3D Shape Cropping

Jean-Sébastien Franco, Benjamin Petit, Edmond Boyer

Vision, Modeling, and Visualization (2013)
Michael Bronstein, Jean Favre, and Kai Hormann (Eds.)

3D Shape Cropping

J.S. Franco[1,2], B. Petit[2], and E. Boyer[2]

[1]Grenoble Universities, LJK & Grenoble, France
[2]Inria, Grenoble, France

Abstract

We introduce shape cropping as the segmentation of a bounding geometry of an object as observed by sensors with different modalities. Segmenting a bounding volume is a preliminary step in many multi-view vision applications that consider or require the recovery of 3D information, in particular in multi-camera environments. Recent vision systems used to acquire such information often combine sensors of different types, usually color and depth sensors. Given depth and color images we present an efficient geometric algorithm to compute a polyhedral bounding surface that delimits the region in space where the object lies. The resulting cropped geometry eliminates unwanted space regions and enables the initialization of further processes including surface refinements. Our approach exploits the fact that such a region can be defined as the intersection of 3D regions identified as non empty in color or depth images. To this purpose, we propose a novel polyhedron combination algorithm that overcomes computational and robustness issues exhibited by traditional intersection tools in our context. We show the correction and effectiveness of the approach on various combination of inputs.

1. Introduction

Hybrid vision sensor systems are becoming more and more prominent with the increasing availability and low cost of depth sensors and general purpose cameras. While such sensors have been popularized for 3D interactive systems, there is a growing interest in using them for 3D reconstruction problems in broad areas including urban modeling [AFM*ne] or hand-held 3D object acquisition [NIH*11]. In this paper, we consider their use within hybrid multi-camera environments. Our focus is in efficiently combining these modalities to quickly build and segment 3D regions of interest and shapes in the form of an enclosing polyhedral model. Such a representation can be useful for a wide range of tasks, such as shape modeling, localization, and tracking.

Our objective is therefore to combine color and depth camera in multi-camera environments where a possibly sparse set of cameras are spatially distributed. In this context, short baseline or continuity between viewpoints can not be assumed, eliminating the SLAM strategy successfully used in other contexts [GK99]. While a few works exist specifically addressing this problem, e.g. [GFP08, KTD* 4], they focus on volumetric representations based on costly space discretizations. Using polyhedral representations, we explore here an alternative surface-based strategy that overcomes the precision-complexity tradeoff of volumetric methods.

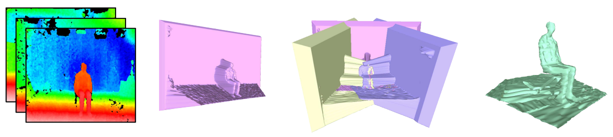
Figure 1: *An example of 3D shape cropping with three depth maps with large baselines. From left to right: input depth maps, contribution from one depth map, intersection of all depth map contributions, the output polyhedron.*

In order to exploit inputs from various sensors, our approach builds on their ability to delimit regions in space where objects lie, as with depth maps and 2D regions provided by depth and color images respectively. The associated algorithm fuses these inputs into a 3D shape approximation delimited by space regions identified as unoccupied by sensors. We coin this process *shape cropping*. The surface produced this way has noticeable intrinsic properties: it is a closed manifold that contains the actual objects present in the scene. Our contribution with respect to the state-of-the-art in multi-view is twofold: we first introduce shape cropping as a powerful pre-processing step to further applications; We second propose a new polyhedron intersection algorithm to solve for cropped geometry given various inputs, rooted in the simplifications and efficiency enhancements appropriate to this problem. Experiments demonstrate that this algorithm provides a fast yet robust alternative to the state-of-the art polyhedron intersection algorithms in this context.

The remainder of the paper is as follows: Section 2 reviews previous works. Section 3 introduces the approach. Section 4 details the intersection algorithm. Section 5 presents experimental results and comparisons before concluding and discussing in Section 6.

2. Previous work

Slam-based methods. Some of the first methods combining depth sensor measurements have arisen in robotics where linear range sensors were used initially for obstacle detection purposes [ME88], their observations aligned using odometry and fused in dense occupancy representations. Dense 2D range representations have then been used to update a map for localization purposes, leading to the well known family of SLAM methods [GK99]. With the advent of dense Time-of-Flight depth imagery and structured-light systems, the computer vision and graphics communities have built on this knowledge to lift the problem in 3D, shifting the focus on obtaining dense reconstructions from a set of continuous viewpoints, using for example ICP variants [RHHL02]. Recently some methods such as [NIH*11] have successfully taken these approaches to real-time integration with cheap off-the-shelf Kinect sensors. This family of approaches uses the assumption of low baseline and continuous camera trajectory to be within the validity domain of ICP and tracking. We explore a different direction with wide-baseline multiple viewpoints, as typically occurring in hybrid multi-camera systems.

