NVIDIA OpenGL in 2016

Mark Kilgard, July 24

SIGGRAPH 2016, Anaheim

Mark Kilgard

My Background

- Principal System Software Engineer
    - OpenGL driver and API evolution
    - Cg (“C for graphics”) shading language
    - GPU-accelerated path rendering & web browser rendering
- OpenGL Utility Toolkit (GLUT) implementer
- Specified and implemented much of OpenGL
- Author of *OpenGL for the X Window System*
- Co-author of *Cg Tutorial*
- Worked on OpenGL for 25 years

NVIDIA’s OpenGL Leverage

Programmable Graphics

GeForce

Debugging with Nsight

Tegra

Quadro

OptiX

Adobe Creative Cloud

OpenGL®
THE WAY
NVIDIA
IT'S MEANT TO BE PLAYED™
TITANFALL

NSIGHT VSE AND OPENGL VR

Jeff Kiel - Manager, Graphics Tools

AGENDA

- Intro to Nsight & Developer Tools
- VR debugging
- GPU Range Profiling
- Roadmap

Compile Debug Profile

Microsoft® DirectX®

OpenGL

NVIDIA CUDA

OpenGL ES

android NDK

GNU C/C++

Trace

OpenCV

NVIDIA VISIONWORKS™

cuDNN

NVTX

NVIDIA Tools eXtension

IDE Integration

Visual Studio®

eclipse

Standalone and CLI

Getting Started...

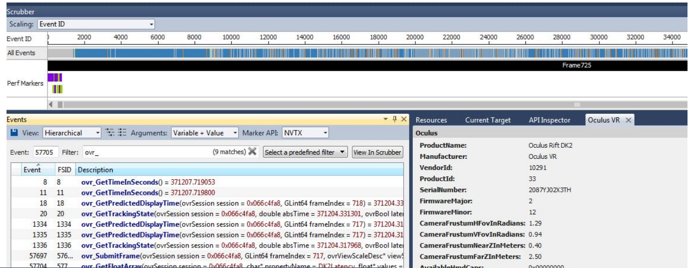
JetPack

Hardware Support

NSIGHT VISUAL STUDIO EDITION 5.2

- VR, Vulkan, and Advanced Graphics Profiling

- New Range Profiler, including OpenGL and DirectX12
- Vulkan Support
- New Geometry View
- Oculus VR SDK support, OpenGL and DX11
- CUDA 8.0 support



Events

View: Hierarchical Arguments: Variable + Value Marker API: NVTX

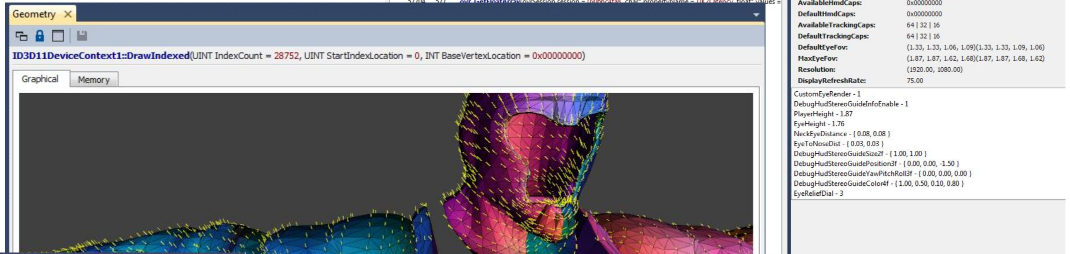
Event: 57705 Filter: ovr_ (9 matches) Select a predefined filter View In Scrubber

```
Event  FSID  Description
    8     8  ovr_GetTimeInSeconds() = 371207.719053
   11    11  ovr_GetTimeInSeconds() = 371207.719800
   18    18  ovr_GetPredictedDisplayTime(ovrSession session = 0x066c4fa8, GLint64 frameIndex = 718) = 371204.33
   20    20  ovr_GetTrackingState(ovrSession session = 0x066c4fa8, double absTime = 371204.331301, ovrBool late
 1334  1334  ovr_GetPredictedDisplayTime(ovrSession session = 0x066c4fa8, GLint64 frameIndex = 717) = 371204.31
 1335  1335  ovr_GetPredictedDisplayTime(ovrSession session = 0x066c4fa8, GLint64 frameIndex = 717) = 371204.31
 1336  1336  ovr_GetTrackingState(ovrSession session = 0x066c4fa8, double absTime = 371204.317968, ovrBool late
57697  576…  ovr_SubmitFrame(ovrSession session = 0x066c4fa8, GLint64 frameIndex = 717, ovrViewScaleDesc* view
57704  577…  ovr_GetFloatArray(ovrSession session = 0x066c4fa8, char* propertyName = DK2Latency, float* values =
```

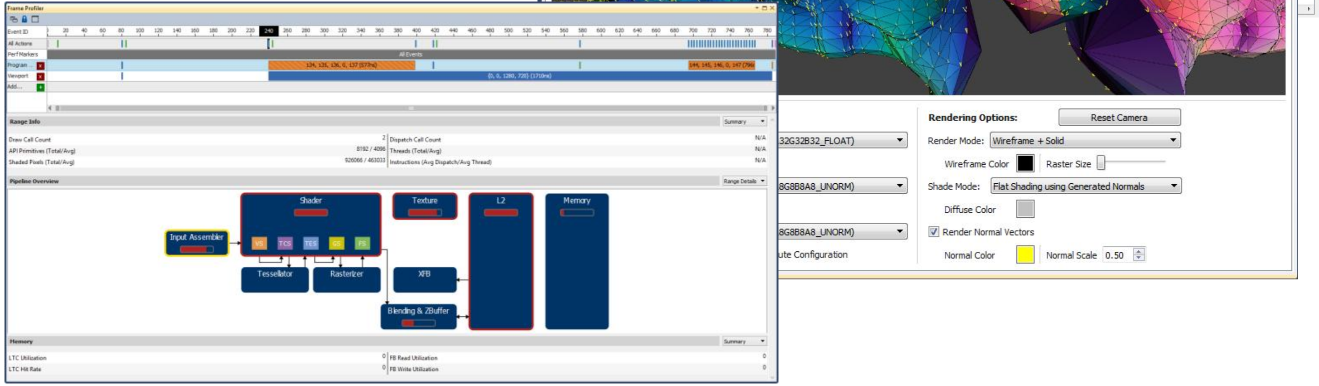
Oculus

ProductName:	Oculus Rift DK2
Manufacturer:	Oculus VR
VendorId:	10291
ProductId:	33
SerialNumber:	2087YJ02X3TH
FirmwareMajor:	2
FirmwareMinor:	12
CameraFrustumHFovInRadians:	1.29
CameraFrustumVFovInRadians:	0.94
CameraFrustumNearZInMeters:	0.40
CameraFrustumFarZInMeters:	2.50
AvailableHmdCaps:	0x00000000
DefaultHmdCaps:	0x00000000
AvailableTrackingCaps:	64 | 32 | 16
DefaultTrackingCaps:	64 | 32 | 16
DefaultEyeFov:	(1.33, 1.33, 1.06, 1.09)(1.33, 1.33, 1.09, 1.06)
MaxEyeFov:	(1.87, 1.87, 1.62, 1.68)(1.87, 1.87, 1.68, 1.62)
Resolution:	(1920.00, 1080.00)
DisplayRefreshRate:	75.00

```
CustomEyeRender - 1
DebugHudStereoGuideInfoEnable - 1
PlayerHeight - 1.87
EyeHeight - 1.76
NeckEyeDistance - { 0.08, 0.08 }
EyeToNoseDist - { 0.03, 0.03 }
DebugHudStereoGuideSize2f - { 1.00, 1.00 }
DebugHudStereoGuidePosition3f - { 0.00, 0.00, -1.50 }
DebugHudStereoGuideYawPitchRoll3f - { 0.00, 0.00, 0.00 }
DebugHudStereoGuideColor4f - { 1.00, 0.50, 0.10, 0.80 }
EyeReliefDial - 3
```



ID3D11DeviceContext1::DrawIndexed(UINT IndexCount = 28752, UINT StartIndexLocation = 0, INT BaseVertexLocation = 0x00000000)



Range Info			
Draw Call Count	2	Dispatch Call Count	N/A
API Primitives (Total/Avg)	8192 / 4096	Threads (Total/Avg)	N/A
Shaded Pixels (Total/Avg)	926066 / 463033	Instructions (Avg Dispatch/Avg Thread)	N/A

Memory			
LTC Utilization	0	FB Read Utilization	0
LTC Hit Rate	0	FB Write Utilization	0

32G32B32_FLOAT)

8G8B8A8_UNORM)

8G8B8A8_UNORM)

ute Configuration

Rendering Options: Reset Camera

Render Mode: Wireframe + Solid

Wireframe Color ■ Raster Size

Shade Mode: Flat Shading using Generated Normals

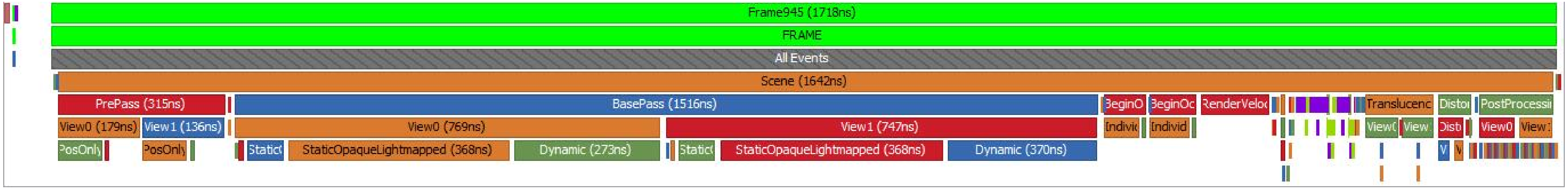
Diffuse Color □

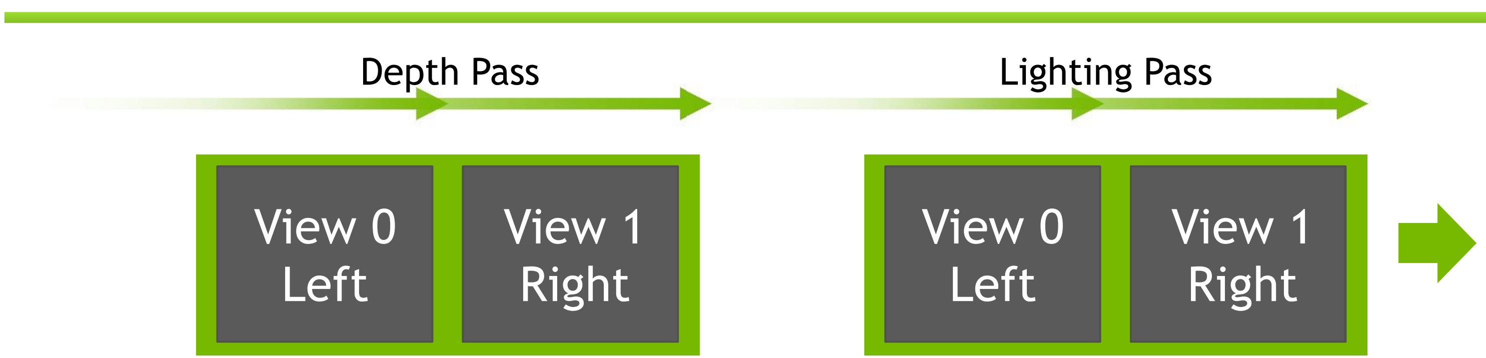
☑ Render Normal Vectors

Normal Color 🟨 Normal Scale 0.50

UE4’S VR ENGINE

- Render pass per eye





Depth Pass

View 0 Left

View 1 Right

Lighting Pass

View 0 Left

View 1 Right

DEMO TIME!

ROADMAP

When you get back from SIGGRAPH: 5.2 RC1

- VR Goodness
  - OCULUS SDK, OpenGL and Direct3D
  - OpenGL Multicast Rendering
- Range Profiler (OpenGL & D3D)
- Vulkan
  - Frame Debugging
  - BETA: Serialized Captures
- DX12 Serialized Captures

September, 2016: 5.2 Final

ROADMAP

Q4 2016: 5.3

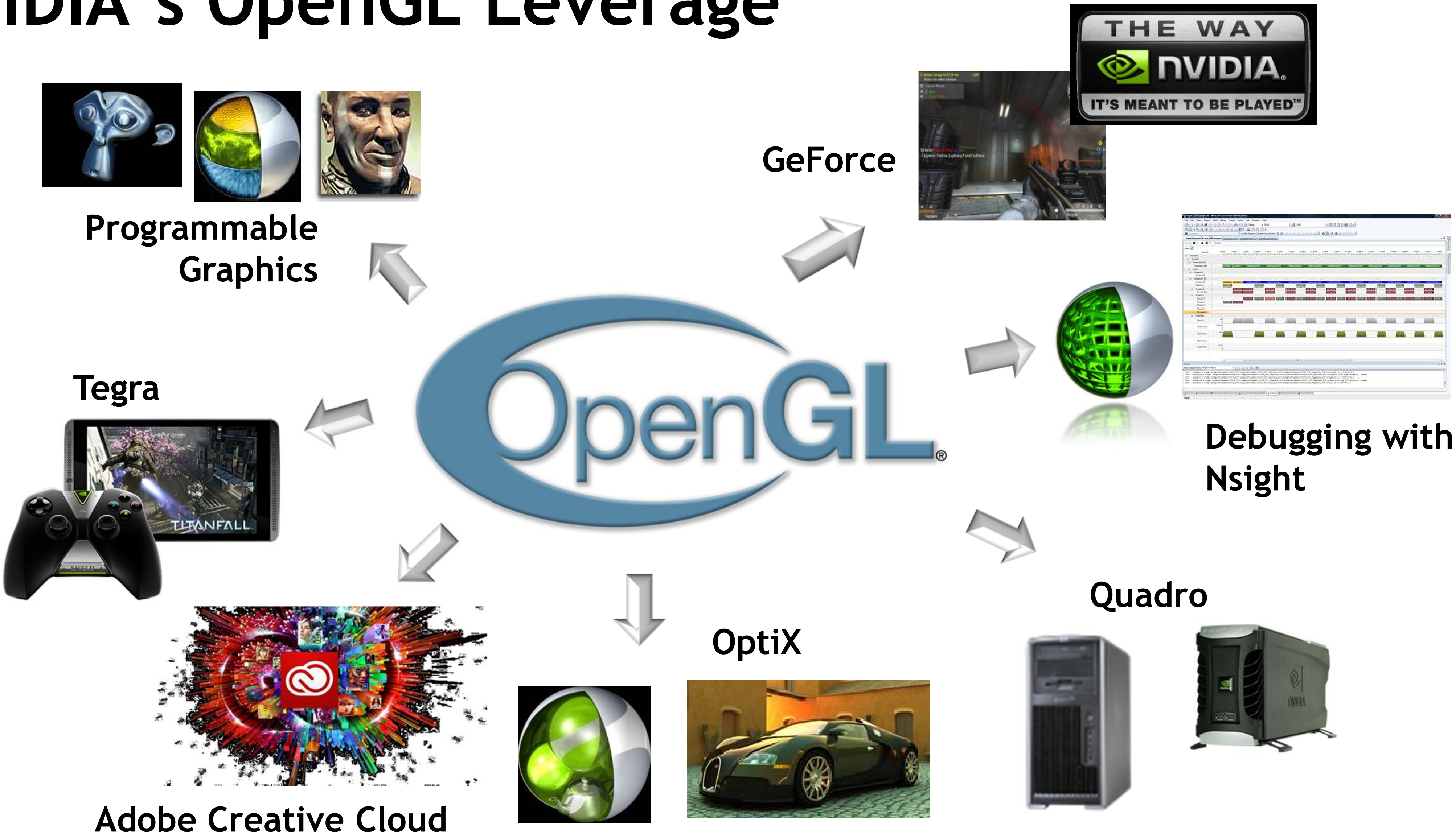
- More VR Goodness
- More Profiler Screens & Metrics
- Shader Perf Returns!
- MS Hybrid Supporp & UWP

The Future

- Vulkan Profiling
- Shader Source Correlated Performance Information
- Shader Debugging on Maxwel & Pascal
- Pipeline Statistics
- Compare API State/Profile Runs
- Path Rendering
- Your Feature Here…

Tell Me What You Need!?!?

NVIDIA's OpenGL Leverage



Programmable Graphics

GeForce

Debugging with Nsight

Quadro

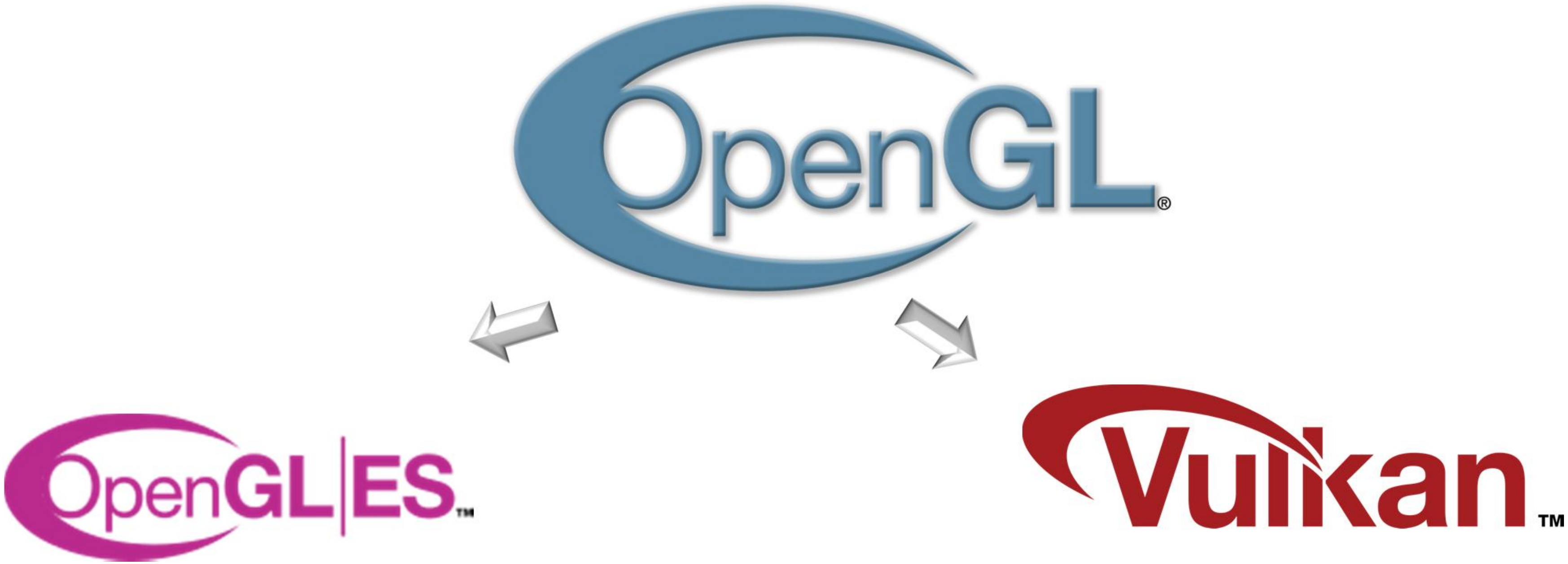
OptiX

Adobe Creative Cloud

Tegra

OpenGL Codebase Leverage

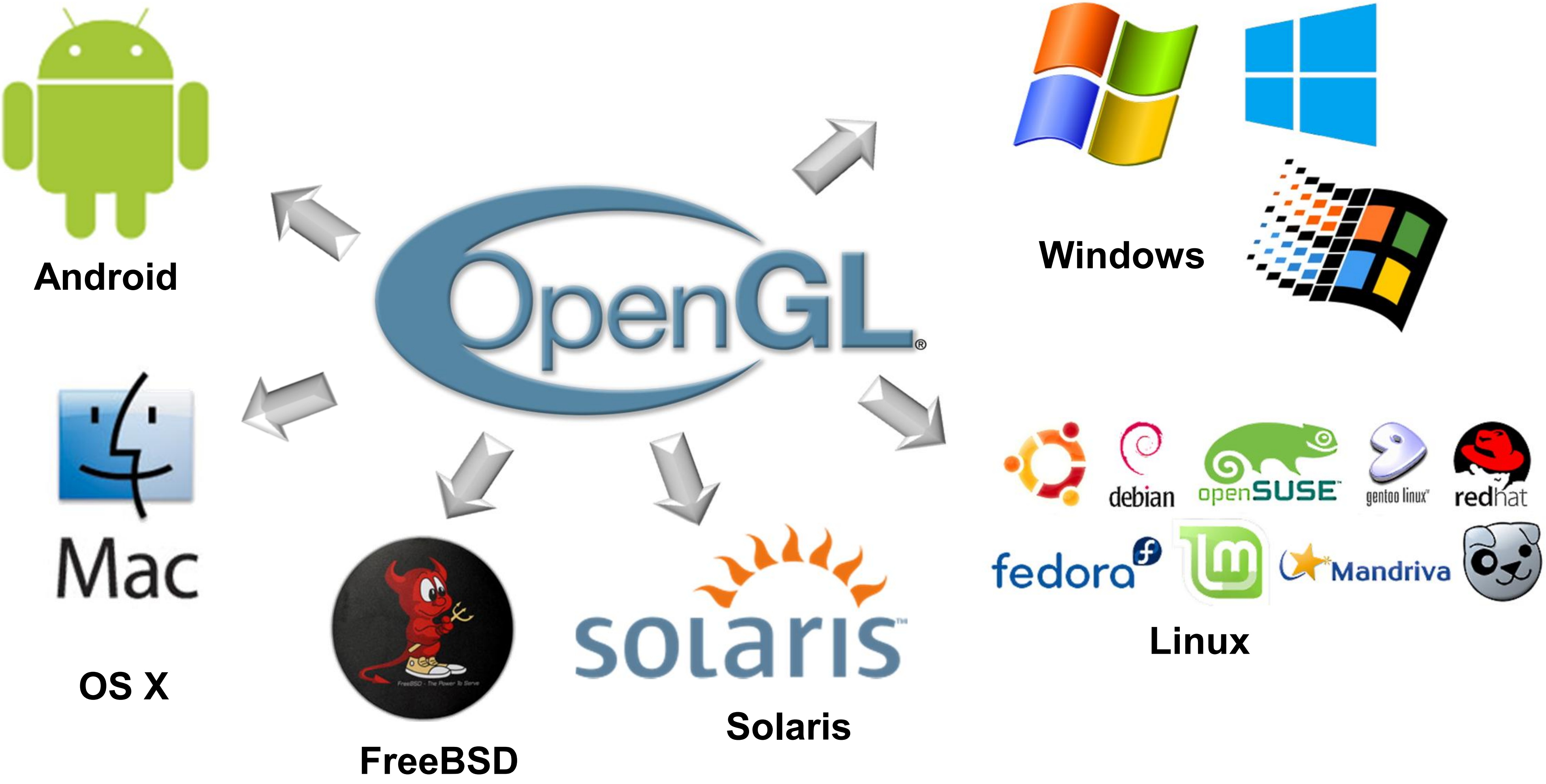
Same driver code base supports multiple APIs



