Real-time GPU-driven Ocean Rendering on Mobile

Hans-Kristian Arntzen
Engineer, ARM

ARM Game Developer Day - London
03/12/2015

Topics

- Tessendorf Fast Fourier Transform method
  - Procedural frequency domain
  - Choppy waves
  - Jacobian factor
- Efficient compute shader FFTs on Mali
- Heightmap rendering with LOD
  - Tessellation
  - Continous LOD Morphed Geo-MipMap
- Mali SDK samples

Tessendorf Fast Fourier Transform Method

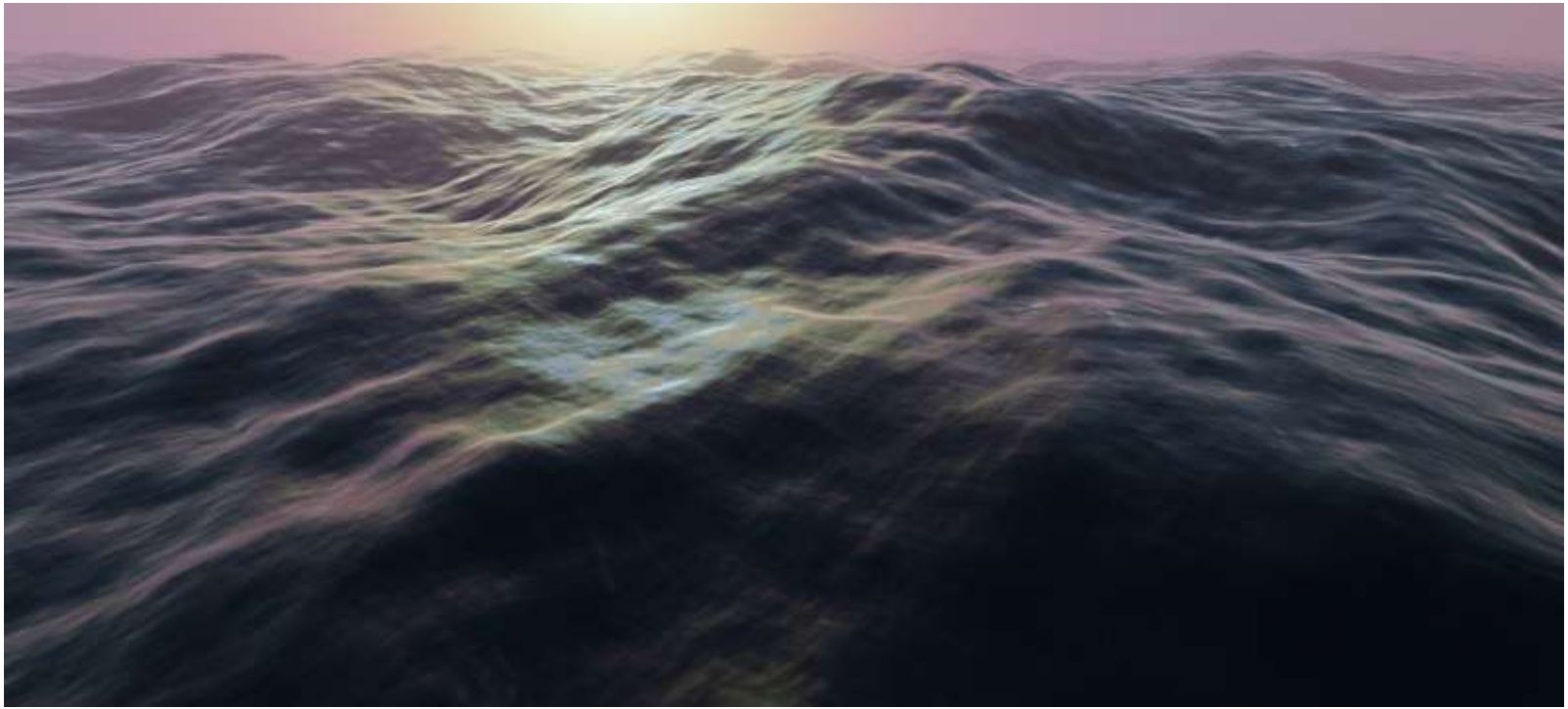
- Began as offline rendering, now realtime with GPU compute
- Ocean waves can be seen as a sum of many waves
- Power distribution can be modelled in the frequency domain
  - Stochastic process
  - Phillips spectrum popular
  - Lots of artistic freedom
- Water can be animated by rotating the phase in frequency domain
  - Different wavelengths traverse at different speeds
- Inverse FFT for heightmap