Large baseline, multi-sensor systems. Hybrid color & depth multi-sensor systems is the particular focus of our work. This type of system is gaining more and more attention for general purpose interactive platforms such as smartroom [WB10] or 3D capture systems. Progress has recently been made in calibrating individual depth sensor characteristics [RDP*11], calibrating a set of depth cameras [KCTT08a] and calibrating a depth-color camera pairs [HHAL11], paving the way to broader multi-sensor usage. Some approaches have started exploring the data fusion problem itself, integrating ToF data with stereo [KTD* 4, DMZC10], or several ToF sensors together [KCTT08b]. Hybrid ToF-image integration has also been proposed using silhouettes [GFP08, BGM06]. Among the different representations used for fusion a large majority is volumetric, such as occupancy grids [GFP08, KTD* 4] or the fusion is on depth maps themselves [DMZC10] but surfaces are not considered. Closer to our purpose [BGM06] uses the GPU to render a silhouette-depth envelope, but the envelope is not explicitly computed and thus not available for other tasks. To the best of our knowledge, no method explicitly computes the fusion as explicit surface representations as proposed in this paper.

There are several advantages of manipulating a surface as opposed to a volumetric representation. The representation is more compact and does not require a space discretization. It is also more suitable for tasks that rely on surface based functions, for instance appearance and curvature functions. In addition, this representation is often required to initialize other tasks, fine scale surface refinement [ZBH11, DP11] for instance.

Polyhedron intersection algorithms. We propose to compute the bounding surface as the intersection of the complements of the regions identified as empty in images, where such regions are polyhedra in 3D. Polyhedral algorithms have been studied in the general case [Hof89] where boolean operations are shown to reduce to face-to-face intersections and treatment of degeneracies lead to complex algorithms. Various off-the-shelf implementations are now available but either exhibit robustness issues (e.g. GTS) or large computational overhead to resolve these issues (e.g. CGAL [HKM07]). As will be further discussed in §4.1, in the following we investigate an alternative dedicated to real data, where generic treatment of degeneracies is not necessary. With these assumptions we present a lightweight solution that relies on a simpler primitive operation than those of traditional libraries, i.e. edge-to-face intersections instead of face-to-face intersections. It proves to be fast while reliable with the visual data we consider.

3. Principle

We are given a set of views of the same scene, indifferently imaging color or depth. We assume the system is calibrated in projection and depth, using e.g. [KCTT08b] or

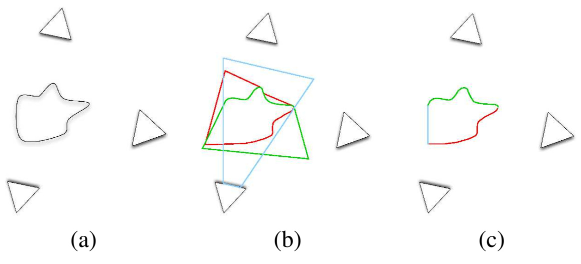

Figure 2: *Principle of the method: (a) Real scene and cameras. (b) In green and red, depth-cones produced by depth cameras, in blue, viewing cone produced by color camera's silhouettes. (c) Intersection of all contributing cones for shape cropping.*

[HHAL11]. Let $I = \{I_1, \cdots, I_n\}$ be the set of color images and $D = \{D_{n+1}, \cdots, D_m\}$ be the set of depth images.

The principle of our method is to construct a 3D bounding surface that fuses the information given by each modality. We here examine the case of two modalities, silhouettes extracted from color images and depth images. Assuming silhouettes $\mathcal{S}^i$ of observed objects of interest can be extracted from color images, each silhouette defines a 3D volume, called a *viewing cone*, which encloses possible locations of the observed shape. View i's viewing cone can be defined as the set of 3D points that project in $\mathcal{S}^i$. Intersecting the viewing cones for all color views typically leads to a discrete *visual hull* [Lau94]. Similarly depth images define an object inclusive region behind the depth surface discretized by the image, and contained within the visibility pyramid of the device. Inspired by the terminology used in [BGM06], we call the delimiting surface of this volume a *depth cone*. We build a polyhedral surface for both types of cones, as described below.

