Graphics, Mobile Computing, APIs and Life

[Dave Shreiner](#)

Director, Graphics and GPU Computing

ARM, Inc.

12 November 2012

Agenda

- ARM and the IP Business
- That Computer in your Pocket
- Graphics: Techniques and Religion
- APIs – using them, designing them, and living to tell about it

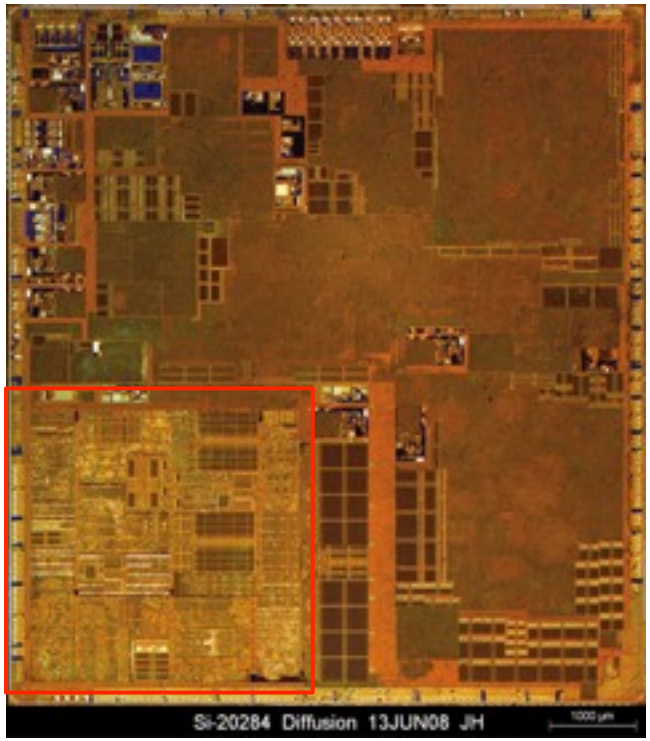
Who is ARM, and what do we do?

- ARM creates designs (*IP*) for digital processors
  - CPUs
  - GPUs
  - Bus fabric
  - Memory controllers
  - Physical IP (memories, caches, etc.)

Cortex
Low-Power Leadership from ARM

mali

- We don't make anything physical
  - Our customers use our designs to make System-on-a-Chip (SoC)
  - Most dies are a mix of IP from various places
    - Multiple vendors
    - In-house designs ("magic sauce")