$$x(n) = \sum_{k=0}^{N-1} X(k) \exp\left(\frac{j2\pi kn}{N}\right)$$

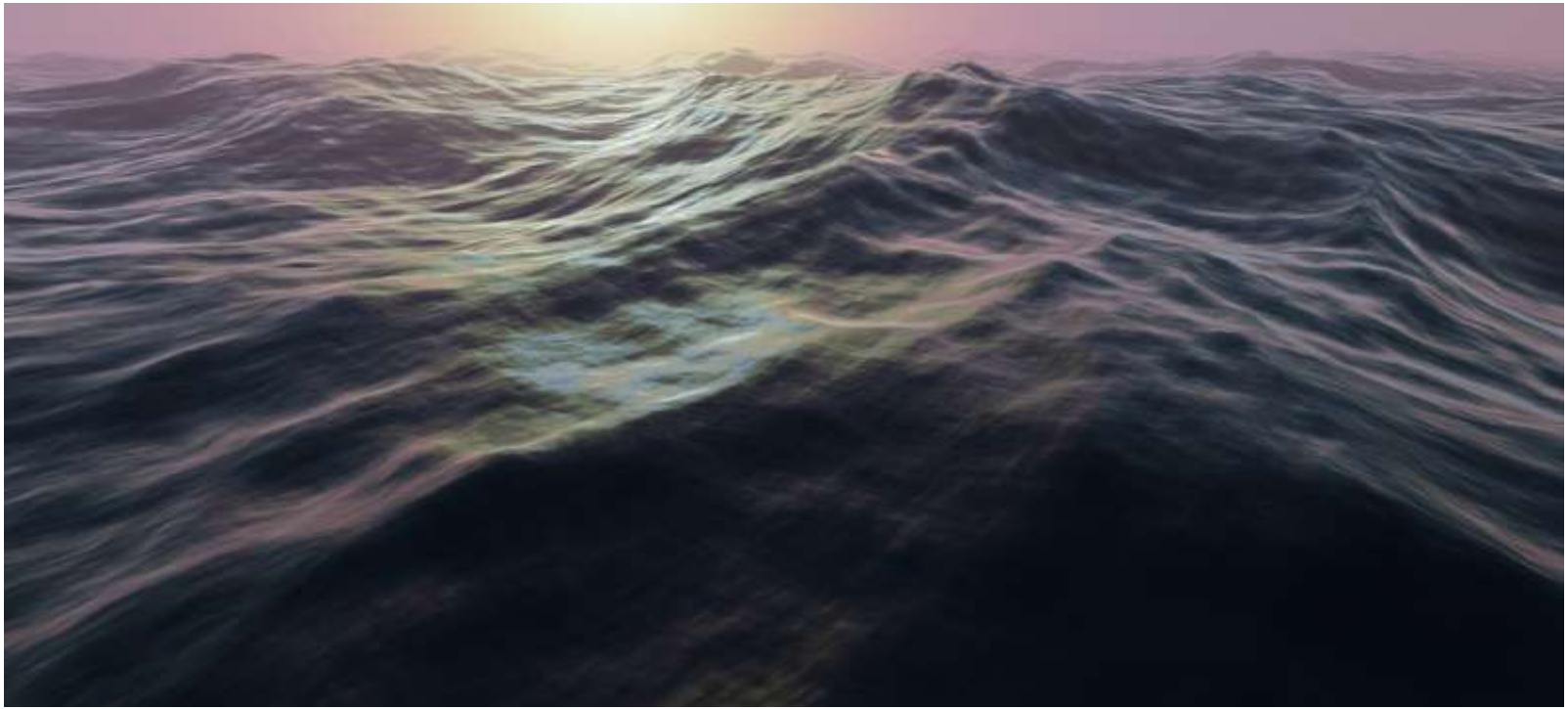
Choppy Waves

- Ocean waves do not behave quite like pure sinusoids
  - Not visually exciting
- Rather, large peaks tend to sharpen
- Significant visual improvement when applied correctly

Without Choppiness



With Choppiness



Choppiness Implementation

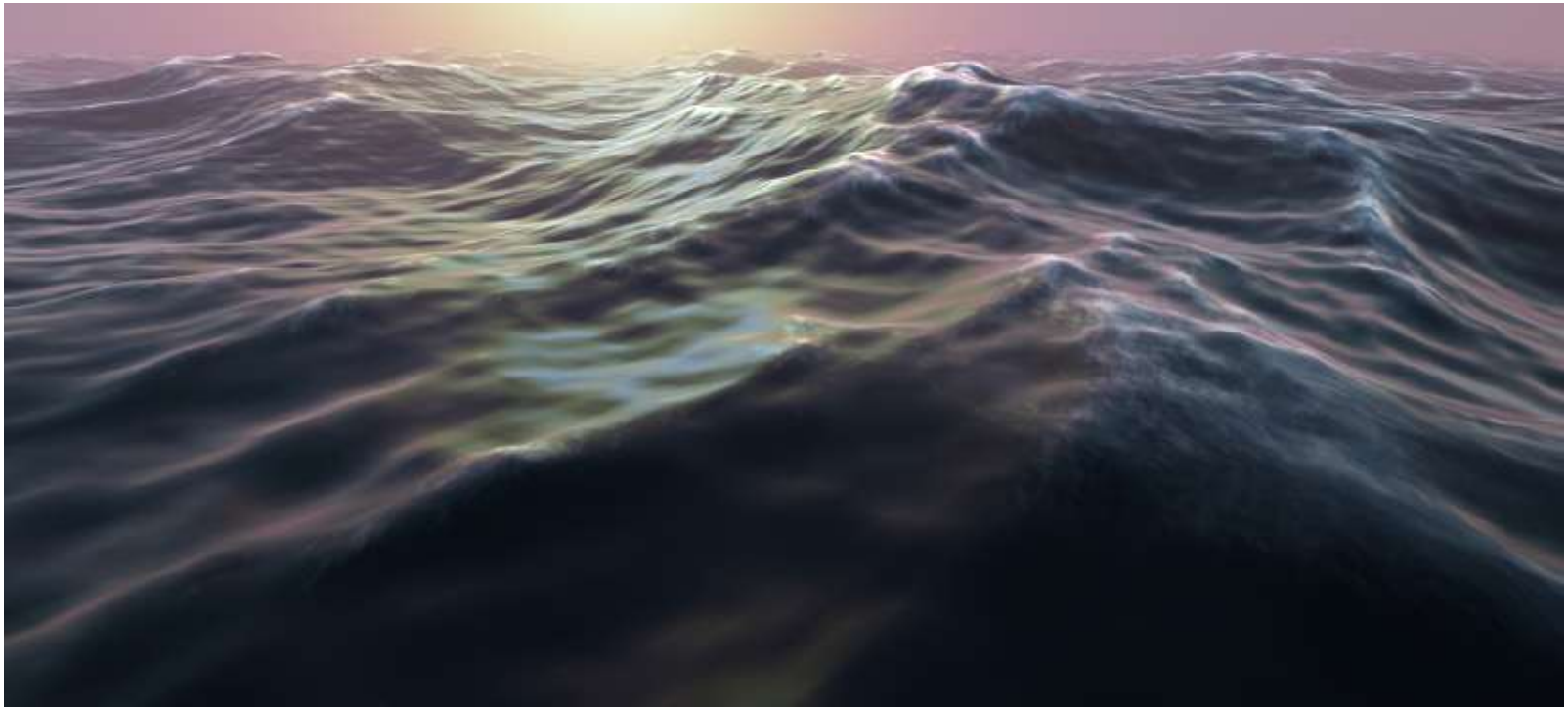
- Adding choppiness directly to Y displacement is impractical
  - Warping the heightmap mesh instead is a simple alternative
- Extend pure Y displacement to XZ displacement
- XZ displacement follows the gradient of the heightmap
  - The mesh will compact at peaks and stretch elsewhere

