Eighth Hungarian Conference on Computer Graphics and Geometry, Budapest, 2016

Hierarchical Volumetric Fusion of Depth Images for SLAM

László Szirmay-Kalos, Balázs Tóth, and Tamás Umenhoffer

Budapest University of Technology and Economics, Hungary

Abstract

With a moving depth camera 3D objects and scenes can be scanned. The scanning algorithm merges depth images into a common volumetric model that stores the distance field in voxels, and simultaneously tracks the camera based on the already built model. To maintain high resolution while fitting the model to the GPU memory, hierarchical volumetric representation is needed. The paper presents a GPU implementation of a hierarchical fusion approach. When fusing depth images into a 3D volumetric model, a crucial task is to mark macro-cells as empty or as intersected by the noisy surface represented by the depth image. This paper proposes a simple marking algorithm for the GPU implementation of hierarchical volumetric fusion. The method is based on multi-level DDA ray-casting. The GPU implementation is of scattering type, but we also show a solution to avoid atomic writes, which improves performance.

1. Introduction

Depth cameras like Kinect measure the distance of points visible in different pixels. The information of a pixel includes the estimate where the surface is intersected by a ray going through the pixel and also that the volume is empty between the camera and the intersected point along the ray. The distance values are noisy due to the inherent noise of the depth camera, which often measures the time of flight of the light or the phase of the returned infrared signal. Moving the camera around the object or in the scene, geometric information of different noisy images can be fused together into a smooth 3D model. Additionally, the temporary result of the fused data can be used to find where the camera is moved, executing Simultaneous Localization And Mapping (SLAM). A famous approach to depth image fusion is the Curless-Levoy algorithm [3] that results in a distance field of the reconstructed surface. The discretized distance field is represented in a voxel grid. As a uniform grid would either have poor resolution or overflow the GPU memory, hierarchical representation is preferred where higher level macrocells are decomposed to smaller cells and finally to voxels only if the surface intersects the cell [2]. Scene objects can be extracted from the volumetric representation as finding the zero level surface of this distance field. The output can be a point cloud generated by the intersection of the zero level surface and the lattice edges of the grid, or a triangle mesh obtained with the marching cubes algorithm [6, 9].

If the camera does not move, the Curless-Levoy algorithm estimates a *Truncated Signed Distance Function* (TSDF) representing the signed distance between the surface and the grid point along the ray defined by the current pixel. The signed distance is positive in front of the surface and negative behind it. To avoid the interference of surfaces on opposite sides and realizing that farther away from the surface the distance value is unreliable since nearer surface may be found in another direction, the signed distance function is truncated. If a point is outside of the truncation interval and its signed distance is positive, it belongs to the empty region between the camera and the surface. If the camera moves, then the same arithmetic mean is used to update distance values. As it is shown in [3], this corresponds to the reconstruction of the isosurface with minimal squared error where the weighting depends on how long the surface is seen recently from different angles.

2. Previous work

With the application of active sensors larger indoor scenes can be scanned at interactive frame rates [4, 8]. KinectFusion [5, 7] is a real-time GPU implementation of the Curless-Levoy volumetric fusion algorithm. To keep high resolution without extensive memory requirements the volume may be partially streamed out or in the GPU memory [16], or a hierarchical representation can be used. In Scalable KinectFusion a three level hierarchy is maintained [2].

3. The new method

We use a two-level hierarchy to represent the TSDF volume. The higher level grid is called macro-voxel array. A macro-voxel cell may or may not be subdivided depending on whether the surface intersects the macro-voxel. In case of subdivision, a $8 \times 8 \times 8$ resolution micro-voxel array is assigned to the macro-voxel cell.

