ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/261399182

CPU-Based Real-Time Surface and Solid Voxelization for Incomplete Point Cloud

CONFERENCE PAPER · AUGUST 2014

DOI: 10.1109/ICPR.2014.475

CITATIONS
2

READS
160

2 AUTHORS:

Frederic Garcia
IEE S.A.
37 PUBLICATIONS **80** CITATIONS
SEE PROFILE

Björn Ottersten
University of Luxembourg
563 PUBLICATIONS **10,639** CITATIONS
SEE PROFILE

All in-text references underlined in blue are linked to publications on ResearchGate, letting you access and read them immediately.

Available from: Frederic Garcia
Retrieved on: 29 February 2016

CPU-Based Real-Time Surface and Solid Voxelization for Incomplete Point Cloud

Frederic Garcia and Björn Ottersten

Interdisciplinary Centre for Security Reliability and Trust (SnT)
University of Luxembourg
{frederic.garcia, bjorn.ottersten}@uni.lu

***Abstract*—This paper presents a surface and solid voxelization approach for incomplete point cloud datasets. Voxelization stands for a discrete approximation of 3-D objects into a volumetric representation, a process which is commonly employed in computer graphics and increasingly being used in computer vision. In contrast to surface voxelization, solid voxelization not only set those voxels related to the object surface but also those voxels considered to be inside the object. To that end, we first approximate the given point set, usually describing the external object surface, to an axis-aligned voxel grid. Then, we slice-wise construct a shell containing all surface voxels along each grid-axis pair. Finally, voxels inside the constructed shell are set. Solid voxelization results from the combination of all slices, resulting in a watertight and gap-free representation of the object. The experimental results show a high performance when voxelizing point cloud datasets, independently of the object's complexity, robust to noise, and handling large portions of data missing.**

Keywords*—*voxelization, point cloud, curve-skeleton, skeletonization, distance transform, real-time.

I. Introduction

Voxelization is an indispensable stage in computational sciences and particularly in computer graphics to model geometric scenes into their equivalent discrete voxel-based representations [1], [2]. By doing so, complex scenes with thousands of polygons are approximated to a discrete 3-D voxel grid, facilitating the task of many computer graphics algorithms such as volume visualization or object collision [3], [4]. As opposed to meshes in computer graphics, object boundaries in computer vision are described in a three-dimensional coordinate system by 3-D points, usually defined by their Cartesian coordinates x, y, z, with respect to a given origin. Surface voxelization is typically rather slow when treating polygonal meshes since not only those voxels corresponding to mesh vertexes are set but also the ones intersected by mesh edges. In contrast, voxelizing a set of 3-D points or point cloud data set directly results from setting those voxels containing at least one 3-D point [5]. This straightforward operation not only reduces the point cloud complexity but enables for fast geometric processing, *i.e.,* fast data access and manipulation.

In this paper, we address the ill-posed problem of solid voxelization, *i.e.,* setting all voxels considered to be inside the object. By doing so, further processing steps such as the computation of the volumetric distance field [6] or complex 3-D descriptors, *i.e.,* curve-skeletons [7], are significantly simplified. A common strategy for solid voxelization of point clouds is to transform the point set to a polygon mesh and then to apply fast mesh voxelization approaches based on graphic hardware [4], [8], [9], or mesh voxelization approaches that propagate the state of a voxel to its surrounding visible voxels [3]. Instead, we propose a new surface and solid point cloud voxelization approach resulting in a watertight and gap-free volumetric representation, avoiding imprecise or even incorrect set voxel due to wrong point cloud to mesh conversion, which is prominent to happen in the presence of large amount of missing data.

The remainder of the paper is organized as follows: Section II covers the literature review on surface and solid voxelization approaches for point cloud datasets. In Section III we introduce a slice-wise surface and solid voxelization approach for scanned point cloud datasets. In Section IV, we evaluate the proposed voxelization approach on our own synthetic 3-D point sets and on the Stanford 3-D data sets. Finally, concluding remarks are given in Section V.

