

# Revisiting The Vertex Cache

## Understanding and Optimizing Vertex Processing on the modern GPU

**Bernhard Kerbl**

Michael Kenzel

Elena Ivanchenko

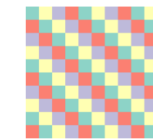
Dieter Schmalstieg

Markus Steinberger

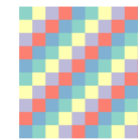
# Last year's talk at HPG'17

## Effective Static Bin Patterns for Sort-Middle Rendering

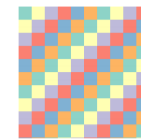
### Common Binning Patterns



GTX 580/680



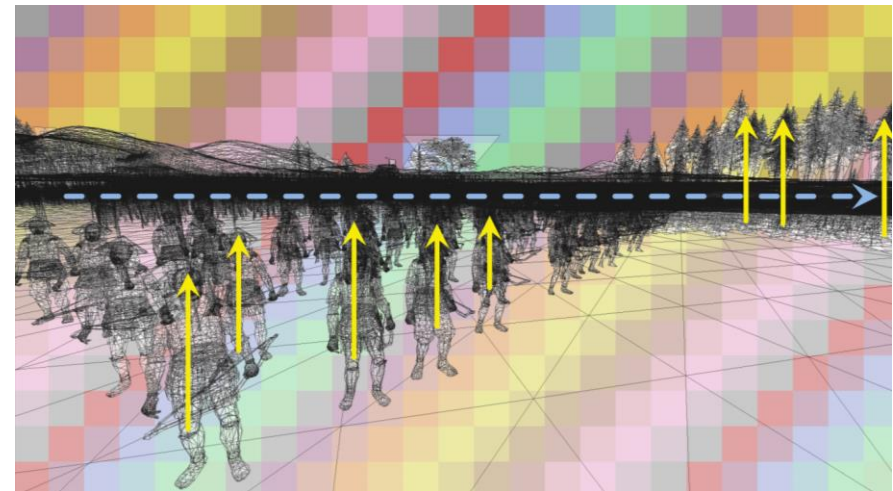
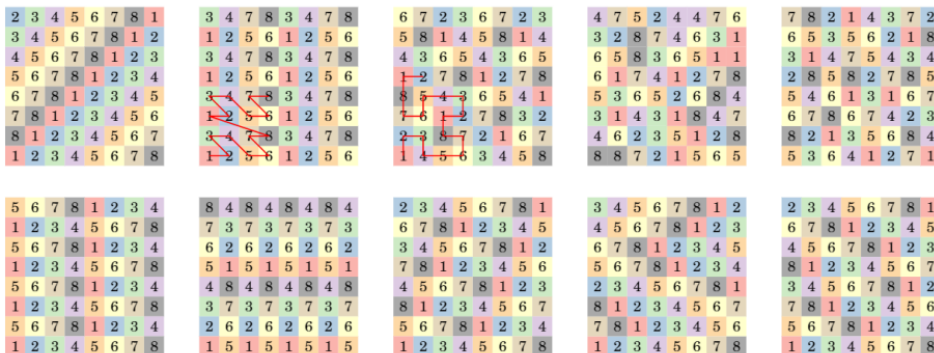
GTX 780 Ti



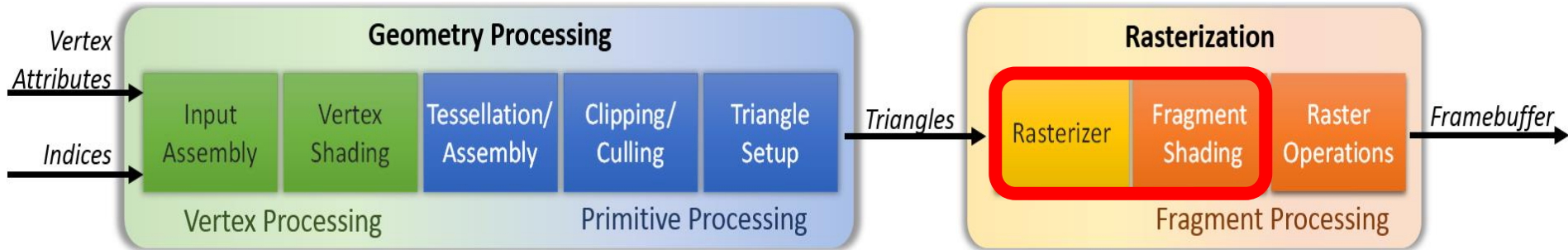
GTX Titan Xp



GTX 1060

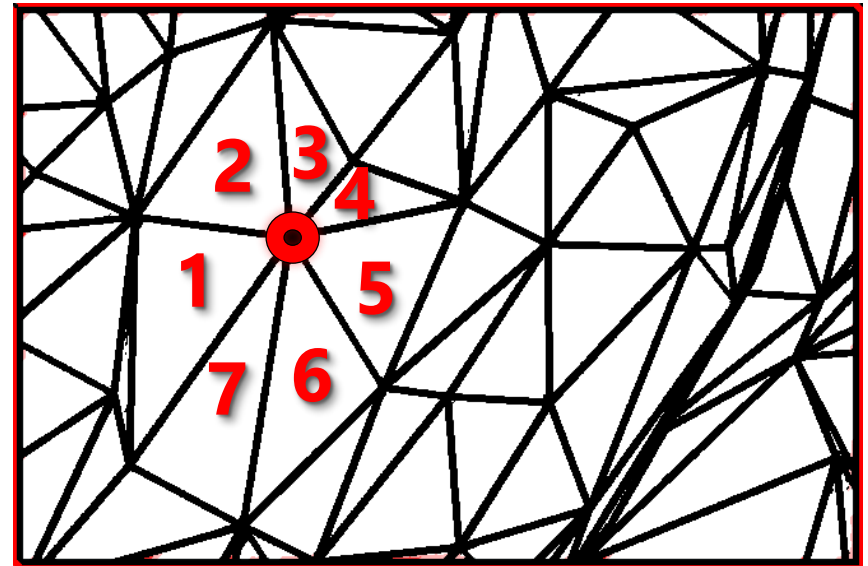
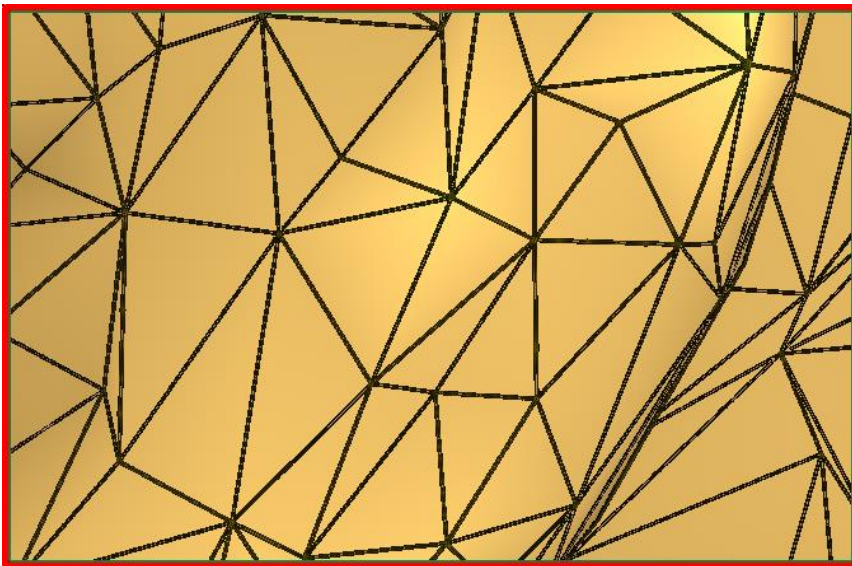


