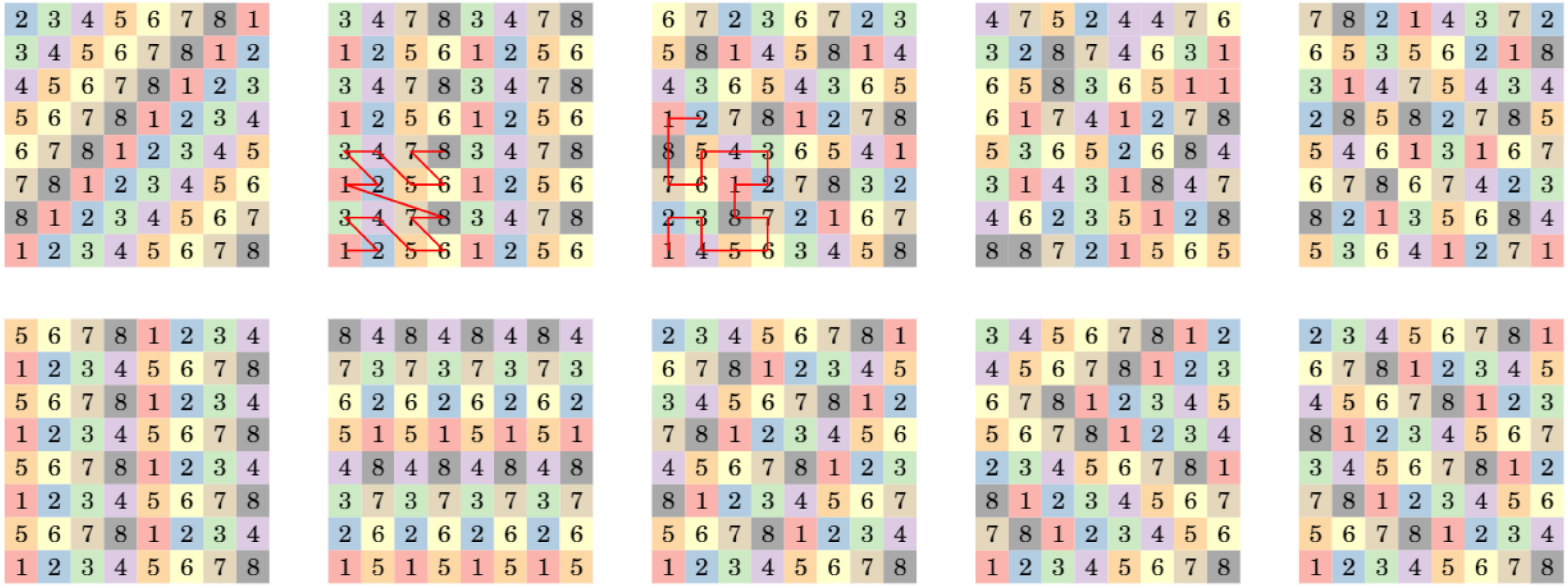
Effective Static Bin Patterns for Sort-Middle Rendering



2	3	4	5	6	7	8	1
3	4	5	6	7	8	1	2
4	5	6	7	8	1	2	3
5	6	7	8	1	2	3	4
6	7	8	1	2	3	4	5
7	8	1	2	3	4	5	6
8	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8

3	4	7	8	3	4	7	8
1	2	5	6	1	2	5	6
3	4	7	8	3	4	7	8
1	2	5	6	1	2	5	6
3	4	7	8	3	4	7	8
1	2	5	6	1	2	5	6
3	4	7	8	3	4	7	8
1	2	5	6	1	2	5	6

6	7	2	3	6	7	2	3
5	8	1	4	5	8	1	4
4	3	6	5	4	3	6	5
1	2	7	8	1	2	7	8
8	5	4	3	6	5	4	1
7	6	1	2	7	8	3	2
2	3	8	7	2	1	6	7
1	4	5	6	3	4	5	8

4	7	5	2	4	4	7	6
3	2	8	7	4	6	3	1
6	5	8	3	6	5	1	1
6	1	7	4	1	2	7	8
5	3	6	5	2	6	8	4
3	1	4	3	1	8	4	7
4	6	2	3	5	1	2	8
8	8	7	2	1	5	6	5

7	8	2	1	4	3	7	2
6	5	3	5	6	2	1	8
3	1	4	7	5	4	3	4
2	8	5	8	2	7	8	5
5	4	6	1	3	1	6	7
6	7	8	6	7	4	2	3
8	2	1	3	5	6	8	4
5	3	6	4	1	2	7	1

5	6	7	8	1	2	3	4
1	2	3	4	5	6	7	8
5	6	7	8	1	2	3	4
1	2	3	4	5	6	7	8
5	6	7	8	1	2	3	4
1	2	3	4	5	6	7	8
5	6	7	8	1	2	3	4
1	2	3	4	5	6	7	8

8	4	8	4	8	4	8	4
7	3	7	3	7	3	7	3
6	2	6	2	6	2	6	2
5	1	5	1	5	1	5	1
4	8	4	8	4	8	4	8
3	7	3	7	3	7	3	7
2	6	2	6	2	6	2	6
1	5	1	5	1	5	1	5

2	3	4	5	6	7	8	1
6	7	8	1	2	3	4	5
3	4	5	6	7	8	1	2
7	8	1	2	3	4	5	6
4	5	6	7	8	1	2	3
8	1	2	3	4	5	6	7
5	6	7	8	1	2	3	4
1	2	3	4	5	6	7	8

3	4	5	6	7	8	1	2
4	5	6	7	8	1	2	3
6	7	8	1	2	3	4	5
5	6	7	8	1	2	3	4
2	3	4	5	6	7	8	1
8	1	2	3	4	5	6	7
7	8	1	2	3	4	5	6
1	2	3	4	5	6	7	8

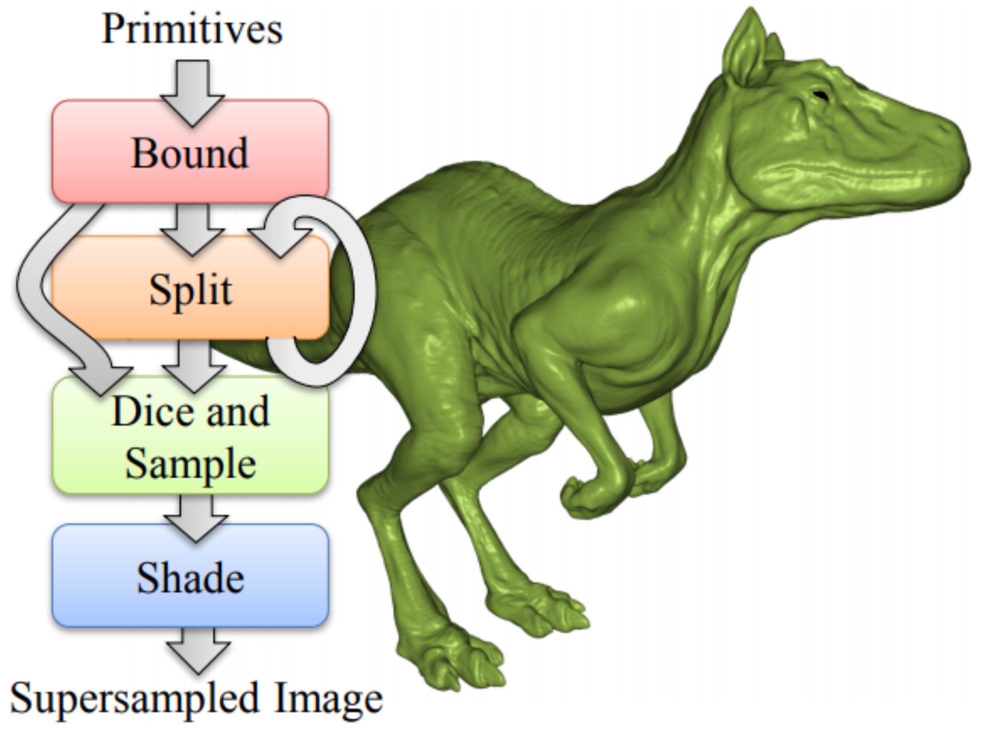
2	3	4	5	6	7	8	1
6	7	8	1	2	3	4	5
4	5	6	7	8	1	2	3
8	1	2	3	4	5	6	7
3	4	5	6	7	8	1	2
7	8	1	2	3	4	5	6
5	6	7	8	1	2	3	4
1	2	3	4	5	6	7	8

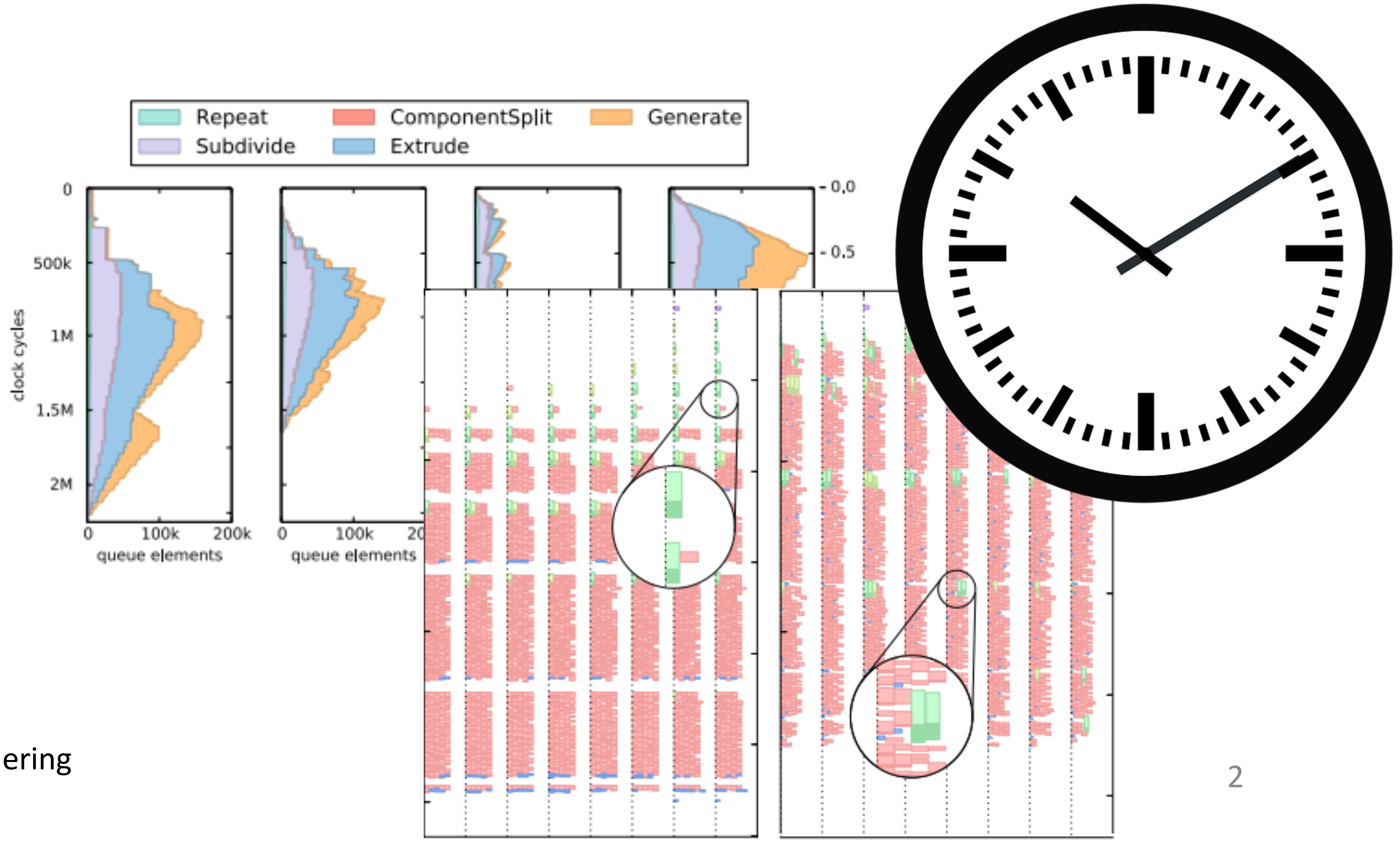
Bernhard Kerbl, Michael Kenzel, Dieter Schmalstieg and Markus Steinberger

Graz University of Technology

Consider the following...

- One day, you are tasked with looking into micro-polygon rendering with complex fragment shading
- Before full development, your supervisor demands performance estimates
- Maximum triangle load, FPS, etc.
- So you get to work...