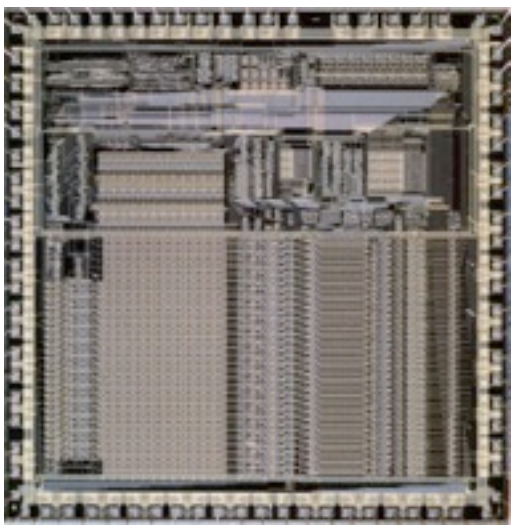


Die shot of TI OMAP 3530

The First ARM Silicon: 26th April 1985

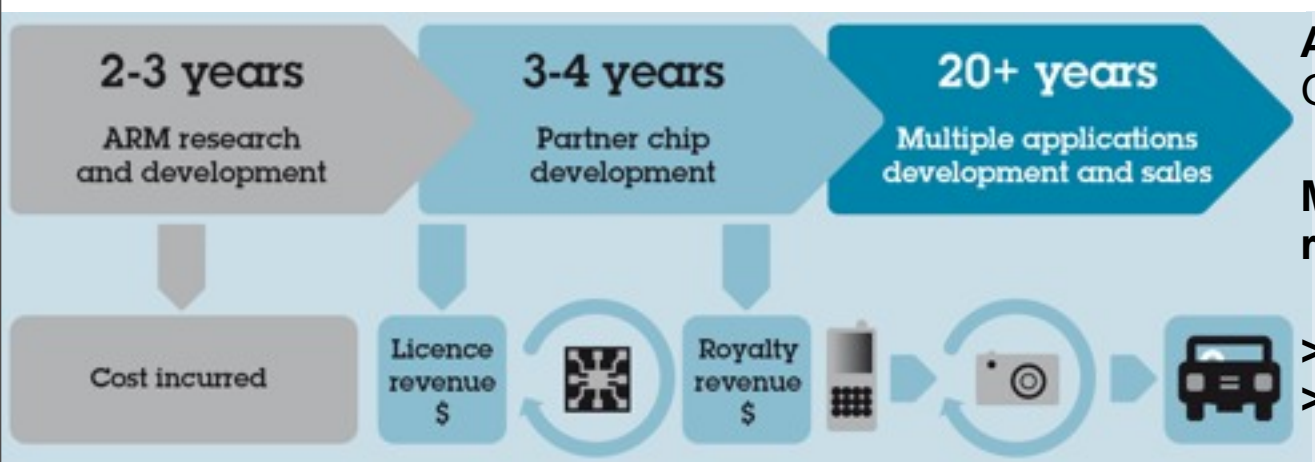
- Acorn Computer needed better CPU performance
  - Evaluated processors from several companies
  - All a bit slow and much too expensive
- 1985: Called simply the “ARM” (Acorn RISC Machine)
  - 25,000 transistors, 3.0µm, 6MHz, 120mW, 50mm^2
- Steve Furber designed the hardware to implement the instruction set developed by Sophie Wilson

“You can’t build a £500 micro around a £100 CPU”
Steve Furber



The ARM Business Model

- Global leader in the development of semiconductor IP
  - R&D outsourcing for semiconductor companies
- Innovative business model yields high margins
  - Upfront license fee – flexible licensing models
  - Ongoing royalties – typically based on a percentage of chip price
  - Technology reused across multiple applications
- Long-term, secular growth markets



Approximately 750 licenses
Grows by 60-90 every year

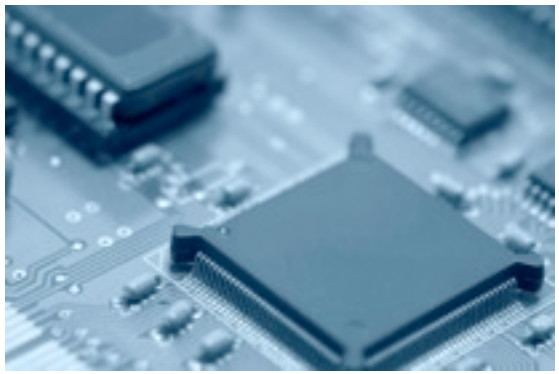
More than 250 potential royalty payers

>6bn ARM-based chips in ’10
>25% CAGR over last 5 years

Graphics in an Embedded/Mobile World

- Life's different when you're connected to a battery
  - People have … expectations
- Mobile vs. Embedded
  - mobile usually implies limited battery-powered
    - mobile phone, MP3 player, GPS
  - embedded may be "tethered", or have more substantial batteries
    - automotive, set-top box, etc.
- Our design space axes
  - Power
  - Performance
  - Silicon-die Area

Desktop vs. Embedded Footprint



- Graphics card – 250W
  - System Power – 600+ W
- Graphics Clock – 772MHz
- 16-way PCIe 2.0 (192.4 GB/s)
- On-board memory – 1.5 GB GDDR5
- 3 Billion gates @ 40 nm

- SoC power < 1W (perhaps 3 for tablets)
- Graphics Clock – 533 MHz
  - Shared memory bus ~ 10GB/s
  - Unified memory architecture
  - On-core memories ~500Kb

