SIGGRAPH2012

The **39th** International **Conference** and **Exhibition** on **Computer Graphics** and **Interactive Techniques**

Unity: iOS and Android - Cross-Platform Challenges and Solutions

Renaldas Zioma

Unity Technologies

SIGGRAPH2012

Mobile devices today

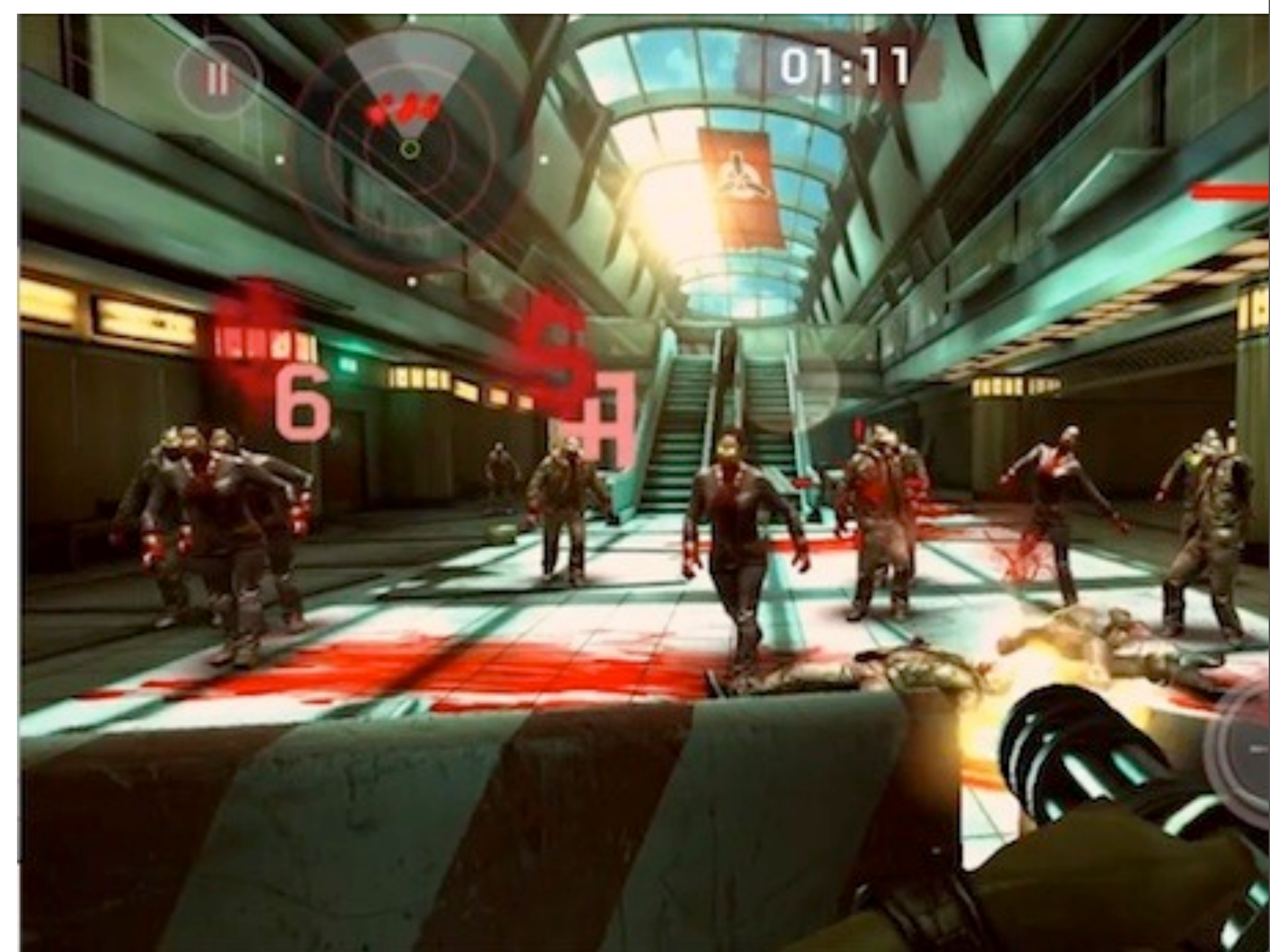
- Can render ...

Dead Trigger courtesy of MadFingerGames

®MADFINGER

Mobile devices today

- Can render ...



Dead Trigger courtesy of MadFingerGames

MADFINGER

Mobile devices today

- Can render this @ 2048 x 1536

CSR Racing courtesy of BossAlien & NaturalMotion

Mobile Platform Challenges

- Different **GPU architectures**
  - API extensions
- Screen **resolutions**
- **Performance** scale
- **Drivers**
- Texture **formats**

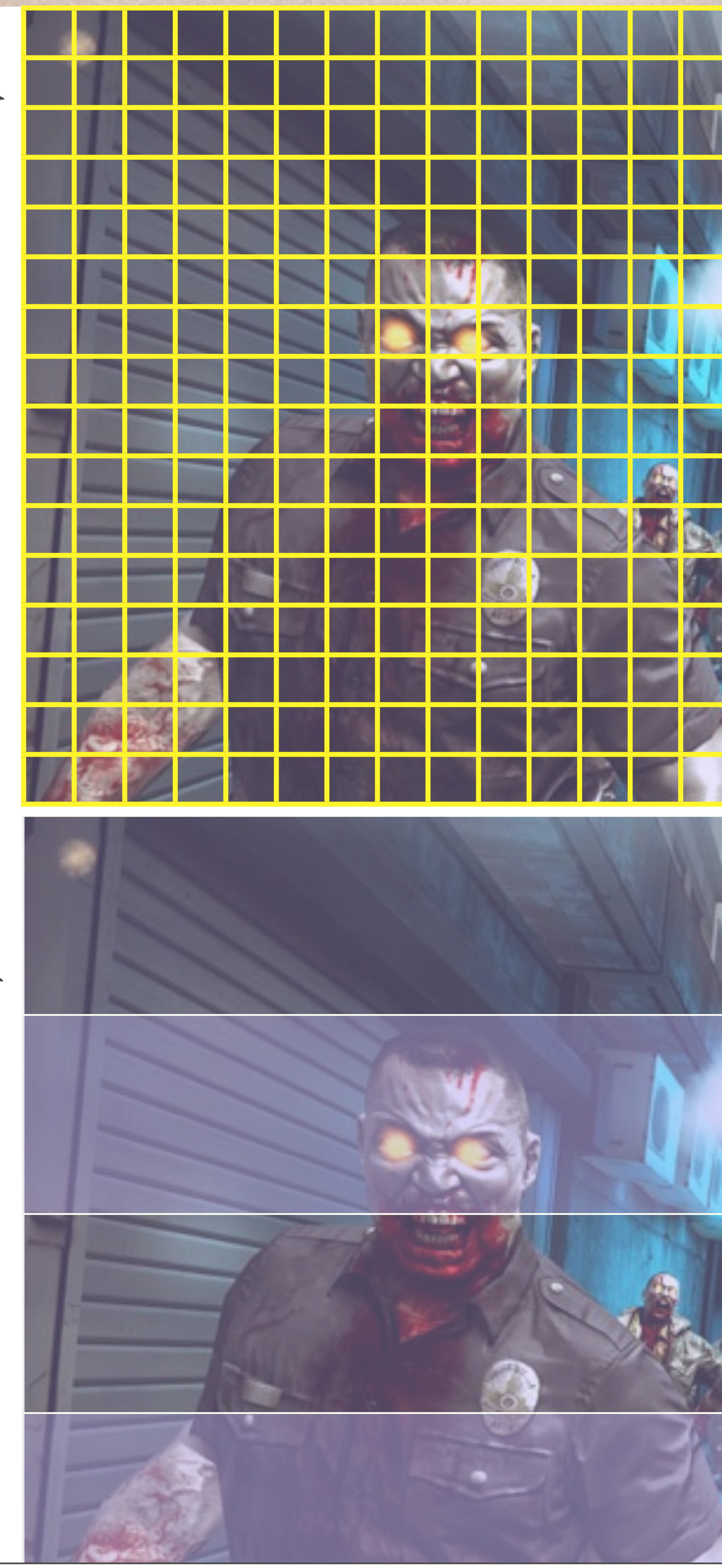
4 (or 5) GPU Architectures

SIGGRAPH2012

- ImgTech PowerVR **SGX** - **TBDR** (TileBasedDeferred)
  - ImgTech PowerVR **MBX** - **TBDR** (Fixed Function)
- ARM **Mali** - **Tiled** (small tiles)
- Qualcomm **Adreno** - **Tiled** (large tiles)
  - Adreno3xx - can switch to Traditional
  - glHint(GL_BINNING_CONTROL_HINT_QCOM, GL_RENDER_DIRECT_TO_FRAMEBUFFER_QCOM)
- NVIDIA **Tegra** - **Traditional**