Primitives → Bound → Split → Dice and Sample → Shade → Supersampled Image



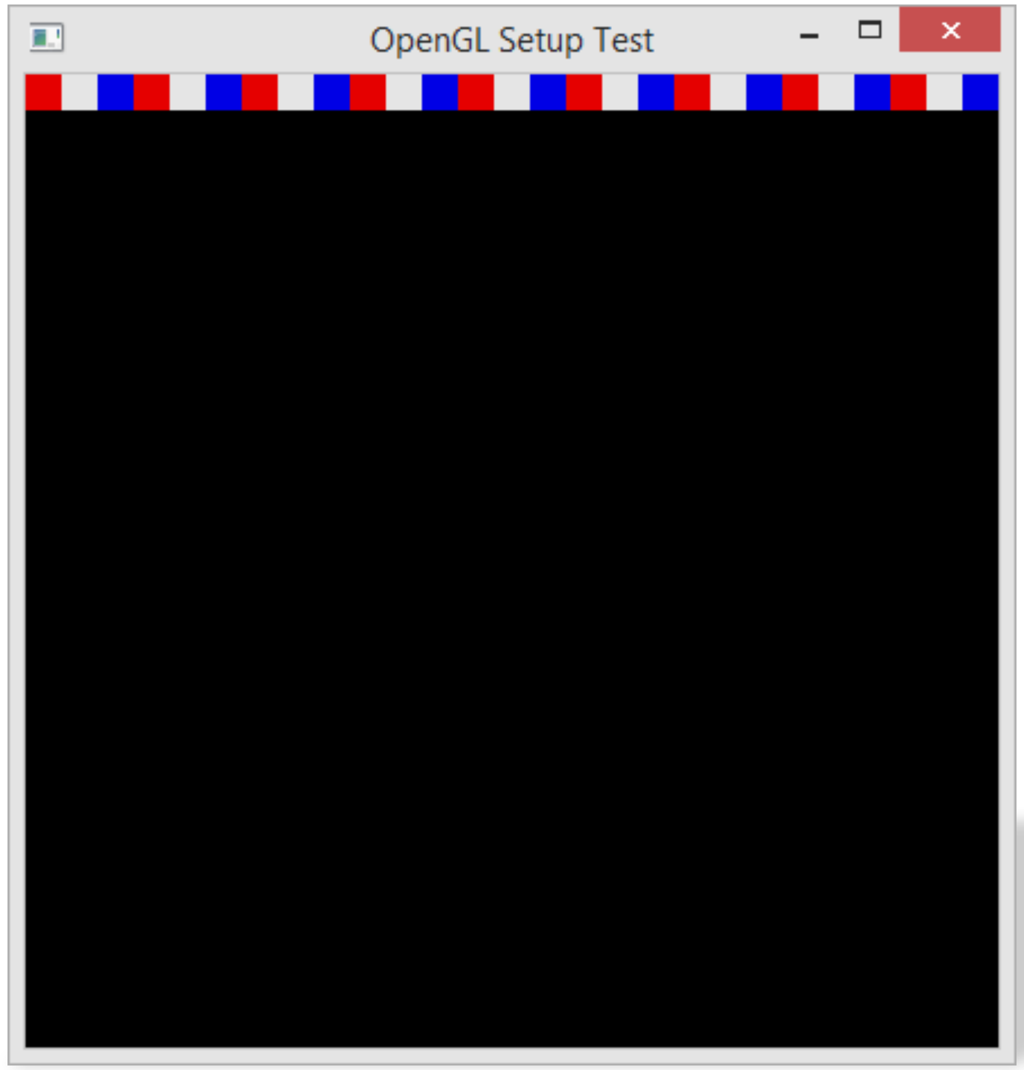
Many pixel-sized triangles I

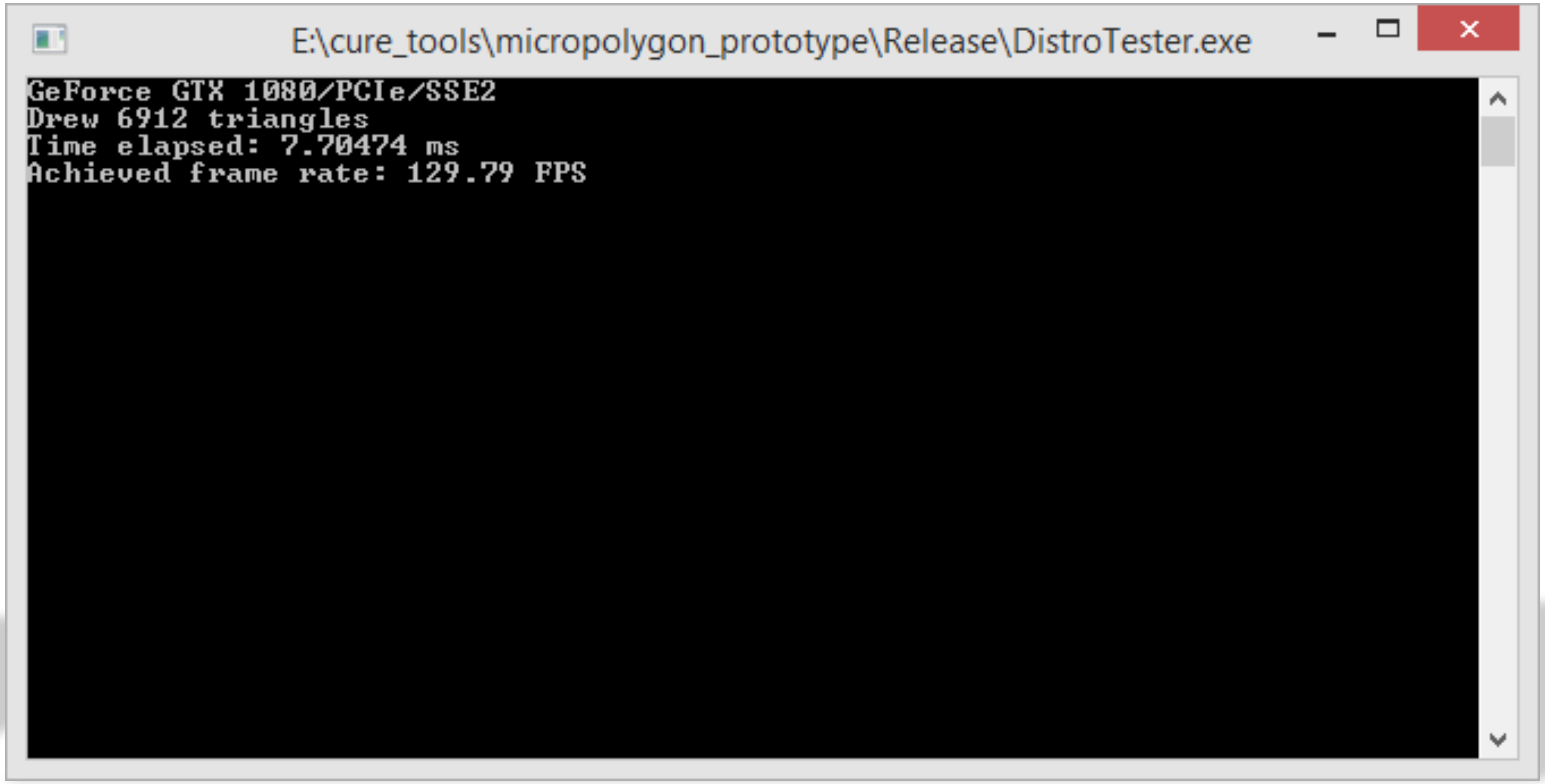
- Generate triangles just big enough to cover single pixels
- Simulate expensive shading with fixed number of operations
- Put triangles side by side for convenience
- Tentative goal: Approx. 7,000 triangles rendered @ 120 FPS

Many pixel-sized triangles II

- Put tiles next to each other (256 triangles per tile, 16x16)

  130 FPS ✓

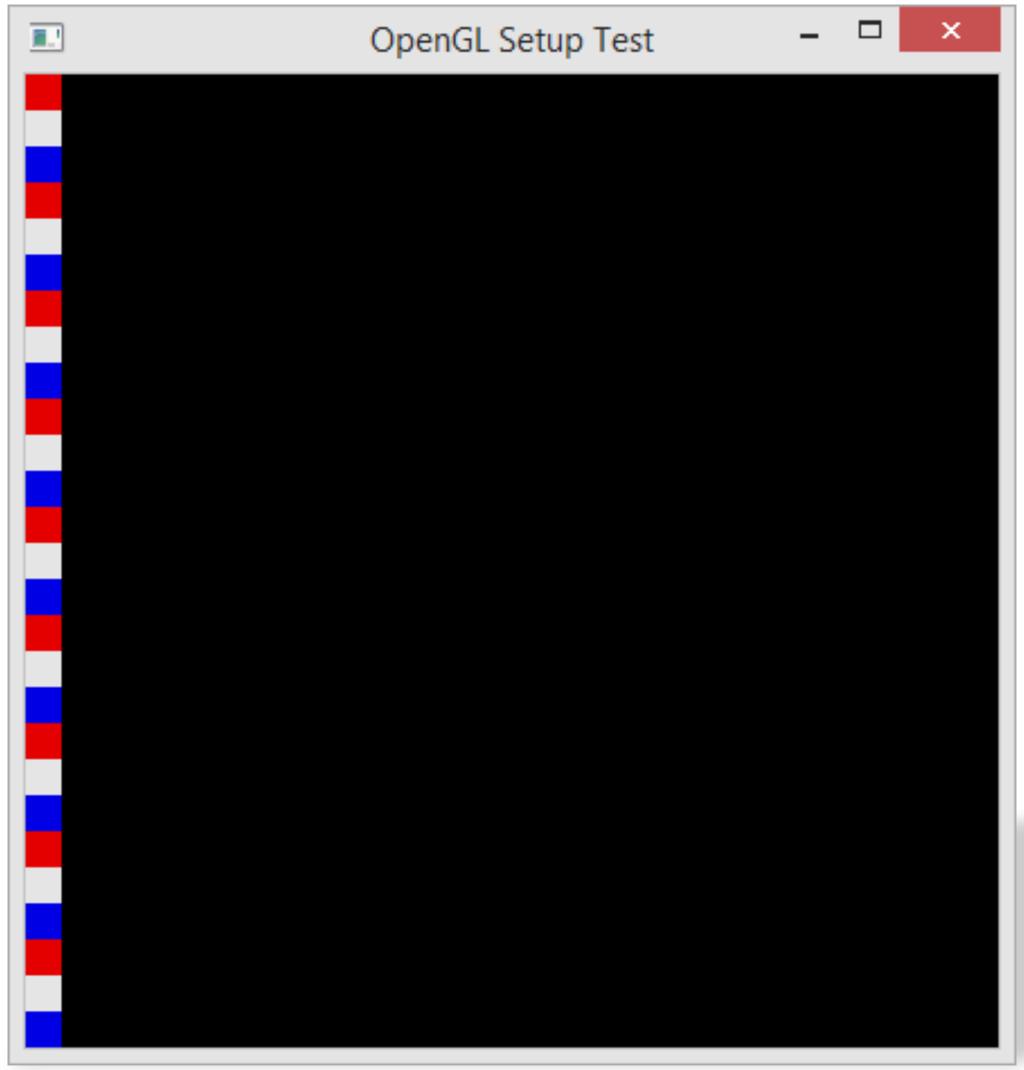


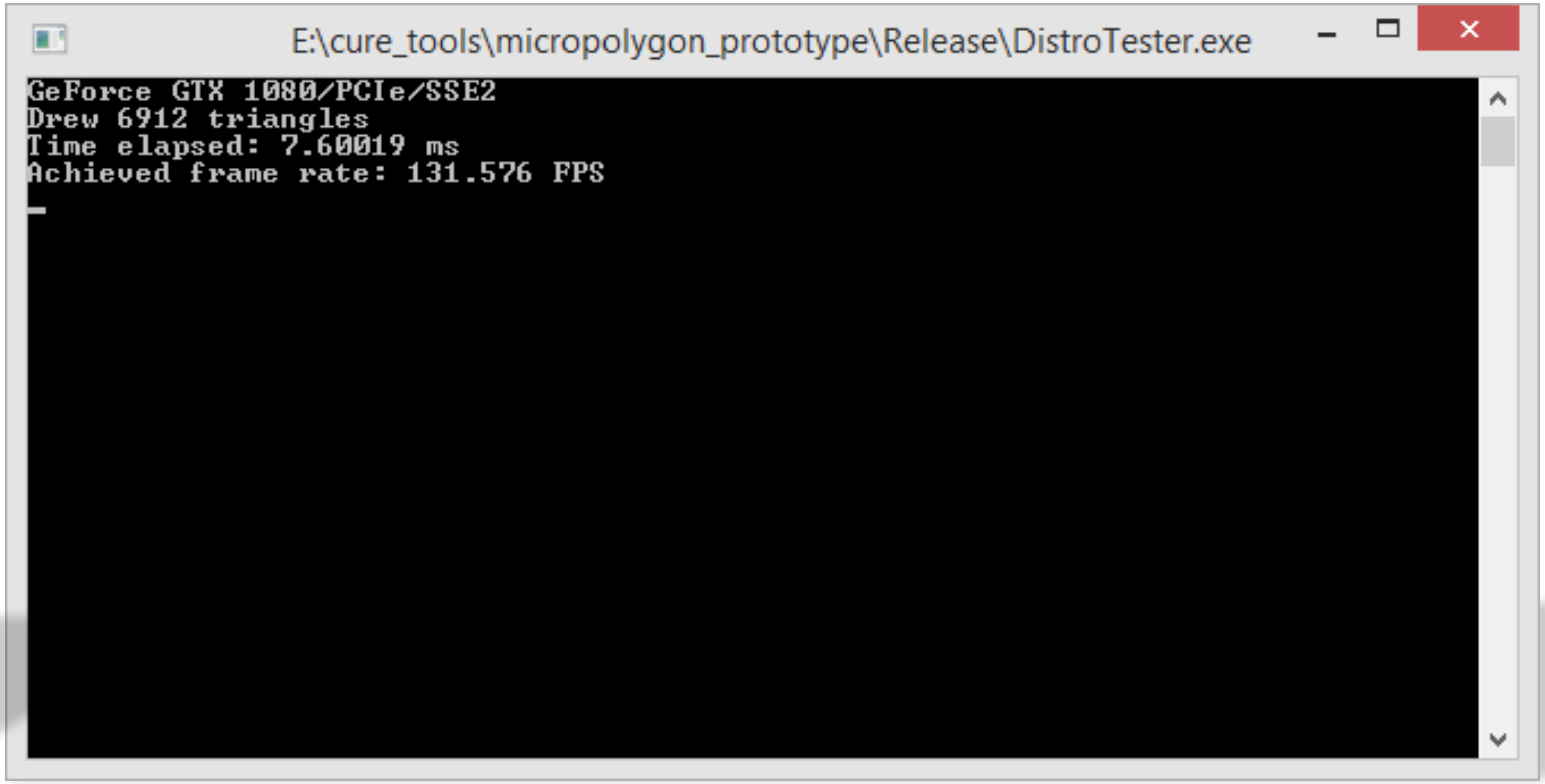

E:\cure_tools\micropolygon_prototype\Release\DistroTester.exe

```
GeForce GTX 1080/PCIe/SSE2
Drew 6912 triangles
Time elapsed: 7.70474 ms
Achieved frame rate: 129.79 FPS
```

