Lecture 18:

Course Recap + 3D Graphics on Mobile GPUs

Interactive Computer Graphics
Stanford CS248, Spring 2018

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

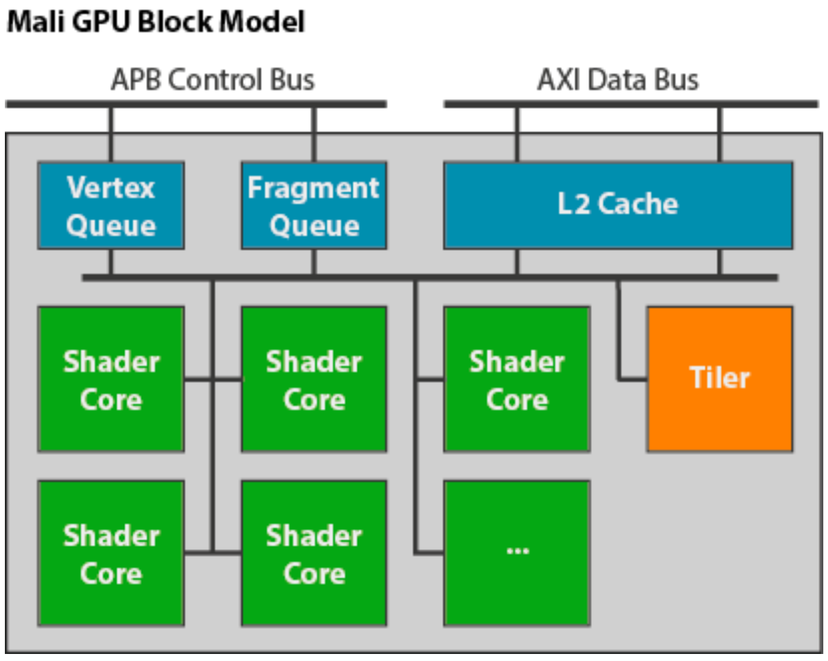
Apple iPhone 8

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

One way to conserve power

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

Another way to conserve power

- Read data from memory less often
  - Redesign algorithms so that they make good use of on-chip memory or processor caches

- 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

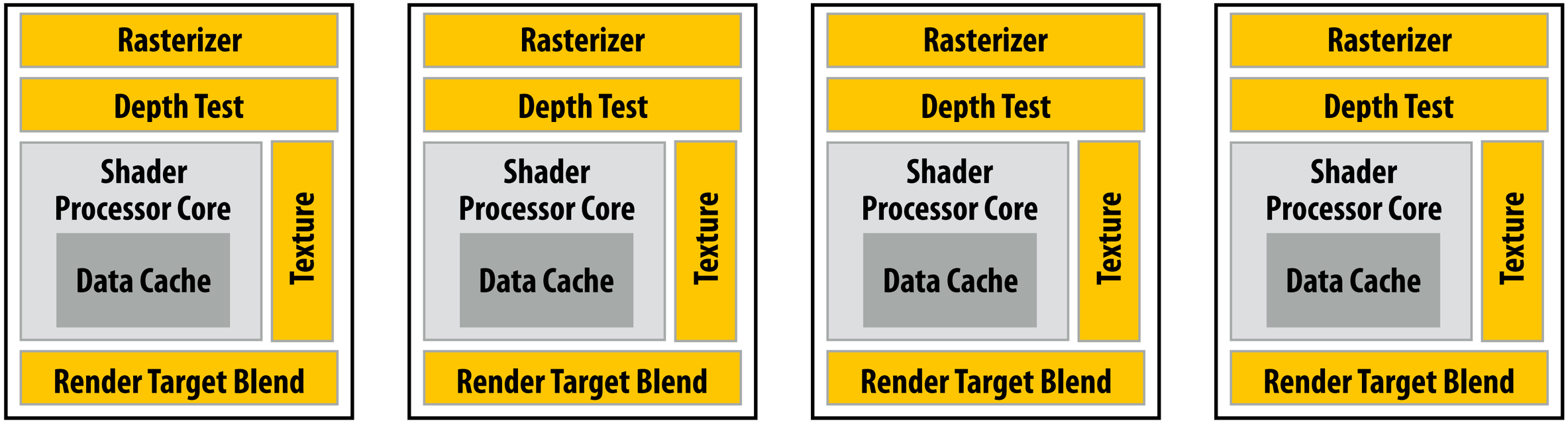
Implications

- 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.

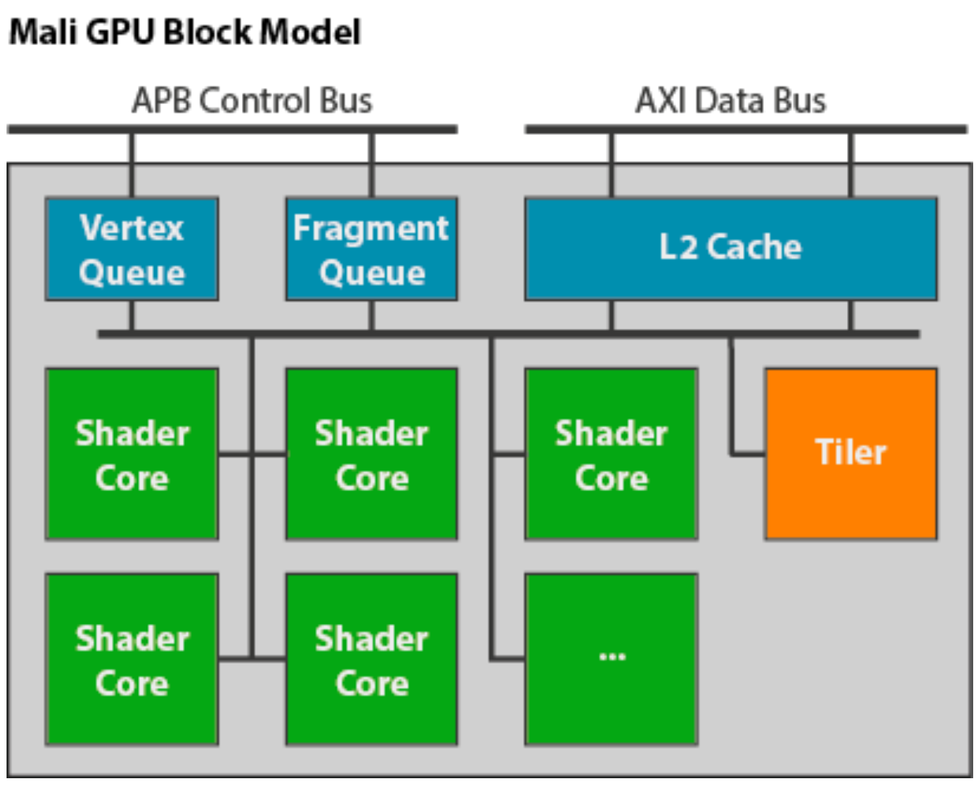
Today: a simple mobile GPU

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

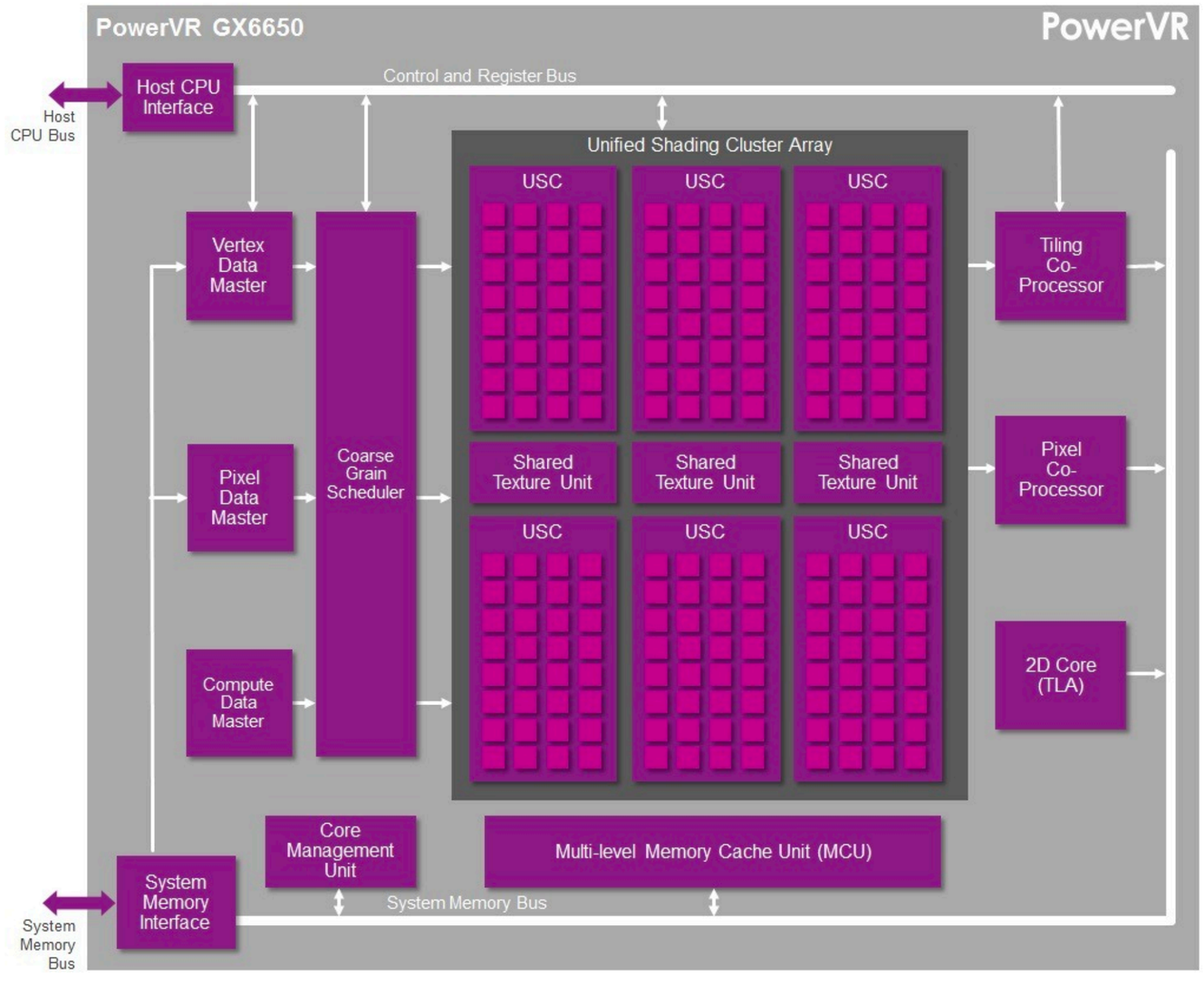


Block diagrams from vendors

ARM Mali G72MP18



Imagination PowerVR (in earlier iPhones)