OpenGL for Embedded, Mobile, and Web

Multi-vendor, explicit, low-level graphics from Khronos

Still the One Truly Common & Open 3D API



Android

Windows

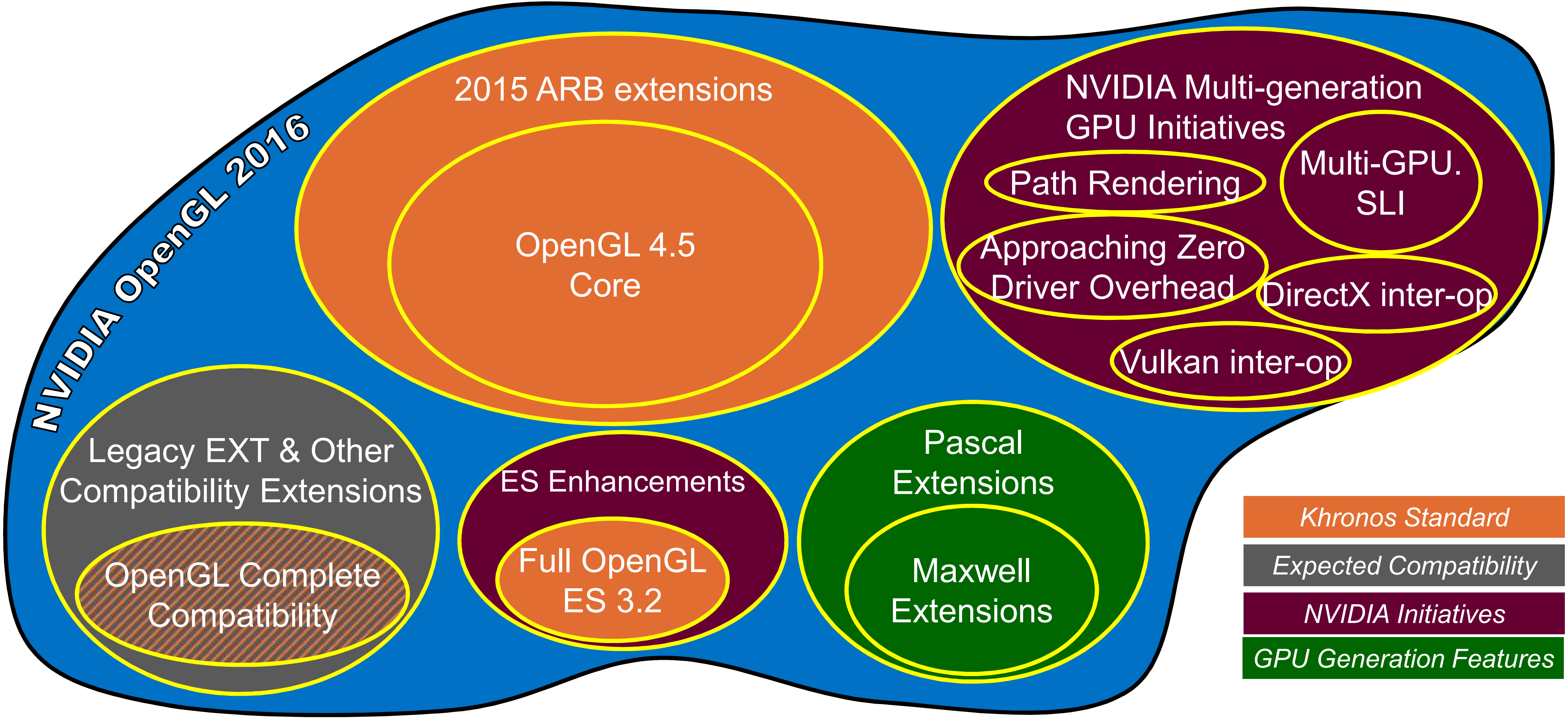
OS X

FreeBSD

Solaris

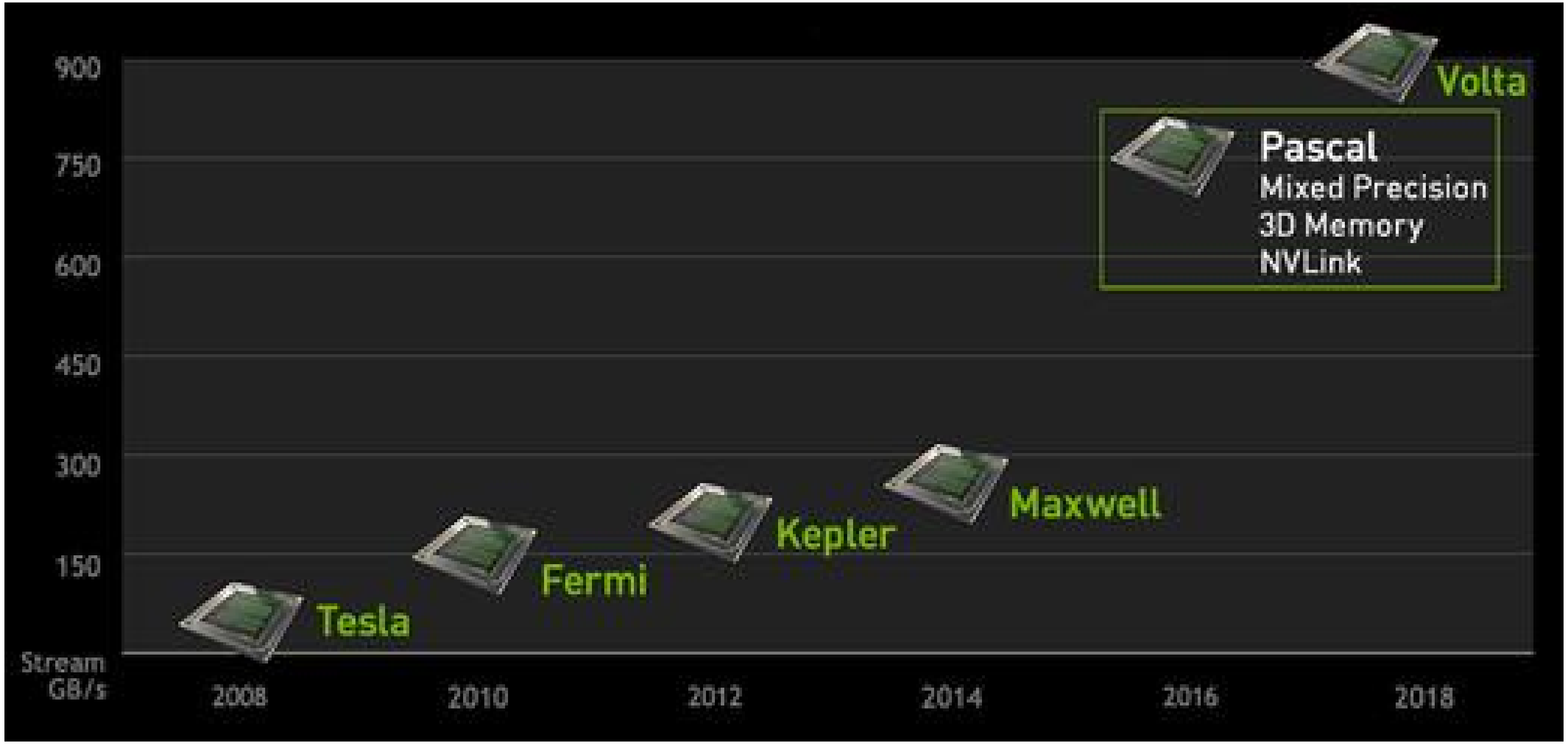
Linux

NVIDIA OpenGL in 2016 Provides OpenGL's Maximally Available Superset



Background: NVIDA GPU Architecture Road Map

What are Maxwell and Pascal mentioned on last slide?



Our interest NVIDIA GPU architectures of interest: Maxwell & Pascal

OpenGL’s Recent Advancements

New ARB Extensions

3 standard extensions, beyond 4.5

- ARB_sparse_buffer
- ARB_pipeline_statistics_query
- ARB_transform_feedback_overflow_query

Maxwell Extensions

- Novel graphics features
- 14 new extensions
- Global Illumination & Vector Graphics focus

2014

2015

2016

OpenGL’s Recent Advancements

OpenGL 4.5

New ARB Extensions

3 standard extensions, beyond 4.5

- ARB_sparse_buffer
- ARB_pipeline_statistics_query
- ARB_transform_feedback_overflow_query

Maxwell Extensions

- Novel graphics features
- 14 new extensions
- Global Illumination & Vector Graphics focus

New ARB 2015 Extension Pack

- **Shader functionality**
  - ARB_ES3_2_compatibility (shading language support)
  - ARB_parallel_shader_compile
  - ARB_gpu_shader_int64
  - ARB_shader_atomic_counter_ops
  - ARB_shader_clock
  - ARB_shader_ballot
- **Graphics pipeline operation**
  - ARB_fragment_shader_interlock
  - ARB_sample_locations
  - ARB_post_depth_coverage
  - ARB_ES3_2_compatibility (tessellation bounding box + multisample line width query)
  - ARB_shader_viewport_layer_array
- **Texture mapping functionality**
  - ARB_texture_filter_minmax
  - ARB_sparse_texture2
  - ARB_sparse_texture_clamp

2014

2015

2016

OpenGL's Recent Advancements

OpenGL 4.5

New ARB Extensions

3 standard extensions, beyond 4.5

- ARB_sparse_buffer
- ARB_pipeline_statistics_query
- ARB_transform_feedback_overflow_query

Maxwell Extensions

- Novel graphics features
- 14 new extensions
- Global Illumination & Vector Graphics focus

2014

New ARB 2015 Extension Pack

- **Shader functionality**
  - ARB_ES3_2_compatibility (shading language support)
  - ARB_parallel_shader_compile
  - ARB_gpu_shader_int64
  - ARB_shader_atomic_counter_ops
  - ARB_shader_clock
  - ARB_shader_ballot
- **Graphics pipeline operation**
  - ARB_fragment_shader_interlock
  - ARB_sample_locations
  - ARB_post_depth_coverage
  - ARB_ES3_2_compatibility (tessellation bounding box + multisample line width query)
  - ARB_shader_viewport_layer_array
- **Texture mapping functionality**
  - ARB_texture_filter_minmax
  - ARB_sparse_texture2
  - ARB_sparse_texture_clamp

2015

SPIR™

OpenGL SPIR-V Support

- Standard Shader Intermediate Representation
- ARB_gl_spirv
- Vulkan interoperability

Pascal Extensions

- Novel graphics features
- 5 new extensions
- Virtual Reality focus

2016

Maxwell OpenGL Extensions

New Graphics Features of NVIDIA’s Maxwell GPU Architecture

- **Voxelization, Global Illumination, and Virtual Reality**
  - NV_viewport_array2
  - NV_viewport_swizzle
  - AMD_vertex_shader_viewport_index
  - AMD_vertex_shader_layer
- **Vector Graphics extensions**
  - NV_framebuffer_mixed_samples
  - EXT_raster_multisample
  - NV_path_rendering_shared_edge
- **Advanced Rasterization**
  - NV_conservative_raster
  - NV_conservative_raster_dilate
  - NV_sample_mask_override_coverage
  - NV_sample_locations,
    now ARB_sample_locations
  - NV_fill_rectangle
- **Shader Improvements**
  - NV_geometry_shader_passthrough
  - NV_shader_atomic_fp16_vector
  - NV_fragment_shader_interlock,
    now ARB_fragment_shader_interlock
  - EXT_post_depth_coverage,
    now ARB_post_depth_coverage

Requires GeForce 950, Quadro M series, Tegra X1, or better

Background: Viewport Arrays

Indexed Array of Viewport & Scissor State

Several Maxwell (and Pascal) extensions build on Viewport Arrays

Viewport arrays introduced to OpenGL standard by OpenGL 4.1

- Feature of Direct3D 11
- First introduced to OpenGL by NV_viewport_array extension

Each viewport array element contains

- Viewport transform
- Scissor box and enable
- Depth range

Provides N mappings of clip-space to scissored window-space

Original conception

- Geometry shader could “steer” primitives into any of 16 viewport array elements
- Geometry shader would set the viewport index of a primitive
- Result: primitive is rasterized based on the indexed viewport array state

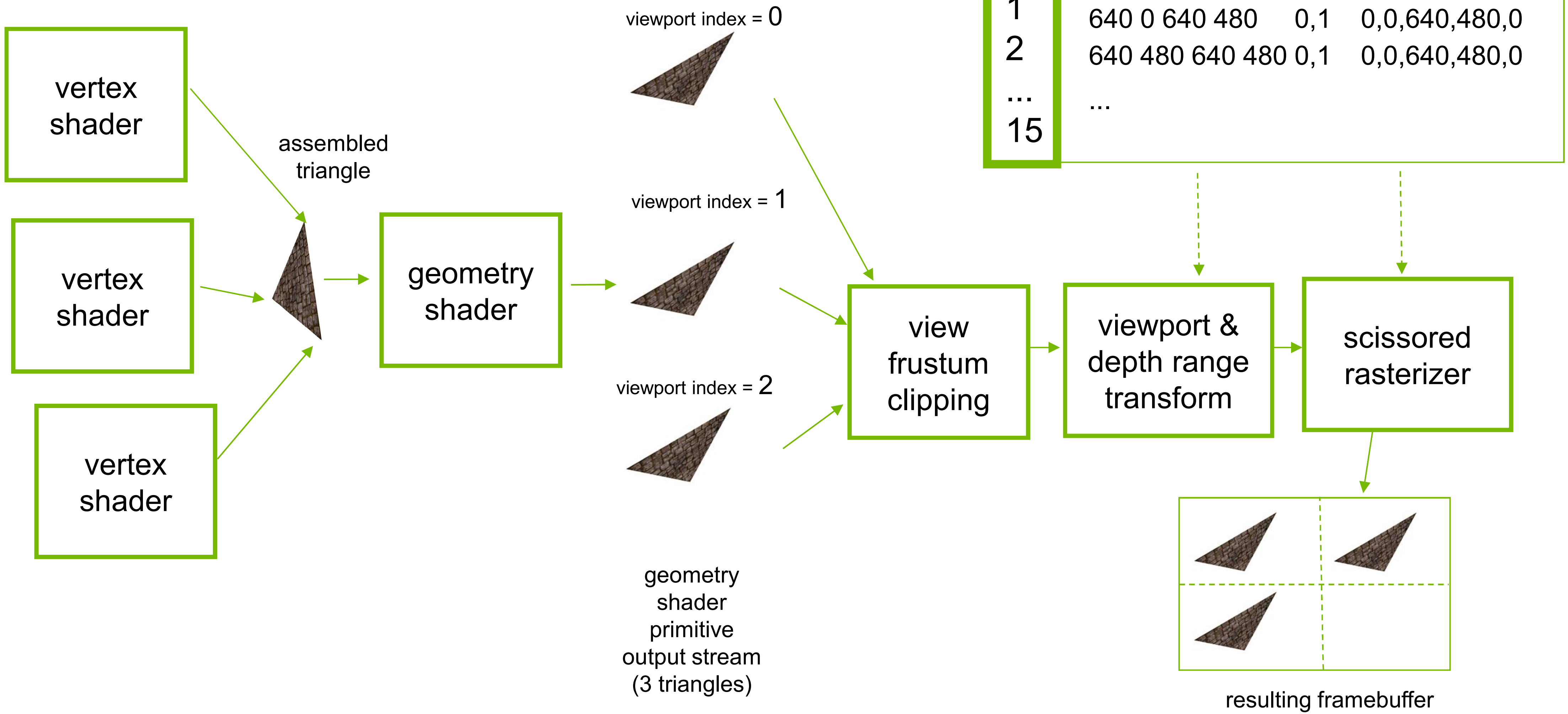
Viewport array state

	x_v y_v w_v h_v	n,f	x_s y_s w_s h_s e_s
0	0 0 640 480	0,1	0,0,640,480,0
1	640 0 640 480	0,1	0,0,640,480,0
2	640 480 640 480	0,1	0,0,640,480,0
…	…		
15			

Viewport Arrays Visualized

Viewport array state

	x_v y_v w_v h_v	n,f	x_s y_s w_s h_s e_s
0	0 0 640 480	0,1	0,0,640,480,0
1	640 0 640 480	0,1	0,0,640,480,0
2	640 480 640 480	0,1	0,0,640,480,0
...	...		
15			



vertex shader

vertex shader

vertex shader

assembled triangle

geometry shader

viewport index = 0

viewport index = 1

viewport index = 2

geometry shader primitive output stream (3 triangles)

view frustum clipping

viewport & depth range transform

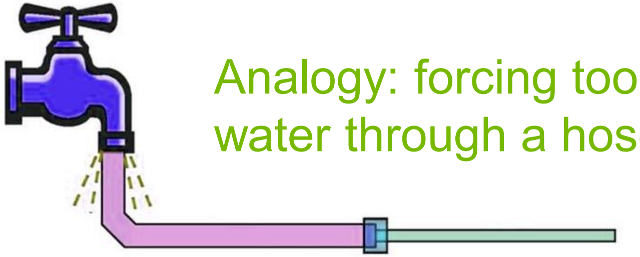
scissored rasterizer

resulting framebuffer

Viewport Index Generalized to Viewport Mask

Maxwell's **NV_viewport_array2** extension

- Geometry shaders & viewport index approach proved limiting...
- Common use of geometry shaders: view replication
  - One stream of OpenGL commands → draws N views
  - But inherently expensive for geometry shader to replicate N primitives
    - Underlying issue: one thread of execution has to output N primitives



Analogy: forcing too much water through a hose

geometry shader

- First fix
  - Replace scalar viewport index per primitive with a viewport bitmask
- Viewport mask does the primitive replication
  - Viewport mask lets geometry shader output primitive to **all**, **some**, or **none** of viewport indices
  - Examples
    - 0xFFFF would replicate primitive 16 times, one primitive for each respective viewport index
    - 0x0301 would output a primitive to viewport indices 9, 8, and 0

Geometry Shader Allowed to "Pass-through" of Vertex Attributes

Maxwell's **NV_geometry_shader_passthrough** Extension

Geometry shaders are very general!

- 1 primitive input → *N* primitives output, where *N* is capped but still dynamic
- input vertex attributes can be arbitrarily recomputed

Not conducive to executing efficiently

- Applications often just want 1 primitive in → constant N primitives out
- with NO change of vertex attributes
- though allowing for computing & output of per-primitive attributes

NV_geometry_shader_passthrough supports a simpler geometry shader approach

- Hence more efficient
- Particularly useful when viewport mask allows primitive replication

Restrictions

- 1 primitive in, 1 primitive out
- BUT writing the per-primitive viewport mask can force replication of 0 to 16 primitives, one for each viewport array index
- No modification of per-vertex attributes

Allowances

- Still get to compute per-primitive outputs
- Examples: viewport mask and texture array layer

Analogy for Geometry Shader "Pass-through" of Vertex Attributes

Efficient, low touch

Slower, high touch

Requires good behavior, many restrictions apply

Fully general, anyone can use this line

Geometry shader just computes per-primitive attributes and passes along primitive "Pass-through" of vertex attributes means geometry shader cannot modify them

Full service geometry shader

Example Pass-through Geometry Shader

Simple Example: Sends Single Triangle To Computed Layer

```
layout(triangles) in;
layout(triangle_strip) out;
layout(max_vertices=3) out;

in Inputs {
  vec2 texcoord;
  vec4 baseColor;
} v_in[];
out Outputs {
  vec2 texcoord;
  vec4 baseColor;
};

void main() {
  int layer = compute_layer();  // function not shown
  for (int i = 0; i < 3; i++) {
    gl_Position = gl_in[i].gl_Position;
    texcoord = v_in[i].texcoord;
    baseColor = v_in[i].baseColor;
    gl_Layer = layer;
    EmitVertex();
  }
}
```

BEFORE: Conventional geometry shader (*slow*)

```
#extension GL_NV_geometry_shader_passthrough : require

layout(triangles) in;
// No output primitive layout qualifiers required.

// Redeclare gl_PerVertex to pass through "gl_Position".
layout(passthrough) in gl_PerVertex {
  vec4 gl_Position;
};
// Declare "Inputs" with "passthrough" to copy members attributes
layout(passthrough) in Inputs {
  vec2 texcoord;
  vec4 baseColor;
};

// No output block declaration required

void main() {
  // The shader simply computes and writes gl_Layer.  We don't
  // loop over three vertices or call EmitVertex().
  gl_Layer = compute_layer();
}
```

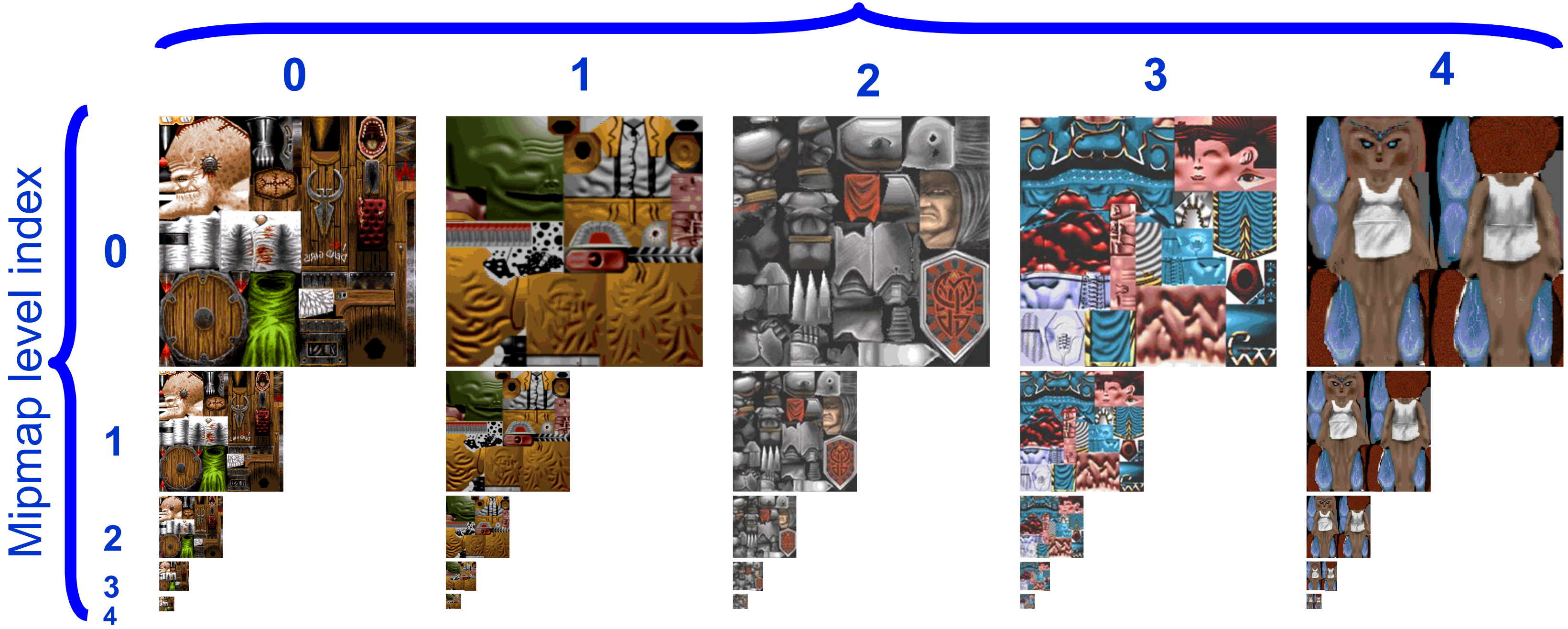
AFTER: Passthrough geometry shader (*fast*)

Outputting Layer Allows Layered Rendering

Allows Rendering to 3D Textures and Texture Arrays

- **Example:** Bind to particular level of 2D texture array with glFramebufferTexture
  Then gl_Layer output of geometry shader renders primitive to designated layer (slice)

Texture array index for texturing, or **gl_Layer** for layered rendering



Example 2D texture array with 5 layers

Aside: Write Layer and Viewport Index from a Vertex Shader

Maxwell’s **AMD_vertex_shader_viewport_index** & **AMD_vertex_shader_layer** Extensions

- Originally only geometry shaders could write the gl_ViewportIndex and gl_Layer outputs

- Disadvantages
  - Limited use of layered rendering and viewport arrays to geometry shader
  - Often awkward to introduce a geometry shader for just to write these outputs
  - GPU efficiency is reduced by needing to configure a geometry shader

- **AMD_vertex_shader_viewport_index** allows gl_ViewportIndex to be written from a vertex shader

- **AMD_vertex_shader_layer** allows gl_Layer to be written from a vertex shader

- Good example where NVIDIA adopts vendor extensions for obvious API additions
  - Generally makes OpenGL code more portable and life easier for developers in the process

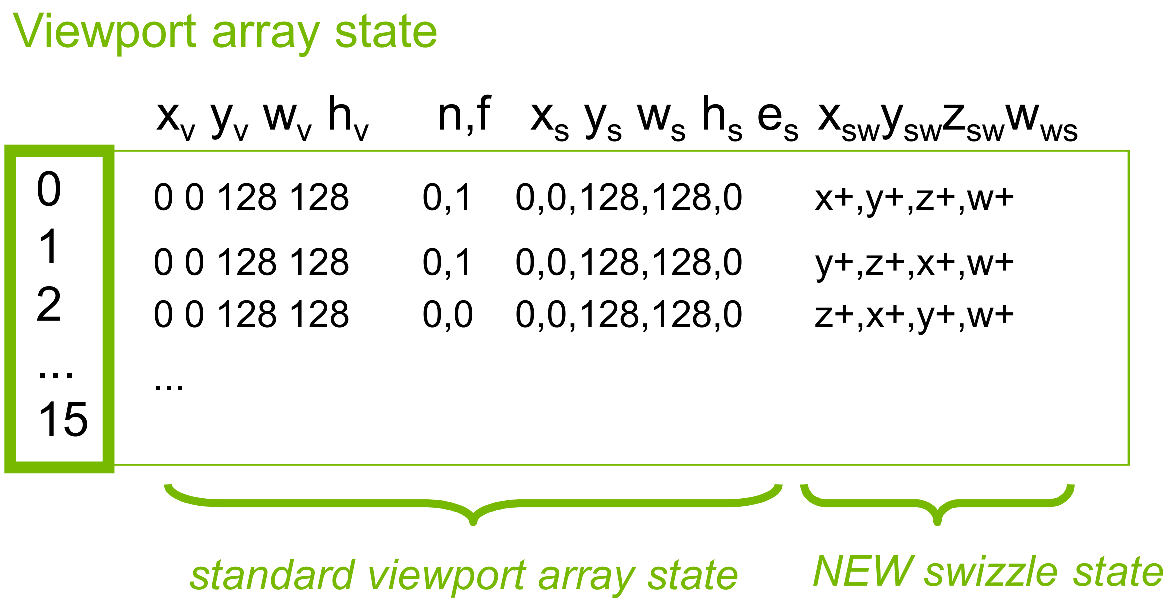
Further Extending Viewport Array State with Position Component Swizzling

Maxwell’s **NV_viewport_swizzle** extension

- Original viewport array state
  - viewport transform
  - depth range transform
  - scissor box and enable

- Maxwell extension adds new state
  - four position component swizzle modes
  - one for clip-space X, Y, Z, and W

- Eight allowed modes
  - GL_VIEWPORT_SWIZZLE_POSITIVE_X_NV
  - GL_VIEWPORT_SWIZZLE_NEGATIVE_X_NV
  - GL_VIEWPORT_SWIZZLE_POSITIVE_Y_NV
  - GL_VIEWPORT_SWIZZLE_NEGATIVE_Y_NV
  - GL_VIEWPORT_SWIZZLE_POSITIVE_Z_NV
  - GL_VIEWPORT_SWIZZLE_NEGATIVE_Z_NV
  - GL_VIEWPORT_SWIZZLE_POSITIVE_W_NV
  - GL_VIEWPORT_SWIZZLE_NEGATIVE_W_NV