II. Background and Related Work

Analogous to a pixel, which represents a value in a 2-D grid, a voxel is a volumetric element that represents values in a regular 3-D grid. Voxelization is thus the approximation of a given point cloud $\mathcal{P}$ to a voxel-based volumetric representation $\mathcal{V}$ of the occupied space [5]. In practice, point clouds are usually created from 3-D scanners such as Time-of-Flight or RGB-D cameras and thus, all points $\mathbf{p}_i \in \mathcal{P}$ only describe the underlying scanned surface of the object. Consequently, the voxelization of a scanned point cloud, to which we refer as *surface voxelization*, results in a 3-D grid of voxels where only those voxels describing the object surface are set. In general, surface voxelization is addressed by spatial decomposition techniques such as kd-trees or octrees [5], [10], [11], from which a resulting volumetric representation enables fast access to point locations and to their corresponding neighbors, *i.e.,* without re-computing distances between each other every time. However, more complex geometric processing, *e.g.,* extracting 3-D object descriptors such as curve-skeletons [7], [12], or computing volumetric distance fields [6]; need to consider not only those voxels representing the external object surface but also those voxels considered to be interior to the object, *i.e., solid voxelization*. Voxelization is a necessary step in graphic computing for data simplification, visibility determination, or collision detection. Consequently, fast and robust octree-based voxelization approaches can be directly applied to point

This work was supported by the National Research Fund, Luxembourg, under the CORE project C11/BM/1204105/FAVE/Ottersten.

cloud datasets after being transformed to polygonal meshes. Octree representations are intended for fast access to voxel locations, with a major advantage of being an organized and space-efficient data structure. Schwarz et al. [4] proposed a novel octree-based sparse voxelization approach that uses data-parallel algorithms on graphics hardware. In their octree-based representation, voxels close to the solid's boundary are stored by finest-level voxels whereas uniform interior and exterior regions are represented by coarser-level voxels (see Fig. 1), addressing the problem of high memory consumption of high-resolution grids. In [3], the authors proposed an alternative polygonal scene to octree-based voxelization approach. In this case, the authors propagate the status of a voxel (inside/outside the object) to the surrounding visible voxels. However, the conversion from scanned point clouds to polygonal meshes used to be imprecise or even incorrect [13] and thus, polygonal mesh-based approaches [3], [4], [8] may fail. Indeed, self occlusions, changes in ambient light conditions, or object material, lead to large portions of data missing with point density variations, *i.e.,* the closest the object surface to the sensing system, the higher the point density, which hampers the point set to polygonal mesh conversion. Besides, this situation is aggravated when modeling complex and dynamic scenes using a multi-view system. In this case, the resulting point cloud after registration has to be re-sampled to account for point redundancy as well as for registration inaccuracies. An alternative approach for voxelizing incomplete point clouds is constructing a convex hull [14]. Numerous algorithms such as Gift Wrap [15] or QuickHull [16] can be found in the literature and are commonly used to compute the convex hull of a finite set of points with an optimal time complexity. However, convex hull-based approaches fail when setting interior voxels of a concave object, *i.e.,* a hand. Next, we propose a new surface and solid voxelization approach to represent scanned point cloud datasets in real-time.

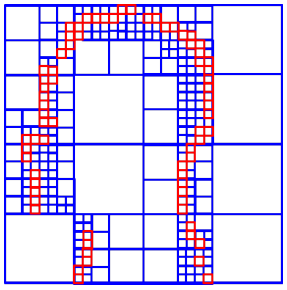


Fig. 1: Surface and solid voxelization using octree representation.

III. Proposed Approach for Surface and Solid Voxelization

Given a point cloud $\mathcal{P}$ represented in a three-dimensional Euclidean space $\mathbf{E}^3 \equiv \mathbf{p}(x,y,z)|1 \leq x \leq X, 1 \leq y \leq Y, 1 \leq z \leq Z$, and describing a set of 3-D points, we first create a bounding 3-D grid of voxels $\mathcal{V}$ with size $I \times J \times K$, *i.e.,* $\mathcal{V} \equiv \mathbf{v}(i,j,k)|1 \leq i \leq I, 1 \leq j \leq J, 1 \leq k \leq K$ with I, J, K being positive integers and $\mathbf{v}(i,j,k) \in 0,1$, *i.e.,* non-object and object voxel respectively. In image visualization, non-object voxels will not be displayed. $\mathcal{V}$ has to be big enough to fit the entire point set, but no bigger to avoid inefficiency, as shown in Fig. 2. To do so, we iterate through all points $\mathbf{p}_i = (p_x, p_y, p_z) \in \mathcal{P}$ to find the minimum $\mathbf{q}_{min} = (p_{x_{min}}, p_{y_{min}}, p_{z_{min}})$ and maximum $\mathbf{q}_{max} = (p_{x_{max}}, p_{y_{max}}, p_{z_{max}})$ 3-D point coordinates to which we constraint the size of $\mathcal{V}$. Note that the dimensions of $\mathcal{V}$ along each orthogonal axis might not coincide. We fix the voxel width to λ, which defines the size of each voxel side and thus, the resolution and accuracy of the discrete voxel representation. The choose of λ depends on further geometric processing, *e.g.,* to compute the curve-skeleton of a human body, a voxel of 1 cubic centimeter in size ensures that important details like the fingers are not lost. Thus, λ needs to be chosen as the best trade-off between accuracy and processing time.

