Lecture 18:

Parallelizing and Optimizing Rasterization on Modern (Mobile) GPUs

Interactive Computer Graphics
Stanford CS248, Winter 2019

Q. What is a big concern in mobile computing?

A. Power

Two reasons to save power

Run at *higher performance*
for a *fixed* amount of time.

Power = heat
If a chip gets too hot, it must be clocked down to cool off

Run at *sufficient performance*
for a *longer* amount of time.

Power = battery
Long battery life is a desirable feature in mobile devices

Mobile phone examples

Samsung Galaxy s9

11.5 Watt hours

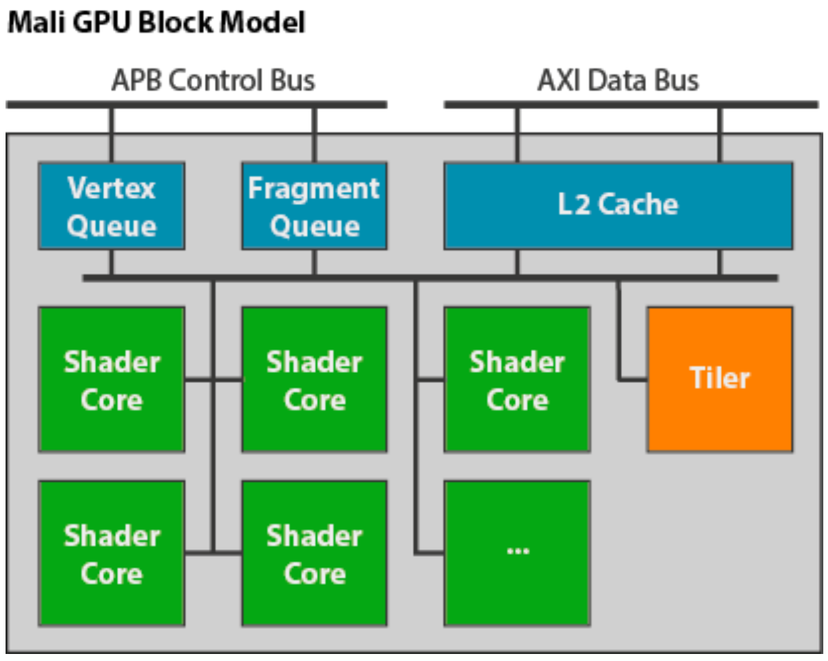
Apple iPhone 8

7 Watt hours

Graphics processors (GPUs) in these mobile phones

Samsung Galaxy s9
(non US version)

ARM Mali G72MP18



Apple iPhone 8

Custom Apple GPU in A11 Processor

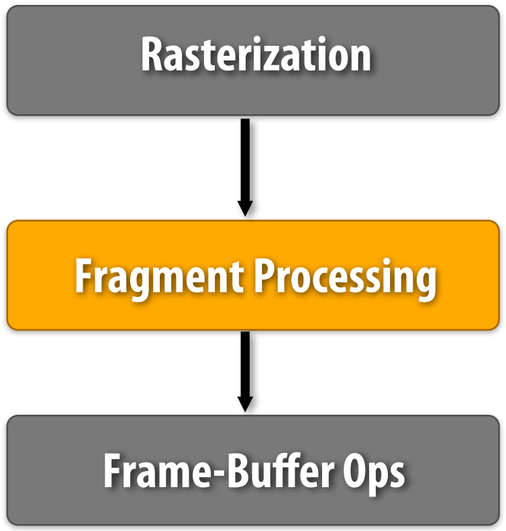
Ways to conserve power

- Compute less
  - Reduce the amount of work required to render a picture
  - Less computation = less power

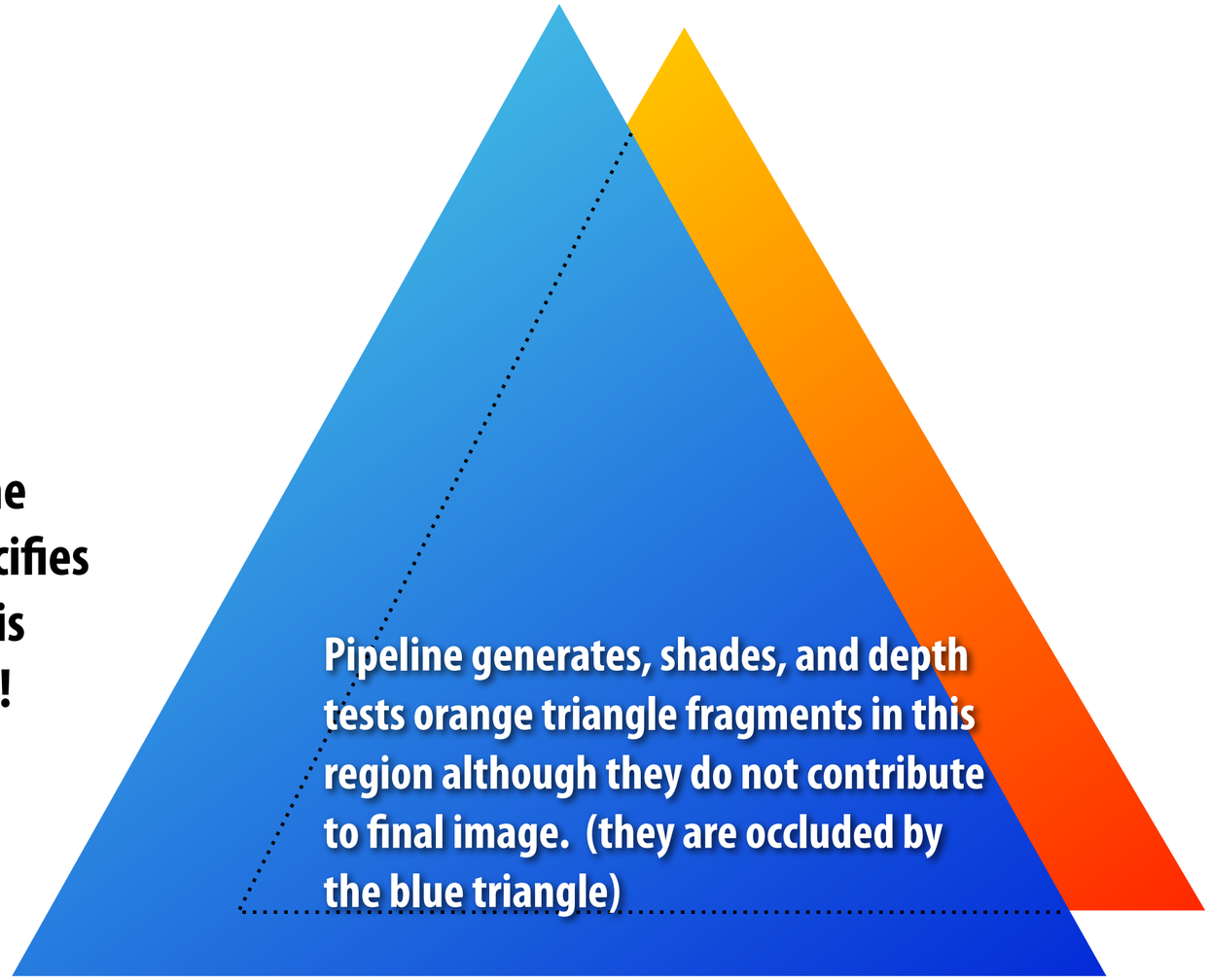
- Read less data
  - Data movement has high energy cost

Early depth culling ("Early Z")

Depth testing as we've described it

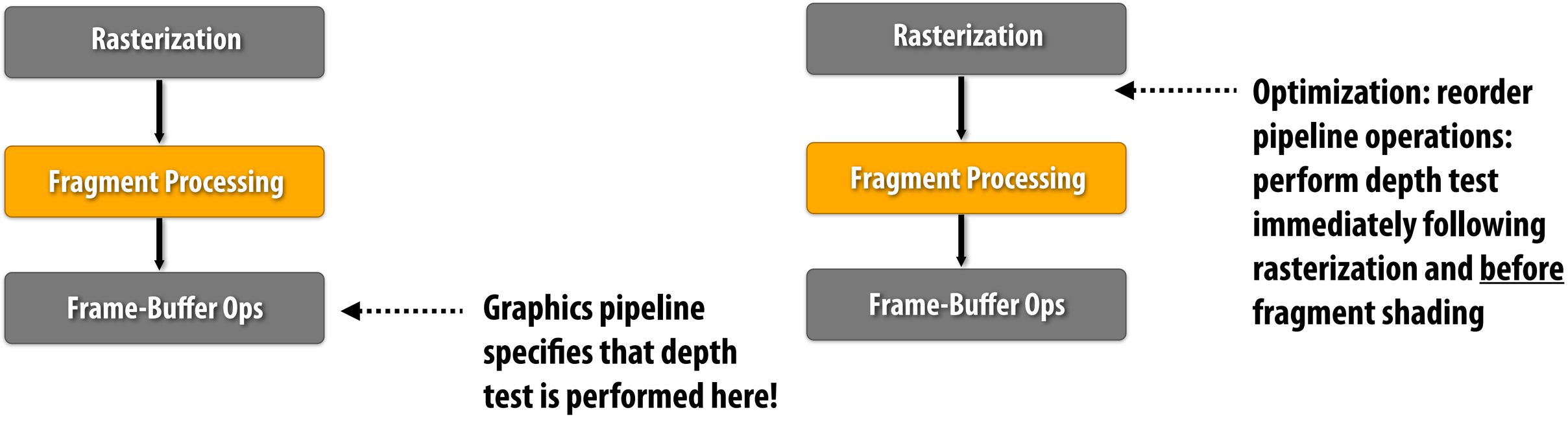


Graphics pipeline abstraction specifies that depth test is performed here!



Early Z culling

- Implemented by all modern GPUs, not just mobile GPUs
- Application needs to sort geometry to make early Z most effective. *Why?*



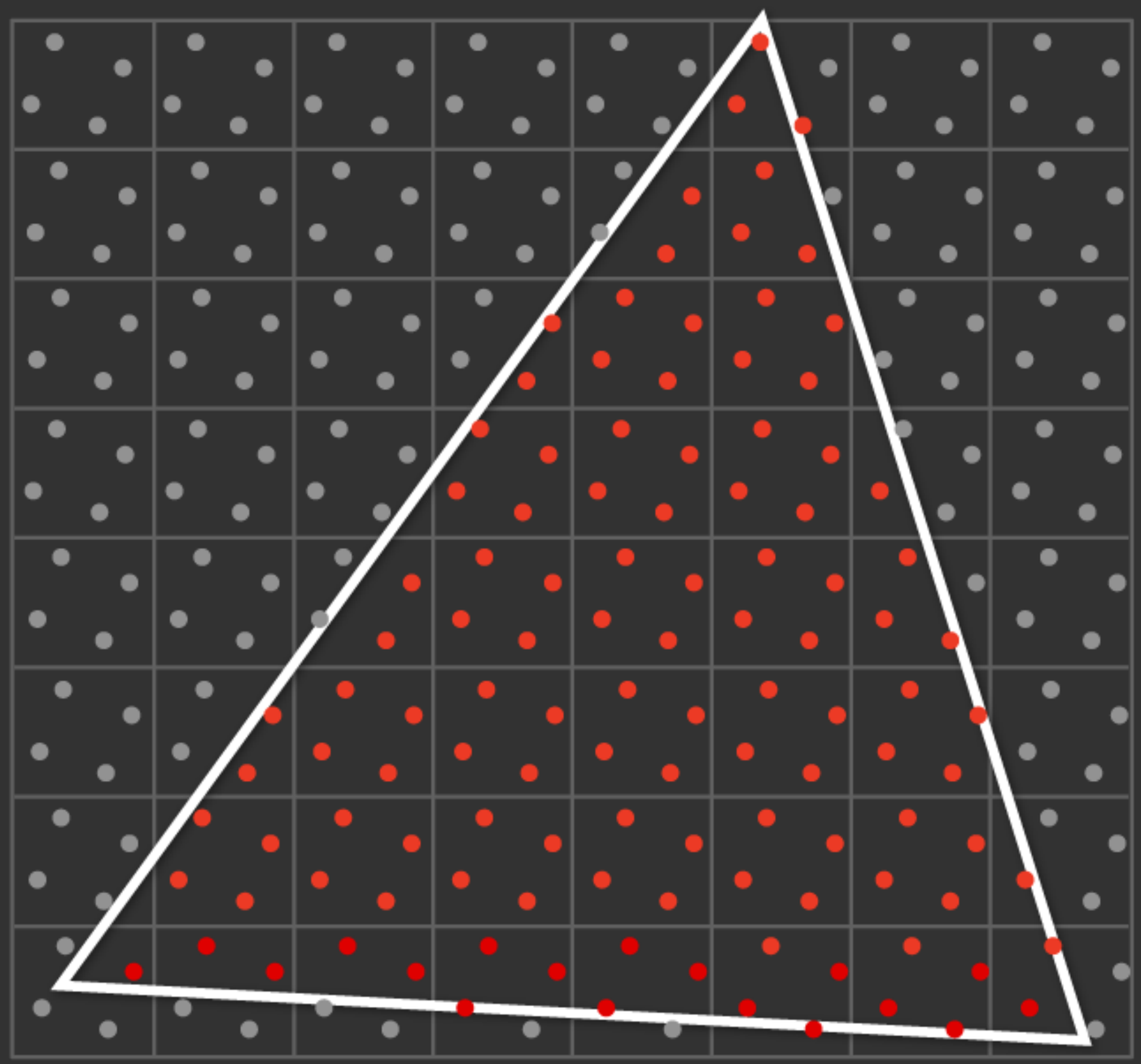
Key assumption: occlusion results do not depend on fragment shading

- Example operations that prevent use of this early Z optimization: enabling alpha test, fragment shader modifies fragment's Z value

