

# Fast Parallel Surface and Solid Voxelization on GPUs: Supplementary Material

Michael Schwarz

Hans-Peter Seidel

Max-Planck-Institut Informatik

## Performance on GeForce GTX 480

Tables 1, 2 and 3 list performance data for our methods obtained on an NVIDIA GeForce GTX 480. They correspond directly to the identically-numbered tables of the paper (for which an NVIDIA GeForce GTX 285 had been employed).

Often showing significant improvements in performance, the new GTX 480 features the following relevant architectural changes with respect to the GTX 285:





- The number of CUDA cores has doubled from 240 (organized into 30 multiprocessors) to 480 (15 multiprocessors), effecting a roughly twofold increase in raw compute power.
- Owing to the introduction of tessellation support, the GTX 480 features four rasterizers, whereas the GTX 285 has just one. This quadruplication partly boosts the performance of the pipeline-based approaches significantly.
- Support for atomic writes has substantially been improved with the GTX 480, which greatly benefits our methods. This is particularly obvious in case of conservative voxelization of the Asian dragon (where conflicting atomic writes render voxelization slower for lower-resolution grids on the GTX 285).

Please note that the employed external libraries `chag:pp` and

CUDPP have not been tuned for this new architecture yet; therefore they may perform suboptimally. For instance, CUDPP uses a hard-coded block size which results in an unnecessarily low occupancy on the GTX 480.

Model	Grid size	Pipe-line	Tri-parall.	Tile-based					
				$T_{total}$	$T_{pairs}$	$T_{sort}$	$T_{tiles}$	#pairs	#tris
Stanford bunny	128 <sup>3</sup>	0.23	0.27	1.03	0.27	0.15	0.31	42k	64.2
	256 <sup>3</sup>	0.31	0.69	1.30	0.28	0.23	0.47	81k	32.2
	512 <sup>3</sup>	1.26	3.55	2.08	0.30	0.44	0.85	145k	14.6
	1024 <sup>3</sup>	7.45	27.17	5.28	0.41	0.77	2.89	301k	7.6
Stanford dragon	128 <sup>3</sup>	1.51	0.84	1.62	0.69	0.15	0.32	40k	81.8
	256 <sup>3</sup>	1.56	1.03	2.35	0.80	0.34	0.71	156k	84.7
	512 <sup>3</sup>	1.75	3.44	4.61	0.95	1.10	1.98	476k	67.3
	1024 <sup>3</sup>	8.22	17.65	8.96	1.22	2.29	4.26	944k	34.0
Asian dragon	128 <sup>3</sup>	11.47	4.76	4.37	3.05	0.10	0.26	21k	81.5
	256 <sup>3</sup>	10.88	4.85	5.17	3.44	0.22	0.49	84k	89.1
	512 <sup>3</sup>	10.42	6.33	7.84	4.42	0.82	1.43	342k	96.3
	1024 <sup>3</sup>	11.42	12.33	15.16	5.07	3.02	5.18	1366k	99.6

**Table 2:** Running time (in ms) for different solid voxelization methods. The break-down for the tile-based approach considers determining the work queue ( $T_{pairs}$ ), sorting it ( $T_{sort}$ ), and processing the tiles ( $T_{tiles}$ ). Moreover, the number of tile/triangle pairs in the queue and the average triangle list length per active tile are provided.