Tiled Architecture

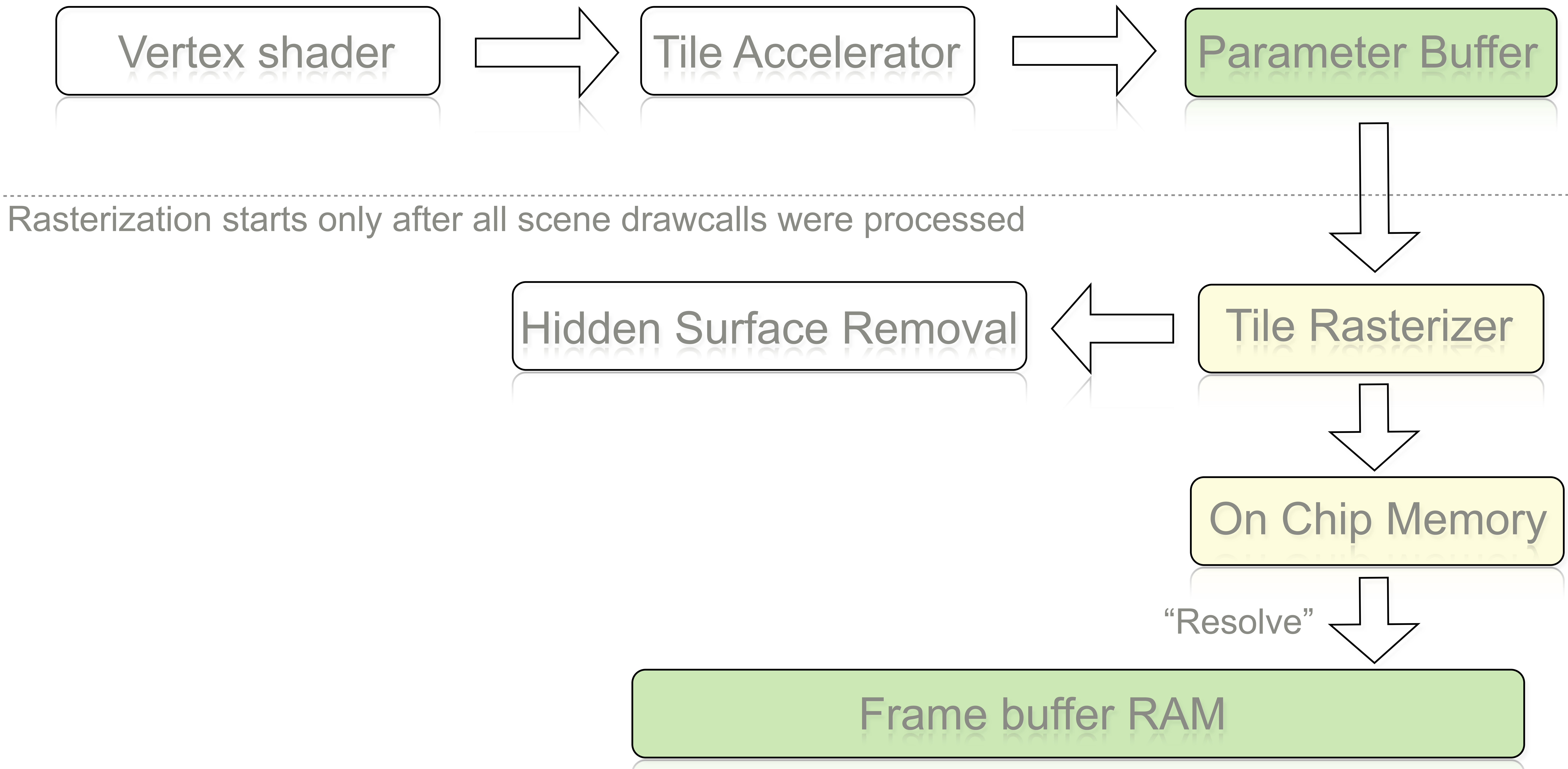
- Splits screen into tiles
  - **small** (for example: 16x16) - SGX, MALI
  - relatively **large** (for example 256K) - Adreno
- Tile memory is on chip - **fast!**
- Once GPU is done rendering tile
  - tile is "**resolved**" - written out to slower RAM



Tiled Deferred Architecture

- Per-drawcall: polygons are transformed, assigned to tiles, **stored** in memory (Parameter Buffer)
- Rasterization starts only **after** all scene drawcalls were processed
  - every tile has access to **all** covering polygons
- Per-tile: Occluded polygons are rejected and **only visible** parts of polygons are rasterized
  - for opaque geometry rasterization will touch every pixel **only once**
  - saves ALU and texture reads

Tiled Deferred Architecture



Not so scary in practice! Just...

- Sort opaque geom differently for Traditional vs Tiled
  - **Tiled: sort by material** to reduce CPU drawcall overhead
  - **Traditional: sort roughly front-to-back** to maximize ZCull efficiency
    - then by material
  - **Tiled Deferred:** render alpha-tested **after** opaque
    - higher chance that expensive alpha-tested pixels will be occluded
- Separate render loop for MBX Fixed Function
  - optimized for low-end devices, can go faster than GLES2.0 loop, no per-pixel lighting, limited postFX possibilities
  - phasing it out
- Be more aggressive with **16bit** framebuffers on Tiled

Not so scary in practice! Just...

SIGGRAPH2012

- Use **EXT_discard_framebuffer** extensions on Tiled
  - will avoid copying data (color/depth/stencil) you're not planning to use
- **Clear** RenderTarget before rendering into it
  - otherwise on Tiled driver will copy color/depth/stencil back from RAM
  - not clearing is **not** an optimization!

Architectural Benefits

- Benefits
  - Tiled: **MSAA** is almost **free** (5-10% of rendering time)
  - Tiled: **AlphaBlend**ing is significantly **cheaper**
  - Tiled: **less** dithering artifacts for **16bit** framebuffers
- Caveats
  - TBDR: RenderTarget switch might be more expensive
  - TBDR: Too much geometry will flush whole pipeline (ParameterBuffer overflow)

Interesting Tiles

SIGGRAPH2012

- Reminds recent works
  - “Tile-based Deferred Shading”, Andrew Lauritzen, SIGGRAPH2010
  - “Tile-based Forward Rendering”, Takahiro Hirada, GDC2012
- ... suitable for high-end GPUs
  - different problems
  - but common solutions

Screen Resolutions (Android)

SIGGRAPH2012

- Most often found resolutions are darker

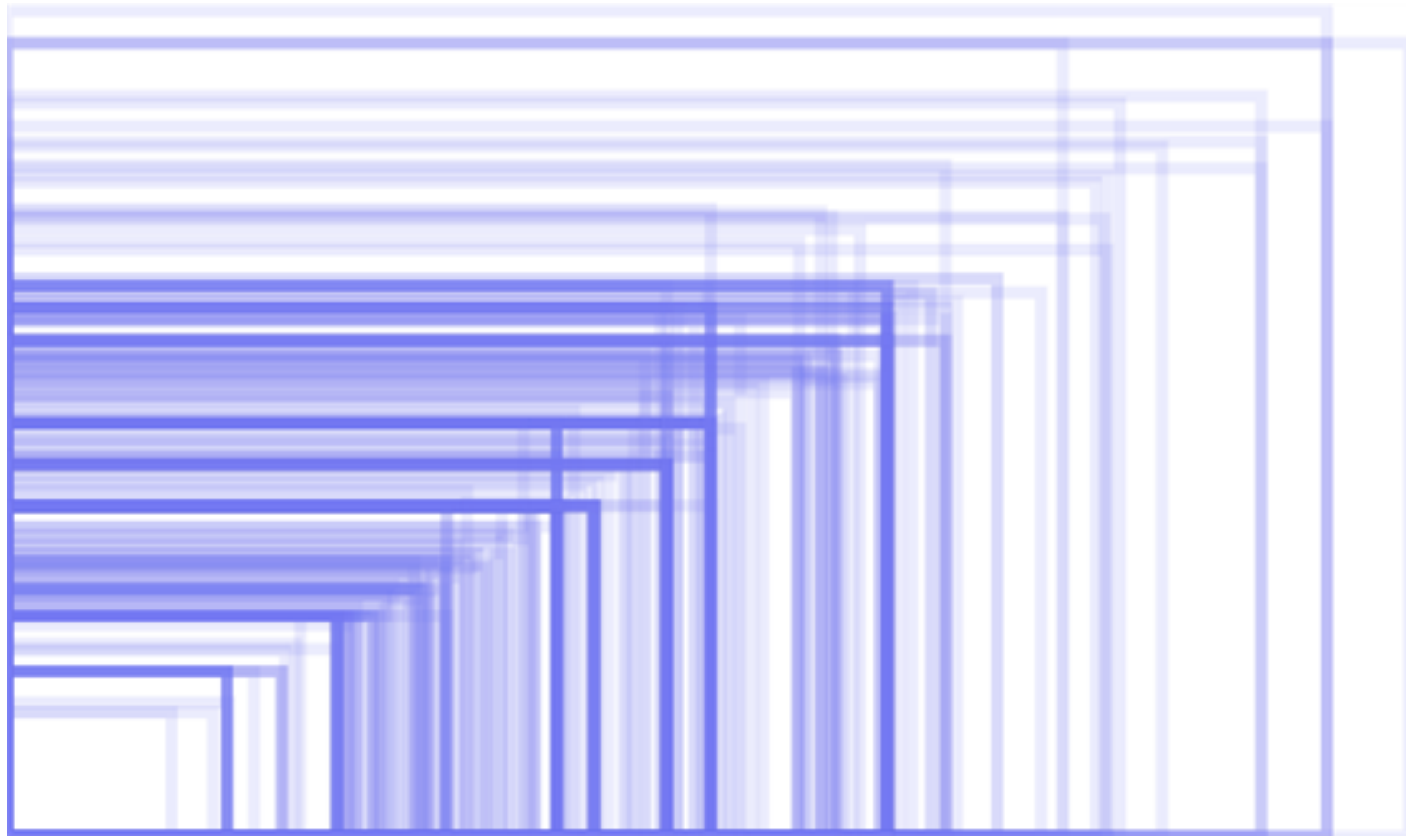


Image is a courtesy of OpenSignalMaps

What is more scary: Drivers!

SIGGRAPH2012

- **Android** specific problem!
- Graphics Drivers
  - bugs
  - performance variations
  - chaos of 90ies is back!
- Quality **is** dramatically improving on IHV side
  - **but** in many cases mobile vendors won't provide new drivers for their devices: don't care / security testing / phased out devices...

What is more scary: Drivers!

