Unleash the Benefits of OpenGL ES 3.1 and the Android Extension Pack

Hans-Kristian Arntzen
Software Engineer, ARM

Dan Galpin
Developer Advocate, Google

Tom Olson
Director, Graphics Research, ARM
Chair, OpenGL ES and GL Next Working Groups

Topics

- OpenGL® ES 3.1
  - Compute shader techniques and best practices
- Android Extension Pack
  - Applications and recommendations
- Mali OpenGL ES SDK examples
- A glimpse of the future: GL Next

ARM® MALI™
Visual Technology

Vulkan™

ARM

Topics

- **OpenGL ES 3.1**
  - **Overview**
  - Introduction to compute shaders
  - Useful compute shader techniques
  - Best practices on ARM® Mali™ Midgard GPUs
- Android Extension Pack
- Mali OpenGL ES SDK samples
- A glimpse of the future: GL Next

ARM

OpenGL ES 3.1

- Status
  - Spec released at GDC 2014
  - Conformance test active since June 2014
  - Exposed in Android L

- Goals
  - Expose features in OpenGL ES 3.0 capable hardware
  - Backward compatible with OpenGL ES 2.0 / 3.0 applications
  - Backward compatible with OpenGL ES 3.0 hardware
  - Enable rapid adoption

OpenGL ES 3.1 Key Features

- Compute shaders
- Indirect draw calls
- Memory resources
  - Images
  - Shader Storage Buffer Objects
- Enhanced texturing
  - Texture gather
  - Multisample textures
  - Stencil textures
- Separate shader objects
- Shading language features
  - Arrays of arrays
  - Bitfield operations
  - Location qualifiers
  - Memory read / write
  - Synchronization primitives
- …and many more

Topics

- **OpenGL ES 3.1**
  - Overview
  - **Introduction to compute shaders**
  - Useful compute shader techniques
  - Best practices on ARM Mali Midgard GPUs
- Android Extension Pack
- Mali OpenGL ES SDK samples
- A glimpse of the future: GL Next

