Testing **Integer indexing** vs. **float indexing** of nodes

1a) The total time measured is GPU time measured by CUDA event. The statistics presented here reflect only relative figures as timings vary over each frame. Corresponding to some tables are also attached snapshots to give an idea of the position of model on the screen.

We measure first for the standard, original version of gigavoxels followed by making only integer indexing change. Then we test by only including improved stream compaction and finally by incorporating both features here.

The tests done on Fermi GPU show that the FPS is affected to some extent when using Integer indexing. However, when using Kepler cards, it seems to be completely unaffected.

Configuration	Total Time	Render Time	Postprocess Time	FPS
Standard	22	13.9	0.29	60
Integer only	26	15.75	0.29	56
Stream Compaction only	22	13.8	0.3	56
Both	26	15.75	0.24	60

1b) Here we obtain the same statistics as 1a but with full screen window mode of Gigavoxels ON.

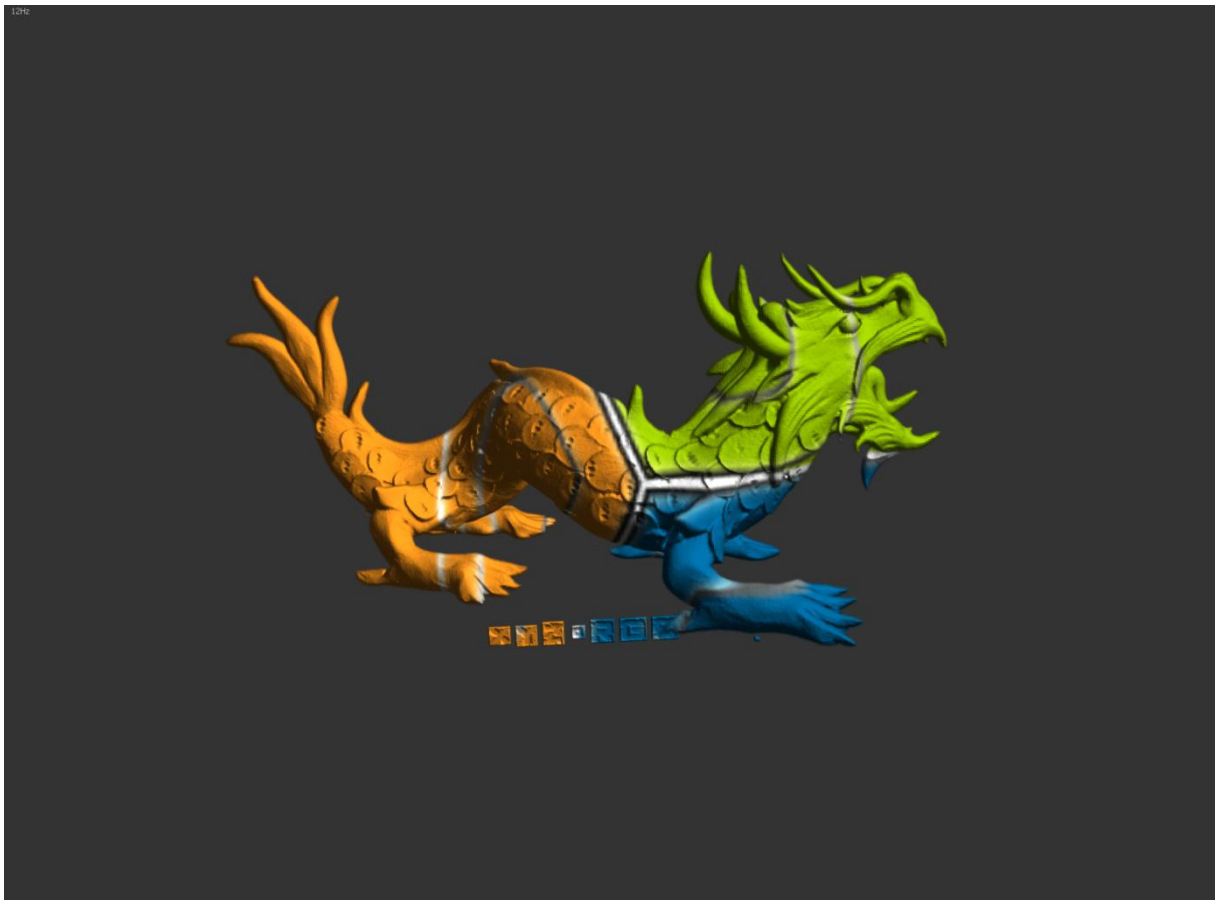
Note that in these cases the model is tilted on the screen.