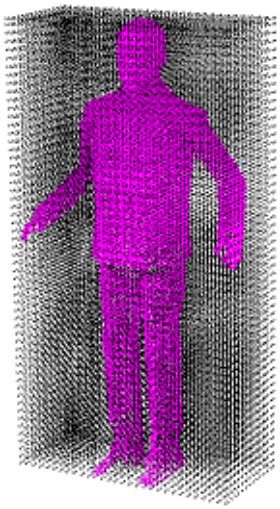


Fig. 2: In black color, the bounding 3-D grid of voxels $\mathcal{V}$ with fixed width ($\lambda = 3$ cm). In purple color, the object points.

A. Surface Voxelization

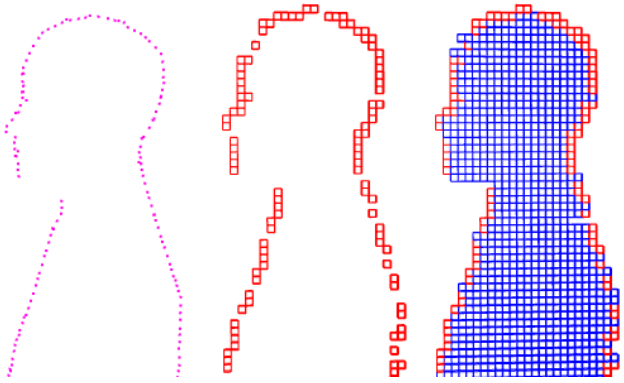
We propose to map each $\mathbf{p}_i = (p_x, p_y, p_z)$ to the closest voxel $\mathbf{v}_i = (v_i, v_j, v_k)$. To do so, we first translate $\mathcal{P}$ to the origin of $\mathcal{V}$ using the offset vector $\mathbf{q}_{min}$. Then, we solidify a shell representing the external object surface by approximating all $\mathbf{p}_i$ to the geometric center of their respective voxel $\mathbf{v}_i$, *i.e.,*

$$\mathbf{v}_i = \begin{cases} 1 & \text{if } \mathbf{p}_i \in [\mathbf{v}_i, \mathbf{v}_i + \lambda] \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

An alternative approach to represent the underlying surface more accurately considers the centroid of the n points contained in a voxel $\mathbf{v}_i$ as voxels coordinates, *i.e.,*

$$\mathbf{v}_i = \frac{1}{n} \cdot \sum_{i=1}^{n} \mathbf{p}_i. \quad (2)$$

However, the gain in accuracy is not worth compared to the increase in processing time (for a well chosen λ). In Fig. 3a we show the scanned $\mathbf{p}_i$ (in purple color) contained in a selected voxel-slice of $\mathcal{V}$. Their surface voxelization is shown in Fig. 3b (surface object voxels in red color), whereas the solid voxelization is shown in Fig. 3c (interior object voxels in blue color).



(a) Scanned points. (b) Surface voxelization. (c) Solid voxelization.

Fig. 3: Selected voxel-slice from $\mathcal{V}$ ($\lambda = 1$ cm) to illustrate (a) the scanned points $\mathbf{p}_i$, (b) their surface voxelization, and (c) their solid voxelization.

B. Solid Voxelization

Although surface voxelization is pretty straightforward, solid voxelization presents a higher complexity entailing to wrong results in the presence of shell holes. Furthermore, solid voxelization is known to be time consuming [1], [3]. In the following, we propose a high-performance 2-step solid voxelization approach based on the use of the 3-D grid of voxels $\mathcal{V}$ introduced in Section III.

