Cutting Edge Graphics for Android

Mathias Schott, Senior Developer Technology Engineer, NVIDIA

Cutting Edge Graphics for Android

YOUR EFFECTS & CREATIVITY

TEGRA K1/X1

OPENGL ES 3.1

AEP

ANDROID EXTENSION PACK

OpenGL ES 3.1 and AEP

OpenGL ES 3.1

arrays of arrays, compute shaders, indirect draw commands, explicit uniform location, atomic counters, support for framebuffers with no attachments, program interface queries, shader helper invocation, shader image load/store operations, shader layout binding, shader storage buffer objects, separate shader objects, stencil texturing, texture gather operations, multisample formats for immutable textures, shader bitfield operations, vertex attribute binding

GL_ANDROID_extension_pack_es31a

ASTC texture compression, advanced blend equations, per-sample shading, texture buffers, "shader model 5", per attachment blend state, stencil only texturing, geometry shaders, texture border clamp mode, multi-sampled texture arrays, image atomics, texture copies, texture cubemap arrays, tessellation shaders, per attachment blend state, debug support, per texture SRGB decode enable

Overview

eye adaptation

screen space reflections

geometric detail

images rendered with Epic Games' Unreal Engine 4

Programmability

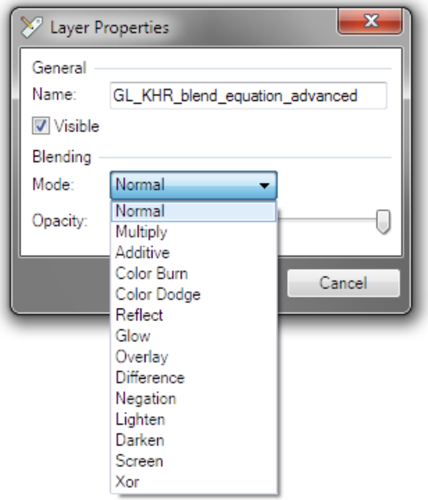
- “Shader Model 5” across all shader stages
- complete set of graphics and compute stages
- full support for integer types/textures/operations
- textures
  - 2D[Array], 2DMS[array], cubemap[array], 3D, buffer
  - fixed point, (unsigned) integer, float
- images
  - aka “read write textures”/UAVs
  - atomic operations

Productivity

- debug support
  - create debug context
  - register callback
  - get detailed errors & warning messages
- copies between textures & renderbuffer
  - single API call CopyImageSubDataEXT
  - can also convert between formats, too ☺
- ASTC Texture Compression
  - https://developer.nvidia.com/astc-texture-compression-for-game-assets

AEP is not just for 3D

```
Mode                      Blend Coefficients
--------------------      -----------------------------------
MULTIPLY_KHR              (X,Y,Z)  = (1,1,1)
                          f(Cs,Cd) = Cs*Cd

SCREEN_KHR                (X,Y,Z)  = (1,1,1)
                          f(Cs,Cd) = Cs+Cd-Cs*Cd

OVERLAY_KHR               (X,Y,Z)  = (1,1,1)
                          f(Cs,Cd) = 2*Cs*Cd, if Cd <= 0.5
                                     1-2*(1-Cs)*(1-Cd), otherwise

DARKEN_KHR                (X,Y,Z)  = (1,1,1)
                          f(Cs,Cd) = min(Cs,Cd)

LIGHTEN_KHR               (X,Y,Z)  = (1,1,1)
                          f(Cs,Cd) = max(Cs,Cd)

COLORDODGE_KHR            (X,Y,Z)  = (1,1,1)
                          f(Cs,Cd) =
                            0, if Cd <= 0
                            min(1,Cd/(1-Cs)), if Cd > 0 and Cs < 1
                            1, if Cd > 0 and Cs >= 1

COLORBURN_KHR             (X,Y,Z)  = (1,1,1)
                          f(Cs,Cd) =
                            1, if Cd >= 1
                            1 - min(1,(1-Cd)/Cs), if Cd < 1 and Cs > 0
                            0, if Cd < 1 and Cs <= 0

HARDLIGHT_KHR             (X,Y,Z)  = (1,1,1)
                          f(Cs,Cd) = 2*Cs*Cd, if Cs <= 0.5
                                     1-2*(1-Cs)*(1-Cd), otherwise

SOFTLIGHT_KHR             (X,Y,Z)  = (1,1,1)
                          f(Cs,Cd) =
                            Cd-(1-2*Cs)*Cd*(1-Cd),
                              if Cs <= 0.5
                            Cd+(2*Cs-1)*Cd*((16*Cd-12)*Cd+3),
                              if Cs > 0.5 and Cd <= 0.25
                            Cd+(2*Cs-1)*(sqrt(Cd)-Cd),
                              if Cs > 0.5 and Cd > 0.25

DIFFERENCE_KHR            (X,Y,Z)  = (1,1,1)
                          f(Cs,Cd) = abs(Cd-Cs)

EXCLUSION_KHR             (X,Y,Z)  = (1,1,1)
                          f(Cs,Cd) = Cs+Cd-2*Cs*Cd
```