Let's consider different workloads

Average triangle size

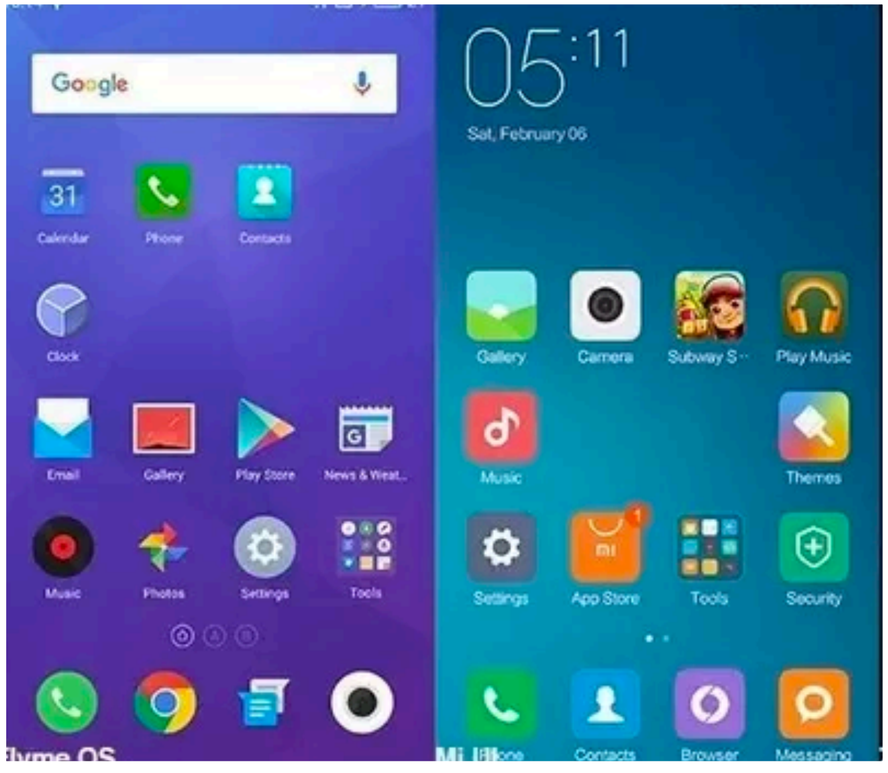
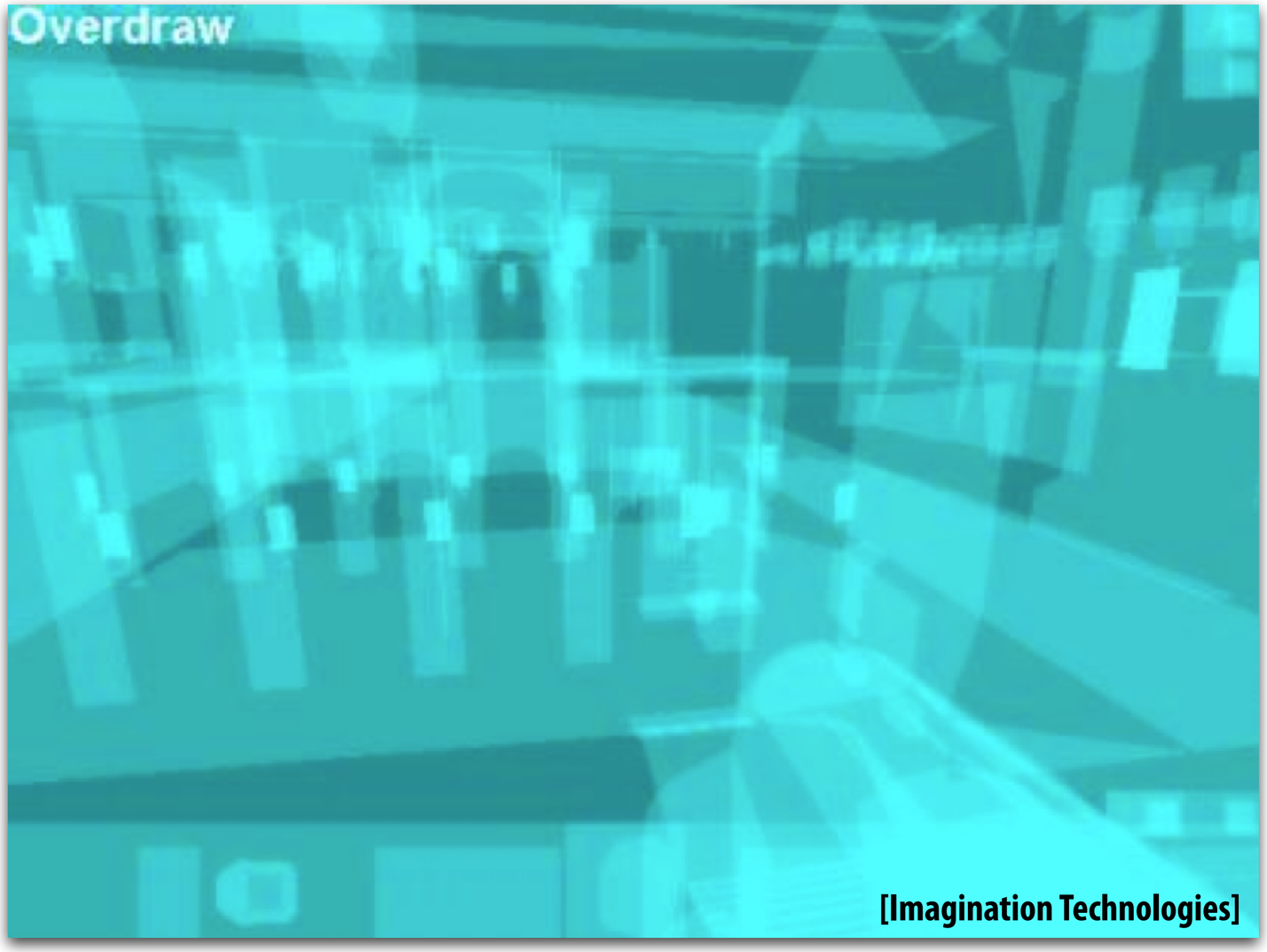


Image credit:
https://www.theverge.com/2013/11/29/5155726/next-gen-supplementary-piece
http://www.mobygames.com/game/android/ghostbusters-slime-city/screenshots/gameShotId,852293/

Let's consider different workloads

Scene depth complexity

Average number of overlapping triangles per pixel



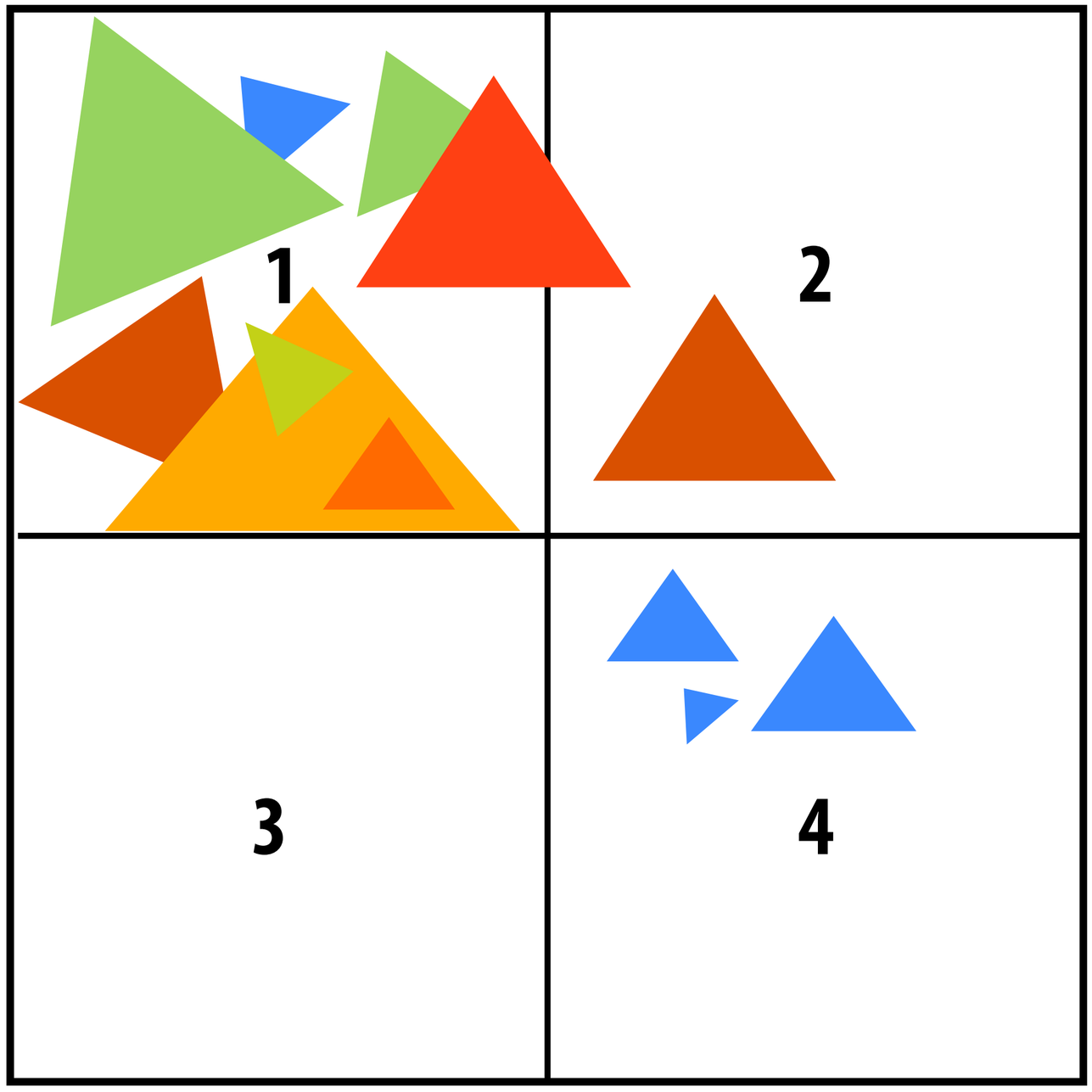
In this visualization: bright colors = more overlap