1) Step 1, complete shell: In order to avoid wrong solid voxelization results, we first solidify the shell of surface voxels representing the object surface (previously set during the surface voxelization stage). Let us consider the slices from J and K axes of $\mathcal{V}$, *i.e.,* $\mathcal{V}_i^{jk}$, with $i = 0, 1, \cdots, I-1$, as illustrated in Fig.4. For each slice $\mathcal{V}_i^{jk}$, we create an array of voxels $\mathcal{A}$ containing all surface voxels. Then we sort them according to their relative distances using the method

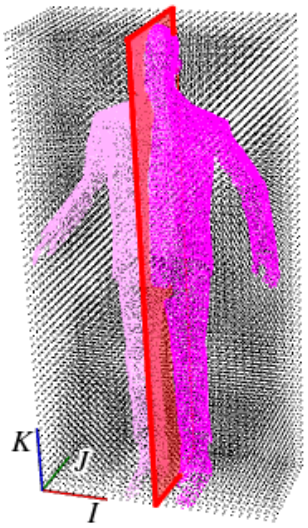


Fig. 4: Selected voxel-slice from the J and K axes $\mathcal{V}_i^{jk}$ (in red color).

Algorithm 1 Sorting surface voxels

```
1: new(A*)
2: new(A')
3: forward ← true
4: currVoxel ← A[0]
5: nextVoxel ← NULL
6: erase(A,A[0])                          ▷ remove A[0] from A
7: insert(A',A[0])                        ▷ appends A[0] to the front of A'
8: while !empty(A) do
9:     dist ← inf
10:    for i ← 1 ... |A| − 1 do
11:        distAux ← d(currVoxel, A[i])
12:        if distAux < dist then
13:            dist ← distAux
14:            nextVoxel ← A[i]
15:        end if
16:    end for
17:    if dist < ε then
18:        if forward then
19:            pushBack(A',nextVoxel)  ▷ appends nextVoxel to the end of A'
20:        else
21:            insert(A',nextVoxel)
22:        end if
23:        currVoxel ← nextVoxel
24:        erase(A, nextVoxel)
25:    else
26:        if forward then
27:            currVoxel ← A'[0]
28:            forward ← false
29:        else
30:            insert(A*,A')
31:            clear(A')                  ▷ empty A'
32:            forward ← true
33:            currVoxel ← A[0]
34:            erase(A,A[0])
35:            insert(A',A[0])
36:        end if
37:    end if
38: end while
39: if !empty(A') then
40:    insert(A*,A')
41: end if
42: return A*
```

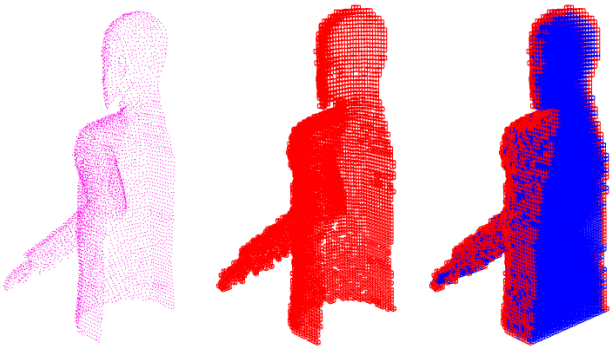
proposed in Algorithm 1, in which $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2$ and $\mathcal{A}^* = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n\}$, being n the total number of arrays with sorted voxels. ϵ is the maximum allowed distance between two nearest voxels. Its value is related to the density of the initial point cloud $\mathcal{P}$. Too small values of ϵ yield to non-complete shells, whereas too big ϵ values might merge non-desired object parts, *e.g.,* both legs or the arms and the torso, in the case of a human model. Finally, we iterate through each array in $\mathcal{A}^*$ setting those non-object voxels intersected by the line segment bounded by each voxel pair ($\mathbf{v}_i$,$\mathbf{v}_{i+1}$).

2) Step 2, shell filling: Once the shell is complete, we proceed by updating their interior voxels to object voxels (1 value). To do so, we iterate row-by-row each voxel-slice $\mathcal{V}_i^{jk}$ propagating the voxel state (object or non-object voxel) to the next voxels as described in Algorithm 2. By using this pretty

Algorithm 2 Shell filling

```
i ← n                                              ▷ n is the selected slice
for k ← 0 ... K − 1 do
    setVoxel ← false
    value ← 0
    for j ← 1 ... J − 1 do
        if setVoxel == true then
            v(i, j, k) ← 1
        end if
        value ← v(i, j, k)
        if value == 1 and value != v(i, j, k − 1) then
            if setVoxel == false then                  ▷ transition starts
                setVoxel ← true
            else                                       ▷ transition ends
                setVoxel ← false
            end if
        end if
    end for
end for
```

simple scan line algorithm, interior object voxels are efficiently set. The same procedure is repeated for $\mathcal{V}^{ik}$ and $\mathcal{V}^{ij}$ slices. The final result is shown in Fig. 5c.