Depth Cone Construction. For each depth map D_j, we build depth cones as follows. We build triangulated height mesh based on the measurements of depth map. To achieve a tradeoff between precision and geometry, we filter and downsample the heightmap. For each sample position x of the map, a corresponding 3D position is computed and connected to its neighbors using 8 triangles. Assuming a bounding size is available for the scene of interest, we close the height mesh by extending it in depth on the sides and tessellating the rear with a set of faces to obtain a closed, object-inclusive depth cone. On some sensors such as the Kinect, certain depth pixels do not contain valid measurements. In order to preserve the containment property, we replace such measurements with the lowest possible depth compatible with the view and scene bounding size.

Viewing Cone Construction. For each silhouette $\mathcal{S}^i$, we start by polygonizing the silhouette map as a 2D polygon. Depending on the properties, various discretization schemes exist such as exact [DRR95] or approximate edge recognition (e.g. Marching Squares). Once polygonized, the contour can be lifted in 3D by computing for each vertex x of the contour discretization, the corresponding 3D ray and joining them to build cone faces. Using the scene bounding size, we truncate the viewing cone at the front and back of the viewing pyramid, in order to generate a finite polyhedron.

4. Efficient Surface Combinations

Once viewing cones and depth cones have been constructed, we need to combine them by computing their intersection, as the most restrictive set of points satisfying the constraints brought by each view. To this goal, one may at first consider using off-the-shelf mesh libraries.

4.1. Contribution

We propose to compute the bounding surface as the intersection of the complements of the regions identified as empty in images, where such regions are polyhedra in 3D. The fundamentals of polyhedron intersection are well documented [Hof89], and several solutions have been proposed to comput polyhedra with off-the-shelf implementations, such as GTS. However these algorithms are fundamentally based on a face-to-face intersection paradigm, where each face of polyhedron A is intersected against face of polyhedron B to compute general boolean operations between two polyhedra. Because they are meant to address the generic cases where degeneracies may happen, this leads to complex algorithms which are difficult to implement robustly without compromises. E.g. GTS only addresses the case of triangular meshes, and is shown to have severe failure cases in our experiments despite this simplification. Among various successful solutions to the robustness issue, exact arithmetic techniques have been used, leading to popular libraries such as CGAL. CGAL relies on Nef Polyhedra [HKM07], which address the generic case with formal properties. However this comes to the price of a huge overhead which strongly penalizes computation time.

To achieve a better efficiency/robustness tradeoff, we propose a new polyhedral intersection algorithm specifically designed for our problem, focused on simplicity and efficiency. As with existing algorithms we focus on the binary intersection case, with two polyhedra. Because we are dealing with real visual data which is noisy, numerical degeneracies and coincident primitives are highly unlikely to occur with double precision arithmetic, as was experimentally validated in previous algorithms dealing with polyhedral combinations of visual data [FB09]. Furthermore, even though the topology of the result might slightly change with small displacements of the geometry, the general shape observed is stable to small input perturbations. Because our focus is on building a good shape approximation rather than exact topology computation, we take advantage of these observations and build our algorithm on the simpler, generic intersection case. The key to the proposed algorithm lies in the edge-centric view of polyhedral intersections. This yields a very simple polyhedral intersection algorithm, with only two types of final edges of $A \cap B$ and guiding the main stages in the algorithm. Edges are either *restricted edges*, i.e. edges that are an intersected restriction of input polyedron edges, or *intersection edges*, arising from the intersection of two faces of the two input polyhedra.

4.2. Notation

Let $A = \{F_A, E_A, V_A\}$ and $B = \{F_B, E_B, V_B\}$ be two input polyhedra and $O = \{F_A, E_O, V_O\}$, the output intersection polyhedron, where F, E and V denoting faces, edges and vertex sets respectively. The primitives and processing steps of the algorithm are shown in figure .

Restricted edges E_r. On one hand there are edges obtained by intersecting initial edges of A with the inside volume of B, and vice-versa, which we call *restricted edges*. The incident vertices of restricted edges are either vertices of the initial polyhedra, or newly created *intersection vertices* lying at the intersection of an initial edge of A and a face of B (or vice-versa).

Intersection edges E_i. The other type of edges results from face to face intersections. An interesting property of these edges under the assumption of genericity is that their incident vertices are necessarily two intersection vertices. Thus if we first compute all restricted edges and their intersection vertices, identification of intersection edges involves no geometric computation (symbolic) but only a search among already computed intersected vertices. We will take advantage of this property in designing the algorithm.