$$D(\mathbf{x}, t) = F^{-1}\left(-j\frac{\mathbf{k}}{k}H(\mathbf{k}, t)\right)$$

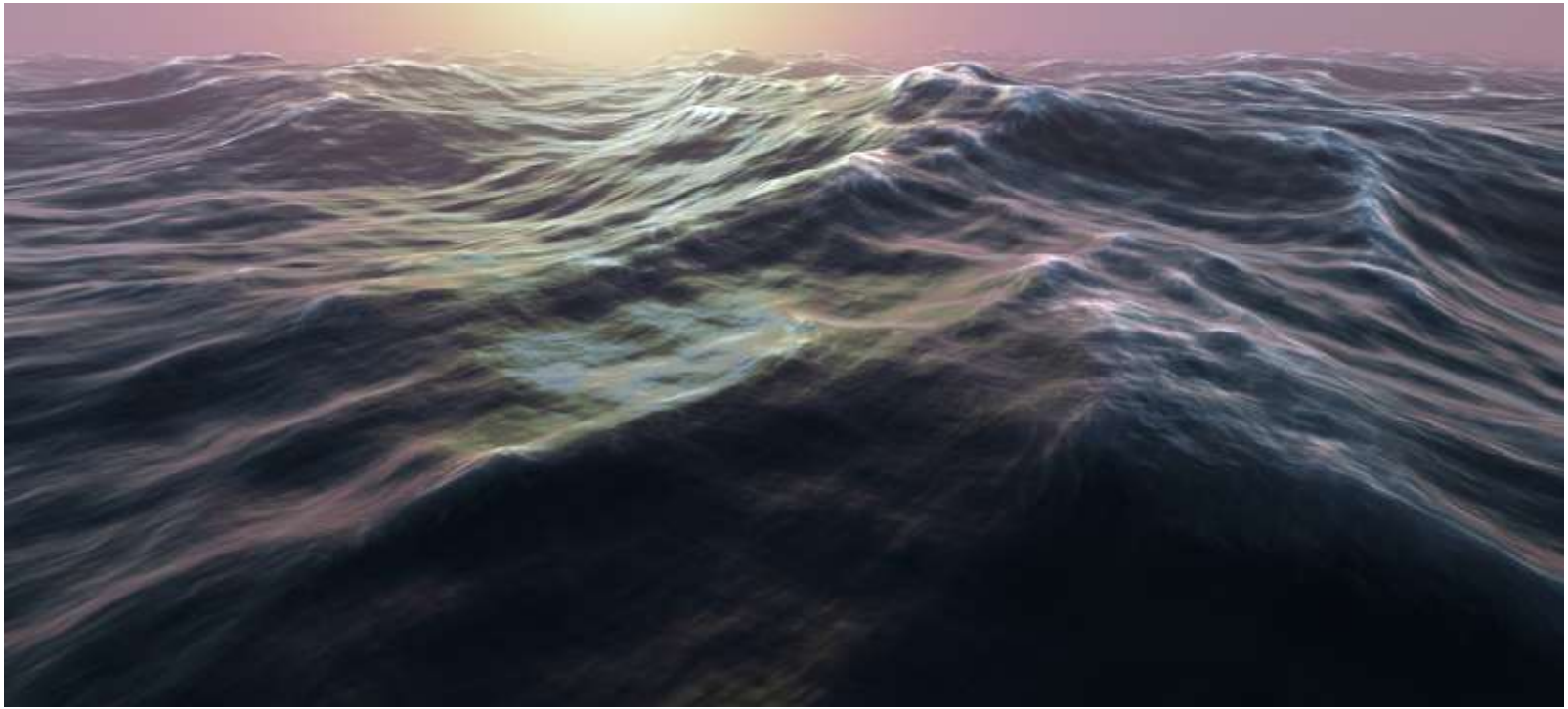
Adding a High-Frequency Normalmap

- Most of the fine details are caused by high frequency waves
- These waves contribute very little to overall displacement
- Higher frequency content should be built in a separate normalmap