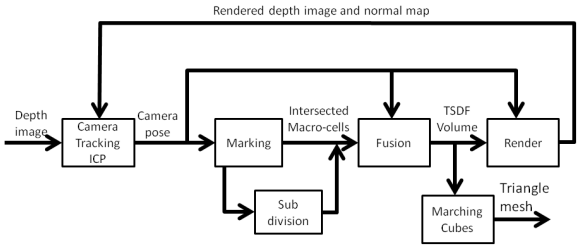

Figure 1: *Algorithm pipeline*

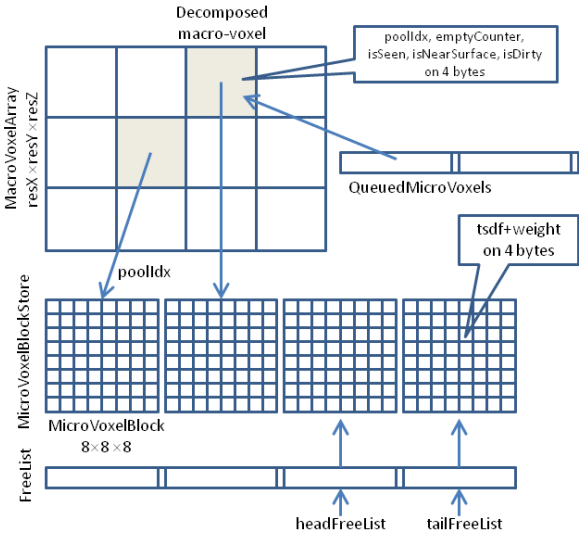

Figure 2: *Organization of the GPU memory that stores the two-level TSDF volume*

The algorithm iteratively executes the following main kernels (Figure 1):

1. *Mark*: Find macro-cells that are empty or affected by the current depth image.
2. *Subdivide*: Assign a micro-voxel block to marked macro-cells that have not been subdivided yet.
3. *Fusion*: Find micro-voxels of marked macro-cells that are empty or affected by the current depth image and fuse, i.e. average their stored TSDF value with the TSDF obtained from the current depth image.
4. *Rendering*: Execute ray casting to render the current estimate of the surface from the current camera also computing the surface normals.
5. *Get depth image*: Read the new depth image from a possibly moving camera and compute the normals of the back projected depth image.
6. *Camera tracking*: Based on the rendered surface and normal vectors and the measured distance values, compute the new camera position and orientation with the Iterated Closest Point (ICP) algorithm [5].

The memory layout supporting the two-level hierarchy is shown by Figure 2. A macro-voxel is represented by a 32 bit long descriptor that contains the 21 bit long micro-voxel array address (poolIdx) or it is -1 if the macro-voxel is not subdivided. Other bits encode whether the current depth image intersects the macro-cell (isNearSurface) and it is seen from the camera (isSeen). Variable emptyCounter stores how many times the macro-voxel was found empty before it has been subdivided.

The free micro-voxel blocks are taken from a list (FreeList). A micro-voxel is represented by a 32 bit long descriptor where 24 bits represent the TSDF and 8 bits the weight.

3.1. Macro-cell marking

Step 1 determines whether higher level macro-cells are intersected by the surface. In classical volumetric fusion the center of the voxel is projected on the window plane of the depth image to locate the pixel where it is visible from the depth camera. If the difference of the depth value of this pixel and the distance of the voxel center from the camera is less than the truncation distance, the voxel is affected, otherwise it is assumed not to be intersected by the currently visible surface. Clearly, this method works only if the voxels are small and they are projected to a single pixel, otherwise sampling artifacts may show up. Higher level macro-cells are obviously not small enough, so a more precise test is needed. The method of Scalable KinectFusion [2] solves this problem by projecting the hexagons of each voxel onto the window plane and conservatively rasterizing the projected polygon to identify the pixels where depth comparison is needed. If rasterization is done in parallel, then an additional reduction is needed to make the final conclusion for the hexagons, i.e. for the whole cell. Although this is a output-driven, i.e. gathering type algorithm [12, 11], but is quite complex and its thread divergence is high since different cells may be projected to highly varying number of pixels. Another drawback of the output-driven approach is that we should explicitly ignore those voxels that cannot be projected onto the window by a culling method.

To address the problems of the marking method of Scalable KinectFusion, we propose an input-driven, i.e. scattering type approach, which automatically ignores invisible cells, easy to implement, and its thread divergence is small. Due to its scattering type, different threads can write the same memory locations, but this problem can be handled

without atomic writes in this special case. Moreover, if a one-frame delay is acceptable, the computational cost of the method is practically zero since it can be done together with the ray-casting step of the iteration, which is needed by the ICP anyway.