Physically Based Rendering Primer

- industry trend in film and recently
  - UE4, Unity5, CryEngine 3.6, Frostbite …
- based on physics of light transport
  - physical quantities (lumens, Candela, …)
- less "ad-hoc" than Blinn-Phong
  - consistent + plausible lighting conditions
- high dynamic range
  - input data
  - render targets
  - tonemap to fixed point for display

Physically Based Material Model

- metallic + non-metallic surfaces
- glossiness vs roughness
- energy conserving -> plausible highlights

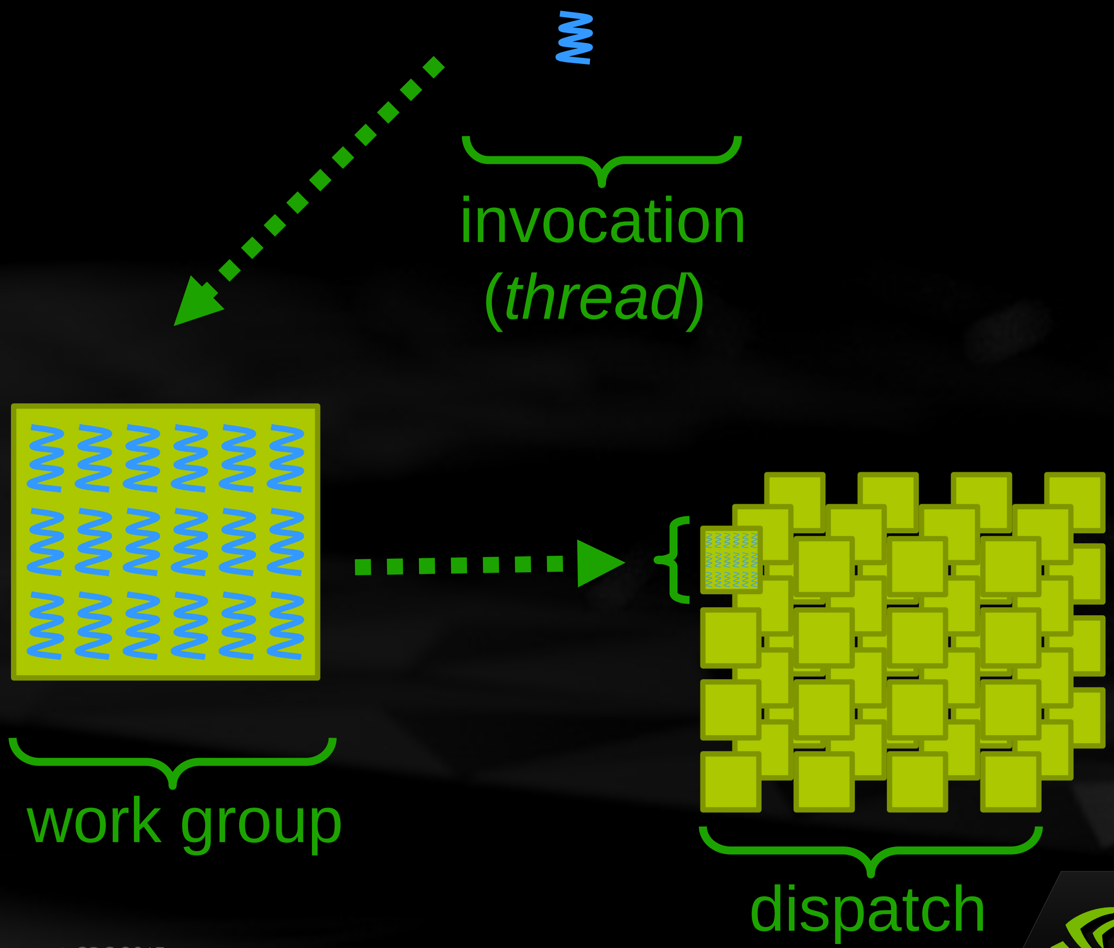
Physically Based Lighting Model

- IES light profiles represent
  - different light bulbs
  - reflections in light fixtures

