Parallel Adaptive Octree Carving for Real-time 3D Modeling

Luciano Soares[*] Clément Ménier[†] Bruno Raffin[‡] Jean-Louis Roch[§]

Lab ID-Imag, INRIA Rhône-Alpes

Abstract

We present a parallel octree carving algorithm applied to real time 3D modeling from multiple video streams. Our contribution is to propose a parallel adaptive algorithm for high performance width-first octree computation. It enables to stop the algorithm at anytime while ensuring a balanced octree exploration.

Keywords: Octree; 3D Modeling; Parallelism; Work Stealing.

Index Terms: I.4.5 [Image Processing and Computer Vision]: Reconstruction—Transform methods

1 Introduction

Computation intensive applications are common in Virtual Reality (VR). Computations take place for processing input data, simulating the virtual environment and rendering. The amount of computations can be very important in particular when using advanced input and output devices like multiple cameras. All these computations are performed under strong performance constraints, i.e. low latencies and sustained frequencies.

We focuse on marker-less 3D modeling. It consists in building a 3D model of the users or objects being filmed by a set of calibrated cameras. This 3D model must be computed in real time from the different video streams before to be injected into the virtual world to enable interactions. This technique has been used in different contexts, for interactive TV, distant real-time rendering of the persons interacting in a Cave [1] or full-body interactions.

To further increase performance, for using more camera images or decrease lantecy, we propose an adaptive parallel algorithm of the classical octree carving method [2]. This method builds an octree coresponding to the volumetric 3D model. This algorithm is particularly interesting as it enables to control the level of details of the computation by limiting the octree max depth. In a VR context where the tradeoff between level of details and computation time is critical, we turn this algorithm into an anytime algorithm that can be stopped when a time limit is reached. This is simply done by enforcing a width-first exploration of the octree. Though trivial to implement on a sequential machine, it is difficult to acheive with high performance on a parallel architecture due to the non uniform distribution of the workload in the octree voxels.

Our algorithm relies on a modified work stealing approach to ensure an high performance width-first octree computation. It ensures all processing units progress syncronously to the bottom of the octree, enabling to stop the algorithm at anytime while ensuring a balanced octree exploration (it avoids to vaste processing power in a deep exploration of an octree branch, while an other process stays several depth level behind). Experimental results shows that execution times can decrease from 441.1 ms on 1 CPU to 31.15 ms on 16 CPUs, significantly outperforming the equivalent non adaptive parallel algorithm.

[*]e-mail: luciano.soares@inrialpes.fr
[†]e-mail: clement.menier@inrialpes.fr
[‡]e-mail: bruno.raffin@imag.fr
[§]e-mail: jean-louis.roch@imag.fr

2 Octree Based Voxel Carving

The algorithm takes as input data video streams from a network of cameras. To ensure a high quality modeling, cameras must be properly calibrated and synchronized. To compute the silhouettes, the background is subtracted from each image, then pixels outside the silhouettes are set to white, while the others are set to black. The octree algorithm is executed on each set of silhouette images taken at the same time. Starting from one initial voxel corresponding to the acquisition space, the algorithm probes each voxel to compute if it lies outside the visual hull, or inside all silhouettes (full voxel). Uncertain voxels (intersecting a silhouette contour) are split in 8 smaller voxels.

3 Parallel Adaptive Octree Algorithm

The goal of this work stealing algorithm is to dynamically schedule the workload for an efficient use of the processors provided by the system.

We consider a task the computations required to test how a voxel projects into the silhouettes. Each voxel is represented as a quadruple of its coordinates and level (i, j, k, d). To manage the workload the voxels are organized in ready lists (RL). Each ready list consists of a vector of voxels and pointers to the first, last and current voxel.

The ready lists are organized in a singly-circularly-linked list. There is one list of ready lists per depth level, called a level list. Each processor is assigned a first ready list from the starting level. Each processor also creates an empty ready list for the next depth level.

Each processor computes the voxels of its ready list. If a voxel is evaluated as empty, it is simply forgotten. If the voxel is full, it is stored in the list of full voxels that owns each processor. These lists define the volume of interest that will be used after completing the algorithm. A "gray" voxel is split into 8 child voxels inserted into the ready list of the next level.

Each processor cycles twice through the level list. When it ends its ready list, a processor scans the level list from a randomly chosen position, looking for a free ready list in the level list until it completes one cycle. If it finds one, it takes it. Else, since there is no more ready list to grab, it performs a second loop, but this time the processor becomes a thief. It traverses the ready lists trying to get part of the remaining voxels. For a target ready list, the thief processor locks the current working pointer of the victim processor (the owner of the list). It checks the amount of remaining voxels to be computed. If this amount is bigger than the logarithm of the size of the ready list, the thief grabs half of the remaining voxels. This aims at avoiding to steal a too small number of voxels, with a stealing overhead that would not be compensated by the new workload balance. The thief creates a new ready list containing these voxels. The working pointer of the victim is unlocked as soon as it can safely restart processing its ready list.

Finally, when a processor ends its second cycle through the current level list, it starts working on the next level, processing the voxels of its ready list if not empty.

4 Results

The computer used for the tests has 8 dual Core AMD^{TM}2.2GHz Opteron processors.

Tests were first performed with two off line series of images. The first one is a sequence of a full body person filmed with 8 cameras, called the Ben benchmark, freely available at `small https://charibdis.inrialpes.fr/`. Each camera image has a resolution of 780x582 pixels. The second benchmark, called Al Capone, is a synthetic 3D model, from which we computed 64 images (resolution of 300x300 pixels). Having 64 cameras is beyond the number of cameras usually available in markerless motion capture environments.