Configuration	Total Time	Render Time	Postprocess Time	FPS
Standard	64	54.7	0.32	16
Integer only	73	64.4	0.32	15
Stream Compaction only	63	54.7	0.32	17
Both	74	64.4	0.29	15

(file qglviewe1.xml)

2a) We change the position of model here to orient it in a more straight position with respect to the camera.

Configuration	Total Time	Render Time	Postprocess Time	FPS
Standard	27	15.2	0.72	57
Integer only	26	17.45	0.29	50
Stream Compaction only	28	15.2	0.36	57
Both	26	17.44	0.25	51



2b) Here the camera position is same as in 2a but with the fullscreen mode ON.

Configuration	Total Time	Render Time	Postprocess Time	FPS
Standard	72	62.8	0.30	15
Integer only	81	74.1	0.33	12
Stream Compaction only	71	62.7	0.27	14
Both	81	74.0	0.31	13

(file qglviewe2.xml)

Some statistics from Kepler card

On a Kepler card (as tested on Pascal's machine), float and integer behave similarly with no difference in frame rates, both in normal and full screen mode.

Stream Compaction exclusively

In order to overstrain the stream compaction, we did two more tests.

3a) We clear the cache every 100 frames thereby increasing the compaction by several folds. The first table is for the titled dragon configuration and the second one for straight configuration.

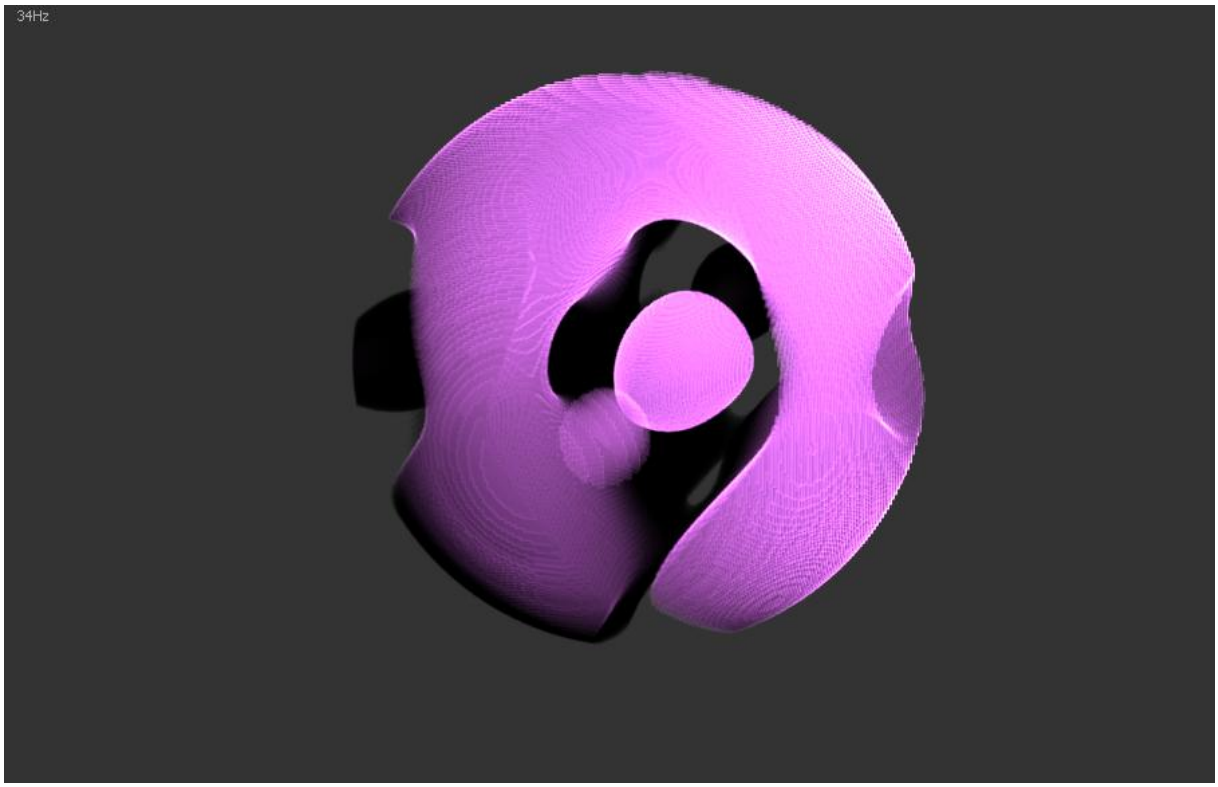
Configuration	Render Time	Postprocess Time
cudppCompact	15.4	8.0
Cuda Compact Chalmers	15.4	7.7
cudppCompact	45.75	8.02
Cuda Compact Chalmers (FS)	45.76	7.9