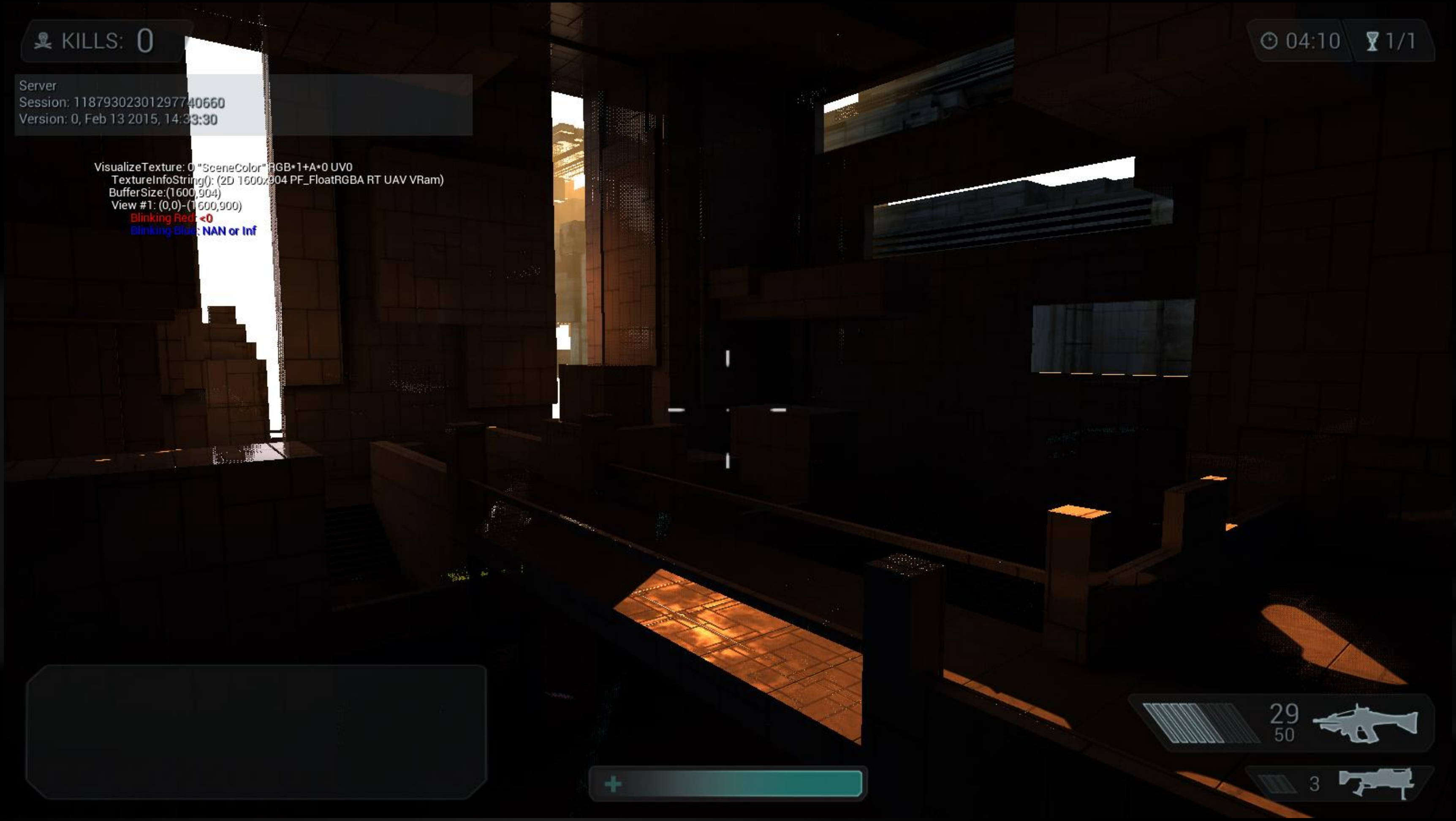
Eye adaptation

Compute Shader Primer

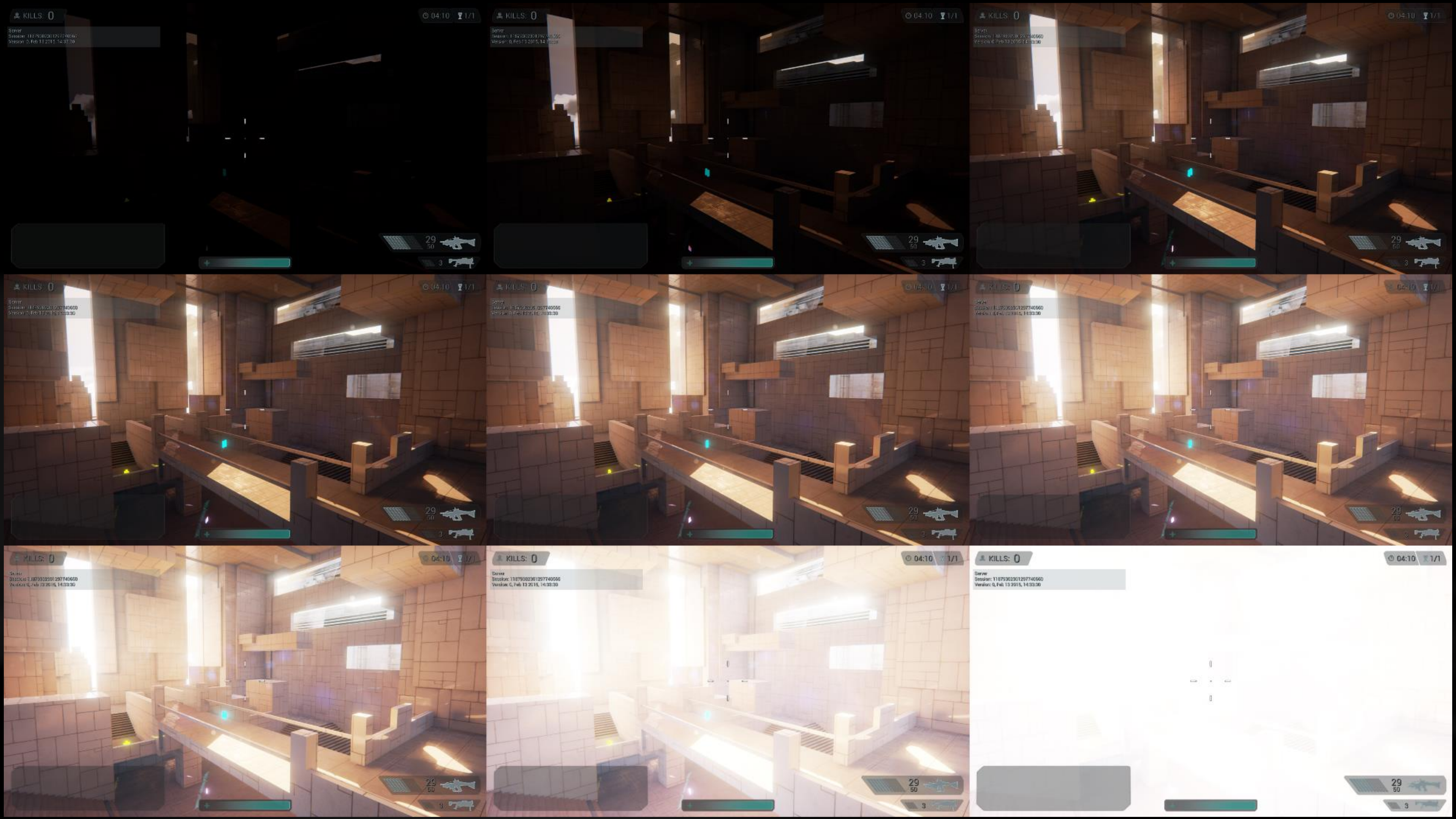
- abstract execution model
- conceptually: "groups of threads execute code in parallel"
- hierarchical thread structure
  - threads in groups in dispatches
- threads in work group
  - share data & sync
- no fixed inputs & outputs
- access to OpenGL resources
  - sample textures
  - read/write images
  - read/write buffers
  - atomic counters
  - AEP: image atomics