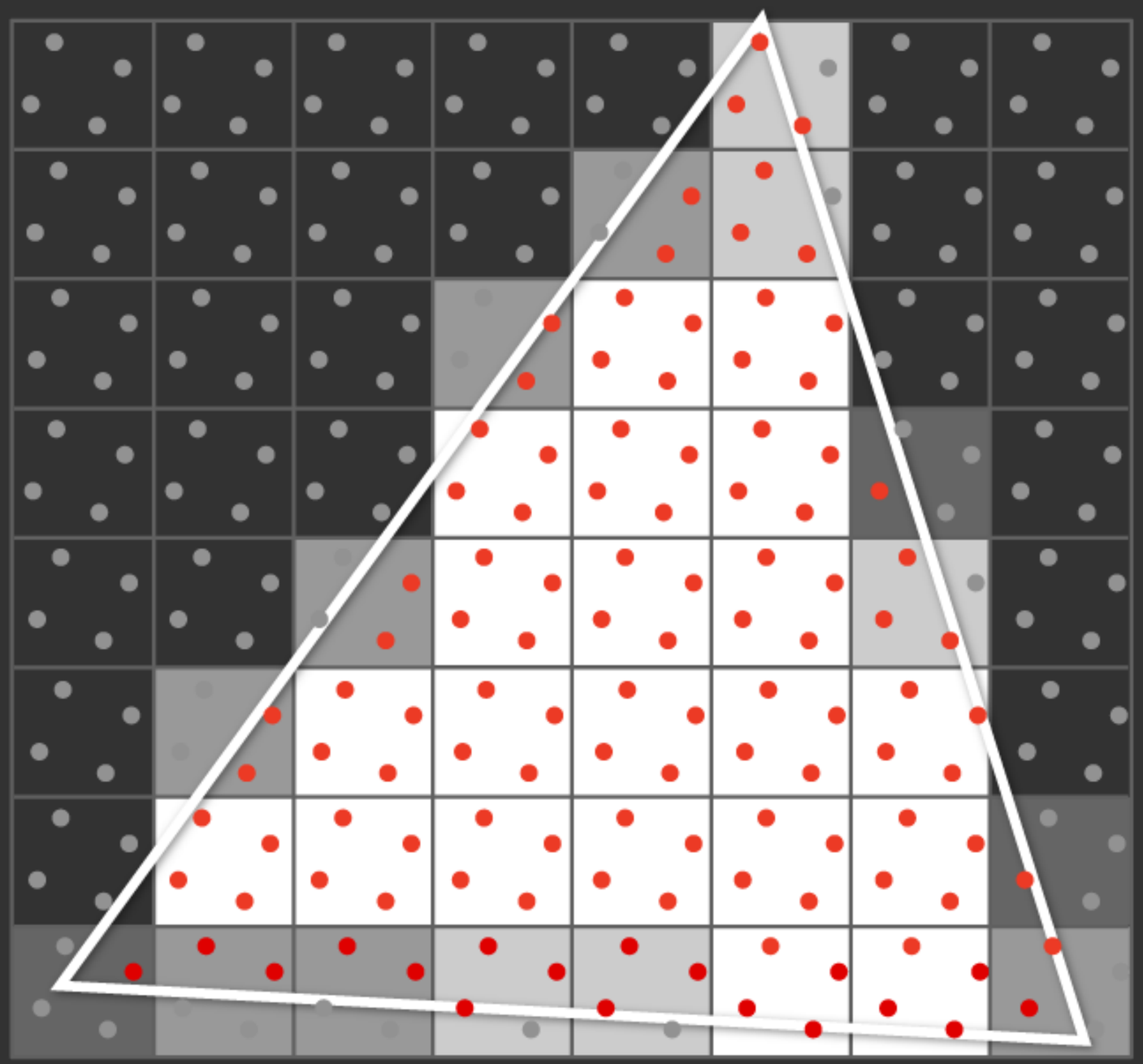
Multi-sample anti-aliasing

Supersampling coverage

Multiple point in triangle tests per pixel. Why?

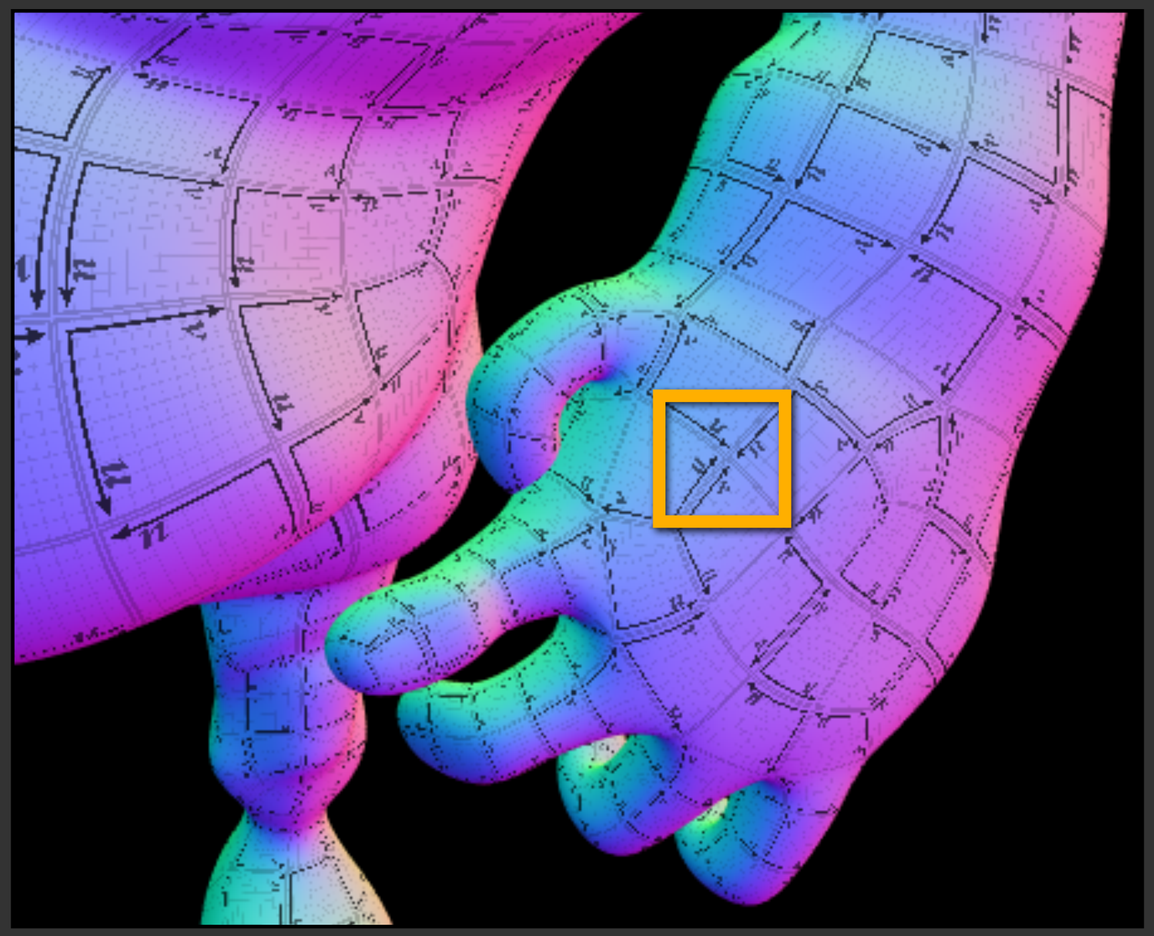


Supersampling to anti-alias triangle edges

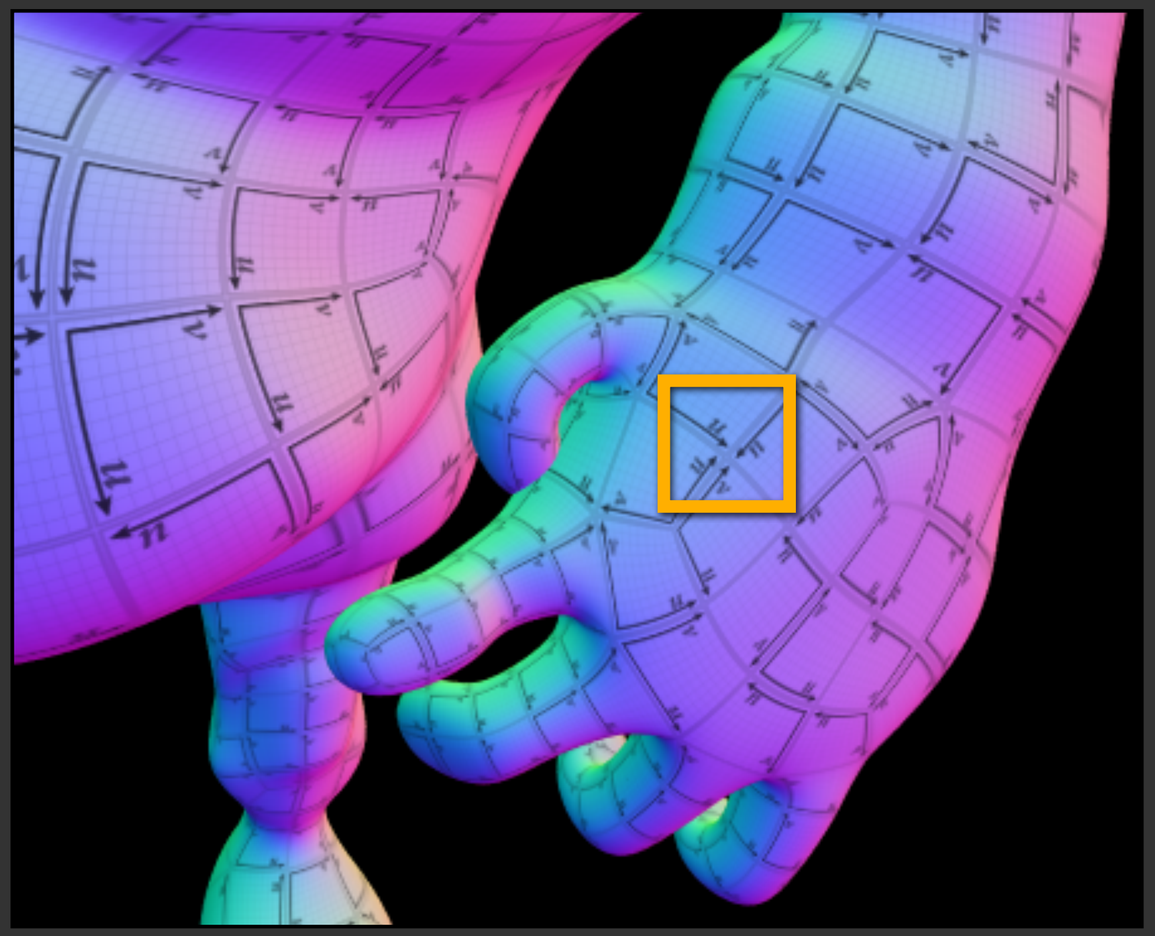


Texture data can be pre-filtered to avoid aliasing

Implication: ~ one shade per pixel is sufficient



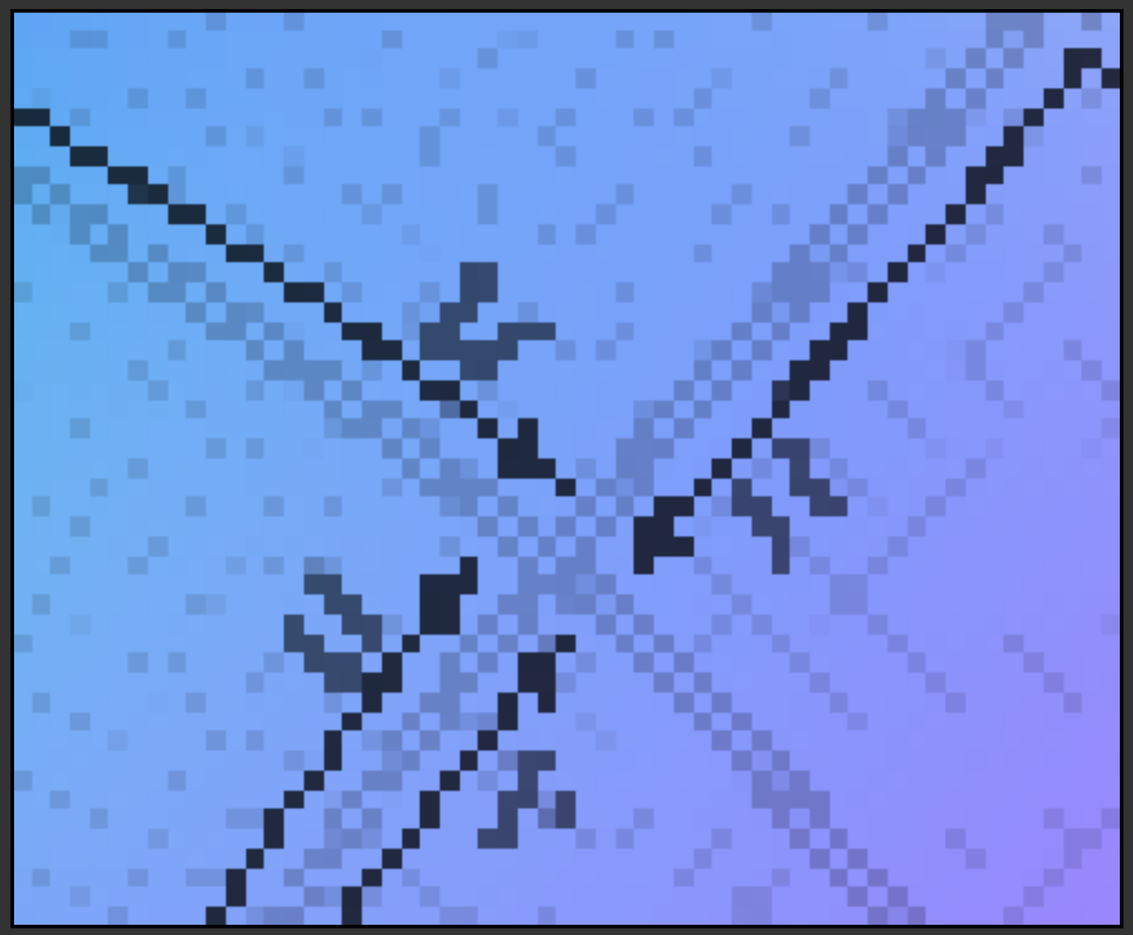
No pre-filtering
(aliased result)