Without High-Frequency Normals



With High-Frequency Normals

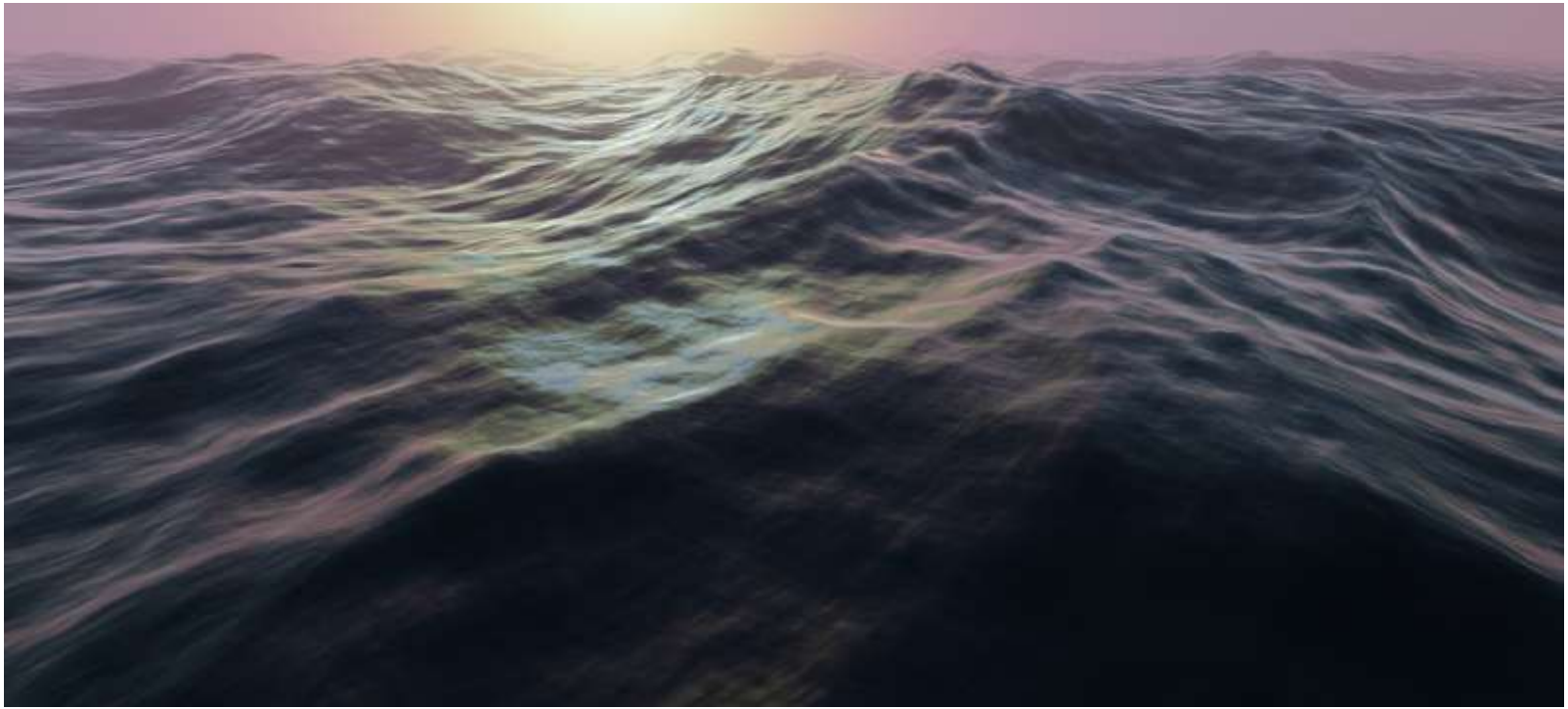


Final Touches With the Jacobian Factor

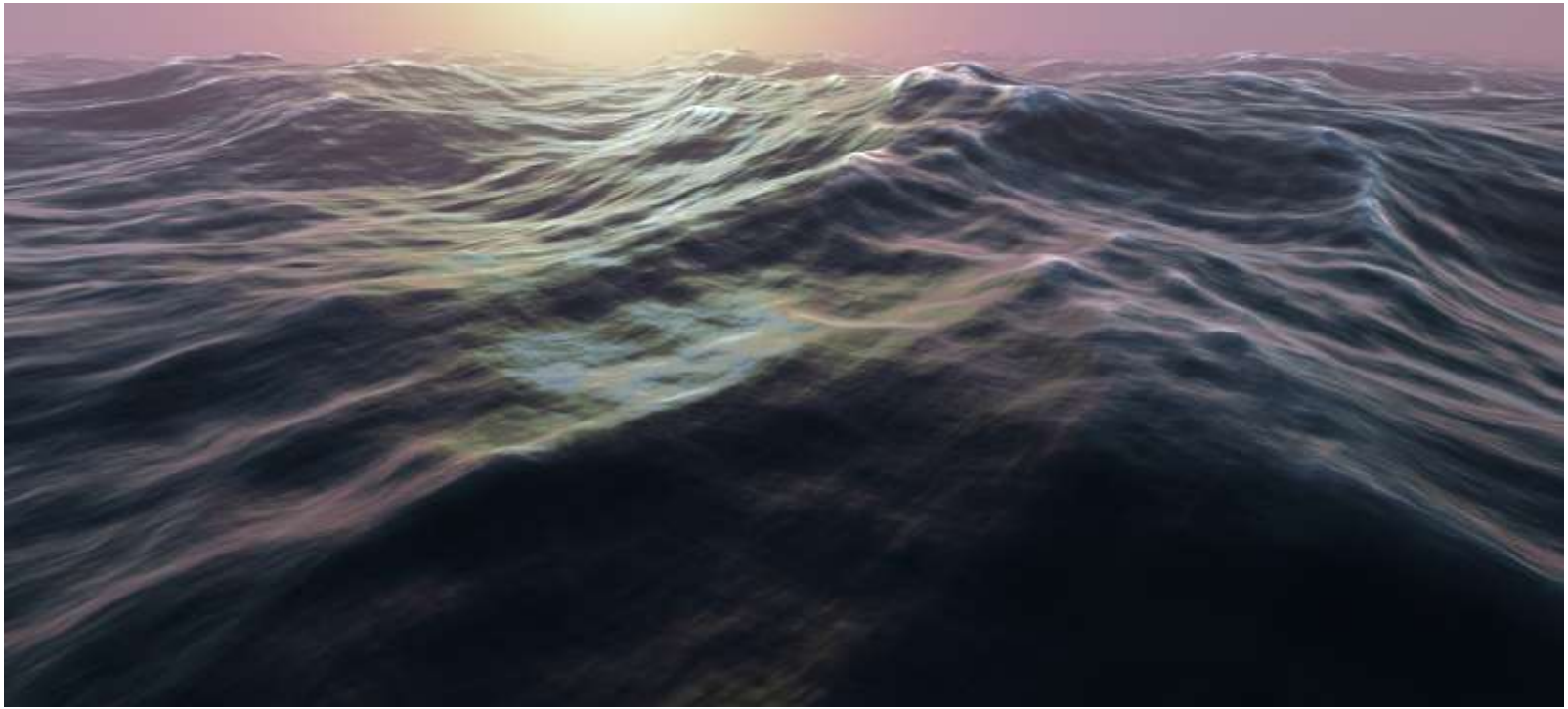
- The Jacobian factor lets the fragment shader detect ”compaction”
- Cost-effective method of adding highlights
- Modulate the low frequency Jacobian with normalmap for a more turbulent look
- Compute shader particle system ideas:
  - Spawn new particles conditionally at crests when computing Jacobian
  - Spawn new particles where intersecting with other terrain