# This year's talk



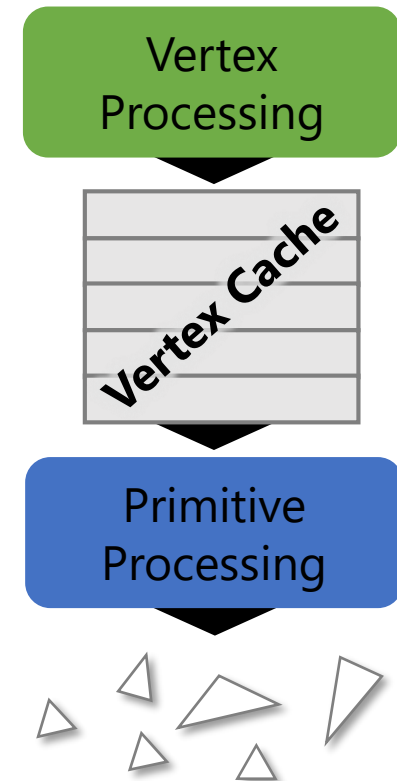
# Reuse in Triangle Meshes

- Exploit high *vertex valence*
- Ex.: Triangle strips, fans...
- Index list can cause  $\ll 1.0$  shaded vertex per triangle



# Post-transform Vertex Cache

- Classic approach [Hoppe 1999]:
  - Caches the last  $N$  **shaded** vertices (hence “post-transform”)
  - FIFO or LRU
- During primitive processing:
  - Vertex needed  $\Rightarrow$  check cache
  - Cache miss  $\Rightarrow$  rerun vertex processing



# Mission Statements

- Assess caching for massively parallel devices
- Identify actual GPU workload distribution scheme
- Optimize vertex input order for the modern GPU

# Aspects of Vertex Reuse

## Vertex Reuse

- Scheduling of vertex processing to
- Exploit locality of vertex references
- This work

## Mesh Optimization

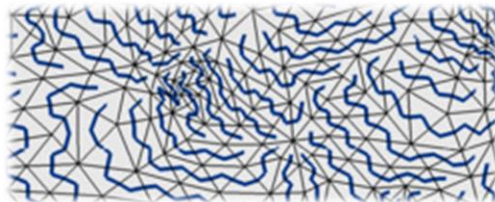
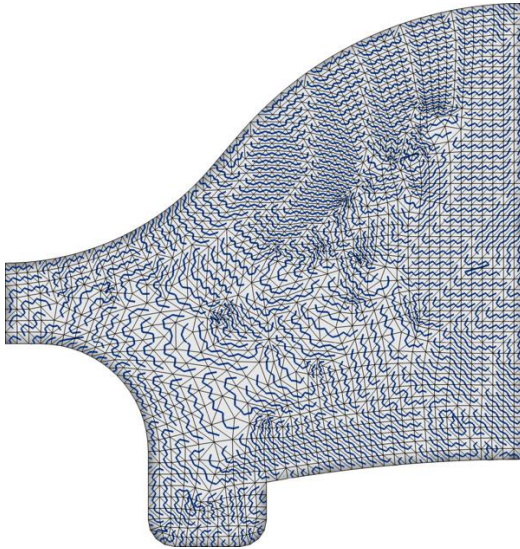
- Reordering of the index stream to
- Maximize locality of vertex references
- Most previous work

# Mesh Optimization Algorithms

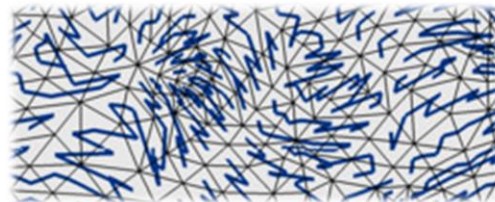
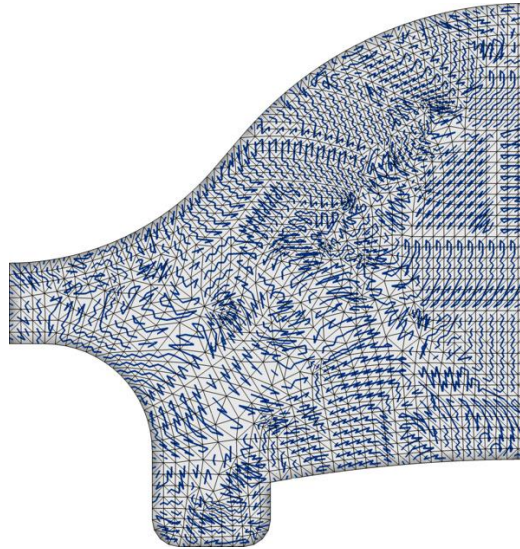
- Exploit existence of cache and reorder vertices to minimize Average Cache Miss Rate (ACMR)
- Greedy algorithms: add new triangles to reordered list based on a score function
- Usually build triangle strips to reduce run time