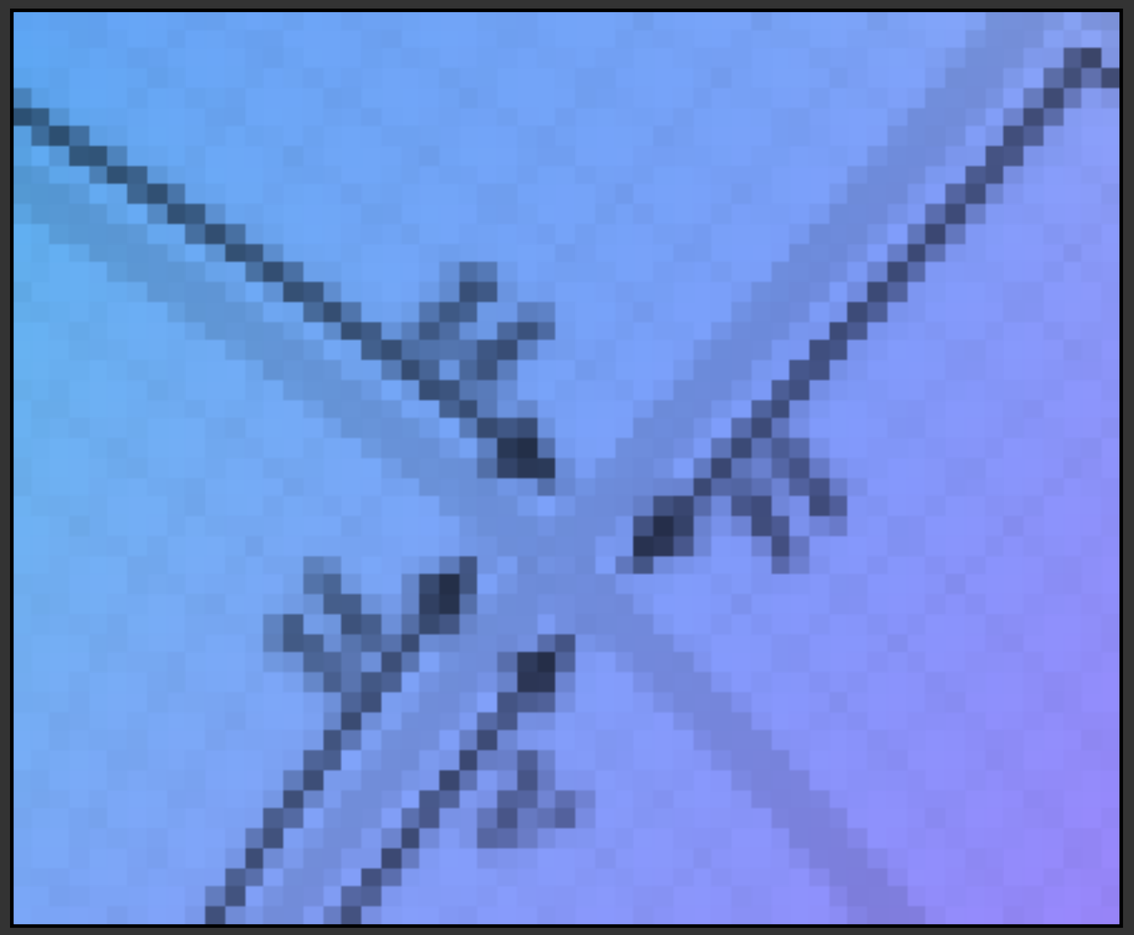
Pre-filtered texture

Texture data can be pre-filtered to avoid

Implication: ~ one shade per pixel is sufficient

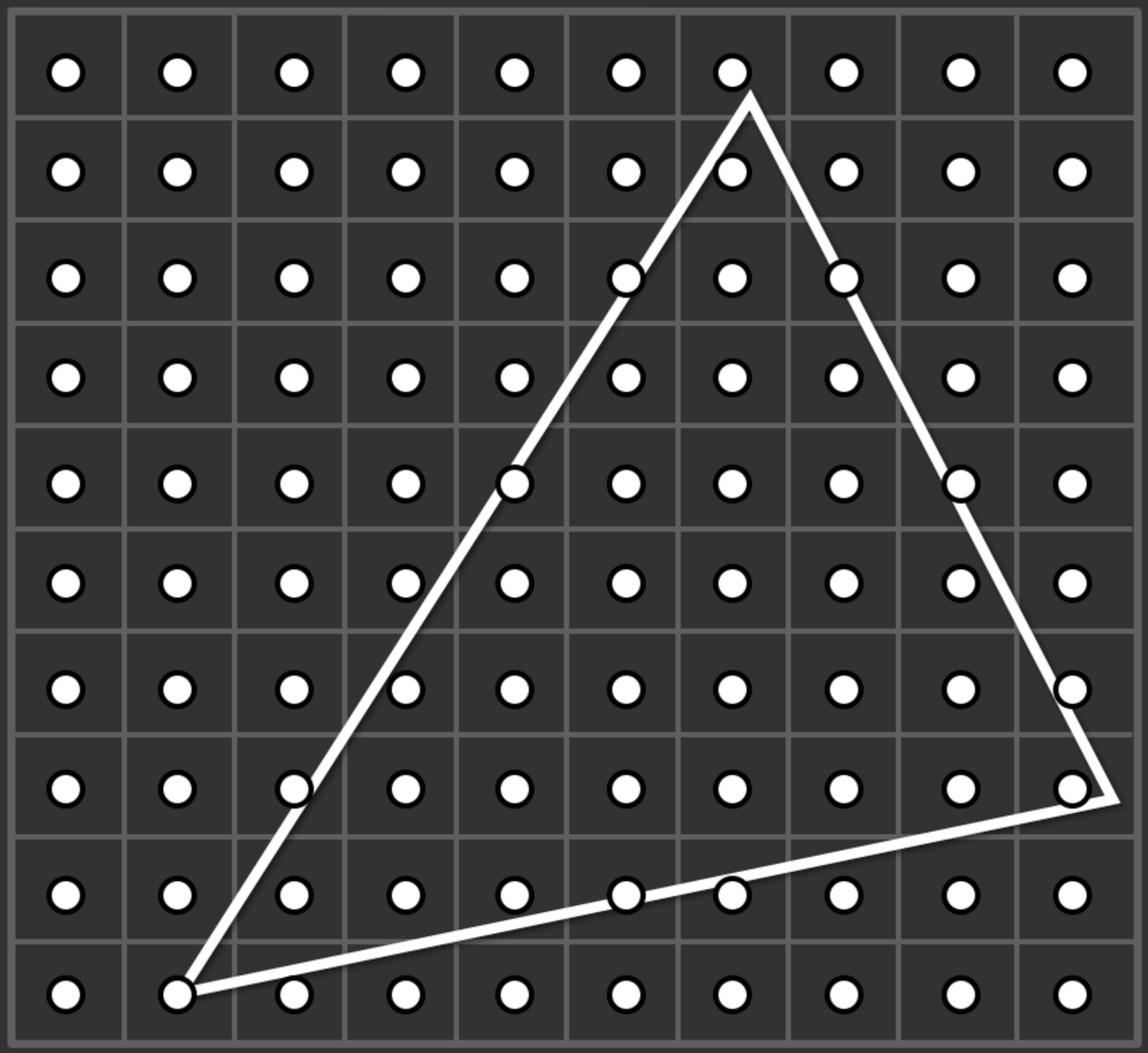


No pre-filtering
(aliased result)



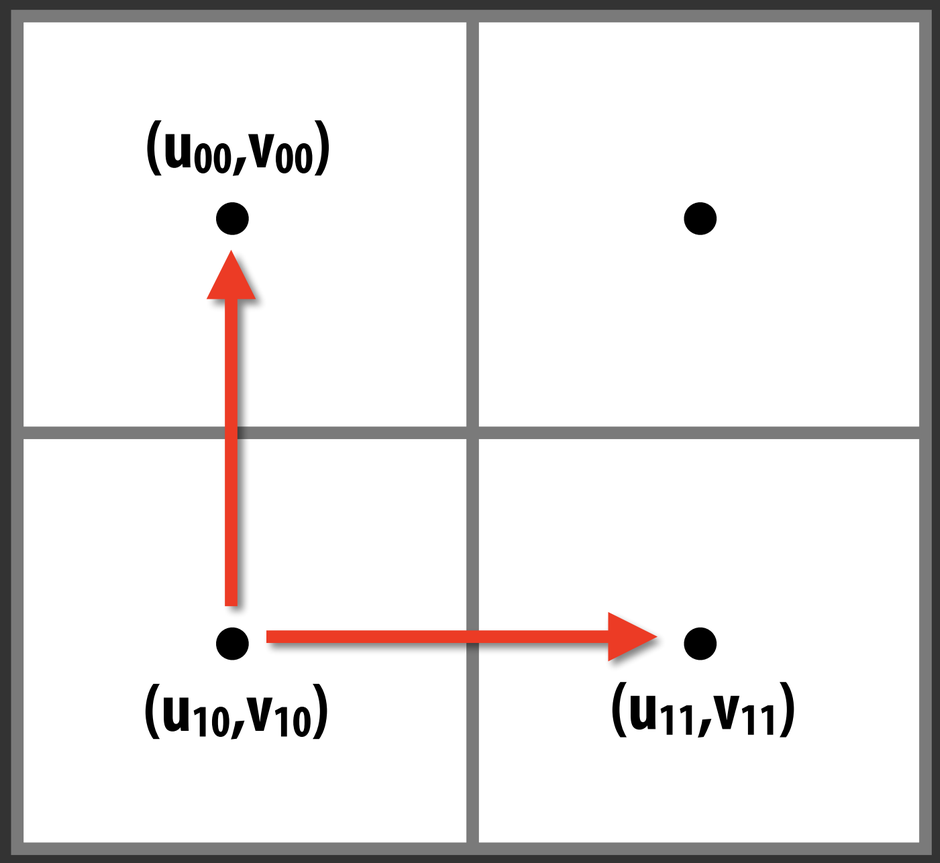
Pre-filtered texture

Shading sample locations

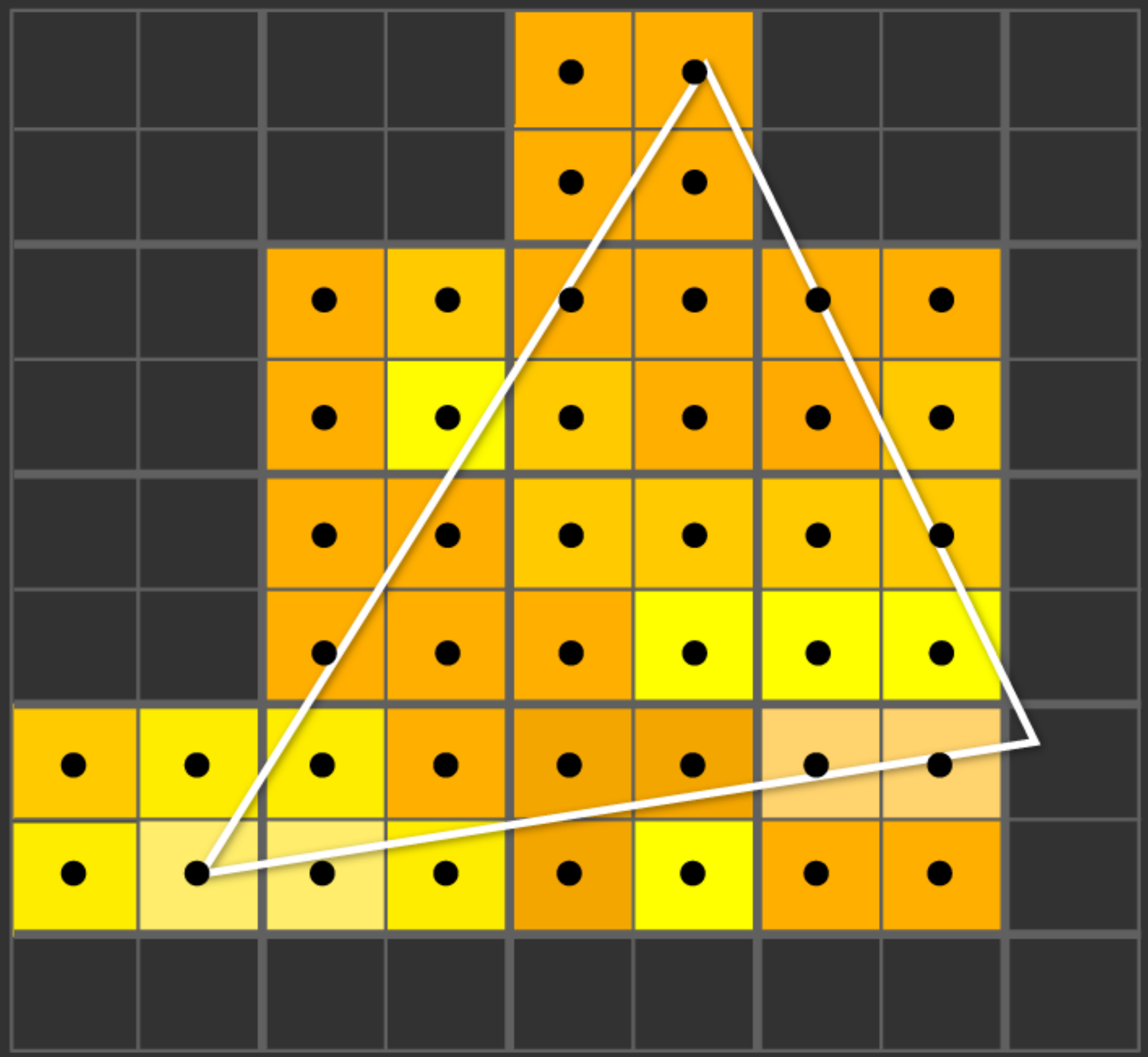


Quad fragments (2x2 pixel blocks)