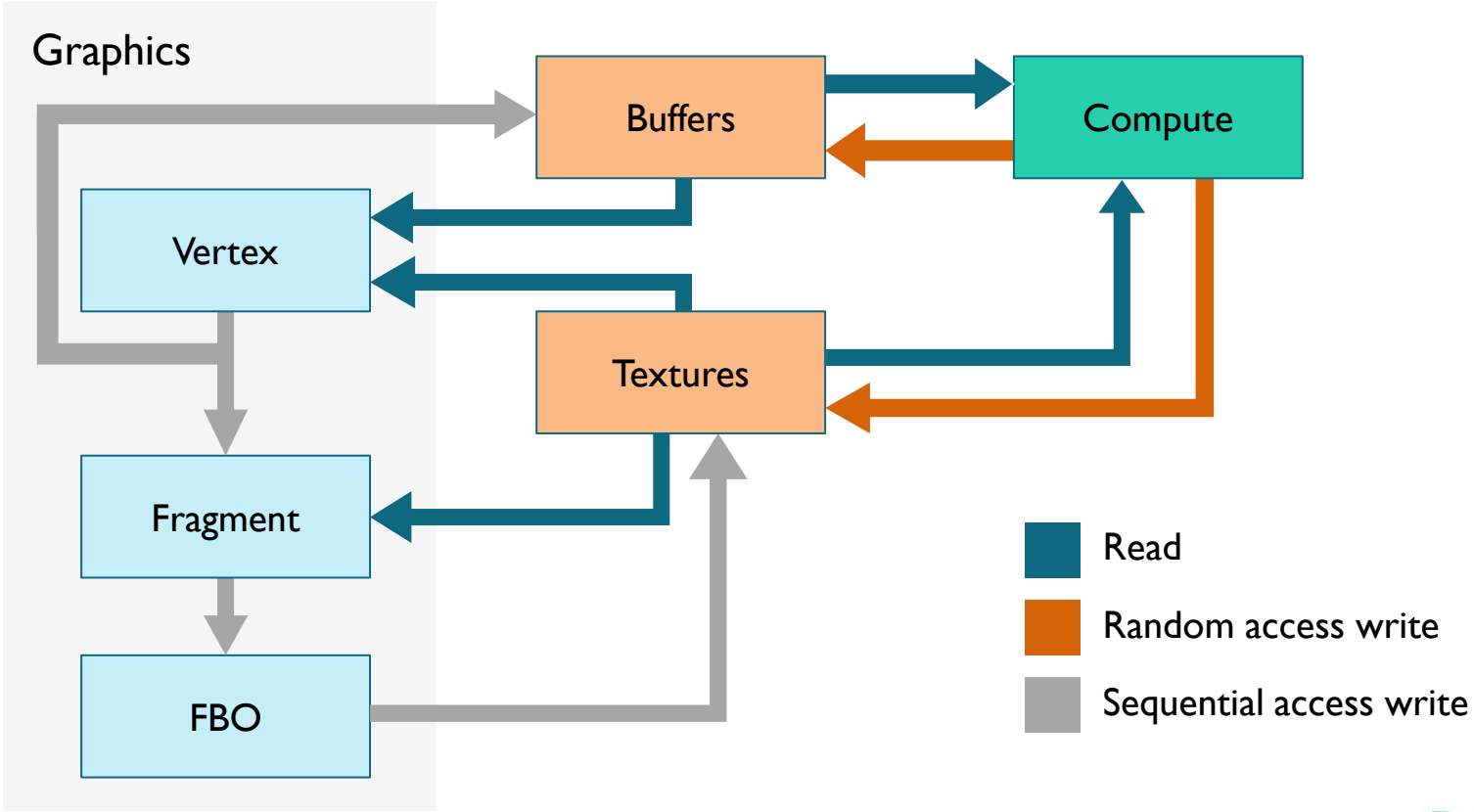
ARM

Introduction to Compute Shaders

- Brings GPGPU functionality to OpenGL ES
- Familiar Open GLSL syntax

- Random access writes to buffers and textures
- Sharing of data between threads

Role of Compute in Graphics



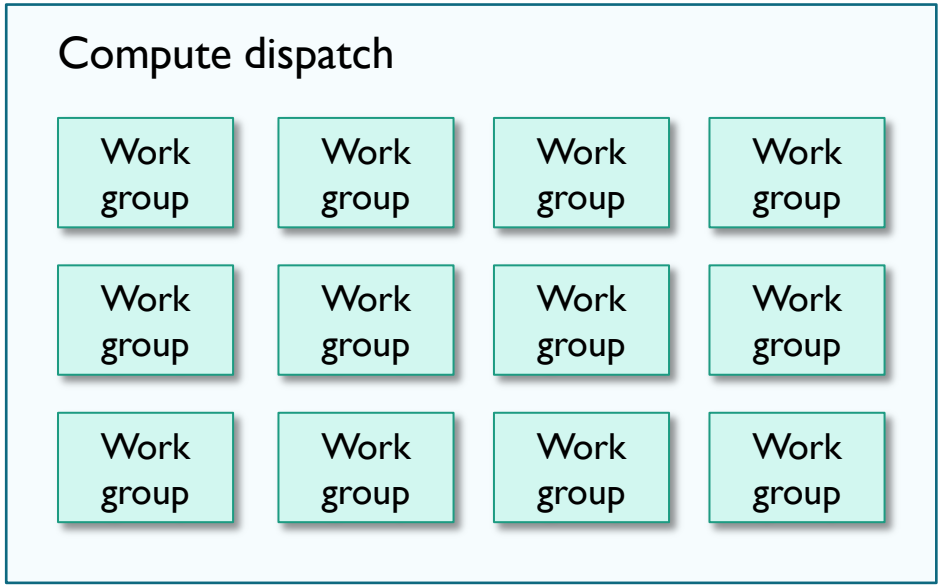
ARM

Compute Model

- Traditional graphics pipeline
  - No random access write
  - Implicit parallelism
  - No synchronization between threads
- Compute
  - Random access writes to buffers and textures
  - Explicit parallelism
  - Full synchronization and sharing between threads in same work group

Work Group – the Compute Building Block

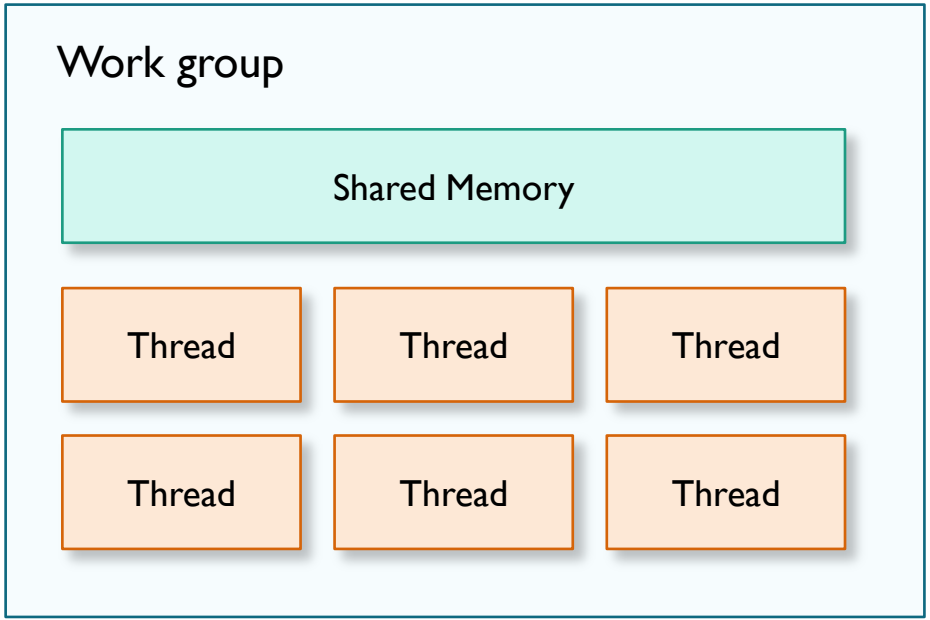
- Independent
- Up to three dimensions



ARM

Work Group

- Shared memory
- Concurrent threads
- Synchronization
- Unique identification
  - gl_LocalInvocation{ID,Index}
  - gl_GlobalInvocationID
  - gl_WorkGroupID



ARM

Hello, Compute World

```
#version 310 es
layout(local_size_x = 1) in;

layout(std430, binding = 0) buffer Output {
    writeonly float data[];
} output;

void main() {
    uint ident = gl_GlobalInvocationID.x;
    output.data[ident] = float(ident);
}
```


ARM

Compiling and Executing a Compute Shader

```
GLuint shader = glCreateShader(GL_COMPUTE_SHADER);
// ... Compile, attach and link here.

glUseProgram(program);
glDispatchCompute(work_groups_x,
    work_groups_y,
    work_groups_z);
```

Integrating Compute with OpenGL® ES 3.1

- Shader Storage Buffer Objects (SSBOs)
  - General purpose writeable buffer objects
- Image load/store
  - Raw access to texture data
- Atomic operations
  - Modify same memory safely from multiple threads
- Synchronization
  - Define dependencies between compute and graphics pipelines