$$J\left(x\right) = J_{xx}J_{yy} - J_{xy}J_{yx}$$

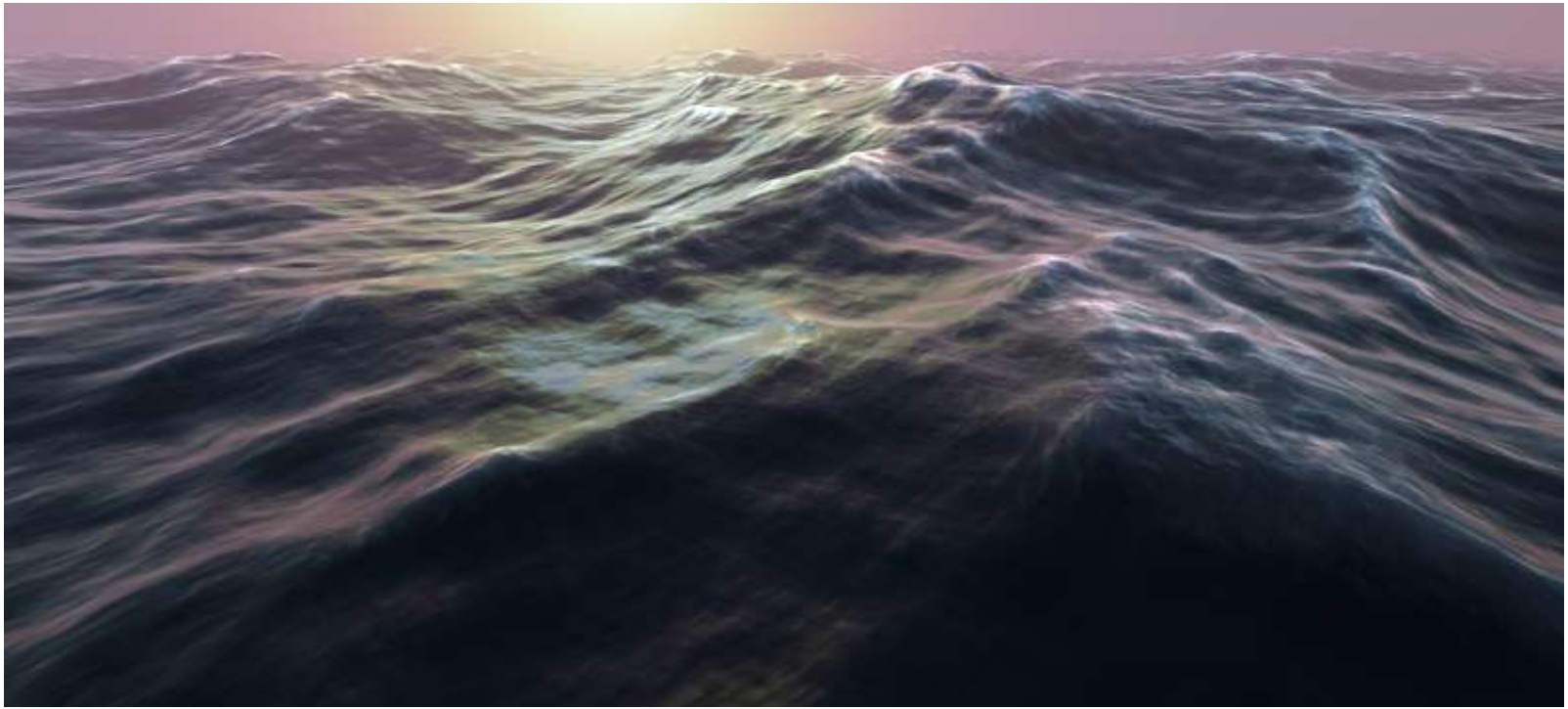
Without Jacobian Factor



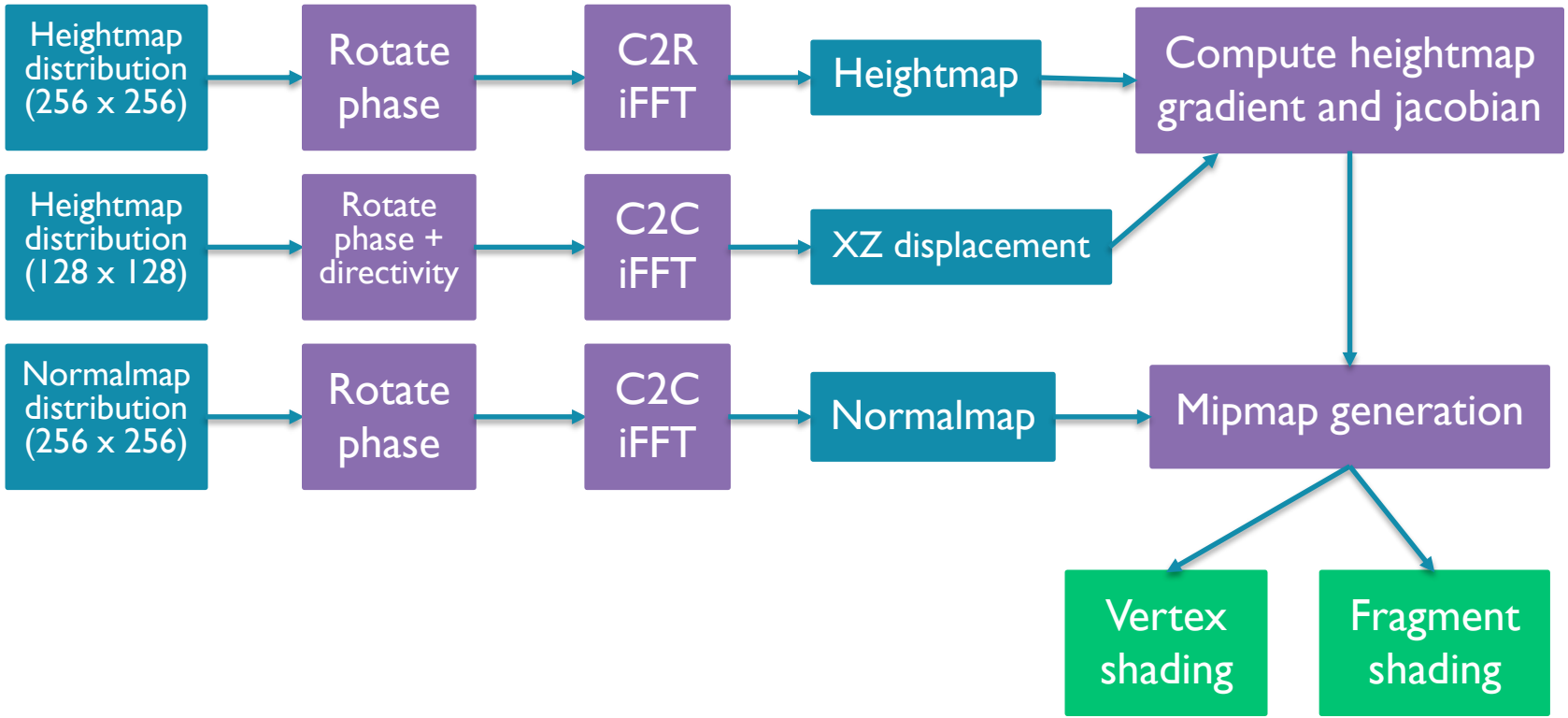
With Jacobian, No Normalmap Modulation



With Jacobian Factor



The FFT Pipeline



ARM

Efficient FFTs on Mali GPUs

- Large FFTs are highly parallel transforms
  - Butterfly stages independent within a pass
  - Efficiently implemented with compute shaders
- Number of passes order log2(N / radix)
  - More passes, more bandwidth
  - Want to use high FFT radices
  - ... But on-chip resources are finite
- Bandwidth is a real concern for GPU FFTs
  - On-chip and external resources must be carefully balanced

FFT Bandwidth Saving Techniques

- Larger radix factors
  - Does more work per pass
  - ... but register pressure kicks in quickly
- FP16
  - Mali has native FP16 support, an easy way to halve required bandwidth
  - Sufficient precision for ocean water
  - unpackHalf2x16() / packHalf2x16() to pack a complex in 32-bit uint
- Shared memory
  - Let one or more FFT passes go via shared memory
  - If sufficient space in L1 caches, even greater bandwidth reductions can be made

GLFFT Library

- OpenGL ES 3.1 and OpenGL 4.3 library for GPU FFT
  - Shameless plug ☺
  - Permissive MIT license
  - On Github
- Core features
  - 1D and 2D transforms
  - Complex-to-complex, complex-to-real and real-to-complex
  - Can benchmark itself to find optimal FFTs for particular GPUs
  - Full FP16 support
  - Large test and bench suite
  - Dual-complex (vec4) support for working with RGBA data

GLFFT Performance

- Collected on Samsung Galaxy S6 (Mali T-760 MP8)

Stage	Time
Update phases	0.5 ms
iFFT	0.8 ms
Generate gradient and jacobian	0.4 ms
Generate mipmaps	0.5 ms
Total	2.2 ms

GLFFT Performance on Very Large FFTs

- 2048x2048 C2C FP16
- Collected on Samsung Galaxy S6 (Mali T-760 MP8)