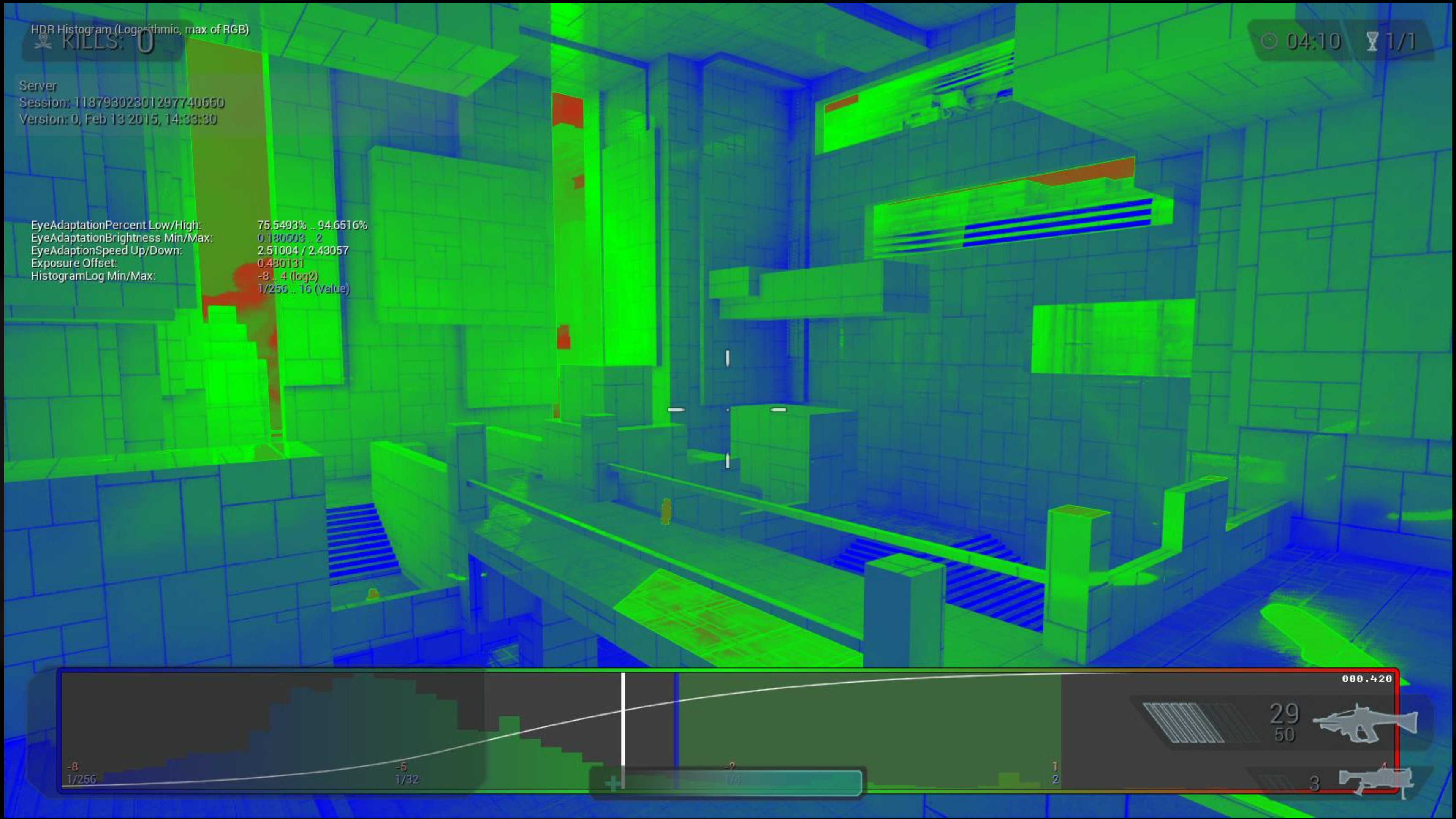
HDR Scene Color



Tonemap for display



Brightness histogram



Average Brightness from Histogram

Screen Space Reflections

Cubemap Array Primer

- Cube map
  - 6 faces of a cube
  - directional texture lookup
- Cube map array
  - array of cube maps
  - additional “layer” integer texture coordinate