Many pixel-sized triangles III

- Horizontal or vertical, performance remains clearly above 120 FPS

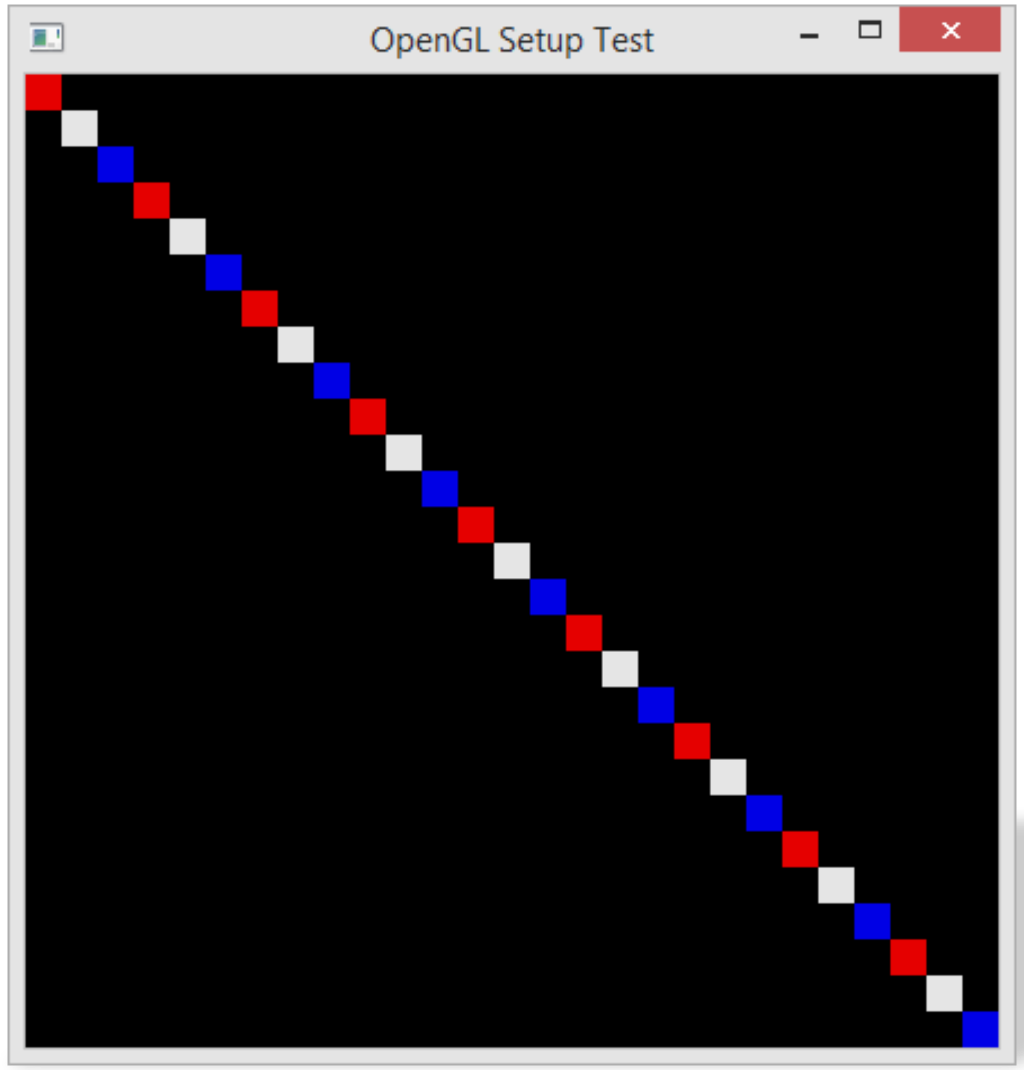


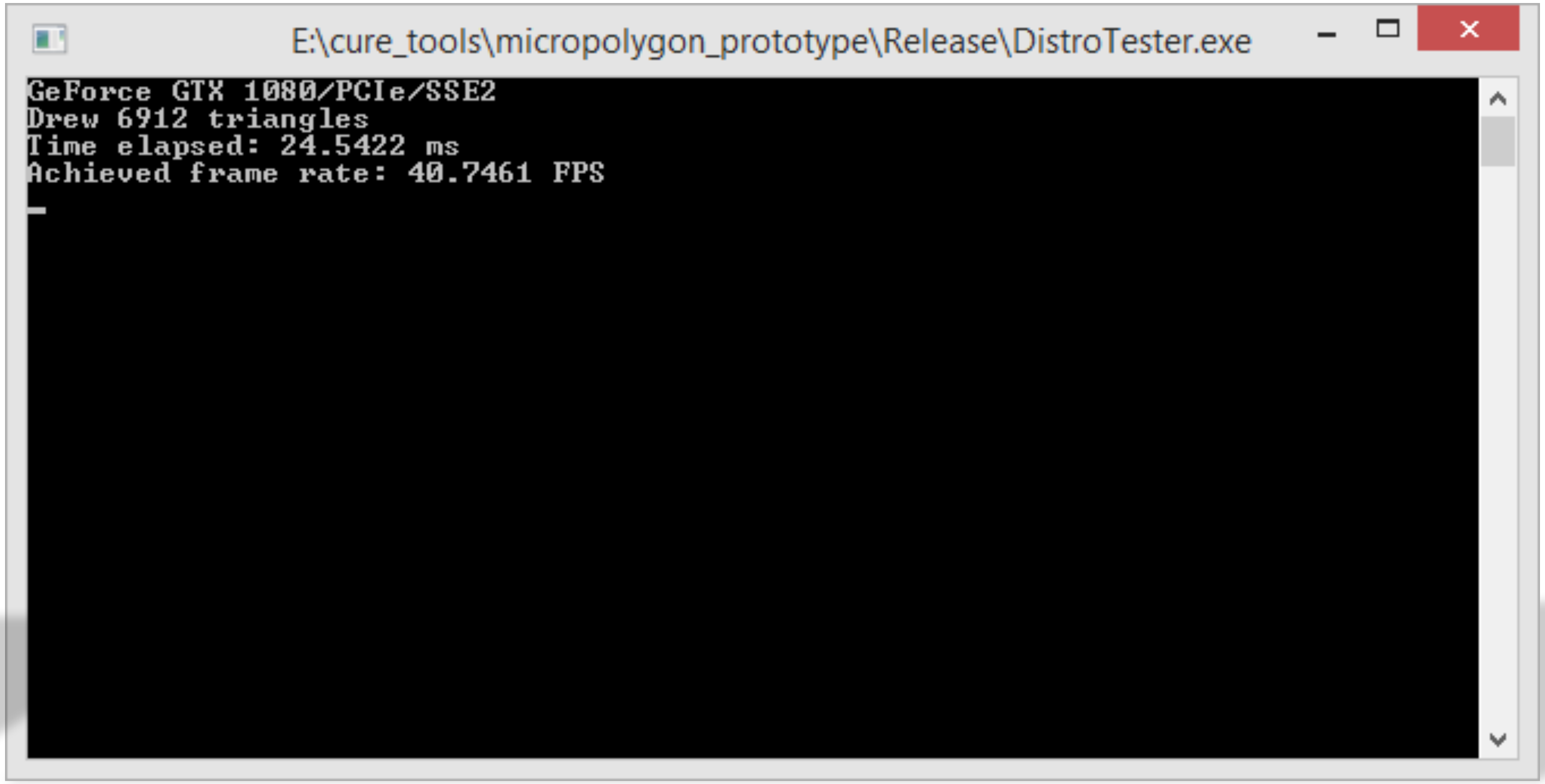

E:\cure_tools\micropolygon_prototype\Release\DistroTester.exe

```
GeForce GTX 1080/PCIe/SSE2
Drew 6912 triangles
Time elapsed: 7.60019 ms
Achieved frame rate: 131.576 FPS
_
```

Many pixel-sized triangles IV

- Diagonal line, because it fills the window nicely...
  Hold on, 40 FPS?

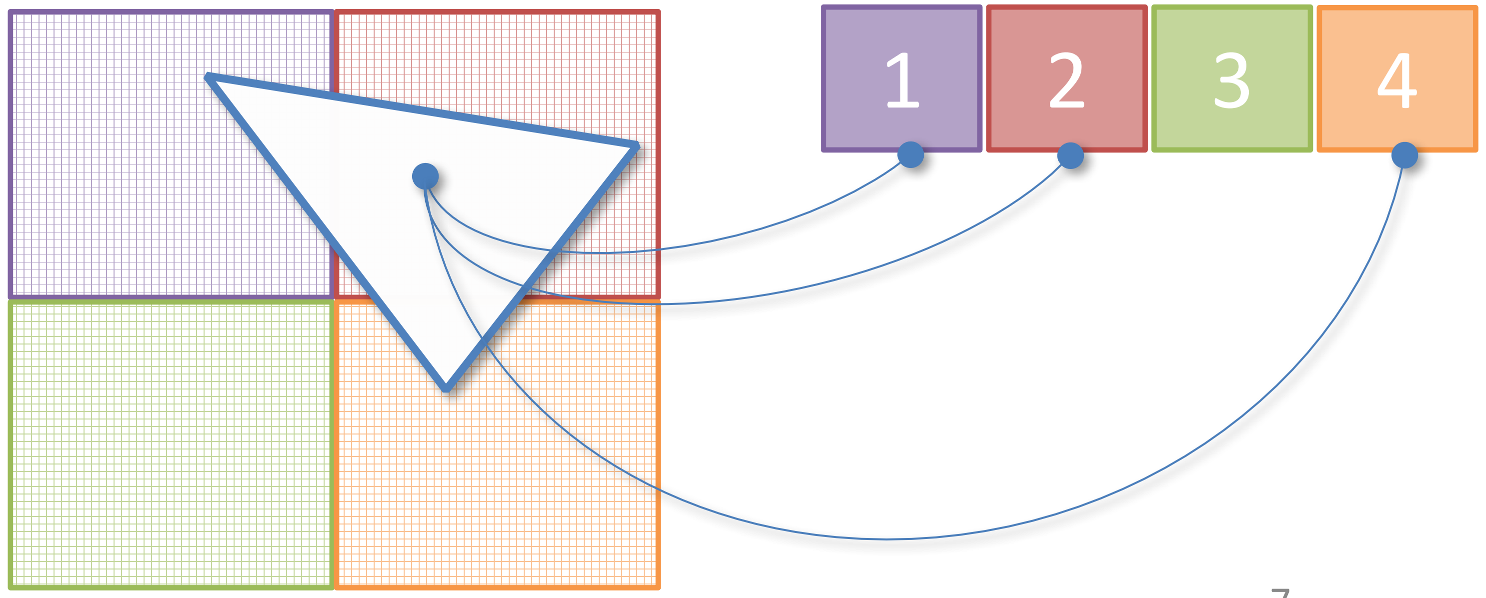



E:\cure_tools\micropolygon_prototype\Release\DistroTester.exe

```
GeForce GTX 1080/PCIe/SSE2
Drew 6912 triangles
Time elapsed: 24.5422 ms
Achieved frame rate: 40.7461 FPS
_
```

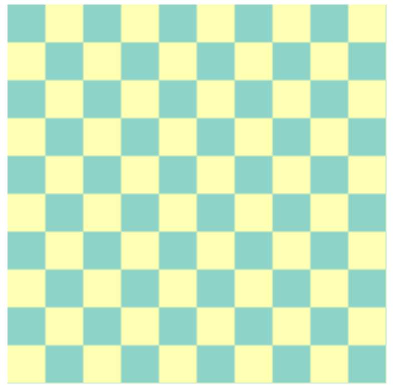
Fragment Load Balancing on GPUs

- Rendering pipeline geared towards sort-middle rendering
- Assigns clip-space triangles to (rectangular) bins for shading
- Fixed assignment rules, one-to-many processor/bin mapping
- Static 2D binning (or “tiling”)

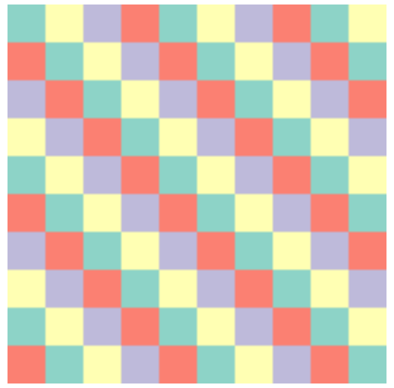


Common Binning Patterns

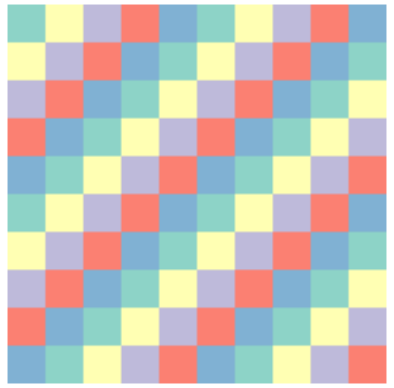
- NVIDIA



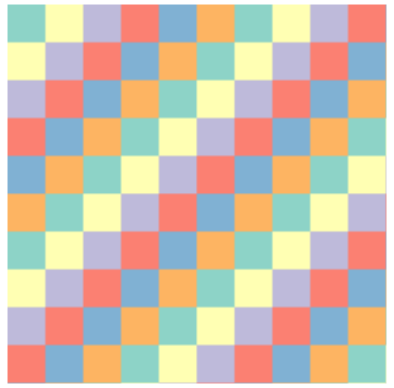
GTX 560 Ti



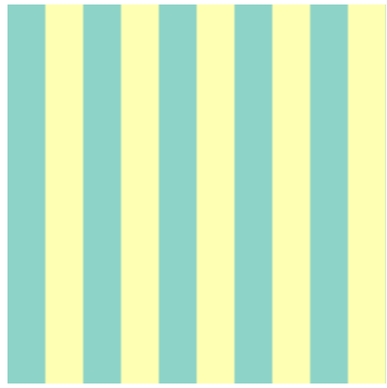
GTX 580/680



GTX 780 Ti

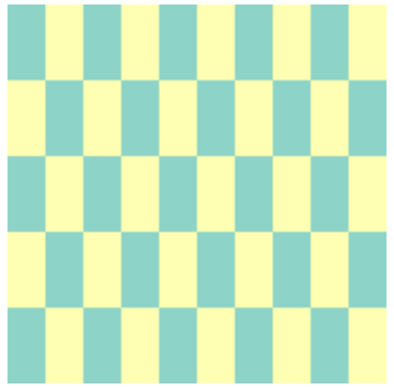


GTX Titan Xp

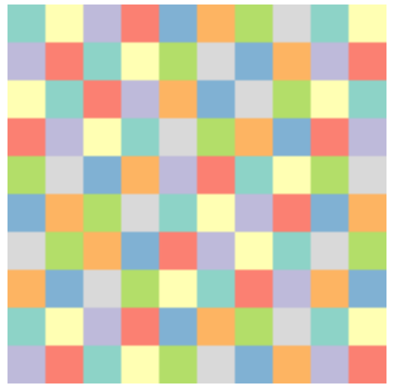


GTX 1060

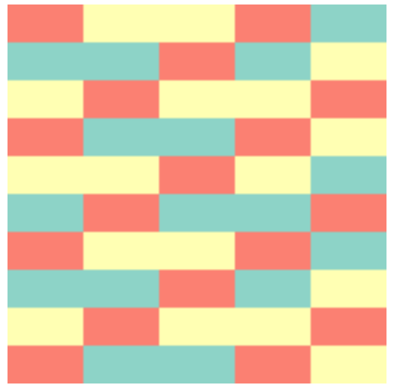
- AMD & Intel



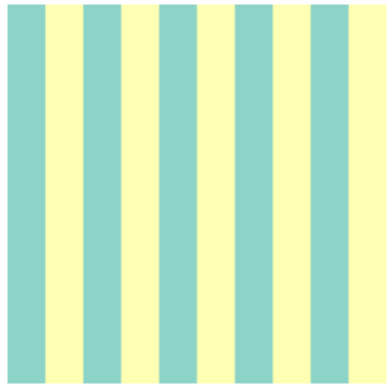
AMD 6770M



AMD R9 270X



Intel HD 530



Intel HD 4000

Rasterizers, Compute Units, GPCs

- Level/naming of balanced resources differ between vendors
  - NVIDIA: Graphics Processing Clusters (GPC)
  - AMD: Compute Units
  - …