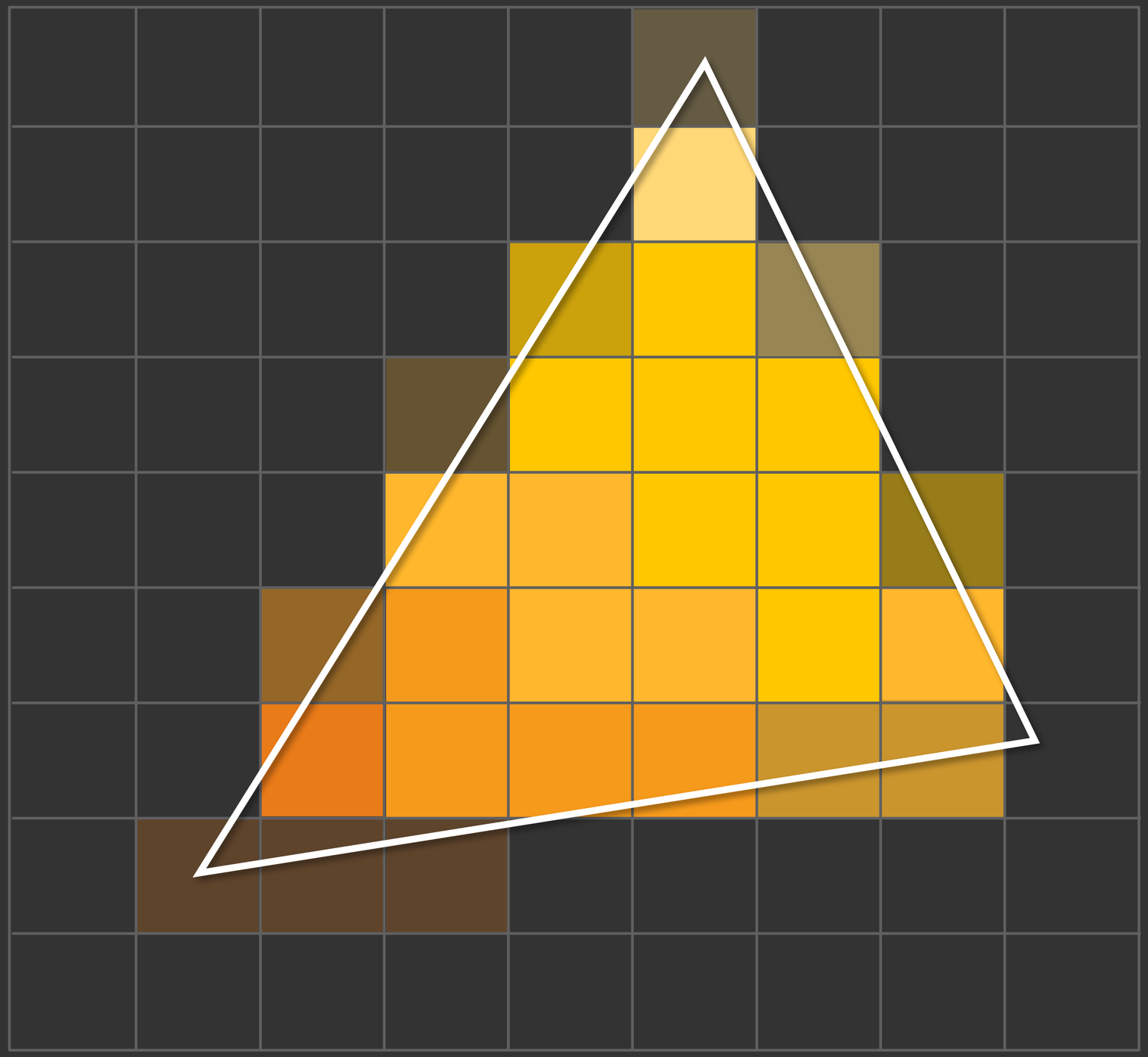
Difference neighboring texture coordinates to approximate derivatives



Shaded quad fragments

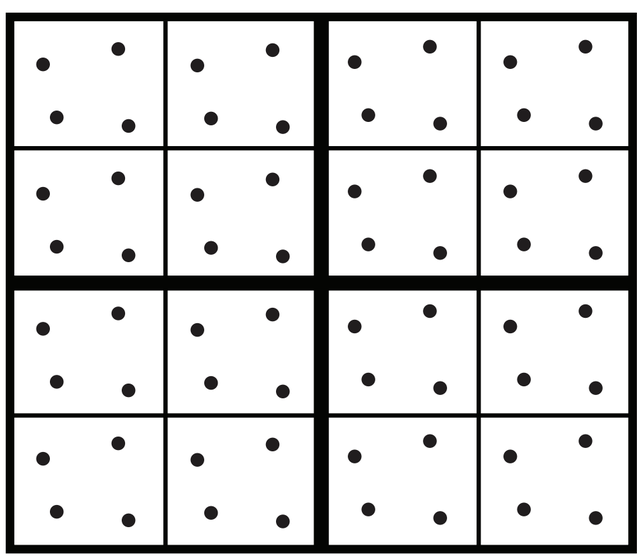


Final result: involving coverage

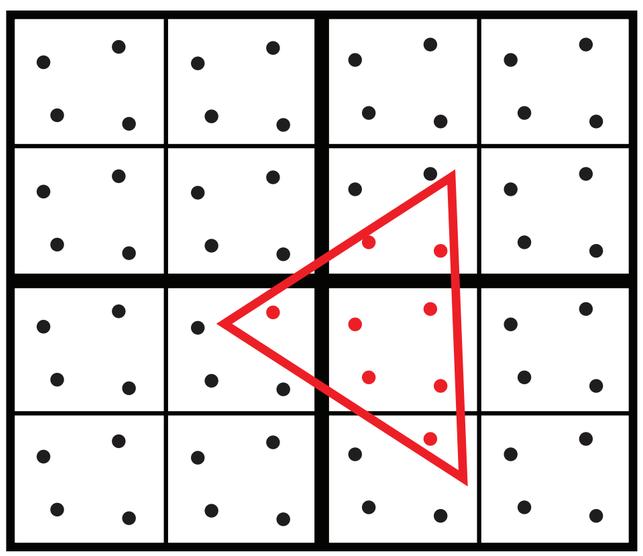


Multi-sample anti-aliasing

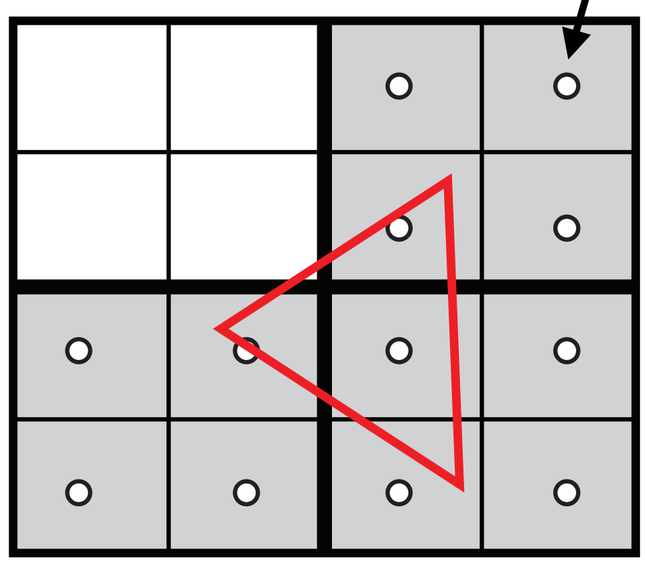
Sample surface visibility at a different (higher) rate than surface appearance.



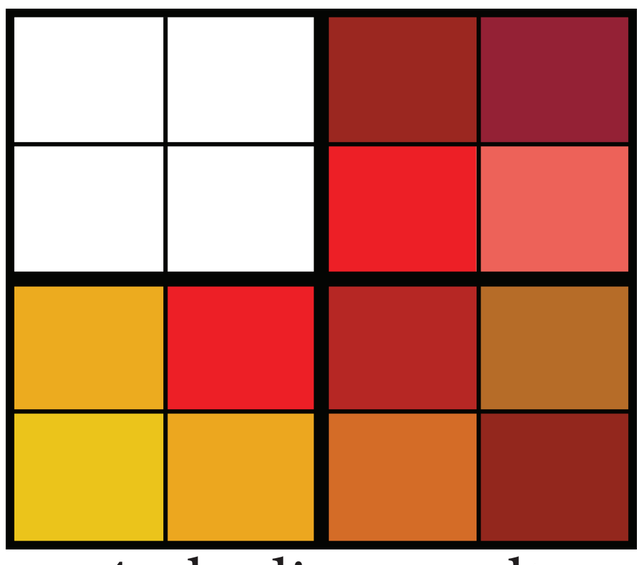
1. multi-sample locations



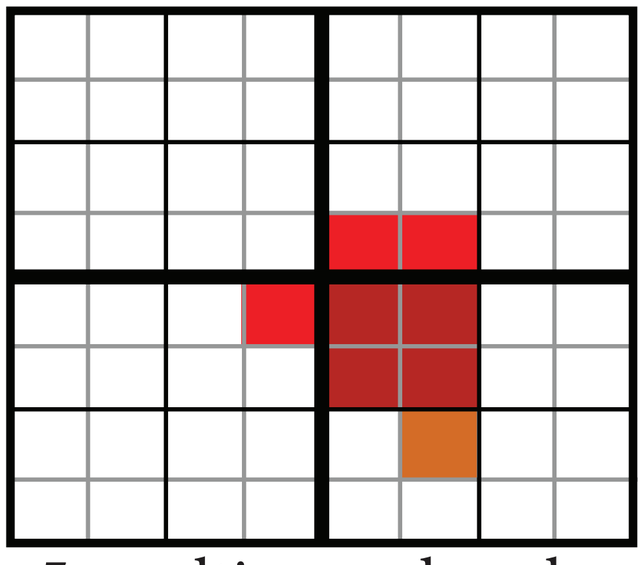
2. multi-sample coverage



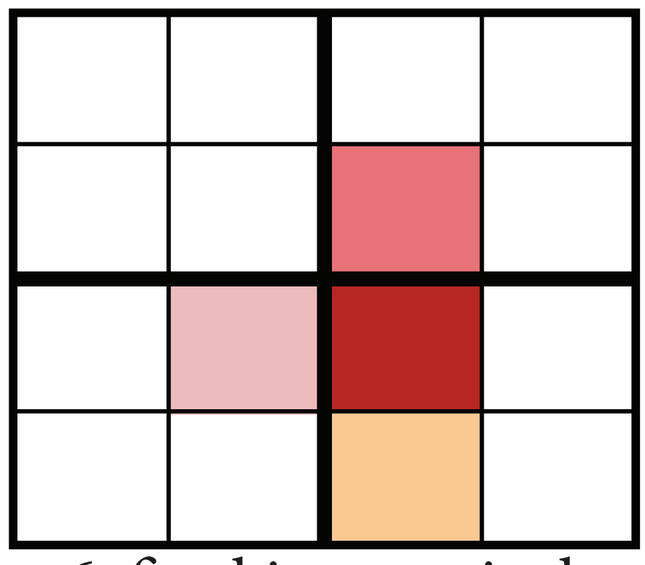
3. quad fragments



4. shading results



5. multi-sample color

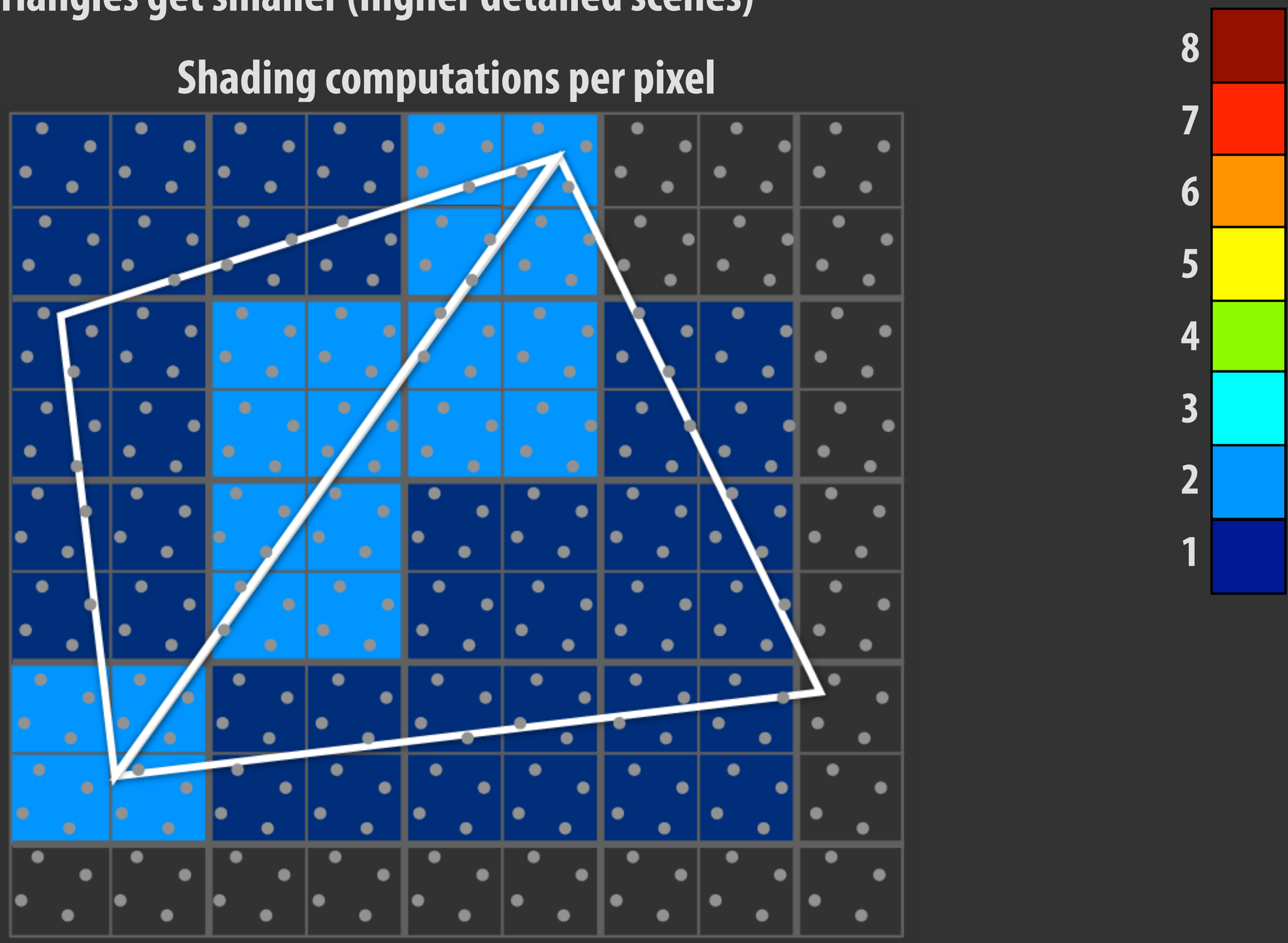


6. final image pixels

Idea: use supersampling to anti-alias detail due to geometric visibility, use texture prefiltering (mipmapped texture access) to anti-alias detail to texture

Problem: pixels along edges shaded multiple times

Ug… technique designed to reduce shading in large triangle case actually increases shading when triangles get smaller (higher detailed scenes)



Read data less often

Reading less data conserves power

- Goal: redesign algorithms so that they make good use of on-chip memory or processor caches
  - And therefore transfer less data from memory