SIGGRAPH2012

- Establish good relations with IHV
- Send bug-reports
- Automatize testing
  - more on auto testing later
- Help Google with their **open-source** testing rig!
  - http://source.android.com/compatibility/downloads.html

Multiple Texture formats

- **Android** specific problem!
- ETC1 mandatory in **GL ES2.0** - but NO Alpha support!
  - Instead platform specific formats: PVRTC, ATC, DXT5, ETC2
  - **No** single format which would be supported on **all** devices
- Uncompressed 16bit for textures with Alpha
  - slow, large
- Yay! **GL ES3.0** solves Alpha - mandatory ETC2
  - Plus new formats: EAC, ASTC

Textures with Alpha in GL ES2.0

SIGGRAPH2012

- Pair of ETC textures: **RGB** in 1st texture + **Alpha** in 2nd
- **Self-downloading** application
  - small bootrstrap app - determines GPU family on 1st run
  - downloads and stores pack with GPU specific assets
    - Unity: AssetBundles
    - GooglePlay: new expansion files (up to 2GB)
- GooglePlay **filtering**
  - build multiple versions of the game, each with textures for certain GPU
  - <supports-gl-texture> tag in AndroidManifest

GPU Architecture: Shader cores

SIGGRAPH2012

- Unified - Vertex & Pixel use the same core
  - Workload balancing
  - **SGX, Adreno, Mali T6xx**
- Traditional - Vertex and Pixel cores are separate
  - Either stage can be bottleneck at any given moment
  - **Tegra, Mali 4xx, MBX**

Skinning + Unified Architecture

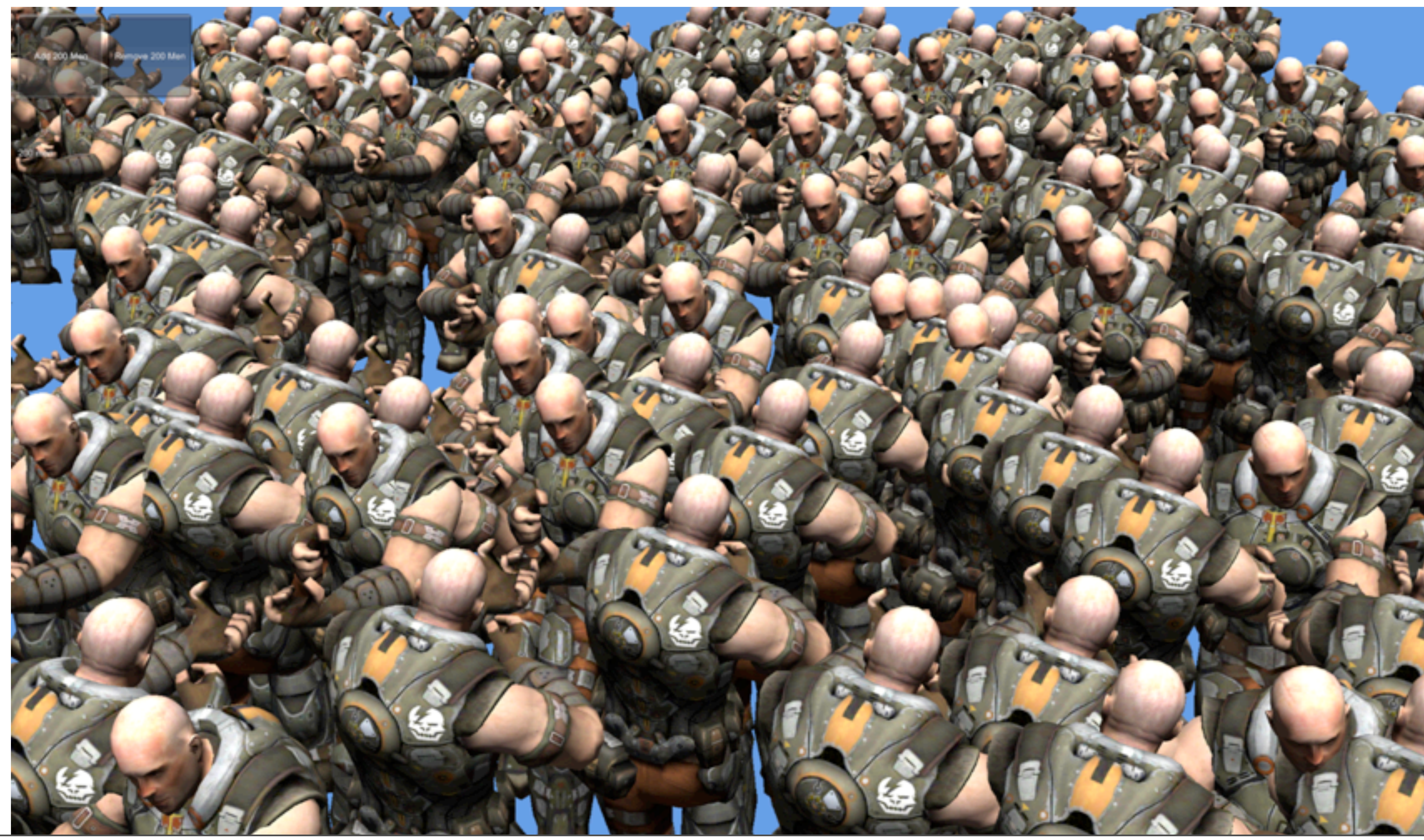
SIGGRAPH2012

- Offload work **from** GPU - skinning on **CPU** with NEON
  - Favors Unified architecture - reduces vertex workload on GPU
  - Tegra non Unified, but has 4 very fast NEON cores - so **good too**
- **Reuse** skinning results: shadows, multi-pass lighting
- Reduces code complexity & shader permutations

Skinning on CPU

SIGGRAPH2012

- Results on **A9 @ 1Ghz** (iPad3), NEON, **1 core**:
  - 1 bones, position+normal+tangent - 12.2 Mverts/sec
  - 2 bones, position+normal+tangent - **11 Mverts/sec**
  - 4 bones, position+normal+tangent - 6.7 Mverts/sec
  - Test: 200 characters each 558 vertices



Skinning on CPU

SIGGRAPH2012

- **Warning:** net result of offloading work to CPU is trickier when power consumption comes to play!
  - game might run faster
  - but can drain battery faster too (NEON is power hungry)

Balancing CPU vs GPU

- Ideally would use DirectX11 **Compute** alike shaders
  - **if** driver could run **same** shader on GPU or CPU depending on platform / workload
  - all reusable geometry transformations and image PostProcessing
- Might be worth trying Transform Feedback in **GL ES3.0**

GPU Architecture: Precision of pixel ops

SIGGRAPH2012

- Optimal precision for GPU family
  - **11/12bit** per-component (*fixed*) - SGX **pre543**, **Tegra**
  - **16bit** per-component (*half*) - **SGX 543**, **Mali 4xx**
  - **32bit** per-component (*float*) - **Adreno, Mali T6xx**
- Watch out for precision conversions
  - most often will require additional cycles!
  - (at least) SGX543 can hide conversion overhead sampling from texture

Precision mixing examples

SIGGRAPH2012

- BAD

```
struct Input {
    float4 color : TEXCOORD0;
};
fixed4 frag (Input IN) : COLOR
{
    return IN.color; // BAD: float -> fixed conversion
}
```

- BAD

```
fixed4 uniform;
...
half4 result = pow (l, 2.2) + uniform; // BAD: fixed -> half conversion
```

- OK

```
half4 tex = tex2d (textureRGBA8bit, uv); // OK: conversion for free
```

Cross platform precision considerations

SIGGRAPH2012

- **sRGB** reads/writes are **not** available on mobiles yet
  - though some hardware supports already
- As a result **linear lighting** is too **expensive**
- Arguable *fixed* point (11bit) can be enough for many pixels
  - do per-pixel lighting in object space
  - do fog per-vertex
  - no depth-shadowmaps
  - for specular could use texture lookup instead of *pow ()*
    - at least 3 cycles (actually 4 to comply with ES standards)
    - *pow ()* result is in *half/float* precision, requires conversion to *fixed*