Viewport array state

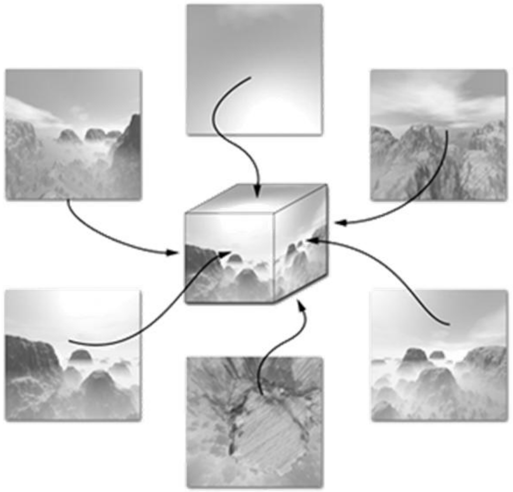
	x_v y_v w_v h_v	n,f	x_s y_s w_s h_s e_s	$x_{sw}y_{sw}z_{sw}w_{ws}$
0	0 0 128 128	0,1	0,0,128,128,0	x+,y+,z+,w+
1	0 0 128 128	0,1	0,0,128,128,0	y+,z+,x+,w+
2	0 0 128 128	0,0	0,0,128,128,0	z+,x+,y+,w+
…	…			
15				

standard viewport array state

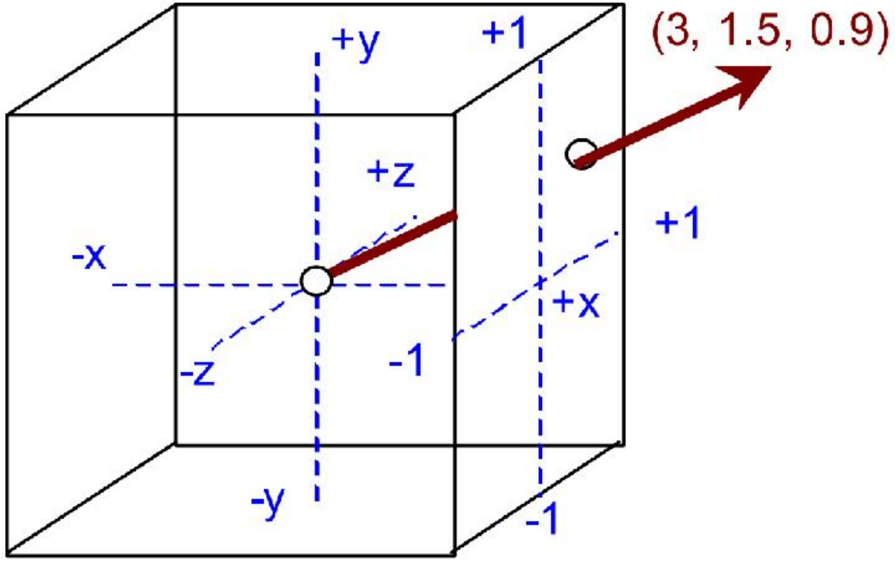
NEW swizzle state

Reminder of Cube Map Structure

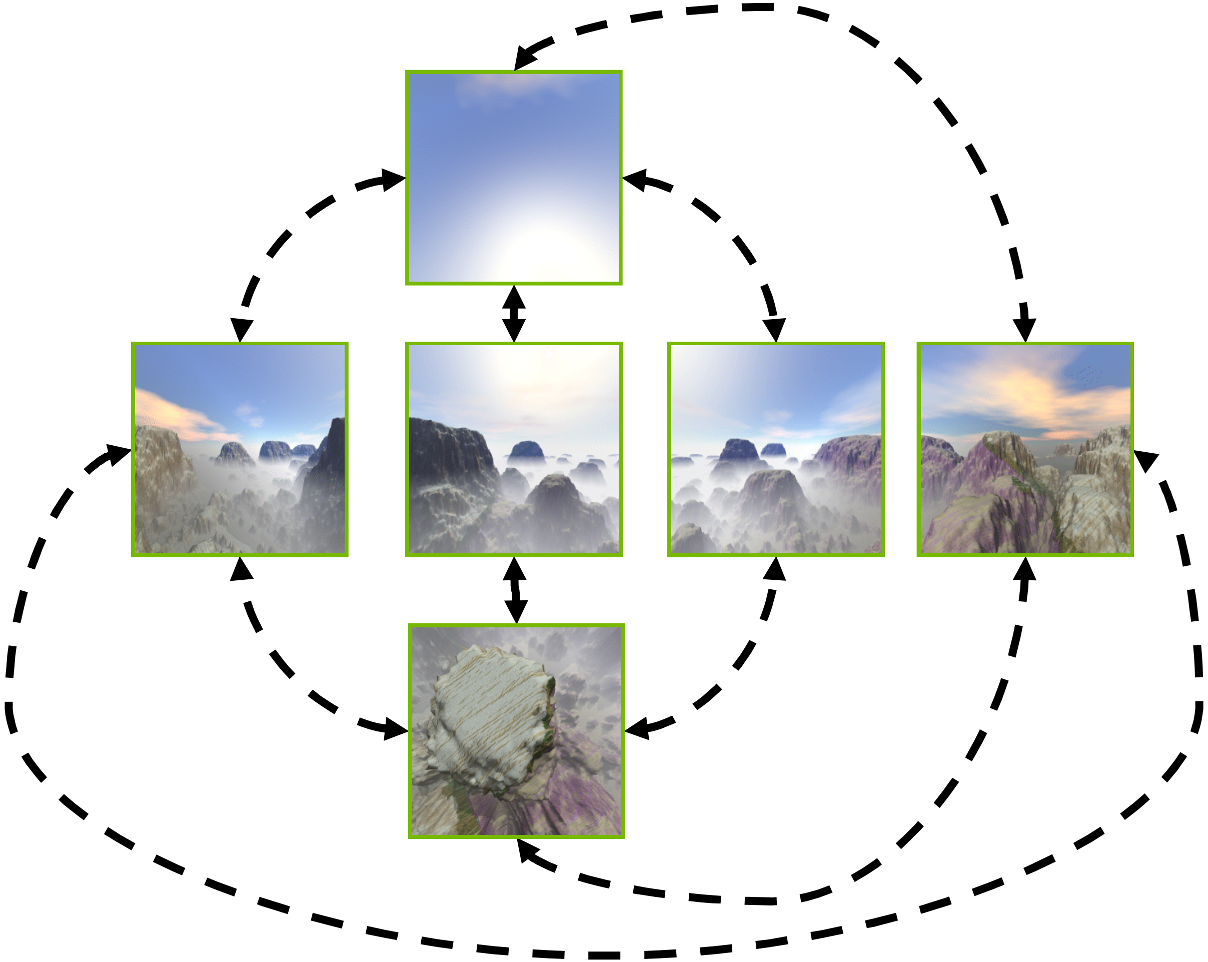
Cube Map Images are Position Swizzles Projected to 2D



- Cube map is essentially 6 images
    - Six 2D images arranged like the faces of a cube
        - +X, -X, +Y, -Y, +Z, -Z
- Logically accessed by 3D (*s*,*t*,*r*) un-normalized vector
    - Instead of 2D (*s*,*t*)
    - Where on the cube images does the vector “poke through”?
        - That’s the texture result
- Interesting question
    - *Can OpenGL efficiently render a cube map in a single rendering pass?*

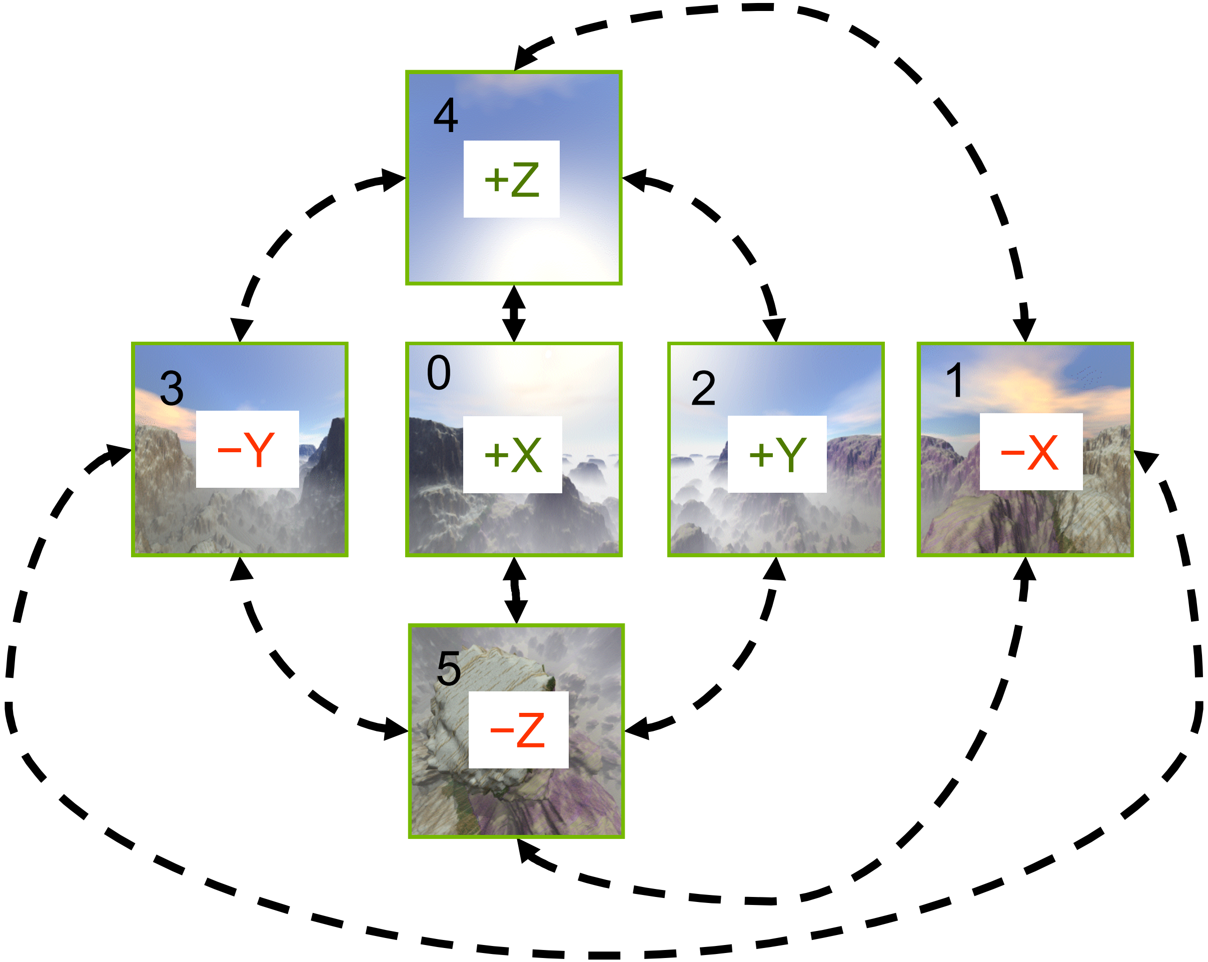


Example of Cube Map Rendering



Example of Cube Map Rendering

Faces Labeled and Numbered by Viewport Index



Layer to Render Can Be Relative to Viewport Index

Bonus Feature of Maxwell's **NV_viewport_array2** extension

- Geometry shader can "redeclare" the layer to be relative to the viewport index
  - GLSL usage
    - `layout(viewport_relative) out highp int gl_Layer;`
- After viewport mask replication, primitive's gl_Layer value is biased by its viewport index
  - Allows each viewport index to render to its "own" layer

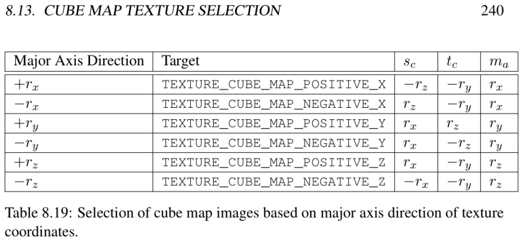
- Good for single-pass cube map rendering usage
  - Use passthrough geometry shader to write 0x3F (6 bits set, views 0 to 5) to the viewport mask
    - Usage: `gl_ViewportMask[0] = 0x3F;   // Replicate primitive 6 times`
  - Set swizzle state of each viewport index to refer to proper +X, -X, +Z,-Y, +Z, -Z cube map faces
    - Requires **NV_viewport_swizzle** extension
  - **Caveat:** Force the window-space Z to be an eye-space planar distance for proper depth testing
    - Requires inverse W buffering for depth testing
    - Swizzle each view's "Z" into output W
    - Make sure input clip-space W is 1.0 and swizzled to output Z
    - Means window-space Z will be one over W or a planar eye-space distance from eye, appropriate for depth testing
    - Requires to have floating-point depth buffer for W buffering

(Naïve) Fast Single-pass Cube Map Rendering

With Maxwell’s NV_viewport_array2 & NV_viewport_swizzle

```
#define pX GL_VIEWPORT_SWIZZLE_POSITIVE_X_NV
#define nX GL_VIEWPORT_SWIZZLE_NEGATIVE_X_NV
#define pY GL_VIEWPORT_SWIZZLE_POSITIVE_Y_NV
#define nY GL_VIEWPORT_SWIZZLE_NEGATIVE_Y_NV
#define pZ GL_VIEWPORT_SWIZZLE_POSITIVE_Z_NV
#define nZ GL_VIEWPORT_SWIZZLE_NEGATIVE_Z_NV
#define pW GL_VIEWPORT_SWIZZLE_POSITIVE_W_NV

glDisable(GL_SCISSOR_TEST);
glViewport(0, 0, 1024, 1024);
glViewportSwizzleNV(0, nZ, nY, pW, pX);  // positive X face
glViewportSwizzleNV(1, pZ, nY, pW, nX);  // negative X face
glViewportSwizzleNV(2, pX, pZ, pW, pY);  // positive Y face
glViewportSwizzleNV(3, pX, nZ, pW, nX);  // negative Y face
glViewportSwizzleNV(4, pX, nY, pW, pZ);  // positive Z face
glViewportSwizzleNV(5, nX, nY, pW, nZ);  // negative Z face
```



8.13. *CUBE MAP TEXTURE SELECTION* 240

Major Axis Direction	Target	s_c	t_c	m_a
$+r_x$	TEXTURE_CUBE_MAP_POSITIVE_X	$-r_z$	$-r_y$	r_x
$-r_x$	TEXTURE_CUBE_MAP_NEGATIVE_X	r_z	$-r_y$	r_x
$+r_y$	TEXTURE_CUBE_MAP_POSITIVE_Y	r_x	r_z	r_y
$-r_y$	TEXTURE_CUBE_MAP_NEGATIVE_Y	r_x	$-r_z$	r_y
$+r_z$	TEXTURE_CUBE_MAP_POSITIVE_Z	r_x	$-r_y$	r_z
$-r_z$	TEXTURE_CUBE_MAP_NEGATIVE_Z	$-r_x$	$-r_y$	r_z

Table 8.19: Selection of cube map images based on major axis direction of texture coordinates.

Getting swizzles from this table from the OpenGL 4.5 specification ensures your swizzles matches OpenGL’s cube map layout conventions

Viewport array state configuration

```
#extension GL_NV_geometry_shader_passthrough : require
#extension GL_NV_viewport_array2 : require

layout(triangles) in;
// No output primitive layout qualifiers required.

layout(viewport_relative) out highp int gl_Layer;

// Redeclare gl_PerVertex to pass through "gl_Position".
layout(passthrough) in gl_PerVertex {
  vec4 gl_Position;
};
// Declare "Inputs" with "passthrough" to copy members
  attributes
layout(passthrough) in Inputs {
  vec2 texcoord;
  vec4 baseColor;
};

void main() {
  gl_ViewportMask[0] = 0x3F;  // Replicate primitive 6 times
  gl_Layer = 0;
}
```

Passthrough geometry shader

non-naïve version would perform per-face culling in shader

GPU Voxelization, typically for Global Illumination

The Other Main Justification for Viewport Swizzle

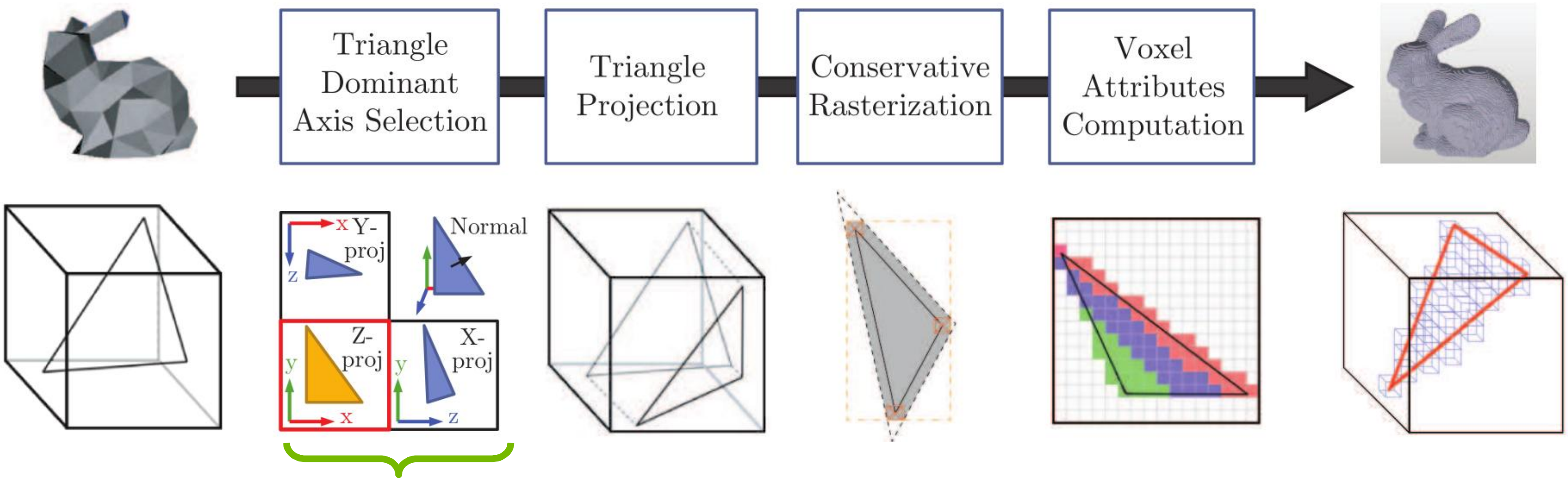
- **Concept:** desire to sample the volumetric coverage within a scene

  Ideally sampling the emittance color & directionality from the scene too

  Input: polygonal meshes

  Output: 3D grid (texture image) where voxels hold attribute values + coverage

Voxelization pipeline

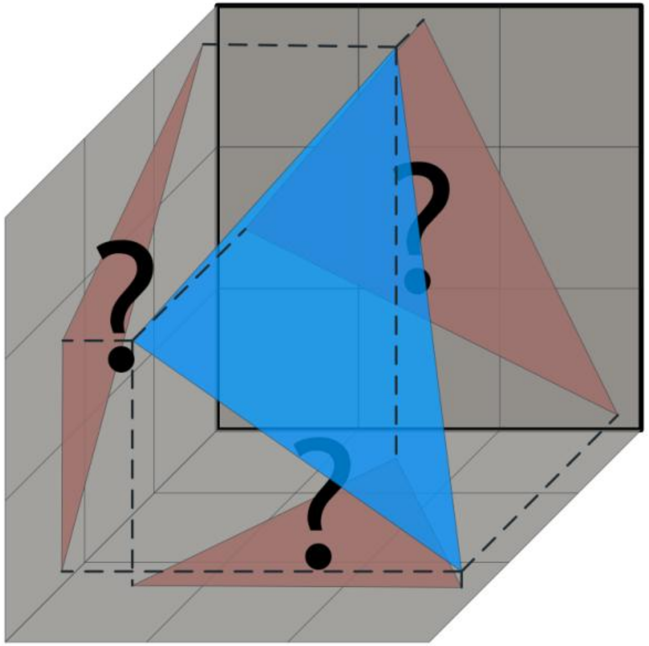

Triangle Dominant Axis Selection
Triangle Projection
Conservative Rasterization
Voxel Attributes Computation
x
Y-proj
z
Normal
Z-proj
y
x
X-proj
y
z

Passthrough geometry shader + viewport swizzle makes this fast

What's Tricky About Voxelization

Skip rendering a 2D image with pixels... because we need a 3D result

- Not your regular rasterization into a 2D image!
- Instead voxelization needs rasterizing into a 3D grid
  - Represented on the GPU as a 3D texture or other 3D array of voxels
- BUT our GPU and OpenGL only know how to rasterize in 2D
  - So exploit that by rasterizing into a "fake" 2D framebuffer
  - **ARB_framebuffer_no_attachments** extension allows rasterizing to framebuffer lacking any attachments for color or depth-stencil
  - The *logical* framebuffer has a width & height, but no pixel storage
- **Approach:** Rasterize a given triangle within the voxelization region on an orthogonal axis direction where triangle has the largest area (X, Y, or Z axis)
  - Then fragment shader does (atomic) image stores to store coverage & attributes at the appropriate (x,y,z) location in 3D grid
  - **Caveat:** Use conservative rasterization to avoid missing features

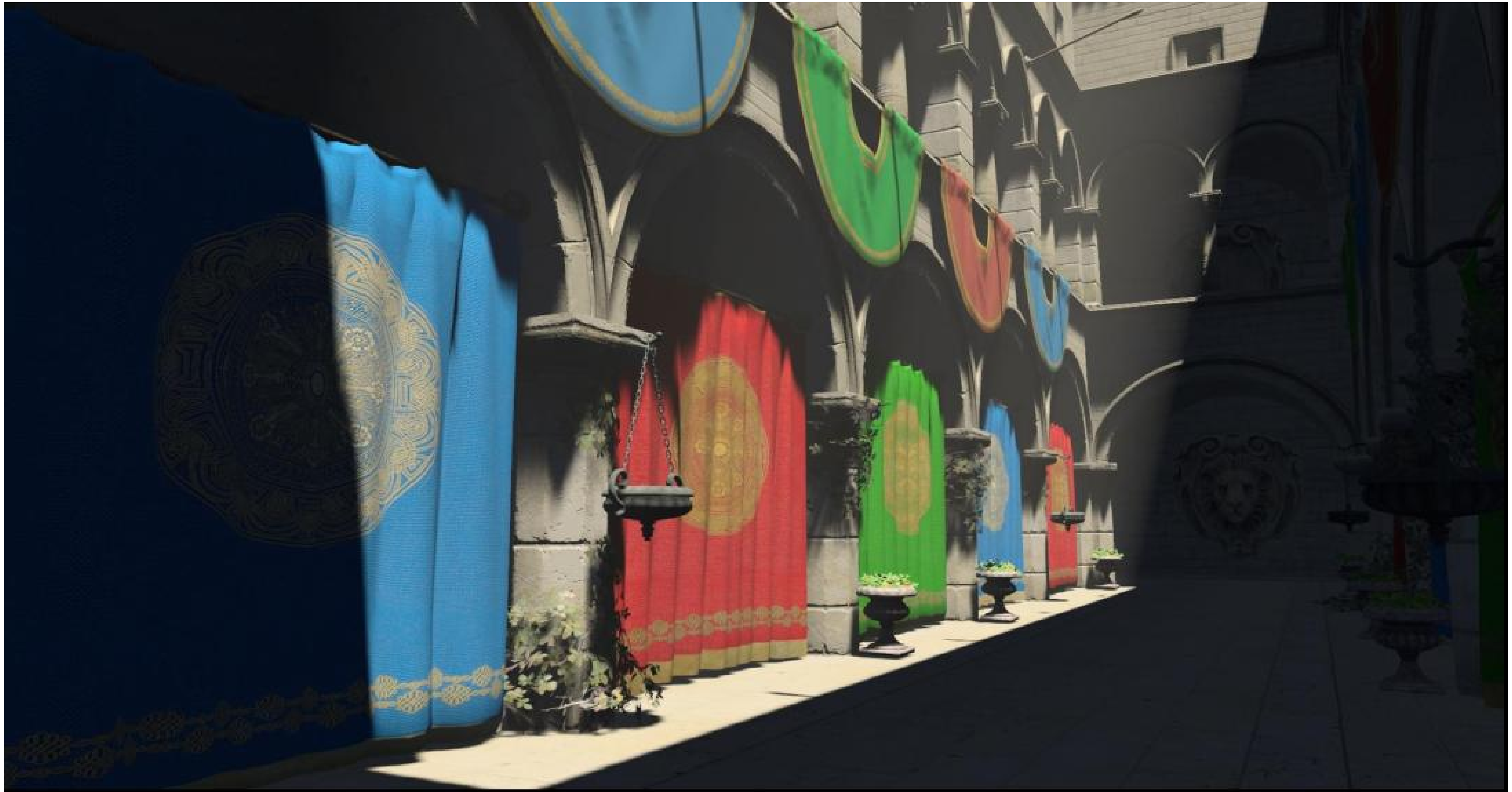


Exact details are involved, but a fast geometry shader & viewport swizzling make Dominant Axis Selection efficient

What’s the Point of Voxelization?