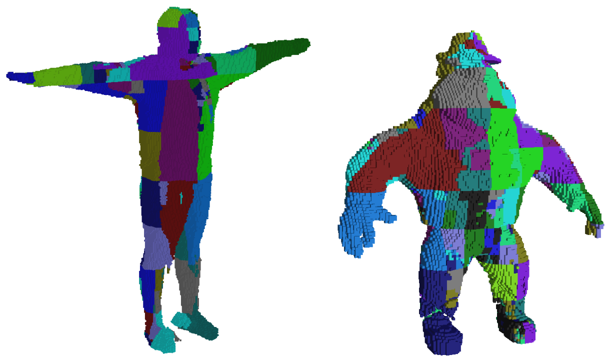

Figure 1: Bem. level 8 (left) and Al Capone level 7 (right).

We first compared a pure sequential implementation of the octree carving algorithm with our parallel code launched with only one thread. The overhead due to the extra code introduced into the algorithm for work stealing is small. It is about 4% for the Ben model and below 1.3% for Al Capone.

We ran the algorithm for both benchmarks with varying numbers of CPUs, time limit and max depth levels. All results are an average on 100 runs. The execution times include the time to load the images stored on a local disk. We plot Fig. 2) the Al Capone execution times (logarithmic scale on the y-axis) and the speed up for both the adaptive algorithm and the same algorithm but with work stealing disabled The gain of using 16 CPUs is very significant with an efficient use of the resources (high speed-ups). The comparison with the non adpative version shows that the overhead incurred by work stealing is compensated by the important performance gain. It also shows that a static task distribution is highly ineffective in the case of the octree carving as the workload is not uniformly distributed.

Ben at max level 8 is computed on 1 processor in about 234.2 ms and only 16.82 ms on 16 processors. The Al Capone at max level 7, goes from about 441.1 ms with 1 CPU to about 31.15 ms with 16 CPUs.

With Ben, we- tested time control by limiting the max depth to 8 and the computation time to 30 ms (Fig. 3). Below 8 processors, the time control is effective, stopping the execution before the max depth is reached. For more than 8 processors the max depth level is reached and the extra processors available are used to decrease the execution time.

We tested the algorithm in a live environment with 5 FireWire cameras (image resolution 780x580) shooting at a user (see the attached video). Cameras are genlocked and each one is connected to one computer, that removes the background and computes the silhouette. Then, the silhouettes are forwarded to the 16 core computer. It computes the octree and sends the list of full voxels to 16 dual opteron computers powering a 16 projector high resolution display wall. These computers render the voxels. This application

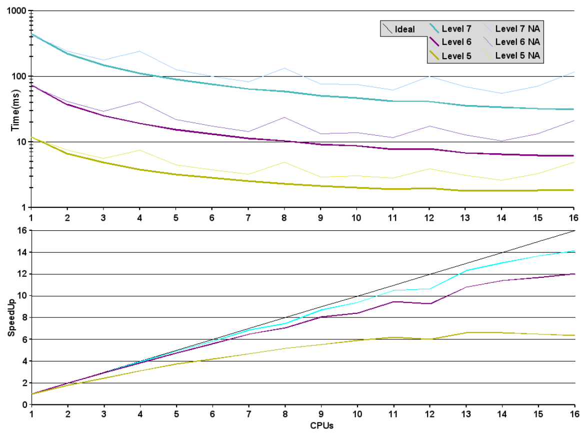

Figure 2: Execution time and speed up for Al Capone with (level5, level6 and Level7) and without (level5 NA, level6 NA, level7 NA) work stealing.

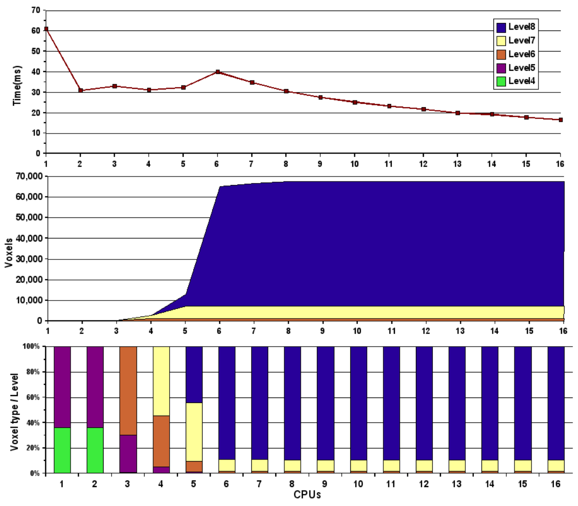

Figure 3: Ben modeling with a 30 ms time limit. The graph plots the total execution time, the middle graph plots the amount of voxels produced per level, and the lower graph the percentage of voxels types.

was developed on top of FlowVR for coupling and distributing the different software components.

5 Conclusion

We introduced a work stealing algorithm for parallel width-first exploration of octree. It enables to stop the execution at anytime to control the tradeoff between level of details and execution time. Applied to otree carving, results shows that this algorithm efficiently uses the computing resources available.

References

[1] M. Gross, S. Wuermlin, M. Naef, E. Lamboray, C. Spagno, A. Kunz, E. Koller-Meier, T. Svoboda, L. V. Gool, K. S. S. Lang, A. V. Moere, and O. Staadt. Blue-C: A Spatially Immersive Display and 3D Video Portal for Telepresence. In *Proceedings of ACM SIGGRAPH 03*, San Diego, 2003.

[2] R. Szeliski. Rapid Octree Construction from Image Sequences. *Computer Vision, Graphics and Image Processing*, 58(1):23–32, 1993.