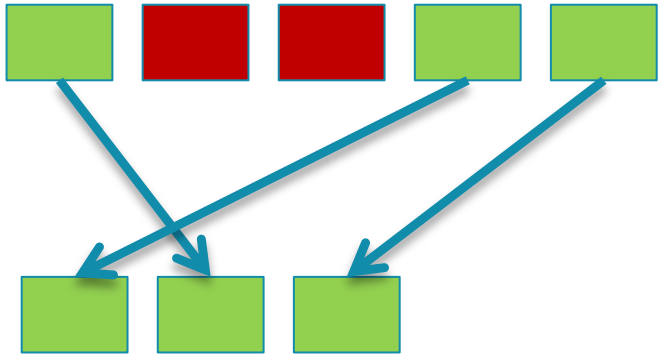
Topics

- **OpenGL ES 3.1**
  - Overview
  - Introduction to compute shaders
  - **Useful compute shader techniques**
  - Best practices on ARM Mali Midgard GPUs
- Android Extension Pack
- Mali OpenGL ES SDK samples
- A glimpse of the future: GL Next

ARM

Parallel Dataset Filtering

- Frustum culling lots of instances
  - Or more generally, filtering datasets
- Input
  - SSBO with per-instance data
- Output
  - Tightly packed SSBO with per-instance data
  - Number of instances which passed arbitrary test
  - Atomic counter hooked up to indirect parameter ☺
  - Combine with physics update?
  - No ordering guarantees



ARM

Culling Snippet

```
readonly buffer InputData   { vec4  in_elems[]; };
writeonly buffer OutputData { vec4 out_elems[]; };
uniform atomic_uint output_count;

void main() {
    uint ident = gl_GlobalInvocationID.x;
    vec4 instance = in_elems[ident];
    if (some_arbitrary_test(instance)) {
        uint unique = atomicCounterIncrement(output_count);
        out_elems[unique] = instance;
    }
}
```

LOD-sorting

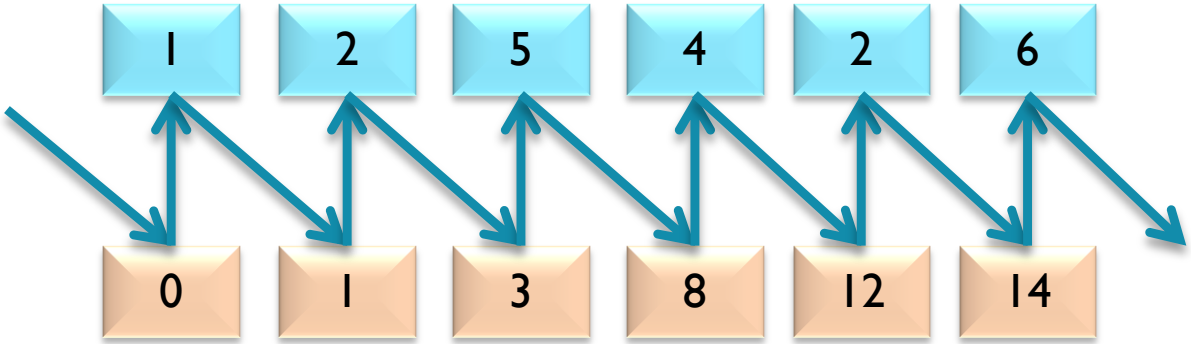
- Improvement to just frustum culling
- Multiple output buffers and multiple counters
- Test distance, determine LOD and push to corresponding buffer
- Crude front-to-back sorting a nice bonus
- Fast SSBO atomics very convenient

```
uint unique = atomicAdd(count_buffer.lods[lod], 1u);

output_buffer.elems[lod * lod_stride + unique]
    = instance;
```

Parallel Prefix Sum

- Well-known compute primitive
- Parallel Radix-sort
- Summed area tables
- See Particle Flow SDK sample for more details ☺



ARM

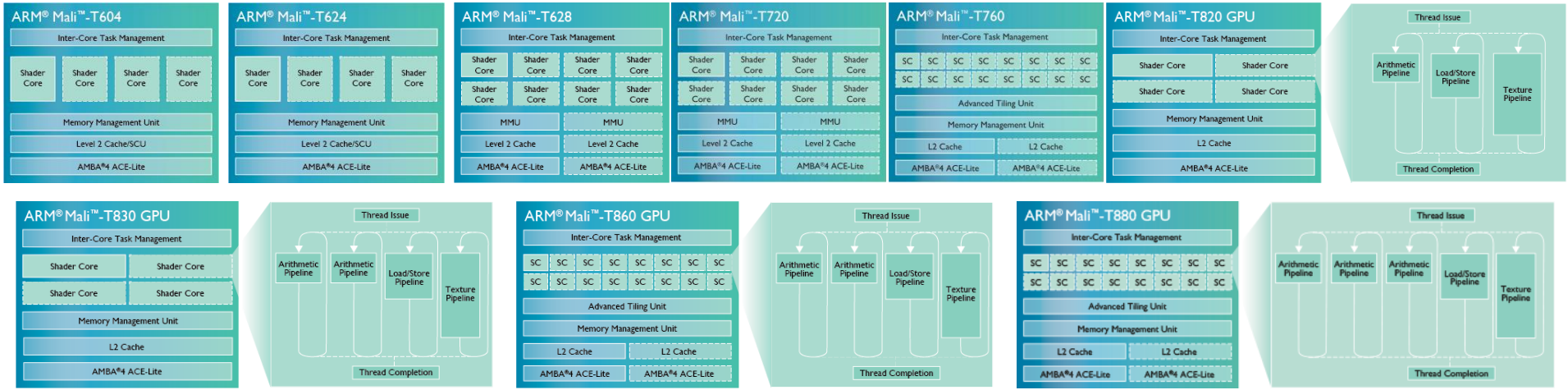
Topics

- **OpenGL ES 3.1**
  - Overview
  - Introduction to compute shaders
  - Useful compute shader techniques
  - **Best practices on ARM Mali Midgard GPUs**
- Android Extension Pack
- Mali OpenGL ES SDK samples
- A glimpse of the future: GL Next

