Towards Scalable and Efficient GPU-Enabled Slicing Acceleration in Continuous 3D Printing

Aosen Wang\textsuperscript{1}, Chi Zhou\textsuperscript{2}, Zhanpeng Jin\textsuperscript{3} and Wenyao Xu\textsuperscript{1}

\textsuperscript{1}CSE of SUNY Buffalo
\textsuperscript{2}ISE of SUNY Buffalo
\textsuperscript{3}ECE of SUNY Binghamton
Outline

01 Motivation

02

03

04

05
Continuous 3D Printing

Continuous 3D printing is a recent technical breakthrough in additive manufacturing [2015]. (Carbon3D)

Two Orders of Magnitude on Printing Acceleration

* This picture comes from internet: https://techcrunch.com/2015/08/20/with-100m-in-funding-carbon3d-will-make-3d-manufacturing-a-reality/
Principle of Continuous 3D Printing (Carbon 3D)

Speedup of Carbon 3D is mainly from **Manufacturing** (wet part)

- **Dry Part (Prefabrication)**: Computing unit slices of the layer images.
- **Wet Part (Manufacturing)**: Mechanical operations to fabricate 3D object from liquid materials.
Prefabrication V.S. Manufacturing

Prefabrication will become the bottleneck of continuous 3D printing.
Outline

01

02 **Background - Slicing**

03

04

05
The task in prefabrication includes three sequential procedures, i.e., slicing, path planning and support generation. Slicing dominates time efficiency in “dry part”.

In continuous 3D printing, image-mask-projection based slicing algorithm is employed. This pixel-independent processing enables massive parallel acceleration.
Methodology
(Slicing Algorithm Analysis)

The entire algorithm includes three cascaded modules, i.e., ray-triangle intersection, trunk sorting and layer extraction.

- The ray-triangle module is to calculate the intersection points between rays on image pixel center and the triangles from STL file.

- The trunk sorting sorts the out-of-order intersection points by ascending order using the bubble sorting in the trunk of each pixel.

- The binary value of each pixel on projected images is identified by incremental updating, so that the topology information is extracted for binary slicing image.

Trunk Sorting takes up a minor part of computation.
Outline

01

02

03 GPU Acceleration

04

05
GPU-Enabled Slicing-I
(Pixelwise Parallel Slicing)

- By the sequential algorithm analysis, we exploit the pixelwise parallelism based on GPGPU architecture.
GPU-Enabled Slicing-I (Pixelwise Parallel Slicing)

- By the sequential algorithm analysis, we exploit the pixelwise parallelism based on GPGPU architecture.
- The entire processing in all three functional modules for one pixel is assigned to a specific thread.
GPU-Enabled Slicing-I (Pixelwise Parallel Slicing)

- By the sequential algorithm analysis, we exploit the pixelwise parallelism based on GPGPU architecture.
- The entire processing in all three functional modules for one pixel is assigned to a specific thread.
- Fully use of precious shared memory on GPU to reduce time-consuming global memory intersections.
GPU-Enabled Slicing-II
(Fully Parallel Slicing)

- PPS still has serial computing components.
- FPS explores the massive thread concurrency
GPU-Enabled Slicing-II
(Fully Parallel Slicing)

- PPS still has serial computing components.
- FPS explores the massive thread concurrency
- This method increases global memory accessing pattern, but is scalable for large-size problem.
- The issue of multi-thread memory writing conflict arises and can be addressed by atomic operation based critical area.
Comparison of Two GPU Implementations

<table>
<thead>
<tr>
<th>Host (CPU)</th>
<th>GPU</th>
<th>STL File</th>
<th>Triangle Mesh Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ray-Triangle Intersection</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Trunk Sorting</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Layer Extraction</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host (CPU)</td>
<td>GPU</td>
<td>Output Slicing Images</td>
<td>Fully Parallel Slicing</td>
</tr>
<tr>
<td><strong>Pixelwise Parallel Slicing</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Comparison of Two GPU Implementations

Host (CPU) -> STL File -> Triangle Mesh Statistics

GPU

Ray-Triangle Intersection

- Load triangle to shared memory
- Calculate the ray index
- Intersect? -> Yes -> Store result to trunk in shared memory
- Intersect? -> No
- Unvisited Triangle? -> Yes

Trunk Sorting

Layer Extraction

Host (CPU) -> Output Slicing Images

Pixelwise Parallel Slicing

Fully Parallel Slicing
Comparison of Two GPU Implementations

**Host (CPU)**

**GPU**

**Ray-Triangle Intersection**

- Load triangle to shared memory
- Calculate the ray index
- Intersect? No
- Store result to trunk in shared memory
- Unvisited Triangle? Yes
- Yes
- No