- A fact you might not have heard:

  — It is *far more* costly (in energy) to load/store data from memory, than it is to perform an arithmetic operation

"Ballpark" numbers

[Sources: Bill Dally (NVIDIA), Tom Olson (ARM)]

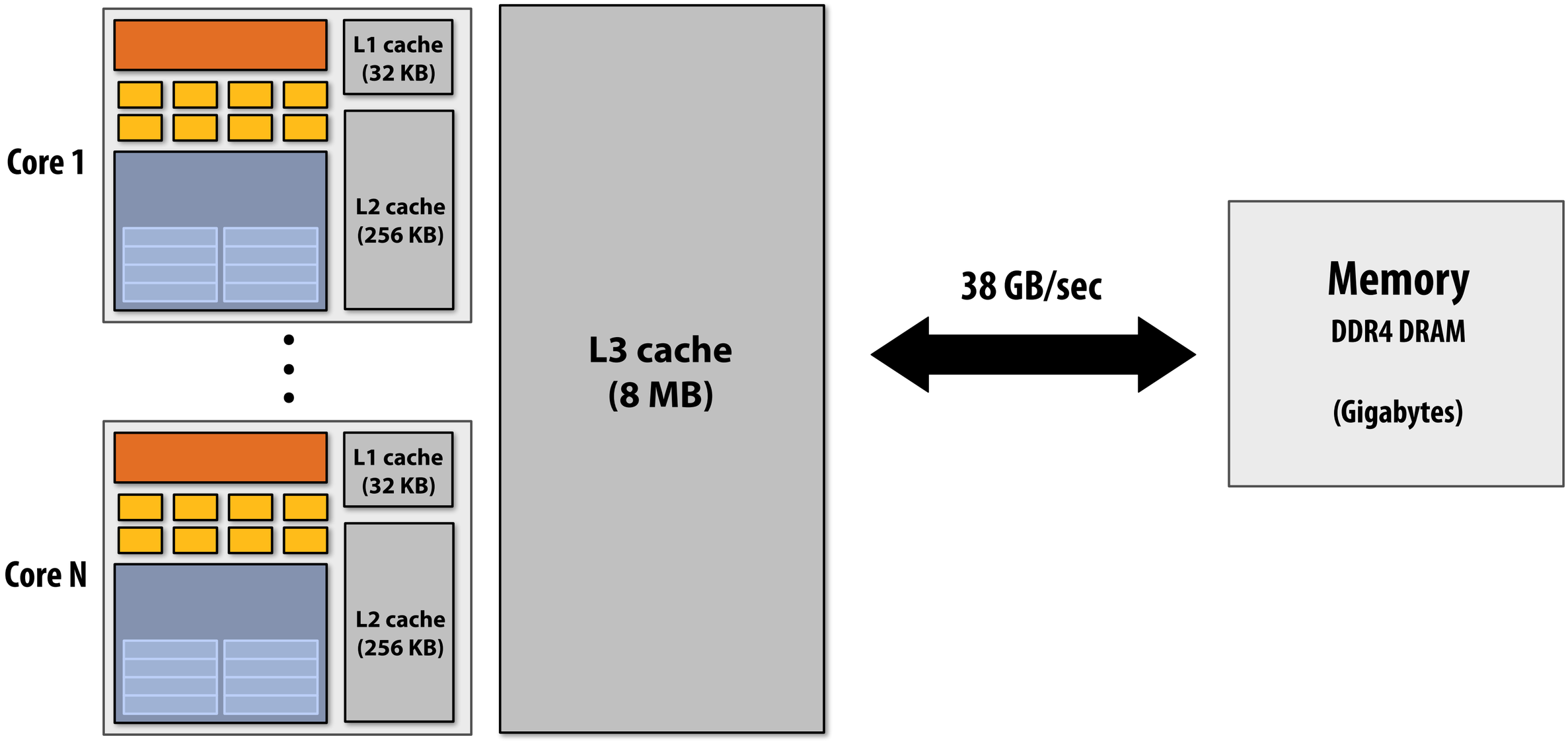
- Integer op: ~ 1 pJ *
- Floating point op: ~20 pJ *
- Reading 64 bits from small local SRAM (1mm away on chip): ~ 26 pJ
- Reading 64 bits from low power mobile DRAM (LPDDR): ~1200 pJ

Implications

- Reading 10 GB/sec from memory: ~1.6 watts

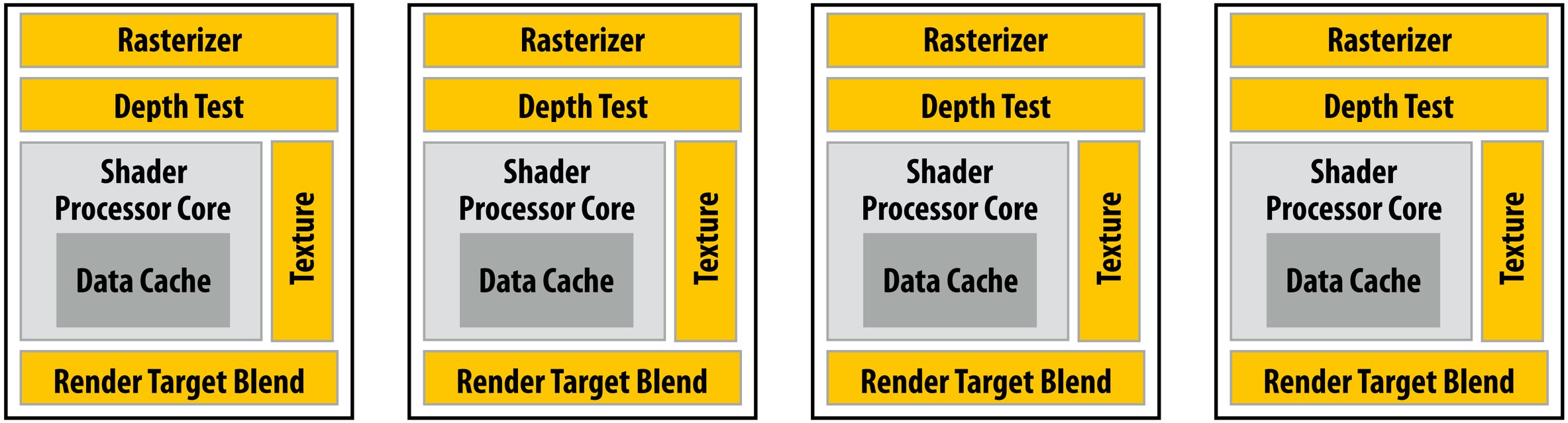
* Cost to just perform the logical operation, not counting overhead of instruction decode, load data from registers, etc.

Review:
What does a data cache do in a processor?



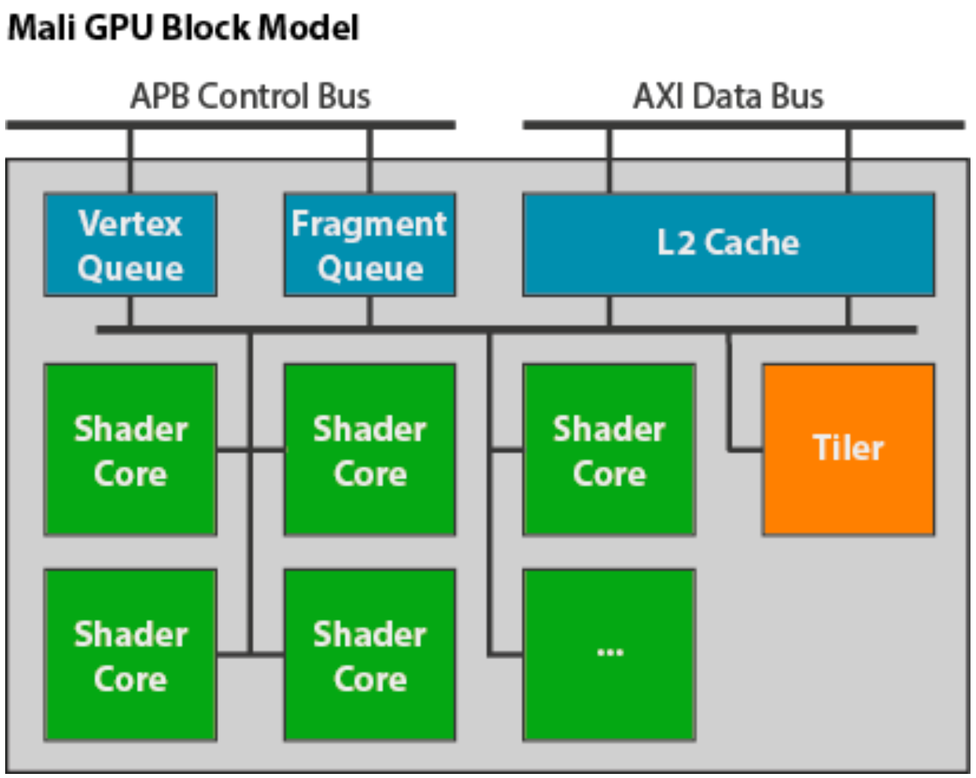
Today: a simple mobile GPU

- A set of programmable cores (run vertex and fragment shader programs)
- Hardware for rasterization, texture mapping, and frame-buffer access

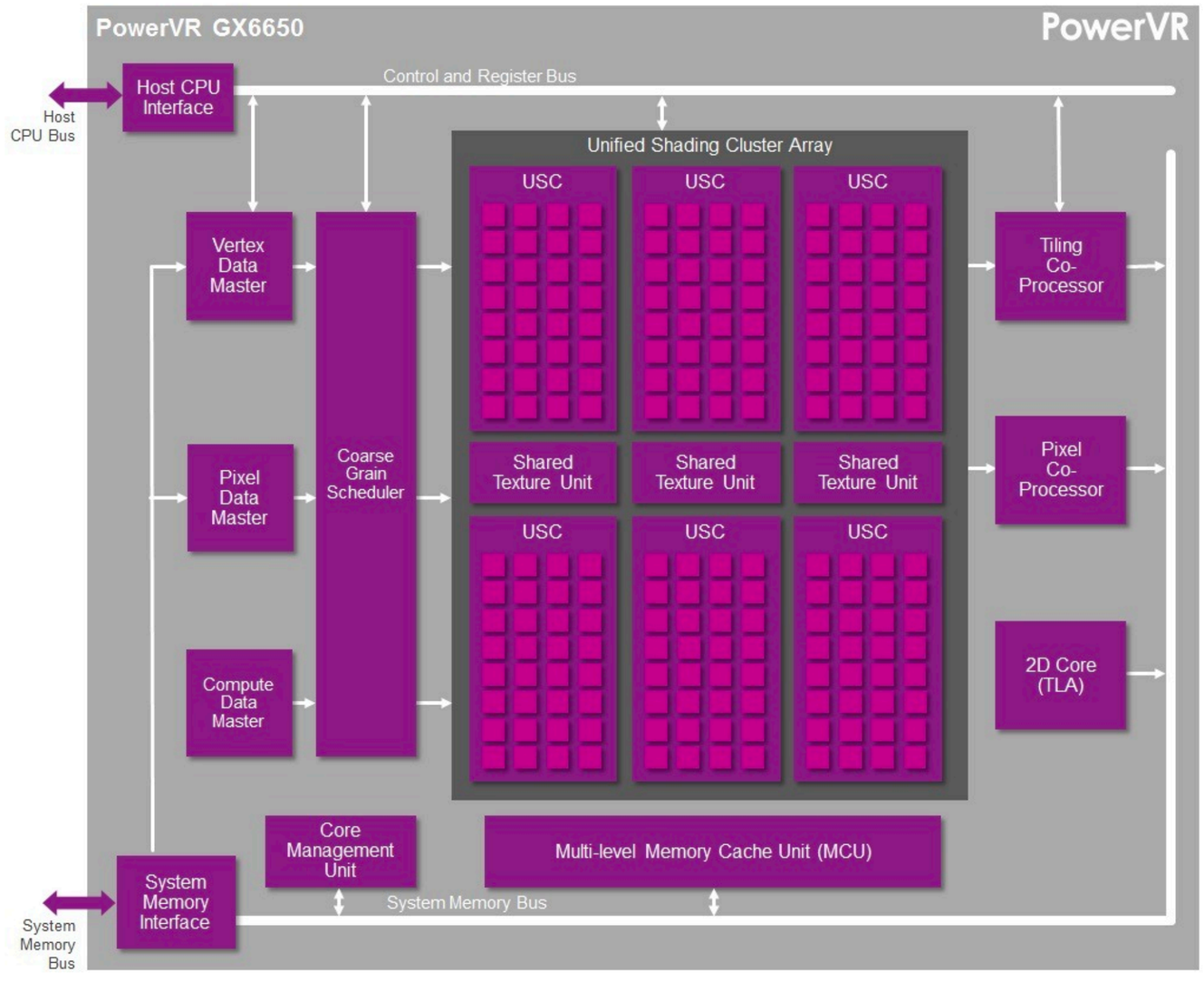


Block diagrams from vendors

ARM Mali G72MP18



Imagination PowerVR (in earlier iPhones)