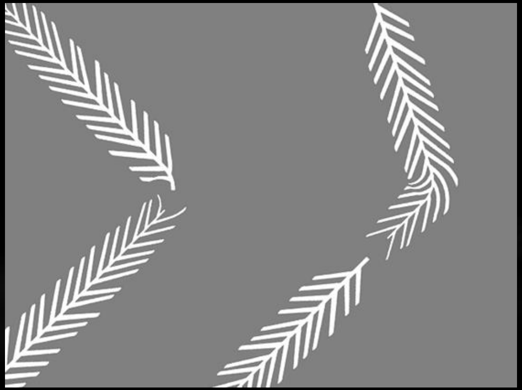
layer = 0

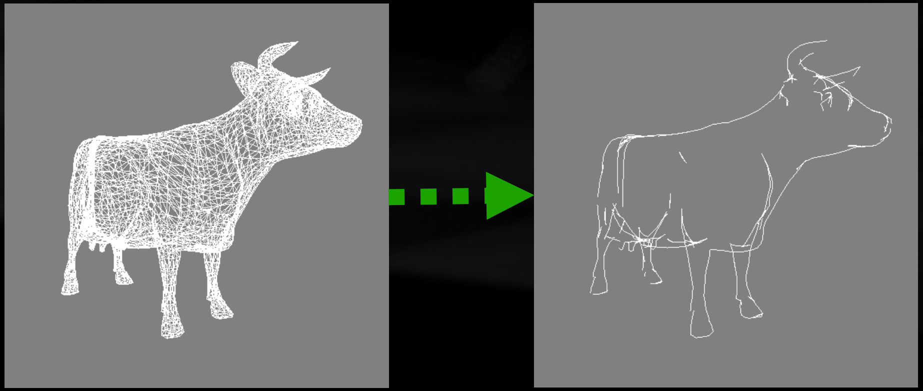
layer = 1

layer = 2

Geometry Shader Primer

- input
  - whole triangle, line, point
- output
  - >= 0 points, line strips, triangle strips
- use cases
  - billboards & point sprites, wide lines
  - silhouette detection
- warning:
  - not intended for arbitrary tessellation
- also adds “gl_Layer” built-in
  - select layer of FBO attachment to render into





Capturing the Reflection Environment

Probe 1

VisualizeTexture: 34 "ReflectionEnvs" RGB*1+A*0 UV0
TextureInfoString(): (Cube[3] 128 PF_FloatRGBA)
BufferSize:(1600,904)
View #1: (0,0)-(1600,900)
Blinking Red: <0
Blinking Blue: NAN or Inf

Probe 2


VisualizeTexture: 34 "ReflectionEnvs" RGB*1+A*0 UV0
TextureInfoString(): (Cube[3] 128 PF_FloatRGBA)
BufferSize:(1600,904)
View #1: (0,0)-(1600,900)
Blinking Red: <0
Blinking Blue: NAN or Inf

Probe 3


VisualizeTexture: 34 "ReflectionEnvs" RGB*1+A*0 UV0
TextureInfoString(): (Cube[3] 128 PF_FloatRGBA)
BufferSize:(1600,904)
View #1: (0,0)-(1600,900)
Blinking Red: <0
Blinking Blue: NAN or Inf