- Focus on noticeable effect on shading performance (measurable)

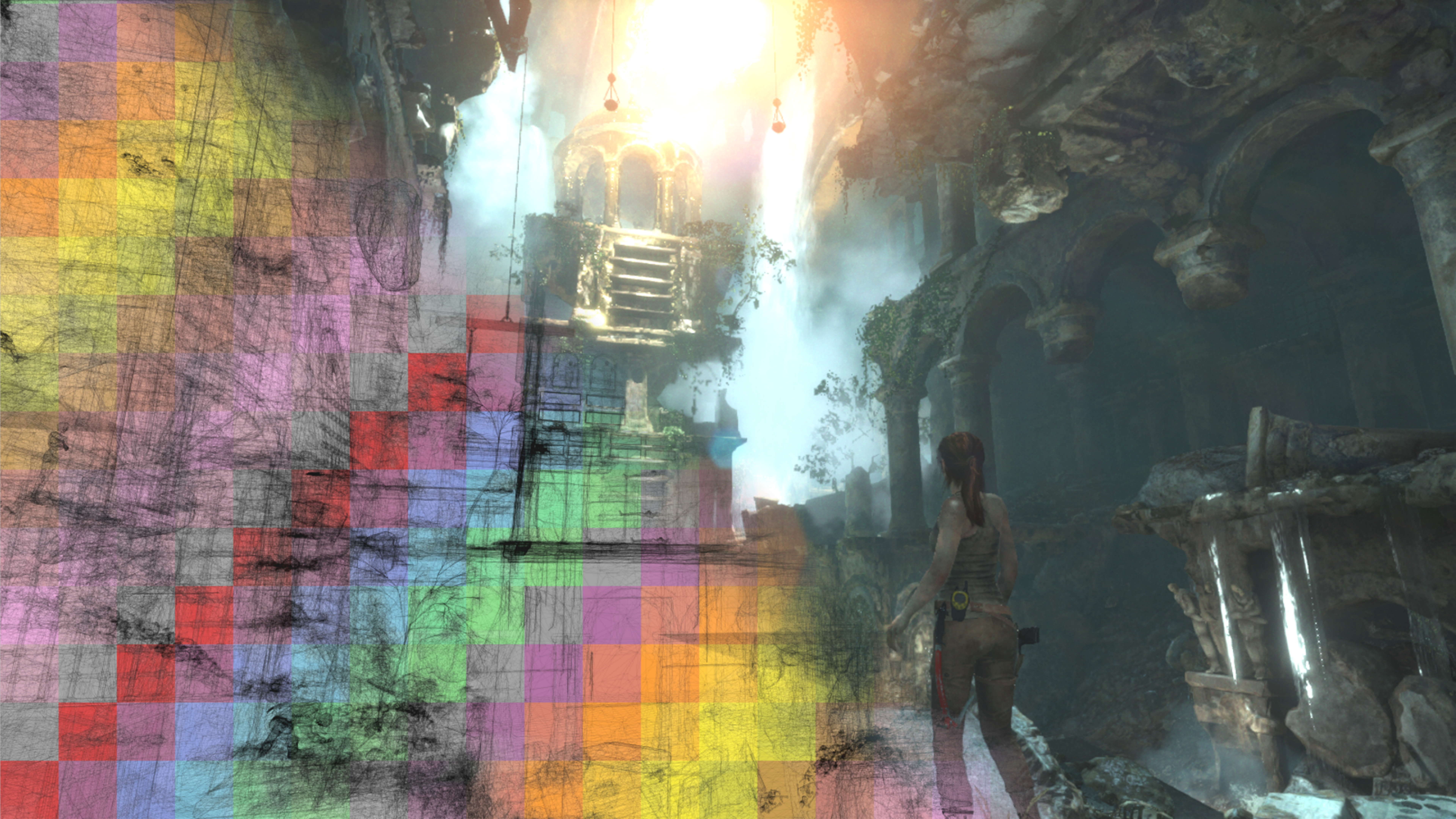
- We will henceforth refer to corresponding units as “rasterizers”

2D Binning Parameters

- Bin size in pixels
- Rasterizer capability (uniform/varying)
- Number of rasterizers employed
- Arrangement scheme (pattern)

2D Binning Parameters

- Bin size in pixels, fixed size assumed
- Rasterizer capability (uniform/varying), uniform assumed
- Number of rasterizers employed, should be independent
- Arrangement scheme (pattern)



Considered Criteria

- Based on realistic geometry data loads

1. Pattern space utilization

2. Clustering of geometry

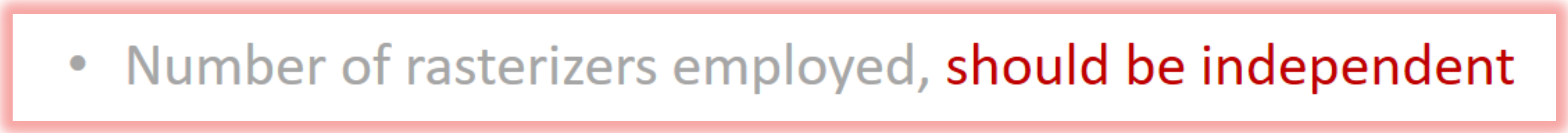
3. Orientation of rendered objects

Space Utilization

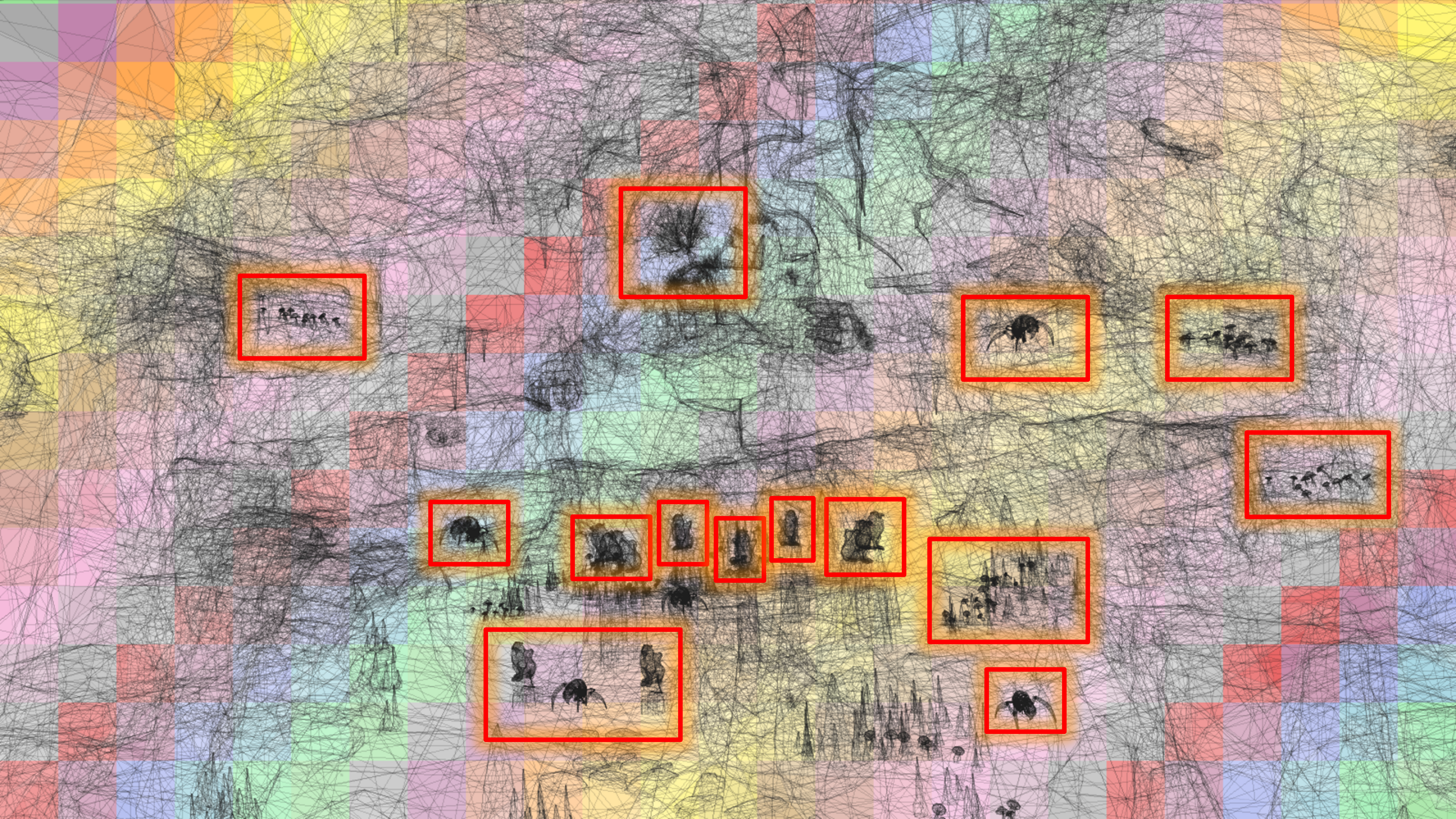
- Thought experiment: vertical pixel column pattern

