (2006) 724–734.

[17] B. Lehner, G. Umlauf, B. Hamann, Survey of techniques for data-dependent triangulations, in: H. Hagen, M. Hering-Bertram, C. Garth (Eds.), GI Lecture Notes in Informatics, Visualization of Large and Unstructured Data Sets, 2007, pp. 178–187.

[18] M. Cervenansksy, Z. Toth, J. Starinsky, A. Ferko, M. Sramek, Parallel gpu-based data-dependent triangulations, Computers and Graphics 34 (2) (2010) 125 – 135.

[19] I. Svalbe, Natural representations for straight lines and the hough transform on discrete arrays, Pattern Analysis and Machine Intelligence, IEEE Transactions on 11 (9) (1989) 941 –950.

[20] V. Kamat, S. Ganesan, A robust hough transform technique for description of multiple line segments in an image, in: Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on, Vol. 1, 1998, pp. 216 –220 vol.1.

[21] R. Guerreiro, P. Aguiar, Incremental local hough transform for line segment extraction, in: Image Processing (ICIP), 2011 18th IEEE International Conference on, 2011, pp. 2841 – 2844.

[22] M. Yvinec, 2D triangulations, in: CGAL User and Reference Manual, 4.0 Edition, CGAL Editorial Board, 2012.

[23] L. P. Chew, Constrained delaunay triangulations, in: Proceedings of the third annual symposium on Computational geometry, SCG '87, 1987, pp. 215–222.

[24] J. R. Shewchuk, General-dimensional constrained delaunay and constrained regular triangulations i: Combinatorial properties, Discrete Comput. Geom. 39 (1) (2008) 580–637.

[25] T. Moller, A fast triangle-triangle intersection test, Journal of Graphics Tools 2 (1997) 25–30.

[26] J.-W. Chang, M.-S. Kim, Efficient triangle–triangle intersection test for OBB-based collision detection, Computers & Graphics 33 (3) (2009) 235–240.

[27] A. D. Sappa, M. A. García, Incremental multiview integration of range images, in: ICPR, 2000, pp. 1546–1549.

[28] E. Fogel, R. Wein, B. Zukerman, D. Halperin, 2D regularized Boolean set-operations, in: CGAL User and Reference Manual, 4.0 Edition, CGAL Editorial Board, 2012.