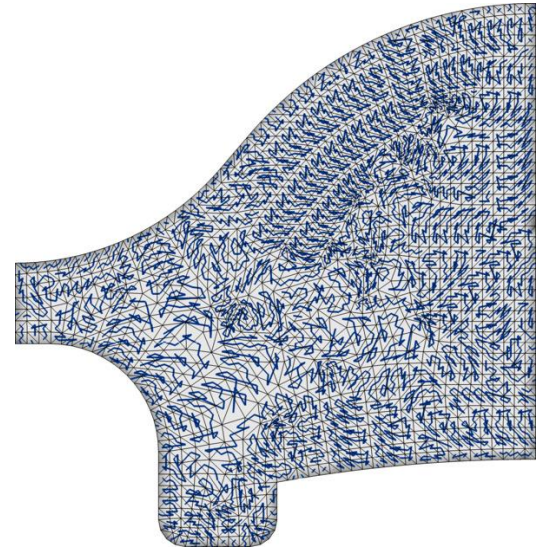
# Cache-based Mesh Optimizers



D3DXMesh  
(Hoppe, 1999)



K-Cache Reorder  
(Lin & Yu, 2006)



AMD Tootle Tipsify  
(Sander et al., 2006)

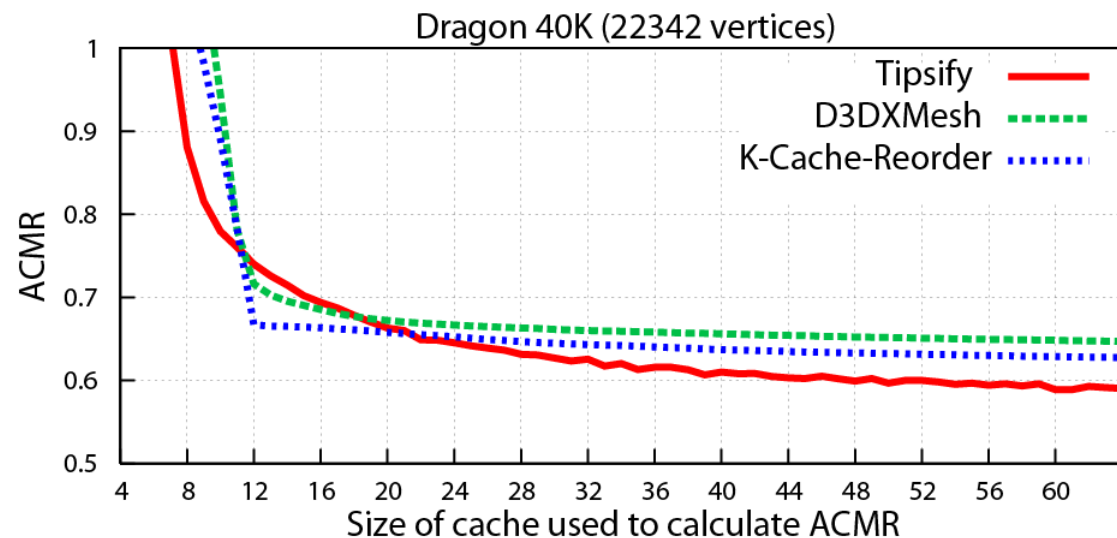
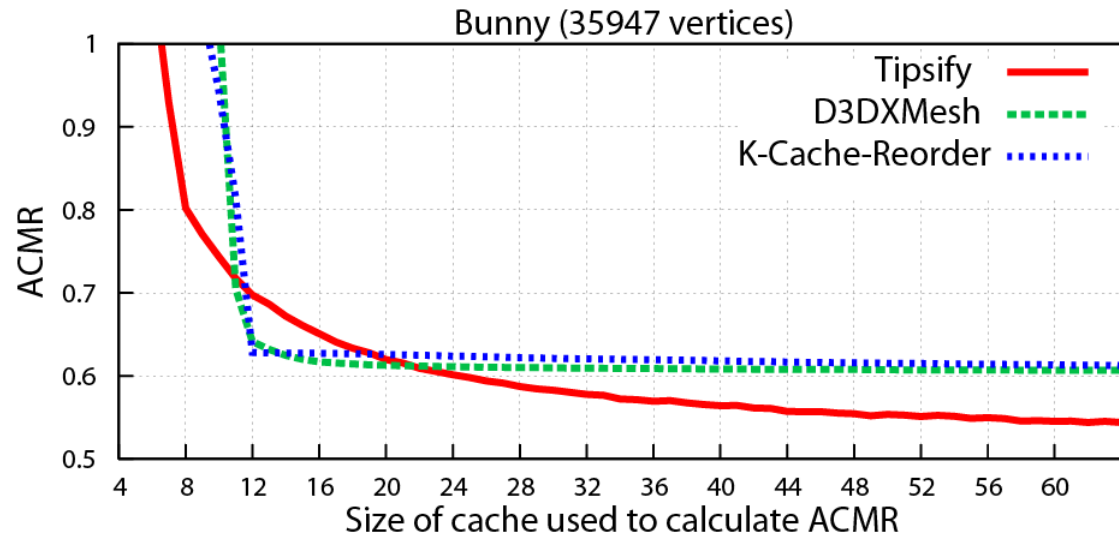
Images used from Pedro V. Sander, Diego Nehab, and Joshua Barczak. *Fast Triangle Reordering for Vertex Locality and Reduced Overdraw*. ACM Transactions on Graphics (Proc. SIGGRAPH) 26(3), August 2007.

# Cache Optimizer Performance

- Ability to reduce overall ACMR
- Parameterized with cache size
- Usually better as cache gets bigger

Images used from Pedro V. Sander, Diego Nehab, and Joshua Barczak. *Fast Triangle Reordering for Vertex Locality and Reduced Overdraw*. ACM Transactions on Graphics (Proc. SIGGRAPH) 26(3), August 2007.