Cross-platform shaders in Unity

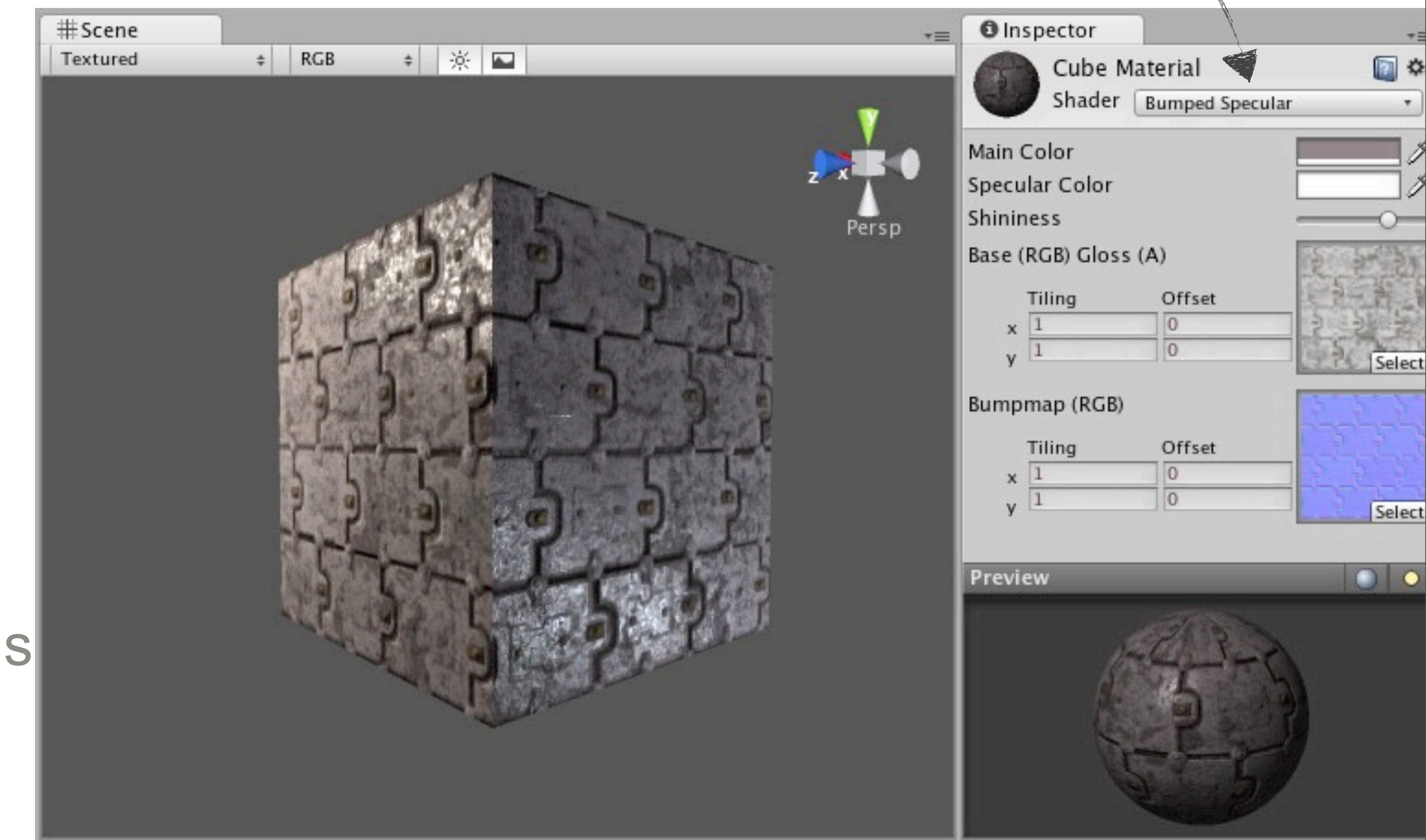
SIGGRAPH2012

- Cg/HLSL **snippets** wrapped in custom language
  - helps to defines state, multipass rendering and lighting setup
- Rationale: maximizing cross-platform applicability
  - abstract from mundane shader details
  - **generate platform specific code** in:
    - HLSL
    - GLSL / **GLSL ES**
    - DirectX or ARB assembly
    - AGL
    - etc

Cross-platform shaders in Unity

SIGGRAPH2012

- Artist specifies high-level shader on the Material
  - ex: "Bumped Specular", "Tree Leaves", "Unlit"
- Run-time picks specific platform shader depending on
  - supported feature set
    - via **Shader Fallback**
  - state (lights / shadows / lightmaps)
    - via builtin **Shader Keywords**
  - user-defined keys
    - via **Shader LOD** + custom Shader Keywords



Shader Fallback

SIGGRAPH2012

- “If this shader can not run on this hardware, then try next one”
- Fallbacks can be chained

```
Shader "Per-pixel Lit" {
    // shader code here ...
    Fallback "Per-vertex Lit"
}
```

Shader Keywords

SIGGRAPH2012

- Built-in and custom shader permutations
- Using shader pre-processor macros

```
#pragma multi_compile LIGHTING_PER_PIXEL
...
#ifdef LIGHTING_PER_PIXEL
// per pixel-lit
#else
// per vertex-lit
#endif

#pragma multi_compile PREFER_HALF_PRECISION
#ifdef PREFER_HALF_PRECISION
  // force all operations to higher precision
  #define scalar half
  #define vec4 half4
#else
  #define scalar fixed
  #define vec4 fixed4
#endif
```

Shader Keywords

SIGGRAPH2012

- Example triggers custom shader permutation from script

```
// Devices with lots of muscle per pixel
if (iPhone.generation == iPad2Gen ||
    iPhone.generation == iPhone4S ||
    iPhone.generation == iPhone3GS)
    Shader.EnableKeyword ("LIGHTING_PER_PIXEL");

// Devices with SGX543
if (iPhone.generation == iPad2Gen ||
    iPhone.generation == iPad3Gen ||
    iPhone.generation == iPhone4S)
    Shader.EnableKeyword ("PREFER_HALF_PRECISION");
```

Shader LevelOfDetail

SIGGRAPH2012

- Shader switch depending on platform performance
  - LOD - integer value

```
Shader "Lit" {
    SubShader { LOD 200 // per pixel-lit ..
    SubShader { LOD 100 // per vertex-lit ..
}
```

- Example triggers shader LOD from script

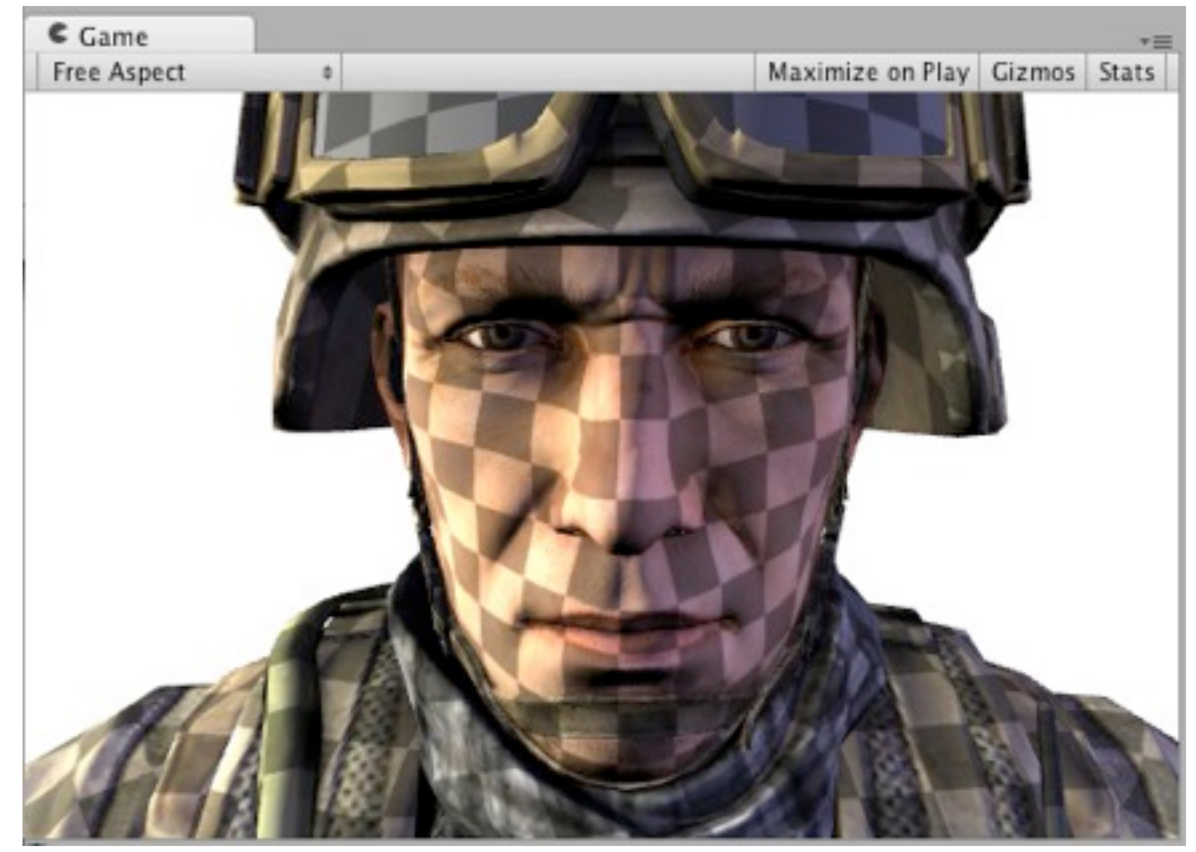
```
// Devices with lots of muscle per pixel
if (iPhone.generation == iPad2Gen ||
    iPhone.generation == iPhone4S ||
    iPhone.generation == iPhone3GS)
    Shader.globalMaximumLOD = 200;
```

Cross-platform shaders

- Surface **shading** and **lighting** snippets
  - Instead of writing full vertex/pixel shader
  - Just snippets of code
- Generate all “cruft” automagically depending on platform and state
  - Shader generation is done offline

```
#pragma surface MySurface Ramp
void MySurface (Input IN, inout SurfaceOutput o) {
    o.Albedo = tex2D (_MainTex, ...);
    o.Albedo *= tex2D (_Detail, ...) * 2;
    o.Normal = UnpackNormal (tex2D (_BumpMap, ...));
}

half4 LightingRamp (SurfaceOutput s, half3 lightDir ...) {
    half2 NdotL = dot (s.Normal, lightDir);
    half3 ramp = tex2D (_Ramp, NdotL);
    half4 l;
    l.rgb = s.Albedo * ramp;
    ...
    return l;
}
```