ARM

Desktop and Mali GPUs are Quite Different

- Especially important to consider for compute
- Compute exposes lower-level details



Trust the Cache

- Global memory isn't necessarily slow
  - After all, textures are global memory
  - On Midgard, global and shared memory are backed by cache
  - … and equally fast

- **Never** copy from global to shared memory just for "caching"
  - Use shared to share results of significant computations

Use shared memory for something useful

```
// Useless read + write, polluting cache with redundant data
shared vec4 textureCache[SIZE];
textureCache[id] = texture(...);
memoryBarrierShared();
barrier();
```

```
// Better, might beat multipassing
// Be aware of data-expansion, consider pack*()/unpack*()
shared vec4 textureCache[SIZE];
textureCache[id] = useful_preprocessing(texture(...));
memoryBarrierShared();
barrier();
```

Don’t Be Afraid of Divergence ...

- Midgard is not a warp architecture
  - Diverging threads does not come at extra cost
  - Complicated algorithms to avoid divergence not needed
  - Be aware of desktop compute code which assumes warps of certain size
- Execution barrier more expensive
  - Synchronizing threads more expensive than on desktop
  - Impact very application dependent
- Basically opposite of desktop GPUs

... but Keep Critical Path Short

- Avoid blocking threads for too long
  - Especially if between two barrier()s

```
barrier();

if (gl_LocalInvocationIndex == 0u)
{
    // Lots of single threaded code here!
    compute_lots_of_stuff();
}

barrier();
```

Prefer Atomics on Global Memory

- Atomics on shared memory typical for desktop
- Overhead of synchronization not worth the reduced contention
- Advice might change if we see 16-core Midgard chips
- If no contention on cache line between cores, practically free ☺
- cl_arm_core_id for GLES in the works

- Atomic counters are not "optimized atomics"
  - Just as fast as atomics on SSBOs on Mali Midgard GPUs
  - Might be useful for performance portability

Avoid Large Numbers of Small Indirect Draws

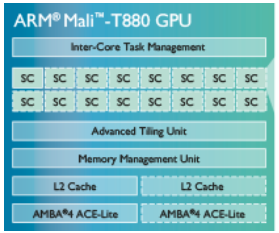
- instanceCount of 0 often just as expensive as 1
- Use sparingly
  - Indirect draw is not a replacement for regular draw calls
- Ideal case is instancing
  - Overhead for an indirect draw is near-constant per draw call

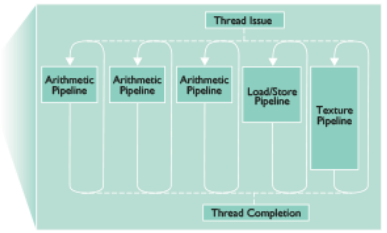
Mind Your Work Group Size

- If using barrier()
  - Consider smaller work group sizes
  - Smaller work groups means more independent work groups on-chip
- If not using barrier()
  - Large work groups preferred, 64 threads tends to be sweet spot
- Work group size should be multiple of 4
- OpenGL ES 3.1 supports at least 128 threads
  - GL_MAX_COMPUTE_WORK_GROUP_INVOCATIONS
- If desired number of threads does not align well with group size
  - Return early or branch over code

Prefer Texture Pipeline When Reading Textures

- texture(Lod)Offset() is excellent, also in compute
  - Edge clamping for filter kernels
  - Avoid wasting ALU on computing texel offsets
- Uses different pipeline than imageStore()
  - More throughput in tripipe
  - Dedicated texture cache also helps
- Use imageLoad() if coherency is needed





ARM

Fragment Shader Still Preferred for Basic Image Processing

- Tiled architecture ensures predictable bulk write-outs
  - ARM Frame Buffer Compression (AFBC)
  - imageStore() goes through cache system instead of tile memory
- Hard to beat fragment, even with clever techniques
- Always profile

Topics

- OpenGL ES 3.1
- **Android Extension Pack**
  - **Overview**
  - ARM® Mali™ hardware support
  - Best practices on ARM® Mali™ Midgard GPUs
- Mali OpenGL ES SDK samples
- A glimpse of the future: GL Next

ARM

Android Extension Pack (AEP)

- What is it?
  - A ‘meta-extension’ rolling up 20 other extensions and features
  - Requires OpenGL ES 3.1
  - Exposed as GL_ANDROID_extension_pack_es31a
- Status
  - Optional feature in Android ‘L’ on platforms supporting OpenGL ES 3.1

Android Extension Pack (AEP)

- Manifest for Requiring AEP

```
<manifest>
    <uses-feature
        android:name="android.hardware.opengles.aep"
        android:required="true" />
    ...
</manifest>
```


ARM

Android Extension Pack – Features

- Unspeakably cool texturing functionality
  - KHR_texture_compression_astc_ldr
- Ease of getting your code working
  - KHR_debug
- Enhanced Fragment Shaders
  - Guaranteed support for shader storage buffers, images and atomics

