The Design of Parallel Solid Voxelization Based on Multi-Processor Pipeline by Program Slicing

Duoduo Liao1, Simon Y. Berkovich2
Computer Science Department
George Washington University
801 22nd Street NW, Washington DC 20052 U.S.A.

Abstract: — This paper presents a new design for parallel solid voxelization using the multi-processor pipeline by slicing the program by forced interrupts. The proposed system provides a simple and effective solution to the on-the-fly parallel solid voxelization by relocating the portions of incoming data from processor to processor. It does not need conventional complicated parallel computing methods, such as load balancing, scheduling etc. This is exactly the advantage of this proposed system. Furthermore, the system only requires practically the same software as ordinary sequential solid voxelization algorithm. In this paper, the general parallel solid voxelization algorithm is summarized as apposed to the general sequential solid voxelization algorithm. The proposed system and the principle of parallel solid voxelization by multi-processor pipeline will be described and discussed in detail. In the end, the conclusions and future work are addressed.

Keywords: — Volume graphics, solid voxelization, parallel computing, multi-processor pipeline, program slicing, interrupts

1 Introduction

Over the past decades, voxelization has been widely used for many applications, such as medical imaging, Computer-Aided Design (CAD), volumetric modeling [12][24], haptic rendering [21], collision detection [5][13], 3D spatial analysis [1], Physics-based simulation [18], and real-time rendering [11].

Voxelization is a process that converts geometric objects from their continuous geometric representation into a set of voxels that best approximates the objects. Surface voxelization normally provides the volumetric representation only for the boundary surfaces without interior voxels, whereas solid voxelization produces the complete volume explicitly. Solid voxelization is accordingly much more difficult and time-consuming due to requiring an inside test for each voxel inside the volume space.

With increasing demands of real-time large volumetric data processing, many parallel and large-scale data volumetric rendering approaches have been studied intensively [8][23]. However, very few parallel voxelization algorithms are investigated so far.

This paper proposed a new approach for parallel solid voxelization using the multi-processor pipeline. This pipeline is designed to slice the program by forced interrupts onto the multiple processors for processing the continuous flow of intensive computations [3][4]. Through this pipeline architecture, the solid voxelization program for a conventional single-CPU computer with 3D accelerator can be automatically scheduled onto multiple processors with interrupt control. Consequently, the speed, quality, and size of the voxelization can be improved significantly.

In the paper, the related work and background will be first introduced in Section 2. Section 3 outlines the general parallel solid voxelization algorithm as apposed to the general sequential solid voxelization algorithm. The proposed system and the principle of parallel solid voxelization by multi-processor pipeline will be described and discussed in detail in Section 4. In Section 5, the conclusions and future work are addressed.

2 Related Work

2.1 Voxelization

Some difference solid voxelization methods have been proposed over the past years. In early year, traditional pure software-based 3D scan-conversion [16] and filtering-based distance volume to avoid aliasing [22] was studied for solid voxelization.

With the rapidly growing power of modern graphics hardware, hardware-accelerated techniques have been employed for solid voxelization. A slice-based solid voxelization is presented in [6] through setting one slice-thick projection as a clipping plane to generate voxel-based closed surface boundary and then applying a logical XOR operation between current slice and previous slice that has already been classified against the solid object. This process is slice-dependent. But the voxelization only produces binary voxels.
Later, a slice-independent solid voxelization algorithm based on stencil masking techniques was proposed in [20] and significantly improved in [19]. In these algorithms, a stencil buffer is utilized as a mask to generate interior voxels for each slice. The voxelization for each slice is independent, which is convenient for parallel processing.

Some other recent hardware-based solid voxelization algorithms [9][11] take advantages of GPU to use rasterization functionality of graphics hardware to generate the surface boundary voxels and then apply 3D scan-filling to complete interior voxelization. These algorithms are very slow due to voxel-by-voxel 3D scan-filling based on pure software computations.

2.2 Parallel Computing

In general, there are two major ways used for multiprocessor processing nowadays as follows:

1) Partition the data and schedule them onto the multiple processors
2) Partition the program into functions and schedule them onto the multiple processors

The first method is suitable for the data-independent circumstance. However, the scheduling may be complicated depending on the application programs. For the second method, in practice, it is often difficult to divide a program in such a way that separate CPUs can execute different portions without interfering with each other. Furthermore, this type of parallel processing requires very sophisticated software called distributed processing software [7][14][15].