Feeds a GPU Global Illumination Algorithm

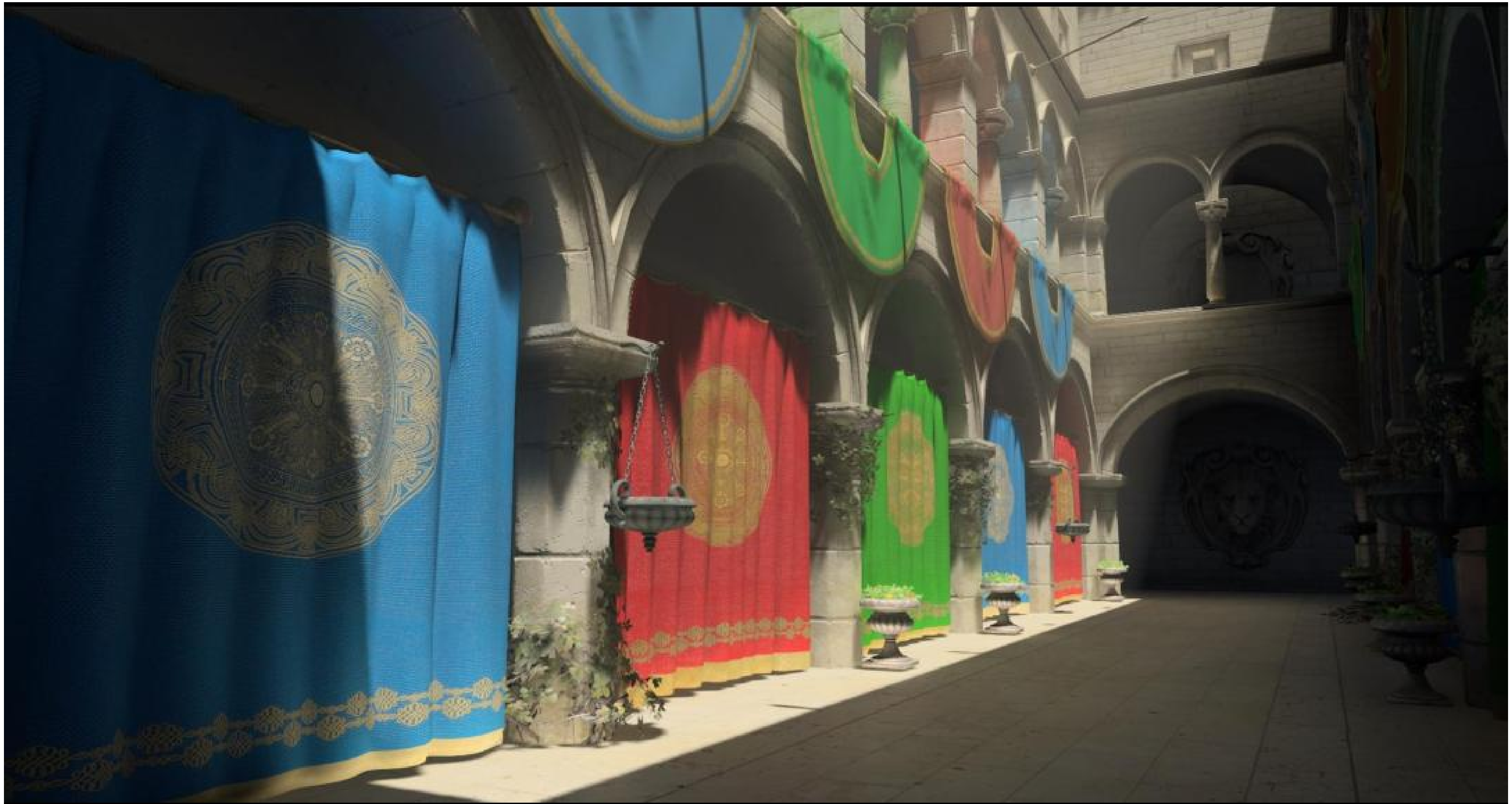
Direct lighting feels over dark



What’s the Point of Voxelization?

Feeds a GPU Global Illumination Algorithm

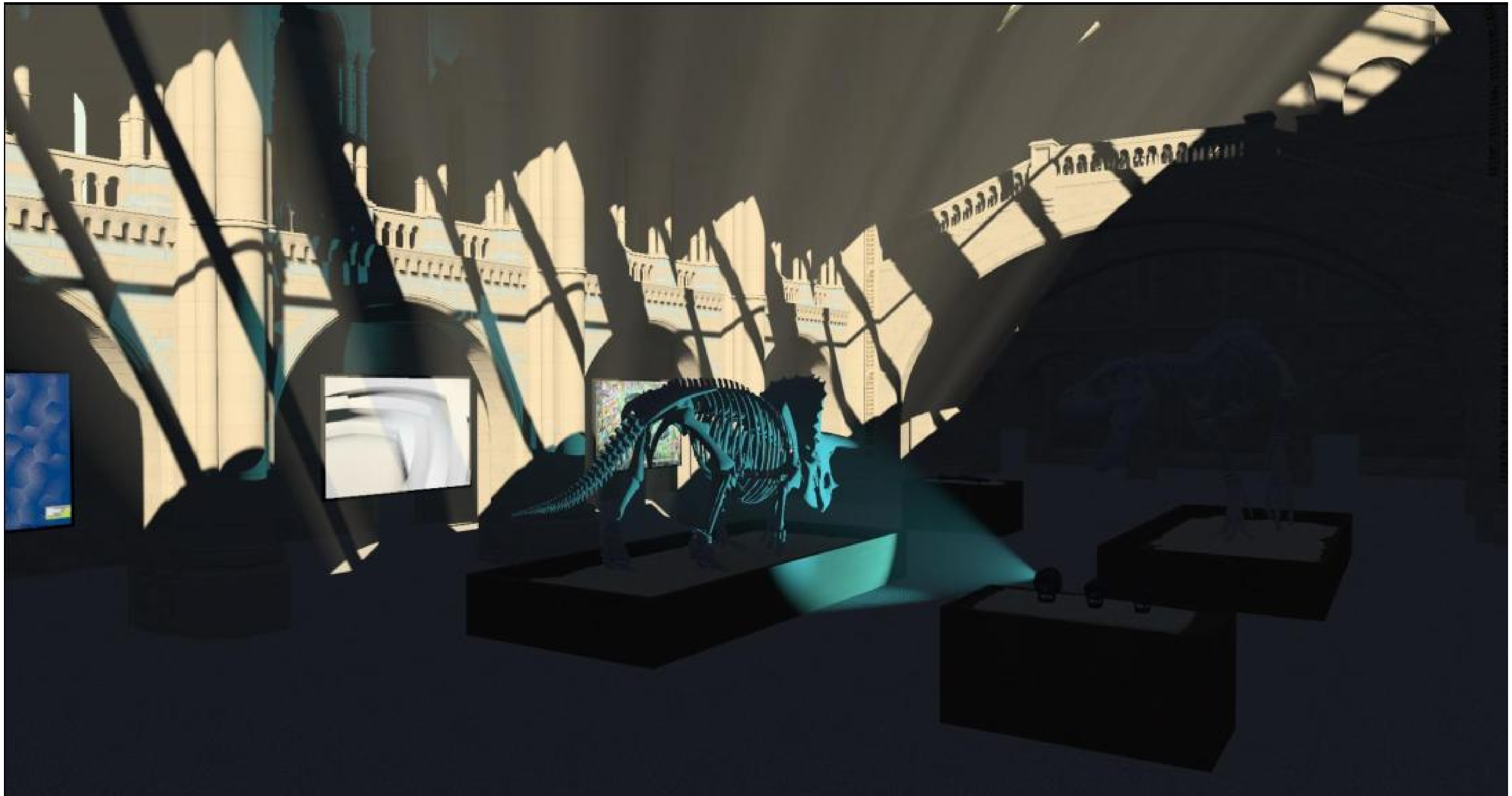
Global illumination with ambient occlusion avoids the over-dark feel



What’s the Point of Voxelization?

Feeds a GPU Global Illumination Algorithm

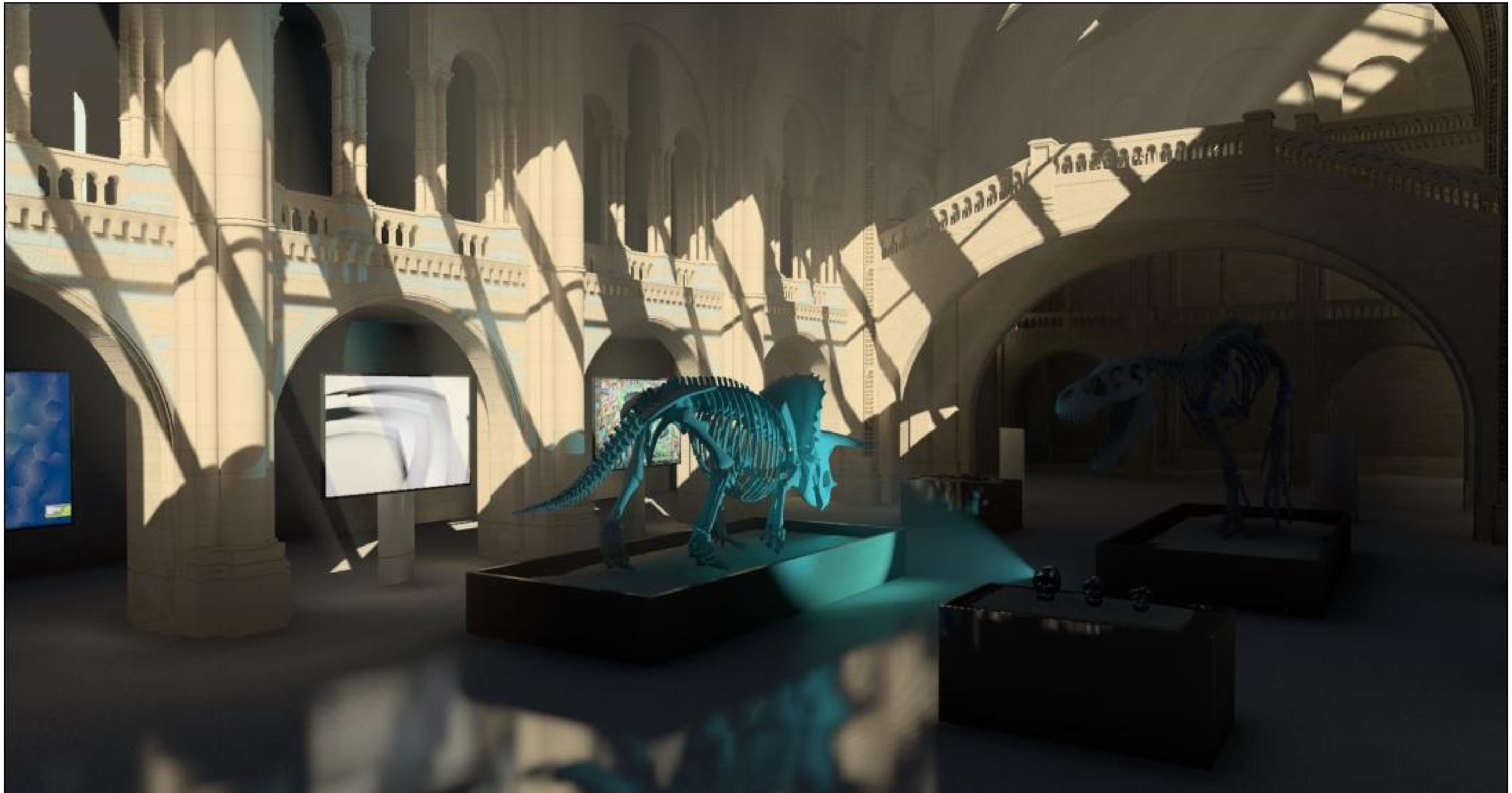
Direct lighting feels over dark



What’s the Point of Voxelization?

Feeds a GPU Global Illumination Algorithm

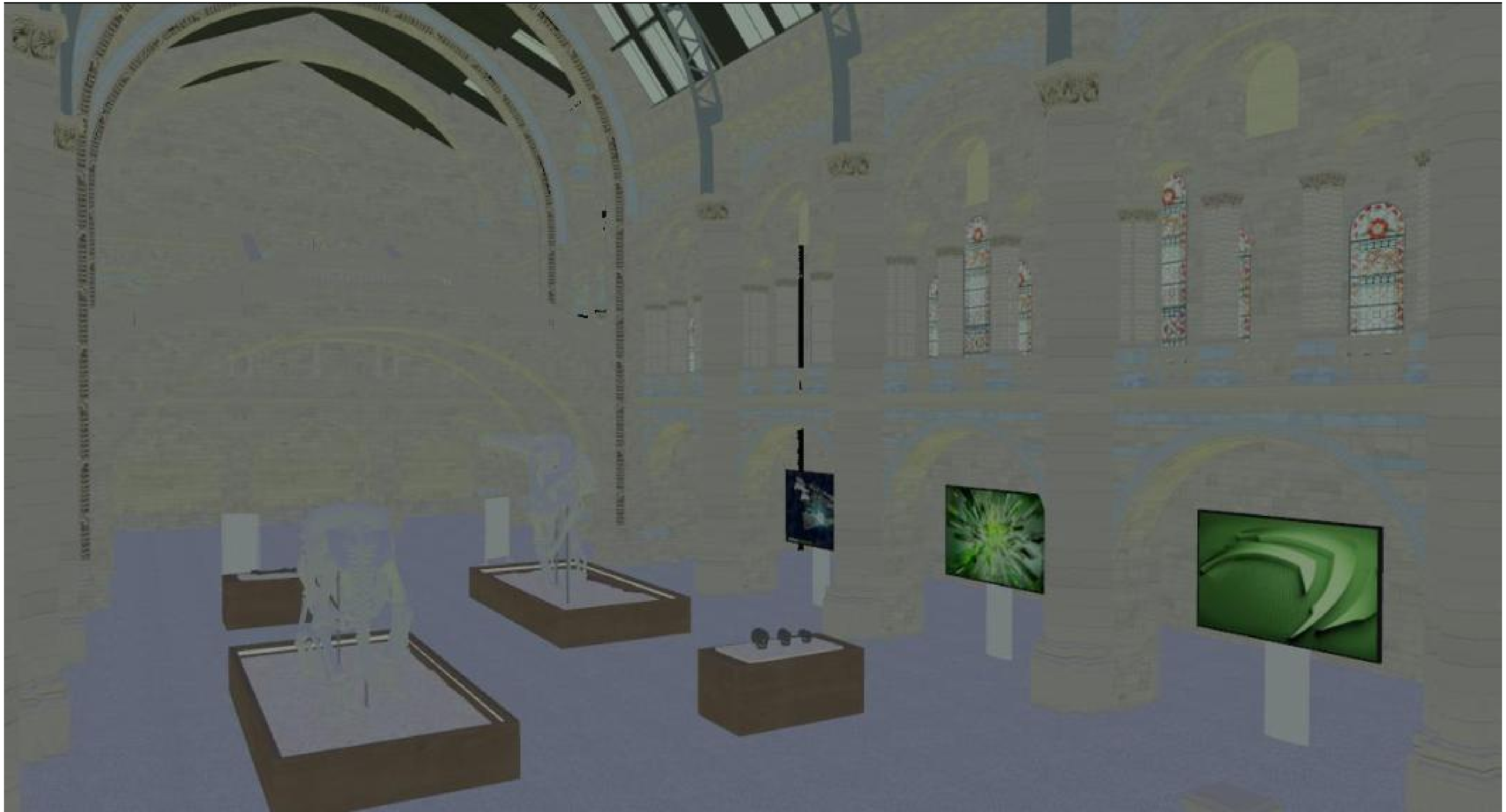
Global Illumination with specular effects capture subtle reflections in floor too



What’s the Point of Voxelization?

Improving the Ambient Contribution on Surfaces

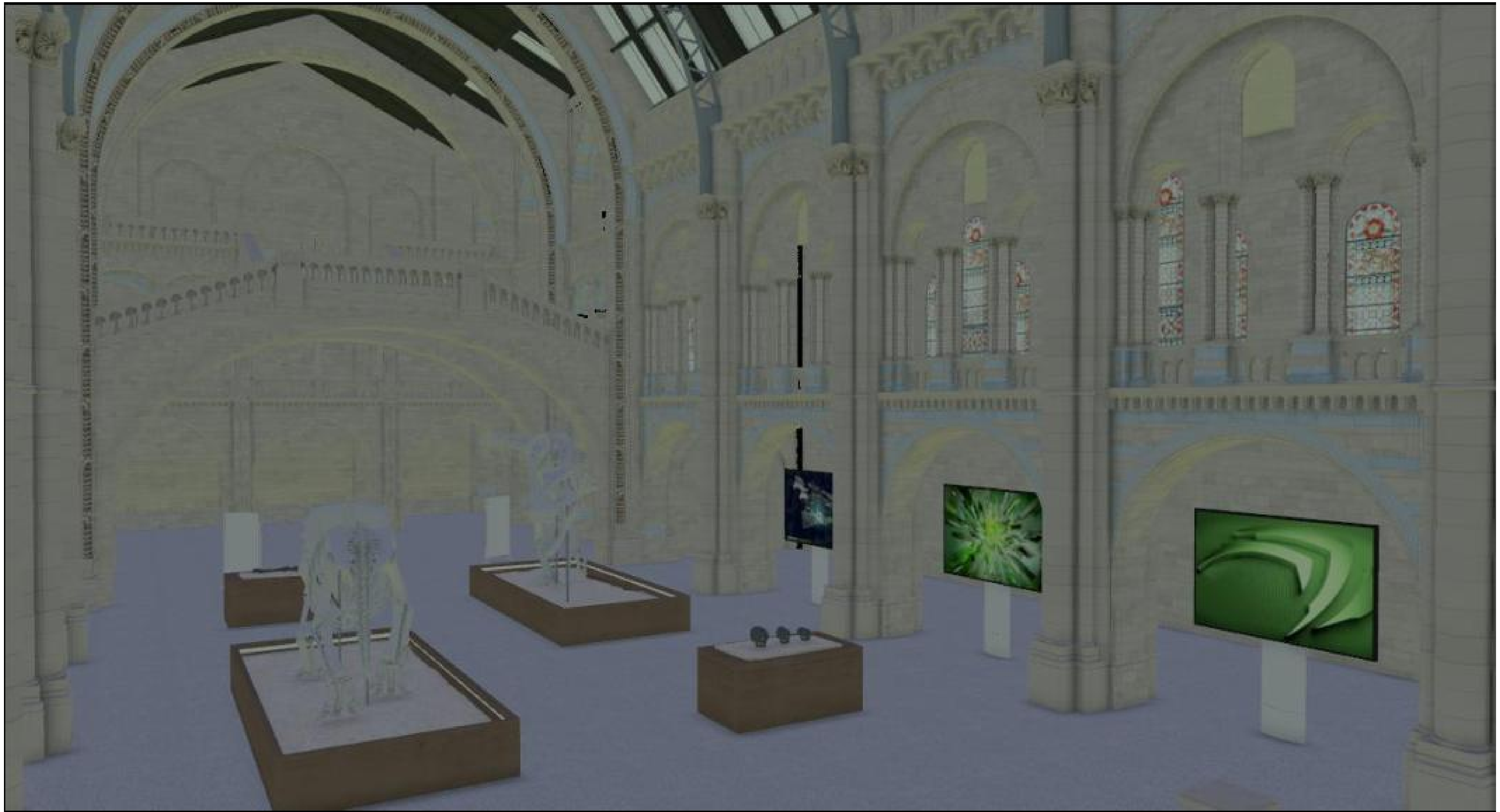
Flat ambient (no diffuse or specular directional lighting shown)



What’s the Point of Voxelization?

Improving the Ambient Contribution on Surfaces

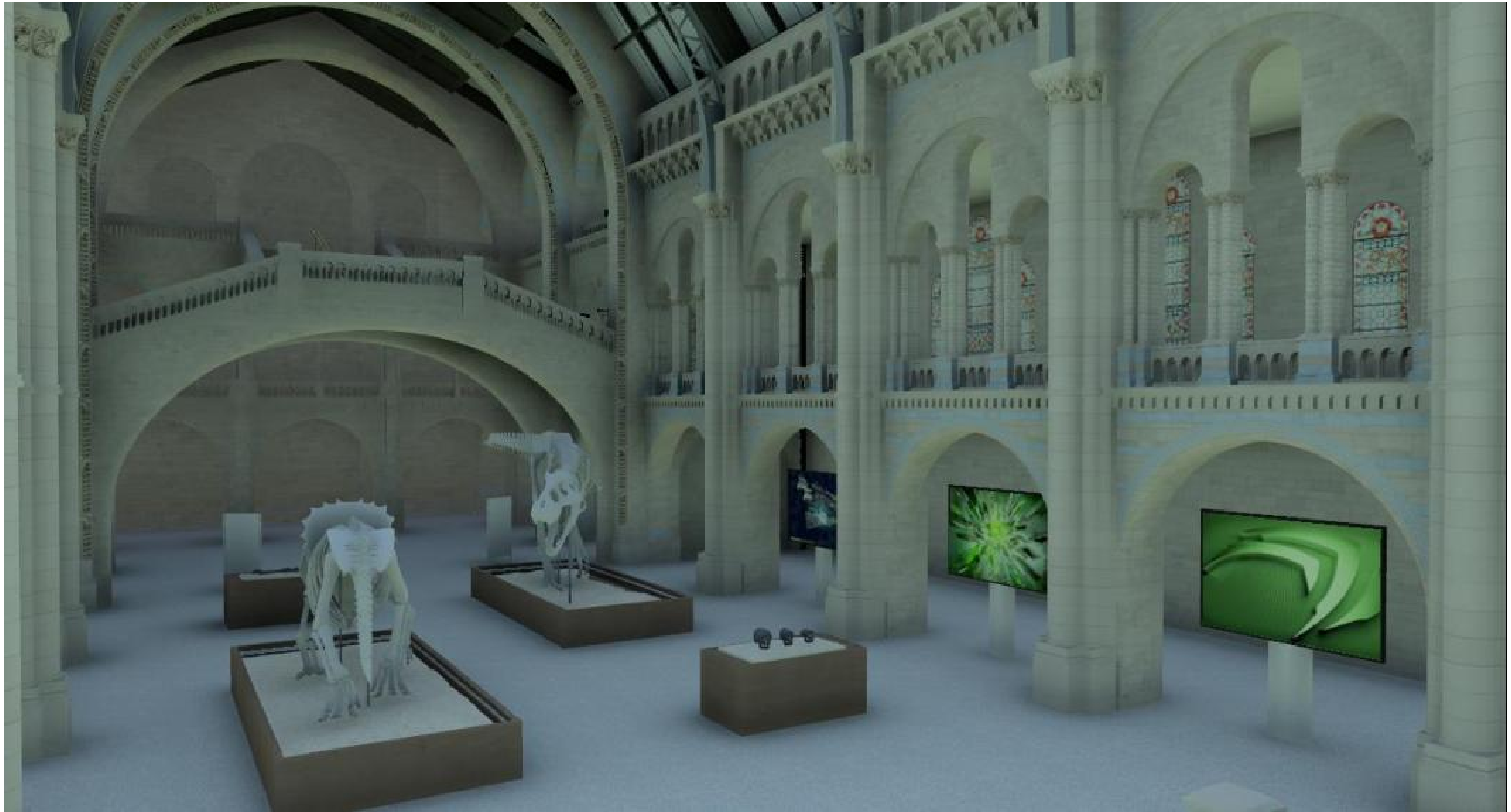
Screen-space ambient occlusion improves the sense of depth a little



What’s the Point of Voxelization?