Android Extension Pack – More Features

- Extended framebuffer operations
  - EXT_draw_buffers_indexed
  - KHR_blend_equation_advanced
- Per-sample processing
  - OES_sample_shading
  - OES_sample_variables
  - OES_shader_multisample_interpolation

Android Extension Pack – Even More Features

- Somewhat less cool texturing functionality
  - EXT_texture_cube_map_array
  - EXT_texture_sRGB_decode
  - EXT_texture_buffer
  - EXT_texture_border_clamp
  - OES_texture_stencil8
  - OES_texture_storage_multisample_2d_array

Android Extension Pack – *Still* More Features

- Tessellation and Geometry
  - EXT_tessellation_shader
  - EXT_geometry_shader
  - EXT_shader_io_blocks
  - EXT_primitive_bounding_box

Topics

- OpenGL ES 3.1
- Android Extension Pack
  - Overview
  - **ARM® Mali™ hardware support**
  - Best practices on ARM® Mali™ Midgard GPUs
- Mali OpenGL ES SDK samples
- A glimpse of the future: GL Next

ARM® MALI™
Visual Technology

ARM

Topics

- OpenGL ES 3.1
- Android Extension Pack
  - Overview
  - ARM® Mali™ hardware support
  - **Best practices on ARM® Mali™ Midgard GPUs**
- Mali OpenGL ES SDK samples
- A glimpse of the future: GL Next

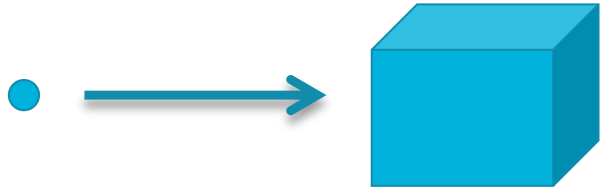
ARM® MALI™
Visual Technology

ARM

Tessellation

- Make use of dynamic subdivision parameters
- Cull patches in control shader if possible (back-facing, frustum)
  - No need to compute triangles if they are all culled in the tiler
- Avoid creating micro-triangles
  - Easy to go overboard with tessellation if not careful
- Spend triangles where they count
  - Triangles forming the silhouette are the most important
- Make use of primitive bounding box extension
  - Allows GPU to cull as early as possible

Geometry Shaders



- Prefer points as input
- Avoid combining with primitive restarts and tessellation
- Avoid layered rendering
  - Have to tile multiple framebuffers at same time, memory hungry
- Set *layout(max_vertices = N)* to match your actual output
- If possible, output constant number of vertices in shader
- Best usecase, amplifing points to lines, triangles, quads, cubes, etc
  - Compute good alternative to many other use cases for geometry ☺

Rest of Android Extension Pack

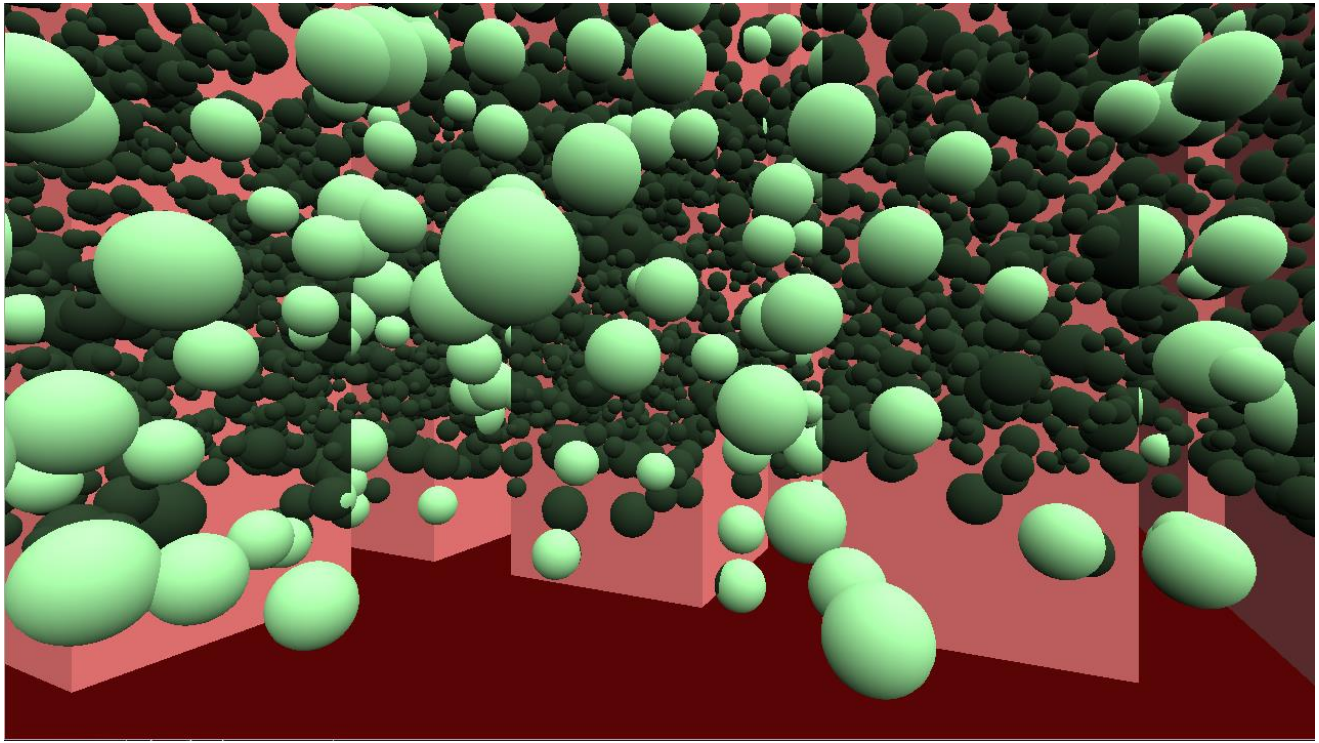
- ASTC
  - Highly recommended ☺
- EXT_copy_image is expensive
  - Handrolling for your particular usecase will likely be much faster
  - imageStore() with data reinterpretation might be what you want
- KHR_debug
  - Recommended, more and more diagnostics will be added with time
- No particular caveats for rest of extension pack