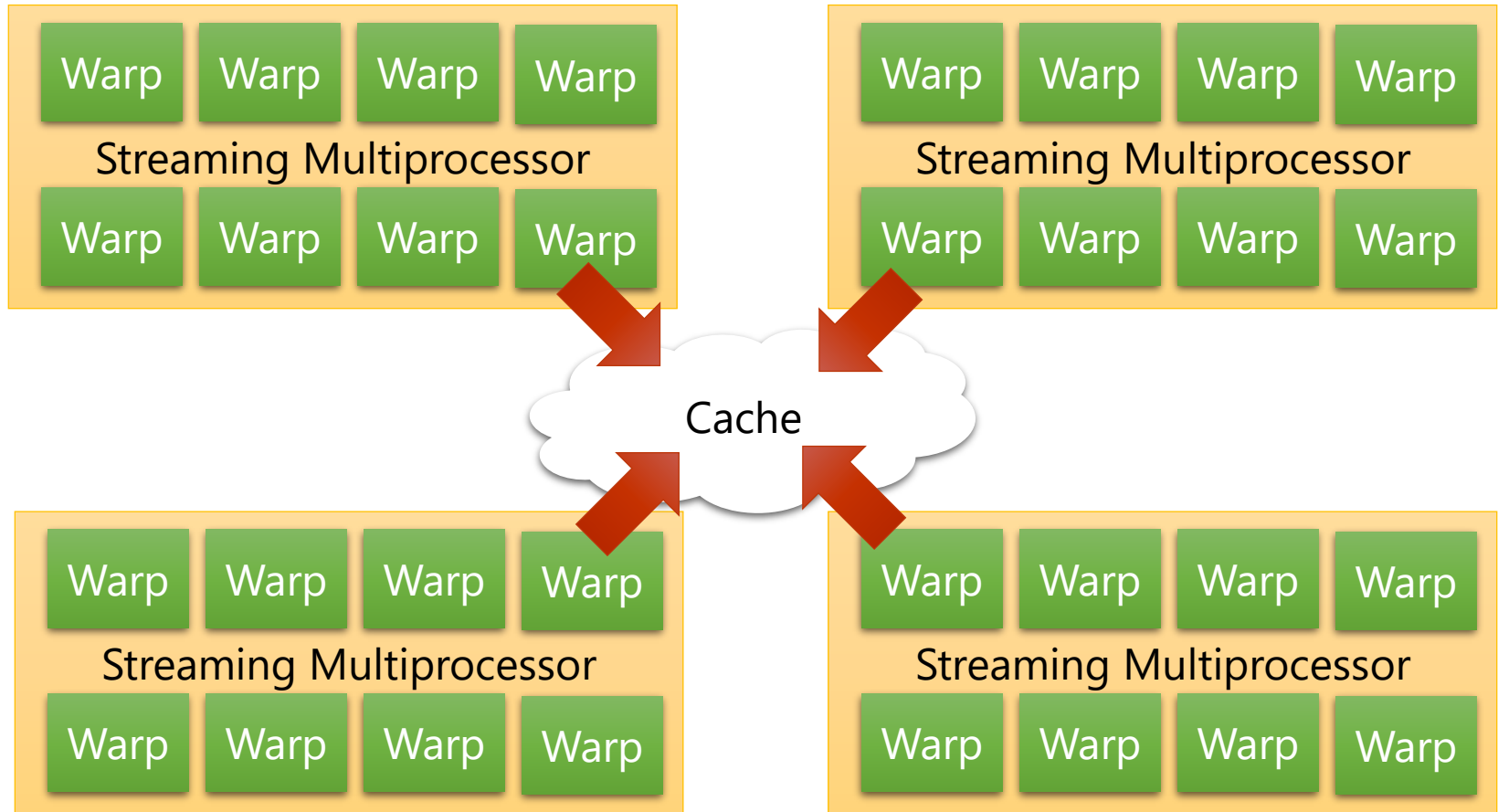
Bernhard Kerbl



# Mission Statements

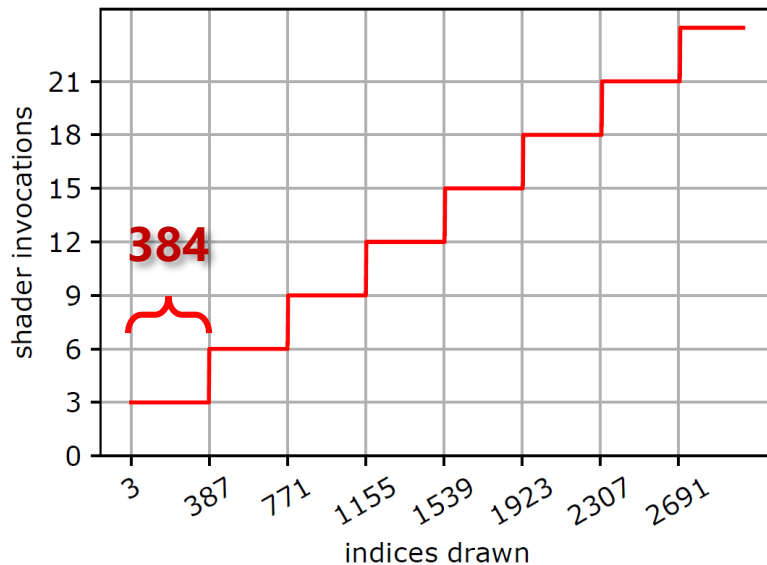
- Assess caching for massively parallel devices
- Identify actual GPU workload distribution scheme
- Optimize vertex input order for the modern GPU

# Cache with Massive Parallelism

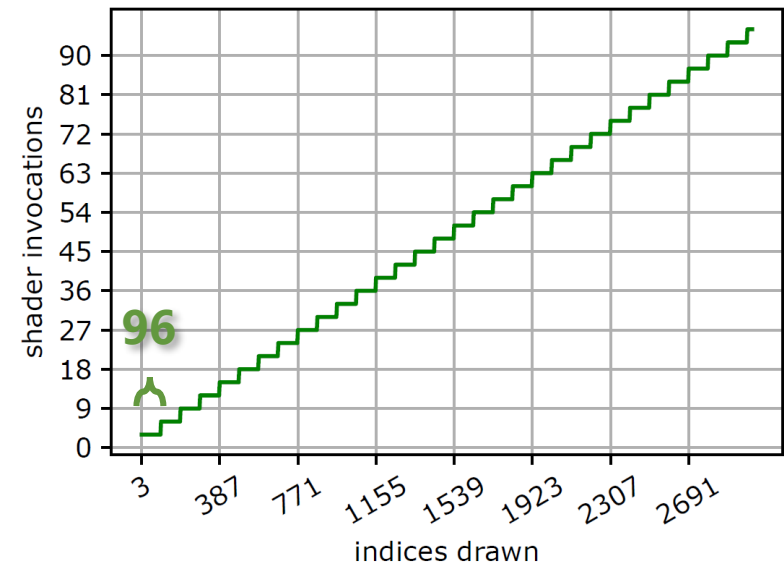


# Counting Vertex Shader Calls

- Use atomic counter for vertex indices (0,1,2|0,1,2|...)
- Atomically increment counter in each shader call



AMD



Nvidia

# In-depth analysis

- Shader Model 6.0 supports wave communication
- Enables us to see the mapping of vertex indices to individual wavefronts for processing
  - On AMD we see large portions of reused range
  - On Nvidia corresponds to full set of reusable vertices

0, 1, 4, 6, 3, 7, 5

6, 7, 8, 2, 5, 3

0, 1, 2, 3, 4, 5

4, 6, 7, 5, 9, 1, 2

# Findings and Interpretation

1. Limited reuse indicates independent batches
2. Contradicts idea of a **central** vertex cache
3. If there are multiple reuse modules (e.g. per SM), they appear to be cleared with every new batch
4. No reuse in **post-transform** manner – submitted load produces optimal parallelism under reuse!