Power

- The *overriding* design concern
- Anything an SoC does consumes energy
  - but not all operations cost the same
- Classifying operations
  - On-die are usually lowest power
    - the one place an IP-vendor has some control
  - On-chip are still manageable
    - e.g., snooping caches
  - Off-chip operations are most costly
    - particularly accessing RAM (“main memory”)
- Bandwidth = Power

Performance

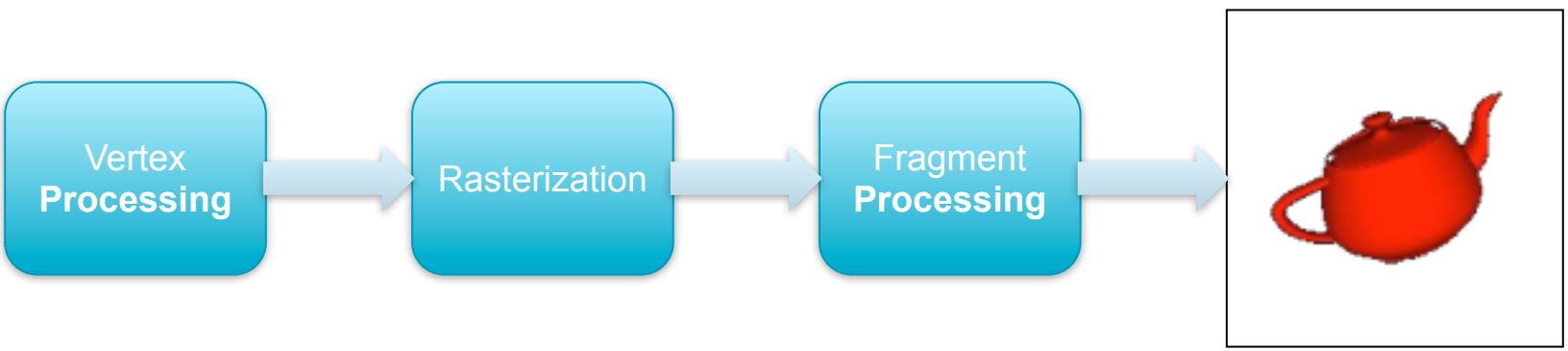
- Measured (badly) in the same way as any other GPU
  - “Speeds and feeds”
    - Triangles/s
    - Pixels/s
    - Texels/s
  - Industry benchmarks
    - Futuremark’s Benchmarks
      - Samuari & Proxycon – OpenGL ES 1.1
      - Taiji Girl & Hoverjet – OpenGL ES 2.0
    - GLbenchmark
      - GLB 2.5 Egypt
- And they always ask about “Bandwidth”
  - It’s like asking how long it takes to drive …

Die Area

- Each square millimeter of ASIC silicon costs about 10¢
  - pretty much regardless of manufacturing processing
- Factors that contribute to area
  - Number of gates
  - Memory sizes
    - Registers
    - Internal scratch memories
    - Caches
  - Memory libraries
  - EDA tool-chain
  - Manufacturing process
    - Power profiles (HP, LP, G)
    - High-performance profiles

Modern Graphics Pipelines

- Molnar, et. al. proposed a classification for graphics architectures
  - sort first, sort middle, or sort last
- Most modern GPUs are *sort middle* machines
  - there's only one framebuffer, and results are determined per-pixel



Pros and Cons for “Traditional” GPUs

Pros

- Minimized latency
- Employ lots of parallelism for massive performance
- Relatively simple designs

Cons

- Lots of area required for all that parallelism
- Uses lots of bandwidth
- Uses lots of power
- Limited scalability for pixel shading
- “Brute force” misses some optimizations
- No practical way to interrupt rendering