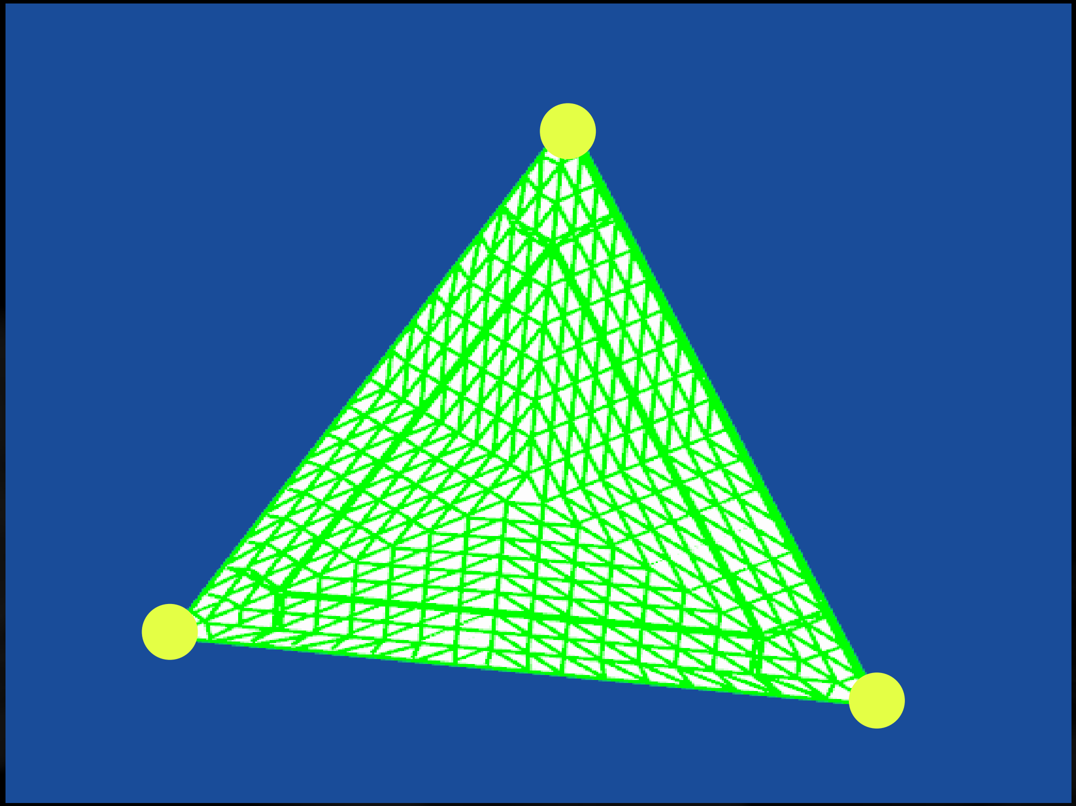
Approximate reflections…

… and combine with shading

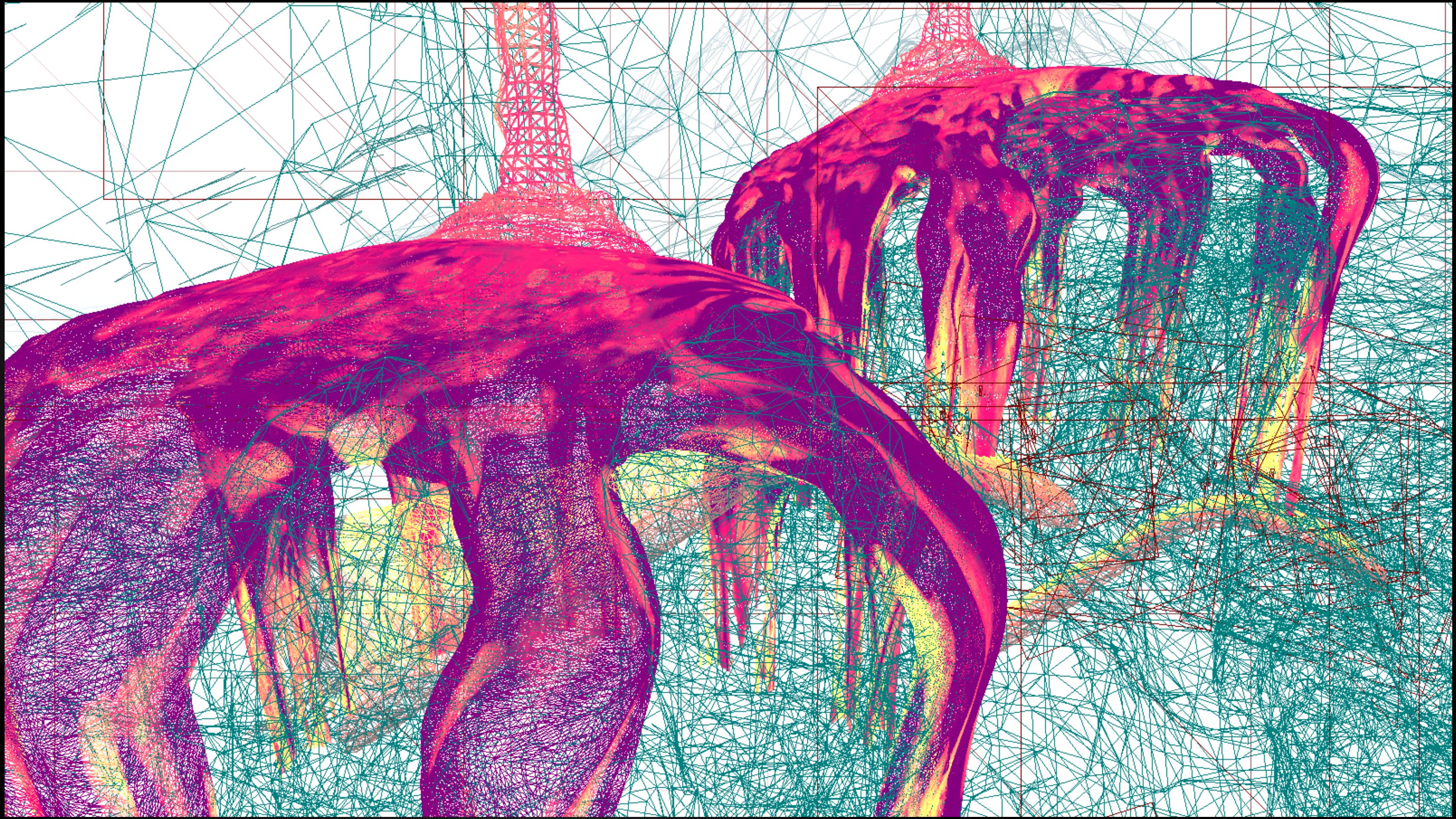
Geometric detail

Tessellation Primer

- patch primitive
  - triangles, quad, iso lines
- tessellation control shader
  - e.g. computes level of detail
- fixed function tessellator
  - generates primitives (using LOD)
- tessellation evaluation shader
  - evaluates position of tessellated vertex
  - Lighting, texcoord transformation …



With “real” content …



… and Shading!

Porting from OpenGL 4

OpenGL 4	OpenGL ES 3.1 + AEP
Direct equivalent	
glBlendFuncSeparatei	glBlendFuncSeparatei**EXT**
GL_DEBUG_OUTPUT	GL_DEBUG_OUTPUT**_KHR**
Functional equivalent	
glFramebufferTexture3D	glFramebufferTexture**Layer**
glColorMaskIndexed	glColorMask**iEXT**
missing	
glPolygonMode	geometry shader for some use cases
glBindFragDataLocation	specify in shader

Porting fragment shaders

OpenGL 4

```
#version 430



uniform vec4 scale;
uniform sampler3D tex;
out vec4 out_value;
void main()
{
    out_value = scale * texture(tex,…);
}
```

OpenGL ES3.1 + AEP

```
#version 310 es
#extension
GL_ANDROID_extension_pack_es31a: enable
precision highp float;
uniform vec4 scale;
uniform highp sampler3D tex;
layout(location = 0) out vec4 out_value;
void main()
{