- All N rasterizers are assigned a separate image column of width w

- If $(N-1)\times w >$ viewport width, at least one rasterizer is idle

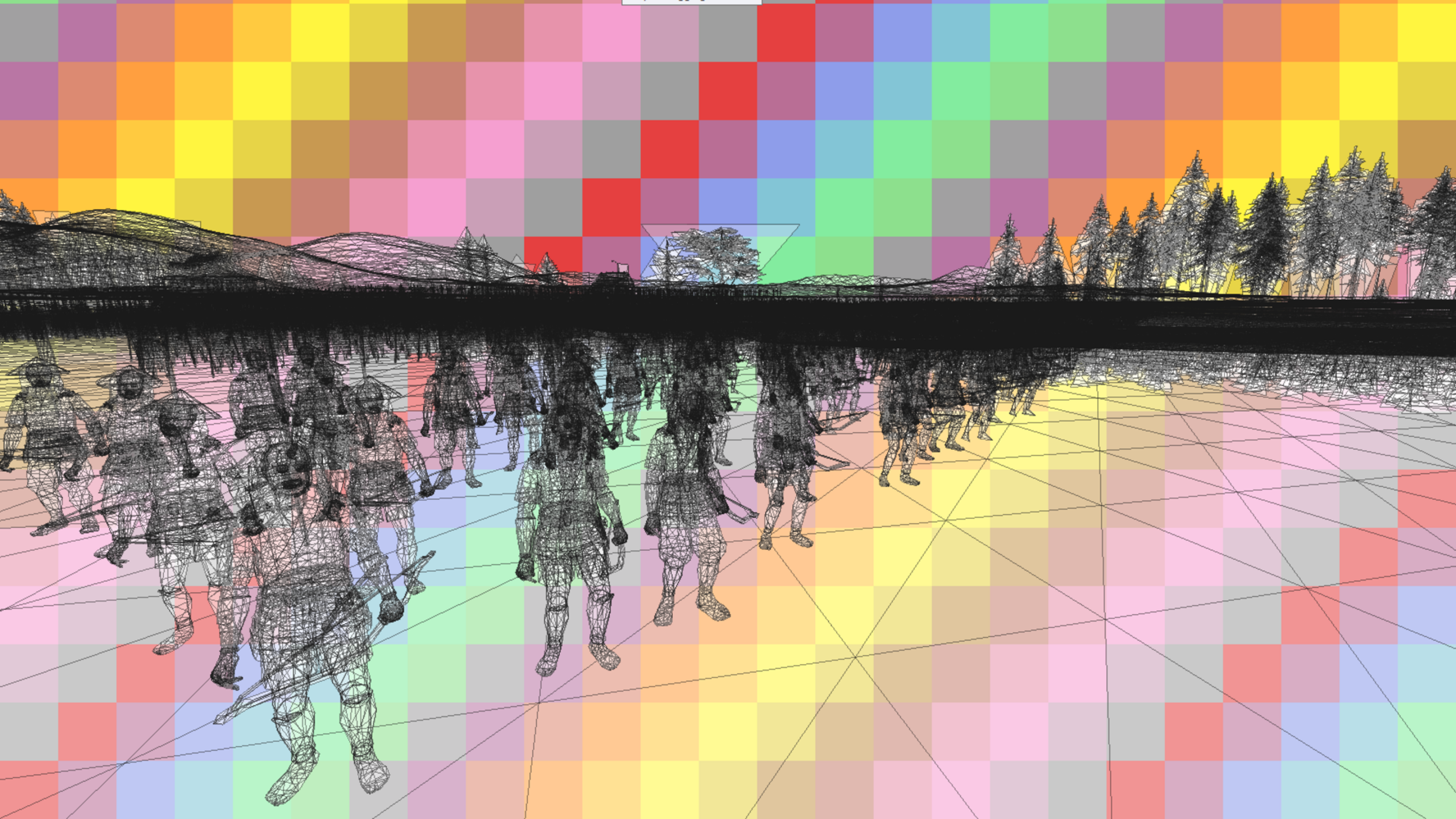


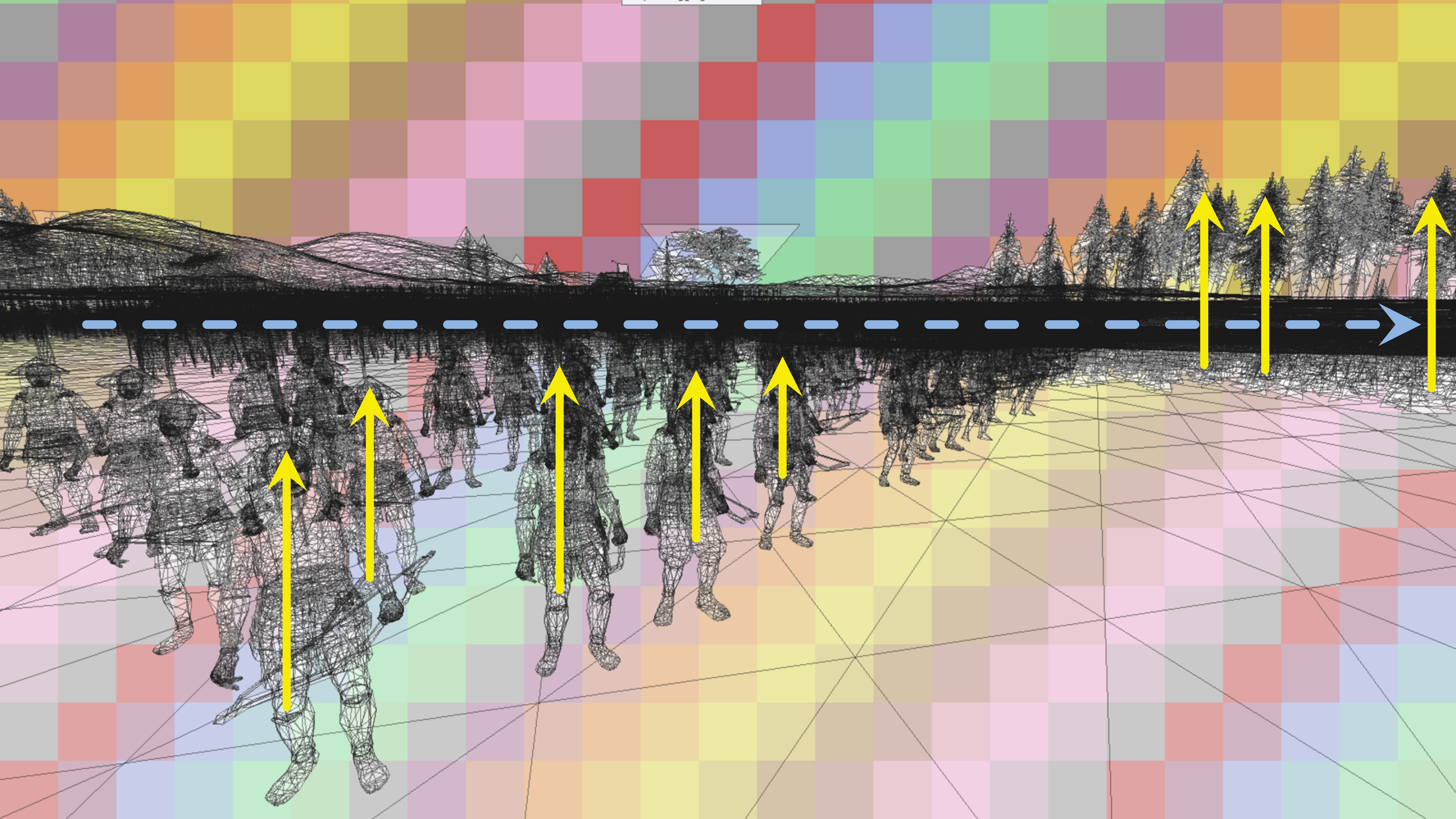
- Number of rasterizers employed, should be independent



Geometry clustering

- Large portions of total geometry confined to localized clusters
- Assigning rasterizer to bins in close proximity prevents balancing
- Experimentally confirmed in our paper for test suite of 200 scenes
- The greater the distance between bins for a rasterizer, the better

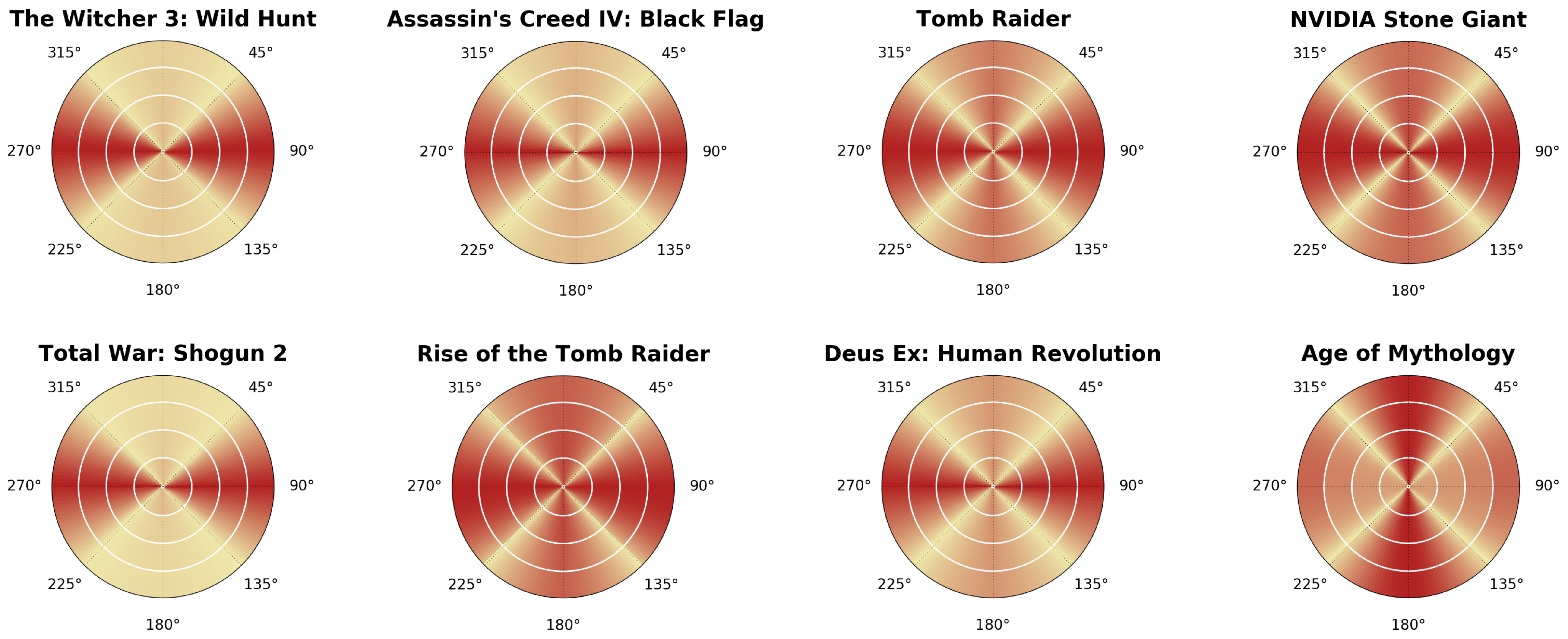




Geometry Orientation

- Assumption: most real-world objects align horizontally or vertically
- Combined influences of gravity, evolution and culture
- Consequently, perfectly diagonal structures are rare
- Requires realistic perspective (no tilted view)

Geometry Orientation



The Witcher 3: Wild Hunt

Assassin's Creed IV: Black Flag

Tomb Raider

NVIDIA Stone Giant

Total War: Shogun 2

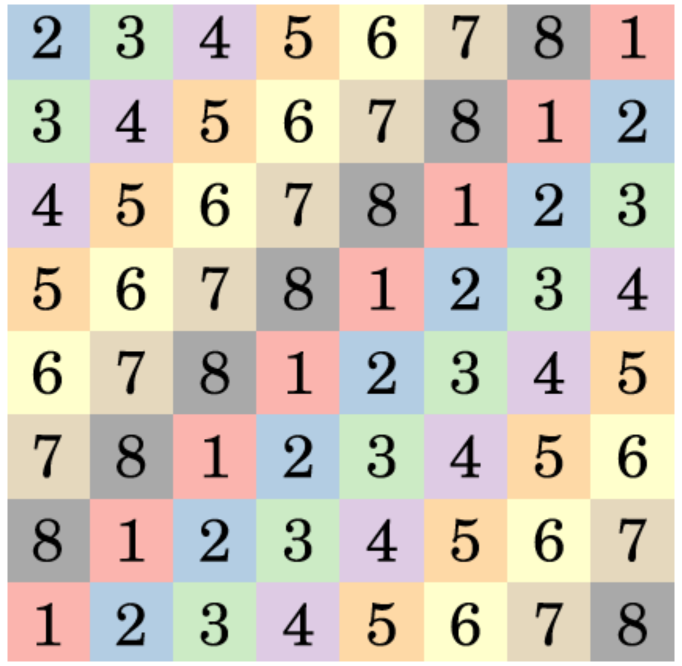
Rise of the Tomb Raider

Deus Ex: Human Revolution

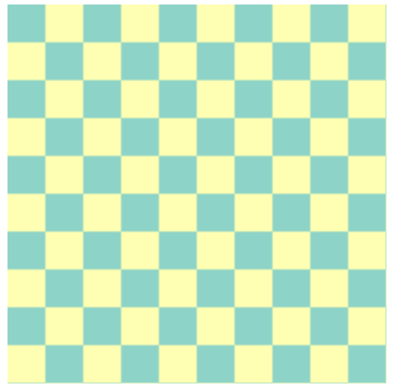
Age of Mythology

Geometry Orientation

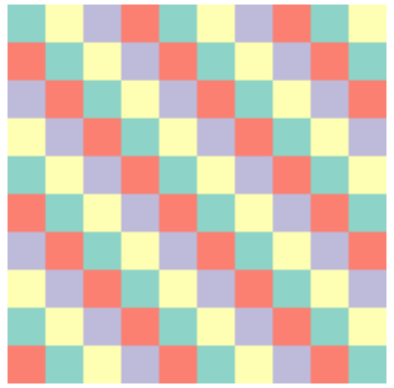
- Avoid horizontal and vertical repetitions in rasterizer assignment
- Diagonal pattern revisited → → →
- Frequently used in previous NVIDIA flagships



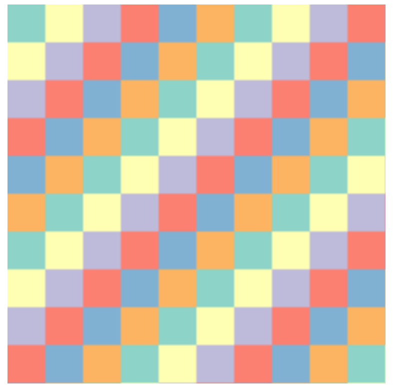
2	3	4	5	6	7	8	1
3	4	5	6	7	8	1	2
4	5	6	7	8	1	2	3
5	6	7	8	1	2	3	4
6	7	8	1	2	3	4	5
7	8	1	2	3	4	5	6
8	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8