Improving the Ambient Contribution on Surfaces

True global illumination for ambient makes the volumetric structure obvious

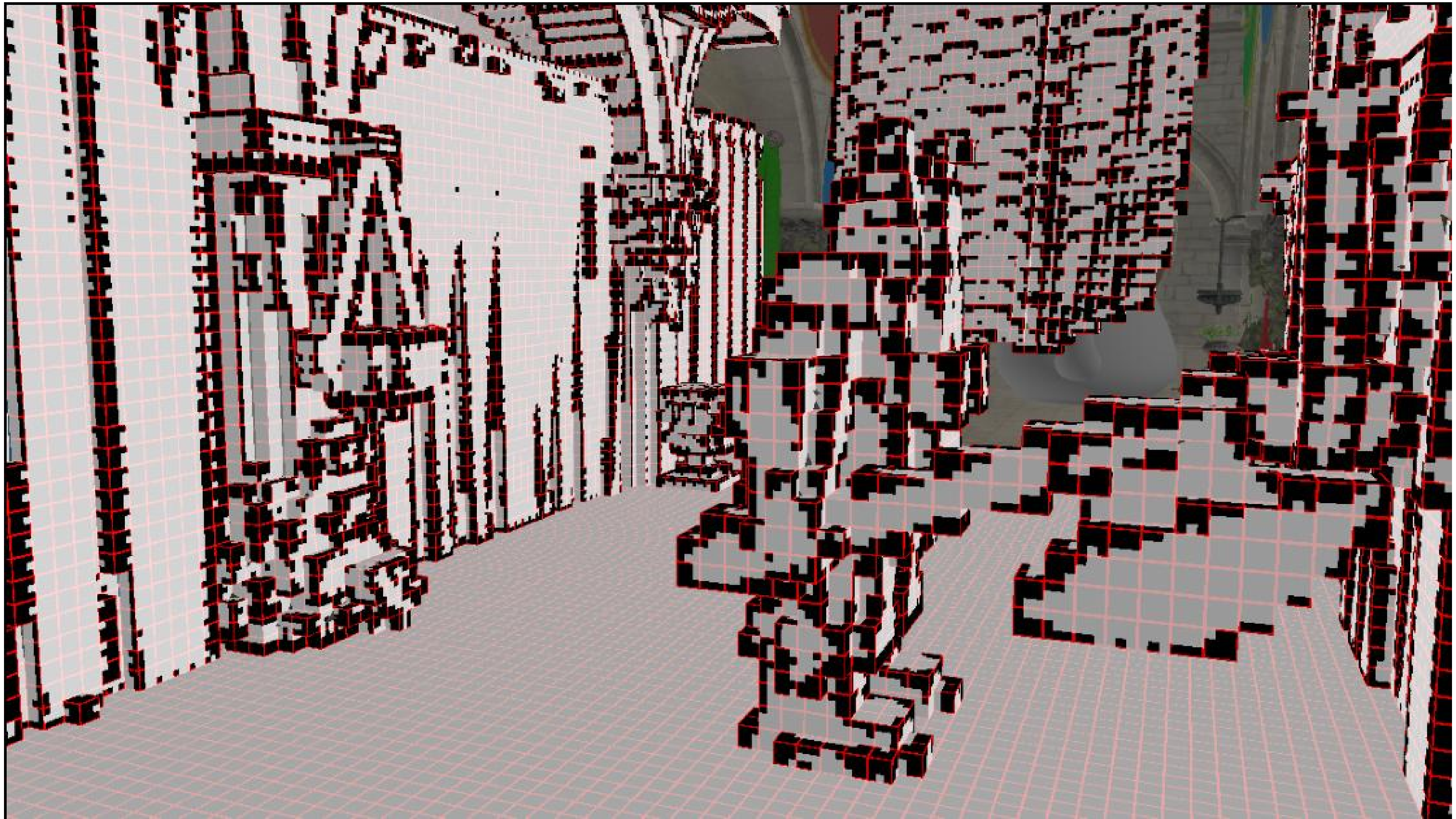


Example Voxelization

Sample scene

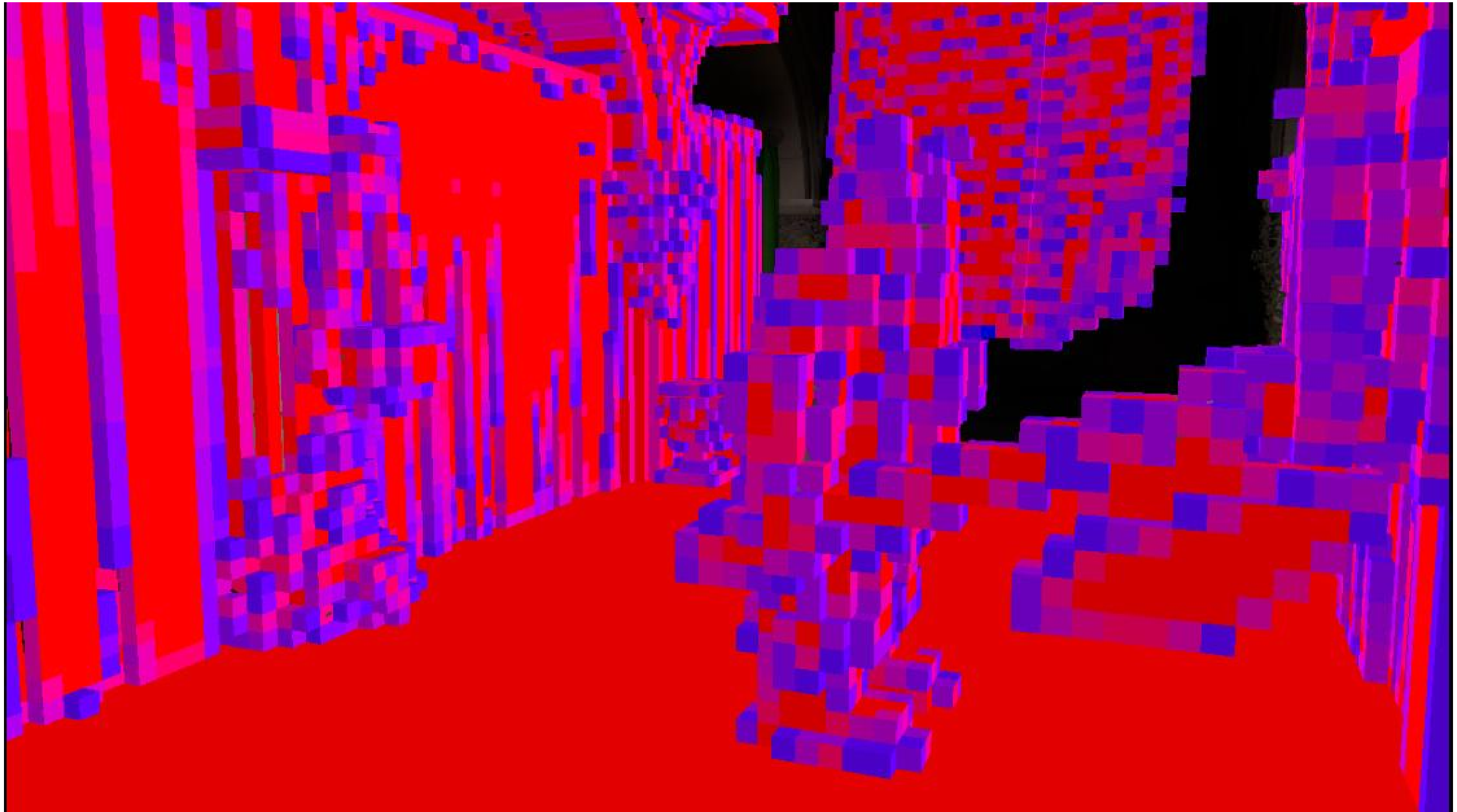
Example Voxelization

Voxelized directional coverage



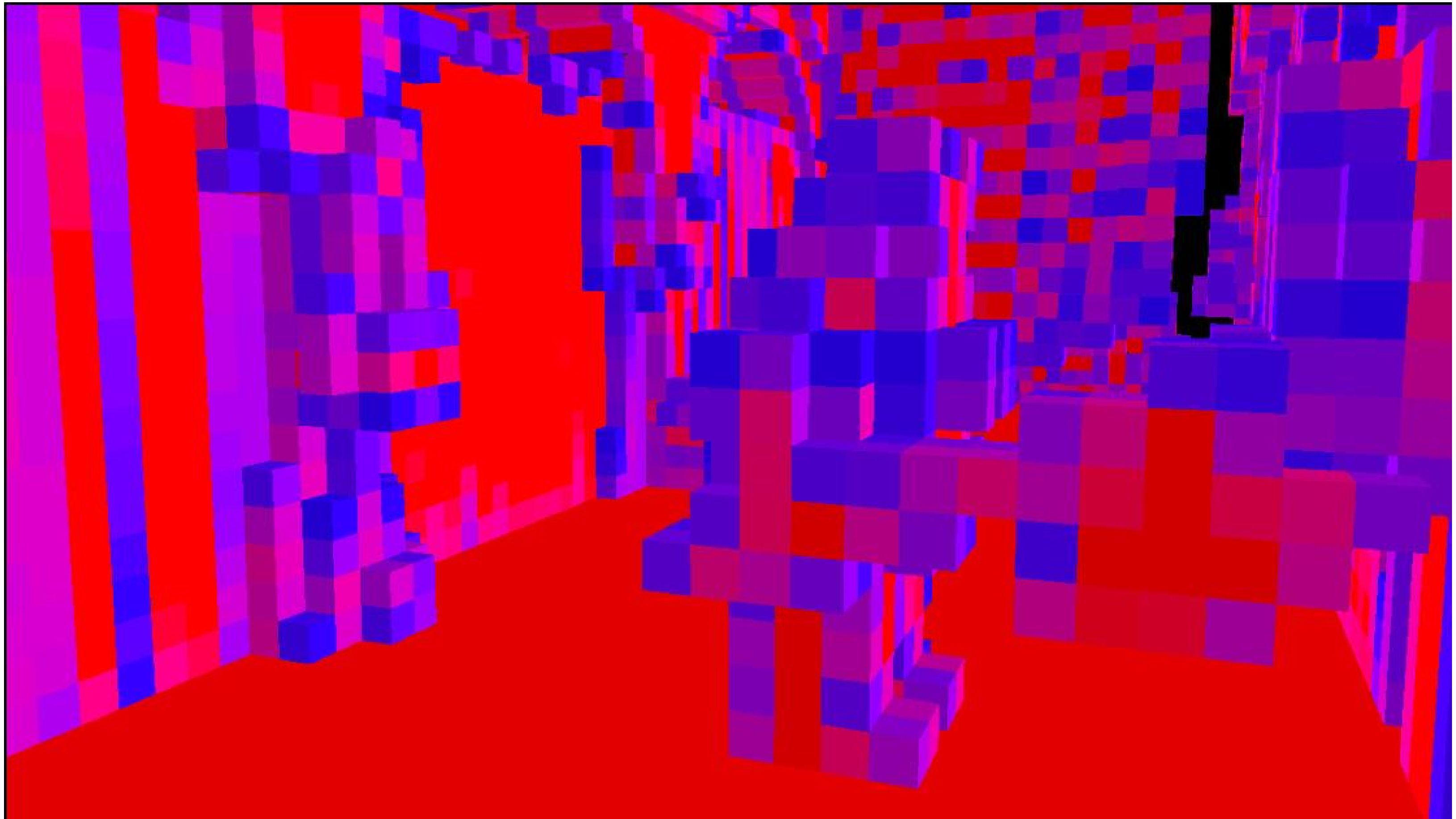
Example Voxelization

Voxelized opacity



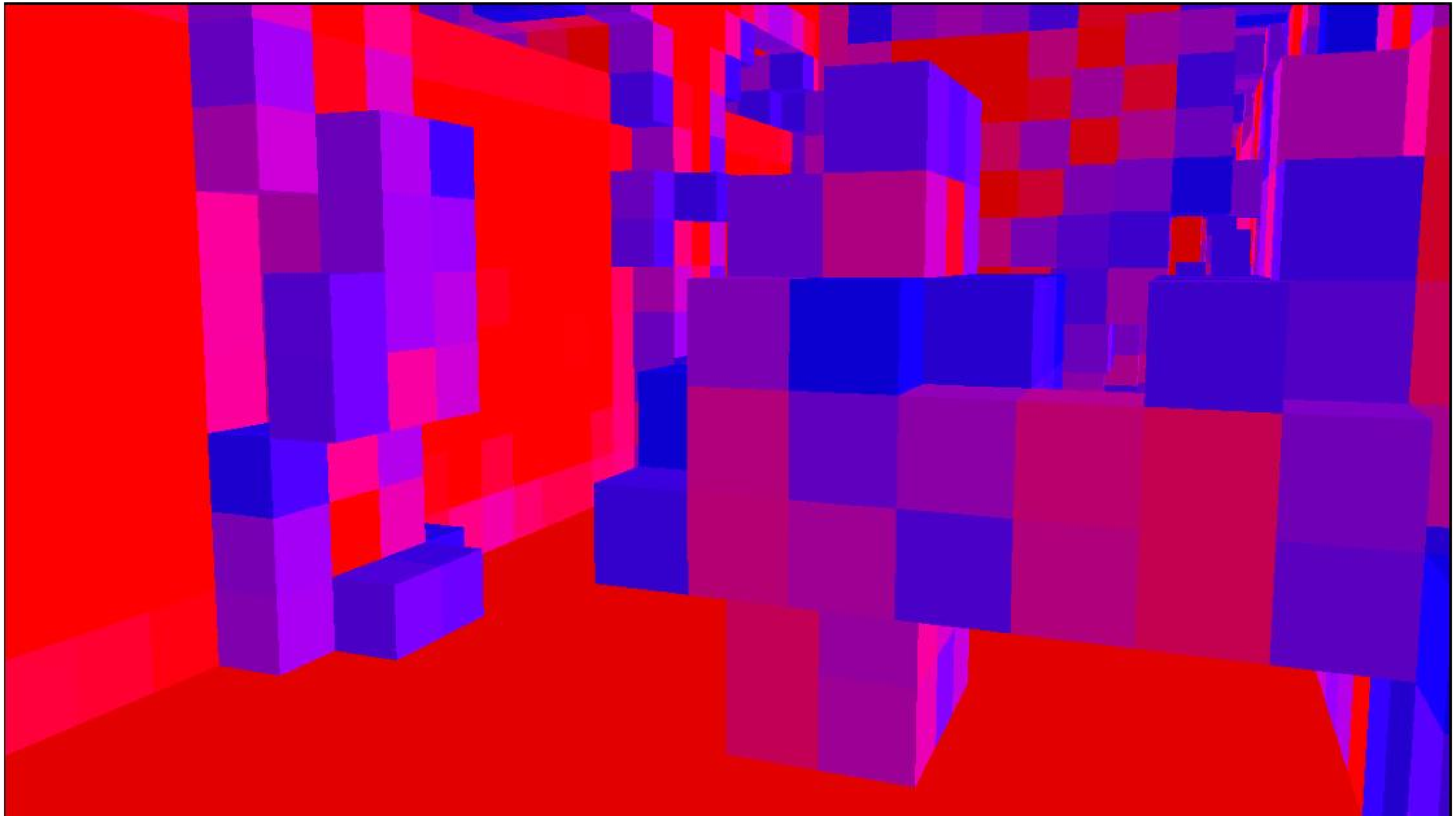
Example Voxelization

Voxelized opacity, downsampled



Example Voxelization

Voxelized opacity, downsampled twice

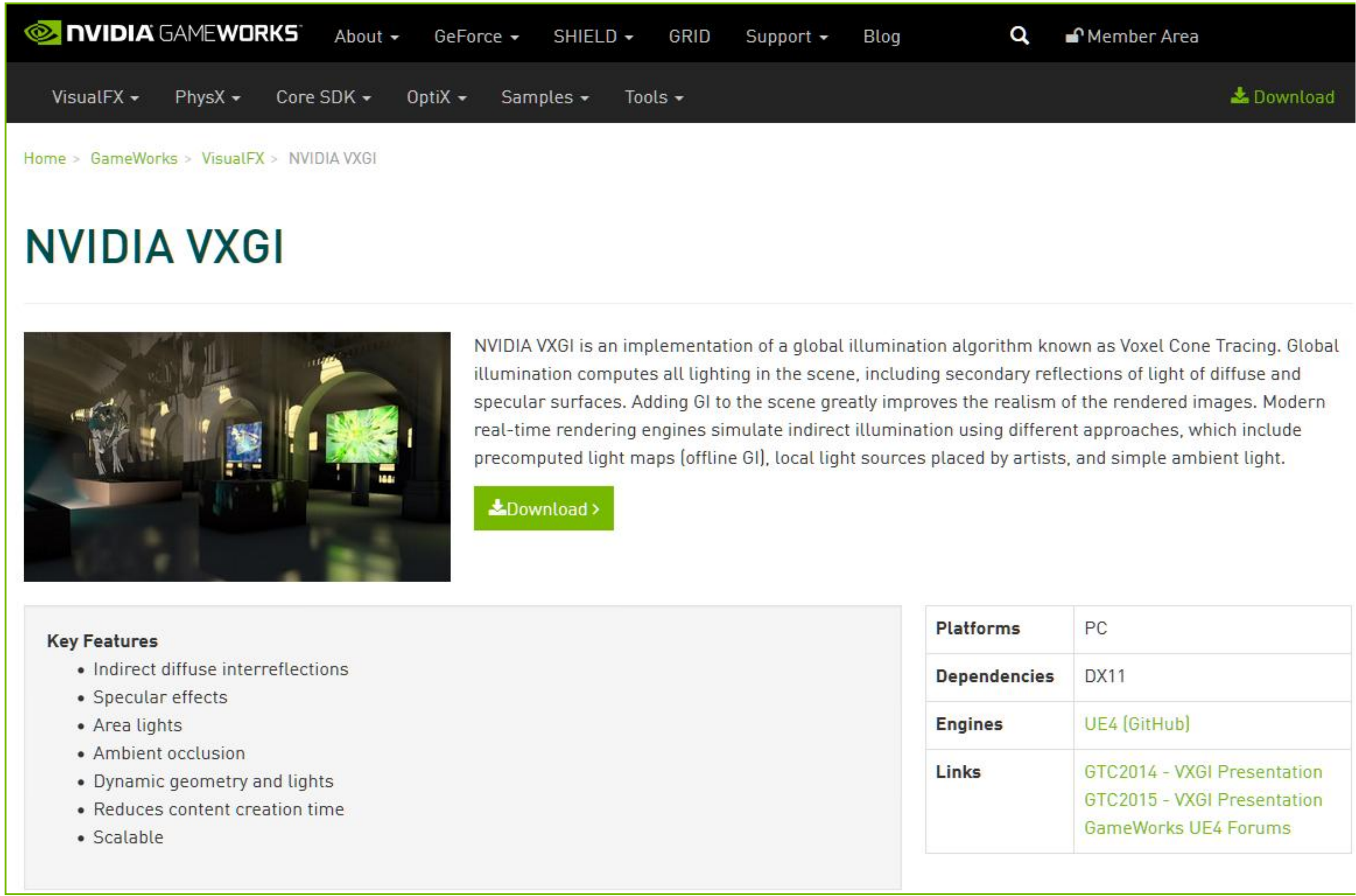


Complete Global Illumination is Complex

NVIDIA Provides Implementations

- Complete implementation included in **NVIDIA VXGI**
  - Implements Voxel Cone Tracing
  - Part of Visual FX solutions
- Implemented for DirectX 11
  - But all the underlying GPU technology is available as OpenGL extensions

NV_viewport_array2
NV_viewport_swizzle
NV_geometry_shader_passthrough
NV_conservative_raster



NVIDIA VXGI

NVIDIA VXGI is an implementation of a global illumination algorithm known as Voxel Cone Tracing. Global illumination computes all lighting in the scene, including secondary reflections of light of diffuse and specular surfaces. Adding GI to the scene greatly improves the realism of the rendered images. Modern real-time rendering engines simulate indirect illumination using different approaches, which include precomputed light maps (offline GI), local light sources placed by artists, and simple ambient light.

Key Features

- Indirect diffuse interreflections
- Specular effects
- Area lights
- Ambient occlusion
- Dynamic geometry and lights
- Reduces content creation time
- Scalable

Platforms	PC
Dependencies	DX11
Engines	UE4 (GitHub)
Links	GTC2014 - VXGI Presentation GTC2015 - VXGI Presentation GameWorks UE4 Forums

Conservative Rasterization

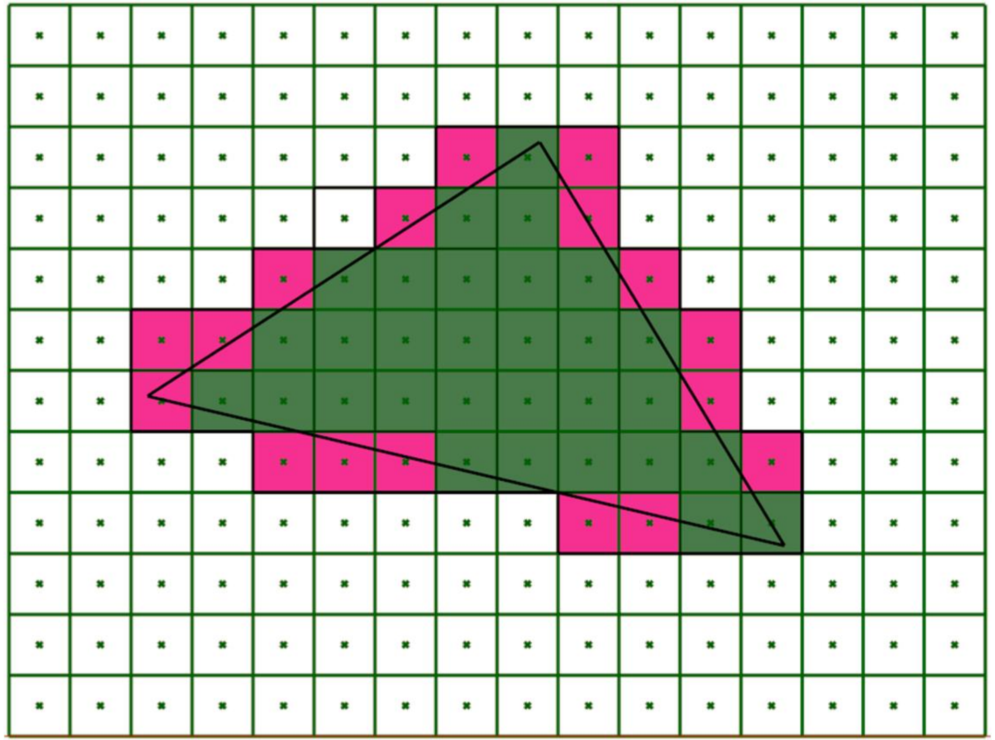
Maxwell’s **NV_conservative_raster** extension

- Mentioned on last slide as an extension used for global illumination
  - **Easy to enable:** `glEnable(GL_CONSERVATIVE_RASTERIZATION_NV);`
  - **Additional functionality:** Also provides ability to provide addition bits of sub-pixel precision
- Conventional rasterization is based on point-sampling
  - Pixel is covered if the pixel’s exact center is within the triangle
  - Multisample antialiasing = multiple pixel locations per pixels
  - Means rasterization can “miss” coverage if sample points for pixels or multisample locations are missed
  - Point sampling can under-estimate ideal coverage
- Conservative rasterization
  - Guarantees coverage if any portion of triangle intersects (overlaps) the pixel square
    - **Caveat:** *after sub-pixel snapping to the sub-pixel grid*
  - However may rasterize “extra” pixels not overlapping pixel squares intersected by the triangle
  - Conservative rasterization typically over-estimates ideal coverage
  - Intended for algorithms such as GPU voxelization where missing coverage results in rendering artifacts—and be tolerant of over-estimated coverage

Conservative Rasterization Visualized

Consider Conventional Rasterization of a Triangle

- Green pixel squares have their pixel center covered by the triangle
- Pink pixel squares intersect the triangle but do NOT have their pixel centered covered

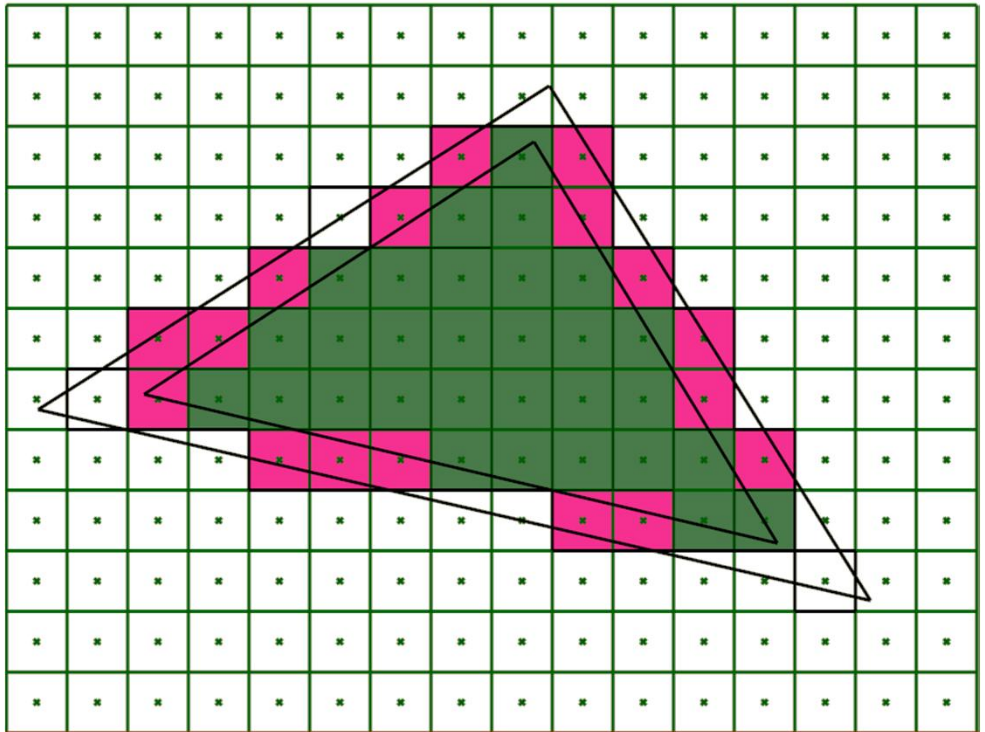


Pink pixel square indicate some degree of under-estimated coverage

Conservative Rasterization Visualized

Consider Conventional Rasterization of a Dilated Triangle

- Push triangle edges away from the triangle center (centroid) by half-pixel width
- Constructs a new, larger (dilated) triangle covering more samples

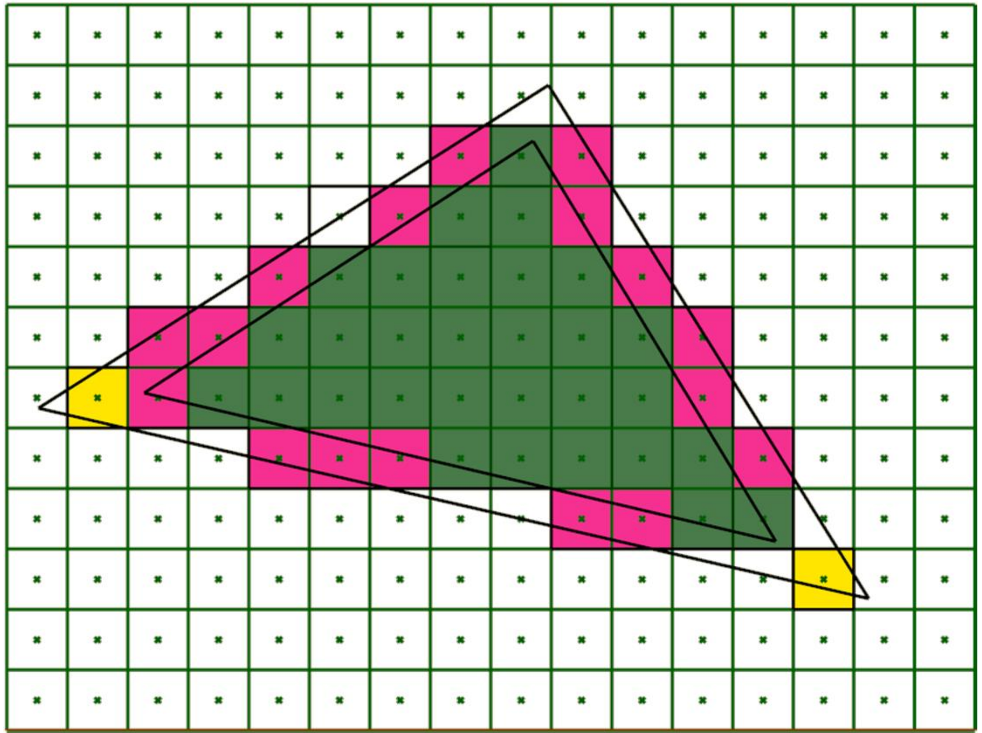


Notice all the pink pixel squares are within the dilated triangle

Conservative Rasterization Visualized

Overestimated Rasterization of a Dilated Triangle

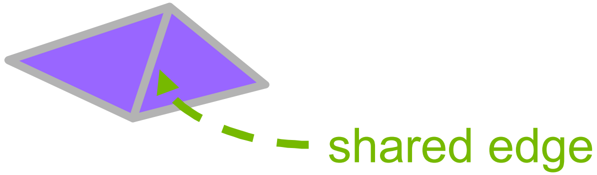
- Yellow pixel square indicate pixels within dilated triangle but not intersected by the original triangle



Notice all the yellow pixel squares are within the dilated triangle

Caveats Using Conservative Rasterization

You have been warned



- Shared edges of non-overlapping rasterized triangles are guaranteed not to have either
  - Double-hit pixels
  - Pixel gaps
- Rule is known as "watertight rasterization"
  - Very useful property in practice
  - **Example:** avoids double blending at edges
  - Coverage can be under-estimated; long, skinny triangles might cover zero samples
- Interpolation at a covered pixel center (or sample locations when multisampling) are guaranteed to return values within bounds of primitives vertex attributes

- Conservative rasterization makes no such guarantee against double-hit pixels
- Indeed double-hit pixels are effective guaranteed along shared triangle edges
- Algorithms using conservative rasterization must be tolerant of over-estimated coverage
  - Long, skinny triangles have more dilation over-estimated coverage error
- Interpolation can become extrapolation when interpolation location is not within the original primitive!

