Efficient Voxel Marking for Hierarchical Volumetric Fusion

László Szirmay-Kalos, Balázs Tóth, and Tamás Umenhoffer

Budapest University of Technology and Economics, Hungary

Abstract

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

Moving a depth camera around a 3D object, geometric information of noisy depth images can be fused together into a smooth 3D model. Additionally, the temporary result of the fused data can be used to track the camera, executing Simultaneous Localization And Mapping (SLAM). A famous approach to depth image fusion is the Curless-Levoy algorithm [CL96] that results in a *Truncated Signed Distance Field* (TSDF) of the reconstructed surface. To avoid the interference of surfaces on opposite sides, the signed distance is truncated. If a point is outside of the truncation interval and its TSDF is positive, it belongs to the empty region between the camera and the 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 macro-cells are decomposed to smaller cells and finally to voxels only if the surface intersects the cell [CBI13]. Volumetric fusion repeats the following steps:

1. *Fusion*: Find 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.
2. *Rendering*: Execute ray casting to render the surface from the current camera also computing the surface normals.
3. *Get depth image*: Read the new depth image from a possibly moving camera and compute the normals of the back projected depth image.
4. *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 [IKH*11].

Step 1 determines whether voxels or higher level macro-cells in case of hierarchical representation 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 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 [CBI13] 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 gathering type algorithm, but is quite complex and its thread divergence is high since different cells may be projected to highly varying number of pixels.

2. The new macro-voxel marking 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, which is needed by the ICP anyway.

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 DDA based voxel traversal method 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

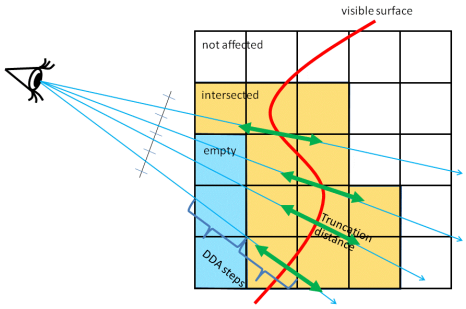

Figure 1: *Macro-voxel marking: Initially all cells are set to not-empty and not-intersected. Threads are assigned to rays and execute multi-level DDA on the higher level macro-voxel grid. Before finding a macro-voxel overlapped by the truncation interval, macro-voxels are marked as empty. When the truncation interval overlaps with the cell, the macro-voxel is marked as intersected.*

as empty (Figure 1). 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.

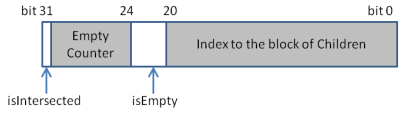

Figure 2: *A single macro-voxel is represented by a 32 bit word containing the index of the block of child-voxels, the flags of intersection and emptiness, and also a counter showing how many times the complete macro-cell was found empty. Note that the flags of intersection and emptiness are put in different bytes, so they can be modified independently.*

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 slower atomic writes. However, in this special case, atomic operations can be saved. The descriptor of a cell are shown by Figure 2, 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. 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. Results

This system is implemented in CUDA. Figure 3 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.

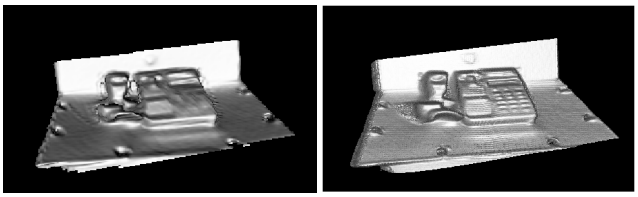

Figure 3: *Comparison of the commercial version of KinectFusion (left) and the proposed algorithm (right) when the two methods allocate the same amount of GPU memory.*

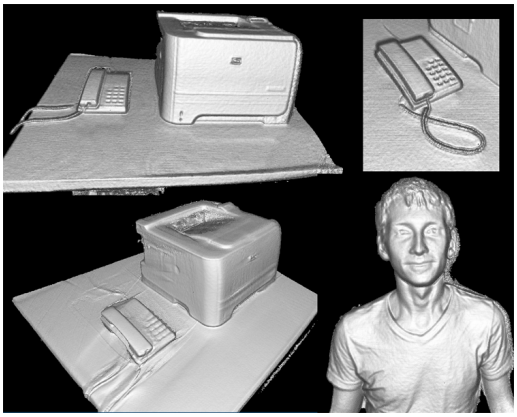

Figure 4: *Scanned objects.*

References

[CBI13] CHEN J., BAUTEMBACH D., IZADI S.: Scalable real-time volumetric surface reconstruction. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 113. 1

[CL96] CURLESS B., LEVOY M.: A volumetric method for building complex models from range images. SIGGRAPH '96, pp. 303–312. 1

[IKH*11] IZADI S. ET AL.: 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, pp. 559–568.