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?*

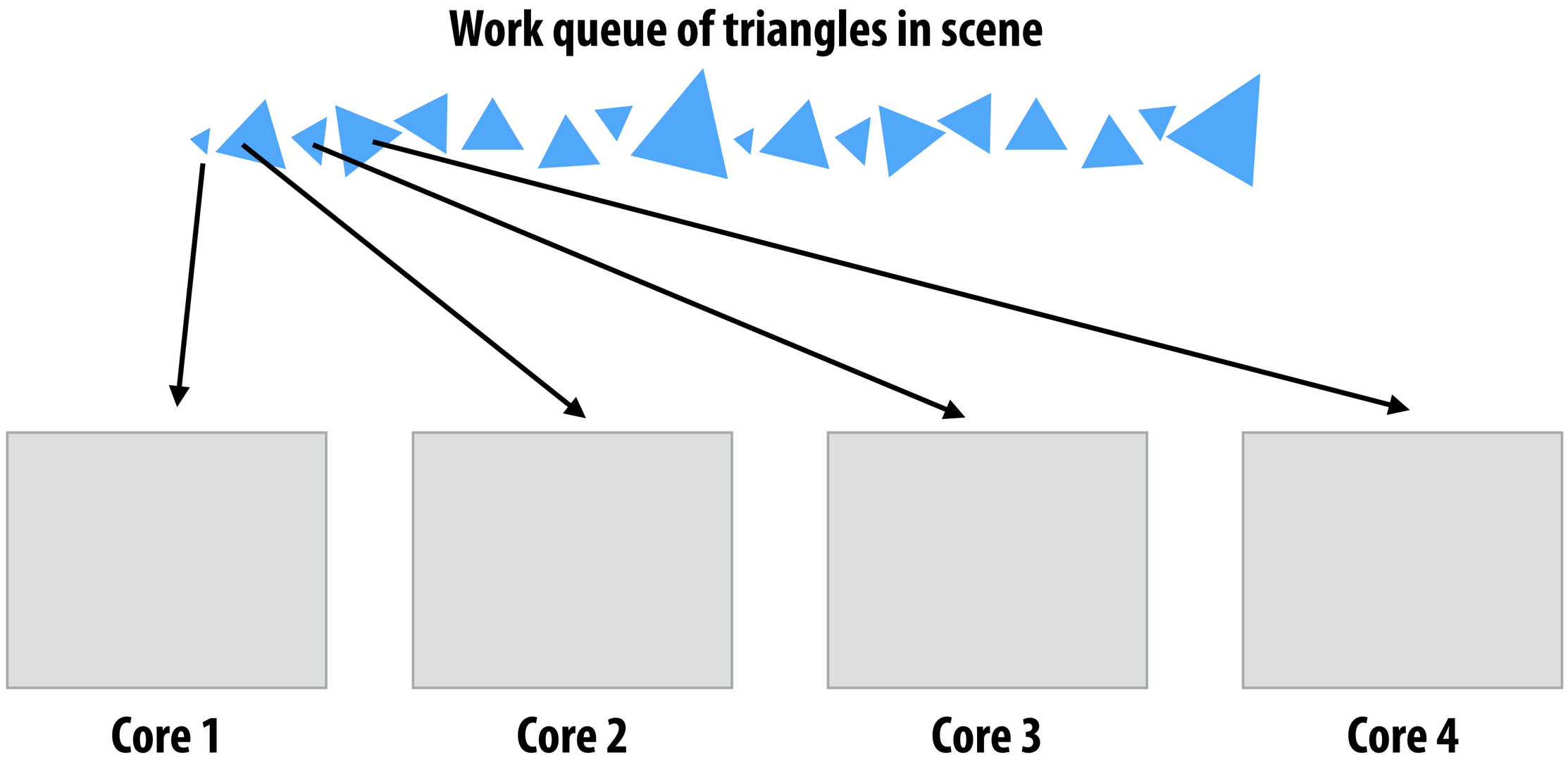
Unequal work partitioning

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



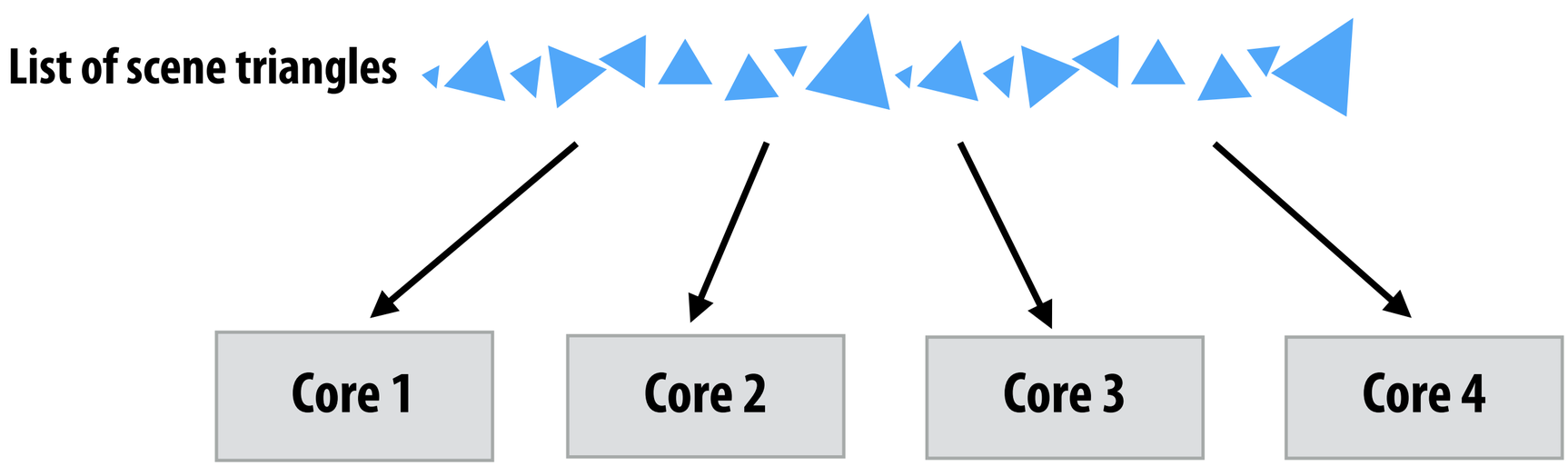
Step 1: parallel geometry processing

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



Step 1 produces list of triangle bins

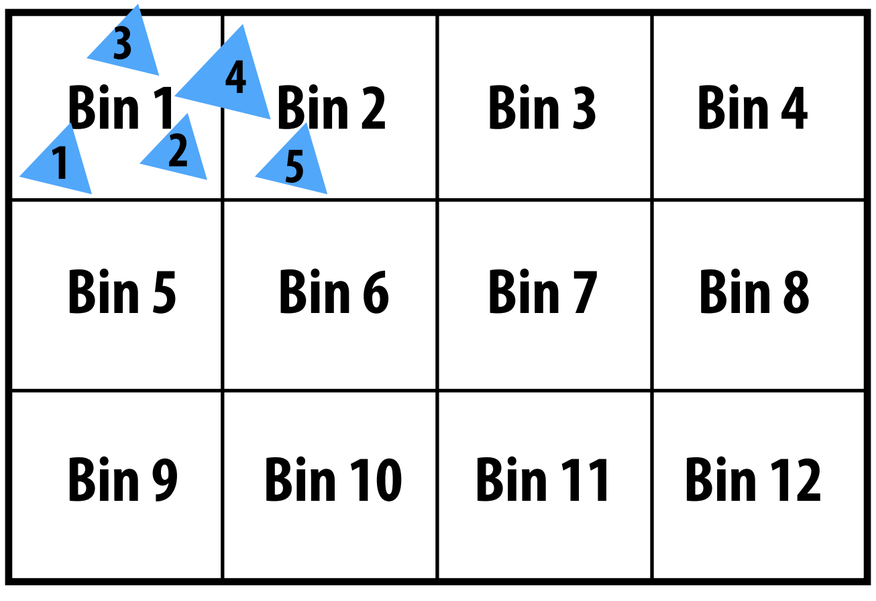
- One bin per "tile" of screen
- Core runs vertex processing, computes 2D triangle/screen-tile overlap, inserts triangle into appropriate bin(s)



After processing first five triangles:

Bin 1 list: 1,2,3,4

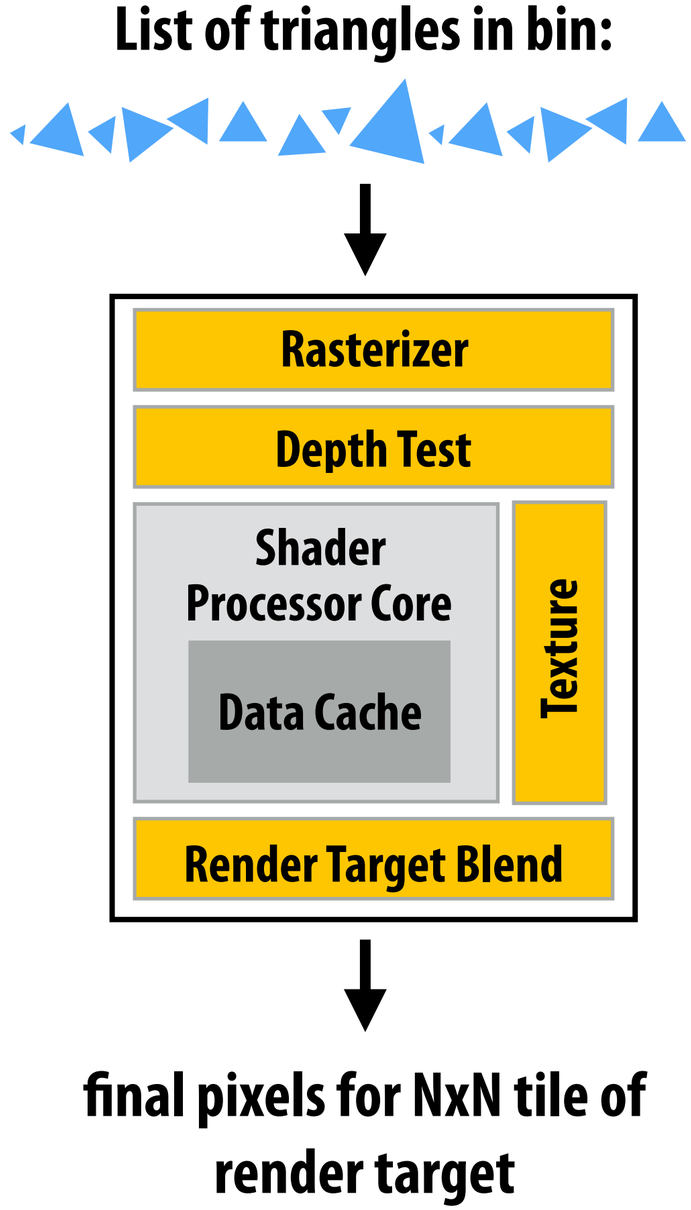
Bin 2 list: 4,5



Step 2: per-tile processing

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

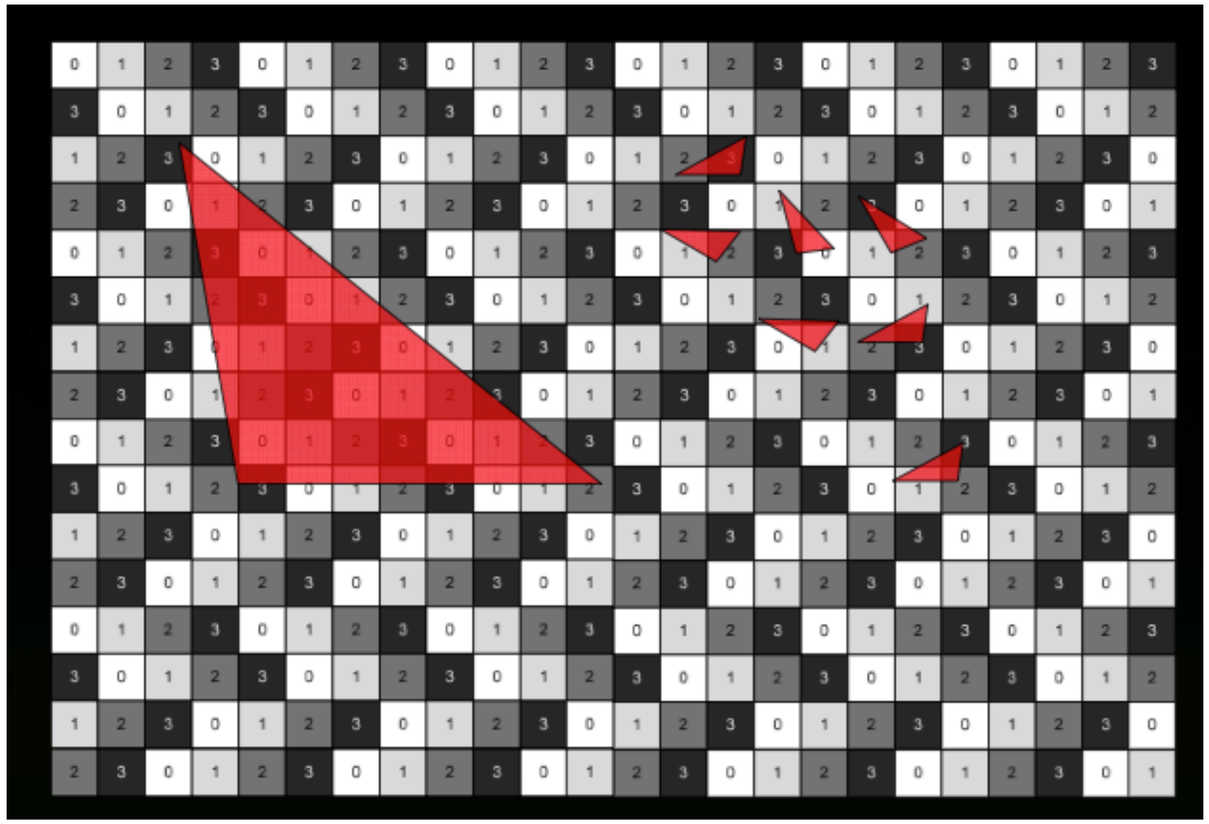
- While more bins to process:
  - Assign bin to available core
  - For all triangles:
    - Rasterize
    - Fragment shade
    - Depth test
    - Render target blend