Topics

- OpenGL ES 3.1
- Android Extension Pack
- **Mali OpenGL ES SDK samples**
- A glimpse of the future: GL Next

ARM

Occlusion Culling with Hierarchical-Z



ARM

Video

ARM

Occlusion Culling with Hierarchical-Z

- Determines visibility of instances and only draws visible objects
- Very useful technique to manage massive instancing on GPU

- Compute shader features
  - Shader storage buffer objects
  - Indirect draw
  - Atomic operations

Occlusion Culling Algorithm – Hi-Z

- Obtain depth map of occluders on GPU
  - Many ways to do this, SDK sample renders depth to a 256x256 texture
- Find multi-resolution conservative depth map by mipmapping
  - Need to do this manually by taking max depth of quad
  - textureGather() ☺

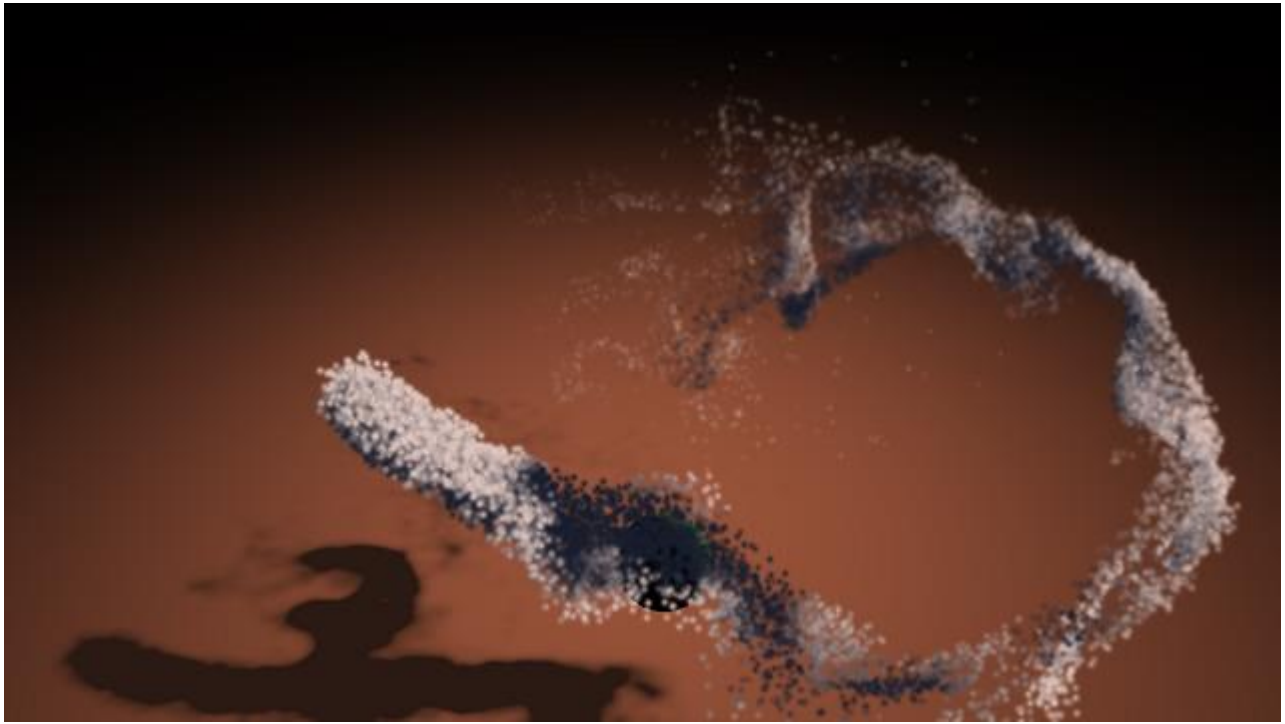
Mip 0 → Mip 4

ARM

Testing Objects

- Before testing occlusion, do frustum test
  - Very cheap. Branchy code not an issue on Mali Midgard ☺
- Find screen space bounding box of object
  - Map bounding box to a mip-level where 4 pixels covers BB
  - Sample depth with GL_LINEAR shadow compare to test occlusion
  - If test is passed, atomically increment instance count and append
- Only GPU knows how much to draw
  - Draw instances with indirect draw
  - If latency is acceptable, reading back instance count to CPU is possible

Particle Flow Simulation with Compute Shaders



ARM

Video

ARM

Particle Flow Simulation with Compute Shaders

- Procedural noise field
- Radix sort particles in compute for correct blending
- 4-layer opacity shadow map

- Compute shader features
  - Shader storage buffer objects
  - Shared memory
  - Memory and execution barriers