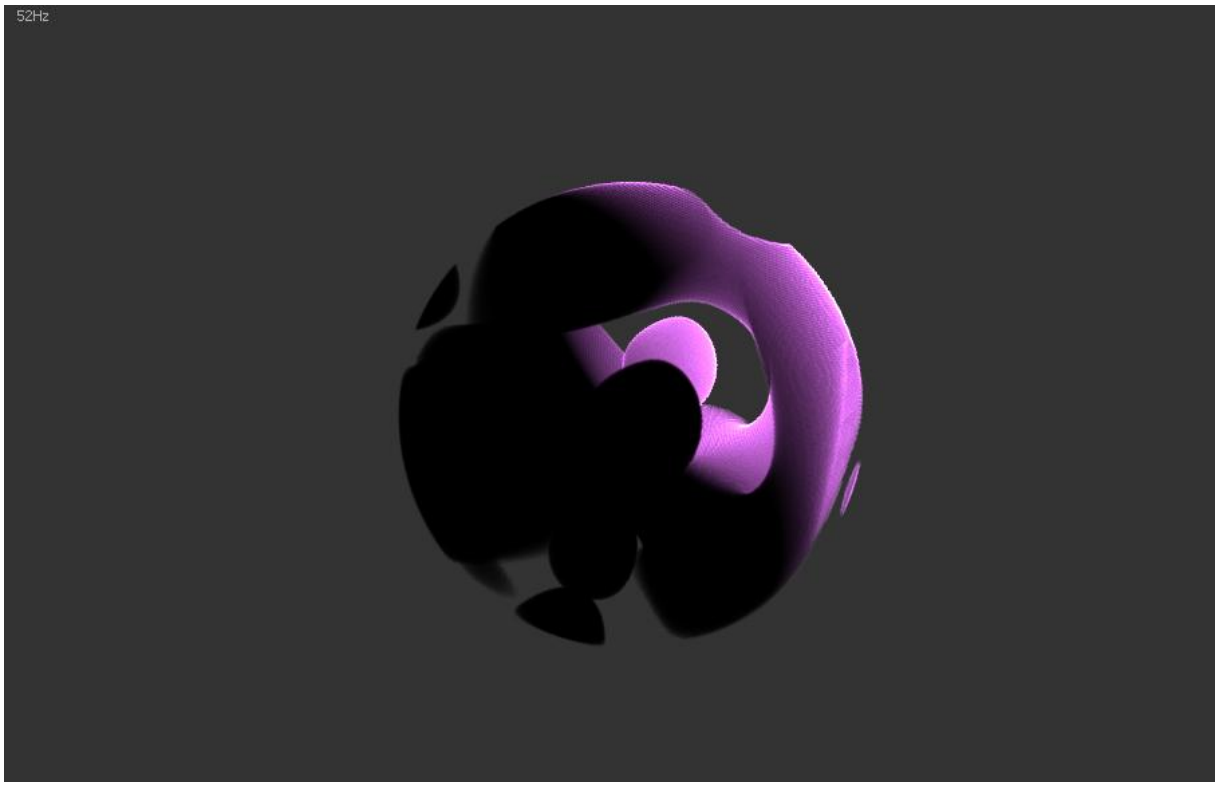
Configuration	Render Time	Postprocess Time
cudppCompact	23.1	8.2
Cuda Compact Chalmers	23.1	8.05
cudppCompact	37.2	8.3
Cuda Compact Chalmers (FS)	37.1	8.2

3b) In this test, we overloaded the stream compaction part by putting it in a ***for*** loop. Here we do not clear the cache but just let the ***for*** loop run over a selected portion of stream compaction which makes sure that a significant number of stream compactions are done every frame.

Configuration	FPS
cudppCompact	30
Cuda Compact Chalmers	31
cudppCompact	6.5
Cuda Compact Chalmers (FS)	6.5