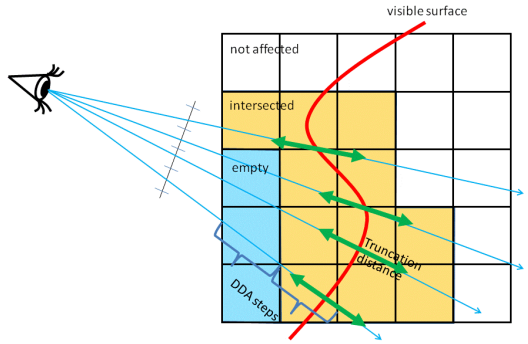

Figure 3: *The proposed macro-voxel marking algorithm. Initially all cells are set to unseen, i.e. not empty and not intersected. Threads are assigned to rays and execute multi-level DDA on the higher level macro-voxel grid starting at the cell closest to the camera. Before finding a macro-voxel overlapped by the truncation interval, macro-voxels are marked as empty. When the truncation interval overlaps with the cell, the macro-voxel is marked as intersected.*

The marking method updates two flags, one indicating emptiness, the other intersection. Initially both flags are cleared. The marking process assigns a GPU thread to every pixel of the depth image. The thread takes the depth value of the camera in this pixel and forms an interval, where the minimum is the depth value minus the truncation distance, and the maximum is the depth value plus the truncation distance. The thread executes a multi-level DDA based voxel traversal method [1] to identify those higher level cells that are intersected by this ray. Until the ray parameter at the exit point of the cell is lower than the minimum value, the visited cells are marked as empty (Figure 3). In empty cells there are no visible surfaces, so truncated signed values are updated accordingly. Cells where the entry-exit interval of the ray parameter overlaps with the truncation interval are possibly intersected by the noisy surface.

As different rays may intersect the same macro-cell, different threads may update the flags of the same cell, causing write collisions and usually necessitating atomic writes, which are slow. However, in this special case, atomic operations can be saved. The descriptor of a cell are shown by Figure 4, where the two flags are put into two bytes of the descriptor word, thus each of them can be accessed without modifying the other flag. If needed, a thread sets a flag independently of its previous value, and all other bits of the byte are constant during the execution of this thread. So the result is independent of the order how threads access these bytes.

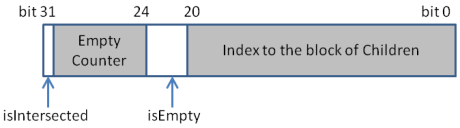

Figure 4: *A single macro-voxel is represented by a 32 bit word containing the index of the block of child-voxels, the flags of intersection and emptiness, and also a counter showing how many times the complete macro-cell was found empty. Note that the flags of intersection and emptiness are put in different bytes, so they can be modified independently.*

When threads are complete, both flags may be still cleared, which means that this macro-cell is not affected. If the intersected flag is set, the surface intersects this macro-cell regardless of the state of the empty flag. If only the empty flag is set, the macro-cell does not contain surface and its children are either empty or not seen.

The discussed marking process is used to identify macro-cells to be empty or intersected, while low level voxels are still processed by projecting their center onto the image plane. Thus, our algorithm identifies those macro-cells where low level processing is necessary. Note that in Step 2 called Rendering, a ray-casting needs to be executed anyway to track the camera by ICP, thus the identification of affected cells can be merged with this step reducing the additional cost of marking to almost zero. However, rendering happens with the old camera position and orientation while the fusion should use the updated camera parameters. As our algorithm marks only macro-cells, the one frame delay does not result in inaccuracies, but it can happen that voxels of a macro-cell are not updated in a frame or are tried to be updated when it is not necessary. Such a loss of a single frame update is tolerable and can happen due to the noise of the depth camera as well.