(a) Scanned points. (b) Surface voxelization. (c) Solid voxelization.

Fig. 5: View of half of the human model to show (a) the scanned points, (b) their surface voxelization, and (c) their solid voxelization.

IV. Experimental Results

In the following we test the proposed surface and solid voxelization approach on our own synthetic data and on the Stanford 3D Scanning Repository [1]. All reported results have been obtained using a Mobile Intel® QM67 Express Chipset with an integrated graphic card Intel® HD Graphics 3000. The proposed approach has been implemented in C++ language using the OpenCV [17] and PCL [18] libraries. Our synthetic data has been generated using V-Rep [19], a very versatile robot simulator tool in which the user can replicate real scenarios. Fig. 6 shows the simulated scene in which we have replicated the multi-view sensing system of our computer vision laboratory. For the synthetic data evaluation we have used the Bill human model with four different configurations,

[1] Stanford 3D Scanning Repository, http://graphics.stanford.edu/data/3Dscanrep/

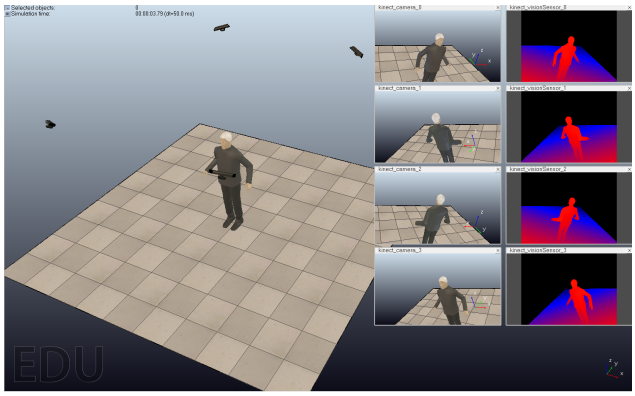
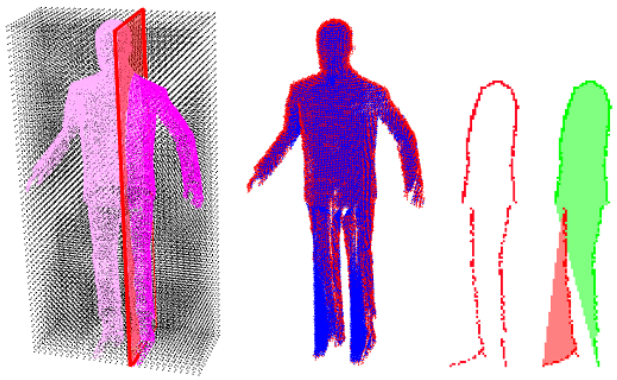


Fig. 6: V-Rep scene to generate synthetic data.

i.e., standing, working, walking, and sitting, as shown in first column of Fig. 8. Table I reports the performance evaluation for the proposed surface and solid voxelization approach on both synthetic and Stanford data. Some visual results on synthetic data are shown in the last two columns of Fig. 8. Visual results for the Stanford 3-D models are shown in the last two rows of Fig. 9. Reported values in Table I show a high performance where most of the processing time is dedicated on sorting surface voxels for shell completeness. An increase on performance as well as on space-efficiency has been obtained by adapting the size of $\mathcal{V}$ to the dimension of the given point set.

When sorting surface voxels using Algorithm 1, multiple shells from a given $\mathcal{V}^{jk}$, $\mathcal{V}^{ik}$, or $\mathcal{V}^{ij}$ slice, can be obtained. This occurs when the distance between two nearest surface voxels is bigger than ϵ, a parameter that we have set to 5 cm in order to do not merge non-desired body parts. However, shells that would cover a single body part might be split into two or more shells, which yields to a wrong shell filling, as illustrated in Fig. 7d. From our tests, this situation occurs when voxelizing the Bill human model using vertical $\mathcal{V}$ slices, *i.e.,*



(a) (b) (c) (d)

Fig. 7: Example of wrong shell filling when considering $\mathcal{V}^{jk}$ slices. (a) In yellow, selected $\mathcal{V}_i^{jk}$. (b) Solid voxelization using $\mathcal{V}^{jk}$ slices. (c) Surface voxelization of $\mathcal{V}_i^{jk}$. (d) Wrong shell filling (in red color).