We propose a simple algorithm in four steps, as summarized in 3:

1. Compute all intersection vertices (numerical)
2. Compute the set of restricted edges from intersected vertices (symbolic)
3. Compute the set of intersected edges (symbolic)
4. Identify faces of the final polyhedron (symbolic)

A key property of the proposed algorithm is thus to isolate and rely on only one numerical geometric computation, that of intersected vertices in step 1. We now present each step's processing scheme in detail in the four following sections.

4.3. Intersected vertices step

The contribution of each edge e of an input polyhedron (e.g. A without loss of generality) to the final intersected surface is a set of restricted edges R_e, contained within e. A single edge of A can contribute several edges to the final result because non-convex parts of the surface B may lead to several intersections of B along e. The edges of R_e are either delimited by existing vertices of the initial polyhedron A, or by newly created intersection vertices as soon as e intersects the surface of B. Thus all vertices of the final polyhedron are either vertices of A inside B, of B inside A, or intersection vertices.

Determining wether a vertex v of A is inside the other polyhedron B is typically solved by shooting a ray from v to infinity and counting the winding number, i.e. the number of intersections with the surface of B. Determining the set of intersection vertices and restricted edges also involves shooting a ray along the direction of each edge e of the initial polyhedra. Thus both problems will be solved efficiently by shooting rays along edges. The problem of ray to polygon intersection has been thoroughly studied in the graphics rendering community, where it is used e.g. for raytracing, and is often simplified by triangulating the polygon. We use a triangulation for each face, for the purpose of performing edge to face intersection tests.

We use algorithm [MT97], which yields the set of intersection vertices by solving a linear system involving barycentric coordinates on the triangles of face B and the linear coordinate t of the intersected edge along e (see Fig. 4-Left).

Each newly created intersection vertex of v_i is stored on a list V_i^e attached to the original edge e. v_i also stores the reference to the edge e and the facet f involved in the intersection. Furthermore, both vertices of e store a reference to the closest intersection on the ray formed by e in euclidian distance. This reference helps determining wether the vertex v is inside or outside B.

At the end of this step a list of intersected vertices V_i is obtained and we know if the vertices V_A are contributing to the final polyhedra or not, we thus know all the vertices of V_O.

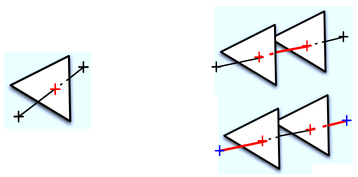

Figure 4: *Left: Intersected vertices step, an intersection vertex (in red) between an edge and a facet. Right: Restricted edges step, two situations where restricted edges (in red) are created using original edges (in blue) and intersected vertices (in red).*

4.4. Restricted Edges step

For each edge e in E_A, we sort the list of intersection V_i^e. We also know if their incident vertices are part of the final polyhedra. This gives us a list of vertices contributing to the restricted edges E_r^e of e, the number of vertices in this list is even by construction for finite solids. We then join the vertices together starting at one end of e (see Fig. 4-Right for two different cases).

This step produces a list of restricted edges E_r.

4.5. Intersection Edges

Existing vertices can be n-valent; however under the assumption of genericity any newly created vertex in V_i falls within a triangle of B and is thus necessarily trivalent. For those vertices we already know one incident edge coming from the connected restricted edge, the only unknown are the other two pending edges.

To connect those pending edges we take a look at the other intersections of the intersected face and the adjacent face of the restricted edge to find one that matches, i.e. which have the opposite combination of intersected and adjacent faces (see Fig. 5-Left).

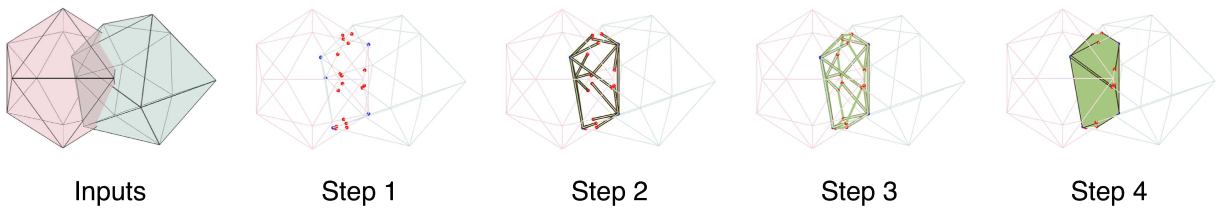

Figure 3: *Steps of the proposed algorithm. Step 1: Intersection vertices, at the locus of edge to face intersections. Step 2: Restricted edges, initial edges restricted to portions inside the other polyhedron. Step 3: intersected edges materializing surface-to-surface intersections of the polyhedra. Step 4: Face identification by oriented walkthrough of the connected edges.*

At the end of this step we have a list of intersected edges E_i and thus we know all the edges of E_O.