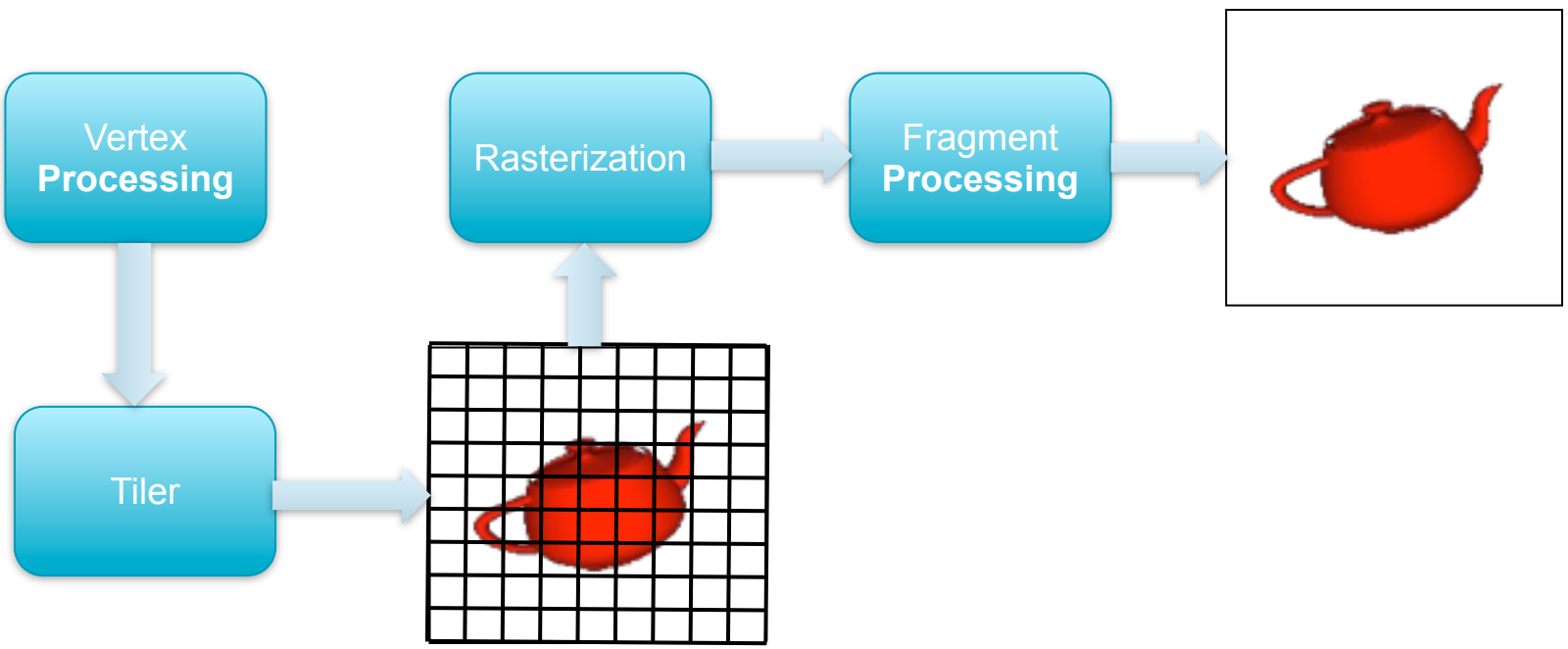
Probably not the best solution for mobile computing

Memory and Framebuffers

- Large in-core memories are undesirable
  - increase core area
  - require power to keep them updated
- Framebuffers require lots of memory
  - multiple buffers: color, depth, stencil
    - may be multi-sampled, which multiples storage requirements
  - multi-buffering requires multiple sets
- Requires lots of bandwidth to talk to those memories
  - Remember “Bandwidth = Power”
- And how large should we make them?
  - nowadays, resolution is the new selling point