Measurement	Result
Time	27.9 ms
Arithmetic throughput	16.5 Gflop/s (estimated)
Bandwidth	7.2 GB/s (estimated)

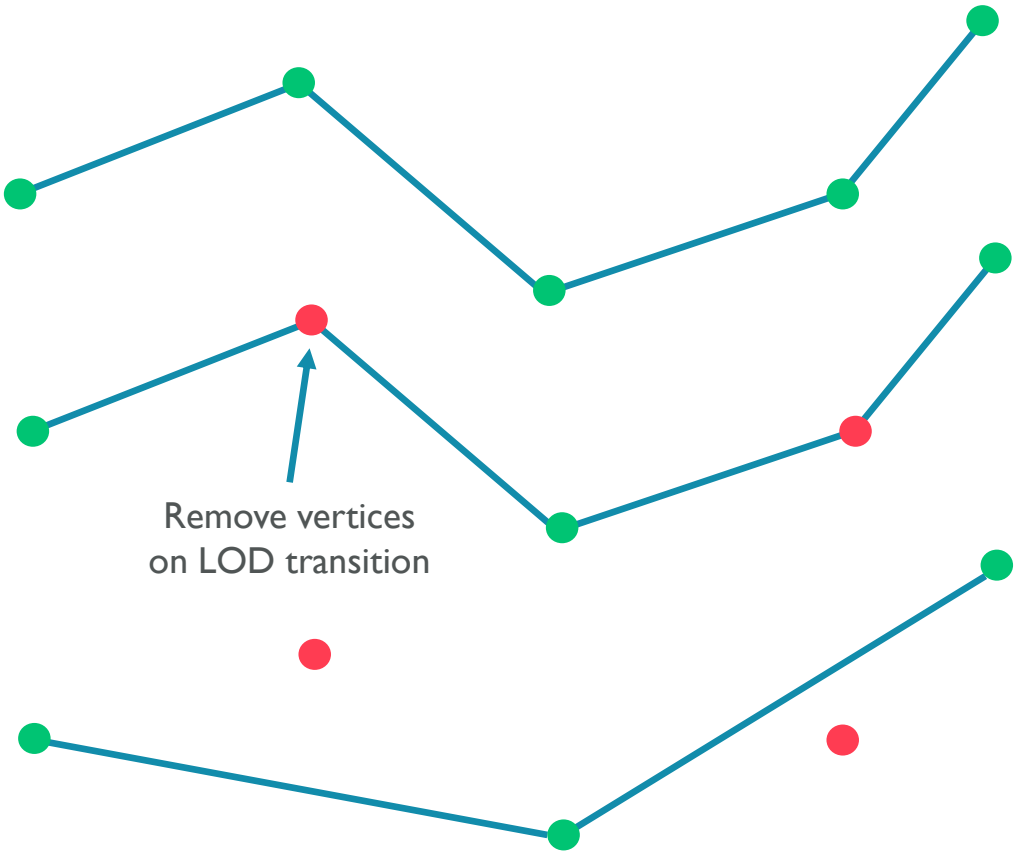
Heightmap Rendering With LOD

- Rendering a large terrain without LOD is not feasible
  - Extreme geometry load
  - Micro-triangles and aliasing
- Good mesh-based LOD is challenging
  - Decades of graphics research

Baseline For Good LOD System

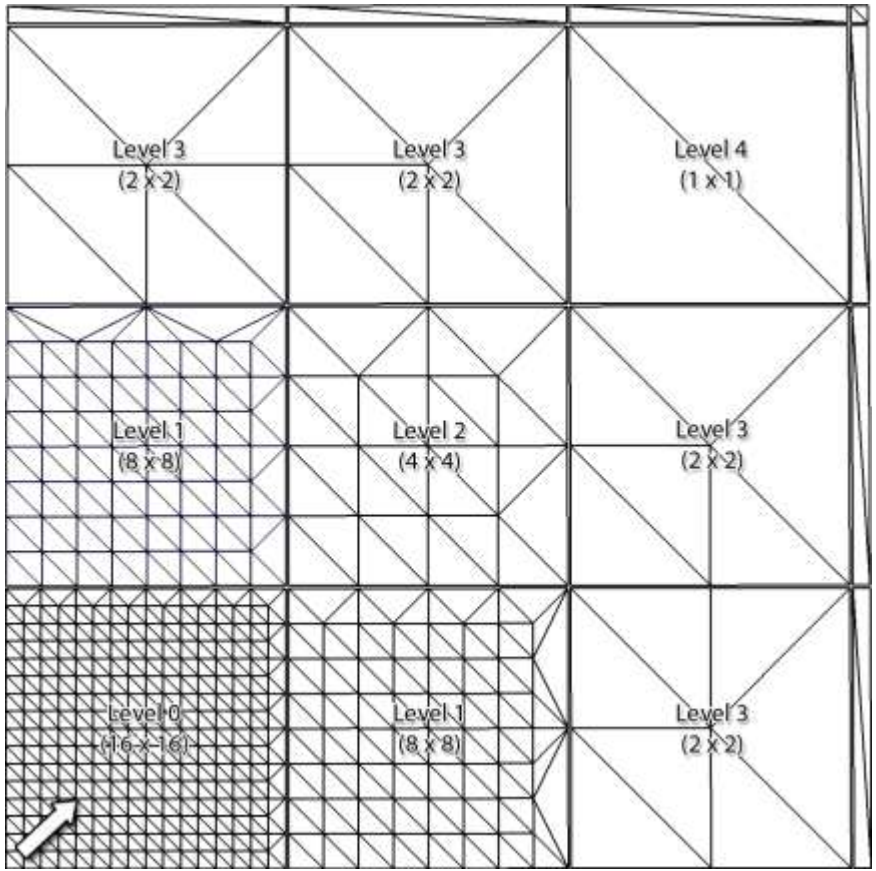
- Continuous LOD is a must-have
  - Naively removing vertices leads to discontinuous LOD (popping)
  - LOD is less noticable when transition is smooth
- Modern implementation is preferable
  - Simple and fast algorithms on CPU
  - Utilize modern GPU features
  - No run-time generation of vertices and indices on CPU