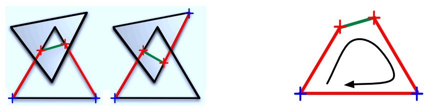

Figure 5: *Left: Intersected edges step, an intersection edge (in green) between two intersected vertices and restricted edges (in red). Right: Face identification step, connecting original and restricted edges (in red) with intersected edges (in green), always turning right.*

4.6. Face Identification

A last step consists in identifying the polyhedron faces F_O. 3D face contours are extracted by walking through the edges E_O taking right turns at each vertex. The created facets are n-gons possibly having inner and outer contours (see Fig. 5-Right).

5. Experiments

5.1. Polyhedron Intersection Validation

In this section, we validate the polyhedron intersection algorithm presented, using synthetic data. We evaluate the performance of the algorithm using the following criteria: the correctness of the output (manifold and closed tests), processing time, number of primitives generated (vertices, faces, triangles). All outputs of our algorithm have successfully tested as being manifold and closed. We also use these criteria for comparison purposes with two state of the art librarires CGAL and GTS. Those comparisons also validate the correction of our algorithm as we compare the geometry with CGAL, which can be used as benchmark for robustness purposes. The tests were conducted on a Core 2 Duo 3Ghz. The other frameworks involved are CGAL and GTS for simple intersection.

CGAL boolean operations are based on the *Nef Polyhedra* algorithm. Comparisons were performed using the supplied *Nef 3 filtered* demo program. Note that on some occasions, CGAL has exhibited some issues with N-gon faces and floating point precision vertex coordinates being converted to exact representation. This is because the vertices of N-gon planar faces are not exactly planar due to their numeric representation. This prevents the method from working in some circumstances without triangulating the given face. Our algorithm does not exhibit this limitation.

Because GTS only manipulates triangular mesh representations, GTS comparisons are done by triangulating the more general input meshes used as input for CGAL and our algorithm.

Both CGAL and GTS are using KD-tree as a fast intersection approximation. Our current proof of concept implementation of our framework performs exhaustive searches for edge to triangle intersections and is therefore quadratic $O(n^2)$. Using k-d trees would typically results in $O(n.\log n)$ complexity for intersection searches, thus we expect a significant speedup in future implementations.

5.1.1. Multiple Depth Cones

First we compare our framework using synthetic depth cones of the Standford bunny. The cones were created using a custom application which extracts the depth buffer from a virtually rendered view of a model, and creates a depth cone using the technique described in section §3. To be compliant with CGAL and GTS input requirements we triangulated the sides and the back of the depth cones. One input depth cone yields about 4K vertices and 8K triangular facets (see Fig. 6). Results are shown in table 1. They are similar for CGAL and our algorithm in terms of number of primitives generated. Note that despite a quadratic implementation, our algorithm outperforms CGAL in processing time. While GTS achieves shorter times, the number of primitives generated is higher due to GTS's intrinsic limitations. We are confident that the execution times of our method and GTS would be comparable with a tree-based implementation of our algorithm's intersection searches. Note that GTS crashed on executions with 8 depth cones and thus failed to produce outputs in this case.

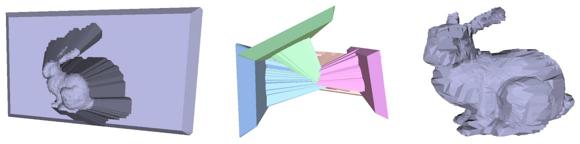

Figure 6: *Bunny with 4 depth cones. In order: one input depth cone, all input depth cones intersecting, visual cropping intersection output.*

Table 1: *Time and output surface comparison for the Standford bunny depth cones intersection.*

Framework	Time	#Vert	#Face	#Tri
Bunny 4 depth cones				
CGAL	1m31s	8K	5K	16K
GTS	2.13s	11K	33K	33K
Proposed	22.86s	8K	5K	16K
Bunny 8 depth cones				
CGAL	3m51s	13K	6K	26K
GTS	Failed: self-intersecting error			
Proposed	59.4s	13K	6K	26K