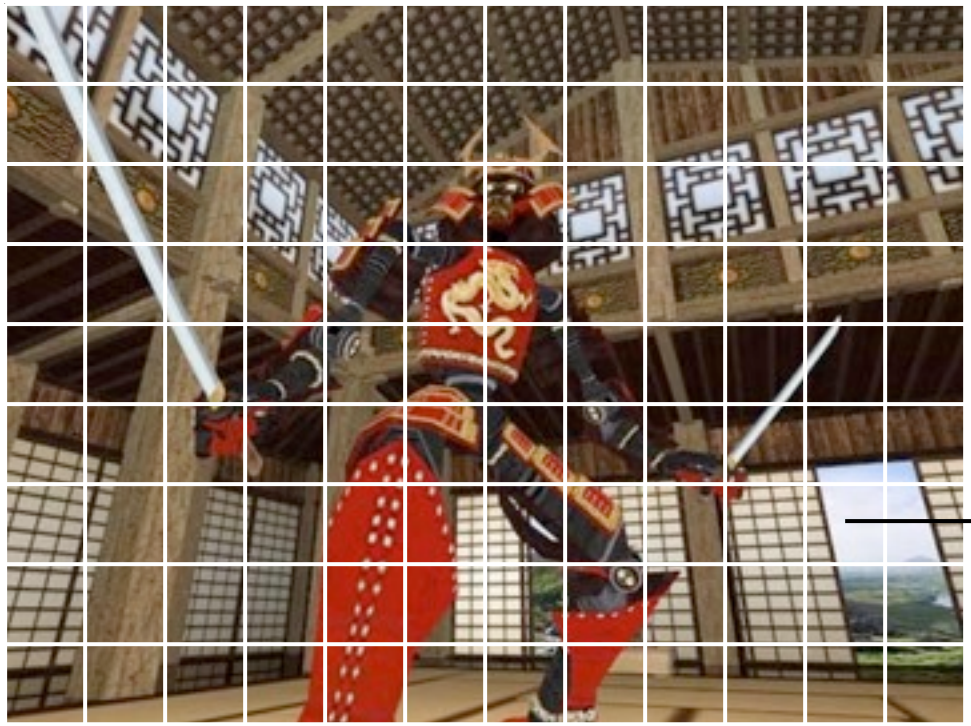
Tiled Graphics Pipelines

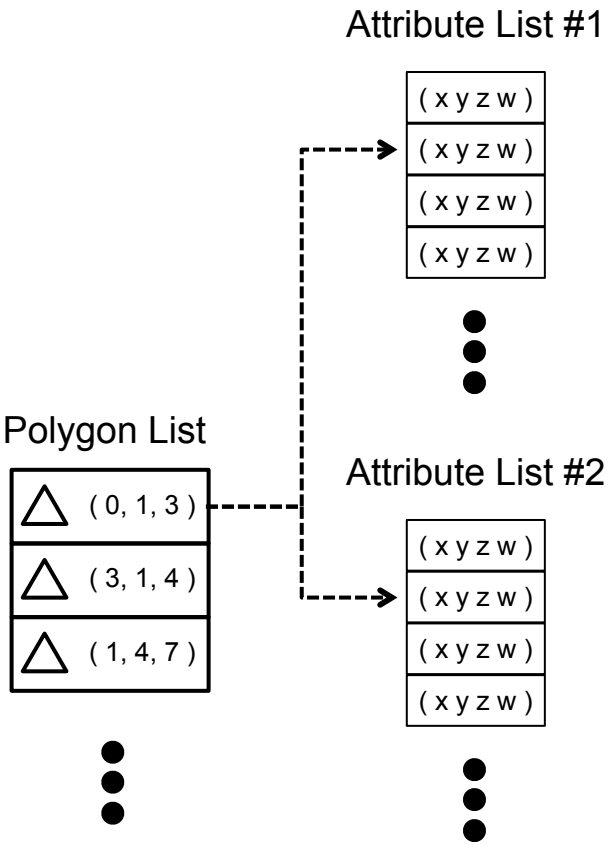
- Partition the problem to move parallelism farther up the pipeline
  - project geometric primitives into screen space, and then bin
    - need to record pipeline state for later use
- still a *sort middle* architecture



Tiling in a Nutshell

- Partition framebuffer into tiles to minimize on-core memory
- assemble image in system memory