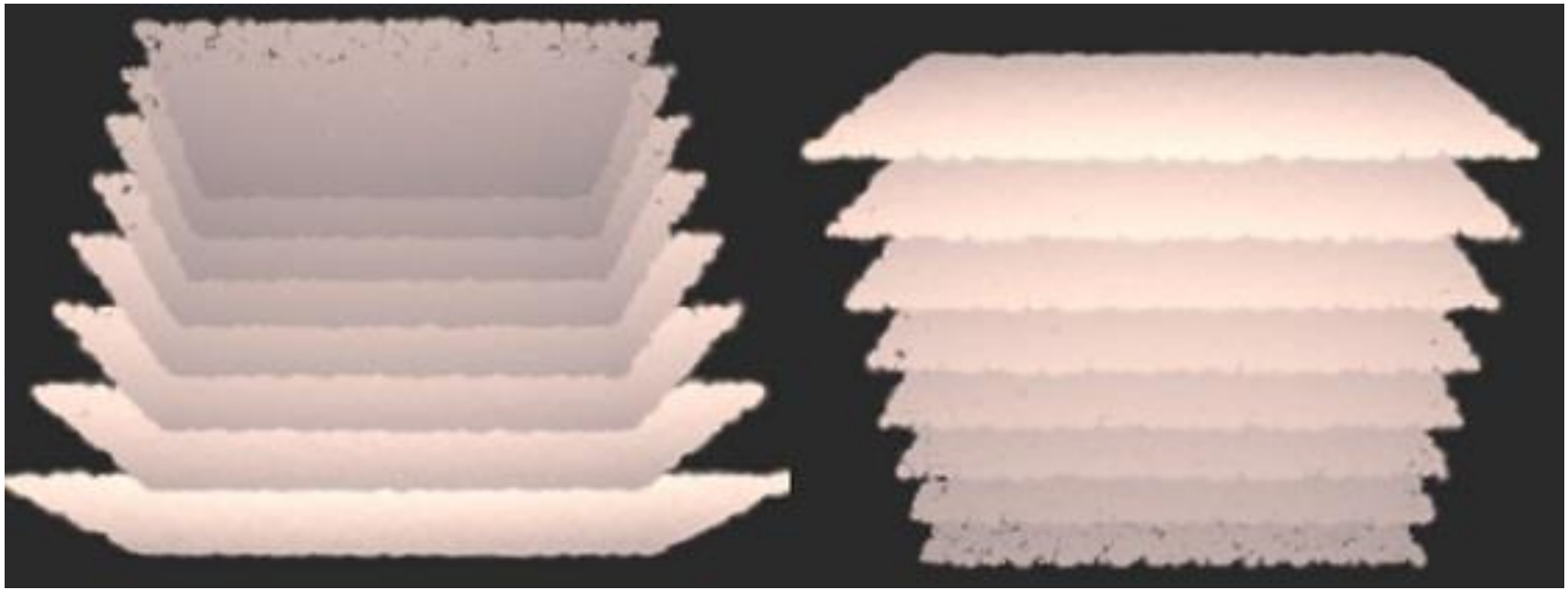
Simulating Particles

- Noisy 3D potential field
  - Turbulent motion to particles
  - Generated procedurally with simplex noise
  - Velocity is gradient of potential
- Update particles in compute
  - Emits new particles or move existing particles
  - Possible with XFB, but compute is cleaner

Drawing Particles

- Basic point sprites with alpha blending
- Blending must happen in correct order
  - Sort back-to-front
- Particle data is owned by GPU
  - Need way to sort particles in parallel

Incorrect vs. Correct Blending Order



ARM

Radix Sort

- Particle depth converted to 16-bit unorm
- Sort on 2 bits at a time
- Parallel prefix sum basic building block
- See full SDK sample for more detail

4-layer Opacity Shadow Map

- Cheap volumetric shadows
- Quantize depth into 4 ranges
- Use RGBA8 channels to store accumulated opacity
- Color mask and blend particles within sub-ranges

Topics

- OpenGL ES 3.1
- Android Extension Pack
- Mali OpenGL ES SDK samples
- **A glimpse of the future: GL Next**

ARM

The Vision

- An open, cross-platform 3D + Compute API for the modern era
  - Clean, modern architecture
  - Multi-thread / multi-core friendly
  - Greatly reduced CPU overhead
  - Full support for both tile-based and immediate-mode renderers
  - Explicit application control of when and where work is done

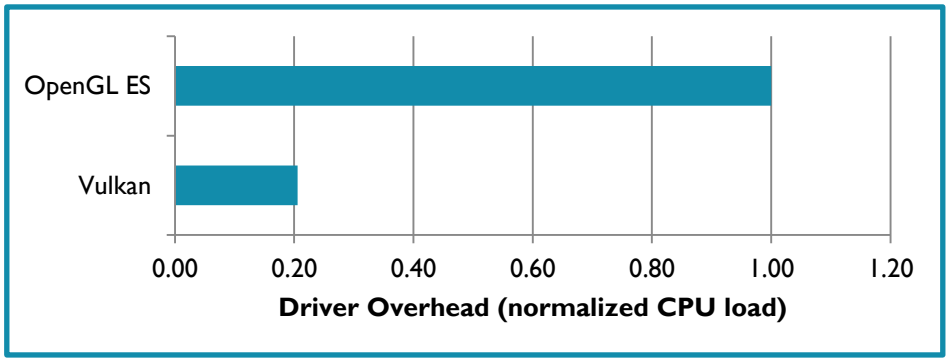
ARM

Status

- Product of unprecedented cooperation between ISVs and IHVs
  - Intense focus has allowed very rapid progress
- Consensus on core functionality
  - ‘Alpha’ header file
  - Spec drafting in progress
- IHV and ISV experiments in flight
  - Prototype drivers
  - Experimental ports of game engines

Vulkan Investigations at ARM

- Prototype driver for Mali-T760 is in progress
  - Not a product – goal is to validate the API
  - Can create textures, UBOs, command buffers and queues
  - Preliminary results are promising
  - Work is ongoing – watch this space!