3.2. Fusion

For fusion, we use the classical method but only for the micro-voxels of subdivided macro-cells. The center of the micro-voxel is projected on the window plane of the depth image to locate the pixel where it is visible from the depth camera. If the difference of the depth value of this pixel and the distance of the voxel center from the camera is less than the truncation distance, the voxel is affected, otherwise it is assumed not to be intersected by the currently visible surface. For the affected micro-voxels, the weighted average of their stored TSDF value and the TSDF obtained from the current depth image is computed, and simultaneously the weight is incremented.

3.3. Rendering

The rendering method is a hierarchical DDA method [10, 13] that switches to micro-voxel steps only for subdivided

macro-cells. In each cell, the ray parameter of the exit point is generated by a single addition. To reduce noise, the volume can also be filtered before rendering [15, 14].

3.4. Camera tracking

The camera tracking method is incremental, if the camera parameters are known in the previous frame, just the rotation and translation between the current and previous frames should be computed, and this new transformation is concatenated to the camera transformation of the previous frame. If we could identify a set of corresponding point pairs on two images, then we could find that rotation and translation which would align the two point clouds making the distance of the new point cloud and the transformation of the previous point cloud close to zero. This is an optimization problem, which can be solved iteratively.

To find possibly corresponding pairs, we exploit the fact that just a little time has elapsed between two frames, so a point remains in the same pixel with high probability [5]. This is not always true, so outliers must be detected and rejected. So we take the distance field and generate a point cloud from the camera of the previous frame. Then we take the current depth map and back project it to obtain a point cloud in the new camera coordinate system. The camera transformation between the current and previous frames is expected to align the two point clouds. Alignment is detected when back projected points are on the zero level surface of the distance field and also that the normal vectors of the back projected mesh at these points are similar to the normal vectors of the zero level surface. If either the distance or the normal vector difference is very large, then probably two non-corresponding points are taken, so they are rejected as outliers. Otherwise, we consider this pair as corresponding and use them while minimizing the total distance of the two point clouds.

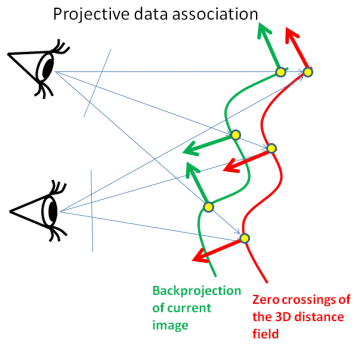

Figure 5: *Modified ICP algorithm applied in volumetric fusion*

4. Results

This system is implemented in CUDA. Figure 6 compares reconstructions of a telephone from Kinect 2 depth images using the Microsoft KinectFusion and our hierarchical method when allowing the same amount of GPU memory. Due to the hierarchical representation, voxel edge length could be reduced from 8 mm to 1 mm. Both algorithms run at real time on NVIDIA 690 GT GPUs.

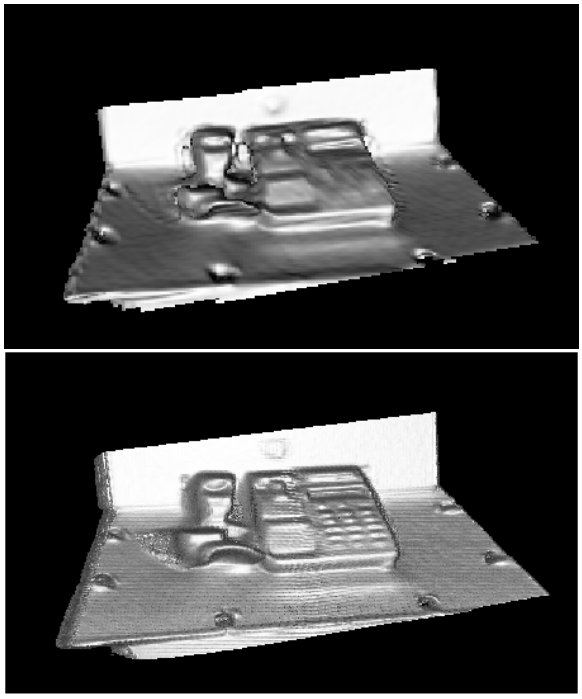

Figure 6: *Comparison of the commercial version of KinectFusion (upper) and the proposed algorithm (lower) when the two methods allocate the same amount of GPU memory.*