Conservative Rasterization Dilate Control

Maxwell’s **NV_conservative_raster_dilate** extension

Provides control to increase the amount of conservative dilation when `GL_CONSERVATIVE_RASTERIZATION_NV` is enabled

Straightforward usage

```
  glConservativeRasterParameterfNV (GL_CONSERVATIVE_RASTER_DILATE_NV, 0.5f);
```

0.5 implies an additional half-pixel offset to the dilation, so extra conservative

Actual value range is [0, 0.75] in increments of 0.25

Initial value is 0.0

Conservative Rasterization versus Polygon Smooth

What’s the difference?

- OpenGL supports polygon smooth rasterization mode since OpenGL 1.0
  - Example usage: glEnable(GL_POLYGON_SMOOTH)
- glEnable(GL_CONSERVATIVE_RASTERIZATION_NV) is different from glEnable(GL_POLYGON_SMOOTH)?
  - Subtle semantic difference
- NVIDIA implements GL_POLYGON_SMOOTH by computing *point-inside-primitive* tests at multiple sample locations within each pixel square
  - So computes *fractional coverage* used to modulate alpha component post-shading
  - Typically recommended for use with glBlendFunc(GL_SRC_ALPHA_SATURATE, GL_ONE) blending enabled
  - Polygon smooth should not over-estimate fractional coverage
- Conservative rasterization works by dilation, as explained
  - Conservative rasterization does not compute a fractional coverage
  - So there is no modulation of alpha by the fractional coverage

Maxwell Vector Graphics Improvements

Maxwell's **NV_framebuffer_mixed_samples** Extension

- **Simple idea:** mixed sample counts
  - Improve antialiasing quality & performance of vector graphics rendering
  - Every color samples gets N stencil/depth samples
- Notion of stencil-depth test changes
  - OLD notion: stencil & depth tests must either fail or pass, Boolean result
  - NEW notion: multiple stencil & depth values per color sample mean the stencil & depth test can "fractionally pass"
- GPU automatically modulates post-shader RGBA color by fractional test result
  - Assumes blending configured
  - Similar to fractional coverage blending in CPU-based vector graphics

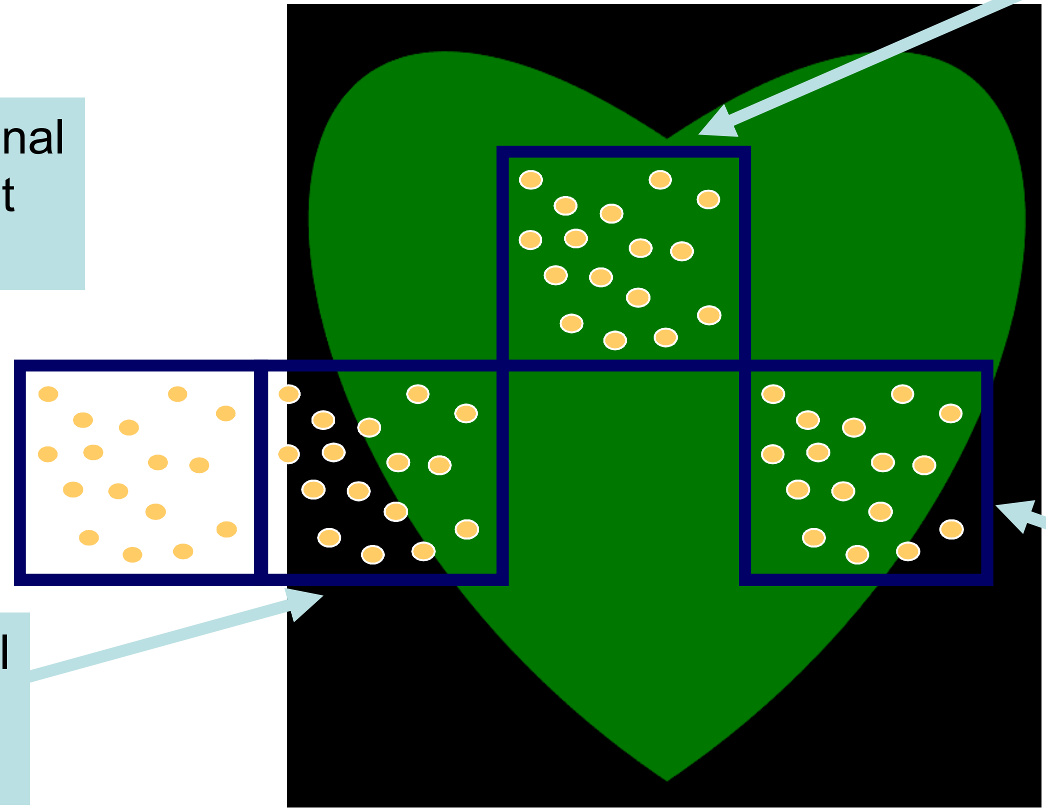
- Advantages
  - Works very cleanly with NV_path_rendering
  - Much reduced memory footprint
    - ¼ at same coverage quality
  - Much less memory bandwidth
  - Superior path rendering anti-aliasing quality, up to 16x
  - Minimal CPU overhead
    - Maxwell provides super- efficient "cover" operation

```
glCoverageModulationNV(GL_RGBA);
```

16:1 Fractional Stencil Test Example

Examine Fractional Stencil Test Results

**1 color sample,
16 stencil samples**



0% fractional stencil test (0 of 16)

100% fractional stencil test (16 of 16)

37.5% fractional stencil test (6 of 16)

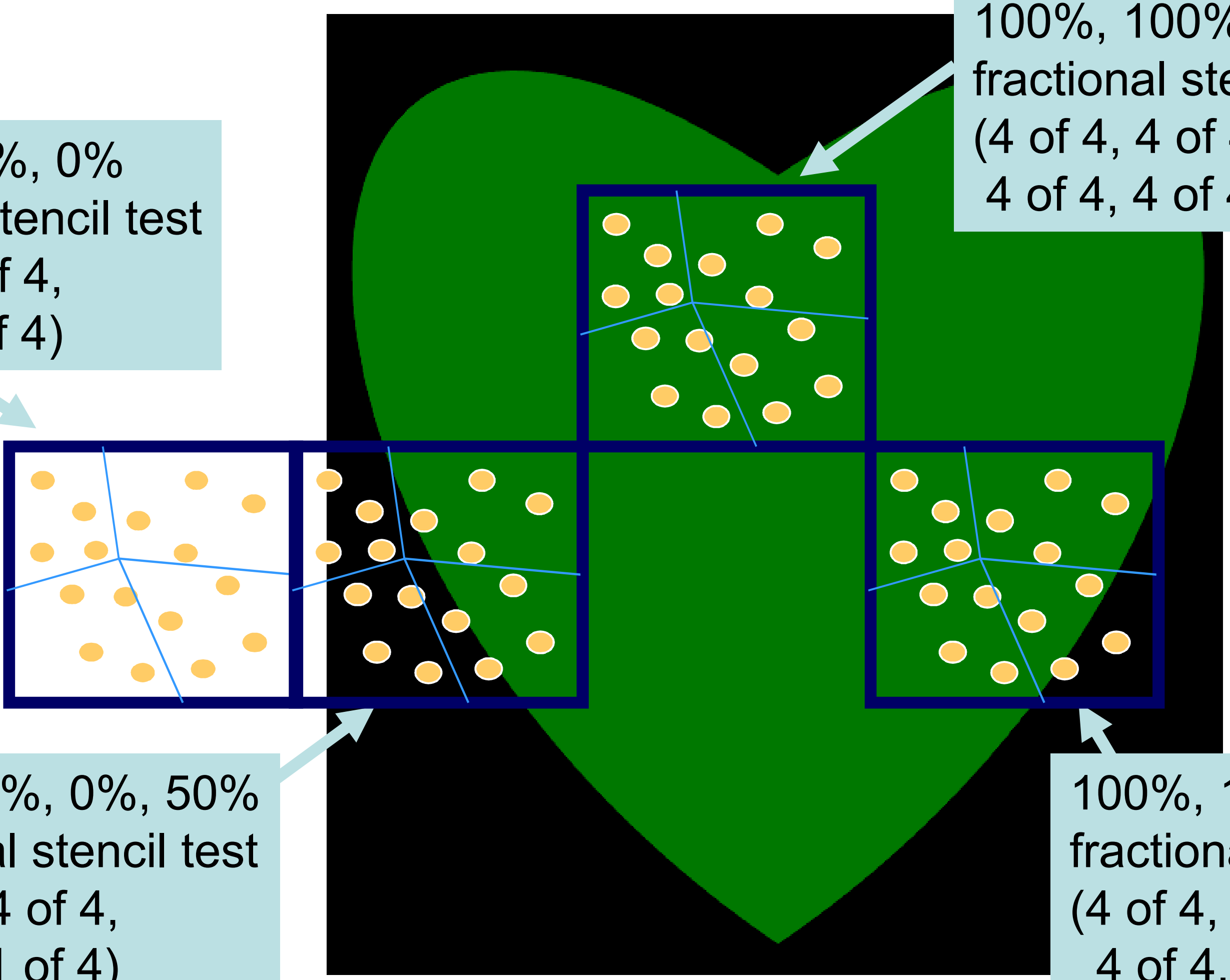
87.5% fractional stencil test (14 of 16)

16:4 Fractional Stencil Test Example

Examine Fractional Stencil Test Results

4 color samples,
16 stencil samples

Each color sample separately modulated and blended!



0%, 0%, 0%, 0%
fractional stencil test
(0 of 4, 0 of 4,
0 of 4, 0 of 4)

100%, 100%, 100%, 100%
fractional stencil test
(4 of 4, 4 of 4,
4 of 4, 4 of 4)

0%, 100%, 0%, 50%
fractional stencil test
(1 of 4, 4 of 4,
0 of 4, 1 of 4)

100%, 100%, 100%, 50%
fractional stencil test
(4 of 4, 4 of 4,
4 of 4, 2 of 4)

Mixed Sample Configurations

Maxwell’s **NV_framebuffer_mixed_samples** Extension

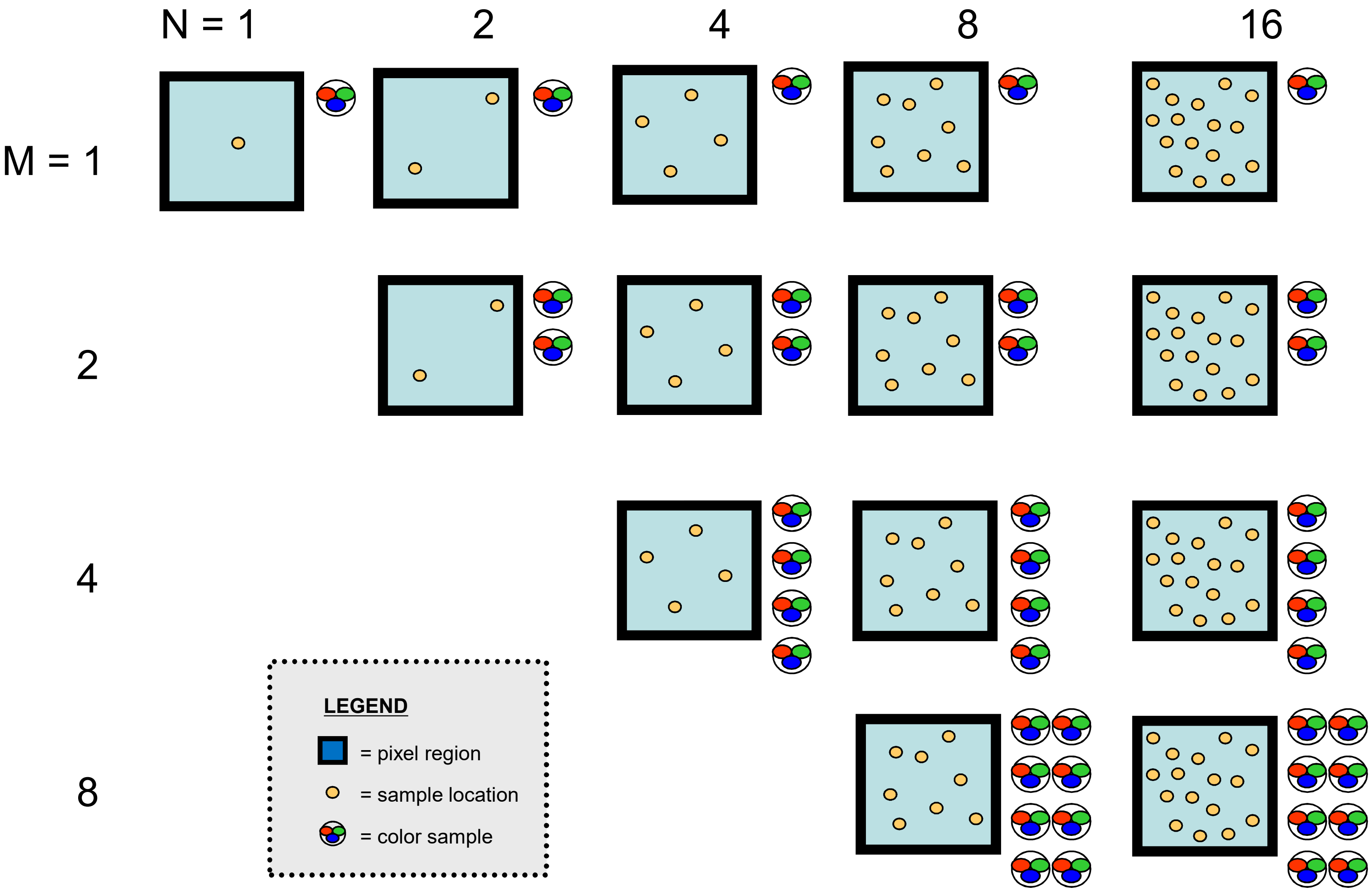
Coverage/stencil samples per pixel

Color samples per pixel

	1x	2x	4x	8x	16x
1x	1:1	2:1	4:1	8:1	16:1
2x		2:2	4:2	8:2	16:2
4x			4:4	8:4	16:4
8x				8:8	16:8

Mixed Samples Visualized

Application determines the quality/performance/memory; many choices



LEGEND

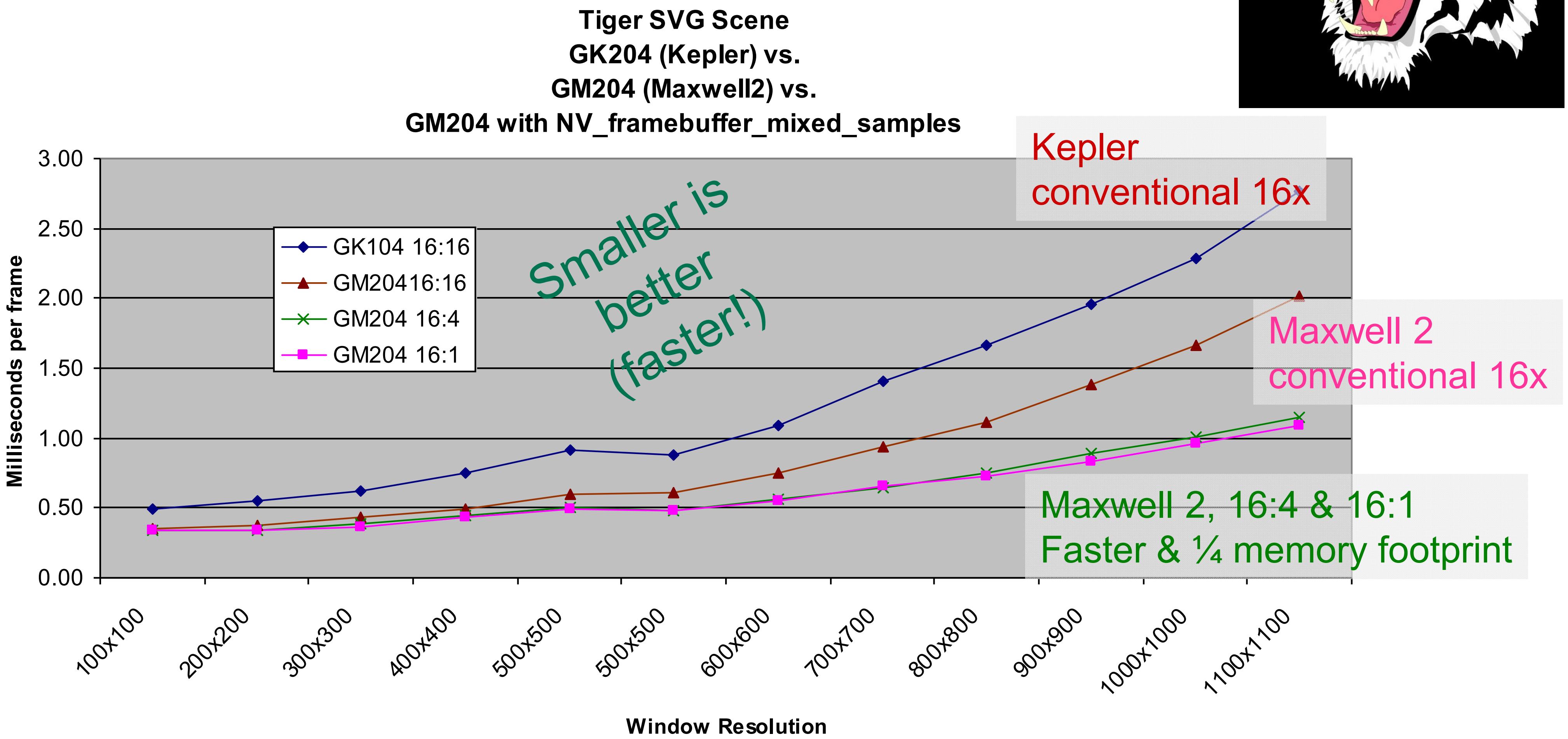
🟦 = pixel region

○ = sample location

● = color sample

Better Vector Graphics Performance

While Using Much Less Framebuffer Memory



Kepler conventional 16x

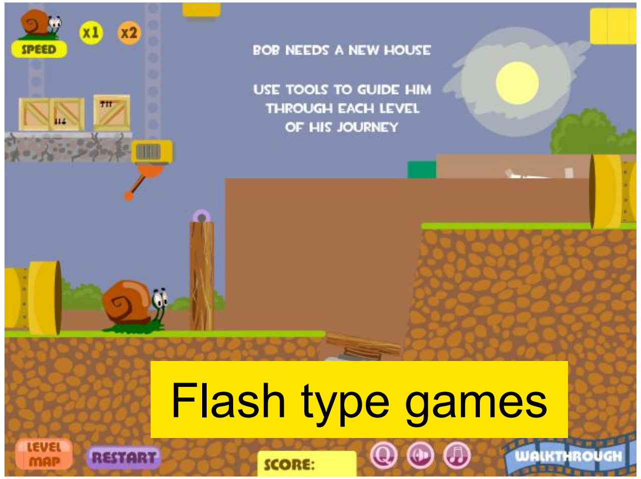
Maxwell 2 conventional 16x

Maxwell 2, 16:4 & 16:1
Faster & ¼ memory footprint

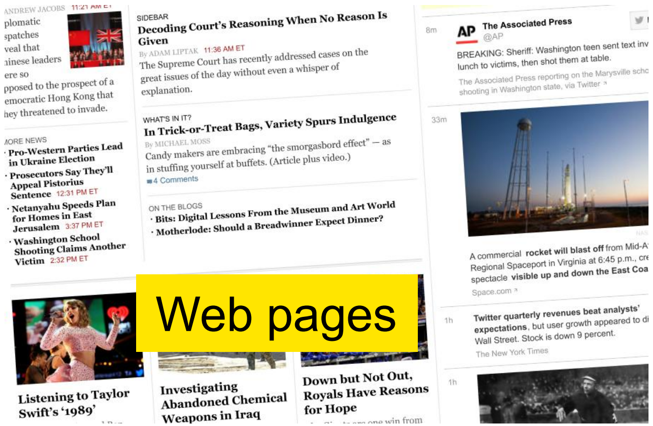
All rendering shown at 16:1 quality

Fast, Flexible Vector Graphics Results

NV_framebuffer_mixed_samples + NV_path_rendering combined



Flash type games



ANDREW JACOBS 11:21 AM ET

plomatic spatches veal that ninese leaders ere so pposed to the prospect of a emocratic Hong Kong that hey threatened to invade.

MORE NEWS

- Pro-Western Parties Lead in Ukraine Election
- Prosecutors Say They'll Appeal Pistorius Sentence 12:31 PM ET
- Netanyahu Speeds Plan for Homes in East Jerusalem 3:37 PM ET
- Washington School Shooting Claims Another Victim 2:32 PM ET

SIDEBAR

Decoding Court's Reasoning When No Reason Is Given

By ADAM LIPTAK 11:36 AM ET

The Supreme Court has recently addressed cases on the great issues of the day without even a whisper of explanation.

WHAT'S IN IT?

In Trick-or-Treat Bags, Variety Spurs Indulgence

By MICHAEL MOSS

Candy makers are embracing "the smorgasbord effect" — as in stuffing yourself at buffets. (Article plus video.)

4 Comments

ON THE BLOGS

- Bits: Digital Lessons From the Museum and Art World
- Motherlode: Should a Breadwinner Expect Dinner?

Listening to Taylor Swift's '1989'

Investigating Abandoned Chemical Weapons in Iraq

Down but Not Out, Royals Have Reasons for Hope

8m AP The Associated Press @AP

BREAKING: Sheriff: Washington teen sent text inv lunch to victims, then shot them at table.