    out_value = scale * texture(tex,…);
}
```

```
glBindFragDataLocation (glGetUniformLocation(“out_value”),0);
```

Porting compute shaders

OpenGL 4

```
#version 430



layout (local_size_x = 64) in;
uniform uvec4 value;
uniform writeonly layout(r32ui) uimageBuffer
image_buf;
void main()
{
    uint index = gl_GlobalInvocationID.x;
    imageStore(image_buffer, index, value.x);
}
glUniformi(glGetUniformLocation("image_buf"),0);
```

OpenGL ES3.1 + AEP

```
#version 310 es
#extension GL_ANDROID_extension_pack_es31a :
enable
// precision highp float; default for compute
layout (local_size_x = 64) in;
uniform highp uvec4 value;
uniform writeonly layout(r32ui, binding = 0)
highp uimageBuffer image_buf;
void main()
{
    highp uint index = gl_GlobalInvocationID.x;
    imageStore(image_buffer, index, value.x);
}
```

SHIELD considerations

- high resolution displays but modest GPU
  - upscale scene + composite with native resolution UI
  - turn some features back (post processing)
- modest CPU
  - tone down cpu-hungry rendering
    - shadows
  - appropriate amount of API & draw calls
- modest memory
  - remove duplicate assets
  - texture compression

Questions? Thank you!

GameWorks

- Get the latest information for developers from NVIDIA and continue the discussion
- gameworks.nvidia.com