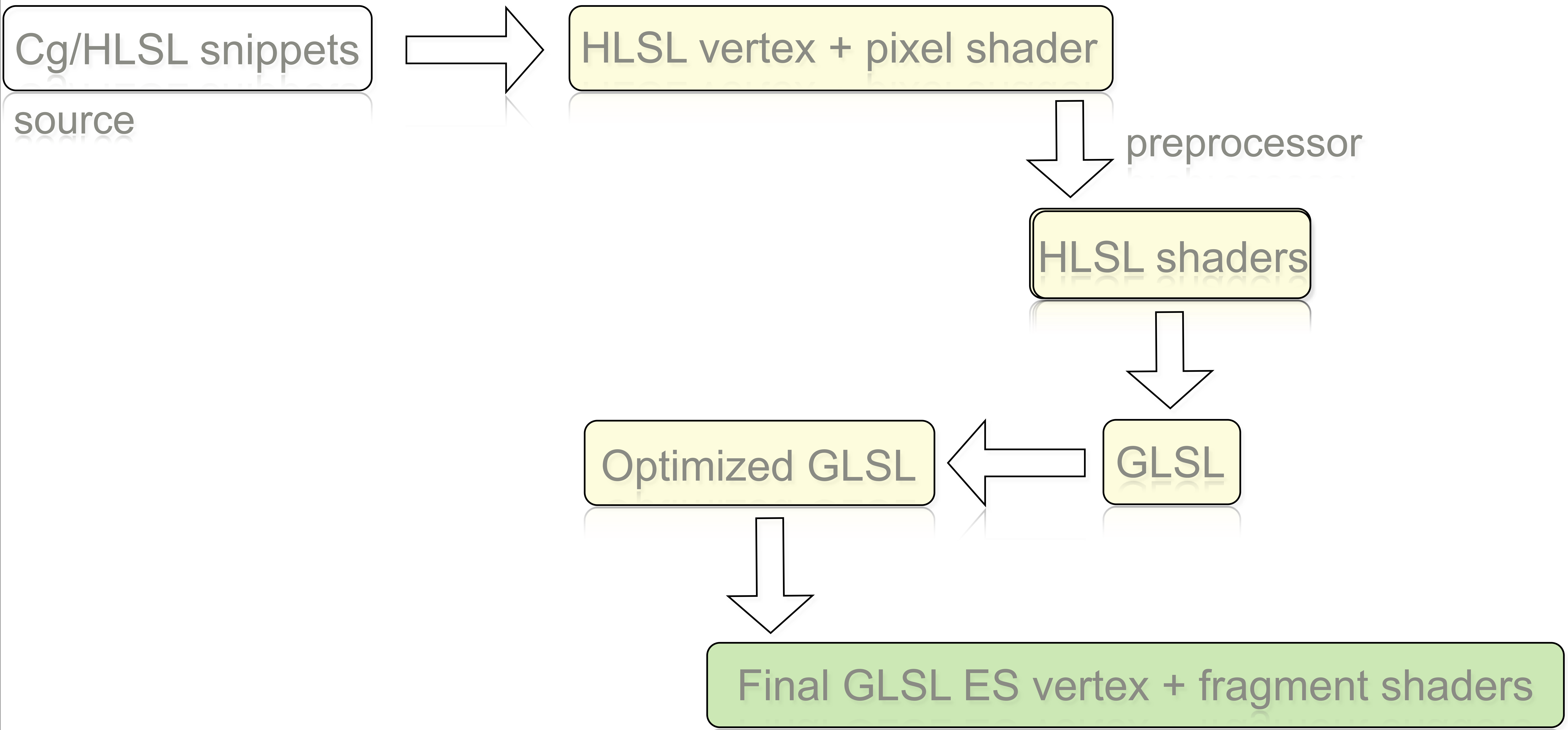


Automatic code optimization

SIGGRAPH2012

- *cgbatch*: Takes Cg/HLSL snippets and generates complete shader code in HLSL
- Preprocessor step
- *hlsl2glsl*: Converts HLSL to GLSL
  - resurrected old ATI's project, fixed & improved. **Open source!** https://github.com/aras-p/hlsl2glslfork
- *glsls-optimizer (1)*: Offline GPU-independent GLSL optimization
  - think inlining, dead code removal, copy propagation, arithmetic simplifications etc.
  - 2 year ago many mobile drivers were bad at optimizations - we had 2-10x improvement
  - Still very valuable
- *glsls-optimizer (2)*: A fork of Mesa3D GLSL compiler that prints out GLSL ES after all optimizations.
  - **Open source!** https://github.com/aras-p/glsl-optimizer

Shader generation steps in Unity



Shader patching

SIGGRAPH2012

- No dedicated hardware for blending, write masking, flexible vertex input in (many) Mobile GPUs
  - instead driver will **patch** shader code
  - significant hiccup on the **first** drawcall /w **new** shader/state combination
- Prewarming
  - force driver to do patching during load time
  - issue drawcalls with dummy geometry for all shader/state combinations
  - in Unity API: *Shader.WarmupAllShaders* ()

Back to scary: ES2.0 API overhead

- Drawcall overhead on CPU
  - 0.05ms per **average** drawcall on CPU (iPad2/iPad3)
  - 600 drawcalls will **max** out CPU @ 30FPS
- Sorted by relative cost:
  - *glDrawXXX*: draw call itself
  - *glUniformXXX*: shader uniform uploads
  - *glVertexAttribPointer*: vertex input setup
  - state change

OpenGL ES2.0 API overhead

SIGGRAPH2012

- It is not just about drawcall counts
  - important to **minimize** number of uniforms and state changes
  - sort by Material
  - optimize uniforms in shaders
- **GL ES2.0** prevents many optimizations
  - uniforms can not be treated as a sequential memory - drawcall setup requires multiple calls
  - uniforms are set per shader - calls on every shader change
  - no means for binding uniform to a specific register - unlike HLSL
- Yay! **GL ES3.0** - Uniform Buffer Object!

Drawcall batching

- First reduce state changes and uniform uploads
- Reduce overhead by grouping multiple objects with the same state into **one** drawcall
- Relies on sorting by material first
  - applicable to **opaque** geometry mostly
  - not applicable to multi-pass lighting either
  - lighting data passed to shaders must be in world or view space

Unity "static" batching

SIGGRAPH2012

- Suitable for **static** environment
- Static VertexBuffer + **Dynamic** IndexBuffer
- @ Build-time
  - objects are combined into a large shared Vertex Buffers
  - sharing same material
- @ Run-time
  - indices of visible objects are appended to **dynamic** Index Buffer

Unity "static" batching

- But **Dynamic** Buffers are **tricky** on some mobile platforms (see next)
- Instead could:
  - @ Build-time organize objects into Octree, traverse in **Morton** order and write to shared Vertex Buffer
  - @ Run-time traverse in the same (**Morton**) order, render visible objects with neighboring Vertex Buffer ranges in a single drawcall
  - like "*Segment Buffering*", Jon Olick, GPU Gems2
  - all buffers are static

Unity "dynamic" batching

SIGGRAPH2012

- Suitable for **dynamic** objects
  - with relatively simple geometry (see below)
- Transform object vertices to world space on **CPU** (NEON)
  - append to shared dynamic Vertex Buffer and render in one drawcall
- Makes sense only for objects with low vertex count
  - otherwise transformation cost would outweigh the cost of the drawcall itself
  - usually **200-800** vertices per object

Dynamic geometry

- **Never** do like this in **GL ES2.0!**

```
for (;my_render_loop;)
    glBindBuffer (..., myBuffer);
    glMapBufferOES (..., GL_WRITE_ONLY_OES);
    // write data
    glUnmapBufferOES (...);
    glDrawElements (...);
```

  - will **block** CPU waiting for GPU to finish rendering from your buffer

Dynamic geometry

- Geometry of known size (skinning) is easy
  - **double/triple** buffer - render from one buffer, while writing to another
  - swap buffers **only** at the end of the frame
- Geometry of **arbitrary** size (particles, batching) is **harder**
  - no fence / discard support in out-of-the-box GL ES2.0
  - Yay for **GL ES3.0**, again!

Dynamic geometry of arbitrary size

SIGGRAPH2012

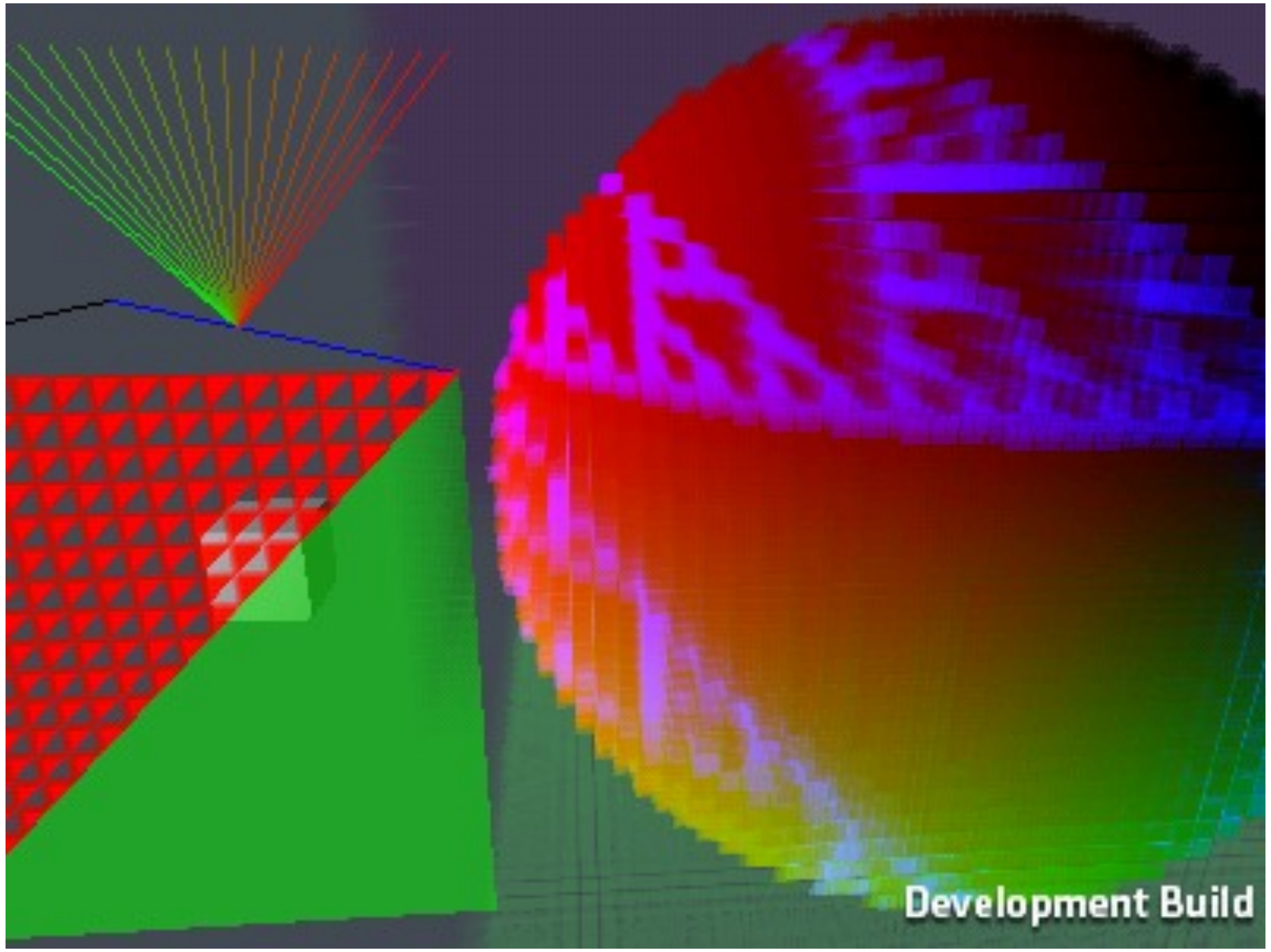
- Buffer renaming/orphaning is supported by **some** drivers (iOS)