Popping Problem



Extending the Geo-MipMap Framework

Geo-MipMap



ARM

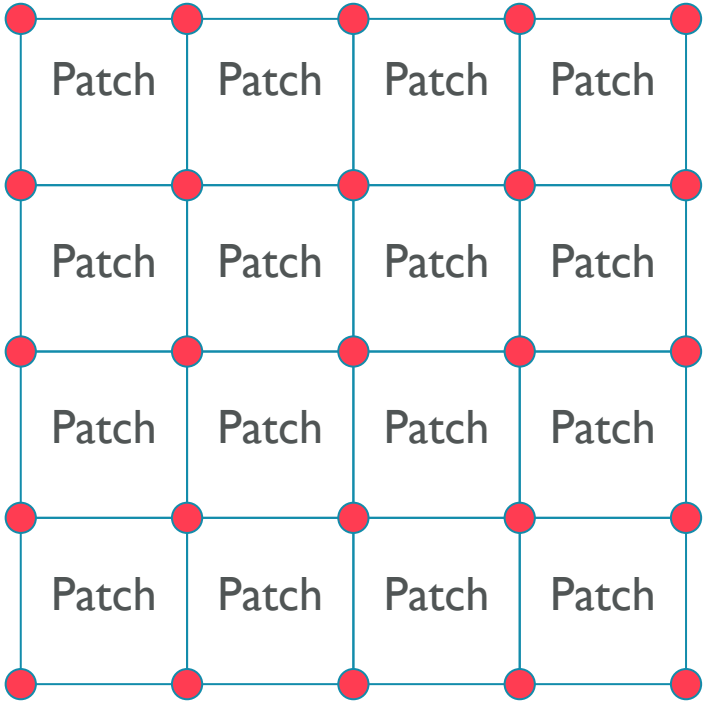
Geo-MipMap

- Well known and understood framework for LOD
  - Basic implementation is simple
- Split the world into large patches
- Adaptively subdivide this patch depending on a metric
  - Assign an LOD to every patch independently
- However, this is not the year 2000 ...
  - The technique is showing its age
  - Does not solve popping artifacts

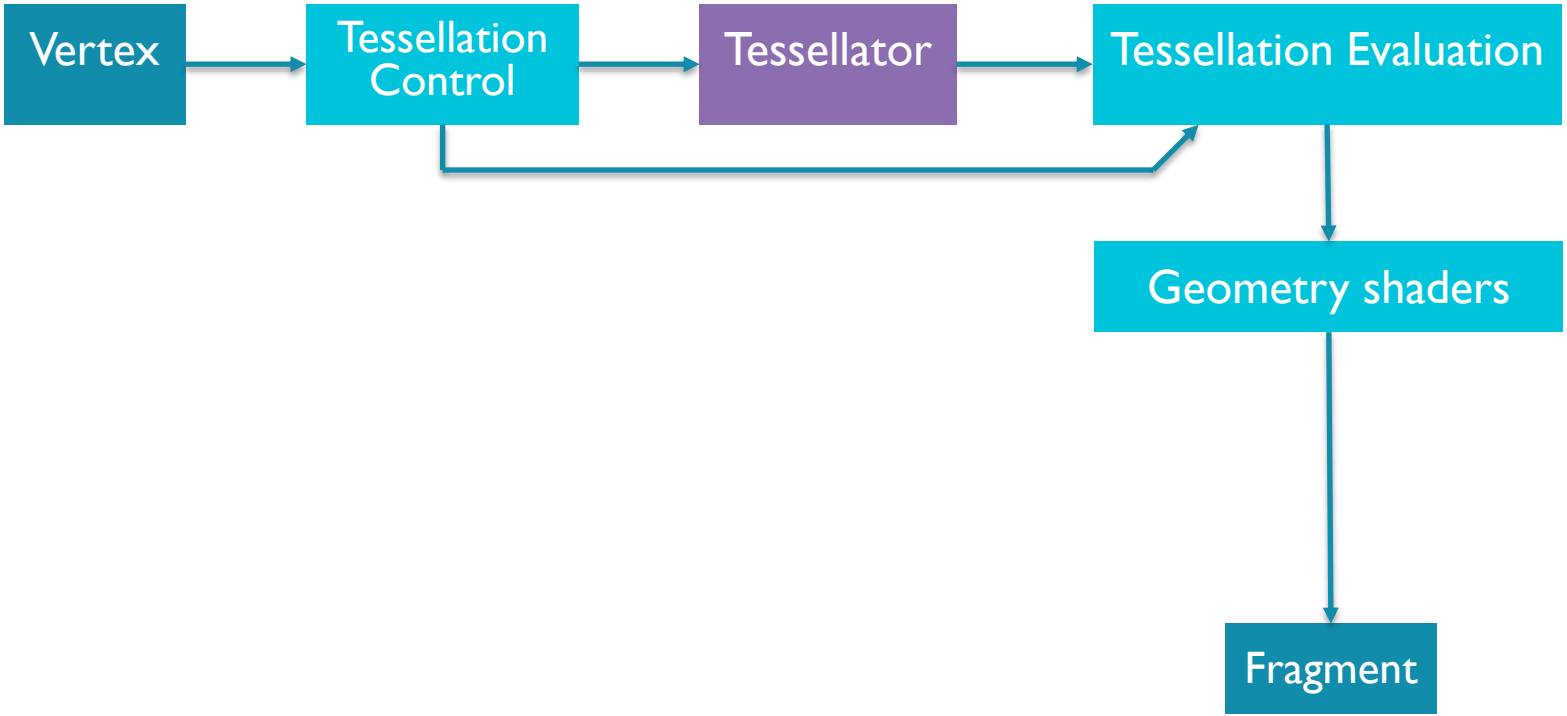
Tessellation, Now in OpenGL ES 3.2

Tessellation

- Built-in concept of a patch which is adaptively subdivided.

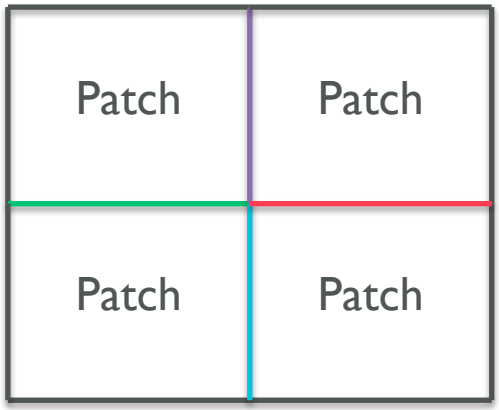


OpenGL ES 3.2 Graphics Pipeline



Control Shader

- Frustum cull patch
- Compute LOD in four corners
- Select outer tessellation levels based on two corners which touch the edge
  - Critical, otherwise cracks will appear
- Inner level is somewhat arbitrary
  - Vertices are not shared with other patches
  - Maximum or average of corners is acceptable