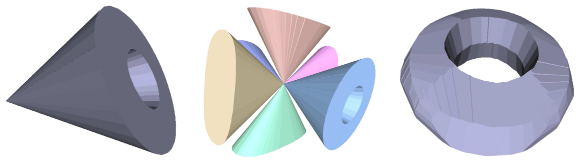

Figure 7: *In order: one input viewing cone, all input viewing cones intersecting, visual hull intersection output.*

Table 2: *Time and output surface comparison for the computation of a torus visual hull.*

Framework	Time	#Vert	#Face	#Tri
CGAL (triangles)	7.519s	1480	708	2960
CGAL (quads)	1m15s	780	362	1560
GTS (triangles)	Failed: self-intersecting error			
GTS (quads)	Failed: non-triangular inputs			
Proposed (triangles)	0.225s	1480	708	2960
Proposed (quads)	0.084s	780	362	1560

5.1.2. Multiple Visual Cones

Our framework is also able to compute visual hull as the intersection tree of multiple viewing cones issued from the different viewpoint of a multi-view video setup. We experimented this possibility using a synthetic reference dataset.

5.1.2.1. Torus 6 Cams: We computed the visual hull of a torus seen by 6 cameras positioned around a sphere. In this case the viewing cones (see Fig. 7) are created using the silhouettes seen by the virtual cameras (as described in section §3). For comparison we used two type of input cones, one with triangulated sides, the other with quads on the side. When possible we compared the computation time and complexity of the output with CGAL and GTS (see table 2). Note the gain in speed and primitive count of using quads against triangles. Again our method outperforms CGAL in computation time. Remarkably the outputs of our algorithm matches that of CGAL with this case of identical inputs. Note that CGAL exhibited difficulties to manipulate quad inputs with the default filtered kernel which failed to load inputs. We have been successful in loading quad meshes using another CGAL geometric kernel, the extended kernel, however this kernel is slower, resulting in slower timings for quad meshes. Our algorithm had no issue in dealing with both. GTS intersections failed to produce outputs for this dataset, again due to internal robustness issue.

5.1.3. Comparison

In both cases, we clearly see the advantage of being able to deal with n-gons facets as it reduce the number of primitives to test when going down the tree of intersections. For example with GTS, due to the triangulation of the facets being done between each level of the operation tree, the number of triangles in the output model increases exponentially when going down the operation tree. We suspects that it is the reason why GTS is not able to perform an operation tree involving more than 4 input models. The intermediate triangulation creates very small triangles which may cause ambiguity, such as the self-intersection case we encountered.

Regarding computation time our framework is generally an order of magnitude faster than CGAL with the inputs tested.

Our framework is still slower than GTS in most cases but as mentioned earlier GTS uses a kd-tree optimization which explains the speedup, speedup that would decrease when intersecting many different cones together due to the complexity of the intermediate surfaces created by GTS. We are confident that with few optimizations and parallelization in our framework we could over perform GTS computation time.

Regarding primitive count, our algorithm consistently produces the leanest models than CGAL and GTS, with CGAL only matching our algorithm when able to deal with identical inputs. All in all, our algorithm was the most tolerant to input conditions and produced a correct result on every dataset tested as opposed to GTS.

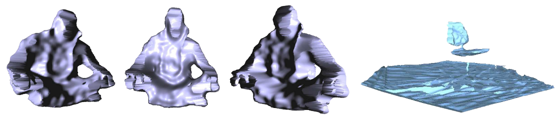
Figure 9: *Left: View from different angle of a man sitting in lotus pose as the intersection of 3 depth cones. Right: A chair in the same setup configuration.*

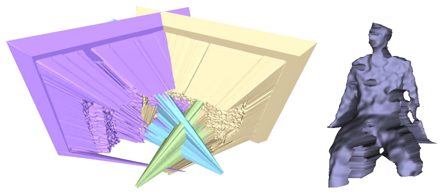
Figure 10: *Left: Two depth cones and two viewing cones intersecting. Right: The 3D visual cropping.*

5.2. Shape Cropping Experiments

We here validate the shape cropping algorithm proposed, by evaluating our framework on real datasets. For this evaluation we used two different setups: (i) a setup with three depth cameras capturing different scenes and producing depth cones, (ii) a setup with two depth and color cameras producing depth cones and viewing cones of a scene.

Note that in this case it was impossible for us to compare with other framework as the depth cones we produce with real data do not have triangular faces on the sides, the top, the bottom and the back face. Therefore both CGAL and GTS are unable to load the meshes for the reason stated above.