HyperTexture generated

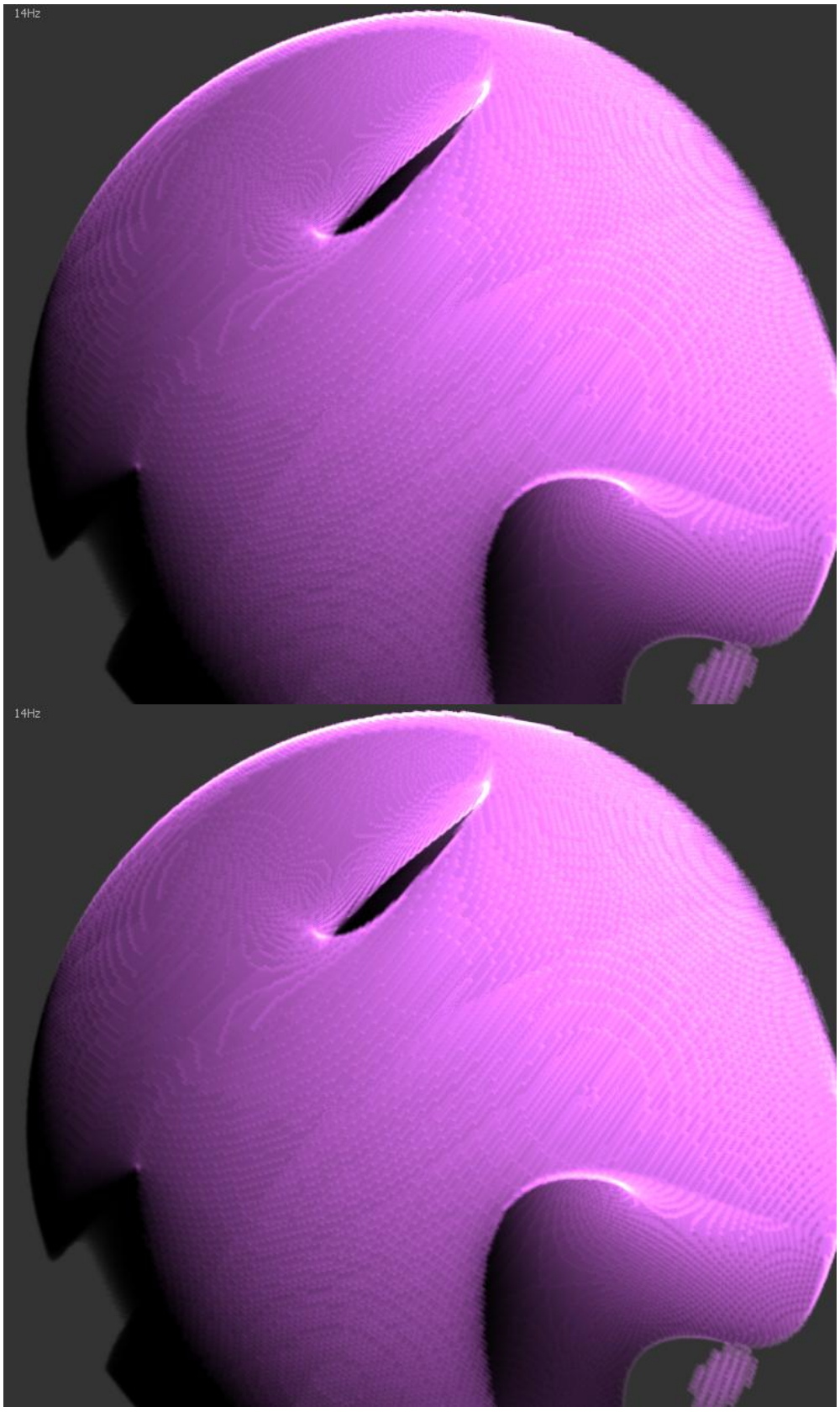




With and without border correction

The ray transition from empty to filled node is backtracked in first image to account for missing samples and not in the second one(frame rates remain the same both in normal and fullscreen mode)

(file qglviewe6.xml)


14Hz
14Hz

Traversing nodes and traversing bricks →Cost analysis

(a) Traversing full normal scene with simple sphere with subdivision for the tree enforced till maximum level for nodes (with drawing): 49 fps

(a1) Traversing full normal scene with simple sphere with subdivision for the tree enforced till maximum level for nodes (without drawing): 47