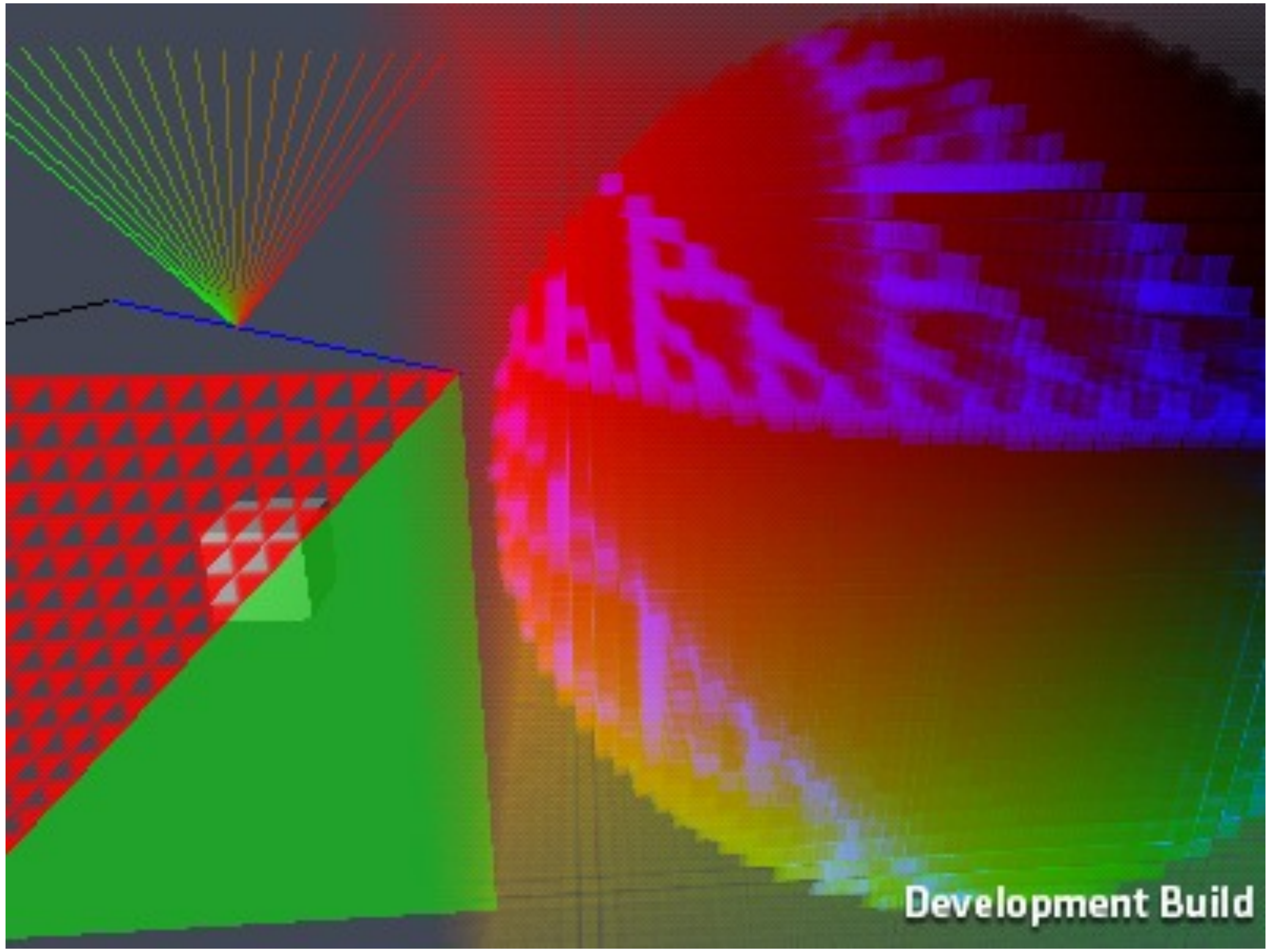
```
// orphan old buffer, driver should allocate new storage
glBufferData (..., bytes, 0, GL_STREAM_DRAW);
glMapBufferOES (..., GL_WRITE_ONLY_OES);
```

- Preallocate multiple buffers
  - write to buffer **once**, mark it “busy” for 1 (or 2) frames and start rendering
  - grab next “non-busy” buffer, otherwise allocate more buffers and continue
  - could use *NV_fence* / *EGL_sync* extension to track if GPU is finished with certain buffer

- Do simulation and write all data to buffers **before** entering render-loop
  - and don’t forget to **double/triple** your buffers