5.2.1. Multiple Depth Cones

The first setup is using three Microsoft Kinect producing depth maps of resolution 640x480. We created the depth cones with the method described in section §3. As the full resolution produces really dense meshes we tested our framework with a downsampled version of the depth maps. Table 3 shows the computation time and the input and output mesh complexity regarding the input reasolution used. Here we give a time range and an approximation of the output vertices/faces number for all sequences: as the input cones yield the same number of faces/vertices, the computation time is approximatively identical for all sequences. Figure 8 shows the 3 depth cones and their intersection for a man sitting on a chair while Fig. 9 shows results on two other sequences: a man sitting in a lotus pose and a chair. The resulting meshes are tested watertight and manifold. The shapes obtained are of reasonable quality given the low number of inputs. Note that the quadratic behaviour explains the large execution times for the highest resolution, which would largely improve with a tree-based intersection search implementation.

5.2.2. Multiple Depth and Visual Cones

The second setup is using two Microsoft Kinect's, but here we use both depth maps and color images of resolution 640x480. We created the depth and the viewing cones with the method described in section §3 and downsampled the depth maps to 320x240. We also performed background subtraction on the color images to get a silhouette of the region of interest.

We arranged the operation tree so that at the first step a depth cone intersects a viewing cone coming from the other camera in order to reduce the number of edge-triangle operations. The total time for the 3 intersections is 5m55s and the output model has 12K vertices and 24K triangles.

Figure 10 shows the two depth cones and the two viewing cones intersecting and intersection result for a man sitting on a table. Again the algorithm obtains reasonable shapes, watertight and manifold, given the number of cameras used.

6. Discussion

We have presented a new algorithm for approximate, surface-based surface computation of depth and color camera inputs. To this goal, we propose a new, simple polyhedral combination algorithm shown to outperform CGAL and GTS for the purpose of dealing with noisy visual data. We validate both the geometric algorithm and shape cropping concepts on synthetic and real datasets. Future work will include speed improvement of the algorithm, which can be reasonable be expected to outperform the GTS implementation timewise. Improvements over the depth cone geometry can be done by filtering the depth map for sparsity of the representation, which should lead to large improvements in the primitive counts of depth cones. The work is promising both for the perspective of fast surface-based modeling for acquisition setups with depth and color cameras, and for the geometric algorithm presented which we are confident will find use for other applications.

References

[AFM*ne] Akbarzadeh A., Frahm J.-M., Mordohai P., Clipp B., Engels C., Gallup D., Merrell P., Phelps M., Sinha S., Talton B., Wang L., Yang Q., Stewenius H., Yang R., Welch G., Towles H., Nister D., Pollefeys M.: Towards urban 3d reconstruction from video. In *3D Data Processing, Visualization, and Transmission, Third International Symposium on* (June), pp. 1–8. 1

[BGM06] Bogomjakov A., Gotsmann C., Magnor M.: Free-viewpoint video from depth cameras. In *Proc. Vision, Modeling and Visualization (VMV) 2006* (Nov. 2006), pp. 89–96. 2, 3

[DMZC10] Dal Mutto C., Zanuttigh P., Cortelazzo G. M.: A probabilistic approach to tof and stereo data fusion. In *Proceeding of the 3DVPT'2010* (June 2010), pp. 3089–3096. 2

Table 3: *Time and input/output surface comparison for the three depth cameras setup with different input resolution.*

		Input depth cone		Output surface	
Downsampled resolution	Time	#Vert	#Face	#Vert	#Tri
80x60	17-19s	76.8K	152.5K	30-35K	60-70K
160x120	11-13m	19.2K	37.8K	7-11K	5-8K
320x240	105-107m	4.8K	9.3K	3-5K	2-4K

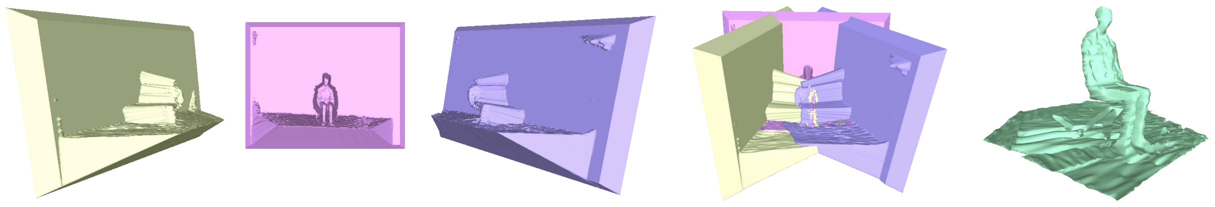
Figure 8: *In order: the 3 depth cones, first apart and then combined, and their intersection.*