GTX 560 Ti



GTX 580/680

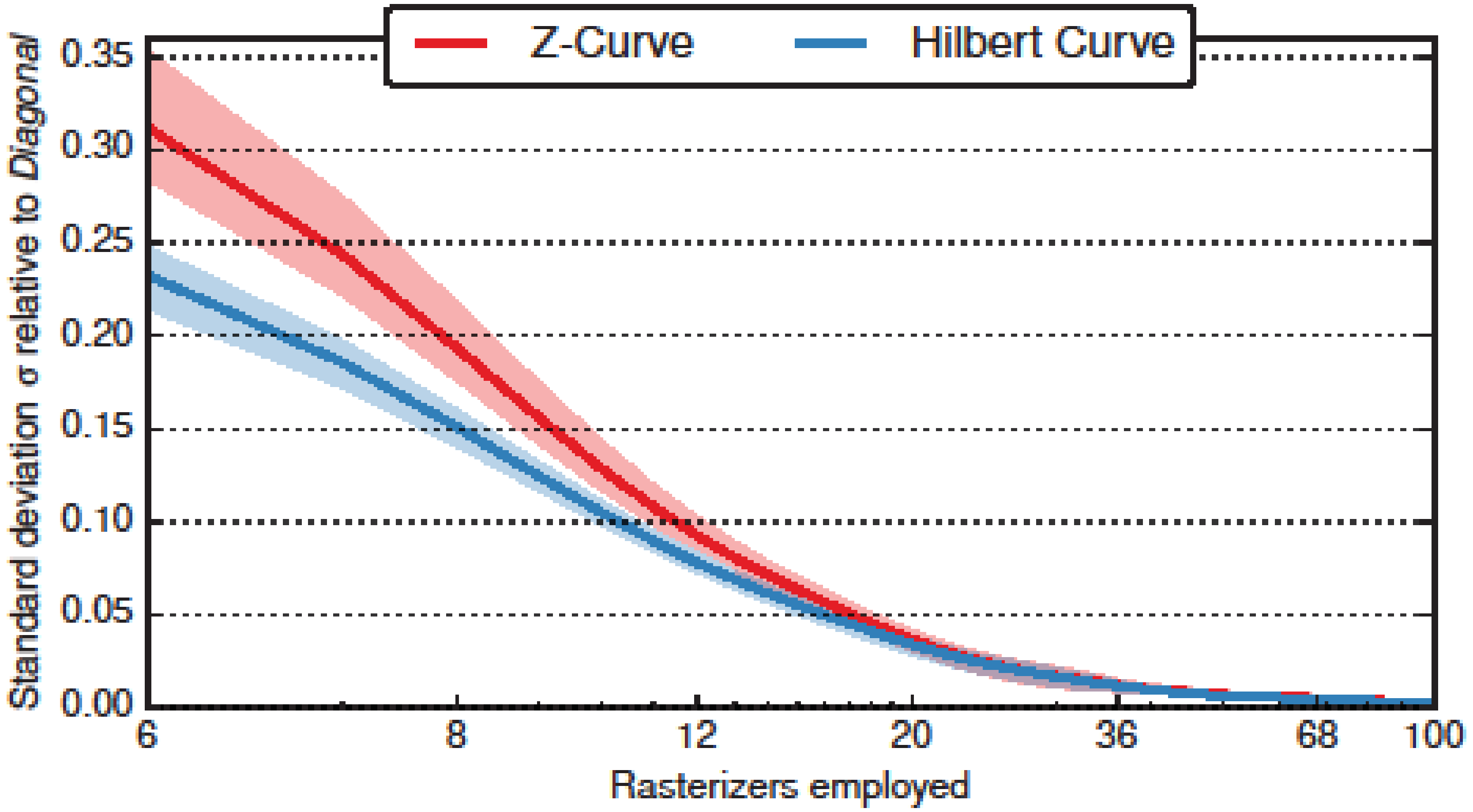


GTX Titan Xp

Evaluated Patterns

Space Filling Curves

- Commonly used for data storage and access patterns
- Arrange tiles according to curve (Z-Curve, Hilbert Curve)
- Rasterizer index = distance *mod* **N**
- Compare fragment load variance

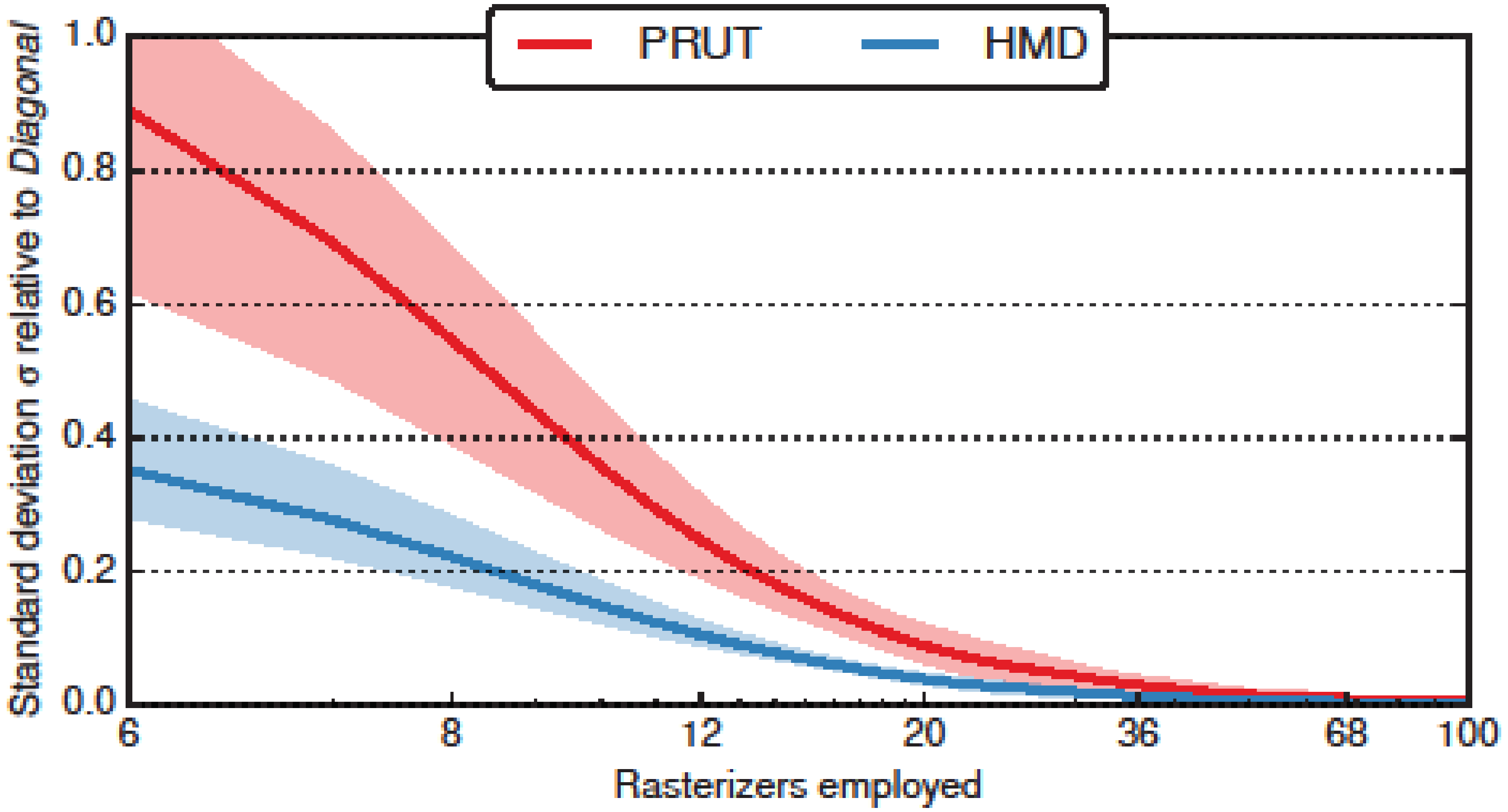


Randomization-based patterns

- Random number generator (**mt19937**) for both patterns

- Pseudo-random uniform distribution (PRUT) – no modifications

- Hierarch. max. distance (HMD)
  - Iterative dart throwing approach
  - Generate samples for each index
  - Always choose farthest sample

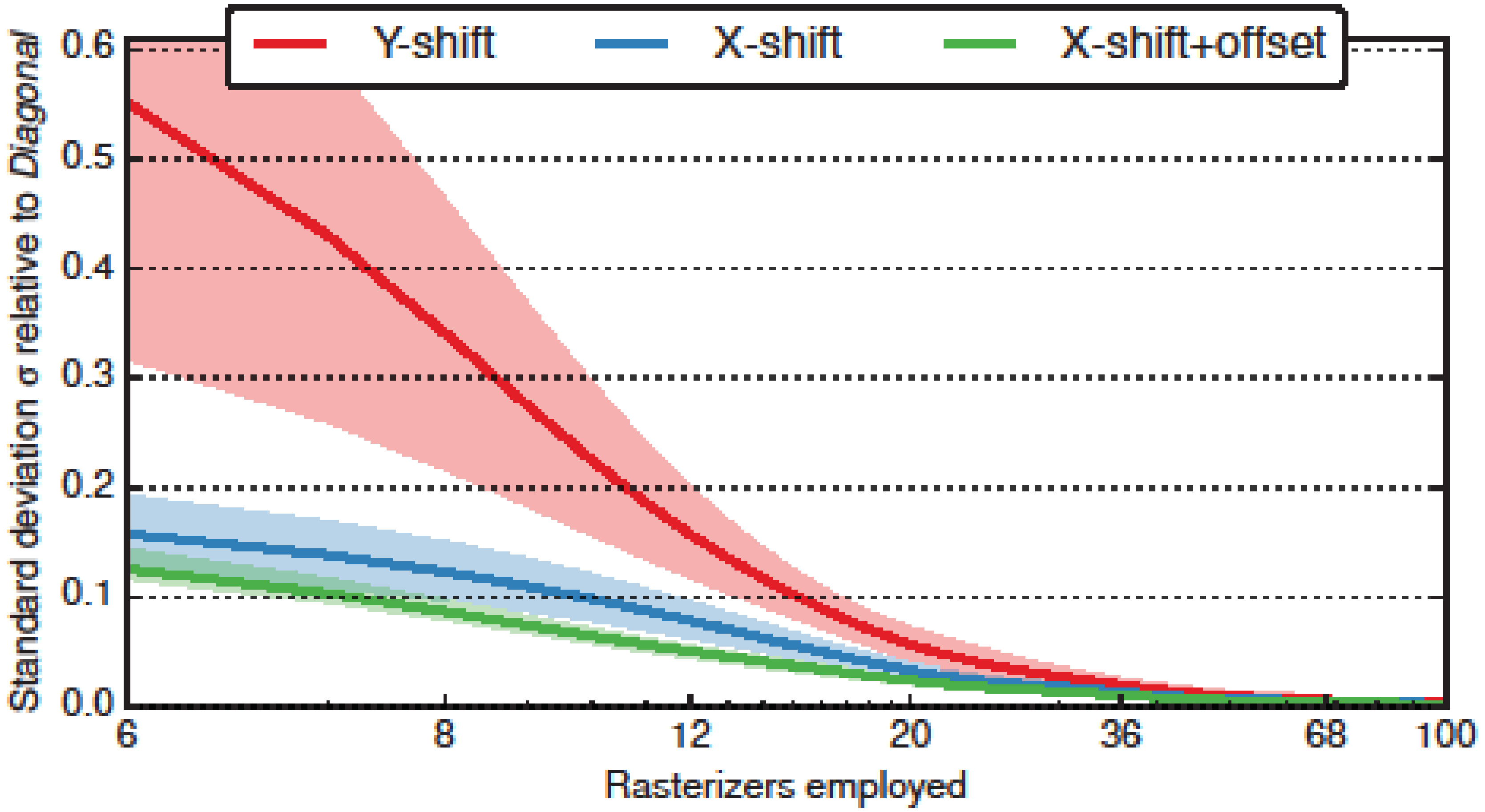


Fixed shift-based patterns

- Y-shift: each column shifted by constant value $\frac{N}{k}$ where $\mathrm{k} = \lfloor\sqrt{N}\rfloor$

- X-shift: similar, but shift rows instead

- X-shift+offset: additional offset
  - Off by 1 in every k^{th} row
  - Pattern only repats after *N* shifts