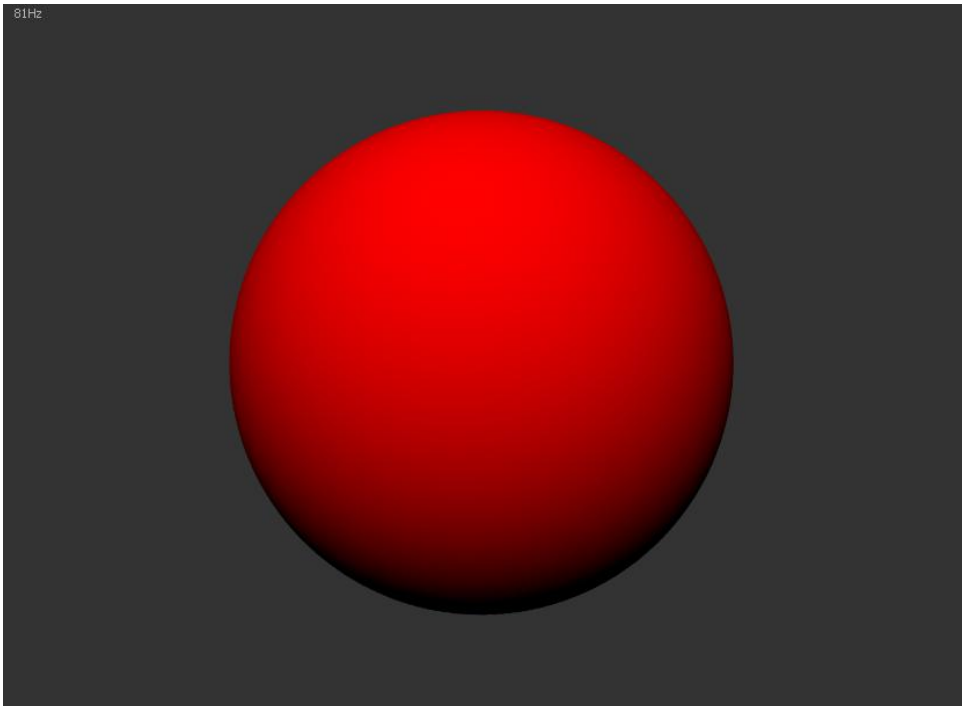
(a2) Traversing full normal scene with simple sphere with subdivision for the tree enforced till maximum level for nodes (without drawing & without brick sampling): 62

(b) Traversing full normal scene with simple sphere: 103

(b1) Traversing full normal scene with simple sphere (without drawing): 55

(b2) Traversing full normal scene with simple sphere (without drawing & without brick sampling): 84

(file qglviewe3.xml for sphere)



(a) Traversing full normal scene with cube with subdivision for the tree enforced till maximum level for nodes (with drawing): 500 - 600

(a1) Traversing full normal scene with cube with subdivision for the tree enforced till maximum level for nodes (without drawing): 124

(a2) Traversing full normal scene with cube with subdivision for the tree enforced till maximum level for nodes (without drawing & without brick sampling): 187

(b) Traversing full normal scene with cube: 500 - 600

(b1) Traversing full normal scene with cube (without drawing): 124

(b2) Traversing full normal scene with cube (without drawing & without brick sampling): 187

(file qglviewe4.xml for cube)

(c) Traversing full normal scene with cube (hidden) with steps as small as 1/1000 of the original values : 40

(d) Traversing full normal scene with cube (hidden) with normal steps but repeating the brick sampling 1000 times : 8

Repeating the tests above for the cube but with LOD 0

(b) Traversing full normal scene with cube: 500 - 600

(b1) Traversing full normal scene with cube (without drawing): around 600

(b2) Traversing full normal scene with cube (without drawing & without brick sampling): 690

(file qglviewe4.xml for cube)

(c) Traversing full normal scene with cube (hidden) with steps as small as 1/1000 of the original values : 67

(d) Traversing full normal scene with cube (hidden) with normal steps but repeating the brick sampling 1000 times : 13

Changing brick sizes to compare performance

Simple sphere

Brick size(w/o border)	FPS	Max. depth	Total bricks	Total voxels	% border memory
2	196	5	37449	299592	87.5
4	168	5	37449	2396736	70.37
8	132	5	37449	19173888	48.8
12	120	5	37449	64711872	37.03
16	110	5	37449	153391104	29.77
20	116	5	37449	299592000	24.87
24	116	5	37449	64710144	21.35
28	56	5	4681		

Dynamic load (dragon)

Brick size(w/o border)	FPS	Max. depth	Total bricks	Total voxels	% border memory
2					
4	115	6	299593	19173952	70.37
8	60	6	299593	153391616	48.8
12	50	5	37449	64711872	37.03
16	39	5	37449	153391104	29.77
20	47	4	4681	37448000	24.87
24	35	4	4681	64710144	21.35

Comparing frame rates based on silhouettes

It seems that this could be an important direction for further consideration. Silhouettes determine a lot on the efficiency front as could be seen in the table below (with same configuration file for all). The difference, for example, between sphere and spherical hypertexture is in the silhouettes added in the latter which already results in a significant change in the FPS.

Model	FPS Normal	FPS Full screen
Cube	377	139
Sphere	88	33
HyperTexture	36	11

(file qglviewe5.xml)