ARM

To learn more…

glNext: The Future of High Performance Graphics

Room 2006, West Hall

Thursday, March 5th, 10:00-11:00am

More on the Next Generation Graphics and Compute API

SF Green Space, 657 Mission St, Suite 200, San Francisco CA 94105

Thursday, March 5th, 12:00-1:30pm and 2:00-3:30pm

ARM

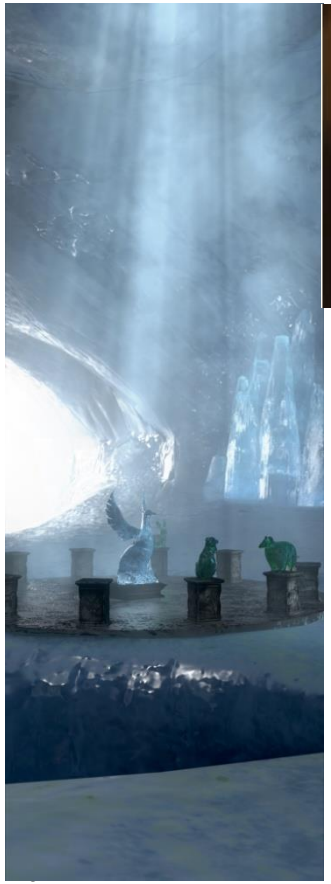
Summary

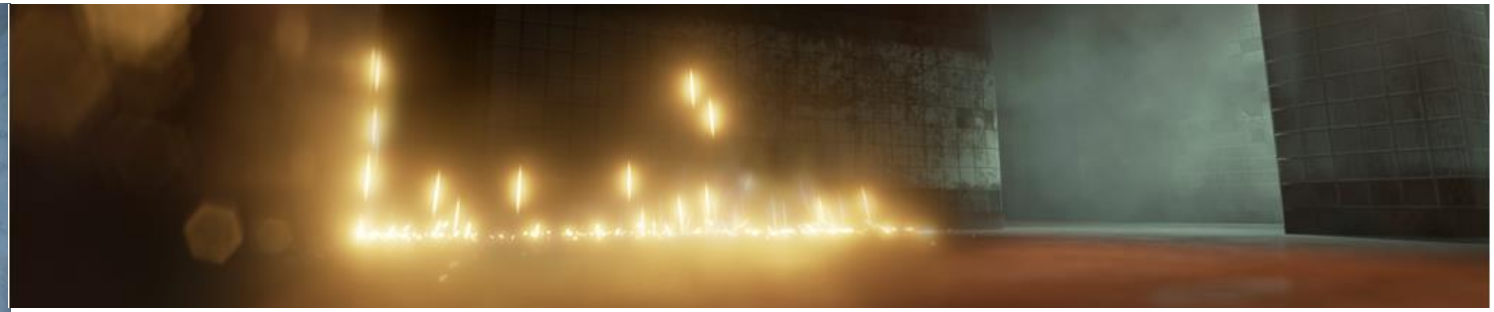
- OpenGL ES 3.1 is here!
  - Compute shaders open up tremendous new possibilities for applications
- Android Extension Pack adds many cool features
  - ASTC-LDR, KHR_debug, computing in the fragment shader, and more
- The Mali OpenGL ES SDK can help you get started
- Vulkan is coming
  - Looks like a great fit to Mali GPU architectures

Further Reading

- Mali OpenGL ES SDK for Linux
  - http://malideveloper.arm.com/develop-for-mali/sdks/opengl-es-sdk-for-linux/
  - http://malideveloper.arm.com/develop-for-mali/sample-code/introduction-compute-shaders-2/
- OpenGL ES 3.1 reference
  - https://www.khronos.org/opengles/sdk/docs/man31/
- ASTC
  - http://www.arm.com/products/multimedia/mali-technologies/adaptive-scalable-texture-compression.php
  - http://malideveloper.arm.com/develop-for-mali/tools/software-tools/astc-evaluation-codec/

To Find Out More….





- ARM Booth #1624 on Expo Floor
  - Live demos
  - In-depth Q&A with ARM engineers
  - More tech talks at the ARM Lecture Theatre

- **http://malideveloper.arm.com/GDC2015**
  - Revisit this talk in PDF and video format post GDC
  - Download the tools and resources

More Talks from ARM at GDC 2015

Available post-show online at Mali Developer Center

- Unreal Engine 4 mobile graphics and the latest ARM CPU and GPU architecture - Weds 9:30AM;West Hall 3003

  This talk introduces the latest advances in features and benefits of the ARMv8-A and tile-based Mali GPU architectures on Unreal Engine 4, allowing mobile game developers to move to 64-bit's improved instruction set.

- Making dreams come true – global illumination made easy – Thurs 10AM;West Hall 3014

  In this talk, we present an overview of the Enlighten feature set and show through workflow examples and gameplay demonstrations how it enables fast iteration and high visual quality on all gaming platforms.

- How to optimize your mobile game with ARM Tools and practical examples – Thurs 11:30AM;West Hall 3014

  This talk introduces you to the tools and skills needed to profile and debug your application by showing you optimization examples from popular game titles.

- Enhancing your Unity mobile game – Thurs 4PM;West Hall 3014

  Learn how to get the most out of Unity when developing under the unique challenges of mobile platforms.

Any Questions???

Ask the best question and win a the **Samsung Galaxy Note 4** with OpenGL ES 3.1 (on Android L)

ARM

Thank You
Questions?