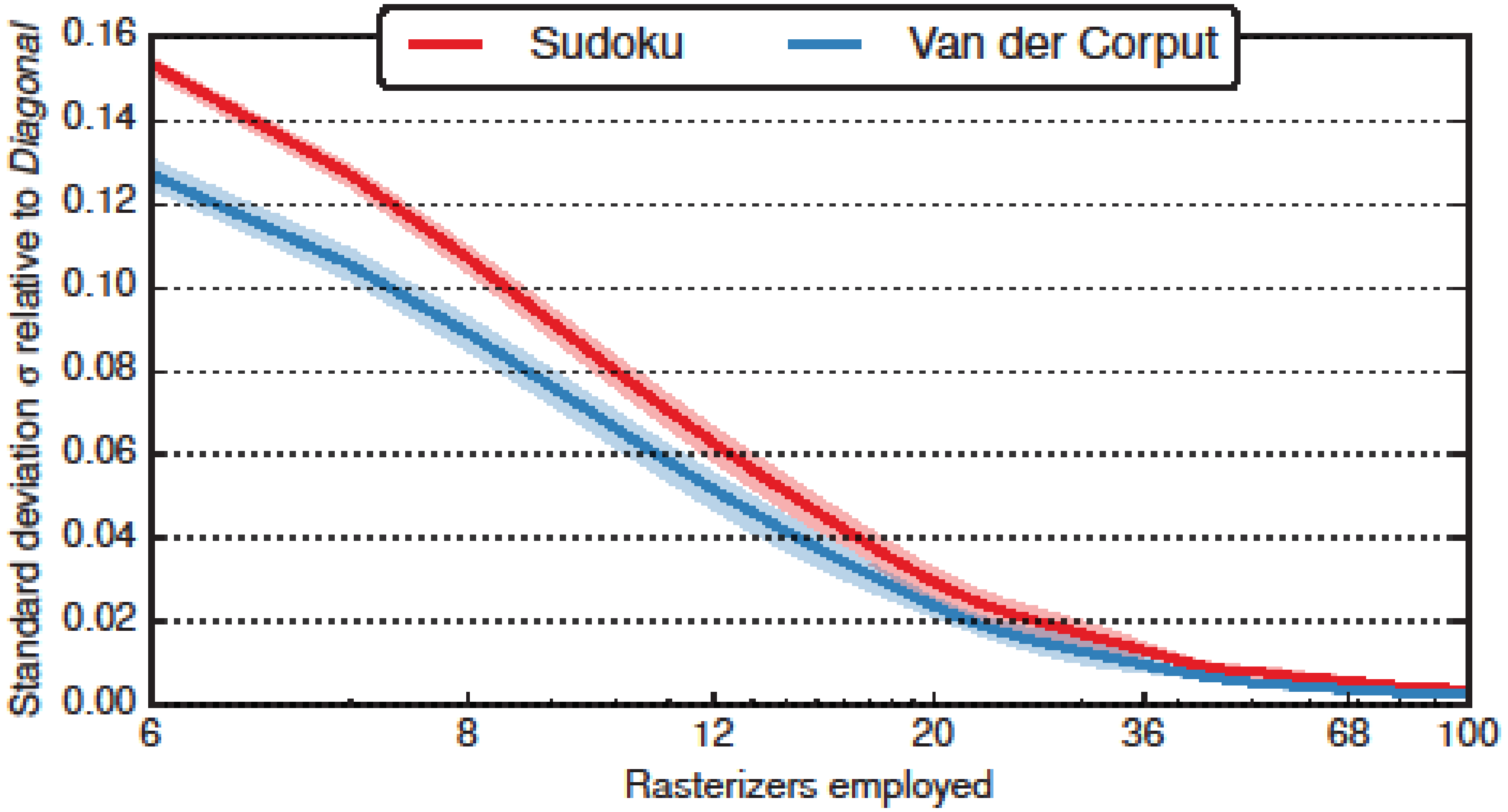


Variable shift-based patterns

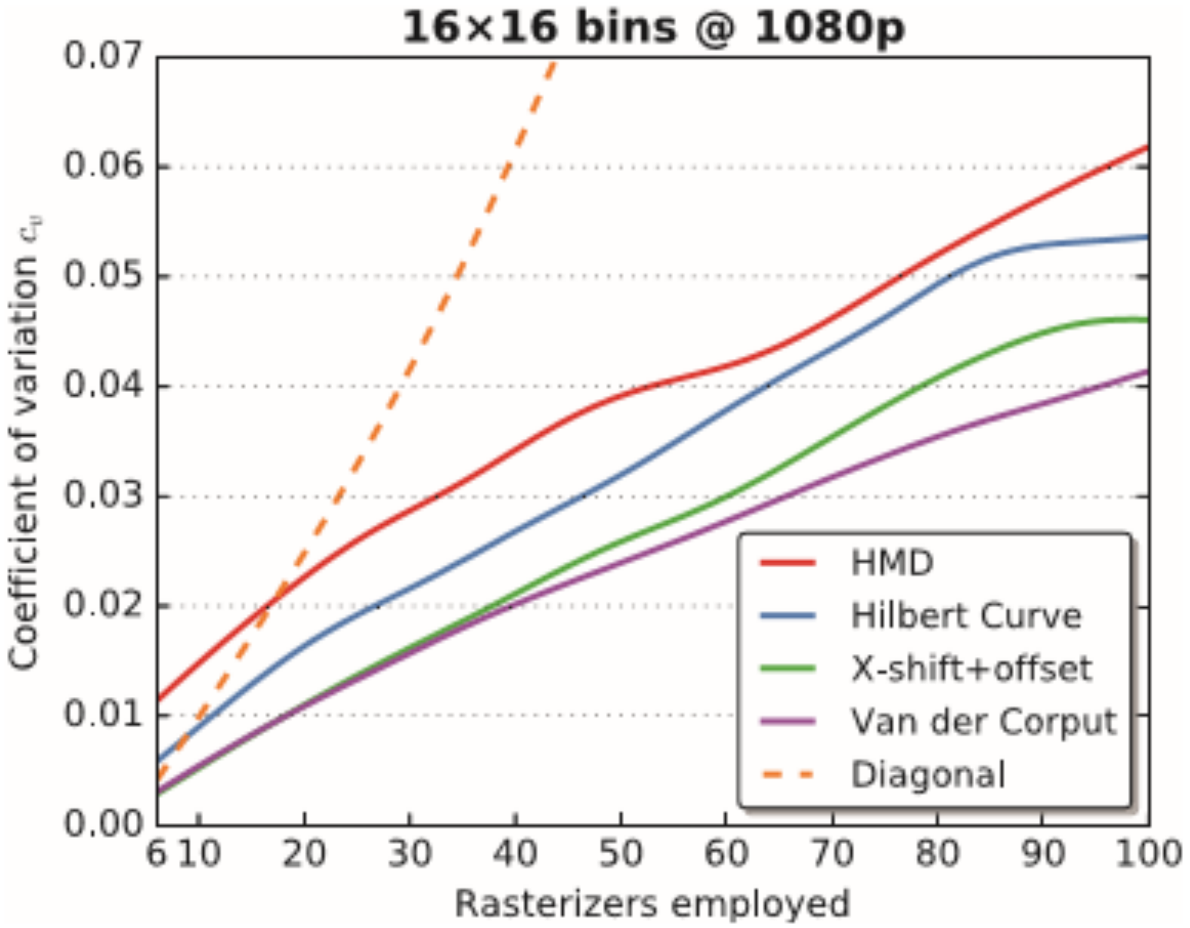
- Sudoku: pick shift value at random each row, disallow duplicates

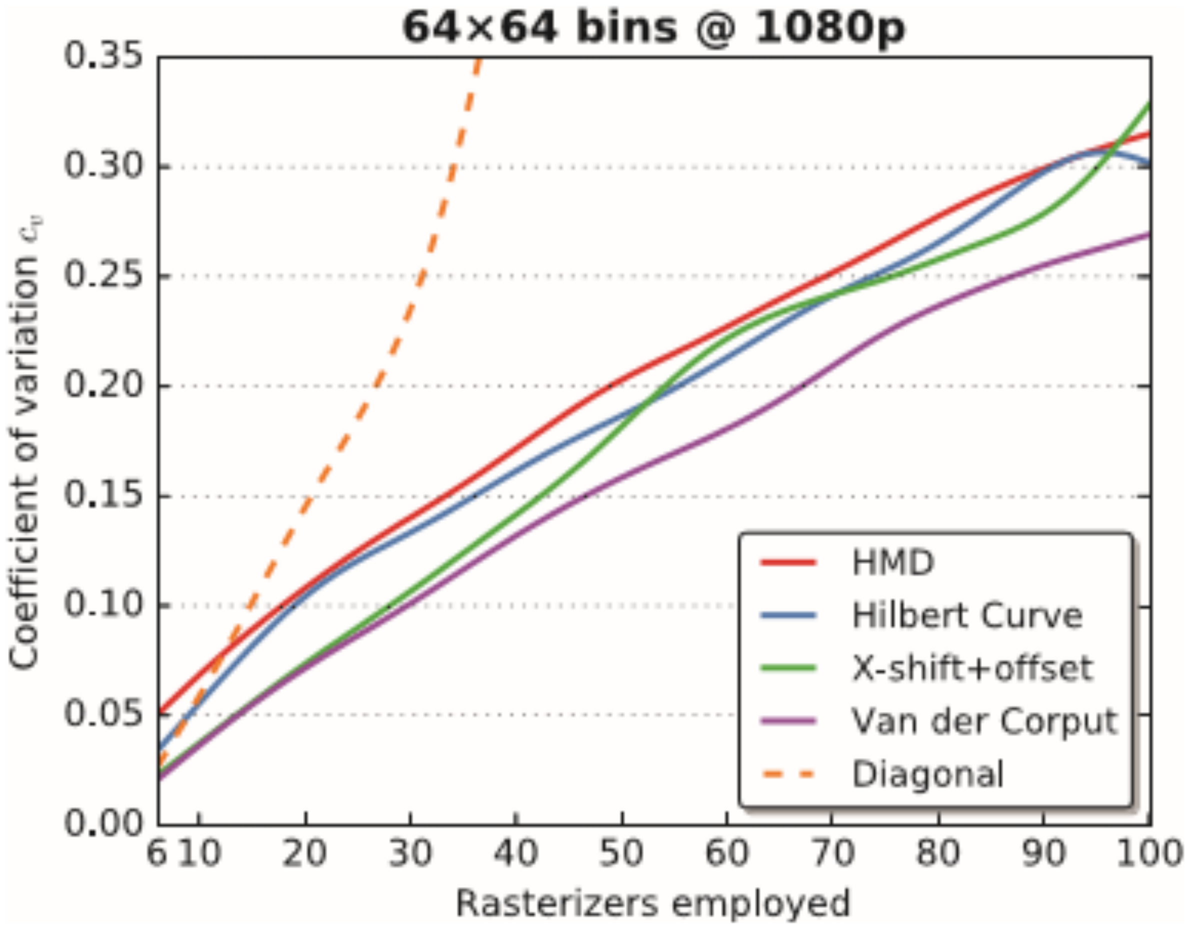
- Van der Corput (VDC): shift using base 2 Van der Corput sequence
  - Multiply by next power of 2 for *N*
  - Skip shifts that exceed *N*
  - Implicitly fulfills Sudoku constraint

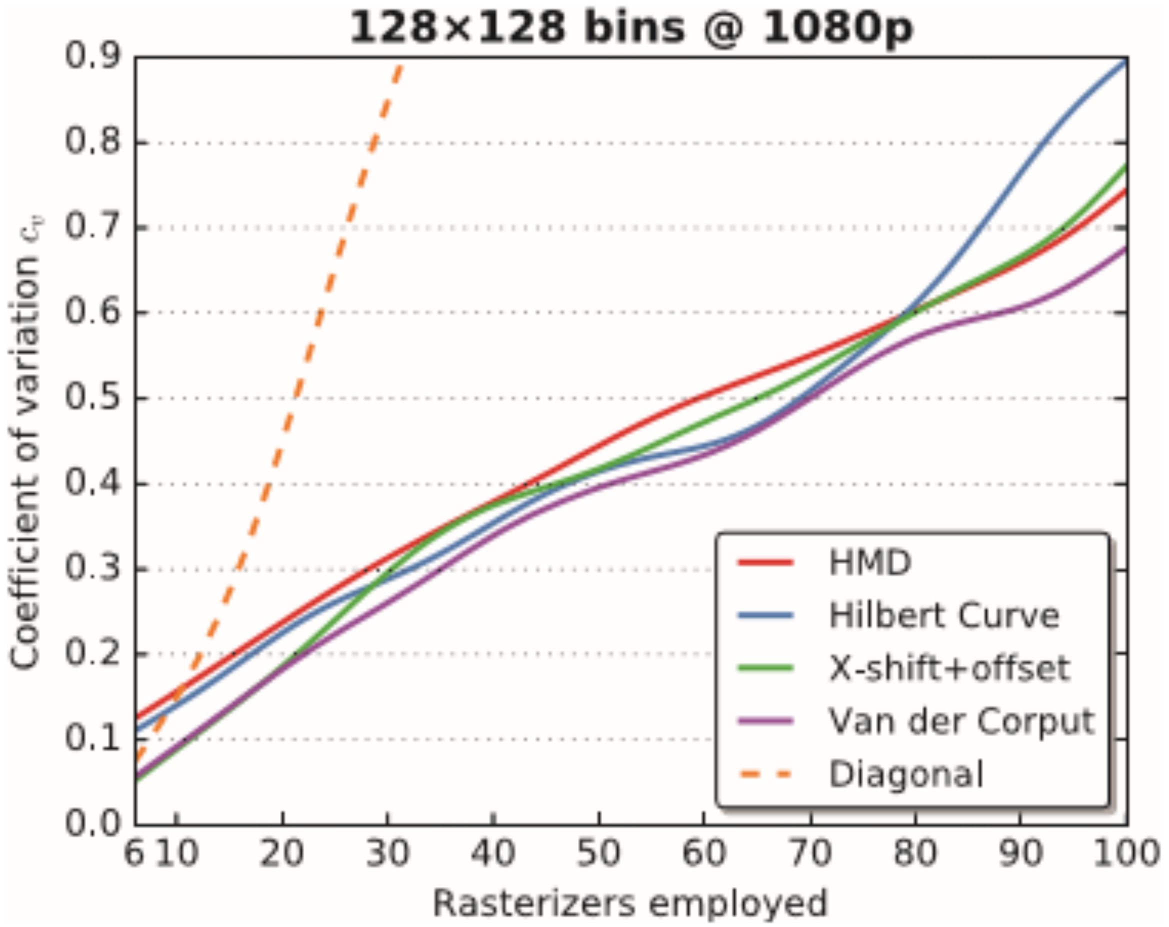
- Similar performance, VDC leads



Overall comparison







Thank you

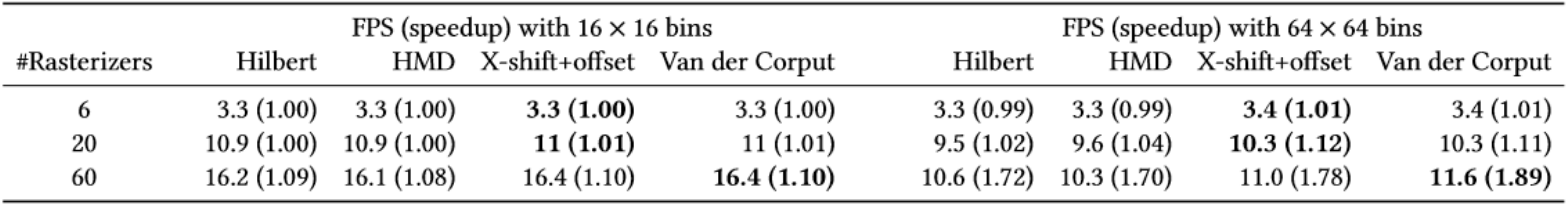
- Questions?

Stone Giant Tech Demo by NVIDIA.