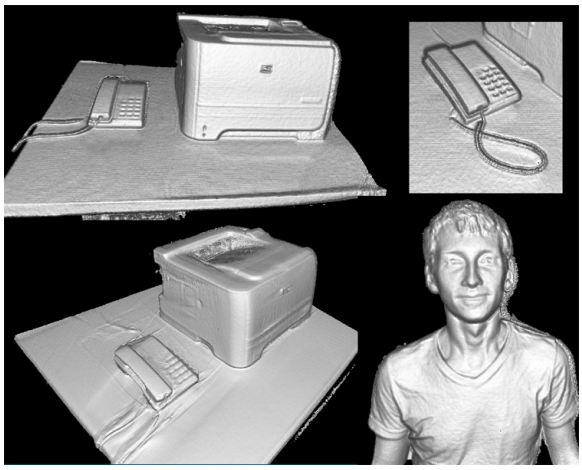

Figure 7: *Scanned objects.*

5. Conclusions

This paper presented a 3D volumetric fusion system that stores the TSDF field in a two-level hierarchical voxel grid. The input of the system is a real-time sequence of depth images of a moving camera, the output is the reconstructed 3D surface generated as the level surface os the computed distance field. The camera is tracked based on the temporary estimate of the scene. We discussed GPU-friendly methods how to select those cells that are affected by the current depth image. The algorithm provides high resolution reconstructions in real-time.

6. Acknowledgements

This work has been supported by OTKA K–104476 and VKSZ-14 SCOPIA projects.

References

1. J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In *Proceedings of Eurographics '87*, pages 3–10, 1987.
2. Jiawen Chen, Dennis Bautembach, and Shahram Izadi. Scalable real-time volumetric surface reconstruction. *ACM Transactions on Graphics (TOG)*, 32(4):113, 2013.
3. Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 303–312, New York, NY, USA, 1996. ACM.
4. Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments. In *In the 12th International Symposium on Experimental Robotics (ISER*. Citeseer, 2010.
5. Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 559–568, New York, NY, USA, 2011. ACM.
6. M. Levoy. Display of surfaces from ct data. *IEEE Computer Graphics and Application*, 8:29–37, 1988.
7. Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.
8. Jörg Stückler and Sven Behnke. Multi-resolution surfel maps for efficient dense 3d modeling and tracking. *Journal of Visual Communication and Image Representation*, 25(1):137–147, 2014.
9. L. Szirmay-Kalos. *Számítógépes grafika*. ComputerBooks, Budapest, 1999.
10. L. Szirmay-Kalos, V. Havran, B. Benedek, and L. Szécsi. On the efficiency of ray-shooting acceleration schemes. In *Proc. Spring Conference on Computer Graphics (SCCG)*, pages 97–106, 2002.
11. L. Szirmay-Kalos and L. Szécsi. General purpose computing on graphics processing units. In A. Iványi, editor, *Algorithms of Informatics*, pages 1451–1495. MondArt Kiadó, Budapest, 2010. http://sirkan.iit.bme.hu/szirmay/gpgpu.pdf.
12. L. Szirmay-Kalos, L. Szécsi, and M. Sbert. *GPU-Based Techniques for Global Illumination Effects*. Morgan and Claypool Publishers, San Rafael, USA, 2008.
13. László Szirmay-Kalos, György. Antal, and Ferenc Csonka. *Háromdimenziós grafika, animáció és játékfejlesztés*. ComputerBooks, Budapest, 2003.
14. László Szirmay-Kalos, Milán Magdics, and Balázs Tóth. Volume enhancement with externally controlled anisotropic diffusion. *The Visual Computer*, pages 1–12, 2016.
15. László Szirmay-Kalos. Filtering and gradient estimation for distance fields by quadratic regression. *Periodica Polytechnica Electrical Engineering and Computer Science*, 59(4):175–180, 2015.
16. Thomas Whelan, Michael Kaess, Maurice Fallon, Hordur Johannsson, John Leonard, and John McDonald. Kintinuous: Spatially extended kinectfusion. 2012.