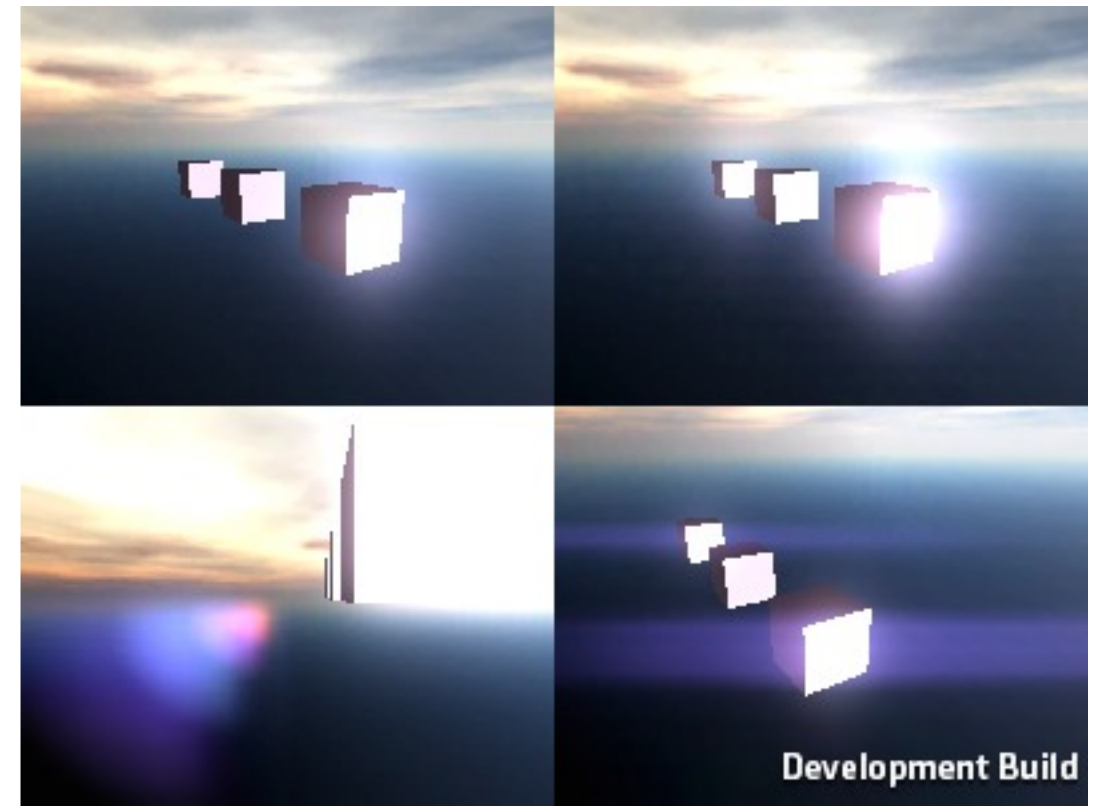
Automated Testing

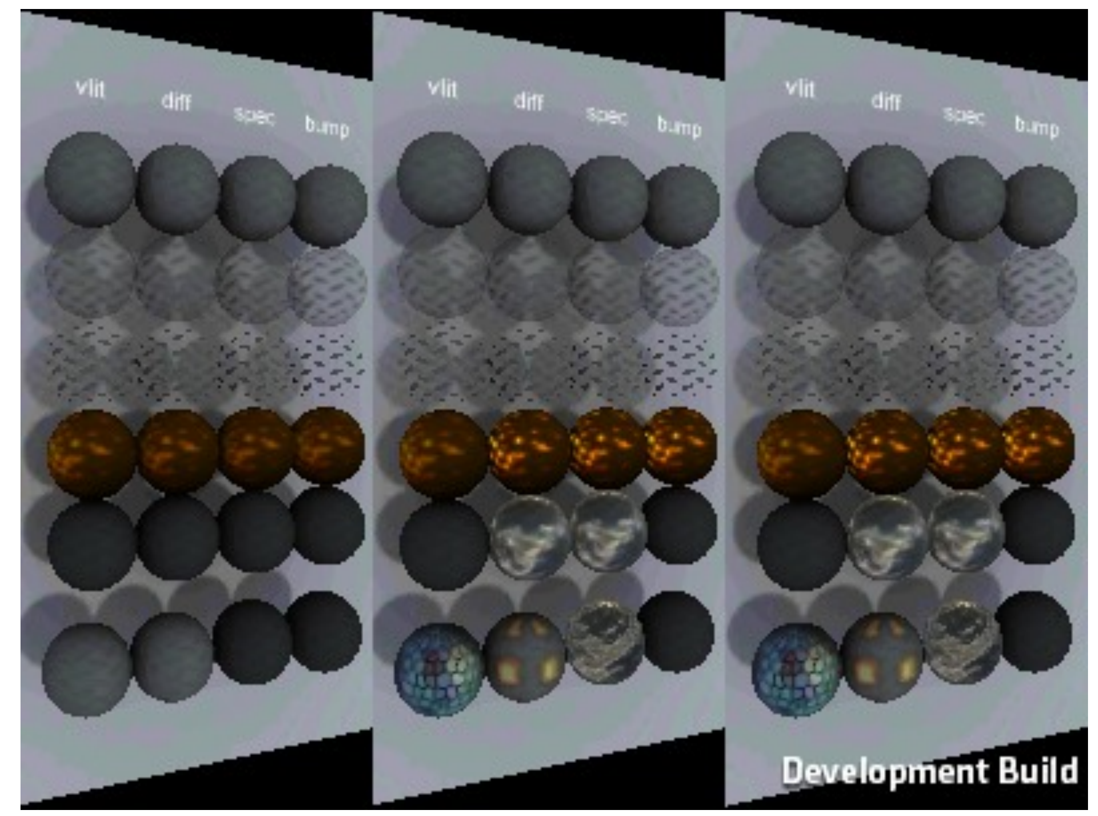


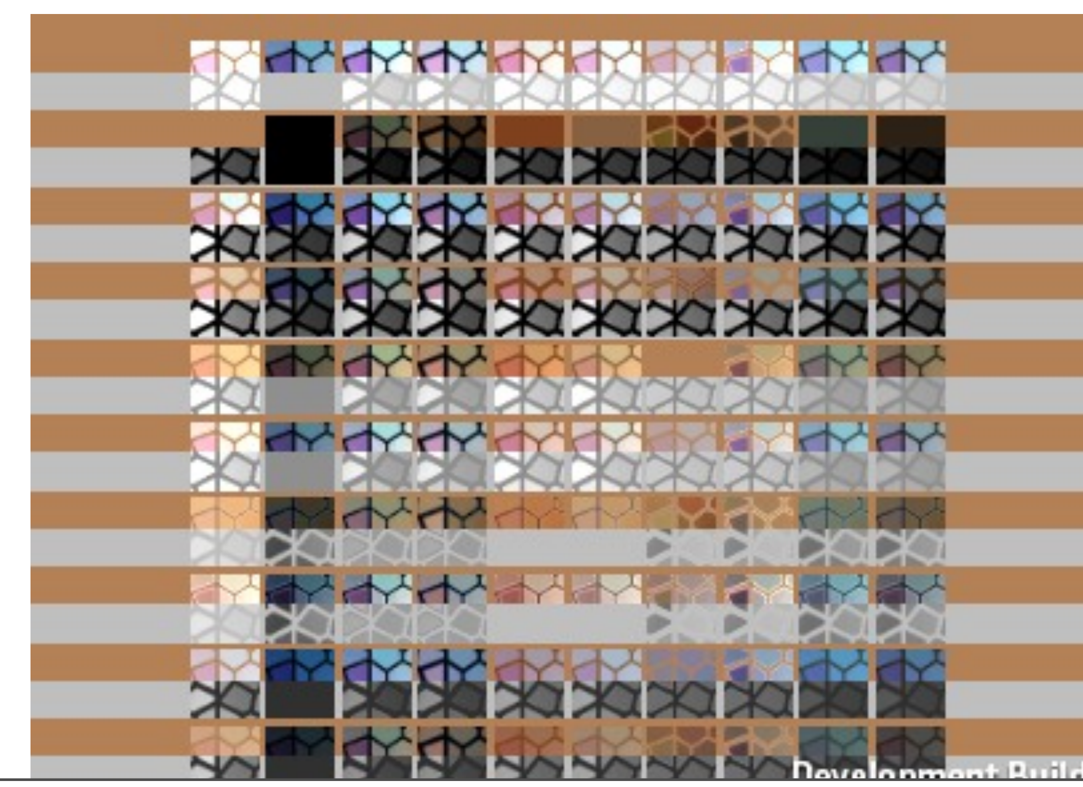


Automated Testing

- Run same content on different devices
  - different OS updates
  - automatic on internal code changes
- Capture screenshots and compare to templates
  - per-pixel comparison
- Simplified scenes to test specific areas
  - our test suite - 238 scenes