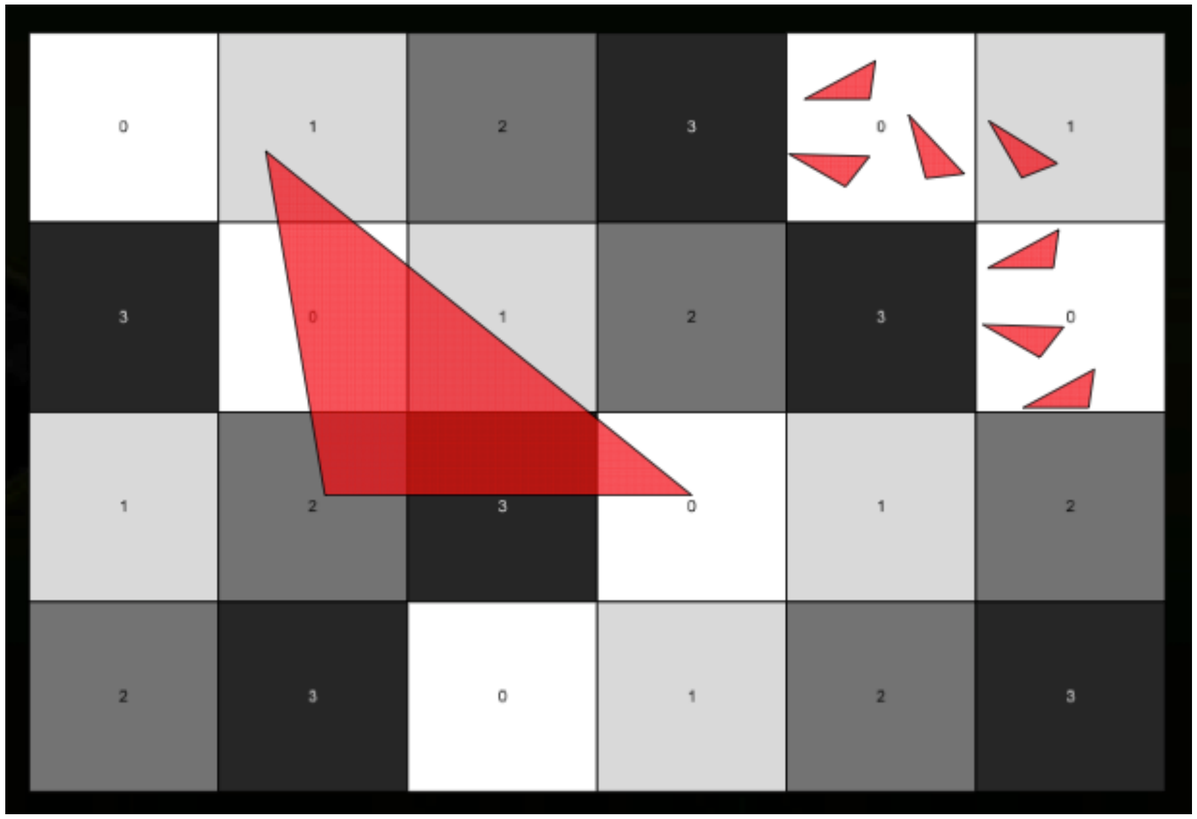
What should the size of the bins be?

What should the size of the bins be?

Fine granularity interleaving



Coarse granularity interleaving



[Image source: NVIDIA]

What size should the bins be?

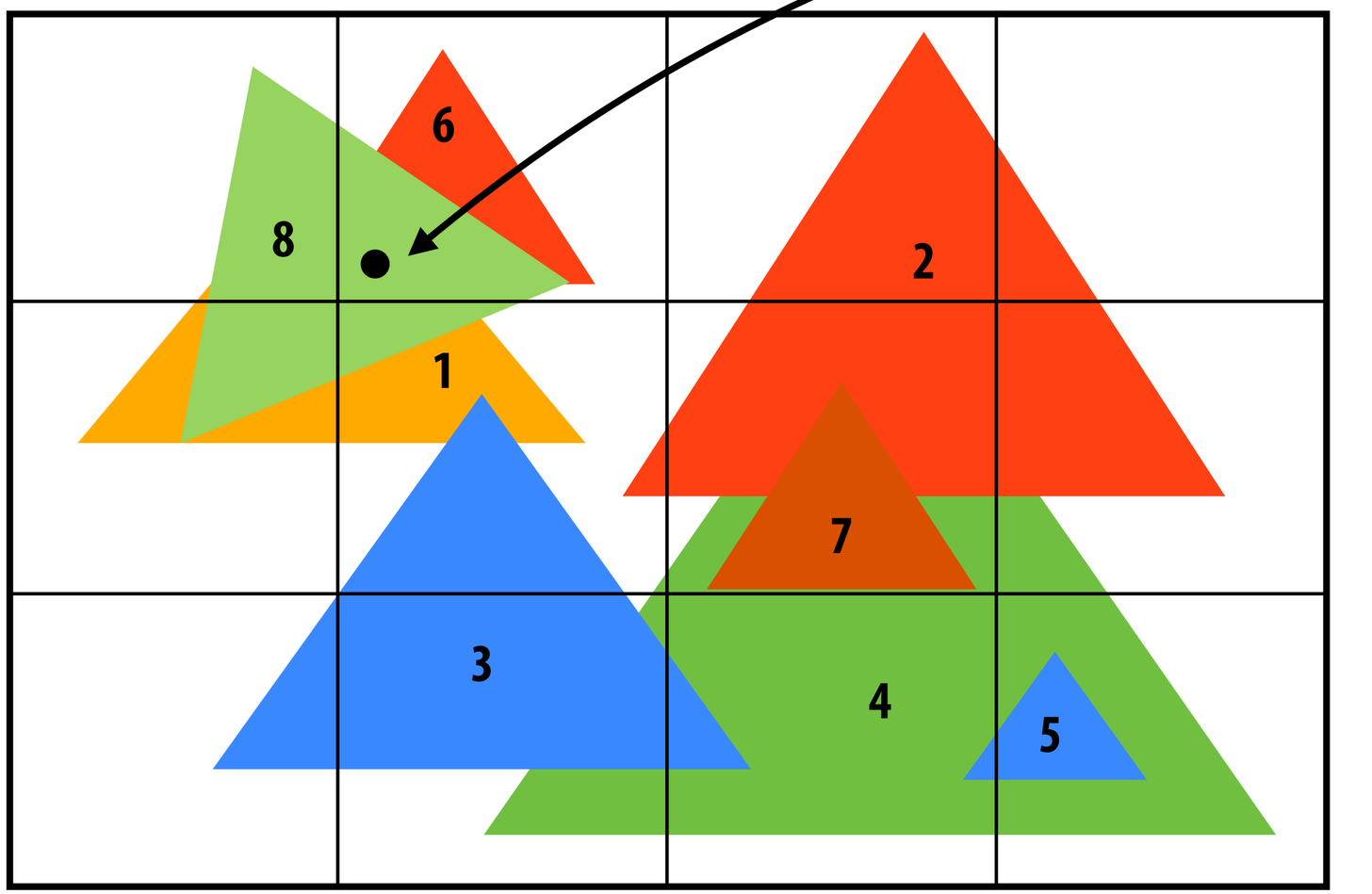
- Small enough for a tile of the color buffer and depth buffer (potentially supersampled) to fit in a shader processor core's on-chip storage (i.e., cache)

- 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?

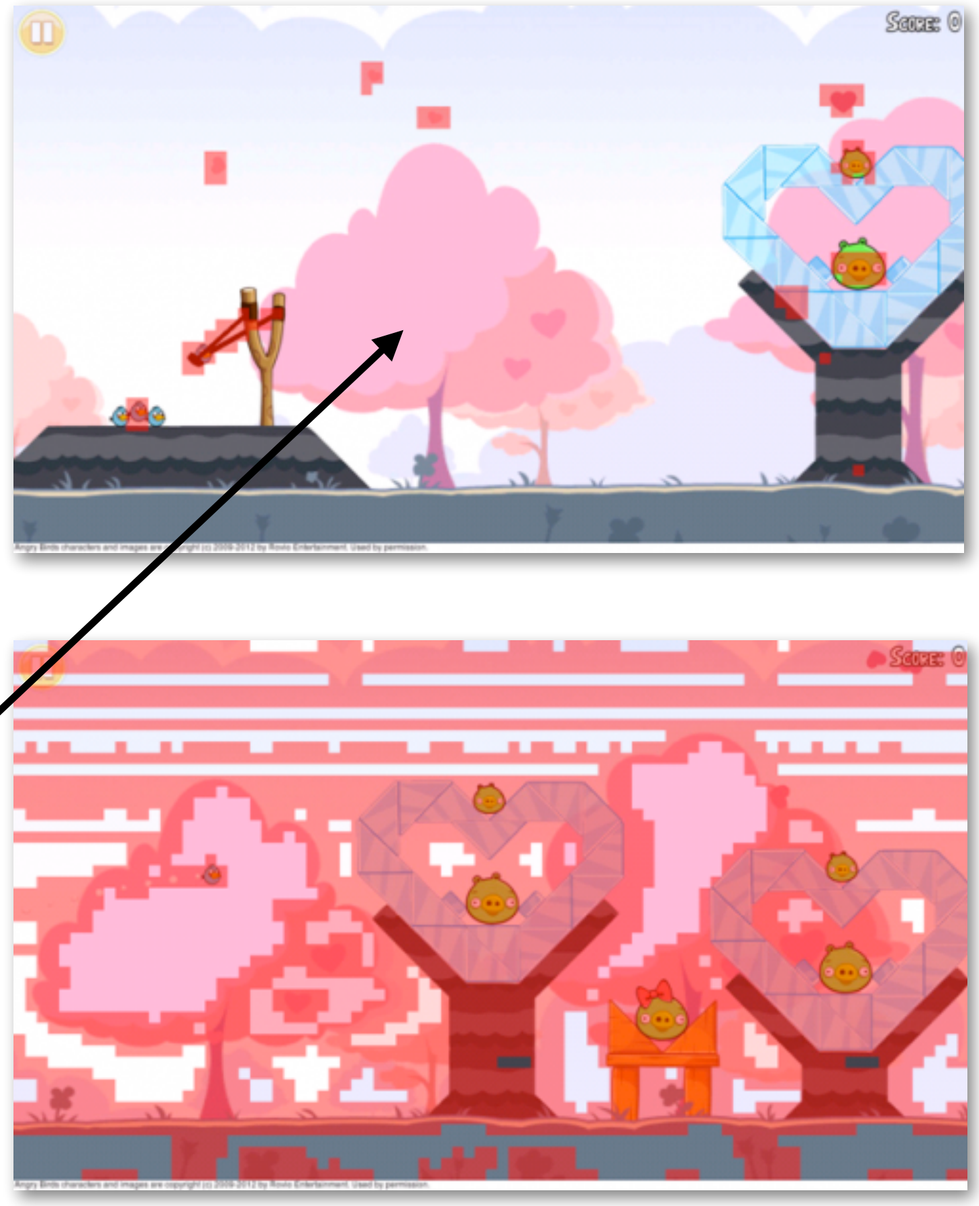
Q. Why doesn't the renderer need to write depth buffer in memory? *

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

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

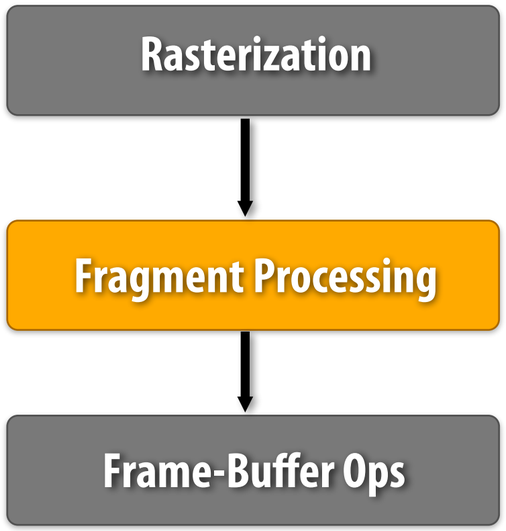


Slow camera motion: 96% of writes avoided
Fast camera motion: ~50% of writes avoided
(red tile = required a memory write)