[DP11] Delaunoy A., Prados E.: Gradient Flows for Optimizing Triangular Mesh-based Surfaces: Applications to 3D Reconstruction Problems dealing with Visibility. *International Journal of Computer Vision* (2011). 2

[DRR95] Debled-Rennesson I., Reveillès J.: A linear algorithm for segmentation of digital curves. *International Journal of Pattern Recognition and Artificial Intelligence 9*, 4 (1995), 635–662. 3

[FB09] Franco J.-S., Boyer E.: Efficient Polyhedral Modeling from Silhouettes. *IEEE Transactions on Pattern Analysis and Machine Intelligence 31*, 3 (2009), 414–427. 3

[GFP08] Guan L., Franco J.-S., Pollefeys M.: 3D Object Reconstruction with Heterogeneous Sensor Data. In *International Symposium on 3D Data Processing, Visualization and Transmission* (Atlanta, United States, 2008). 1, 2

[GK99] Gutmann J., Konolige K.: Incremental mapping of large cyclic environments. In *Proc. IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)* (Monterey, California, 1999), p. 318–325. 1, 2

[HHAL11] Hansard M., Horaud R. P., Amat M., Lee S.: Projective alignment of range and parallax data. In *Proceeding of the IEEE Conference on Computer Vision and Pattern Recognition* (Colorado Springs, CO, June 2011), IEEE Computer Society Press, pp. 3089–3096. 2

[HKM07] Hachenberger P., Kettner L., Mehlhorn K.: Boolean operations on 3d selective nef complexes: Data structure, algorithms, optimized implementation and experiments. *Comput. Geom. Theory Appl. 38*, 1-2 (Sept. 2007), 64–99. 2, 3

[Hof89] Hoffmann C. M.: The problems of accuracy and robustness in geometric computation. *IEEE Computer* (1989), 31–41. 2, 3

[KCTT08a] Kim Y. M., Chan D., Theobalt C., Thrun S.: Design and calibration of a multi-view tof sensor fusion system. *Computer Vision and Pattern Recognition Workshop 0* (2008), 1–7. 2

[KCTT08b] Kim Y. M., Chan D., Theobalt C., Thrun S.: Design and calibration of a multi-view tof sensor fusion system. In *IEEE CVPR Workshop on Time-of-flight Computer Vision* (Anchorage, USA, 2008), IEEE, pp. 1–7. 2

[KTD* 4] Kim Y. M., Theobalt C., Diebel J., Kosecka J., Miscusik B., Thrun S.: Multi-view image and tof sensor fusion for dense 3d reconstruction. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on* (27 2009-Oct. 4), pp. 1542–1549. 1, 2

[Lau94] Laurentini A.: The visual hull concept for silhouette-based image understanding. *IEEE Trans. Pattern Anal. Mach. Intell. 16*, 2 (1994), 150–162. 3

[ME88] Matthies L., Elfes A.: *Integration of Sonar and Stereo Range Data Using a Grid-Based Representation.* IEEE Comput. Soc. Press, 1988, pp. 727–733. 2

[MT97] MÃŰller T., Trumbore B.: Fast, minimum storage ray-triangle intersection. *journal of graphics, gpu, and game tools 2*, 1 (1997), 21–28. 4

[NIH*11] Newcombe R. A., Izadi S., Hilliges O., Molyneaux D., Kim D., Davison A. J., Kohli P., Shotton J., Hodges S., Fitzgibbon A. W.: Kinectfusion: Real-time dense surface mapping and tracking. In *ISMAR'11* (2011), pp. 127–136. 1, 2

[RDP*11] Reynolds M., Doboš J., Peel L., Weyrich T., Brostow G. J.: Capturing time-of-flight data with confidence. In *CVPR* (2011). 2

[RHHL02] Rusinkiewicz S., Hall-Holt O., Levoy M.: Real-time 3d model acquisition. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 438–446. 2

[WB10] Wilson A. D., Benko H.: Combining multiple depth cameras and projectors for interactions on, above and between surfaces. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology* (New York, NY, USA, 2010), UIST '10, ACM, pp. 273–282. 2

[ZBH11] Zaharescu A., Boyer E., Horaud R. P.: Topology-adaptive mesh deformation for surface evolution, morphing, and multi-view reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence 33*, 4 (April 2011), 823–837. 2