Let's consider different workloads

Average triangle size

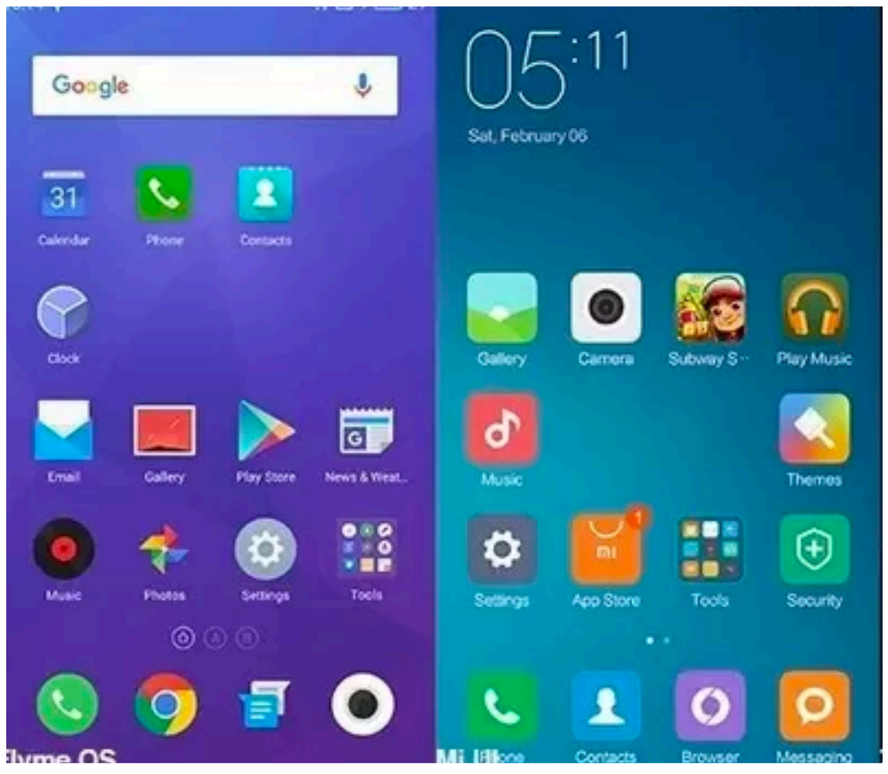
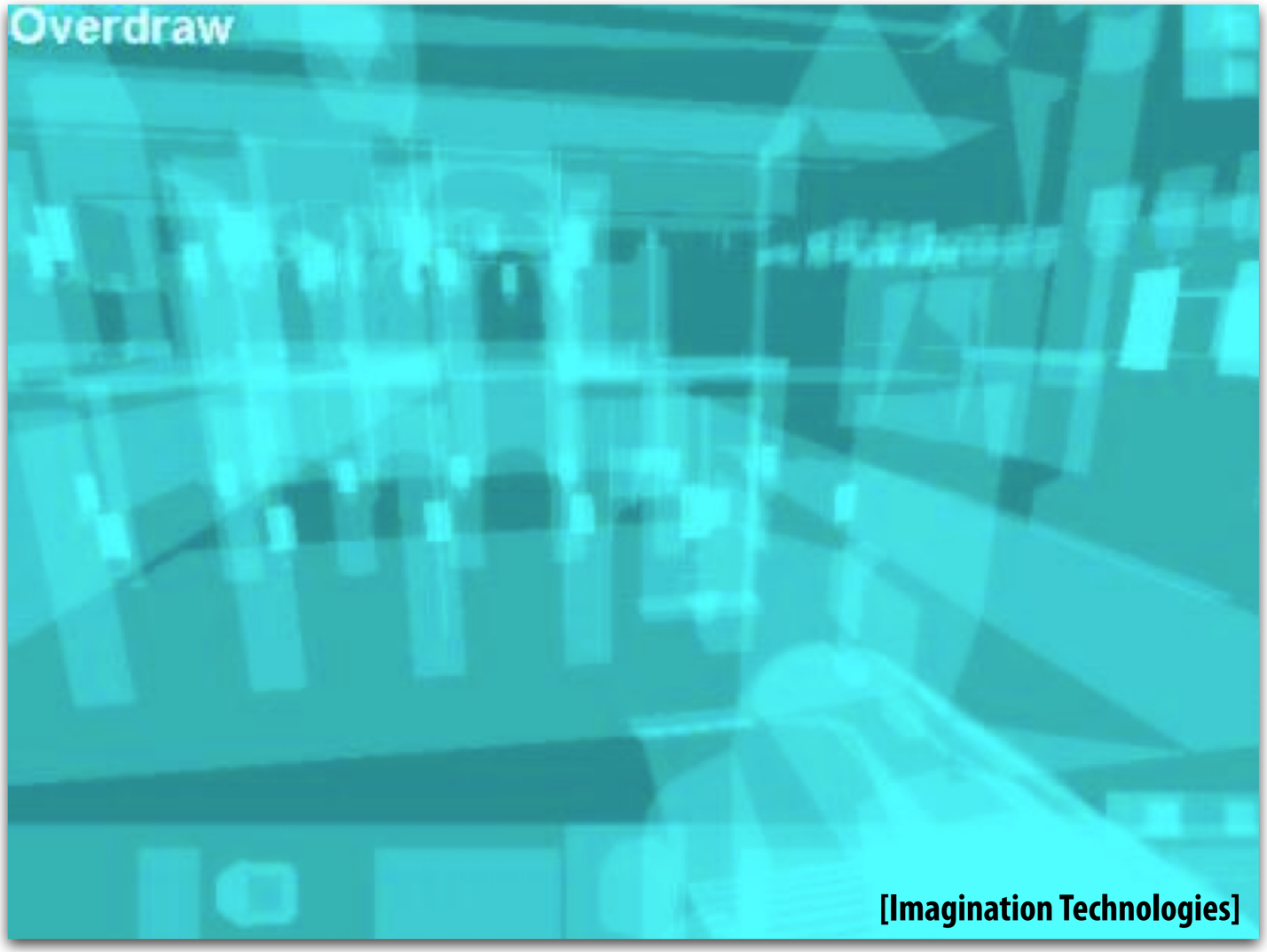


Image credit:
https://www.theverge.com/2013/11/29/5155726/next-gen-supplementary-piece
http://www.mobygames.com/game/android/ghostbusters-slime-city/screenshots/gameShotId,852293/

Let's consider different workloads

Scene depth complexity

Average number of overlapping triangles per pixel



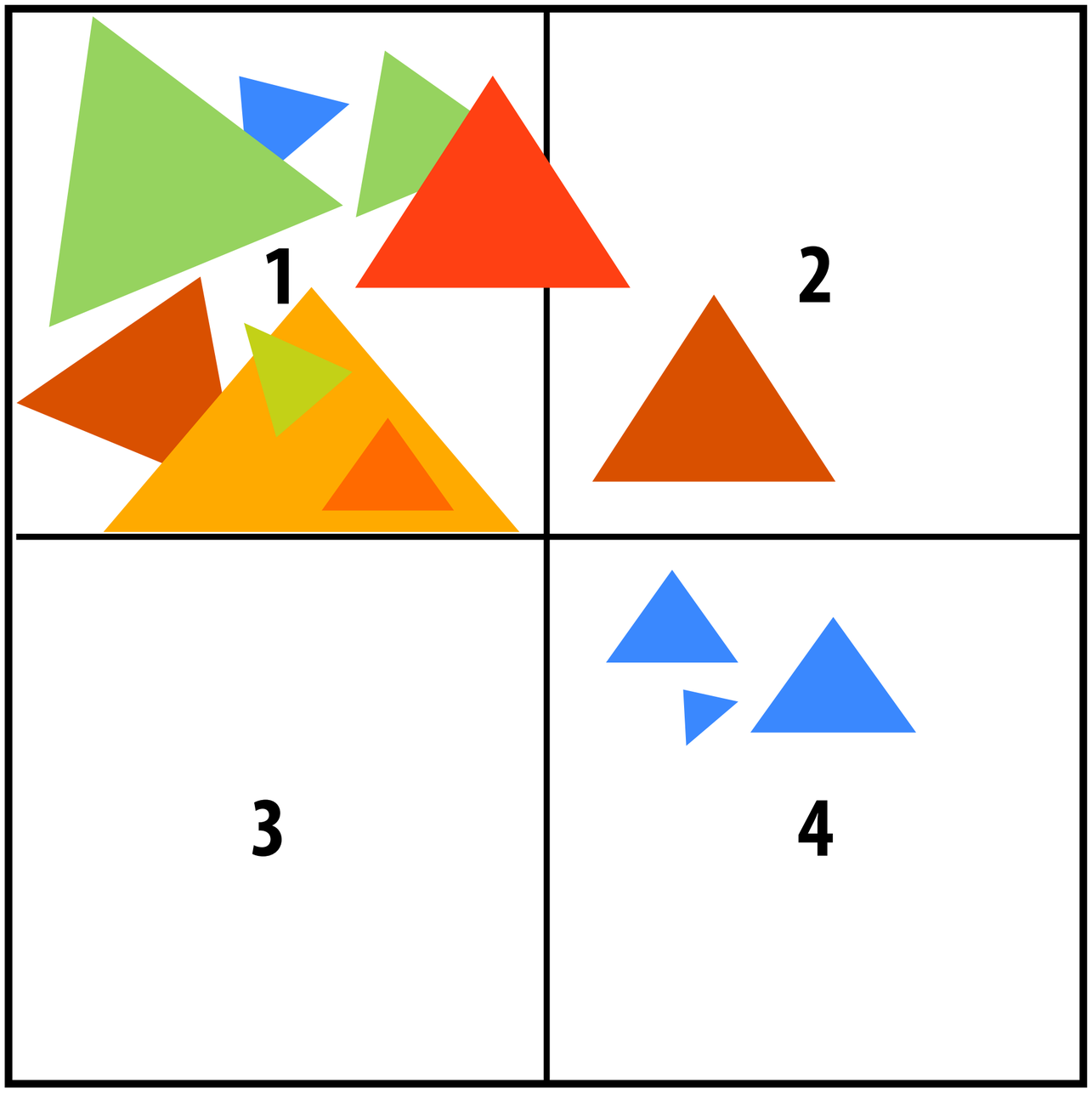
In this visualization: bright colors = more overlap

One very simple solution

- Let's assume four GPU cores
- Divide screen into four quadrants, each processor processes all triangles, but only renders triangles that overlap quadrant
- *Problems?*

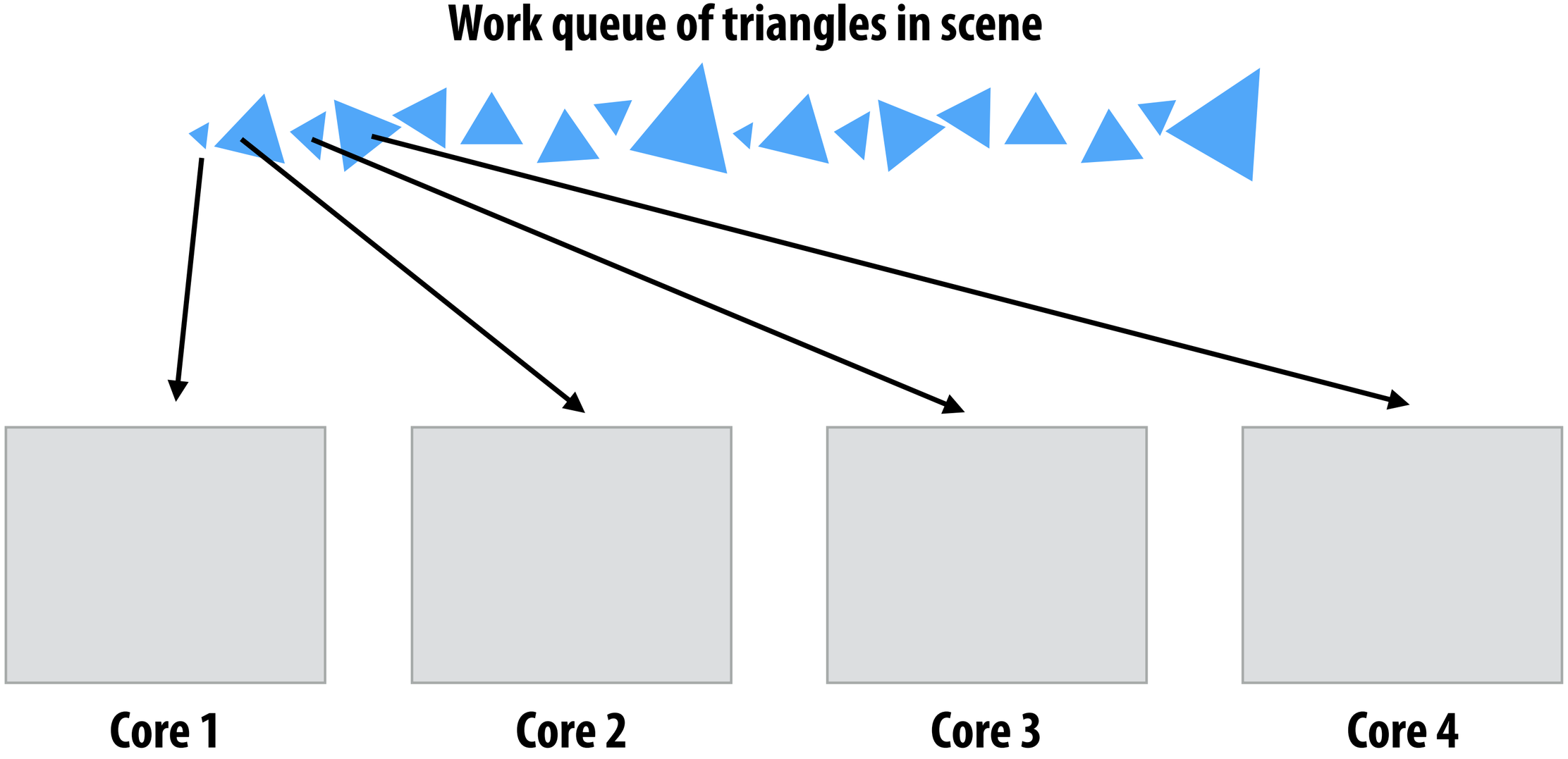
Unequal work partitioning

(partition the primitives to parallel units based on screen overlap)



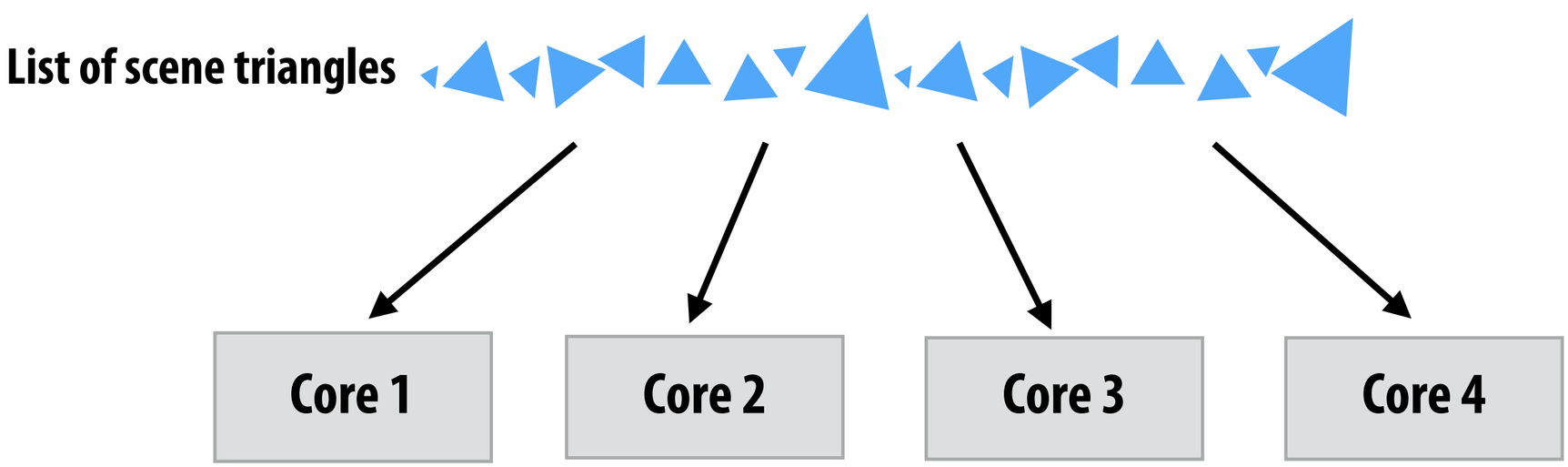
Step 1: parallel geometry processing

- Distribute triangles to render to the processors (e.g., round robin)
- Processors performs vertex processing