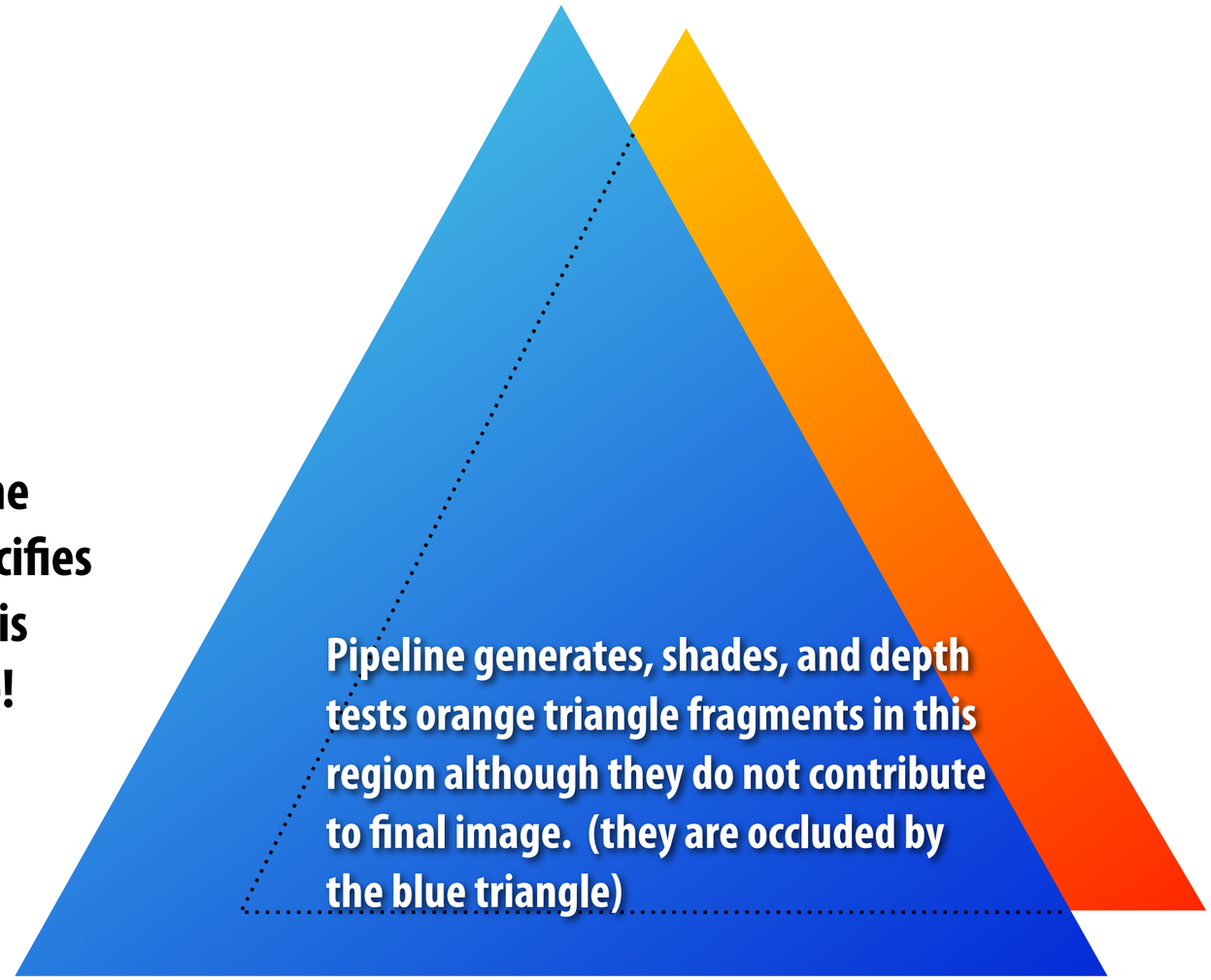
[Source: Tom Olson http://community.arm.com/groups/arm-mali-graphics/blog/2012/08/17/how-low-can-you-go-building-low-power-low-bandwidth-arm-mali-gpus]

Minimizing computation performed

Depth testing as we've described it

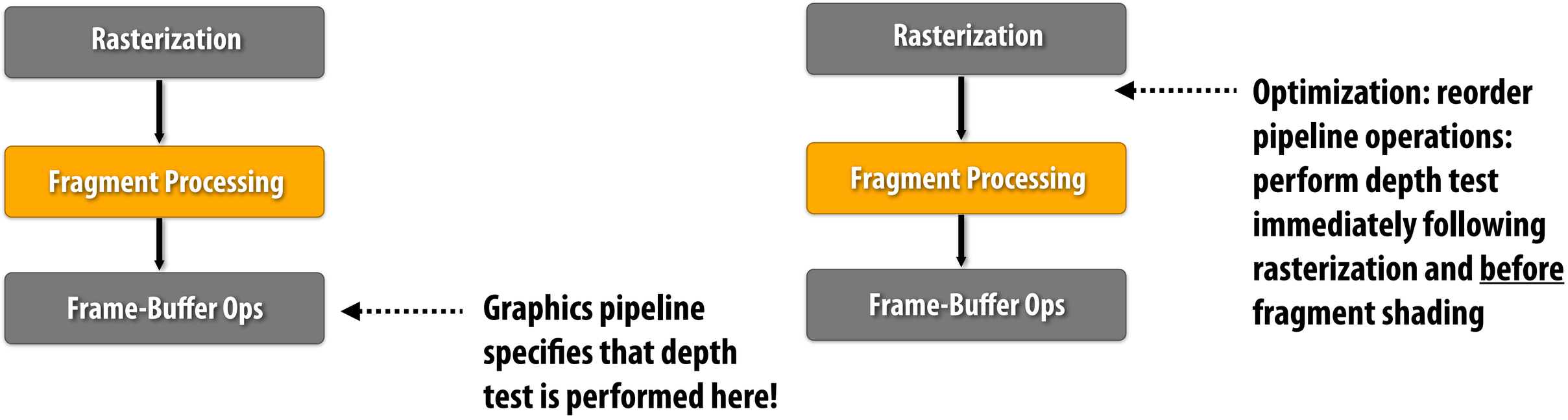


Graphics pipeline abstraction specifies that depth test is performed here!



Early Z culling

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

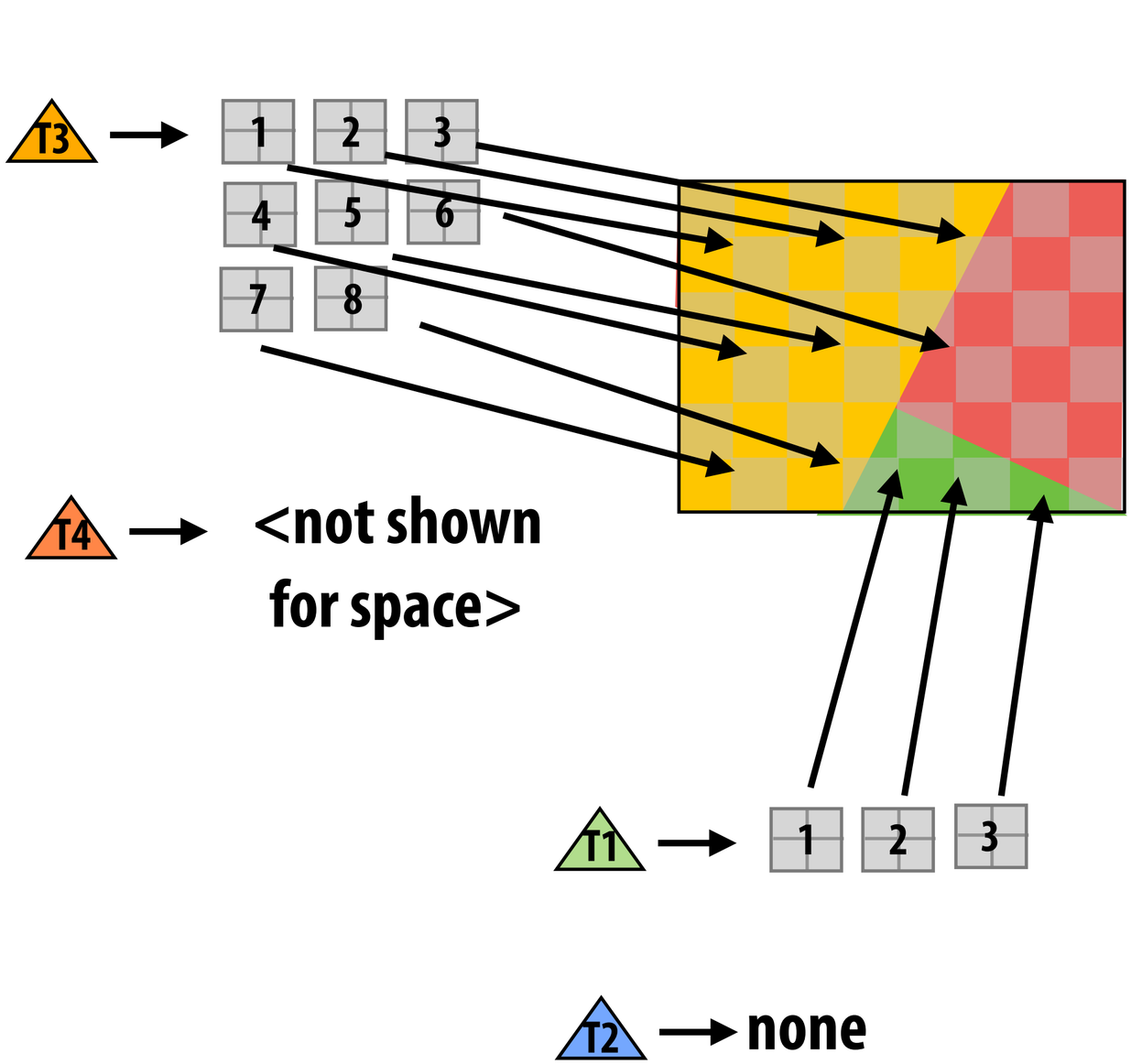


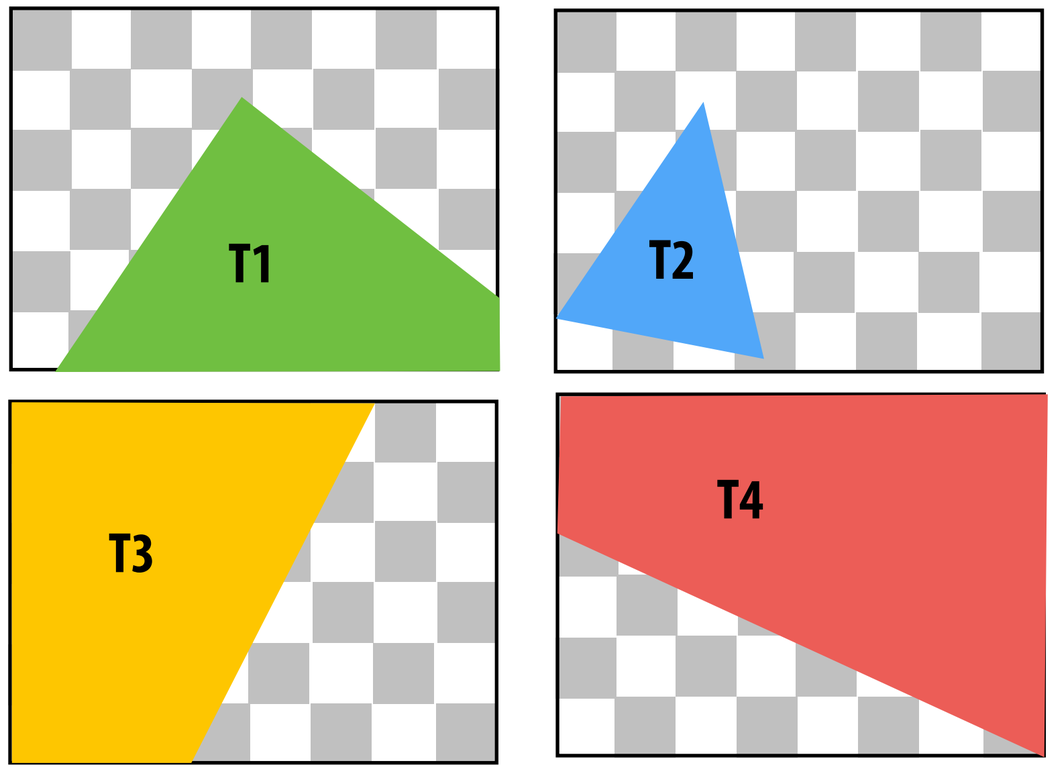
Key assumption: occlusion results do not depend on fragment shading

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

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





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.

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

Graphics today

Computer graphics expands beyond the "nuts and bolts" ideas of 2D / 3D drawing and mesh representation we've discussed in class (and is useful well beyond games and movies!)

