SIMD Optimization In Volume Rendering And Gaussian Filtering

(Rigorous Thesis)

Anton Vaško
Content

- Volume visualization
- Brief Overview of Volume Rendering Algorithms
- About SIMD
- SIMD Optimization in Ray Casting
- SIMD Optimization in Gaussian Filtering
- Conclusion
Quick Intro

- volume = 3D grid (discrete data)
- common in medical imaging (CT, MR, PET, SPECT ...)
- volumes can be large
  \[ 512 \times 512 \times 512 \times 32 \text{bits} = 512 \text{ MB} \]
Volume visualization

volume visualization:

improving speed of rendering:
1. adjusting volume data
2. optimizing rendering algorithms
Adjusting volume data

- removing spurious particles
- reducing the number of non-air octree cubes
- achieved by smoothing – e.g. Gaussian Filtering
HW Volume Rendering

- volume = 3D texture
- volume rendering = mapping 3D texture on cube or GPGPU raycasting (shaders)
- advantages: pretty fast (all is done in HW)
- disadvantages: need for new (main-stream or better) graphics cards with big RAMs
SW Volume Rendering

- more rendering techniques, e.g.
  1. Brute Force (ray casting)
  2. Shear-Warp Factorization
  3. Multiresolution Min-Max Octrees
  4. and ... more ...
- disadvantages: not so fast, of course
- advantages: runs practically everywhere
- can be optimized for SIMD
SIMD (1)

- Single Instruction Multiple Data

- Extended CPU instruction set
SIMD (2)

- SIMD support on current CPUs:
  1. Intel - MMX, SSE, SSE2, SSE3
  2. AMD – MMX, 3DNow!, SSE
  3. AMD64 – MMX, 3DNow!, SSE, SSE2, SSE3

- Decision:
  optimizing for 3DNow! and SSE
Approaches

- common aspect in volume rendering – processing a lot of data
- 2 basic approaches:
  - precalculate everything possible
  - vs
  - calculate everything on the fly
Present Time

- CPU is much faster than RAM
- calculating on the fly
- simple algorithms (ray casting)
- need for rewriting old algorithms

Decision:

SIMD Optimization of Ray Casting
SIMD in Ray Casting

Important parts:

- Bricking Data
- Entry Point Buffer
- Ray Casting
Bricking Data

- Non-linear storing in RAM
- Reasons:
  1. CPU cannot read from RAM but from cache. Cache miss = stall for ≈40 clocks
  2. Locality concept
  3. Increased algorithm’s data throughput with SIMD
Table 5.2: Impact of Bricking on Calculation Times [ms].

<table>
<thead>
<tr>
<th>$B_x$</th>
<th>$B_y$</th>
<th>$B_z$</th>
<th>EPB</th>
<th>RC-Init</th>
<th>RC-Accum</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>256</td>
<td>128</td>
<td>9.94</td>
<td>15.27</td>
<td>1426.91</td>
<td>1452.12</td>
</tr>
<tr>
<td>128</td>
<td>128</td>
<td>128</td>
<td>7.33</td>
<td>11.08</td>
<td>743.73</td>
<td>762.14</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>64</td>
<td>7.28</td>
<td>10.13</td>
<td>148.25</td>
<td>165.66</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>32</td>
<td>8.93</td>
<td>9.67</td>
<td>73.95</td>
<td>92.55</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>16</td>
<td>18.72</td>
<td>9.62</td>
<td>54.07</td>
<td>82.41</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>16</td>
<td>30.34</td>
<td>9.88</td>
<td>51.89</td>
<td>92.11</td>
</tr>
</tbody>
</table>
Entry Point Buffer

- Similar to Depth buffer
- Parallel projection → rasterizing just one block and copying
- Doing `memset` and `memcpy` is ‘delicacy’ for SIMD
## EPB Results