TABLE I: Running time of the proposed surface and solid voxelization approach on both, the synthetic data shown in Fig. 8 and the Stanford 3-D models shown in Fig. 9, along with the number of resulting voxels depending on the processed voxel-slice of $\mathcal{V}$ (we set ϵ to 5 cm for the synthetic data and to 2 cm for the Stanford data. Units are in ms).

Model	$\mathcal{V}$ size	Selected voxel-slice	Nb. surface voxels	Nb. solid voxels	Running time		
					Surface voxelization	Solid voxelization	Sort surface voxels
Standing Bill (23650 points)	$88 \times 42 \times 168$	$\mathcal{V}^{ijk}$	15655	68234	3.2	43.6	290.3
		$\mathcal{V}^{ij}$		67268		16.3	38.2
		$\mathcal{V}^{ik}$		47574		14.0	160.3
		$\mathcal{V}^{jk}$		53200		13.3	91.8
Working Bill (21188 points)	$54 \times 64 \times 161$	$\mathcal{V}^{ijk}$	14406	64614	2.9	129.9	225.0
		$\mathcal{V}^{ij}$		62722		45.0	35.1
		$\mathcal{V}^{ik}$		51413		42.4	97.3
		$\mathcal{V}^{jk}$		48591		42.5	92.6
Walking Bill (21293 points)	$54 \times 85 \times 168$	$\mathcal{V}^{ijk}$	14039	61825	3.0	42.8	204.2
		$\mathcal{V}^{ij}$		60162		15.7	33.4
		$\mathcal{V}^{ik}$		47913		15.0	92.9
		$\mathcal{V}^{jk}$		53618		12.1	77.9
Sitting Bill (17085 points)	$54 \times 72 \times 127$	$\mathcal{V}^{ijk}$	12238	61016	2.4	34.7	175.5
		$\mathcal{V}^{ij}$		57722		12.8	36.9
		$\mathcal{V}^{ik}$		45797		12.5	72.0
		$\mathcal{V}^{jk}$		75564		9.4	66.6
Stanford Bunny (34519 points)	$79 \times 61 \times 79$	$\mathcal{V}^{ijk}$	15874	85110	4.7	91.6	265.1
		$\mathcal{V}^{ij}$		78941		31.3	85.0
		$\mathcal{V}^{ik}$		68697		30.9	106.8
		$\mathcal{V}^{jk}$		77662		29.4	73.3
Dragon (84969 points)	$103 \times 47 \times 74$	$\mathcal{V}^{ijk}$	24089	42606	11.9	93.7	660.9
		$\mathcal{V}^{ij}$		37355		30.6	192.2
		$\mathcal{V}^{ik}$		29158		24.1	338.2
		$\mathcal{V}^{jk}$		32130		39.0	130.5
Asian Dragon (141644 points)	$162 \times 109 \times 73$	$\mathcal{V}^{ijk}$	34531	38332	18.8	303.2	929.7
		$\mathcal{V}^{ij}$		30882		103.0	372.0
		$\mathcal{V}^{ik}$		28579		101.1	376.1
		$\mathcal{V}^{jk}$		29713		99.1	181.6

$\mathcal{V}^{ik}$ or $\mathcal{V}^{jk}$, with gaps of data above ϵ cm. We note that the best solid voxelization results from $\mathcal{V}^{ij}$ slices. However, shell filling from $\mathcal{V}^{ij}$ might also fail when voxelizing a human model lying on the floor. We thus propose to combine the individual solid voxelization from each $\mathcal{V}^{ij}$, $\mathcal{V}^{ik}$, and $\mathcal{V}^{jk}$ slices in a unique volumetric grid $\mathcal{V}^{ijk}$.

V. Conclusion

A scheme to compute the surface and solid voxelization of incomplete point cloud datasets has been described. Our main contribution is in using an axis-aligned 3-D grid of voxels to which we approximate the given point cloud set. Surface voxelization directly results from approximating the point set to the voxel grid. By doing so, we reduce the amount of data to be treated, registration inaccuracies and noise within depth measurements. We solidify the volumetric grid slice-wise, *i.e.,* we first determine a closing shell considering its surface voxels and then we fill it by updating its interior voxels. The combination of the filled shells results in an accurate surface and solid voxelization of the given point set, independently of its complexity. The experimental evaluation shows a high-performance, which makes it practical for further geometric processing such as the computation of distance fields or extracting 3-D object descriptors such as curve-skeletons. An extra runtime improvement can be obtained by launching the shell filling procedure for each grid axes pair in parallel, which would reduce its processing time approximatively by a factor of 3 (for an equally dimensioned $\mathcal{V}$).