Computational photography

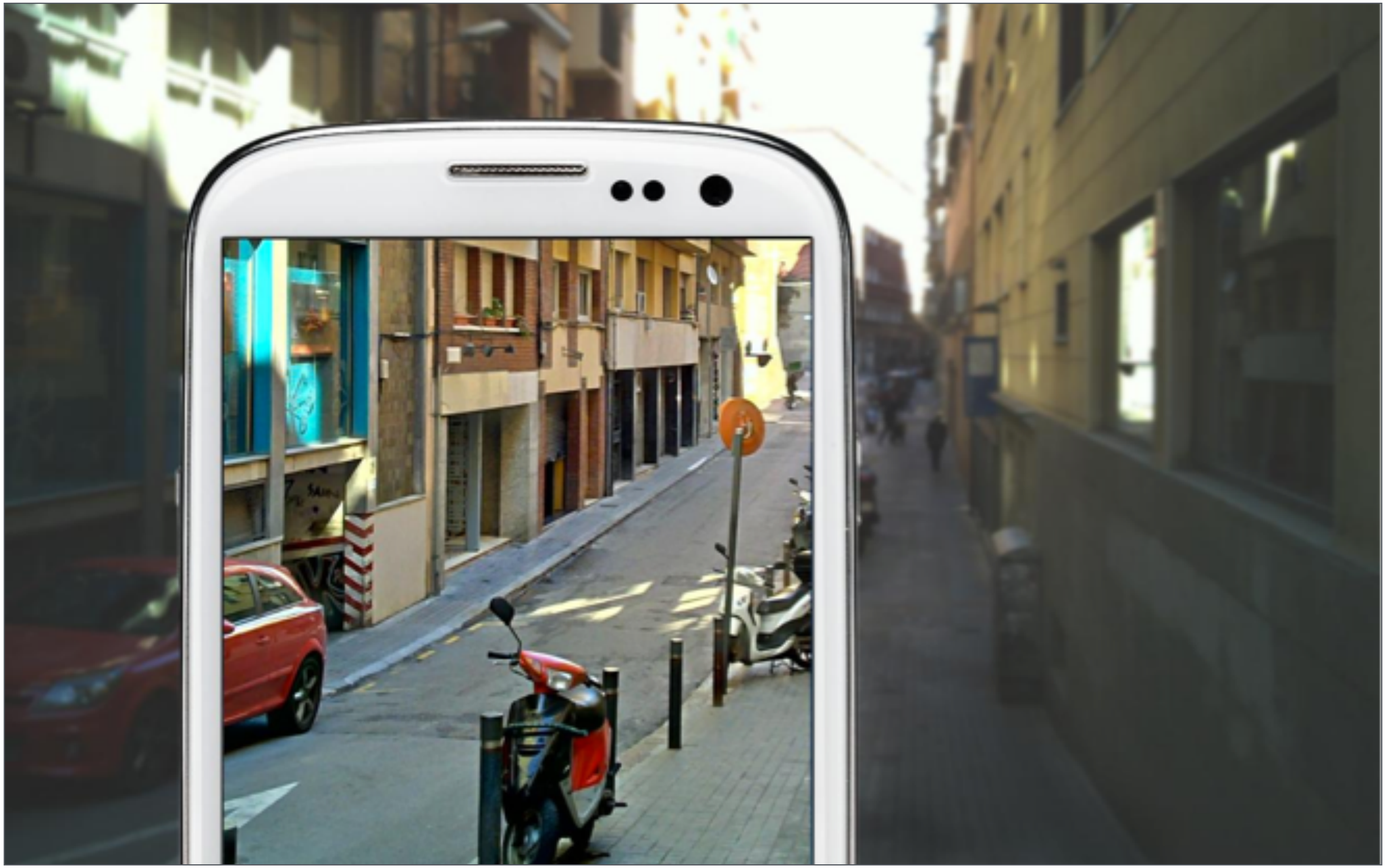
- Using computation (and increasingly machine learning) to make more aesthetic photographs, simulate behavior of more complex lenses, etc.

Google Pixel 2 Portrait mode

Image credit: Google / Matt Jones (https://ai.googleblog.com/2017/10/portrait-mode-on-pixel-2-and-pixel-2-xl.html)

Computational photography

- Using computation (and increasingly machine learning) to make more aesthetic photographs, simulate behavior of more complex lenses, etc.

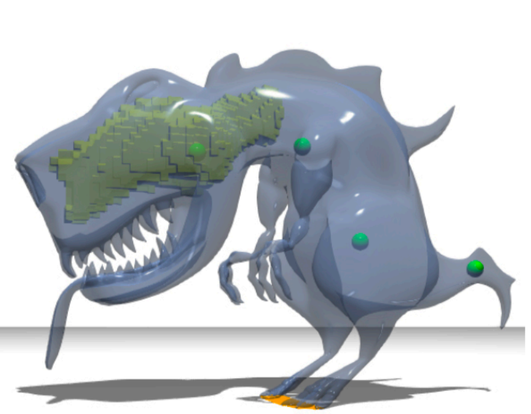
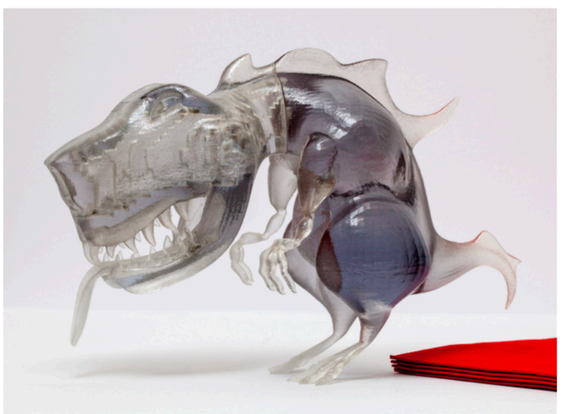


High Dynamic Range Imaging (HDR)

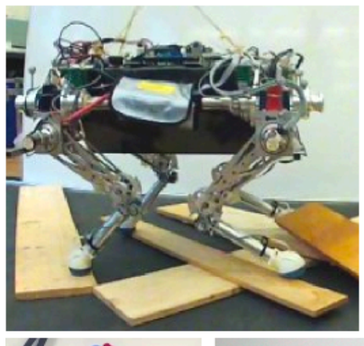
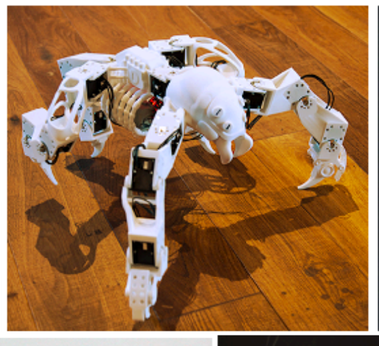
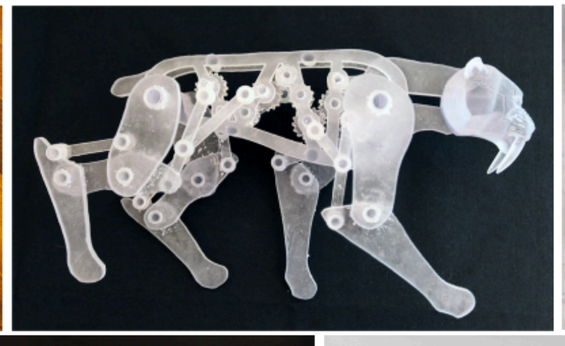
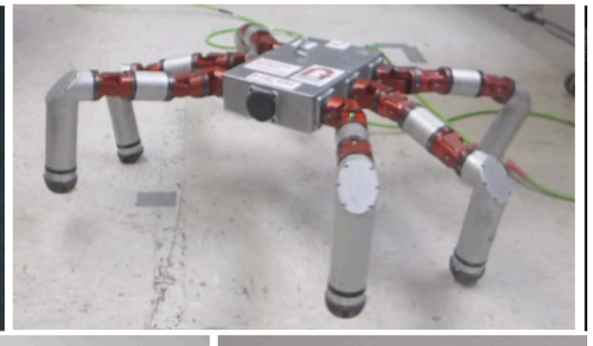
Creating physically plausible models

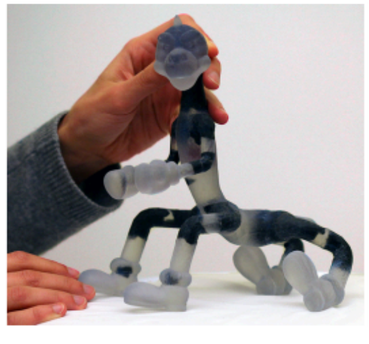
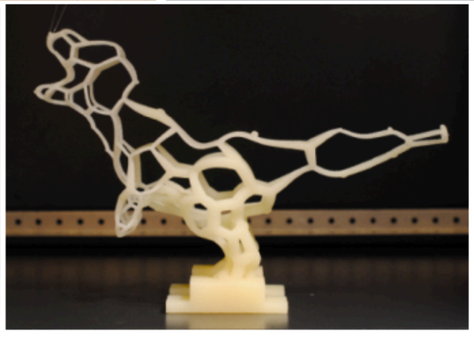
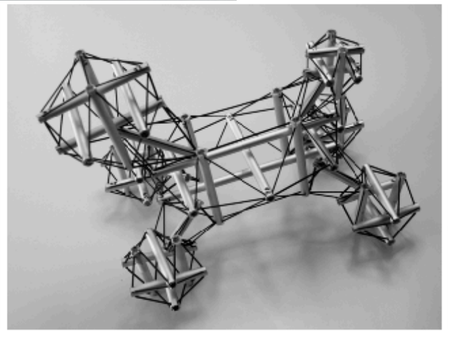
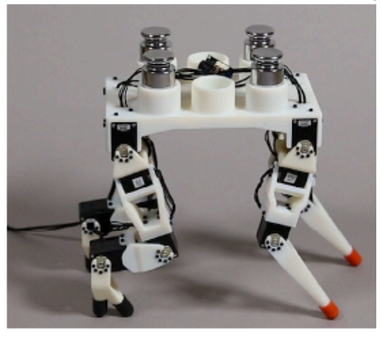
- Via 3D printing, fabrication
- Creatures that locomotes, furniture that stands, etc.