Another promising parallel computing method is based on MP pipeline architecture by dividing the program in equal duration by forced interrupts as described in [3][4]. The most advantage of this architecture is to use practically the same software as a sequential computer and be able to continuously process intensive information flows.

3 The General Parallel Solid Voxelization Algorithms

In this paper, the core solid voxelization will use the algorithm described in [19][20], CreateSolidSliceByMasking(), for each slice voxelization based on graphics hardware acceleration. In this algorithm, stencil buffers and depth buffers will be employed as well. Furthermore, each slice can be processed independently. This can make the multi-processor pipeline simple.

The sequential algorithm for solid voxelization [19] is as follows:

Fig. 1 The sequential algorithm for solid voxelization

An entire volume is computed based on each slice. Each slice can be computed independently. A simple yet inexpensive way of parallelization is to partition the volume into slices and assign them to processors. Since there is no communication between any slices, there is no need for voxelization of adjacent slices to adjacent processors.

Figure 2 shows the flow chart of general parallel slice-independent voxelization of geometry objects.

Fig. 2 The flow chart of parallel solid voxelization

The general parallel algorithm of slice-independent solid voxelization is described below:

Algorithm 1: Sequential Solid Voxelization

SolidVoxelization_Sequential()

Input:

\( O \) ← a closed surface-based object
\( Nx, Ny, Nz \) ← size of the output volume

Output:

\( V \) ← the voxelized solid volume

Begin

1 Set viewport, projection, and background color
2 Generate the display list \( L \) for geometric object \( O \) drawing.
3 For each slice \( i = 0 \) to \( Nz - 1 \) do
   \( i \)th solidSlice ← CreateSolidSliceByMasking \((L, i)\)
   Insert \( i \)th solidSlice into the 3D texture volume \( V \)
End

4 Output the entire volume \( V \)
End
Algorithm 2: General Parallel Solid Voxelization

SolidVoxelization_Parallel()

Input:
\( O \leftarrow \) a closed surface-based object
\( Nx, Ny, Nz \leftarrow \) size of the output volume

Output:
\( V \leftarrow \) the voxelized solid volume

Begin
1. Set viewport, projection, and background color
2. Generate the display list \( L \) the geometric object \( O \) drawing.
3. For each slice \( i \), all or portion of the slice is assigned to the processor do
   \( \text{i-th solidSlice} \leftarrow \text{CreateSolidSliceByStencilMask}(L, i) \)
   if (The full slice voxelization is done)
      Insert \( \text{i-th solidSlice} \) into the 3D texture volume \( V \)
End
4. Output the entire volume \( V \)
End

Fig.3 The general parallel algorithm for solid voxelization

In the parallel version, all the slices in the volume space are not voxelized on the same processor. Each processor voxelizes only all or portions of the slices which are assigned to it.

For implementation on a multi-processor, the input geometry object can be loaded into the shared-memory. The output volume-buffer can be stored in shared-memory as well. The framebuffers including stencil buffers and z-buffers for each slice are local to the processor (i.e., Graphics Processor Unit (GPU)).

4 Parallel Solid Voxelization by Multi-processor Pipeline

4.1 The Multi-processor Pipeline by Slicing the Program by Forced Interrupts

The multi-processor pipeline presented in [2][4] is a simple and effective solution to the problem of real-time solid voxelization by relocating the portions of incoming data from processor to processor in a helicoidal pattern among multiple processors on the fly.

Fig.4 Helicoidal pattern in on-the-fly computations by a microprocessor pipeline (Loading, Processing, and Unloading rotate in the columns in the cycle: \( L \rightarrow P \rightarrow U \rightarrow L \))

4.2 Parallel Solid Voxelization by Multi-processor Pipeline

1) The Proposed System

This multi-processor pipeline architecture offers a feasible and effective solution to the problem of real-time solid voxelization by relocating the portions of incoming data from processor to processor in a helicoidal pattern among multiple processors on the fly.