**Trunk Sorting**

- Sort the trunk using bubble sorting

**Layer Extraction**

**Host (CPU)**

**Output Slicing Images**

**Pixelwise Parallel Slicing**

**STL File**

**Register and Shared Memory**

**Global Memory Interaction**

**STL File**

**Identify ray index and triangle information**

**Atomic Operation based Critical Area**

**Load triangle to shared memory**

**Intersect? Yes**

**Store result to trunk in global memory**

**Load intersections in trunk to shared memory**

**Sort the trunk using bubble sorting**

**Save the sorted vector back to trunk in global memory**

**Fully Parallel Slicing**

**Output Slicing Images**
Comparison of Two GPU Implementations

Host (CPU)

Pixelwise Parallel Slicing

Host (CPU)

Fully Parallel Slicing

GPU

Ray-Triangle Intersection

STL File

Load triangle to shared memory

Calculate the ray index

Intersect?

Yes

No

Store result to trunk in shared memory

Unvisited Triangle?

Yes

No

STL File

Register and Shared Memory

Global Memory Interaction

Identify ray index and triangle information

Load triangle to shared memory

Intersect?

Yes

No

Atomic Operation based Critical Area

Store result to trunk in global memory

Trunk Sorting

Sort the trunk using bubble sorting

Load intersections in trunk to shared memory

Sort the trunk using bubble sorting

Save the sorted vector back to trunk in global memory

Layer Extraction

Calculate cutting plane for layer

Calculate binary value on the pixel

Store binary pixel value in global memory

Unvisited Layer?

Yes

No

Identify layer number and cutting plane height

Load intersections in trunk to shared memory

Calculate the binary value

Store binary pixel value in global memory

Output Slicing Images
Comparison of Two GPU Implementations

**PPS:** all tasks in fast shared memory, less global memory access, no multi-thread conflict.

**FPS:** recycle-free processing, atomic operation based critical area to address conflict issue.
Experimental Setup

- We use cycle-accurate simulators for CPU and GPU computing platforms.
- Sniper is a typical simulator for x86 architecture and GPGU-Sim is a good simulating tool to check statistics of GPGPU architecture.
- Sniper is configured as Intel Xeon X5550 with 2.66GHz frequency while GPGPU-Sim is configured as Nvidia Geforce GTX480 with 700MHz.
- We choose four representative 3D objects: Club, Android, Ring and Bunny. They have different triangle mesh size, as 3290, 10926, 33730 and 69664.
Experiment: Time Efficiency

- Fully parallel slicing achieves the best performance in three schemes.
- Considering the processing frequency difference, PPS gains one order of magnitude improvement and FPS even obtains two orders acceleration.
Experiment: Scalability

- We choose three layer numbers: 10, 100 and 1000.
- FPS scheme on GPU can achieve about two orders time efficiency compared with CPU case.
- As layer number increases, layer extraction dominates the entire runtime.
- Trunk Sorting takes a subtle proportion.

- We choose three image resolutions: 128*64, 256*128 and 512*256.
- PPS holds one order of magnitude speedup and FPS achieves about two orders time efficiency compared to CPU.

Scalable of both PPS and FPS

<table>
<thead>
<tr>
<th>Time (Million Cycles)</th>
<th>Layer Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td>CPU</td>
<td></td>
</tr>
<tr>
<td>Ray-Triangle</td>
<td>517.38</td>
</tr>
<tr>
<td>Trunk Sorting</td>
<td>12.06</td>
</tr>
<tr>
<td>Layer Extract</td>
<td>33.11</td>
</tr>
<tr>
<td>Total</td>
<td>562.55</td>
</tr>
<tr>
<td>GPU</td>
<td></td>
</tr>
<tr>
<td>Ray-Triangle</td>
<td>0.300</td>
</tr>
<tr>
<td>Trunk Sorting</td>
<td>0.020</td>
</tr>
<tr>
<td>Total</td>
<td>7.965</td>
</tr>
</tbody>
</table>

Scalable of both PPS and FPS

- We choose three layer numbers: 10, 100 and 1000.
- FPS scheme on GPU can achieve about **two orders time efficiency** compared with CPU case.
- As layer number increases, layer extraction dominates the entire runtime.
- Trunk Sorting takes a subtle proportion.
Outline

01
02
03
04
05 Conclusions
Conclusions

- We investigated slicing algorithm acceleration on GPGPU architecture for continuous 3D printing.
- We developed pixelwise parallel slicing and fully parallel slicing implementations.
- Experiments demonstrate the effectiveness and scalability of our implementation.

In the future:
- We will design new implementations on the new hardware platform, such as FPGA or more powerful GPU.
- We will exploit pipeline property between prefabrication and manufacturing.
Q & A

Thank you!

Address comments to wenyaoxu@buffalo.edu