Fabricate models that are balanced to stand

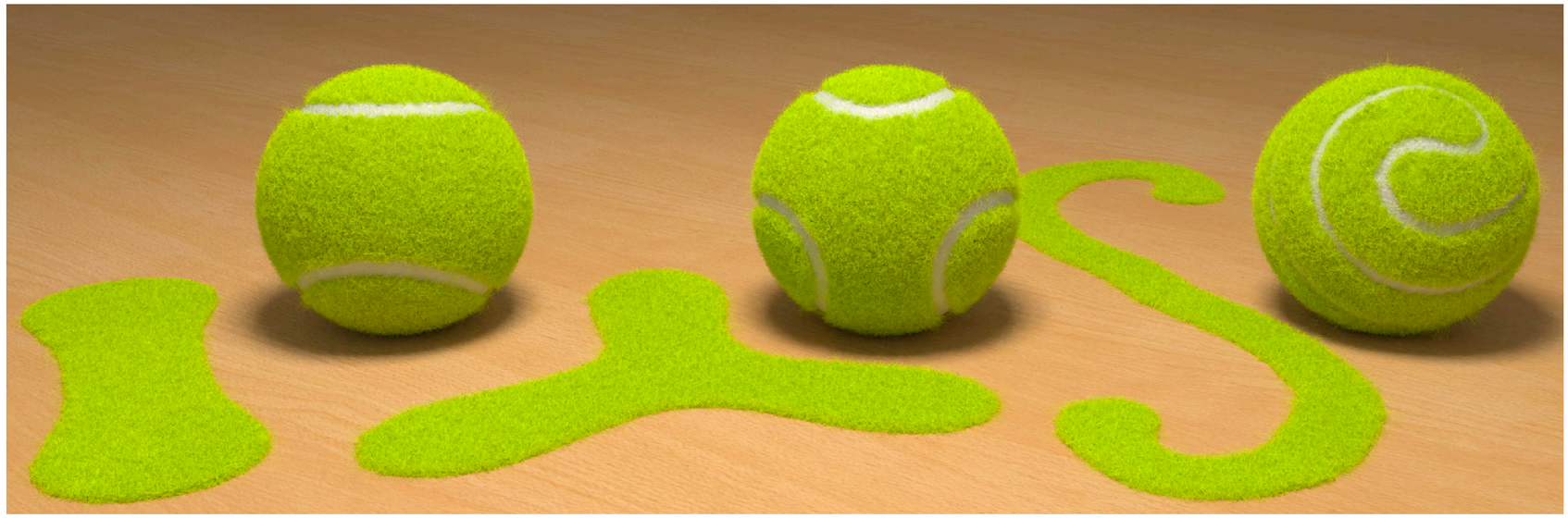





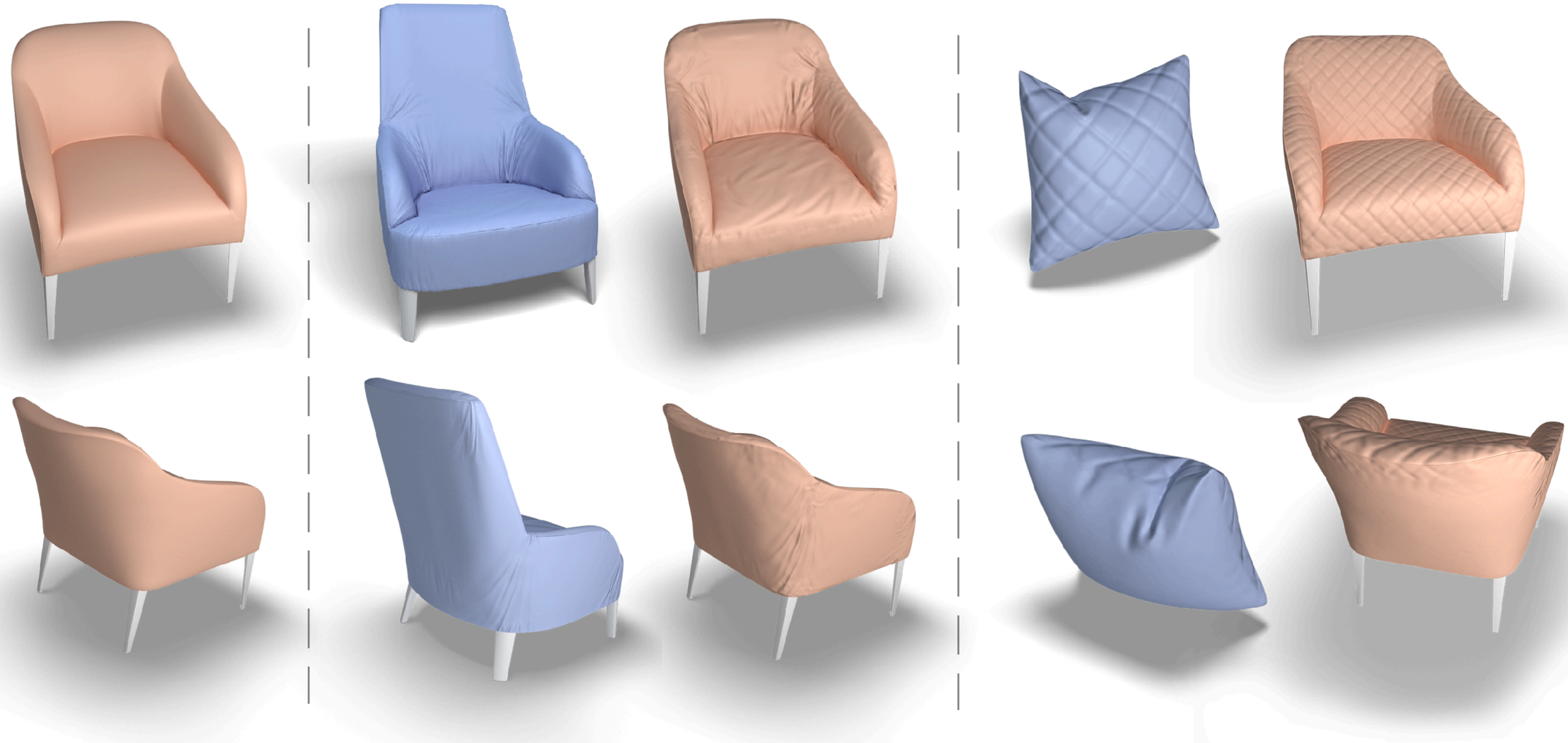





Fabricate robots that can balance and move

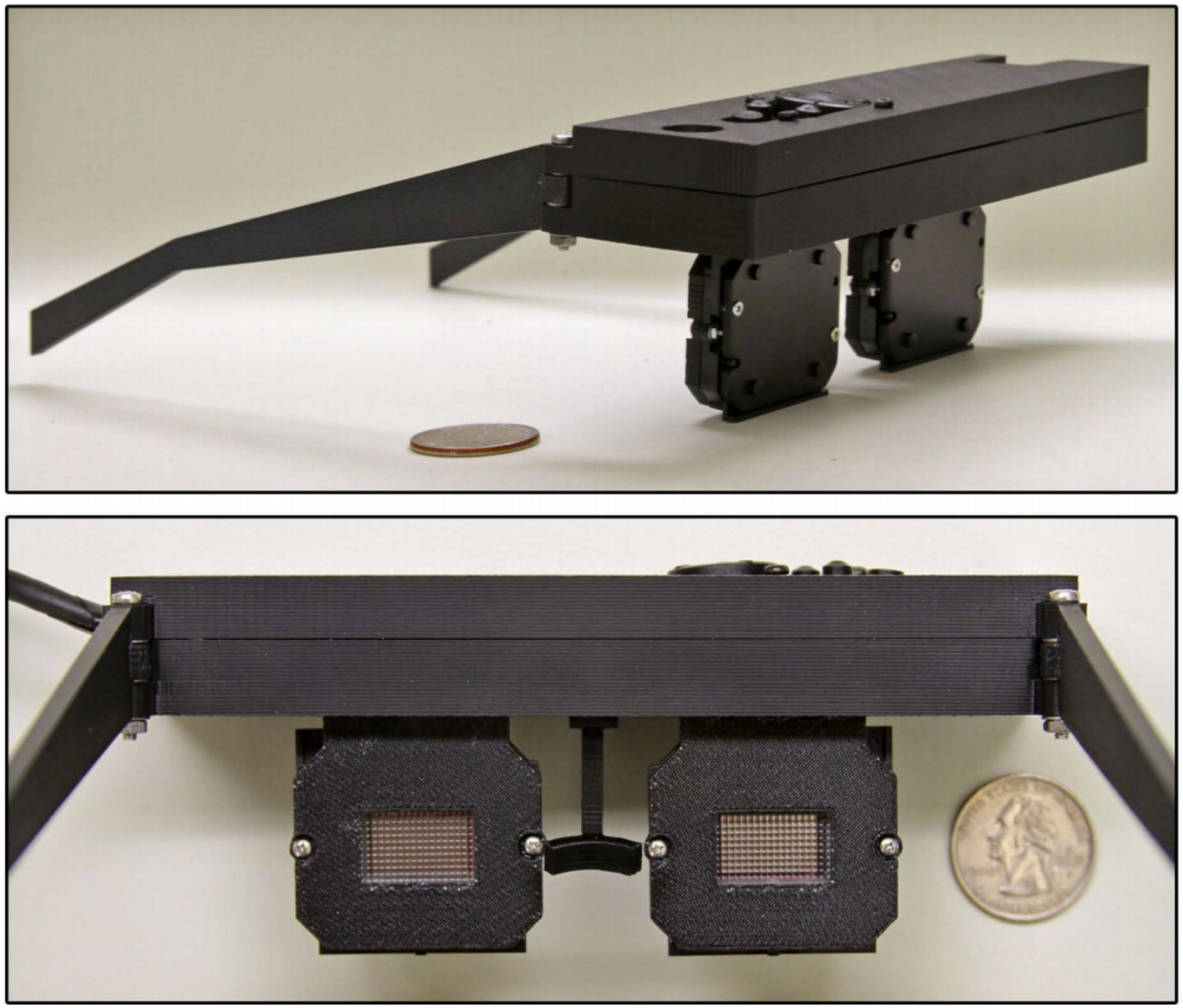
Advanced geometry processing

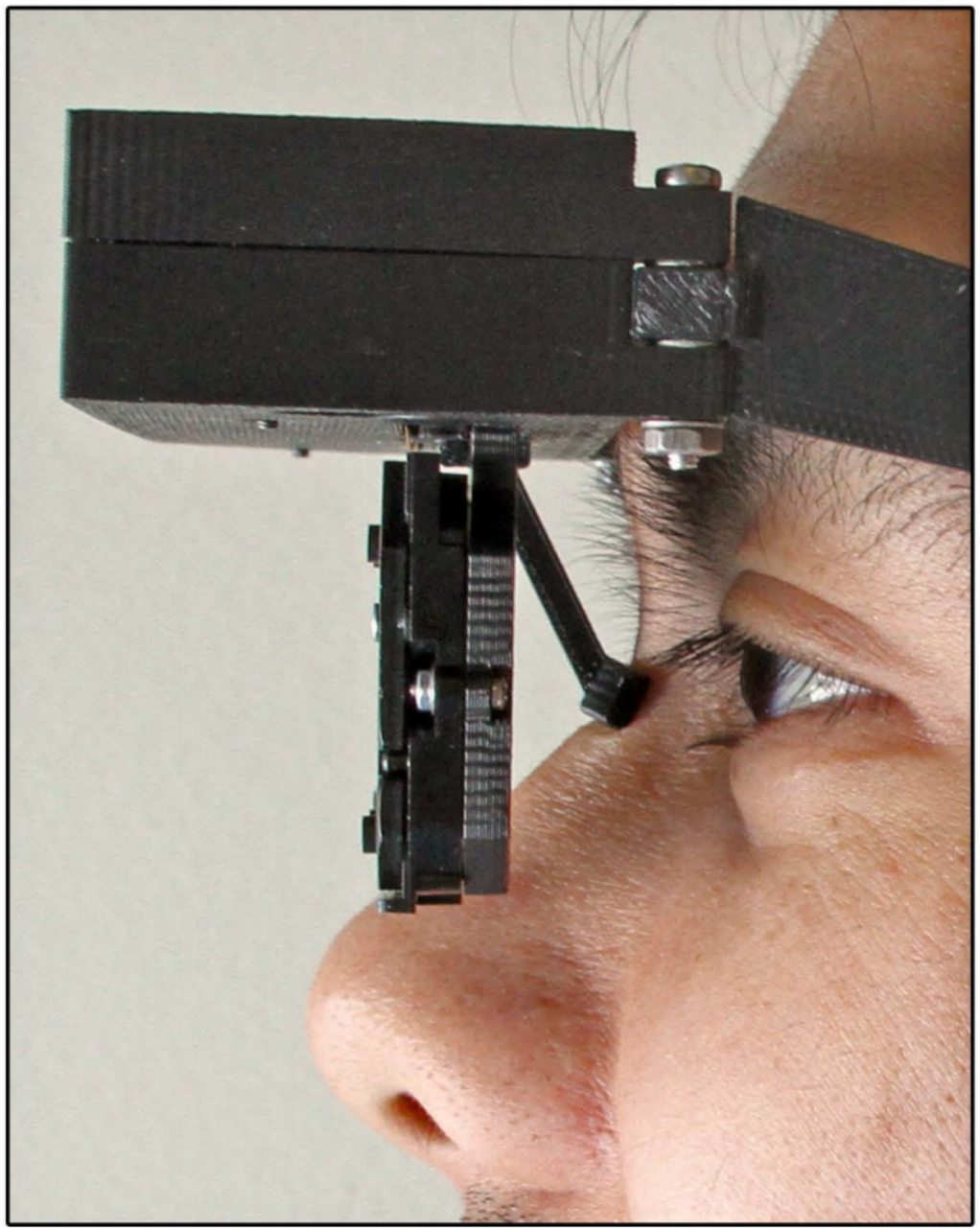
Fundamental questions about alignment, similarly, symmetry, etc…