The Associated Press reporting on the Marysville scho shooting in Washington state, via Twitter

33m

A commercial rocket will blast off from Mid-A Regional Spaceport in Virginia at 6:45 p.m., cre spectacle visible up and down the East Coa

Space.com

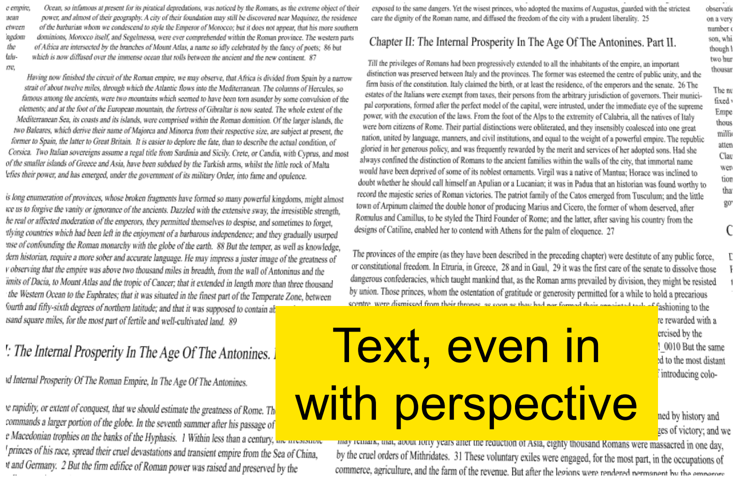
1h

Twitter quarterly revenues beat analysts' expectations, but user growth appeared to di Wall Street. Stock is down 9 percent.

The New York Times

1h

Web pages



Ocean, so infamous at present for its piratical depredations, was noticed by the Romans, as the extreme object of their power, and almost of their geography. A city of their foundation may still be discovered near Mequinez, the residence of the barbarian whom we condescend to style the Emperor of Morocco; but it does not appear, that his more southern dominions, Morocco itself, and Segelmessa, were ever comprehended within the Roman province. The western parts of Africa are intersected by the branches of Mount Atlas, a name so idly celebrated by the fancy of poets; 86 but which is now diffused over the immense ocean that rolls between the ancient and the new continent. 87

Having now finished the circuit of the Roman empire, we may observe, that Africa is divided from Spain by a narrow strait of about twelve miles, through which the Atlantic flows into the Mediterranean. The columns of Hercules, so famous among the ancients, were two mountains which seemed to have been torn asunder by some convulsion of the elements; and at the foot of the European mountain, the fortress of Gibraltar is now seated. The whole extent of the Mediterranean Sea, its coasts and its islands, were comprised within the Roman dominion. Of the larger islands, the two Baleares, which derive their name of Majorca and Minorca from their respective size, are subject at present, the former to Spain, the latter to Great Britain. It is easier to deplore the fate, than to describe the actual condition, of Corsica. Two Italian sovereigns assume a regal title from Sardinia and Sicily. Crete, or Candia, with Cyprus, and most of the smaller islands of Greece and Asia, have been subdued by the Turkish arms, whilst the little rock of Malta efies their power, and has emerged, under the government of its military Order, into fame and opulence.

is long enumeration of provinces, whose broken fragments have formed so many powerful kingdoms, might almost ce us to forgive the vanity or ignorance of the ancients. Dazzled with the extensive sway, the irresistible strength, he real or affected moderation of the emperors, they permitted themselves to despise, and sometimes to forget, tlying countries which had been left in the enjoyment of a barbarous independence; and they gradually usurped nse of confounding the Roman monarchy with the globe of the earth. 88 But the temper, as well as knowledge, dern historian, require a more sober and accurate language. He may impress a juster image of the greatness of y observing that the empire was above two thousand miles in breadth, from the wall of Antoninus and the imits of Dacia, to Mount Atlas and the tropic of Cancer; that it extended in length more than three thousand the Western Ocean to the Euphrates; that it was situated in the finest part of the Temperate Zone, between fourth and fifty-sixth degrees of northern latitude; and that it was supposed to contain ab usand square miles, for the most part of fertile and well-cultivated land. 89

: The Internal Prosperity In The Age Of The Antonines.

d Internal Prosperity Of The Roman Empire, In The Age Of The Antonines.

e rapidity, or extent of conquest, that we should estimate the greatness of Rome. Th commands a larger portion of the globe. In the seventh summer after his passage of e Macedonian trophies on the banks of the Hyphasis. 1 Within less than a century, princes of his race, spread their cruel devastations and transient empire from the Sea of China, t and Germany. 2 But the firm edifice of Roman power was raised and preserved by the

exposed to the same dangers. Yet the wisest princes, who adopted the maxims of Augustus, guarded with the strictest care the dignity of the Roman name, and diffused the freedom of the city with a prudent liberality. 25

Chapter II: The Internal Prosperity In The Age Of The Antonines. Part II.

Till the privileges of Romans had been progressively extended to all the inhabitants of the empire, an important distinction was preserved between Italy and the provinces. The former was esteemed the centre of public unity, and the firm basis of the constitution. Italy claimed the birth, or at least the residence, of the emperors and the senate. 26 The estates of the Italians were exempt from taxes, their persons from the arbitrary jurisdiction of governors. Their municipal corporations, formed after the perfect model of the capital, were intrusted, under the immediate eye of the supreme power, with the execution of the laws. From the foot of the Alps to the extremity of Calabria, all the natives of Italy were born citizens of Rome. Their partial distinctions were obliterated, and they insensibly coalesced into one great nation, united by language, manners, and civil institutions, and equal to the weight of a powerful empire. The republic gloried in her generous policy, and was frequently rewarded by the merit and services of her adopted sons. Had she always confined the distinction of Romans to the ancient families within the walls of the city, that immortal name would have been deprived of some of its noblest ornaments. Virgil was a native of Mantua; Horace was inclined to doubt whether he should call himself an Apulian or a Lucanian; it was in Padua that an historian was found worthy to record the majestic series of Roman victories. The patriot family of the Catos emerged from Tusculum; and the little town of Arpinum claimed the double honor of producing Marius and Cicero, the former of whom deserved, after Romulus and Camillus, to be styled the Third Founder of Rome; and the latter, after saving his country from the designs of Catiline, enabled her to contend with Athens for the palm of eloquence. 27

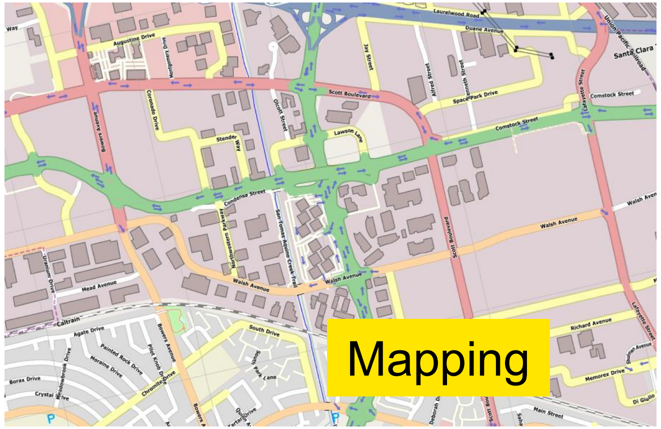
The provinces of the empire (as they have been described in the preceding chapter) were destitute of any public force, or constitutional freedom. In Etruria, in Greece, 28 and in Gaul, 29 it was the first care of the senate to dissolve those dangerous confederacies, which taught mankind that, as the Roman arms prevailed by division, they might be resisted by union. Those princes, whom the ostentation of gratitude or generosity permitted for a while to hold a precarious sceptre, were dismissed from their thrones, as soon as they had performed their appointed task of fashioning to the re rewarded with a ercised by the !_0010 But the same d to the most distant f introducing colo-

ned by history and ges of victory; and we may remark, that, about forty years after the reduction of Asia, eighty thousand Romans were massacred in one day, by the cruel orders of Mithridates. 31 These voluntary exiles were engaged, for the most part, in the occupations of commerce, agriculture, and the farm of the revenue. But after the legions were rendered permanent by the emperors

observati on a very number o son, whi though t two hur thousar

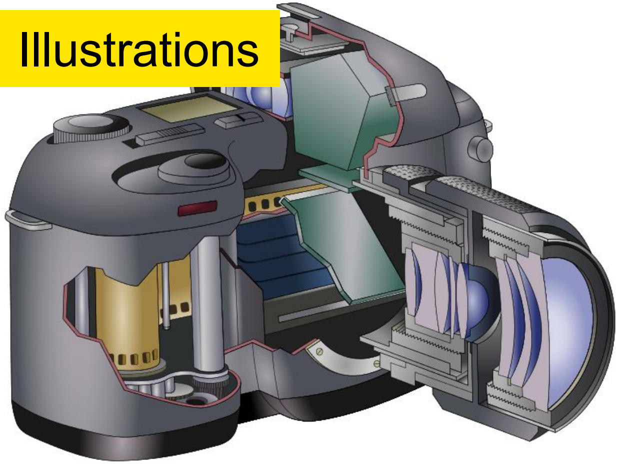
The ne fixed Empe thous milli atten Clau wer tion tha go

Text, even in with perspective



Mapping

Illustrations



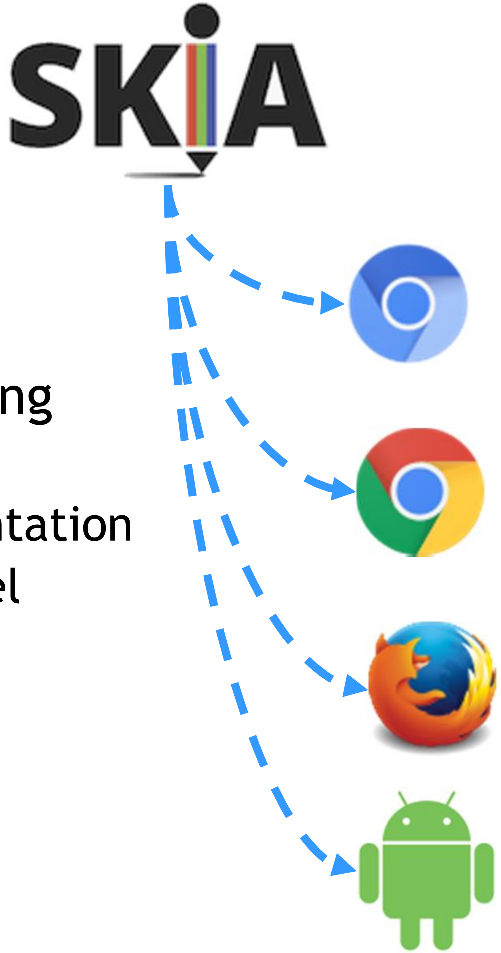
POST
ATM
SHOP
HOTEL
24
JULY

Emojis! ☺

NVIDIA OpenGL Features Integrated in Google’s Skia 2D Graphics Library

- Skia is Google’s 2D graphics library
  - Primarily for web rendering
  - Used by Chromium, Firefox, and Google’s Chrome browser

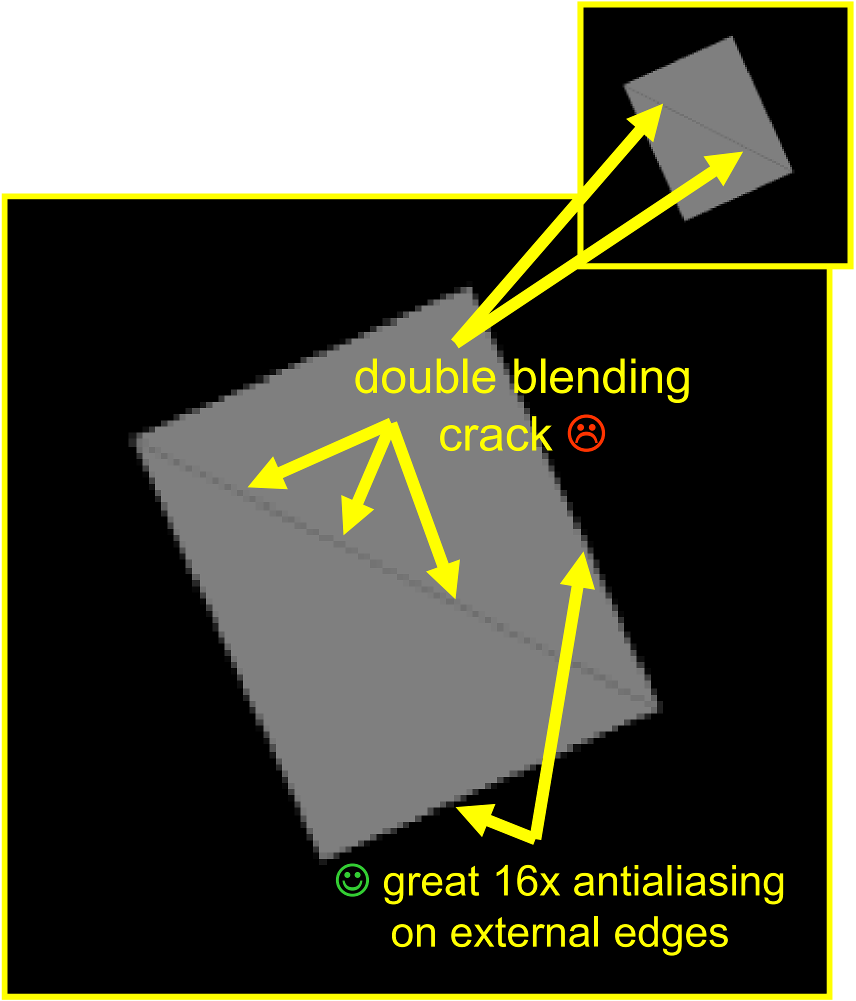
- Skia has support today for GPU-acceleration with OpenGL exploiting
  - NV_path_rendering for vector graphics filling & stroking
  - NV_framebuffer_mixed_samples for efficient framebuffer representation
  - EXT_blend_func_extended for extended Porter-Duff blending model
  - KHR_blend_equation_advanced for advanced Blend Modes



Naïve Mixed Sample Rendering Causes Artifacts

Requires Careful use of NV_framebuffer_mixed_samples

- Easy to render paths with NV_path_rendering + NV_framebuffer_mixed_samples
  - **Reason:** two-step "Stencil, then Cover" approach guarantees proper coverage is fully resolved in first "stencil" pass, then color is updated in "cover" pass
  - Just works by design
- But what if you want to render a simple convex shape like a rectangle with conventional rasterization & mixed samples?
  - Draw rectangle as two triangles
    - Into 16:1 mixed sample configuration
  - But fractional coverage modulation causes seam along internal edge!

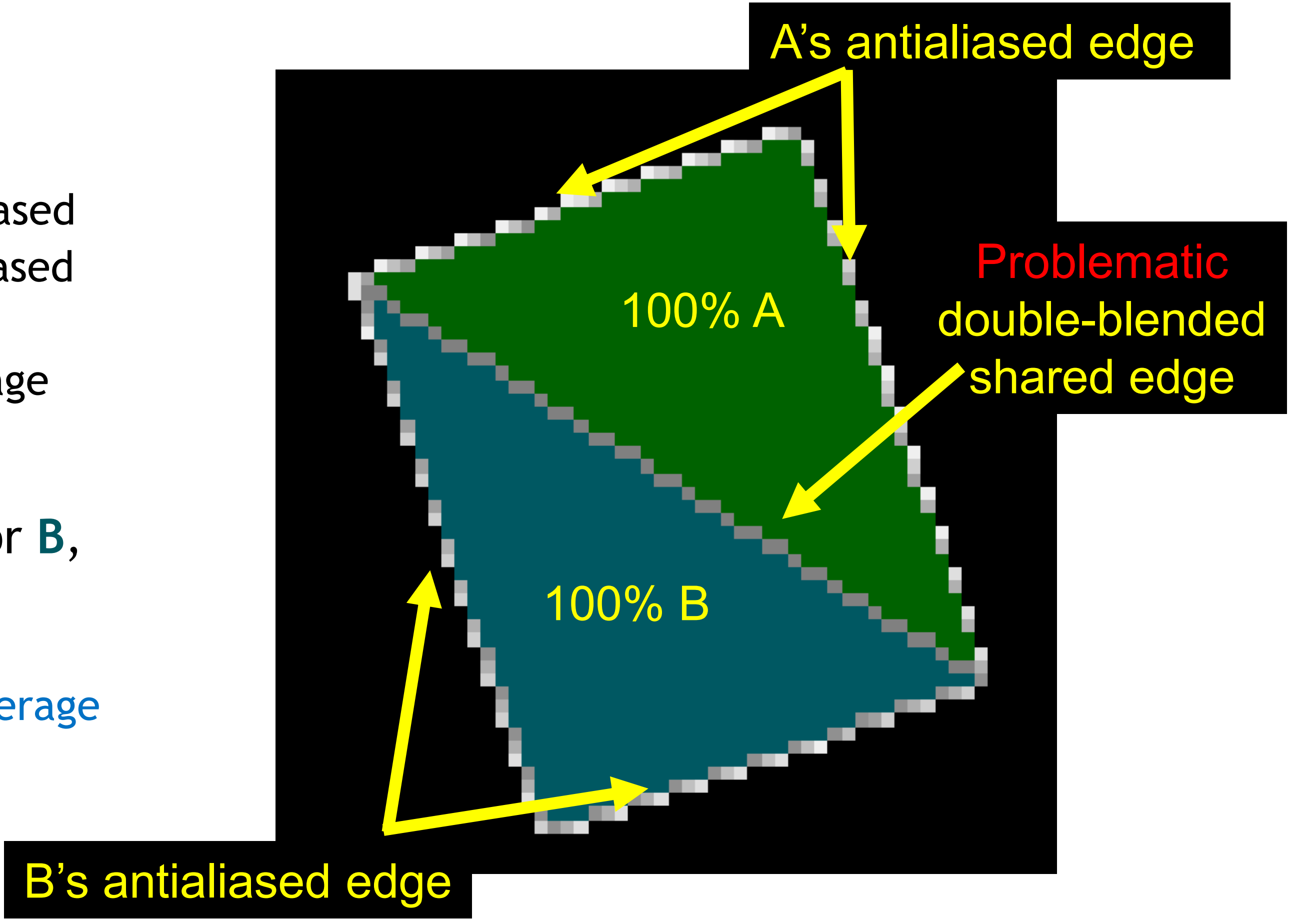


4x pixel magnification

Examine the Situation Carefully

Maxwell's **NV_sample_mask_override_coverage** Extension Helps

- Two triangles A and B
  - Where A is 100% fine
  - Where B is 100% fine
  - External edge of A is properly antialiased
  - External edge of B is properly antialiased
  - PROBLEM is shared edge
  - Both triangles claim fractional coverage along this edge
    - Causes Double Blending
- Can we "fix" rasterization so either A or B, but never both claim the shared edge?
  - YES, Maxwell GPUs can
  - Using NV_sample_mask_override_coverage extension



Solution: Triangle A Claims Coverage or B Claims, But not Both

Handle in fragment shader: by overriding the sample mask coverage

```
void main() {

  gl_FragColor = gl_Color;

















}
```

trivial fragment shader

BEFORE: Simply output interpolated color

```
#version 400 compatibility
#extension GL_NV_sample_mask_override_coverage : require
layout(override_coverage) out int gl_SampleMask[];
const int num_samples = 16;
const int all_sample_mask = 0xffff;

void main() {

  gl_FragColor = gl_Color;

  if (gl_SampleMaskIn[0] == all_sample_mask) {
    gl_SampleMask[0] = all_sample_mask;
  } else {
    int mask = 0;
    for (int i=0; i<num_samples; i++) {
      vec2 st;
      st = interpolateAtSample(gl_TexCoord[0].xy, i);
      if (all(lessThan(abs(st),vec2(1))))
        mask |= (1 << i);
    }
    int otherMask = mask & ~gl_SampleMaskIn[0];
    if (otherMask > gl_SampleMaskIn[0])
      gl_SampleMask[0] = 0;
    else
      gl_SampleMask[0] = mask;
  }
}
```

AFTER: Interpolate color + resolve overlapping coverage claims

Solution: Triangle A Claims Coverage or B Claims, But not Both

Handle in fragment shader: by overriding the sample mask coverage

```
void main() {

  gl_FragColor = gl_Color;










}
```

BEFORE: Simply output interpolated color

sample mask override coverage support

```
#version 400 compatibility
#extension GL_NV_sample_mask_override_coverage : require
layout(override_coverage) out int gl_SampleMask[];
const int num_samples = 16;
const int all_sample_mask = 0xffff;

void main() {

  gl_FragColor = gl_Color;

  if (gl_SampleMaskIn[0] == all_sample_mask) {
    gl_SampleMask[0] = all_sample_mask;
  } else {
    int mask = 0;
    for (int i=0; i<num_samples; i++) {
      vec2 st;
      st = interpolateAtSample(gl_TexCoord[0].xy, i);
      if (all(lessThan(abs(st),vec2(1))))
        mask |= (1 << i);
    }
    int otherMask = mask & ~gl_SampleMaskIn[0];
    if (otherMask > gl_SampleMaskIn[0])
      gl_SampleMask[0] = 0;
    else
      gl_SampleMask[0] = mask;
  }
}
```

early accept optimization

additional re-rasterization epilogue

AFTER: Interpolate color + resolve overlapping coverage claims

NV_sample_mask_override_coverage

What does it allow?

- **BEFORE:** Fragment shaders can access sample mask for multisample rasterization
  - Indicates which individual coverage samples with a pixel are covered by the fragment
  - Fragment shader can also “clear” bits in the sample mask to discard samples
  - But in standard OpenGL, no way to “set” bits to augment coverage
    - Fragment’s output sample mask is always bitwise AND’ed with original sample mask

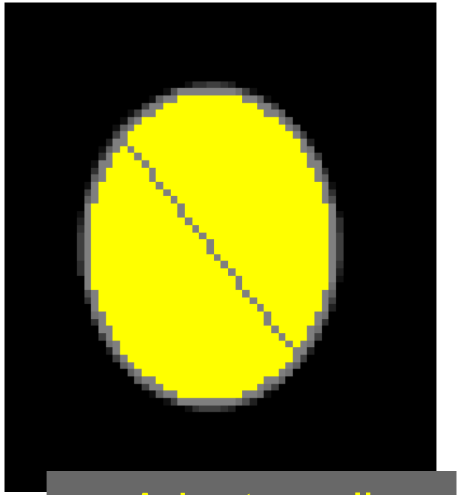
- **NOW:** Maxwell’s NV_sample_mask_override_coverage allows overriding coverage!
  - The fragment shader can completely rewrite the sample mask
  - Clearing bits still discards coverage
  - BUT setting bits not previously set <u>augments</u> coverage

- Powerful capability enables programmable rasterization algorithms
  - Like example in previous slide to fix double blending artifacts

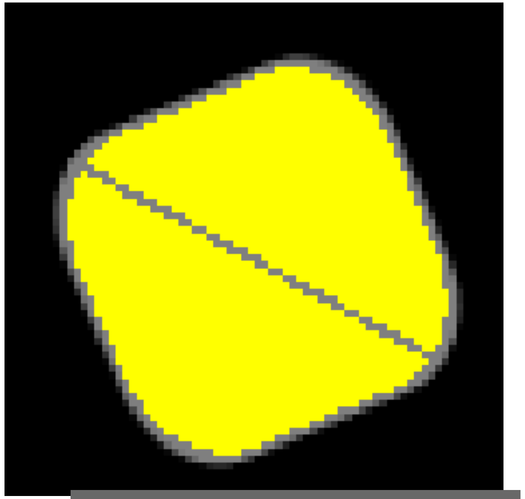
Other Sample Mask Coverage Override Uses

- Handles per-sample stencil test for high-quality sub-pixel clipping
- These techniques integrated today into Skia

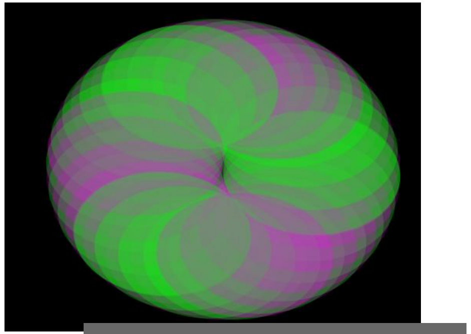
Works for general quadrilaterals, even in drawn in perspective



Adapts well to drawing circles and ellipses



And even rounded rectangles



Example: 16x quality blended ellipses

Lacked time to talk about these extensions

Maxwell OpenGL Extensions

New Graphics Features of NVIDIA's Maxwell GPU Architecture

- Voxelization, Global Illumination, and Virtual Reality
  - NV_viewport_array2
  - NV_viewport_swizzle
  - AMD_vertex_shader_viewport_index
  - AMD_vertex_shader_layer
- Vector Graphics extensions
  - NV_framebuffer_mixed_samples
  - EXT_raster_multisample
  - NV_path_rendering_shared_edge

- Advanced Rasterization
  - NV_conservative_raster
  - NV_conservative_raster_dilate
  - NV_sample_mask_override_coverage
  - NV_sample_locations, now ARB_sample_locations
  - NV_fill_rectangle
- Shader Improvements
  - NV_geometry_shader_passthrough
  - NV_shader_atomic_fp16_vector
  - NV_fragment_shader_interlock, now ARB_fragment_shader_interlock
  - EXT_post_depth_coverage, now ARB_post_depth_coverage

Requires GeForce 950, Quadro M series, Tegra X1, or better

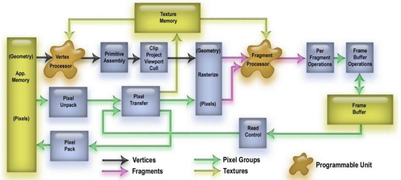
2015: In Review

OpenGL in 2015 ratified 13 new standard extensions

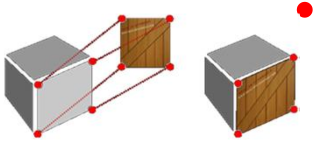
- **Shader functionality**

  - ARB_ES3_2_compatibility
    - *ES 3.2 shading language support*
  - ARB_parallel_shader_compile
  - ARB_gpu_shader_int64
  - ARB_shader_atomic_counter_ops
  - ARB_shader_clock
  - ARB_shader_ballot

- **Graphics pipeline operation**



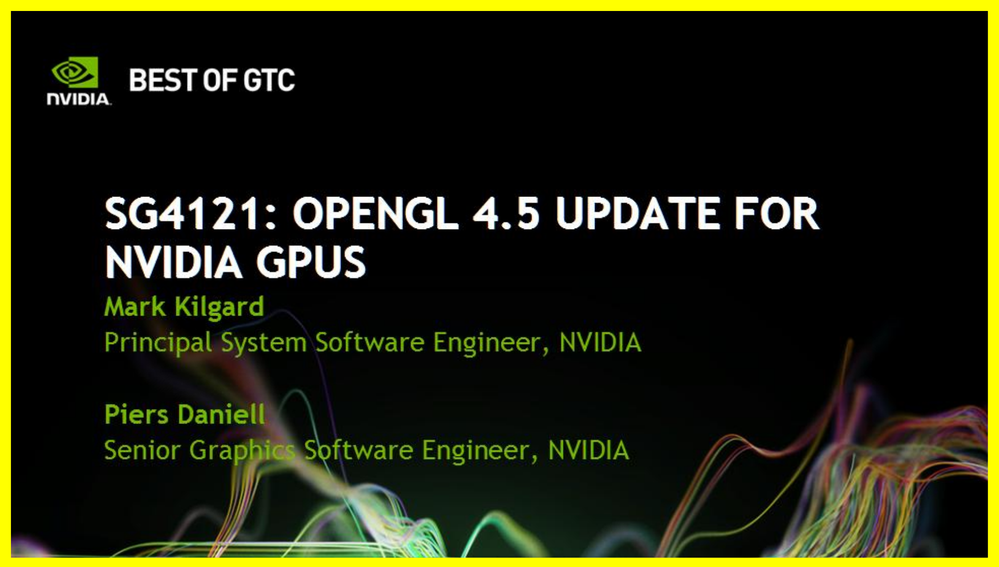
  - ARB_fragment_shader_interlock
  - ARB_sample_locations
  - ARB_post_depth_coverage
  - ARB_ES3_2_compatibility
    - *Tessellation bounding box*
    - *Multisample line width*
  - ARB_shader_viewport_layer_array



- **Texture mapping functionality**
  - ARB_texture_filter_minmax
  - ARB_sparse_texture2
  - ARB_sparse_texture_clamp

Need a Full Refresher on 2014 and 2015 OpenGL?