Step 1 produces list of triangle bins

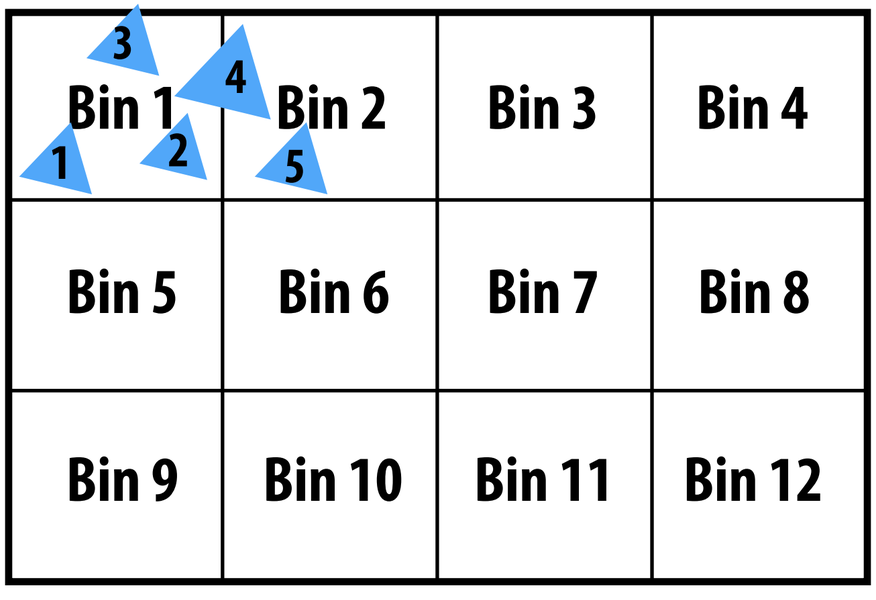
- One bin per "tile" of screen
- Core runs vertex processing, computes 2D triangle/screen-tile overlap, inserts triangle into appropriate bin(s)



After processing first five triangles:

Bin 1 list: 1,2,3,4

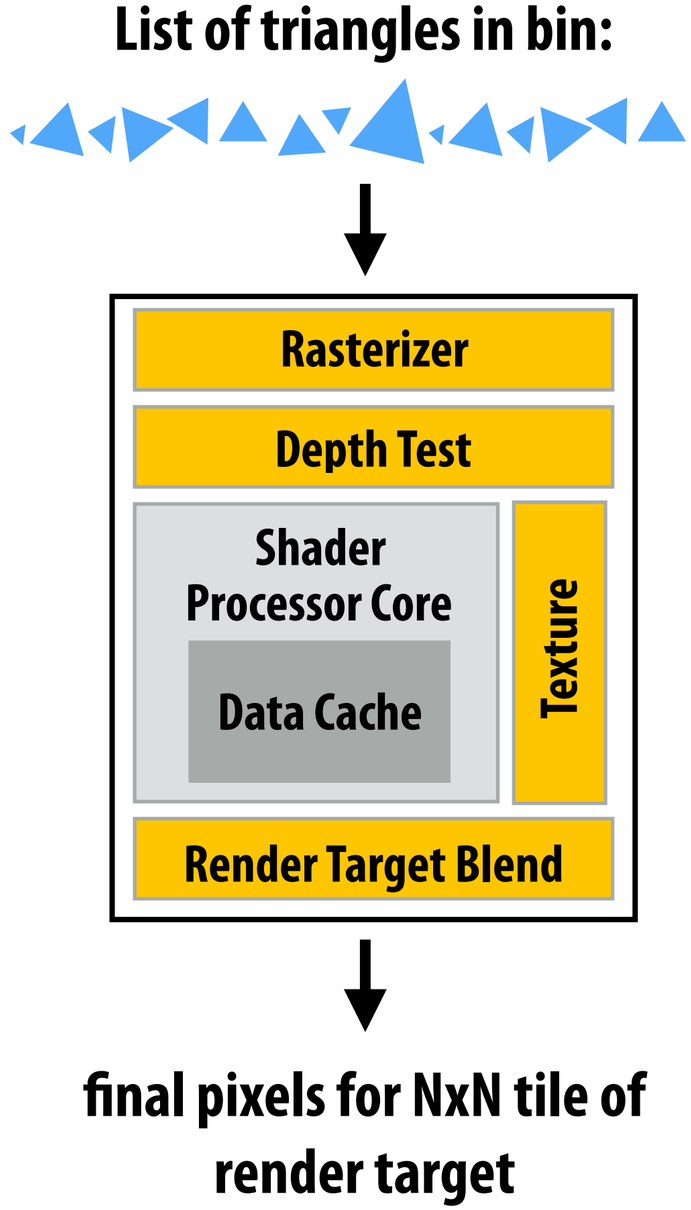
Bin 2 list: 4,5



Step 2: per-tile processing

- Cores process bins in parallel performing rasterization fragment shading and frame buffer update

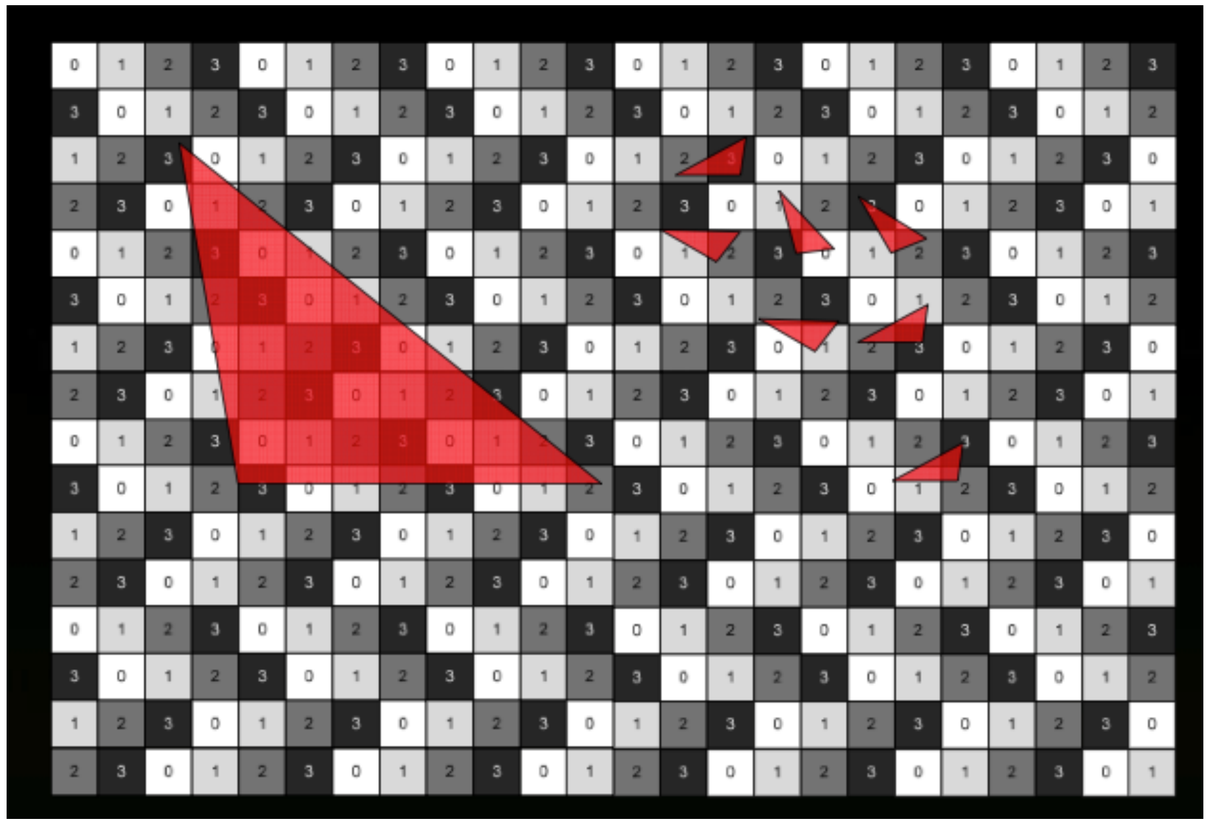
- While (more bins left to process):
  - Assign bin to available core
  - For all triangles in bin:
    - Rasterize
    - Fragment shade
    - Depth test
    - Render target blend



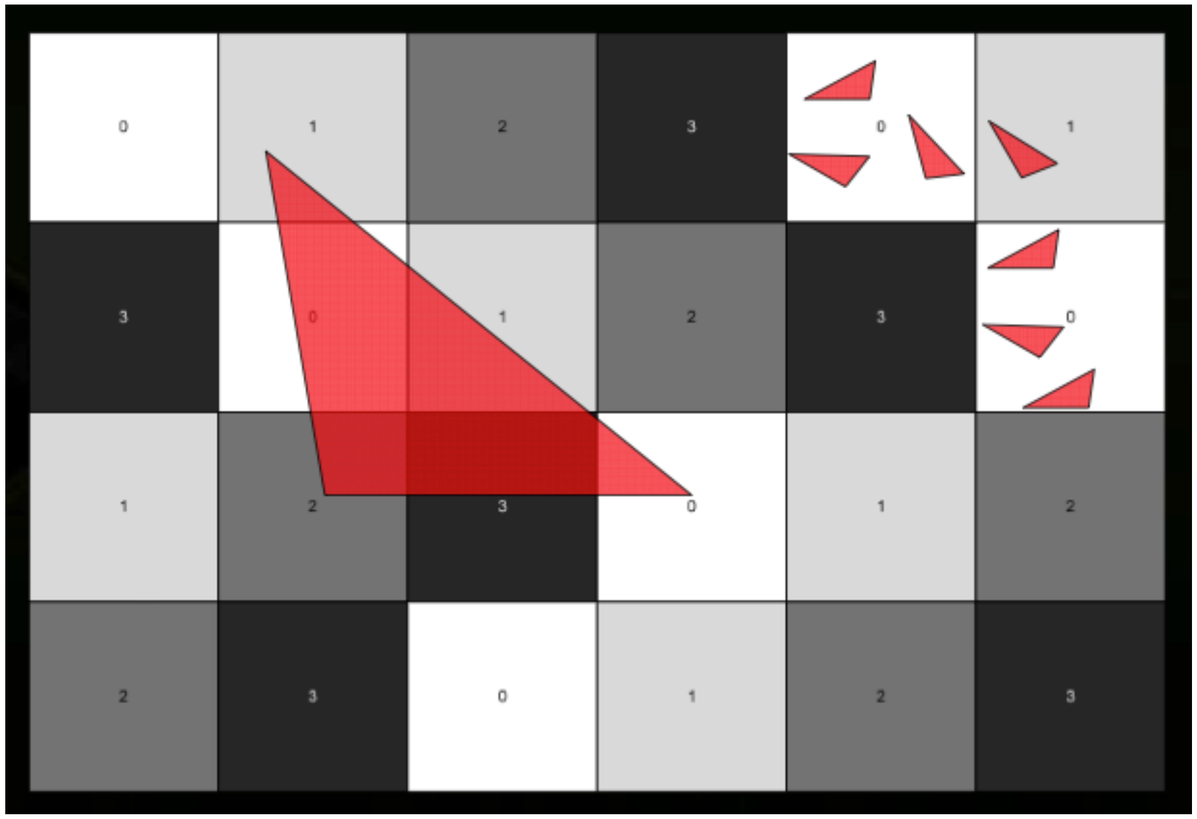
What should the size of the bins be?

What should the size of the bins be?

Fine granularity interleaving



Coarse granularity interleaving



[Image source: NVIDIA]

What size should the bins be?