References

[1] D. Cohen-Or and A. Kaufman, "Fundamentals of surface voxelization," *Graphical Models and Image Processing*, vol. 57, no. 6, pp. 453–461, 1995.

[2] A. Kaufman, D. Cohen, and R. Yagel, "Volume graphics," *IEEE Computer*, vol. 26, no. 7, pp. 51–64, 1993.

[3] D. Haumont and N. Warze, "Complete polygonal scene voxelization," *Journal of Graphics Tools*, vol. 7, no. 3, pp. 27–41, 2002.

[4] M. Schwarz and H.-P. Seidel, "Fast parallel surface and solid voxelization on gpus," *ACM Trans. Graph.*, vol. 29, no. 6, pp. 179:1–179:10, 2010.

[5] R. B. Rusu, "Semantic 3d object maps for everyday manipulation in human living environments," Ph.D. dissertation, Tecnische Universitatet Muenchen, 2009.

[6] G. J. Grevera, "Distance transform algorithms and their implementation and evaluation," in *Deformable Models*, ser. Topics in Biomedical Engineering. International Book Series. Springer New York, 2007, pp. 33–60.

[7] C. Arcelli, G. Sanniti di Baja, and L. Serino, "Distance-driven skeletonization in voxel images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 4, pp. 709–720, 2011.

[8] R. Bakken and L. Eliassen, "Real-time 3d skeletonisation in computer vision-based human pose estimation using gpgpu," in *IEEE International Conference on Image Processing Theory, Tools and Applications (IPTA)*, 2012, pp. 61–67.

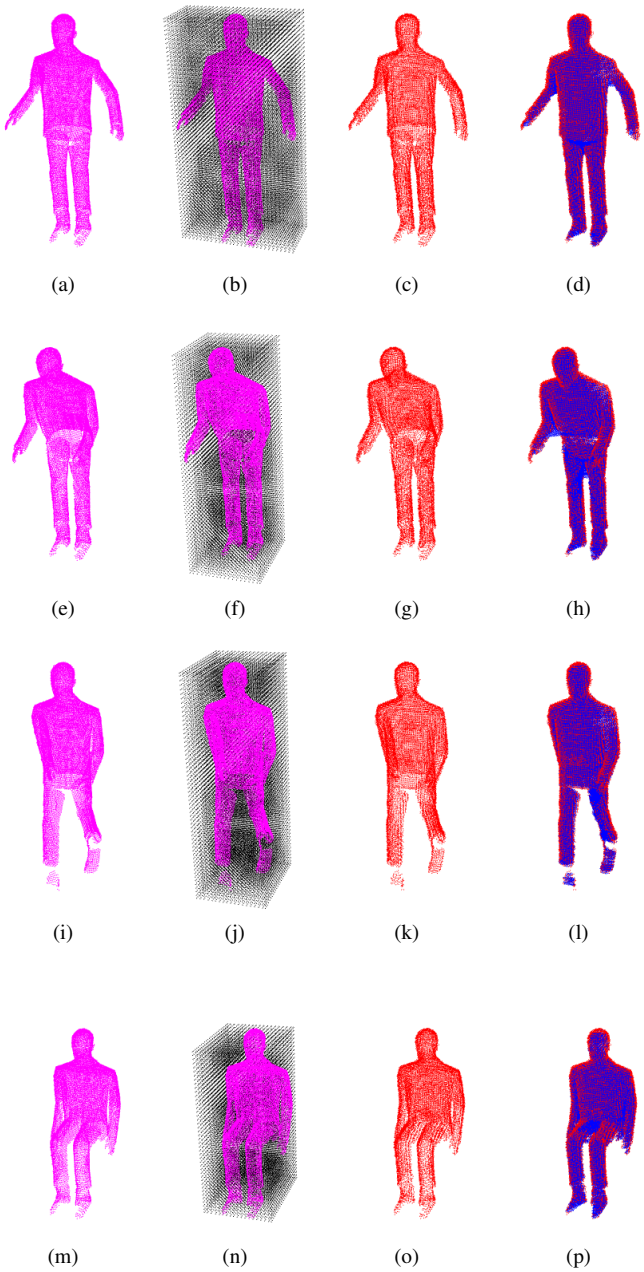


Fig. 8: Surface and solid voxelization on synthetic data. $1^{st}row$: Standing Bill. $2^{nd}row$: Working Bill. $3^{rd}row$: Walking Bill. $4^{th}row$: Sitting Bill. $1^{st}col.$: point set. $2^{nd}col.$: volumetric grid. $3^{rd}col.$: surface voxelization. $4^{th}col.$: solid voxelization.