Model	Grid size	Pipeline-based conserv. voxel.	Our conservative voxelization					6-separating voxelization			
			Simple	Specialized	Sorted	#voxels	1D/2D/3D	Simple	Specialized	Sorted	#voxels
Stanford bunny (69,666 tris) 	128 <sup>3</sup>	3.28	0.69	0.79	1.63	54k	10%/49%/41%	0.68	0.60	1.18	33k
	256 <sup>3</sup>	3.78	1.61	1.22	2.12	222k	0%/20%/79%	1.50	0.89	1.60	133k
	512 <sup>3</sup>	6.24	9.89	2.97	4.40	900k	0%/ 9%/91%	9.00	2.35	3.46	535k
	1024 <sup>3</sup>	21.44	58.27	10.91	13.79	3627k	0%/ 3%/97%	54.57	8.99	10.96	2147k
Shrub (751,399 tris) 	128 <sup>3</sup>	38.82	3.33	3.71	5.97	102k	75%/18%/ 7%	3.15	2.79	5.08	74k
	256 <sup>3</sup>	45.47	6.24	5.70	7.28	426k	57%/18%/25%	5.91	4.06	5.80	274k
	512 <sup>3</sup>	60.43	15.06	10.41	11.18	1705k	46%/17%/38%	13.85	7.24	8.10	1022k
	1024 <sup>3</sup>	118.01	79.57	27.60	25.39	6776k	22%/27%/52%	72.03	19.31	17.25	3924k
Stanford dragon (871,414 tris) 	128 <sup>3</sup>	46.35	3.45	3.46	6.65	39k	89%/10%/ 1%	3.44	2.50	5.74	25k
	256 <sup>3</sup>	51.01	3.91	3.90	7.15	162k	64%/28%/ 8%	3.96	3.19	6.24	98k
	512 <sup>3</sup>	59.41	4.53	6.18	9.27	660k	24%/39%/36%	4.44	4.46	7.06	387k
	1024 <sup>3</sup>	78.68	13.90	14.49	16.20	2664k	8%/24%/68%	12.06	10.17	11.50	1539k
XYZ RGB Asian dragon (7,218,906 tris) 	128 <sup>3</sup>	288.80	15.54	13.81	35.47	22k	99%/ 1%/ 0%	15.26	11.36	32.77	16k
	256 <sup>3</sup>	333.22	18.26	17.65	44.49	91k	97%/ 3%/ 0%	17.96	14.73	36.69	61k
	512 <sup>3</sup>	405.45	24.62	22.54	43.60	374k	88%/11%/ 1%	24.25	16.67	38.89	233k
	1024 <sup>3</sup>	539.86	24.39	22.64	48.51	1516k	60%/32%/ 8%	23.99	18.20	40.07	897k

**Table 1:** Running time (in ms) for different surface voxelization methods, along with the number of resulting voxels and the encountered percentage of cases with a 1D, 2D and 3D bounding box. For comparison, the pipeline-based approach by Zhang et al. [2007] is included.

Model	Stanford bunny				Stanford dragon				XYZ RGB Asian dragon			
Binary grid size	512 <sup>3</sup>	1024 <sup>3</sup>	2048 <sup>3</sup>	4096 <sup>3</sup>	512 <sup>3</sup>	1024 <sup>3</sup>	2048 <sup>3</sup>	4096 <sup>3</sup>	512 <sup>3</sup>	1024 <sup>3</sup>	2048 <sup>3</sup>	4096 <sup>3</sup>
Scalar-valued grid size	128 <sup>3</sup>	256 <sup>3</sup>	512 <sup>3</sup>	1024 <sup>3</sup>	128 <sup>3</sup>	256 <sup>3</sup>	512 <sup>3</sup>	1024 <sup>3</sup>	128 <sup>3</sup>	256 <sup>3</sup>	512 <sup>3</sup>	1024 <sup>3</sup>
<b>Total</b>	3.14	5.88	15.30	51.59	6.77	12.29	24.30	66.28	20.03	23.59	54.31	88.64
<b>Determine active level-1 nodes</b>	0.85	1.12	1.96	6.87	2.55	3.77	4.96	9.59	10.69	10.20	29.05	27.55
<b>Construct octree</b>	0.78	1.31	2.93	8.86	0.78	1.16	2.38	6.80	0.71	1.01	1.79	4.39
<b>Voxelize into octree</b>	0.97	2.53	7.88	27.34	2.93	6.37	14.93	43.56	7.55	11.05	21.44	52.18
<b>Propagate inside/outside</b>	0.19	0.49	1.57	5.96	0.13	0.35	1.13	4.23	0.11	0.23	0.69	2.46
<b>Compute coverage factor</b>	0.05	0.14	0.54	2.11	0.03	0.12	0.41	1.55	0.02	0.08	0.24	0.90
<b>Stored level-0 nodes</b>	5.30%	2.71%	1.37%	0.69%	3.82%	2.00%	1.01%	0.51%	2.02%	1.10%	0.57%	0.29%
<b>Octree size</b>	3.2 MB	13.2 MB	53.7 MB	216 MB	2.3 MB	9.7 MB	39.6 MB	159 MB	1.2 MB	5.3 MB	22.3 MB	90.8 MB

**Table 3:** Running time (in ms) for our sparse solid voxelization method, including a break-down. The percentage of explicitly stored level-0 nodes provides a measure of sparseness. The given octree size accounts for both the structure and the data.