### Table 5.4: Optimization of stage EPB.

<table>
<thead>
<tr>
<th>PC</th>
<th>Duration [ms]</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>None</td>
<td>3DNow</td>
</tr>
<tr>
<td>PC1</td>
<td>19.13</td>
<td>14.4</td>
</tr>
<tr>
<td>PC2</td>
<td>48.9</td>
<td>40.2</td>
</tr>
<tr>
<td>PC3</td>
<td>26.39</td>
<td>23.0</td>
</tr>
</tbody>
</table>
Ray Casting

2 parts:

1. Initializing rays according Entry Point Buffer → SIMD optimizable
2. Processing rays (accumulation):
   not ‘SIMD optimization friendly’
Ray Casting Results

Table 5.6: Optimization of the Ray Initialization stage.

<table>
<thead>
<tr>
<th>PC</th>
<th>Duration [ms]</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>None</td>
<td>3DNow!</td>
</tr>
<tr>
<td>PC1</td>
<td>9.5</td>
<td>6.0</td>
</tr>
<tr>
<td>PC2</td>
<td>22.0</td>
<td>-</td>
</tr>
<tr>
<td>PC3</td>
<td>10.03</td>
<td>7.06</td>
</tr>
</tbody>
</table>
## Ray Casting Results

### Table 5.7: SIMD optimization of the Accumulation stage.

<table>
<thead>
<tr>
<th>PC</th>
<th>Duration [ms]</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>None</td>
<td>3DNow!</td>
</tr>
<tr>
<td>PC1</td>
<td>55.7</td>
<td>54.7</td>
</tr>
<tr>
<td>PC2</td>
<td>122.1</td>
<td>-</td>
</tr>
<tr>
<td>PC3</td>
<td>53.53</td>
<td>52.23</td>
</tr>
</tbody>
</table>
SIMD and Gaussian Filtering

- separable – 3 passes
- symmetric – saving multiplications
- floating-point
- filtering in-place
- without temporary buffer – how?
Extended volume 1

- new technique - extended volume

Figure 4.1: Position of the original volume in the extended volume during filtering. Left - at the beginning before filtering. Middle - after FilterX. Right - after FilterY. After FilterZ the filtered volume is positioned again in the top left of the extended volume.
Extended volume 2

- reducing passes

Figure 4.2: Position of the original volume in the extended volume during filtering. Left - at the beginning before filtering. Right - after FilterXY. After FilterZ the filtered volume is positioned again in the top left of the extended volume.

- parallelization of computation for SIMD
Limitations

- number of SIMD registers – 8 (x86)
- extended volume – additional extended slices
- Decision – 3 additional slices, 3 additional lines for each slice =>
- Gauss3, Gauss5, Gauss7
Results

Table 6.9: Results of gauss3 on PC3.

<table>
<thead>
<tr>
<th>Total</th>
<th>Duration [s]</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>None</td>
<td>3DNow!</td>
</tr>
<tr>
<td>volume1</td>
<td>0.1035</td>
<td>0.0051</td>
</tr>
<tr>
<td>volume2</td>
<td>0.8517</td>
<td>0.0442</td>
</tr>
<tr>
<td>volume3</td>
<td>6.9225</td>
<td>0.4362</td>
</tr>
<tr>
<td>volume4</td>
<td>379.7929</td>
<td>47.1538</td>
</tr>
</tbody>
</table>
Conclusion

- Real time volume rendering – GPU
- Quality volume rendering – CPU (12% speedup with SIMD)
- Gaussian Filtering – positively SIMD
- SIMD Optimization principles can be often successfully used even when not immediately followed by assembler implementation! (bricking, ...)
Future work

- Optimization of filtering:
  1. Gauss9, Gauss11, Gauss13, Gauss15 (16 SIMD registers in x86-64)
  2. General separable filtering (not only Gaussian, no kernel size restriction)
  3. Nonseparable filtering
Thanks for your attention!

vasko.anton@gmail.com