Software Simulated Results

- At higher number / greater bin size, differences show

- Dynamic balancing between vertex and fragment stage cannot compensate fragment load discrepancy



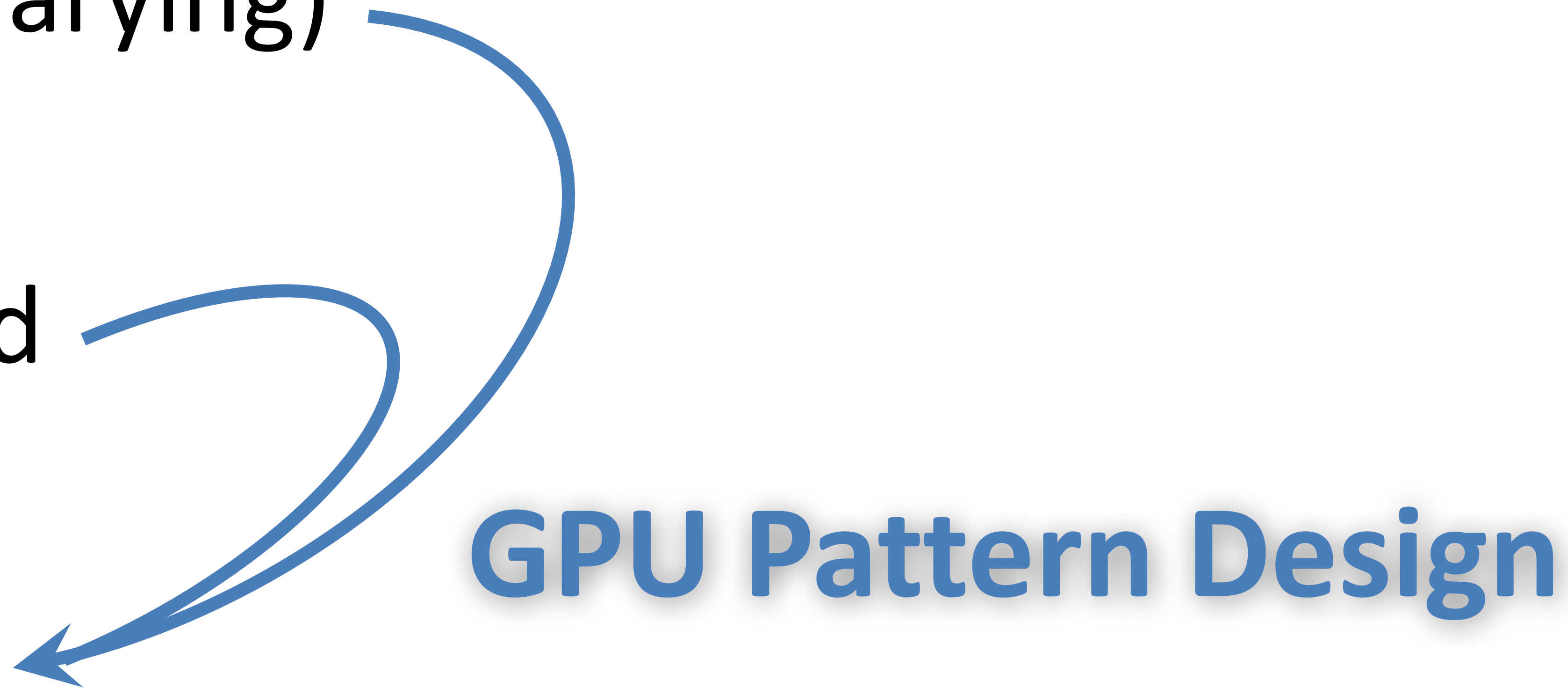
	FPS (speedup) with 16 × 16 bins				FPS (speedup) with 64 × 64 bins			
#Rasterizers	Hilbert	HMD	X-shift+offset	Van der Corput	Hilbert	HMD	X-shift+offset	Van der Corput
6	3.3 (1.00)	3.3 (1.00)	**3.3 (1.00)**	3.3 (1.00)	3.3 (0.99)	3.3 (0.99)	**3.4 (1.01)**	3.4 (1.01)
20	10.9 (1.00)	10.9 (1.00)	**11 (1.01)**	11 (1.01)	9.5 (1.02)	9.6 (1.04)	**10.3 (1.12)**	10.3 (1.11)
60	16.2 (1.09)	16.1 (1.08)	16.4 (1.10)	**16.4 (1.10)**	10.6 (1.72)	10.3 (1.70)	11.0 (1.78)	**11.6 (1.89)**

Tested Pattern Categories

- Space Filling Curves
- Randomization-based patterns
- Fixed shift-based patterns
- Variable shift-based patterns

2D Binning Parameters

- Bin size in pixels, fixed size assumed (8x8, 16x8, 16x16, …)
- Rasterizer capability (uniform/varying)
- Number of rasterizers employed
- Arrangement scheme (pattern)

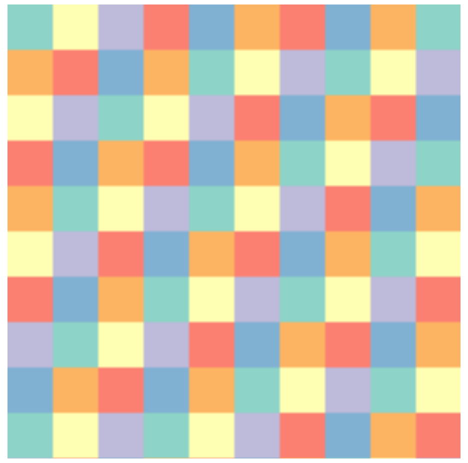


GPU Pattern Design

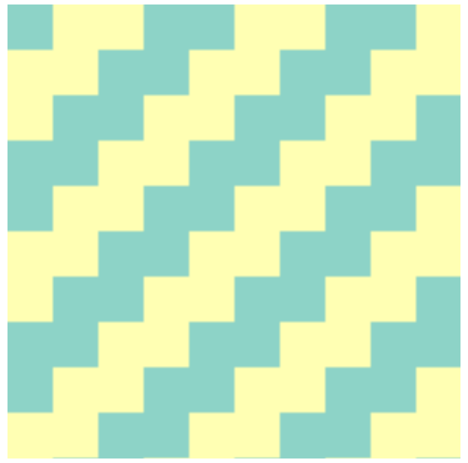
Software Simulated Results

- Full streaming software rendering pipeline running on GPU
- DirectX 9 features (vertex/fragment processing, shading, ...)
- Developed with C++/CUDA
- Evaluate best patterns for our test suite of 200 captured scenes

Binning Pattern Deviants (NVIDIA)

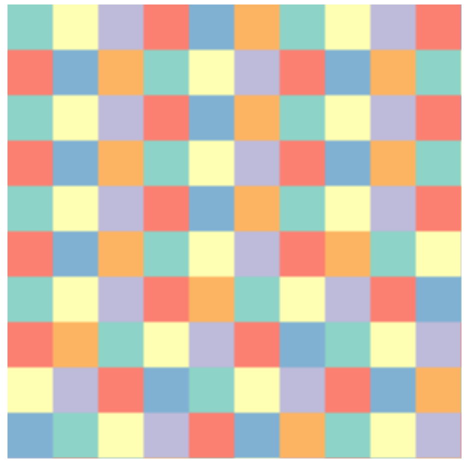


GTX Titan X

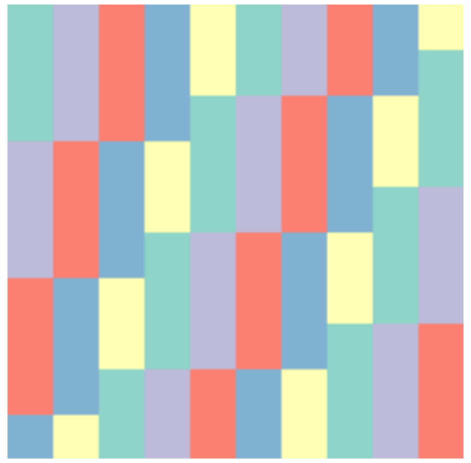


GTX 960

Uniform rasterizer capabilities



GTX 980 Ti

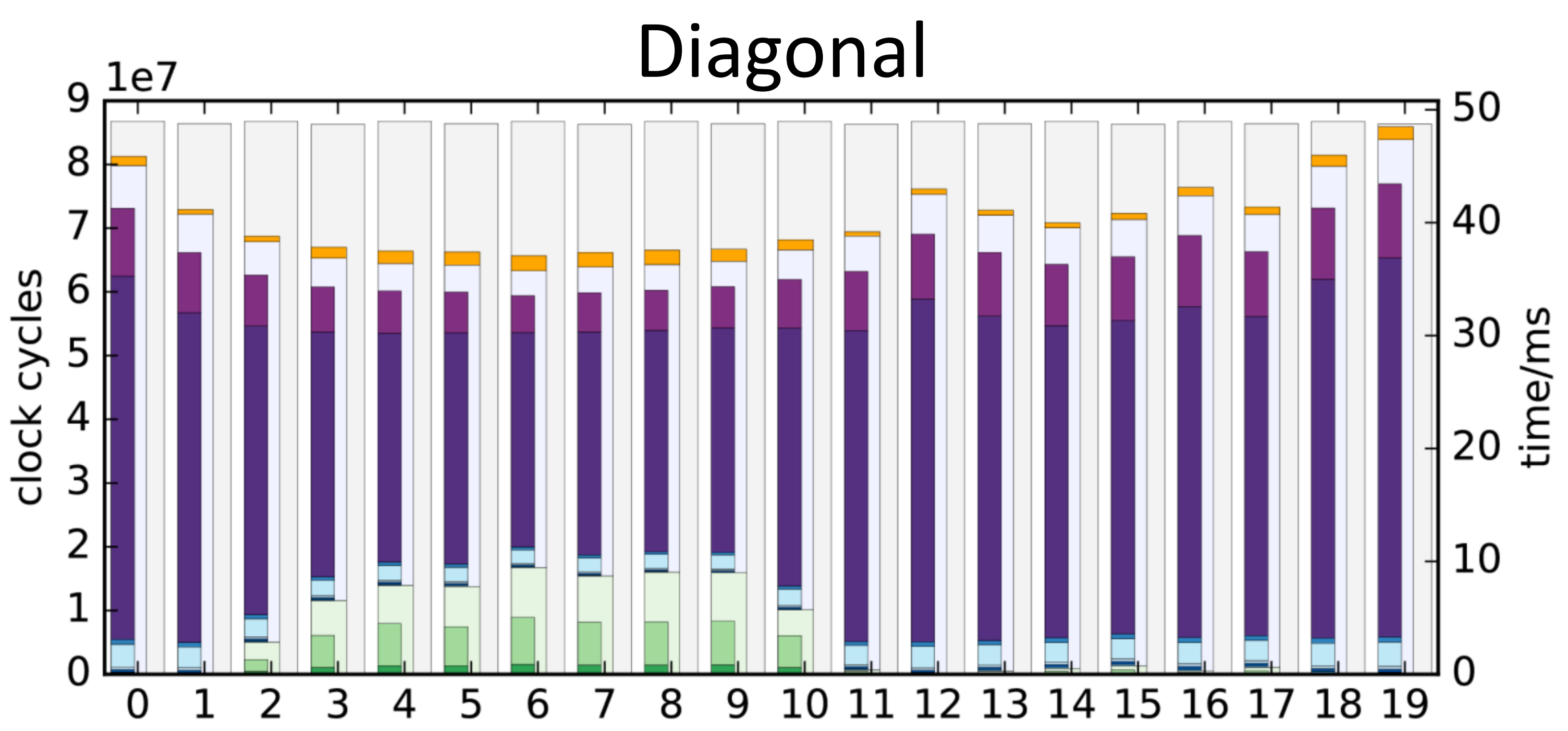


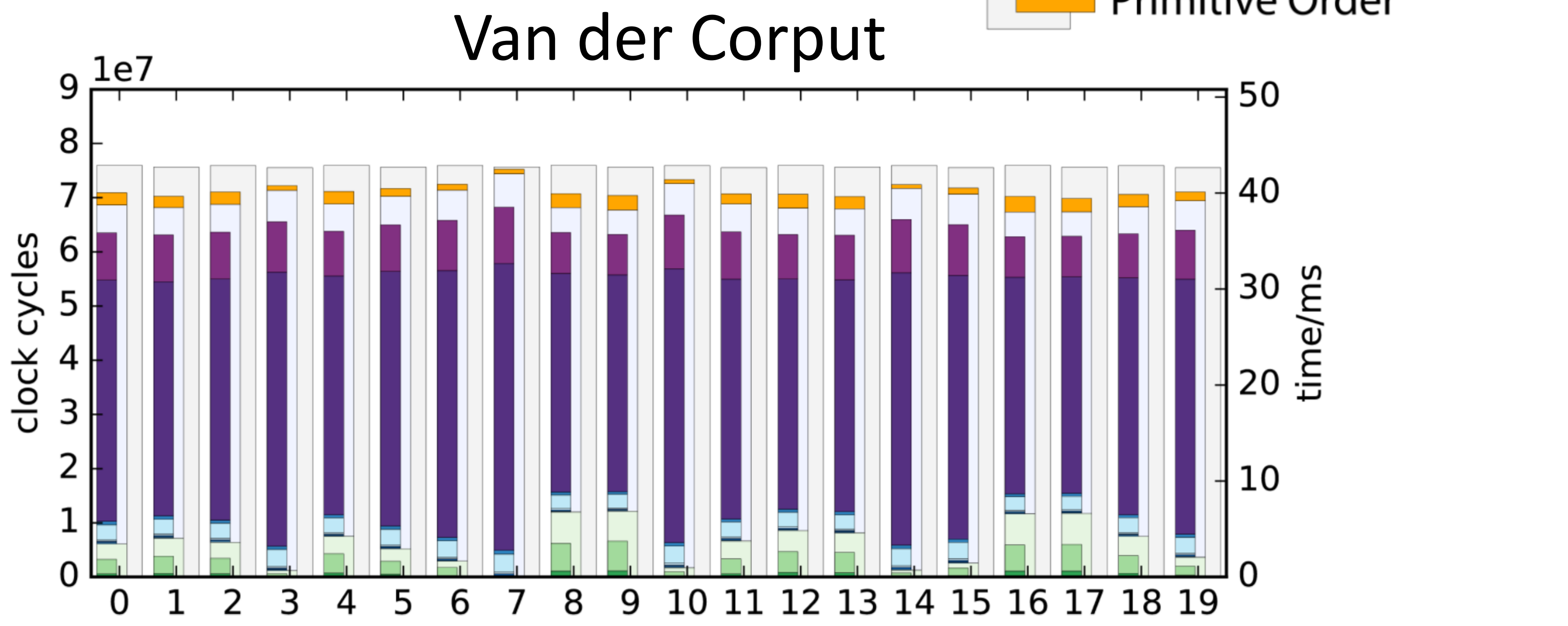
GTX Titan

Variable rasterizer capabilities

Software Simulated Results

- At higher number / greater bin size, differences show
- Dynamic balancing between vertex and fragment stage cannot compensate fragment load discrepancy





You can try this yourself

- General approach – timing based:
  - Write complex fragment shader performing M instructions
  - Draw $2 \times N$ triangles covering one pixel at location p, record time
  - Alternately draw N triangles at p and N triangles at another location p‘
  - If elapsed time decreases significantly, a different „rasterizer“ was hit
  - Experiment with N, M to get clear results

- NVIDIA: Use *shader_thread_group* to find cores pixels submit to

Graz University of Technology

Geometry Orientation

- Subdivide viewport into pixel lines with given orientation (angle)
- Sample along lines and compute total number of fragments
- Compute variance of fragment count over all lines
- Repeat for all desired orientations / angles and compare