PGP - Model Voxelization

Juraj Joščák <xjosca00@stud.fit.vutbr.cz>

December 21, 2017

1 Introduction

The aim of this project was to implement data-parallel algorithms for model voxelization on graphics hardware. Binary voxel representations are heavily employed in computer graphics. Offering a regular representation that is independent from object and surface complexity, voxelizations are used in domains as diverse as 3D shape matching, visibility processing or collision detection. Since voxelization is often a performance-critical, integral part, several algorithms have been proposed to perform voxelization of triangle meshes on the GPU. This project uses principles described in [SS10].

2 Theory

Conservative voxelization requires identifying all voxels a triangle overlaps and hence it is crucial to utilize a fast test for triangle/voxel overlap. Given a triangle T with vertices **v0**, **v1**, **v2** and an axis-aligned box B (e.g. a voxel) with minimum corner p and maximum corner $p + \Delta p$, we observe that T overlaps B if their axis aligned bounding boxes overlap and also:

$$(\langle n, p\rangle + d_1) * (\langle n, p\rangle + d_2) \leq 0 \tag{1}$$

and:

$$\bigwedge_{i=0}^{2} (\langle n_{e_i}^{xy}, p_{xy}\rangle + d_{e_i}^{xy} \geq 0) \tag{2}$$

for each of planes XY, YZ and ZX, while:

n is normal vector of T

$d1 = \langle n, c - v_0 \rangle$

$d2 = \langle n, (\Delta p - c) - v_0 \rangle$

$$c = \left(\begin{Bmatrix} \Delta p_x, & \mathrm{n}_x > 0 \\ 0, & \mathrm{n}_x \leq 0 \end{Bmatrix} \begin{Bmatrix} \Delta p_y, & \mathrm{n}_y > 0 \\ 0, & \mathrm{n}_y \leq 0 \end{Bmatrix} \begin{Bmatrix} \Delta p_z, & \mathrm{n}_z > 0 \\ 0, & \mathrm{n}_z \leq 0 \end{Bmatrix} \right) \quad (3)$$

$$n_{e_i}^{xy} = (-e_{i,y}, e_{i,x}) * \begin{Bmatrix} 1, & \mathrm{n}_z \geq 0 \\ \text{-1}, & \mathrm{n}_z < 0 \end{Bmatrix}$$

$$d_{e_i}^{xy} = -\langle n_{e_i}^{xy}, v_{i,xy} \rangle + max\{0, \Delta p_x * n_{e_i,x}^{xy}\} + max\{0, \Delta p_y * n_{e_i,y}^{xy}\}$$

All values in 3 only need to be computed once per triangle. Then it is only neccessary to check for 1 and 2 for each box (voxel).

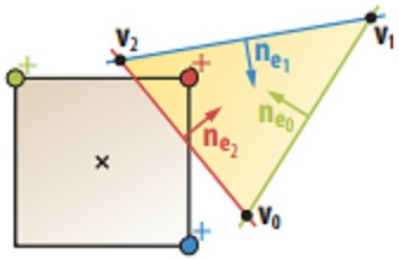

Figure 1: Critical points for evaluating the edge functions , annotated with the function result's sign. (Taken from [SS10])

3 Solution

Application makes use of OpenGL compute shaders.

Each instance of the shader program takes one triangle as input. Then program loops over all voxels overlapped by this triangle's bounding box. Voxel is set, if criteria 1 and 2 are met. Atomic or function is used to set voxels, in case multiple threads try to set the same voxel simultaneously

Two shader storage buffers store input and output respectively. Each triangle in input buffer is represented by nine floating point numbers (three coordinates for each vertex). Output buffer represents a three-dimensional voxel grid. Every voxel is represented by single integer i where $i = 1$ means set and $i = 0$ means unset.

Input buffer can be then used without modification to render the original model, but output buffer needs to be converted. After voxelization, application loops over the output buffer and writes coordinates of each set voxel into yet another buffer. Here, every set voxel is represented by three floating point numbers (coordinates in 3D space), unset voxels are ommited. This buffer is then used to render voxels as points and geometry shader is used to turn this points into cubes of right size and orientation.

Some code from [JJ16] was reused.

4 Evaluation

Resulting voxelized model looks as expected:

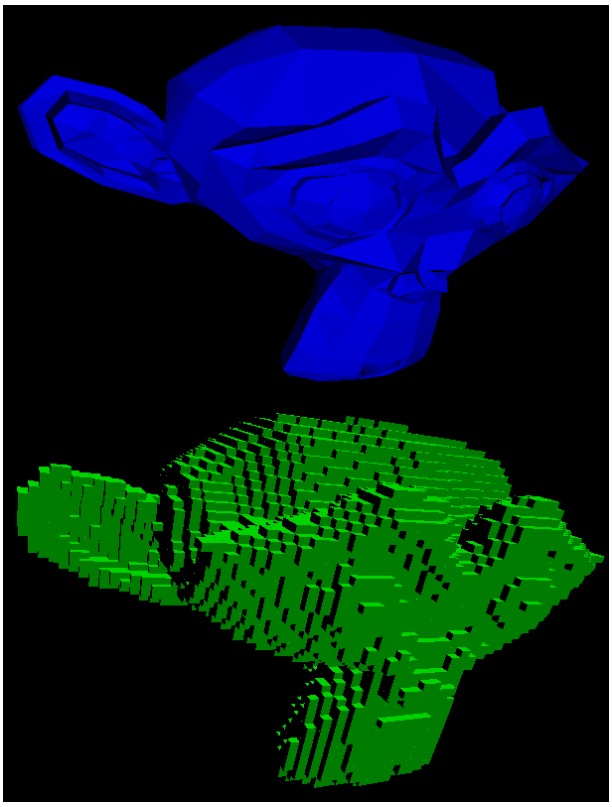

Figure 2: Top: original model **Bottom:** voxelized

GPU used to take performance measurements in table 1 was AMD HD6850:

Table 1: Performance results

Model (#triangles)	**resolution (x-axis)**	**time spent voxelizing (ms)**
Rotated cube (12)	30	15
-||-	70	46
-||-	150	176
Monkey (967)	30	20
-||-	70	39
-||-	150	105
Stanford bunny (4968)	30	9
-||-	70	185
-||-	150	141

5 Conclusion

Application works as expected. It is interesting, how triangle count and voxel resolution affect performance. Triangles are processed in parallel, but voxels in near each triangle are processed sequentially. This means a simple mesh can take longer to voxelize than more complicated one using the same resolution. Regardless, performance is satisfactory and can be without doubt further increased by optimization.

6 Appendix

Application requires:
- CMake (3.7 or higher)
- SDL2
- GLEW

Run as:

```
Voxelization.exe <resolution in x-axis> <path to *.obj file>
```

References

[JJ16] Michal Tóth Juraj Joščák. 3d logická hra s principy deformace prostoru. Master's thesis, BUT, 2016.

[SS10] Michael Schwarz and Hans-Peter Seidel. Fast parallel surface and solid voxelization on gpus, 2010. [Online; accessed 15-December-2017].