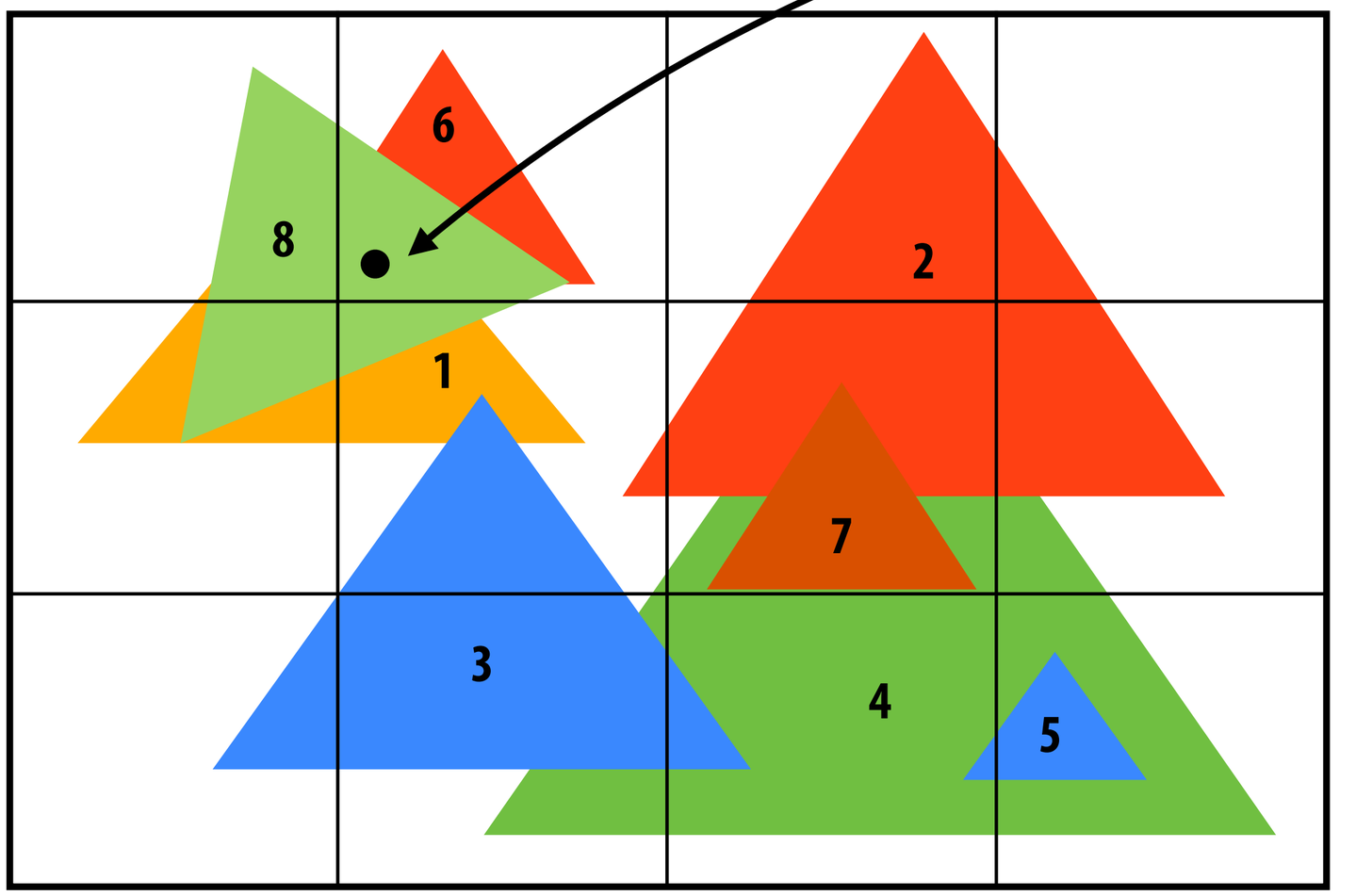
- Small enough for a tile of the color buffer and depth buffer (potentially supersampled) to fit in a shader processor core's on-chip storage (i.e., cache)

- Tile sizes in range 16x16 to 64x64 pixels are common

- ARM Mali GPU: commonly uses 16x16 pixel tiles

Tiled rendering "sorts" the scene in 2D space to enable efficient color/depth buffer access

Consider rendering without a sort:
(process triangles in order given)



This sample updated three times, but may have fallen out of cache in between accesses

Now consider step 2 of a tiled renderer:

```
Initialize Z and color buffer for tile
for all triangles in tile:
  for all each fragment:
    shade fragment
    update depth/color
write color tile to final image buffer
```

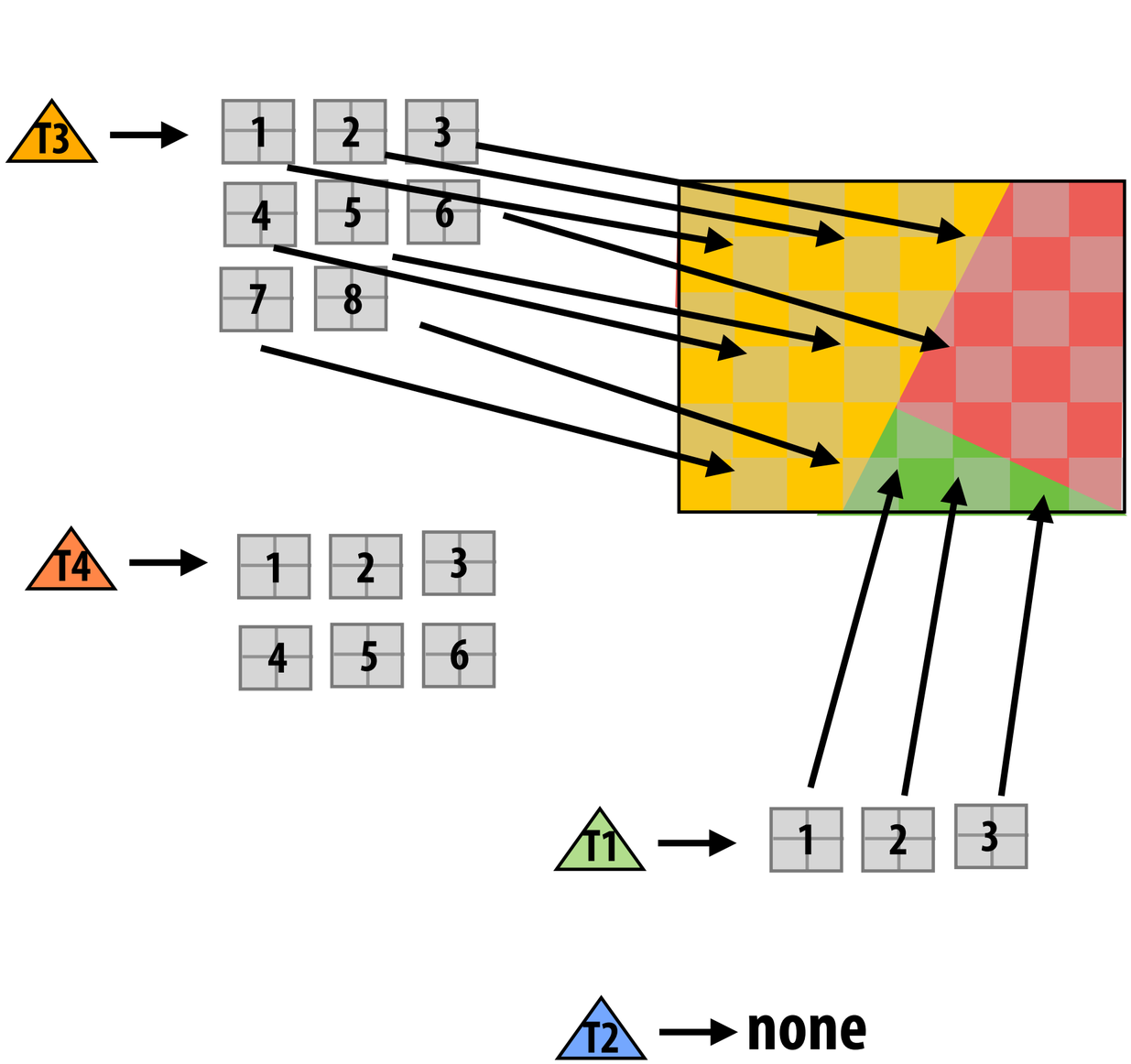
Q. Why doesn't the renderer need to read color or depth buffer from memory?

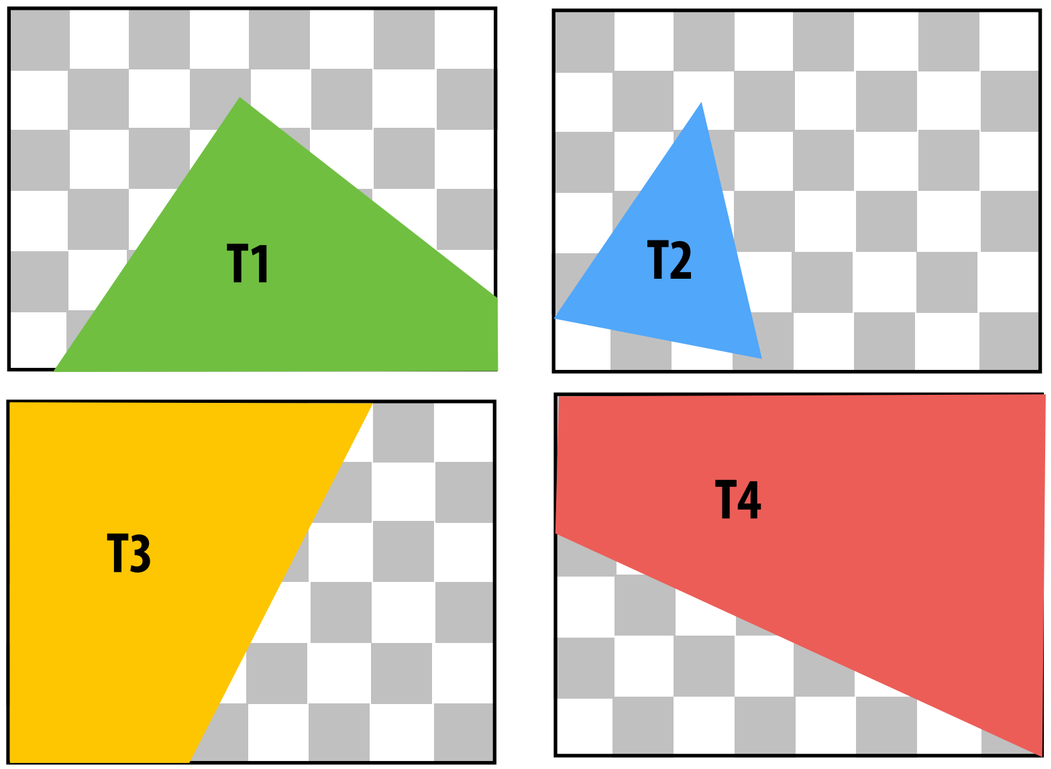
Q. Why doesn't the renderer need to write depth buffer in memory? *

*** Assuming application does not need depth buffer for other purposes.**

Tile-based deferred rendering (TBDR)

- Mobile GPUs implement deferred shading in the hardware!
- Divide step 2 of tiled pipeline into two phases:
- Phase 1: compute what triangle/quad fragment is visible at every sample
- Phase 2: perform shading of only the visible quad fragments

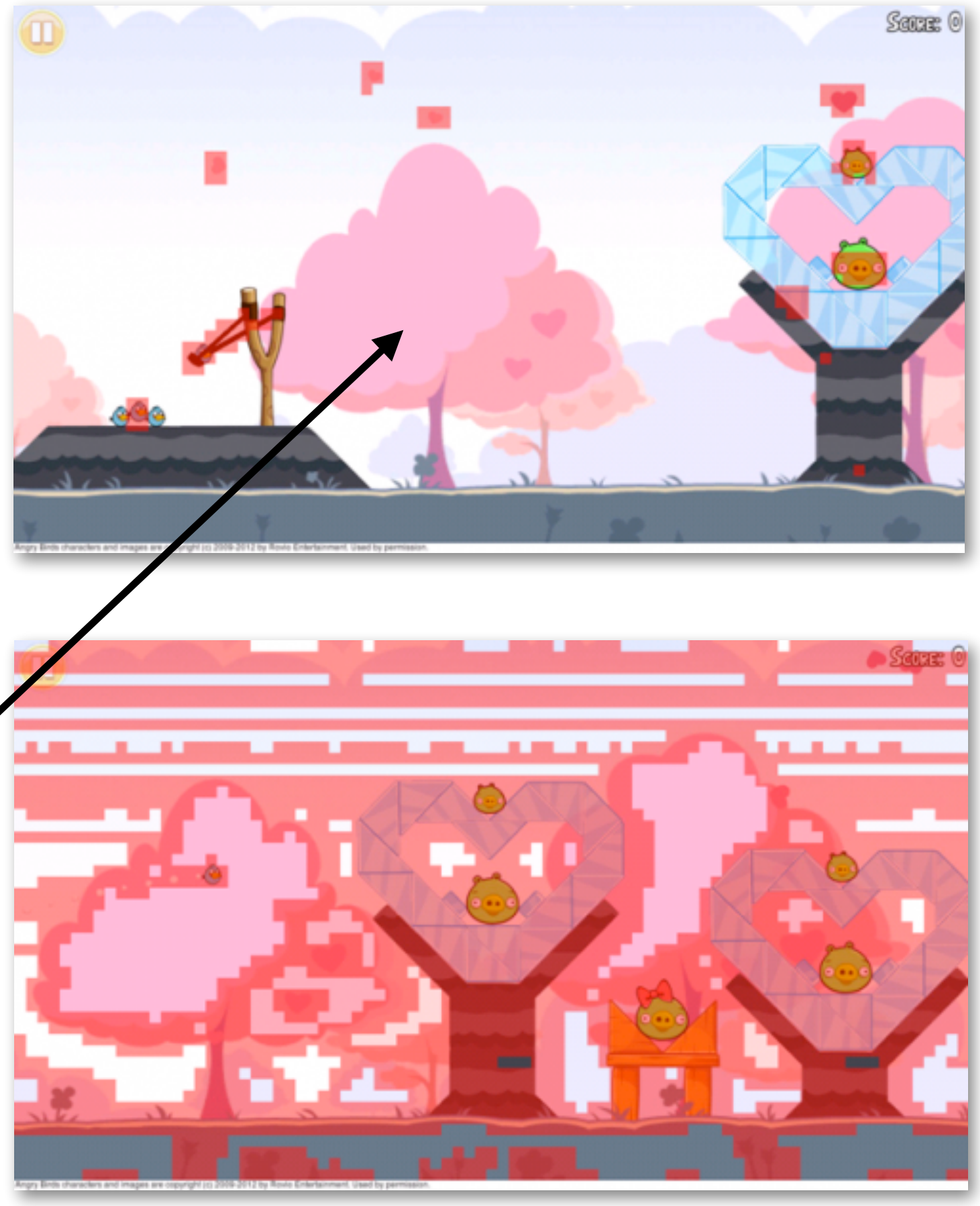




Mobile GPU architects go to many steps to reduce bandwidth to save power

- Compress frame buffer
- Compress texture data
- Eliminate unnecessary memory writes!

- Frame 1:
  - Render frame as normal
  - Compute hash of pixels in each tile on screen
- Frame 2:
  - Render frame tile at a time
  - Before storing pixel values for tile to memory, compute hash and see if tile's contents are the same as in the last frame
    - If yes, skip memory write



Slow camera motion: 96% of writes avoided
Fast camera motion: ~50% of writes avoided
(red tile = required a memory write)

[Source: Tom Olson http://community.arm.com/groups/arm-mali-graphics/blog/2012/08/17/how-low-can-you-go-building-low-power-low-bandwidth-arm-mali-gpus]

Summary

- Mobile 3D graphics implementations are highly optimized for power efficiency
  - Tiled rendering for bandwidth efficiency*
  - Deferred rendering to reduce shading costs
  - Many additional optimizations such as buffer compression, eliminating unnecessary memory ops, etc.

- If you enjoy these topics, consider CS348K (Visual Computing Systems — offered in Fall quarter)

* Not all mobile GPUs use tiled rendering as described in this lecture.