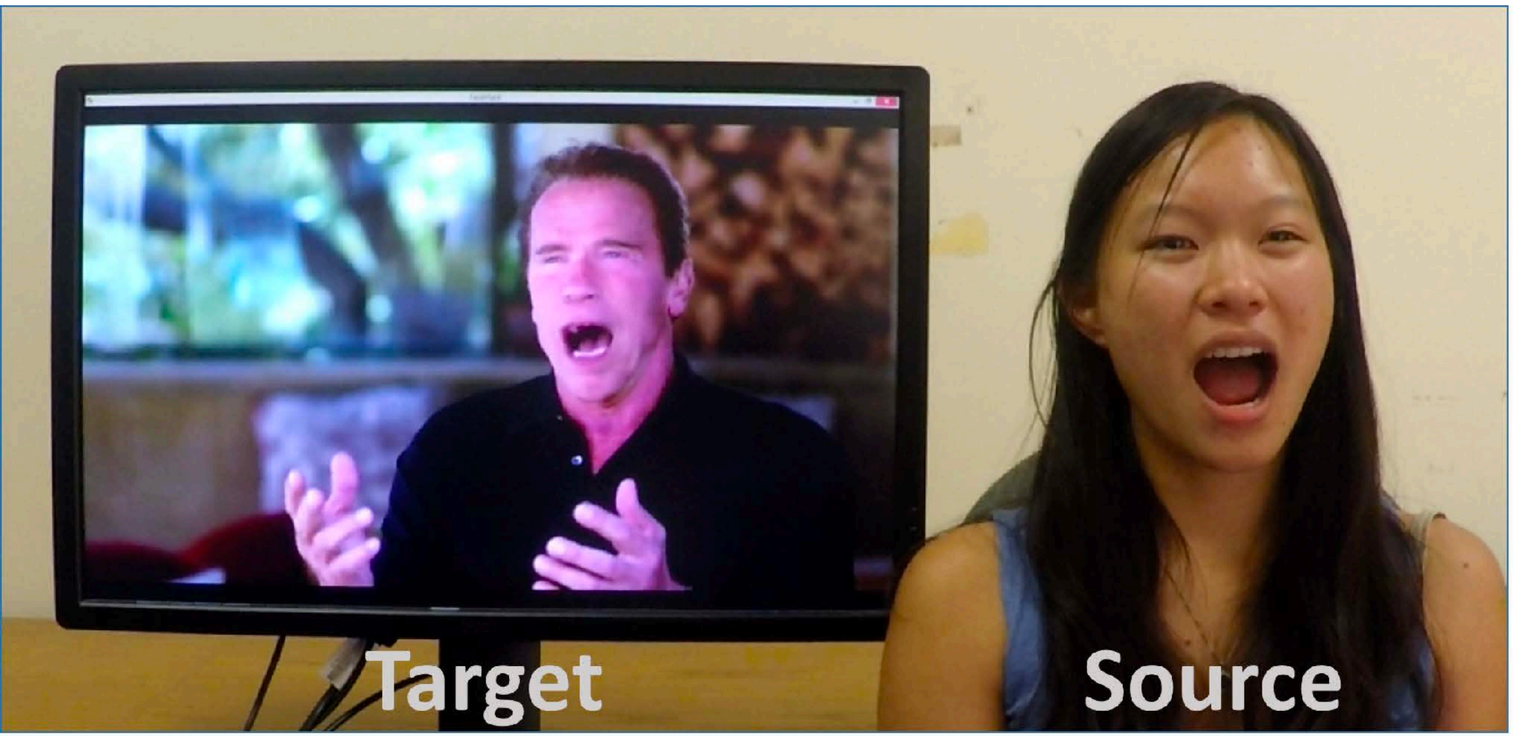
Advanced displays/rendering for VR/AR

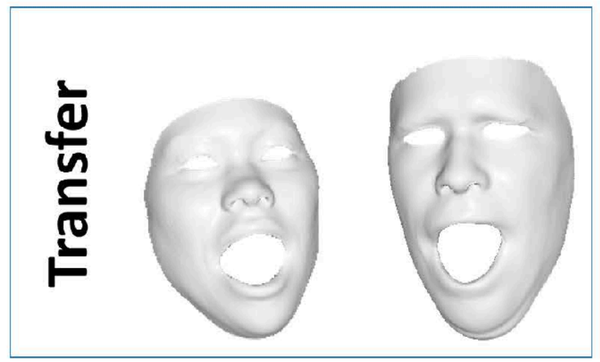


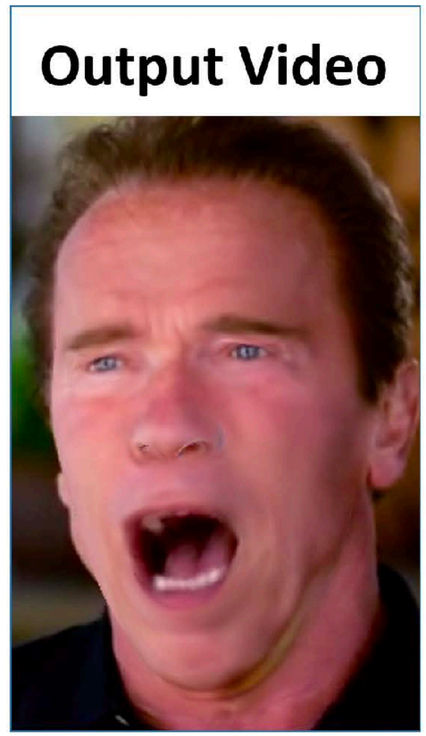


Near eye light field display

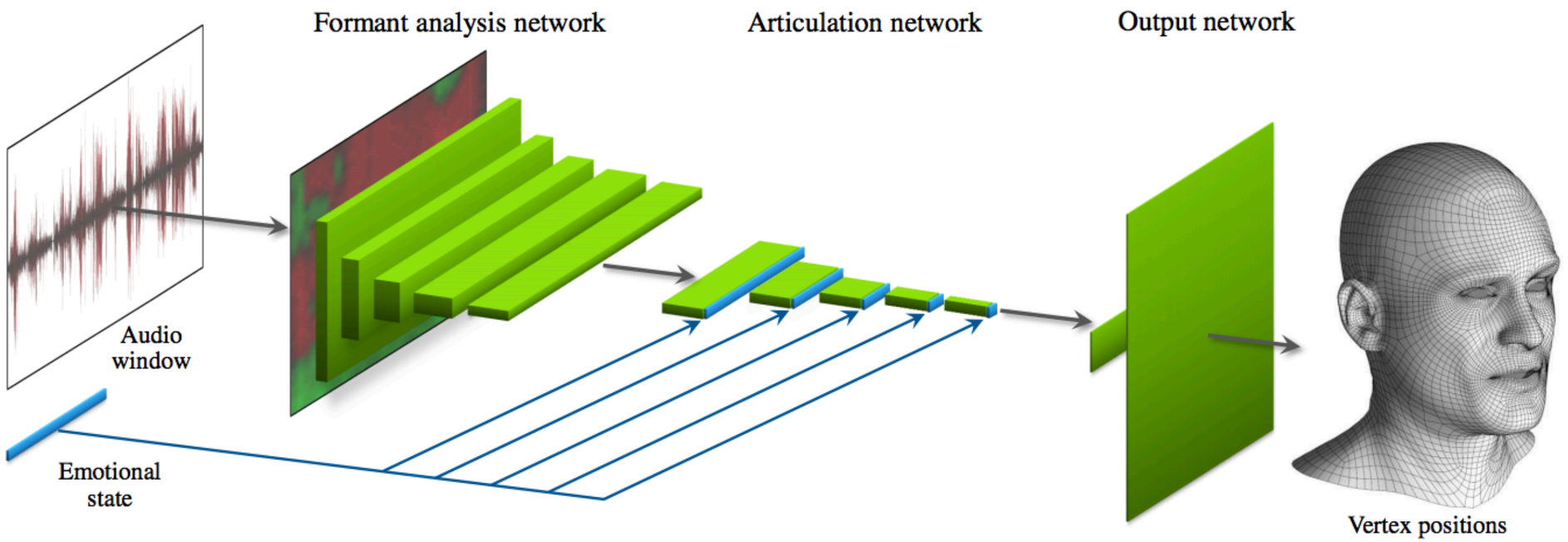
Content creation and capture







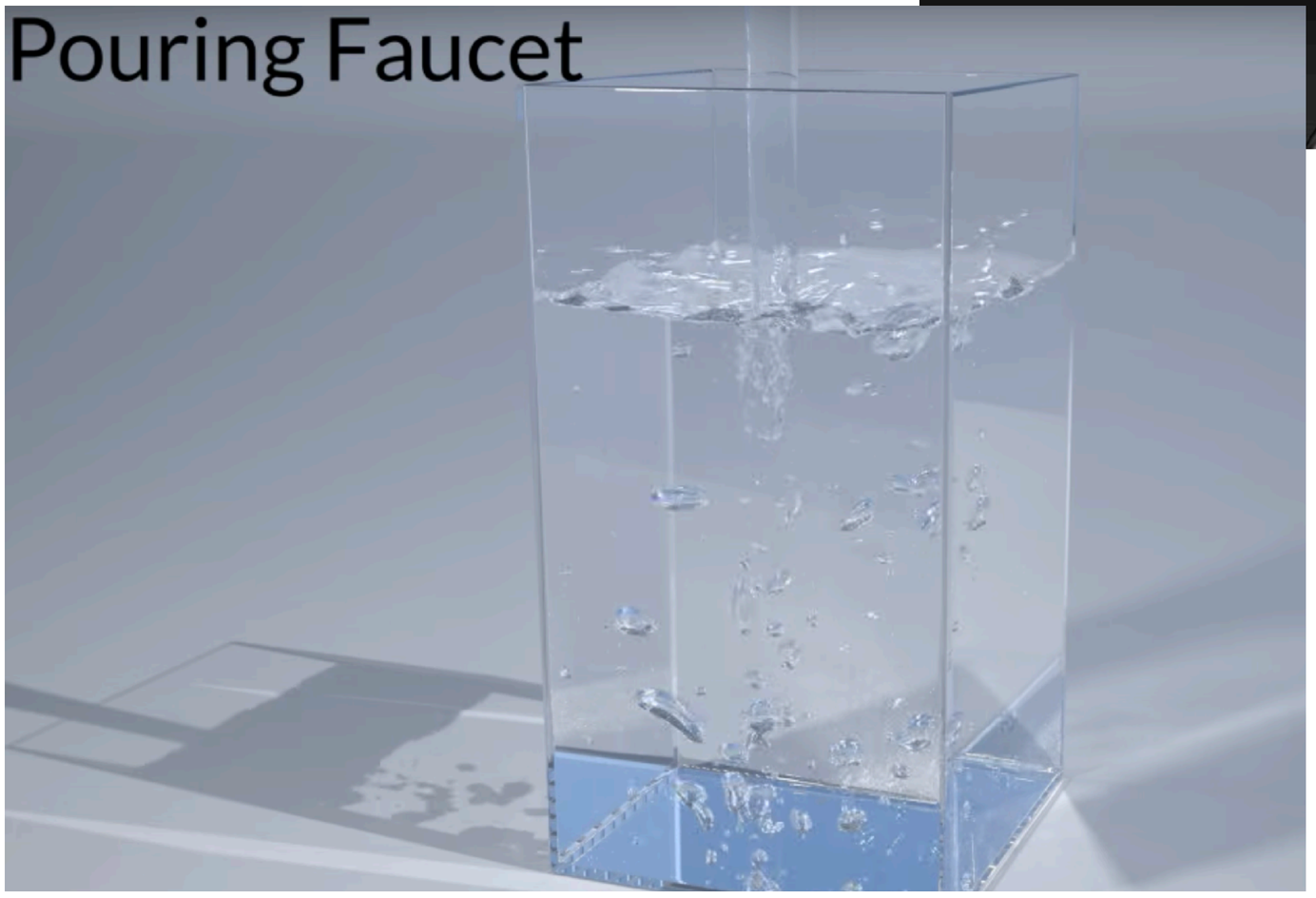
Manipulating actors by performance capture



Audio input to mesh animation

Synthesizing/simulating sound

- Simulating sound of scenes, not just their appearance



More graphics classes at Stanford

- Image Synthesis Techniques, (CS348B, Spring), theory and practice of advanced ray tracing (Hanrahan)
- Visual Computing Systems, (CS348K, Fall), parallel systems design for computational photographic, deep learning, 3D graphics (Fatahalian)
- Animation and Simulation (CS348C, Winter), deep dive into animation and simulation techniques (James)
- Computer Game Design (CS146, Fall), make your own games in Unity (James)
- Virtual Reality (EE267, Spring), focuses on display, tracking hardware for VR (Wetzstein)
- Computational Imaging and Display (EE367/CS448i, Winter), advanced course on display design (Wetzstein)

Thanks for being a great class!

Good luck on projects! Make sure you have fun, that's the point!

See you on the 9th!