# GPU Batching

- Hardware tries to consolidate idea of vertex reuse and massively parallel independent processing
- Solution: reuse should not be detected after vertex transformation, but **before**
- Analyze input stream and make explicit choices on how to split to enable reuse **and** load balancing



# Mission Statements

- Assess caching for massively parallel devices
- Identify actual GPU workload distribution scheme
- Optimize vertex input order for the modern GPU

# The Batch Predictor

- Analyzes input stream and splits list to produce batches of *primitives* to balance workload
- Can be implemented in hardware or software
- Considers at least 3 limiting factors
  - Number of indices in batch
  - Number of shader calls
  - Retention model for reusing vertices in batch

# Outlining the Batch Predictor



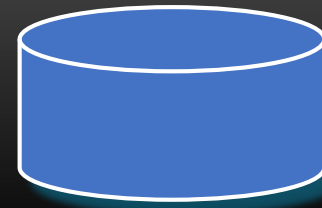
Input: {0, 1, 2|3, 2, 4|2, 5, 1|6, 7, 8|...}



Start at triangle: **0**

End at triangle: **3**

*Retention Model*



0, 1, 2, 3, 2, 4, 2, 5, 1, 6

*Batch Indices*

0, 1, 2, 3, 4, 5, 6

*Shader Calls*



# Nvidia Batches and Reuse

- Max. indices in batch: **96** (or *1 triangle per thread*)
- Max. shader calls: **32** (or *1 shader per thread*)
- No caching required for retention model: simple look back at last  $N$  indices, those are remembered
  - Q: How long can Nvidia remember vertex indices?
  - A: **42** (approximately, depending on order in triangle)

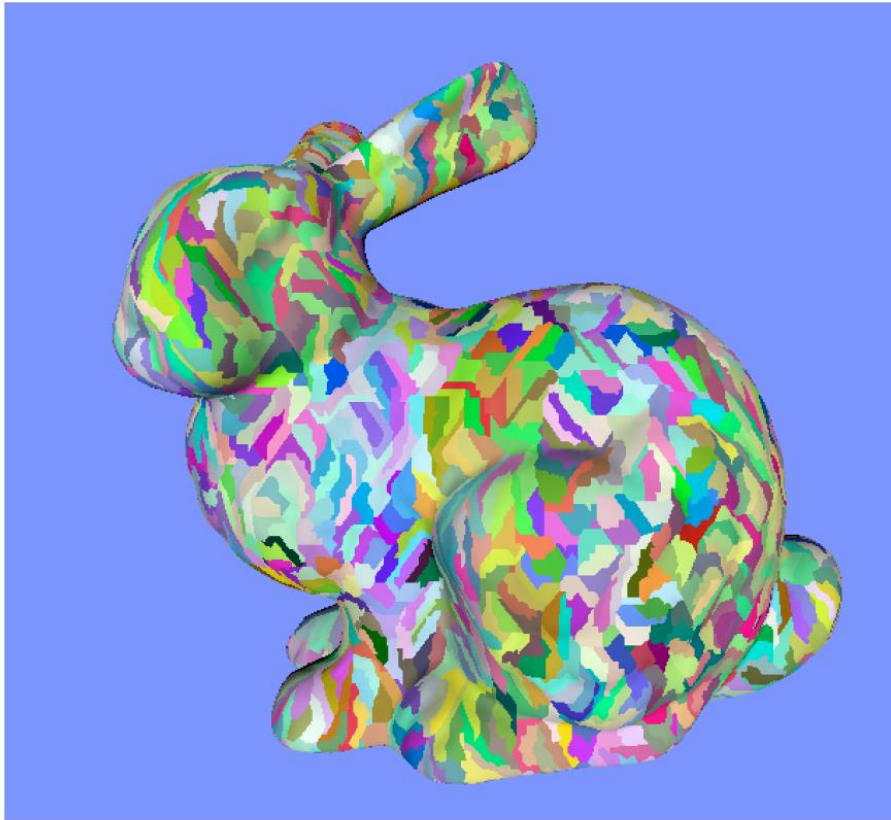
# AMD Batches and Reuse

- Batches have a consistent length of 384 indices
- **But:** Batches don't necessarily correlate with achieved ACMR
- AMD employs second-tier assignment of indices into batches to individual wave fronts
- 15 vertices can be reused in LRU cache