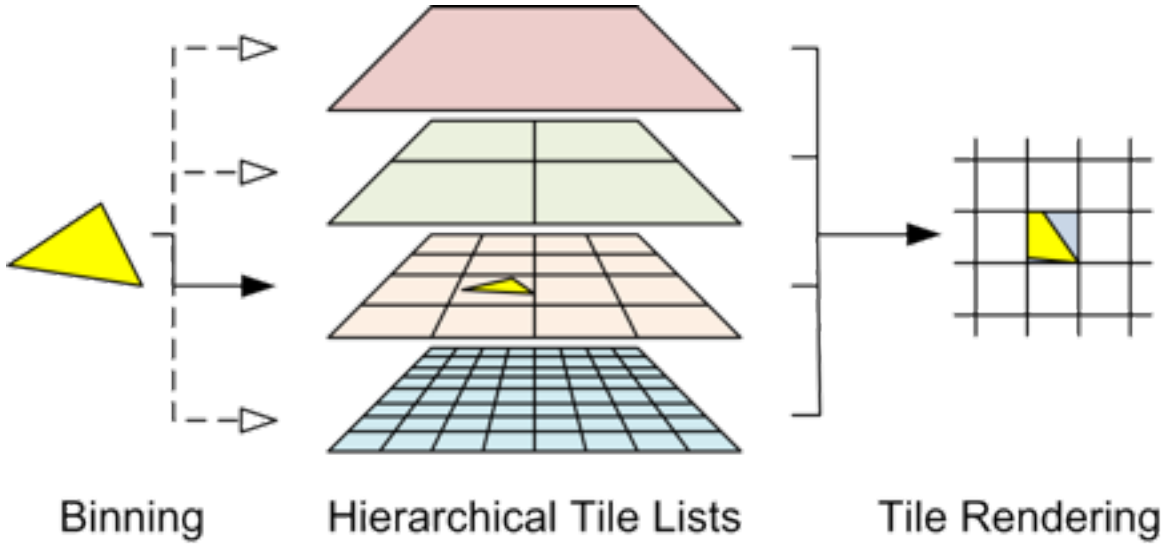
Advantages of Tiled Rendering

- Binning allows each tile to be rendered separately
  - each bin has complete information for rendering a tile
  - permits parallel rendering of tiles
    - allows *linear* pixel-fill scaling
      - that's really important as screen geometries increase in size
- Allows "interruptible" rendering
  - discard and restart rendering for a tile
  - provides a natural boundary for rendering jobs
- Aids throughput slow memory systems
  - multi-buffering allows processing of next tile while current tile awaits write-back

Disadvantages of Tiled Rendering

- Effectively always double-buffered
  - Single-buffering is challenging
  - possible, need to initialize memory with previously rendered tile
    - techniques like shadow passes are “inconvenient”
- Additional latency due to binning operation
  - can’t start rendering until application issues swap-buffers
- Additional memory required for primitive storage
  - need space for tile lists
  - driver may additional copies of application attribute data
    - VBOs (vertex-buffer objects) effectively remove this issue

Advanced Hierarchical Tiling



- Lower memory bandwidth than linear tiling
  - High-performance scalable binning architecture
  - Multiple bin sizes efficiently decompose primitives for shading
- Memory footprint now predictably related to scene complexity
  - Independent of primitive size and screen resolution