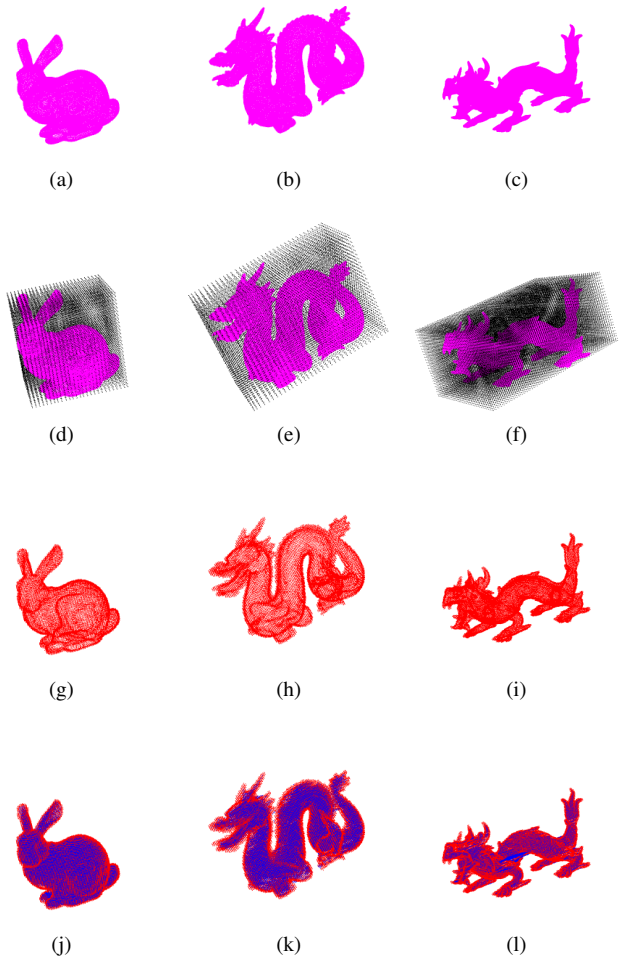


Fig. 9: Surface and solid voxelization using the Stanford 3-D Scanning Repository. $1^{st}col.$: Stanford Bunny. $2^{nd}col.$: Dragon. $3^{rd}col.$: Asian Dragon. $1^{st}row$: point set. $2^{nd}row$: volumetric grid. $3^{rd}row$: surface voxelization. $4^{th}row$: solid voxelization.

[9] D. Zhao, C. Wei, B. Hujun, Z. Hongxin, and P. Qunsheng, "Real-time voxelization for complex polygonal models," in *IEEE Pacific Conference on Computer Graphics and Applications*, 2004, pp. 43–50.

[10] C. Crassin and S. Green, "Octree-based sparse voxelization using the gpu hardware rasterizer," in *OpenGL Insights*. CRC Press, Patrick Cozzi and Christophe Riccio, 2012.

[11] X. Wu, W. Liu, and T. Wang, "A new method on converting polygonal meshes to volumetric datasets," in *IEEE International Conference on Robotics, Intelligent Systems and Signal Processing*, vol. 1, 2003, pp. 116–120.

[12] N. Cornea, D. Silver, and P. Min, "Curve-Skeleton Properties, Applications, and Algorithms," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 3, pp. 530–548, 2007.

[13] S. Kansal, J. Madan, and A. Singh, "A systematic approach for cad model generation of hole features from point cloud data," in *IEEE 3rd International Advance Computing Conference (IACC)*, 2013, pp. 1385–1393.

[14] F. P. Preparata and M. I. Shamos, "Convex hulls: Basic algorithms," in *Computational Geometry*. Springer New York, 1985.

[15] T. H. Cormen, C. E. Leiserson, R. L. R. Rivest, and C. Stein, "Finding the convex hull," in *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2009.

[16] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Trans. Math. Softw.*, vol. 22, no. 4, pp. 469–483, 1996.

[17] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*, 1st ed. O'Reilly Media, 2008.

[18] "Point Cloud Library (PCL)," http://pointclouds.org/, December 2013.

[19] "Virtual robot experimentation platform (v-rep)," http://www.coppeliarobotics.com/, December 2013.