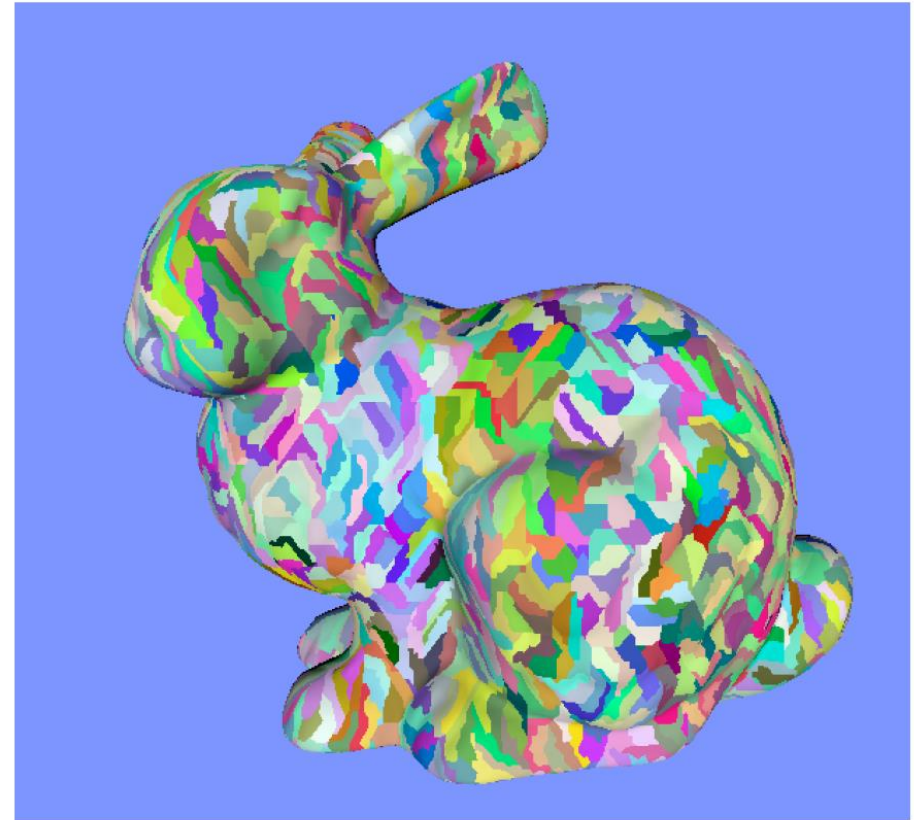
# Prediction Quality and Remarks

- For AMD, incomplete batch function causes error
- On Nvidia, prediction is exact for models with fewer than  $2^{16}$  vertices
- Remaining error originates from larger test scenes
- For now inexplicable ACMR artifacts when indices  $i, j$  with  $\left\lfloor \frac{i}{2^{16}} \right\rfloor \neq \left\lfloor \frac{j}{2^{16}} \right\rfloor$  appear in the same batch

# Predicting Batch Composition



Nvidia predicted



Nvidia measured

# Mission Statements

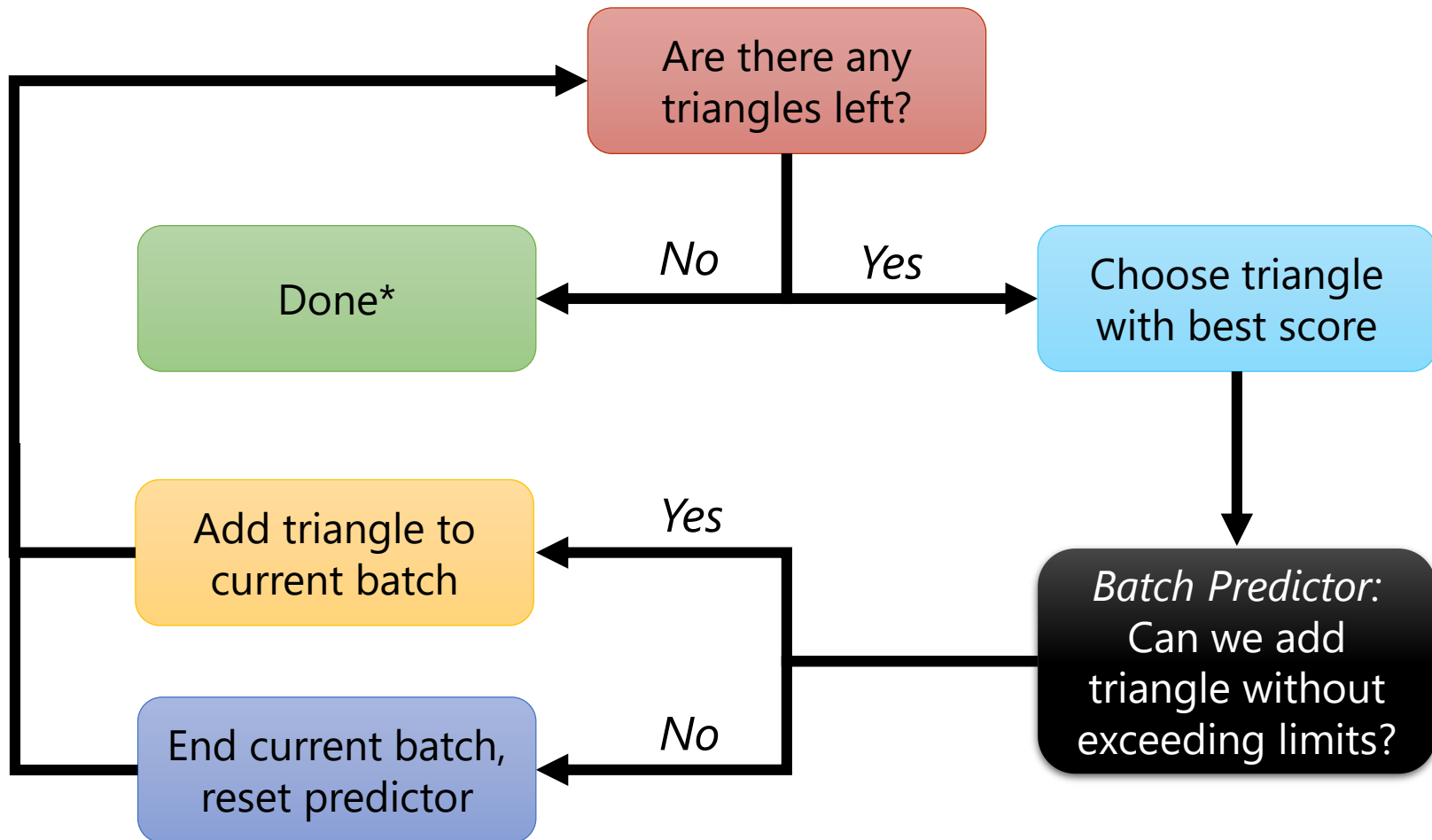
- Assess caching for massively parallel devices
- Identify actual GPU workload distribution scheme
- Optimize vertex input order for the modern GPU



# Mesh Optimization Algorithm

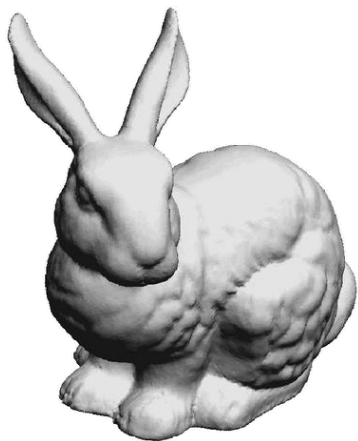
- Greedy algorithm inserts new triangles into batch based on a score function
- Score for each triangle is defined by four factors:
  - **Vertex Reuse** : #vertices already loaded and available
  - **Vertex Valence** : #unused triangles that share its vertices
  - **Face Distance** : average distance to other batch faces
  - **Neighborhood** : prefer neighbors of existing batches

# Algorithm Overview



# Evaluating our Approach

- Used on established models as well as triangle sets from recent video games
- Compared achieved ACMR to alternatives