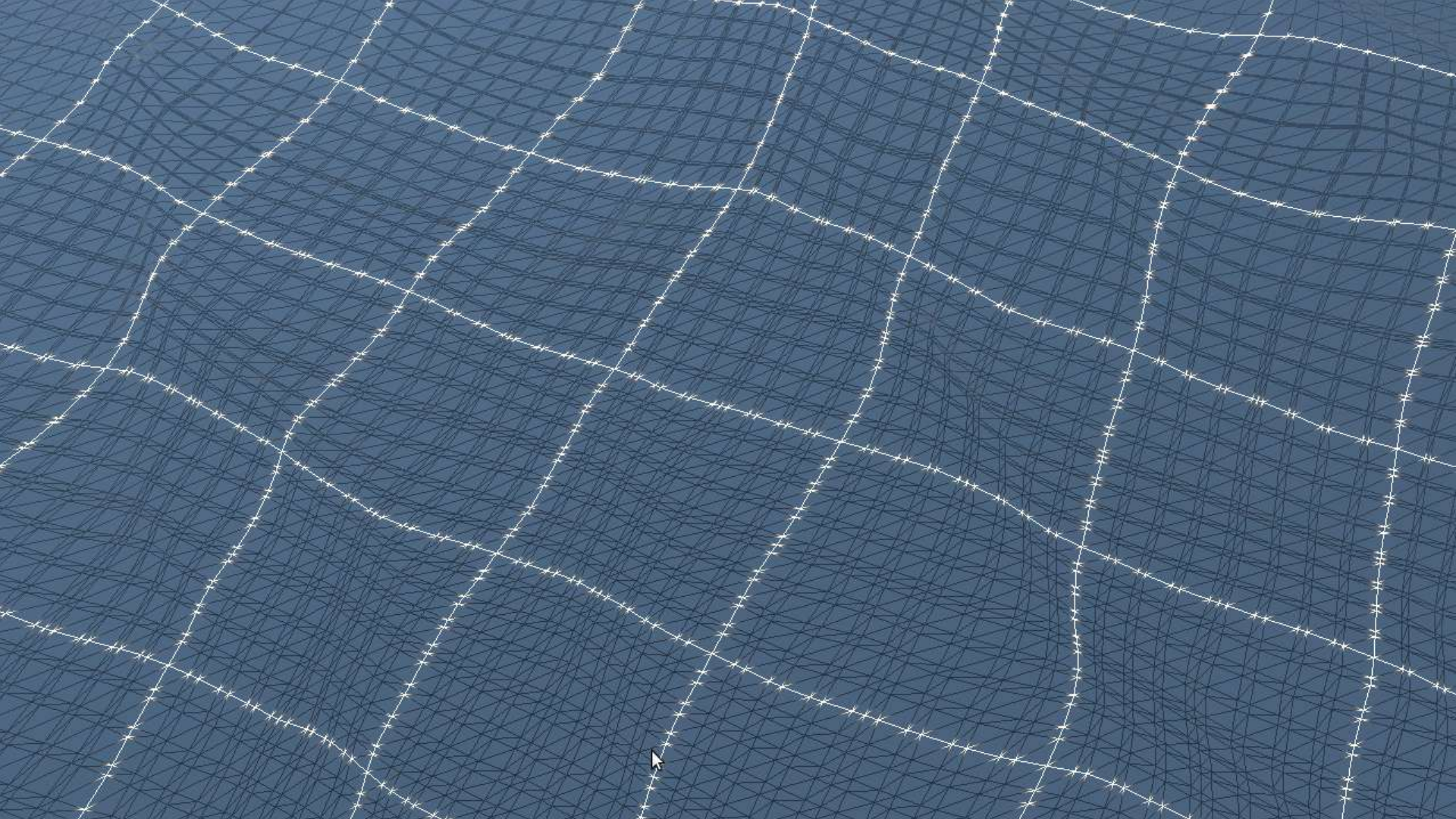


Evaluation Shader

- Interpolate corner data based on gl_TessCoord
  - LOD factor for sampling heightmap
  - Patch-local position
- Fractional even spacing subdivision mode
  - Better match with texture sampling
- Sample heightmap texture with interpolated LOD

Continuous LOD Morphing Geo-MipMap

- Takes inspiration from many different approaches to LOD
  - Vertex Shader Geomorphing
  - Tessellation
  - CDLOD
- Provides OpenGL ES 3.0 alternative to quad patch tessellation
  - Special case of tessellation
  - Simplifies sufficiently for use in pure vertex shaders



Core Approach

- Pre-tessellate quad meshes at 2, 4, 8, 16, 32, 64, ... tessellation factors
- At fractional LODs, odd vertices slide towards even vertices
  - At full morph, mesh is exactly the same as a lower quality LOD
  - Same principle as CDLOD method
- No stitching meshes
  - Minimal number of unique meshes

Edge-Cases

- Mesh LOD factor is not necessarily equal to outer levels on edges
- Like Tessellation, patches sharing an edge must agree on LOD
  - Necessary to use maximum LOD
  - One of 5 possibilities (4 edges or inner) selected with branchless logic

```
// Branch-less selection
// aLODWeights either all 0 or single element set to 1.
in vec4 aLODWeights;
bool innerVertex = all(equal(aLODWeights, vec4(0.0)));
float lod = dot(aLODWeights, EdgeLODs);
lod = mix(lod, InnerLOD, innerVertex);
```

Adapt Mesh Grid to Any LOD

- Patches can have an arbitrary LOD
  - In turn, edges can have an arbitrary LOD
  - Not restricted by quad-tree structure, unlike CDLOD
- Snap to LOD and lerp
  - Adapts popping-free to any LOD greater than mesh LOD

```
// Snap to grid corresponding to floor(lod) and ceil(lod).
uvec2 mask = (uvec2(1u) << uvec2(ufloor_lod, ufloor_lod + 1u)) - 1u;
// Round towards center of patch.
uvec4 rounding = aPosition.zwzw * mask.xxyy;
vec4 lower_upper_snapped = vec4((aPosition.xyxy + rounding) & ~mask.xxyy);

// Then lerp between them to create a smoothly morphing mesh.
return mix(lower_upper_snapped.xy, lower_upper_snapped.zw, fract_lod);
```

Mali OpenGL ES SDK for Android

- New code samples and tutorials
  - Ocean
  - OVR_multiview
  - Tessellation
  - Geometry shaders
  - Multisampled framebuffers

For more information visit the Mali Developer Centre:

http://malideveloper.arm.com

- Revisit this talk in PDF and audio format post event
- Download tools and resources

Thank you!

ARM

Questions?

References

- Mali OpenGL ES SDK for Android
  - http://malideveloper.arm.com/resources/sdks/mali-opengl-es-sdk-for-android/
- GLFFT
  - https://github.com/Themaister/GLFFT
- Simulating Ocean Water
  - http://graphics.ucsd.edu/courses/rendering/2005/jdewall/tessendorf.pdf
- CDLOD
  - http://www.vertexasylum.com/downloads/cdlod/cdlod_latest.pdf
- Geo-MipMapping
  - http://www.flipcode.com/archives/article_geomipmaps.pdf