The Mali GPU portfolio

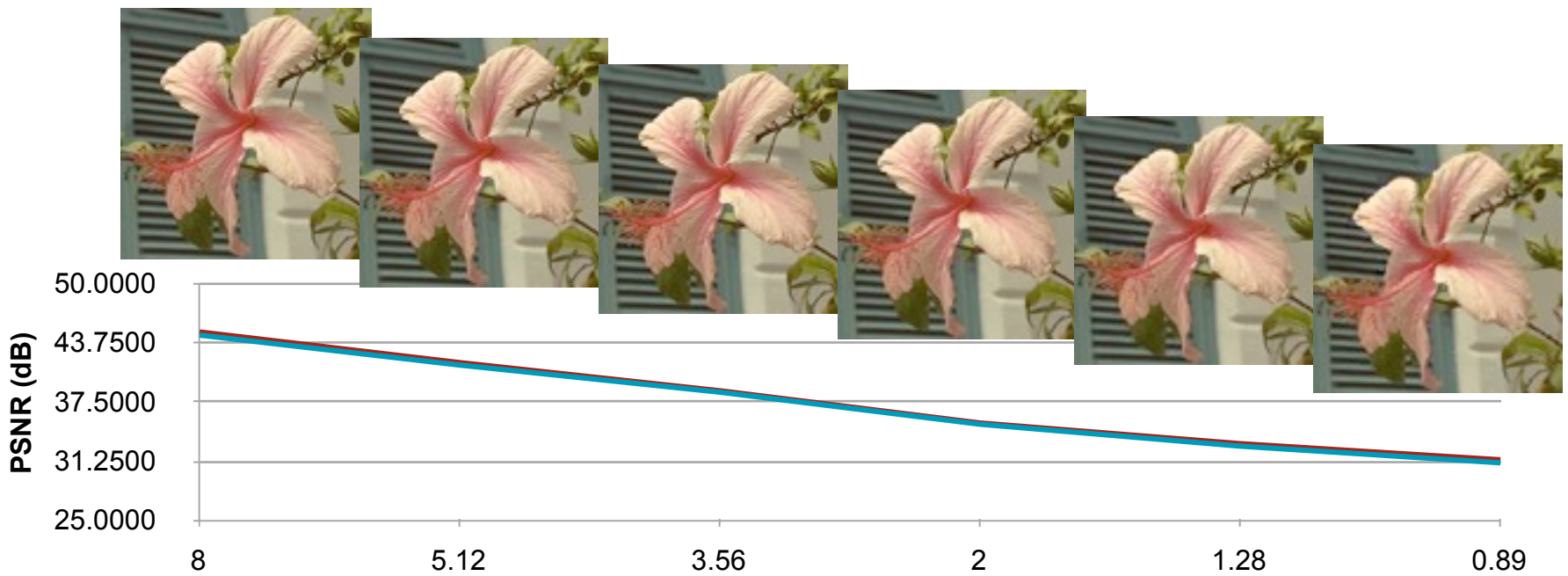
- Mali-200
  - Entry level, OpenVG® and OpenGL® ES 2.0
  - Leading anti-aliasing for superior image quality
  - World's most licensed OpenGL ES 2.0 core
- Mali-300
  - Ideal configuration for mid range use cases
  - Graphics acceleration with OpenVG and OpenGL ES
  - Efficient energy and bandwidth usage
- Mali-400MP
  - High performance OpenVG and OpenGL ES 2.0
  - World's first multi-core embedded GPU
  - Scalable pixel processing up to 1080p displays
  - Leading power efficiency and reduced bandwidth
- Mali-T604
  - Tri-pipe for performance and flexibility
  - GPGPU computing with OpenCL™ 1.1
  - State of the art bandwidth reduction
  - DirectX® 11 and next generation Khronos graphics standards

Unified Memory Considerations

- SoCs generally share memory across all components
  - CPU, graphics, video, etc.
- Benefits for graphics
  - effectively limitless graphics memory
  - sharing images (e.g., from video engine as textures) usually requires no memory copying
  - potential advantages in systems with CPU coherence
- Disadvantages for graphics
  - bus bandwidth shared among all devices
    - can affect latency