Bunny

Bernhard Kerbl



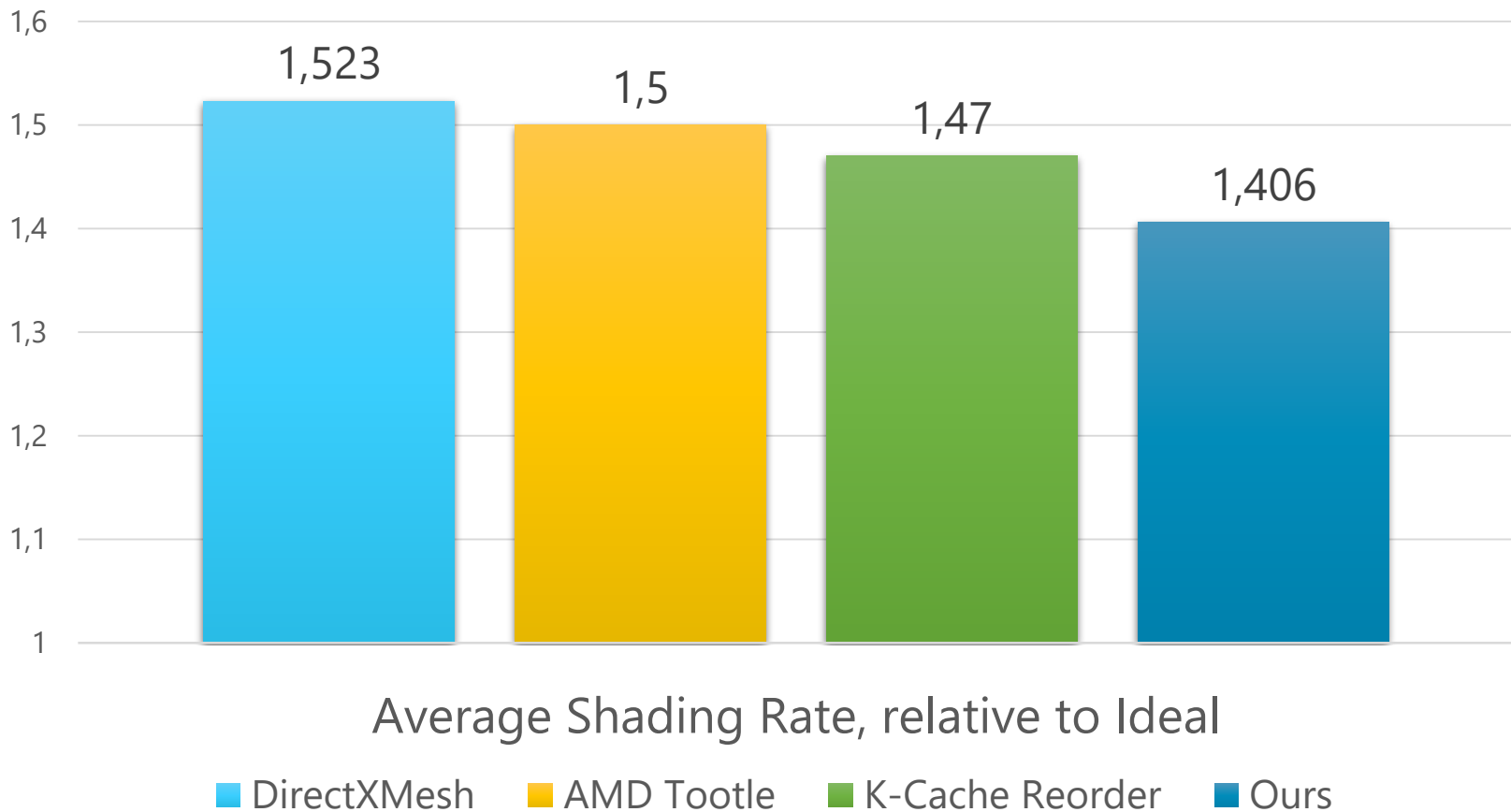
The Witcher 3  
(tw)