Fig. 5 Multi-processor pipeline system for parallel solid voxelization

Such a system for the voxelization algorithm as shown in Figure 5 can consist of a plurality of CPU and GPU combination sets (CGPU), a shared-memory, and a shared-cache memory for data processing. The shared-memory is used for storing input geometry object data. Similarly, the voxelization may produce a large amount data result. The shared-memory can be used for storing the volumetric data only if the data will be immediately used for other applications, such as volume rendering after...
voxelization. The shared-cache is used for storing the display lists, groups of frequently used commands that have been stored for later repeated execution in multiple times. Using display lists can reduce the cost of repeatedly transmitting the data over the bus. The performance can be improved by using display lists. The output unit at the end of the pipeline can be hard disks or frame buffers. These frame buffers could be internal framebuffers in CGPUs or external framebuffers.

All of the required processing of the incoming data can be conveniently performed on-the-fly using the CGPU. The CGPU is able to implement on-the-fly the solid voxelization algorithm within a block of data a given size. From the functional point of view, this pipeline can be treated as a sequence which contains multiple CGPUs with their own banks of memory including frame buffers for GPU.

An important feature of this pipeline is that uses practically the same software as a sequential computer [4] A program for the multi-processor pipeline for voxelization can be developed on an ordinary sequential computer. To run on this system, the program would just have to incorporate some interrupts.

### 2) The Principle of Parallel Voxelization by Multi-processor Pipeline

The principle of parallel voxelization by the multi-processor pipeline is described in Figure 6 as follows.

In Figure 6, each step will go through the entire multi-processor pipeline. The next step cannot start until the previous step is finished. For both Step 1 and Step 2, the outputs need to be put into the memory in multi-processor so that it can be loaded into next stage as inputs. Step 1 and Step 2 are indispensable for voxelization.

In Step 1, the input data will be a stream of polygons which construct the geometry objects. As soon as the first CGPU is filled it can start processing the received block of data while the subsequent data stream is going into the second CGPU. When the second CGPU is loaded it can start processing its block of data while the data stream is going into the third CGPU. The output will be a display list for drawing the geometry objects.

After step 1, the geometry object data is still in shared-memory and the display list will be put into shared-cache. All object data and display list will be retrieved by all the same programs in all the CGPUs.

In Step 2, beside the geometry object data and display list, the incoming data includes the number of the slice according the Algorithm 2 in Figure 3. All the CGPUs will be filled with the same geometry object data and display list all the time. Only the number of the slice may be different for each CGPU. In this step, consequently, the input data except the number of the slice do not need to be relocated from CGPU to next CGPU. Upon an interrupt, a CGPU will unload its memories to the next CGPU. Similarly, the outcomes in the framebuffers including stencil buffers and depth buffers after the processing in this CGPU should be moved to the framebuffers in the next CGPU correspondingly. The CGPU will also pass the program status word so that next CGPU could resume operations where the previous CGPU left off.

The Figure 7 shows an example of a 3-CGPU pipeline as follows:

![Fig. 7 Parallel voxelization by 3-CGPU pipeline](image)

### 5 Conclusions and Future work

This paper presents an effective scheme for real-time parallel solid voxelization based on multi-processor pipeline by slicing the program by forced interrupts. The proposed system provides a simple and effective solution to the on-the-fly parallel solid voxelization by
relocating the portions of incoming data from processor to processor. It does not need conventional complicated parallel computing methods, such as load balancing, scheduling etc. This is exactly the advantage of this proposed system. 

More specifically, there are several advantages of this system as follows:

1) Provides the continuous data processing of intensive information flows
2) Only requires practically the same software as ordinary sequential solid voxelization algorithm
3) Avoid the data duplication by using shared-memory
4) Avoid the need for synchronization among the processors
5) Avoid busy waiting of processors on a spin-lock
6) Provide good load balancing without much computation for load balancing
7) Eliminate the data dependencies by using slice-independent voxelization algorithm

Furthermore, the time complexity of this parallel algorithm will only affect the length of the pipeline and the lag between the data relocation. In general, this new approach can significantly improve the total performance of the solid voxelization.

In the future, the detailed architecture and construction will be developed based on the principle and design as proposed in this paper. The system will be emulated and validated by some testing applications.

Another worth mention is that this multi-processor pipeline also provides an important solution to processing the continuous intensive information flows without limitation by size. Consequently, this would be very helpful to real-time large-data voxelization as well as real-time volumetric modeling and volume rendering for large data. Further development on these will be promising in practice.

References