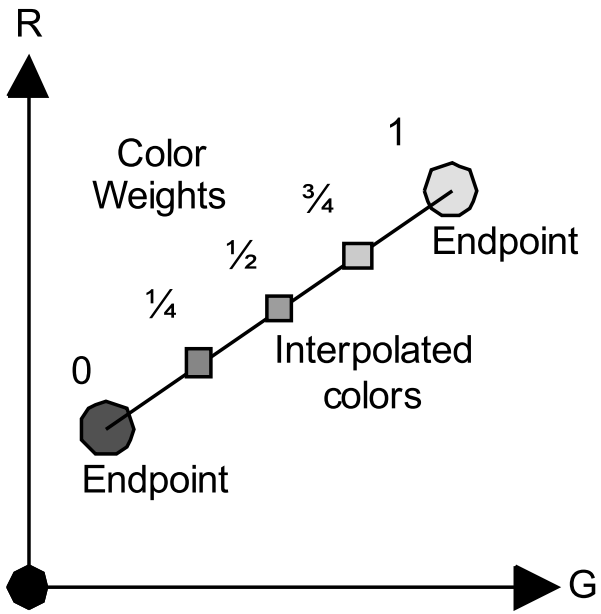
Adaptive Scalable Texture Compression

- New texture compression standard developed by ARM
  - Adopted as standard texture compression for future OpenGL ES versions
- Increased quality and fidelity at low bit-rates
- Expansive range of input formats offers complete flexibility
  - Orthogonal choice of base format (L / LA / RGB / RGBA)
  - 2D and 3D textures, addition of high-dynamic range pixel formats



ASTC Texture Compression

- Gradient-based
  - Per-texel “weights” specify interpolation in block-global color spaces
  - Weights can be coarsely quantized and sampled
  - Weights can be divided into planes for uncorrelated components
  - Chosen on a per-block basis



Texel Weights

0	0	¼	¾
0	0	¼	1
¼	¼	½	1
¾	1	1	1

stored with block

Transaction elimination

Frame N

Maintain a list of signatures for each tile

Transaction elimination

Frame N

Frame N+1

Maintain a list of signatures for each tile

Compare to sigs calculated for frame N+1

Where signatures match, don't write the tile

Transaction elimination

Frame N

Frame N+1

Maintain a list of signatures for each tile

Compare to sigs calculated for frame N+1

Where signatures match, don't write the tile

- Surprisingly effective, even on FPS games and video

APIs and Power

- This needs to be the new thought paradigm …
  - Industry's calling in "Energy Efficient Programming"
- For OpenGL (ES), there are a few things to keep in mind
  - Use texture compression
  - Manage FBOs carefully
  - Use buffer objects

Reducing Power with Buffer Objects

- Tile-based renderers need to store graphics data for later rendering
  - this implies a lot of copying a lot of data
- Situation is worsened by certain API idioms
  - client-side vertex arrays are evil
  - the problem is we can't keep track of data changes from the application
    - all we get is a pointer to host memory
    - this forces us to make a copy at every draw call
    - if only we knew when the application updated the data hanging off that pointer
- Buffer Objects (particularly VBOs) can have a massive impact for tile-based renderers
  - modifications to data are strictly controlled by the API
    - `glBufferData`, `glBufferSubData`, `glMapBuffer`, etc.

Being a Considerate Renderer

- Indexed rendering (e.g., using `glDrawElements`) can be used for good or evil
  - it's also really useful for reducing redundant computation,
  - and can be engage transformed vertex caching
  - but, it can also really affect power
- We essentially need to scan the index list to determine which vertices you're going to use
  - more bandwidth, more copying
- A few pointers:
  - try to keep the range of your indices and VBOs of similar size
  - store indices in VBOs a well (it gives us more time to analyze them)

What I Do in My Day Job

- "It's been three years since I rendered my last triangle …" ☺
- Officially, I label myself a "technologist" nowadays
- I spend my days concentrating on three things:
  - Predicting the future
  - Wrangling API design
  - Email

Working with Technology Standards

- About ½ my job is working with the Khronos Group
  - OpenGL, OpenGL ES, OpenCL, COLLADA, WebGL, WebCL, etc.
- What Khronos does:
  - analyzes and integrates advancing technology into APIs
  - creates specifications
  - designs conformance testing suites
- How this happens
  - Choose *n* engineers from *m* companies expressing *k* opinions
    - $n \geq m$, $k \rightarrow \infty$
    - discuss
  - a few good skills to have in this scenario
    - negotiation
    - mediation
    - leadership

When You Come for a Job

- Programming skills
  - C, preferably C++
  - scripting language: Python, Perl
- Communication skills
  - PowerPoint is everywhere
  - know how to write (English)
  - don't be afraid to talk in front of a group
    - this isn't strictly required, but will probably advance faster than any other skill
- Programming skills (addendum)
  - debuggers, profilers, assembly
  - understand caches, code locality, efficiency