Revisiting the Vertex Cache



Happy Buddha

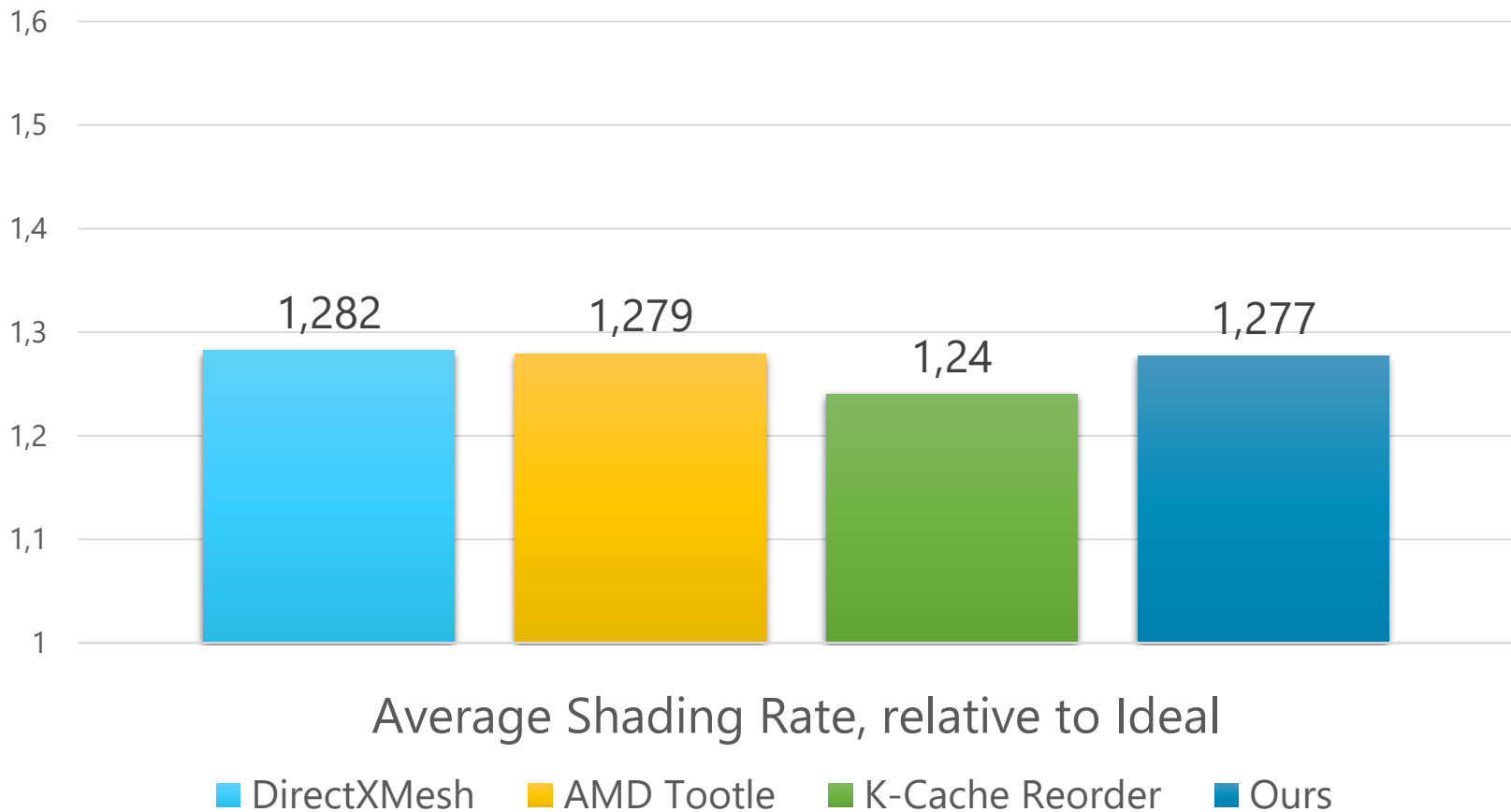
# Optimizers Performance Nvidia





	<i>DirectXMesh</i>	<i>AMD Tootle</i>	<i>K-Cache</i>	<i>Ours</i>	<i>Ideal</i>
Sphere	0.83	0.83	0.82	<b>0.81</b>	0.50
Bunny	0.84	0.86	0.84	<b>0.82</b>	0.50
Happy Buddha	0.98	0.98	0.95	<b>0.81</b>	0.50
XYZRGB Dragon	1.07	1.08	1.10	<b>0.82</b>	0.50
Tree	2.07	2.09	2.09	<b>2.06</b>	2.06
AoM 1	0.97	0.88	0.86	<b>0.84</b>	0.60
AoM 2	0.95	0.81	0.81	<b>0.78</b>	0.48
Black Flag 1	0.87	0.88	0.85	<b>0.83</b>	0.59
Black Flag 2	1.27	1.28	1.26	<b>1.24</b>	1.11
Deus Ex 1	0.88	0.90	<b>0.85</b>	0.89	0.61
Deus Ex 2	0.87	0.88	0.84	<b>0.84</b>	0.62
Stone Giant 1	0.87	0.88	0.83	<b>0.83</b>	0.53
Stone Giant 2	0.89	0.89	0.85	<b>0.84</b>	0.56
Shogun 1	1.01	1.00	0.97	<b>0.92</b>	0.74
Shogun 2	0.98	0.98	0.95	<b>0.94</b>	0.74
Tomb Raider 1	0.95	0.93	0.89	<b>0.87</b>	0.68
Tomb Raider 2	0.93	0.92	0.89	<b>0.88</b>	0.66
The Witcher 1	0.87	0.89	0.87	<b>0.84</b>	0.55
The Witcher 2	1.43	1.41	1.39	<b>1.37</b>	1.23

# Optimizers Performance AMD



Test Scene	<i>DirectXMesh</i>	<i>AMD Tootle</i>	<i>K-Cache</i>	<i>Ours</i>	<i>Ideal</i>
Sphere	<b>0.66</b>	0.68	0.67	0.72	0.50
Bunny	<b>0.68</b>	0.72	0.70	0.72	0.50
Happy Buddha	0.73	0.75	<b>0.71</b>	0.75	0.50
XYZRGB Dragon	<b>0.67</b>	0.71	0.69	0.72	0.50
Tree	2.06	2.07	2.07	<b>2.06</b>	2.06
AoM 1	0.85	0.77	<b>0.74</b>	0.77	0.60
AoM 2	0.81	0.69	0.68	<b>0.68</b>	0.48
Black Flag 1	0.74	0.74	<b>0.73</b>	0.75	0.59
Black Flag 2	1.19	1.20	<b>1.19</b>	1.20	1.11
Deus Ex 1	0.77	0.79	<b>0.75</b>	0.82	0.61
Deus Ex 2	0.75	0.76	<b>0.73</b>	0.74	0.62
Stone Giant 1	0.73	0.75	<b>0.71</b>	0.74	0.53
Stone Giant 2	0.77	0.77	<b>0.73</b>	0.76	0.56
Shogun 1	0.88	0.86	<b>0.84</b>	0.84	0.74
Shogun 2	0.87	0.88	<b>0.85</b>	0.86	0.74
Tomb Raider 1	0.83	0.81	<b>0.78</b>	0.80	0.68
Tomb Raider 2	0.81	0.81	<b>0.78</b>	0.79	0.66
The Witcher 1	<b>0.72</b>	0.75	0.73	0.75	0.55
The Witcher 2	1.35	1.33	<b>1.31</b>	1.32	1.23

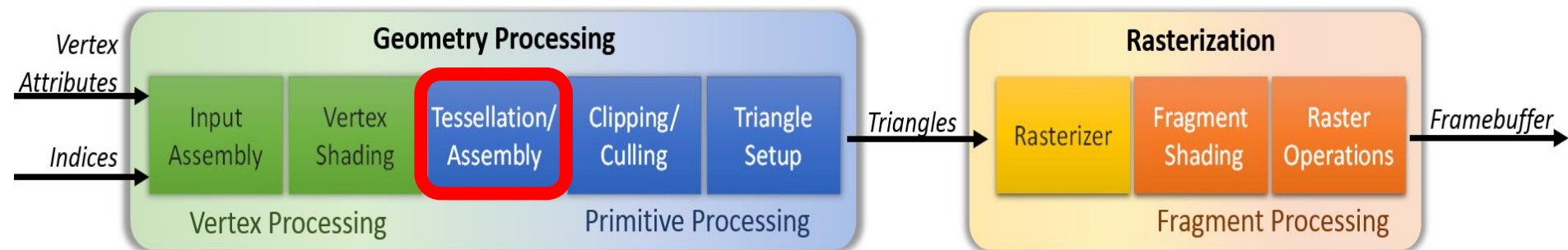
# AMD Results Interpretation

- More modest results for batching on AMD cards
- Multiple reasons
  - Overall simpler algorithm
  - ASR is much lower in general
  - Larger batch → closer to central retention, less benefit
  - Batching function incomplete
  - Second-tier assignment not yet fully understood



# Future Directions

- Fully decipher AMD, Intel batching function
- Tie entire solution into an easy framework
- Next stop: Tessellation?



# Thank you!

- Questions?