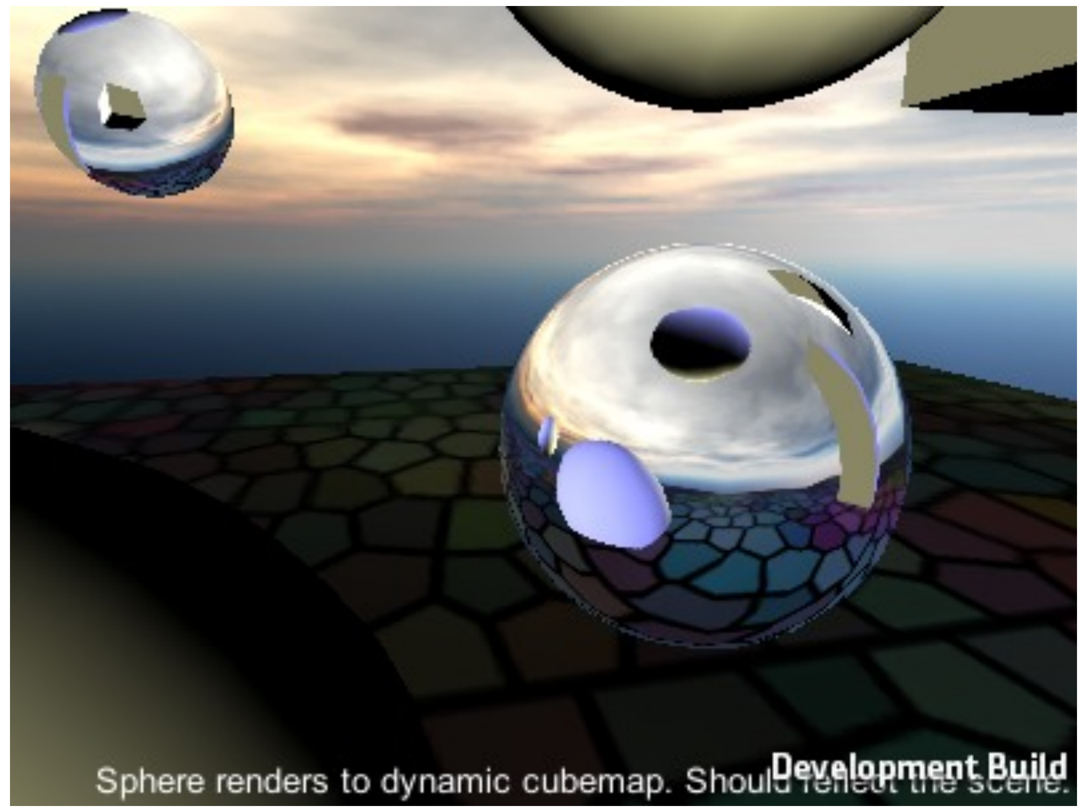


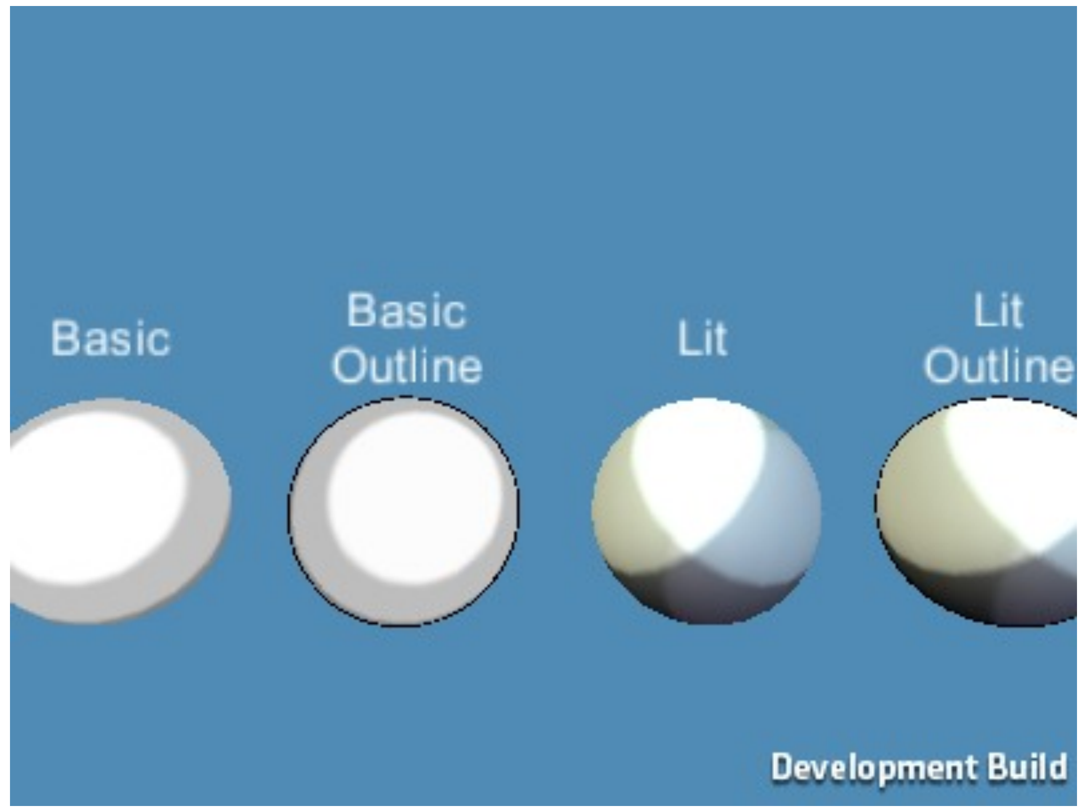


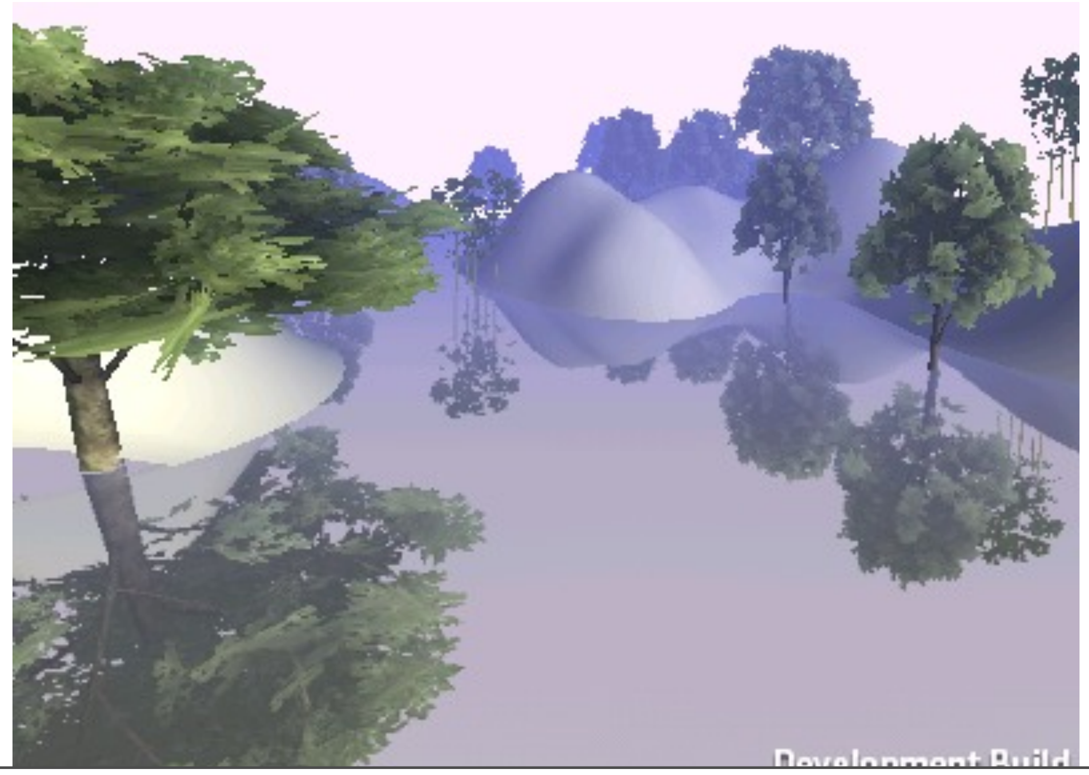
Automated Testing

SIGGRAPH2012

- Devices we use
  - Nexus One (Adreno 205)
  - Samsung Galaxy S 2 (Mali 400)
  - Nexus S / Galaxy Nexus (SGX 540)
  - Motorola Xoom (Tegra2)
- Why not more?







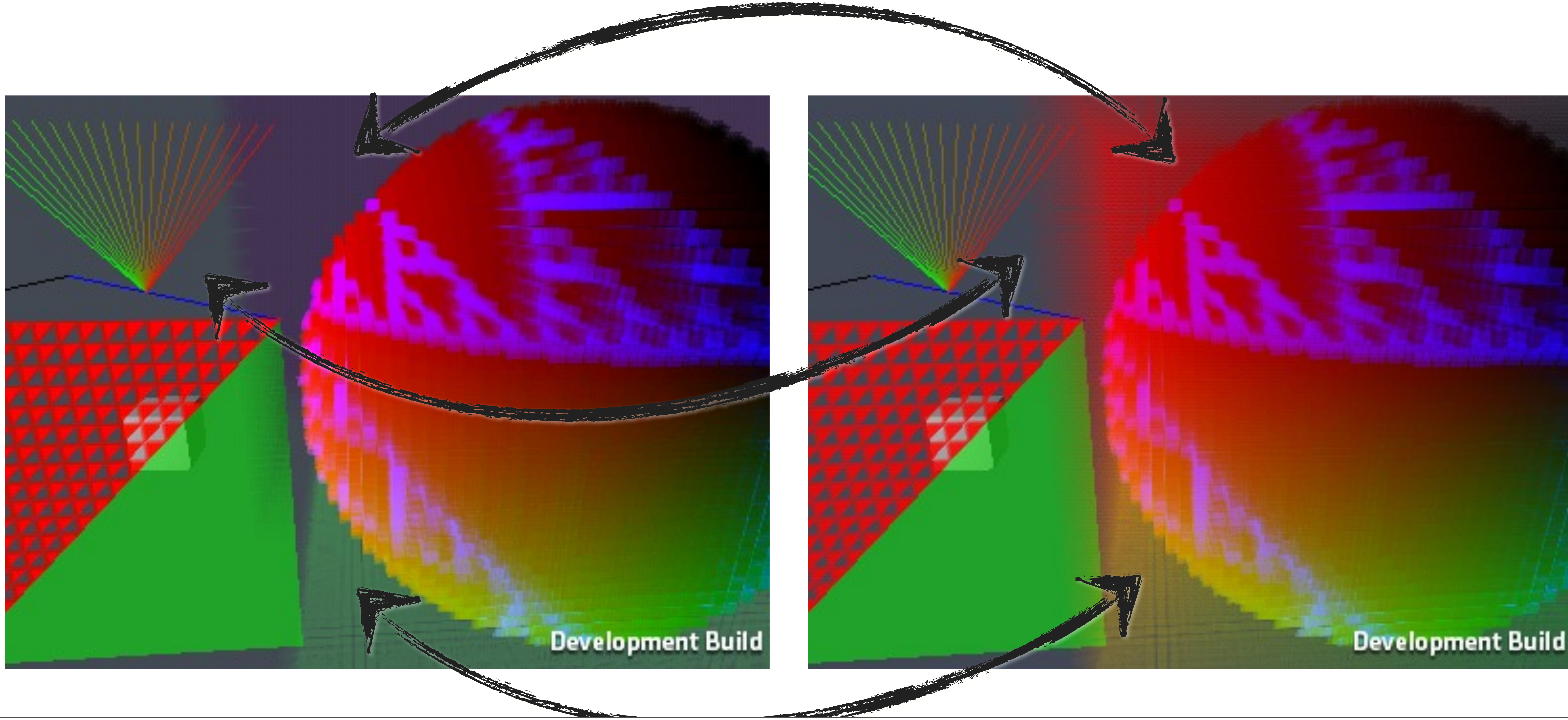
Automated Testing Challenges

SIGGRAPH2012

- Some devices simply crash on some tests
  - drivers, argh!
- Some shader variations will just produce wrong results
- Loosing connection with the host
  - hooking up **more than one** device per PC makes connection more likely to **fail**
- Test results might differ significantly from device to device
- But there are people who manage to workaround this

Fun with Automated Testing

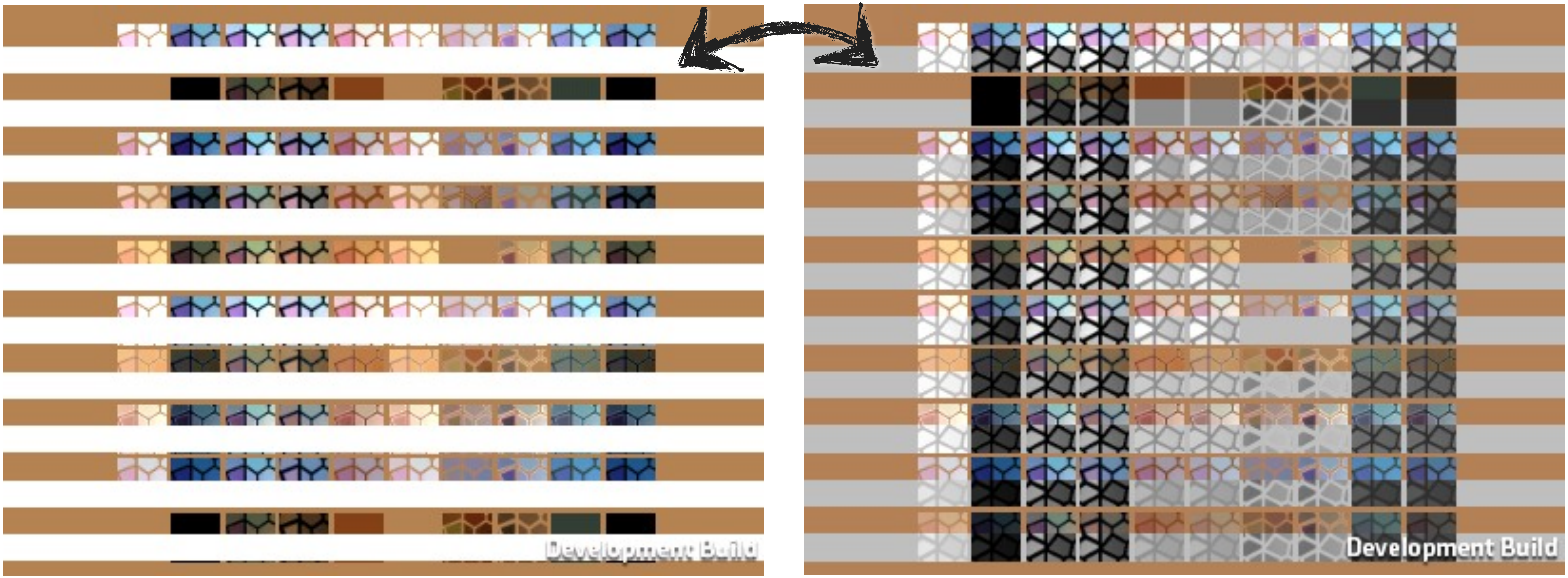
- **AlphaBlend**ing differences on 2 distinct GPUs



Fun with Automated Testing

SIGGRAPH2012

- **BlendMode** differences on 2 distinct GPUs



Fun with Automated Testing

- ReadPixels

Q&A

SIGGRAPH2012

@__ReJ__