- Honestly, lots of functionality in 2014 & 2015 if you’ve not followed carefully



BEST OF GTC

SG4121: OPENGL 4.5 UPDATE FOR NVIDIA GPUS

Mark Kilgard
Principal System Software Engineer, NVIDIA

Piers Daniell
Senior Graphics Software Engineer, NVIDIA

OpenGL for 2015

Mark Kilgard, Principal System Software Engineer

Available @ http://www.slideshare.net/Mark_Kilgard

Pascal GPU OpenGL Extensions

New for 2016

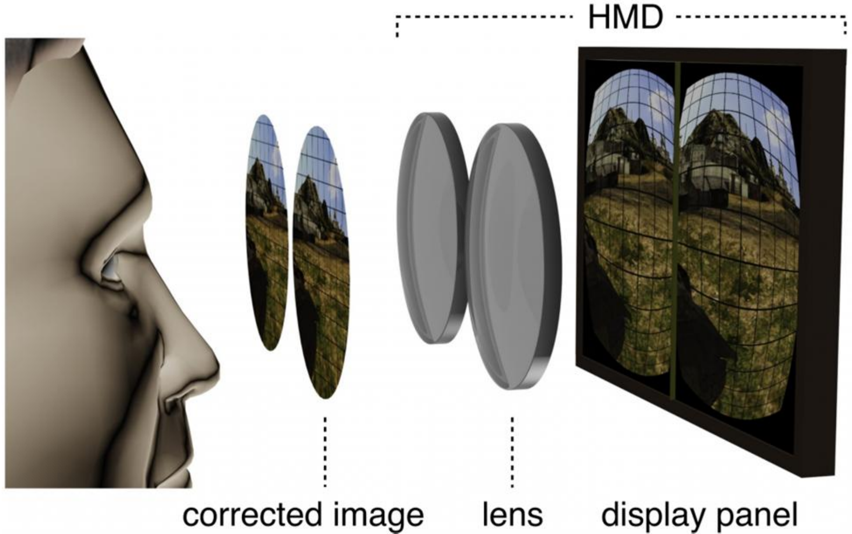
- Pascal has 5 new OpenGL extensions
  - Major goal: improving Virtual Reality support
- Several extensions used in combination
  - NV_stereo_view_rendering
    - efficiently render left & right eye views in single rendering pass
  - NV_viewport_array2 + NV_geometry_shader_passthrough—discussed already
  - NV_clip_space_w_scaling
    - extends viewport array state with per-viewport re-projection
  - EXT_window_rectangles
    - fast inclusive/exclusive rectangle testing during rasterization
    - Multi-vendor extension supported on all modern NVIDIA GPUs
- High-end Virtual Reality with two GPUs
  - New explicit NV_gpu_multicast extension
    - Render left & right eyes with distinct GPUs

VR SLI

Basic question

Why should the Virtual Reality (VR) image shown in a Head Mounted Display (HMD) *feel real*?

Ignoring head tracking and the realism of the image itself… just focused on the image generation



Why HMD's Image ≈ Perception of Reality

HMD image ≈ lens image

⇩

≈ lens(screen)

⇩

≈ lens(lens-1(rendered image))

⇩

≈ rendered image

⇩

≈ pin hole image

⇩

≈ eye view

⇩

≈ perception of reality

by optics
lens image = lens(screen)

by warping
screen ≈ lens-1(rendered image)

by composition
image ≈ lens(lens-1(image))

by rendering model
rendered image ≈ pin hole image

by anatomy
pin hole image ≈ eye view

by psychology
eye view ≈ perception of reality

Portion of transformation involving GPU rendering & resampling

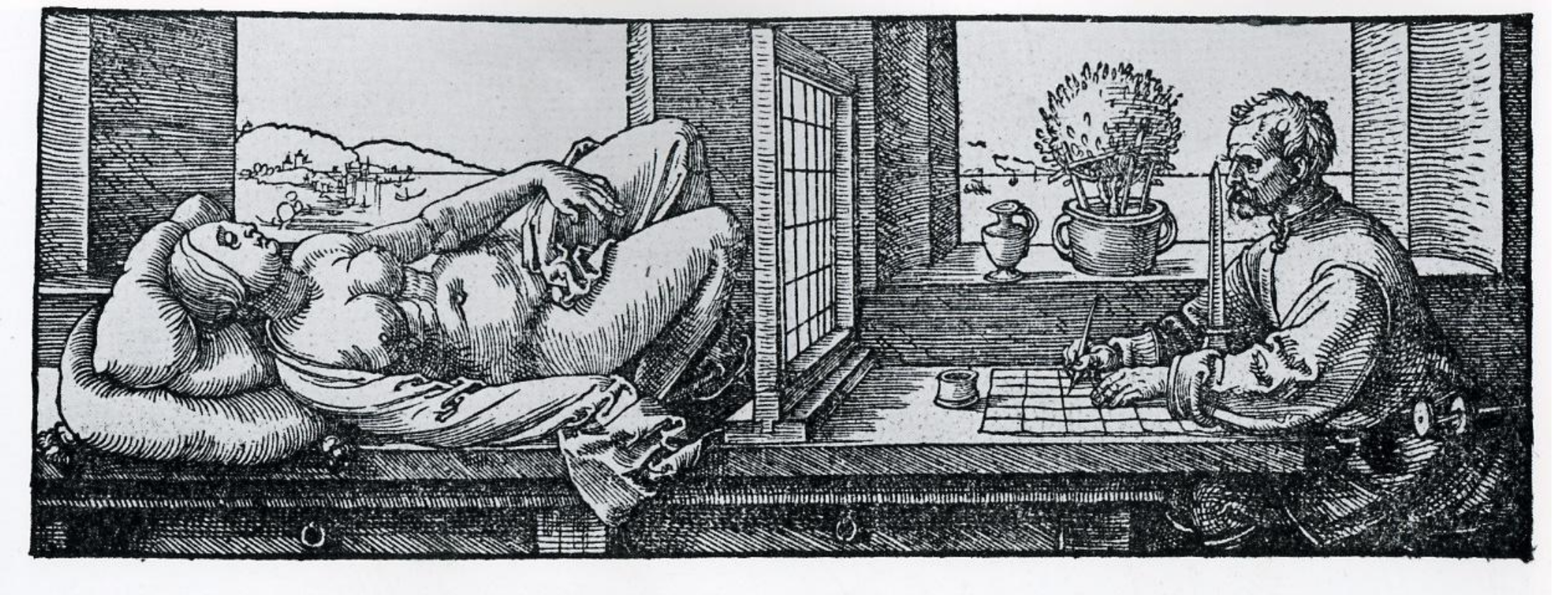
Twin goals

1. *Minimize HMD resampling error*
2. *Increase rendering efficiency*

Goal of Head Mounted Display (HMD) Rendering

- **Goal:** perceived HMD image ≈ visual perception of reality
  - Each image pair on HMD screen, as seen through its HMD lens, should be perceived as images of the real world
- Assume pin hole camera image ≈ real world
  - Traditional computer graphics assumes this
    - Perspective 3D rasterization idealizes a pin hole camera
  - Human eye ball also approximately a pin hole camera
- perceived HMD image = lens(screen image)
  - Function lens() warps image as optics of HMD lens does
- screen image = lens^{-1}(pin hole camera image)
  - Function lens^{-1}() is inverse of the lens image warp
- perceived image ≈ lens(lens^{-1}(pin hole camera image))
- pin hole camera image ≈ eye view

Pin Hole Camera Ideal



Albrecht Dürer: Artist Drawing with Perspective Device

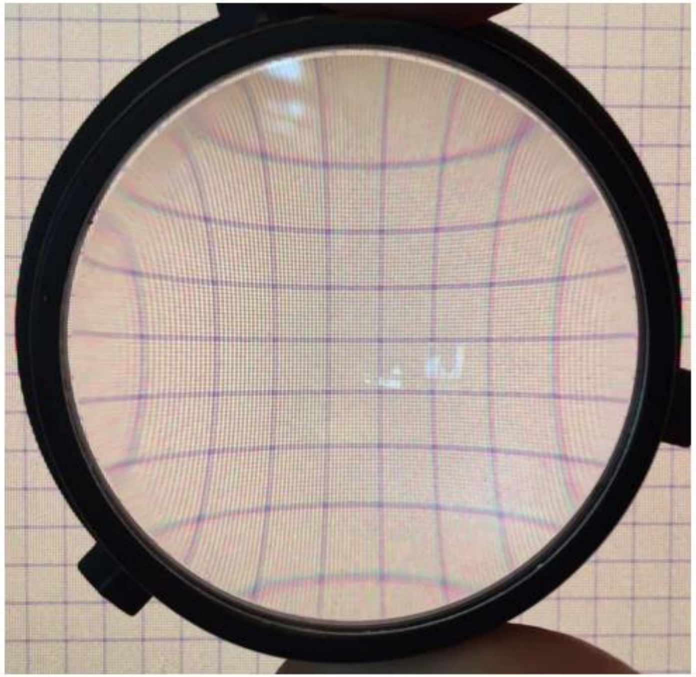
Normal computer graphics generally good at rendering "pin hole" camera images

And people are good at interpreting such images as 3D scenes

But HMDs have a non-linear image warping due to lens distortion

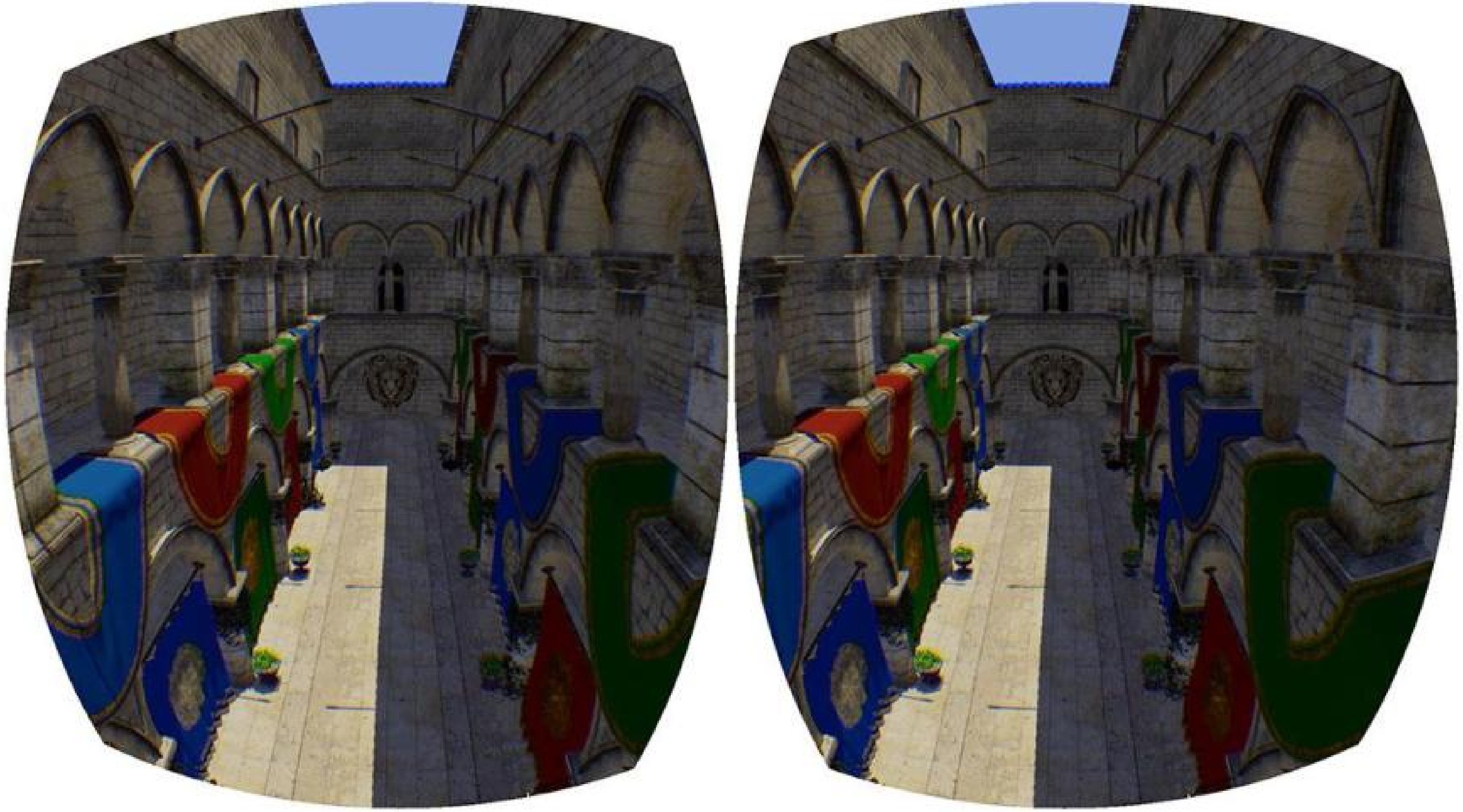
Lens Distortion in HMD

- Head-mounted Display (HMD) magnifies its screen with a lens
- Why is a lens needed?
  - To feel immersive
    - Immersion necessitates a wide field-of-view
  - So HMD lens “widens” the HMD screen’s otherwise far too narrow field-of-view
- Assume a radial symmetric magnify
  - Could be a fancier lens & optics
  - BUT consumer lens should be inexpensive & lightweight



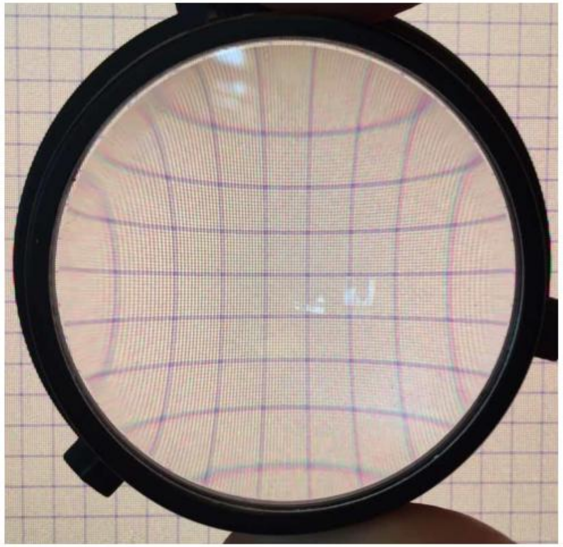
Graph paper viewed & magnified through HMD lens

Example HMD Post-rendering Warp

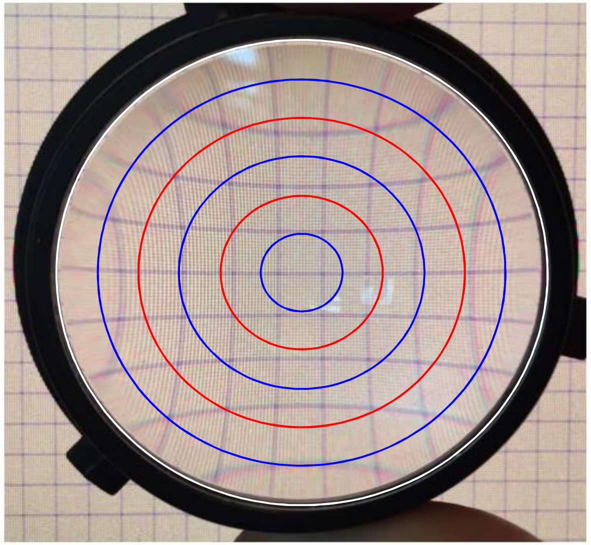


Lens Performs a Radial Symmetric Warp

Adding circles to image shows distortion increases as the radius increases



Original Image



Overlaid with circles

Pin-hole Camera Image Assumptions

- Assume a conventionally rendered perspective image
  - In other words a pin-hole camera image
- *r* is the distance of a pixel (*x*,*y*) relative to the center of the image at (0,0) so

$$r = \sqrt{x^2 + y^2}$$

- Theta is the angle of the pixel relative to the origin

$$x = r\cos\theta$$

$$y = r\sin\theta$$

- Assume pin hole camera image has maximum radius of 1
  - So the X & Y extent of the images is [-1..1]

Radius Remapping for an HMD Magnifying Lens

- A lens in an HMD magnifies the image
  - What is magnification really?
  - Magnifying takes a pixel at a given radius and “moves it out” to a larger radius in the magnified image
- In the HMD len’s image, each pin-hole camera pixel radius *r* is mapped to alternate radius $r_{lensImage}$

Essentially a Taylor series approximating actual optics of lens

$$r_{lensImage} = (1 + k_1 r^2 + k_2 r^4 + \ldots)\, r_{display\mathrm{Im}age}$$

- This maps each pixel (*x*,*y*) in the pin-hole camera image to an alternate location ($x_{lensImage}$,$y_{lensImage}$)
  - Without changing theta

$$r_{displayImage} = \frac{r_{lensImage}}{1 + k_1 r^2 + k_2 r^4 + \ldots}$$

Lens Function Coefficients for Google Cardboard

Lens coefficients k_1 & k_2 are values that can be measured
Additional coefficients (k_3, etc.) are negligible

Coefficients for typical lens in Google Cardboard

$k_1 = 0.22$
$k_2 = 0.26$

Big question

Can we render so the amount of resampling necessary to invert a particular lens's distortion is minimized?

Radius Remapping for Lens Matched Shading (LMS)

- Assume a conventionally rendered perspective image
    - In other words a pin-hole camera image
- *r* is the distance of a pixel (x,y) relative to the center of the image at (0,0) so

$$r = \sqrt{x^2 + y^2}$$

- Theta is the angle of the pixel relative to the origin

$$x = r\cos\theta$$

$$y = r\sin\theta$$

OLD: Conventional “pin hold” camera rendering

- Lens Matched Shading provides an alternate radius r_{LMS} for the same pixel (x_{LMS},y_{LMS})

$$r_{LMS} = \frac{r}{1 + p\,r\left|\cos\theta\right| + p\,r\left|\sin\theta\right|}$$

- This maps each pixel (*x*,*y*) to an alternate location
    - Without changing theta

$$x_{LMS} = r_{LMS}\cos\theta$$

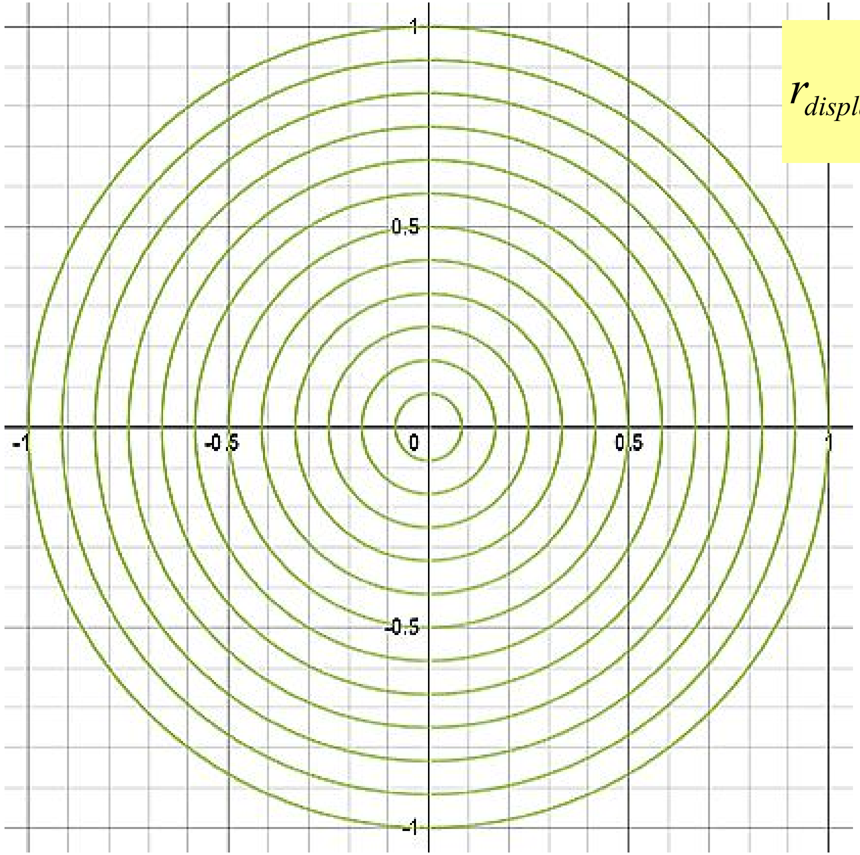
$$y_{LMS} = r_{LMS}\sin\theta$$

NEW: Lens Matched Shading rendering

HMD’s Inverse Lens Warp

Concentric circles in pin hole camera view gets “squished” by inverse lens transform

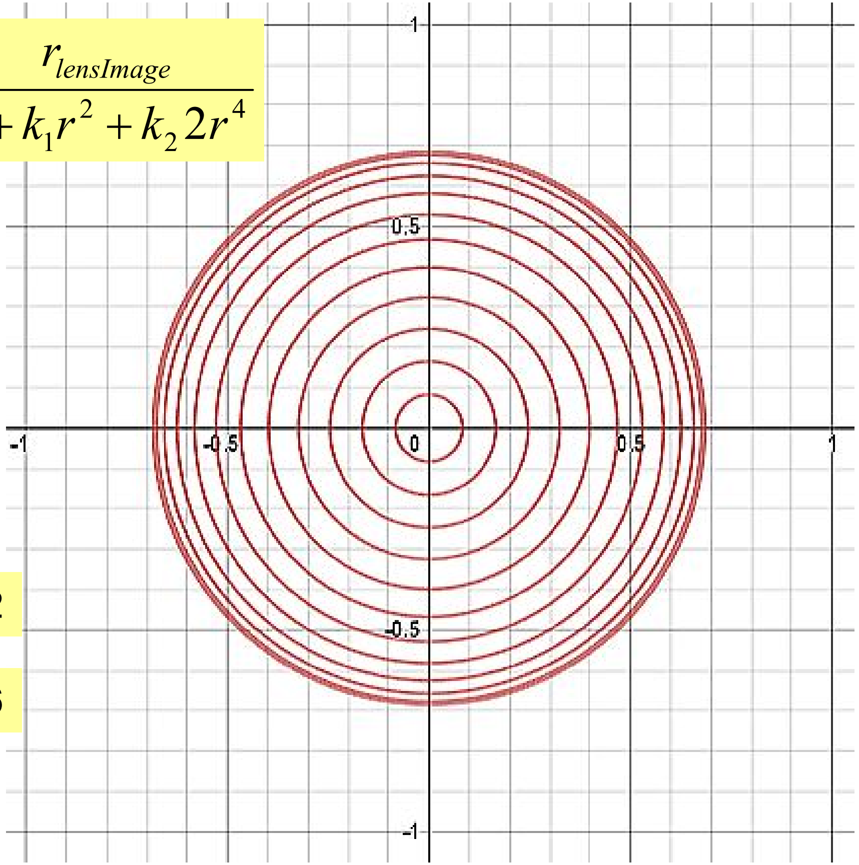
$$r_{displayImage} = \frac{r_{lensImage}}{1 + k_1 r^2 + k_2 2r^4}$$



pin hole camera view
(conventionally rendered image)

k_1 = 0.22

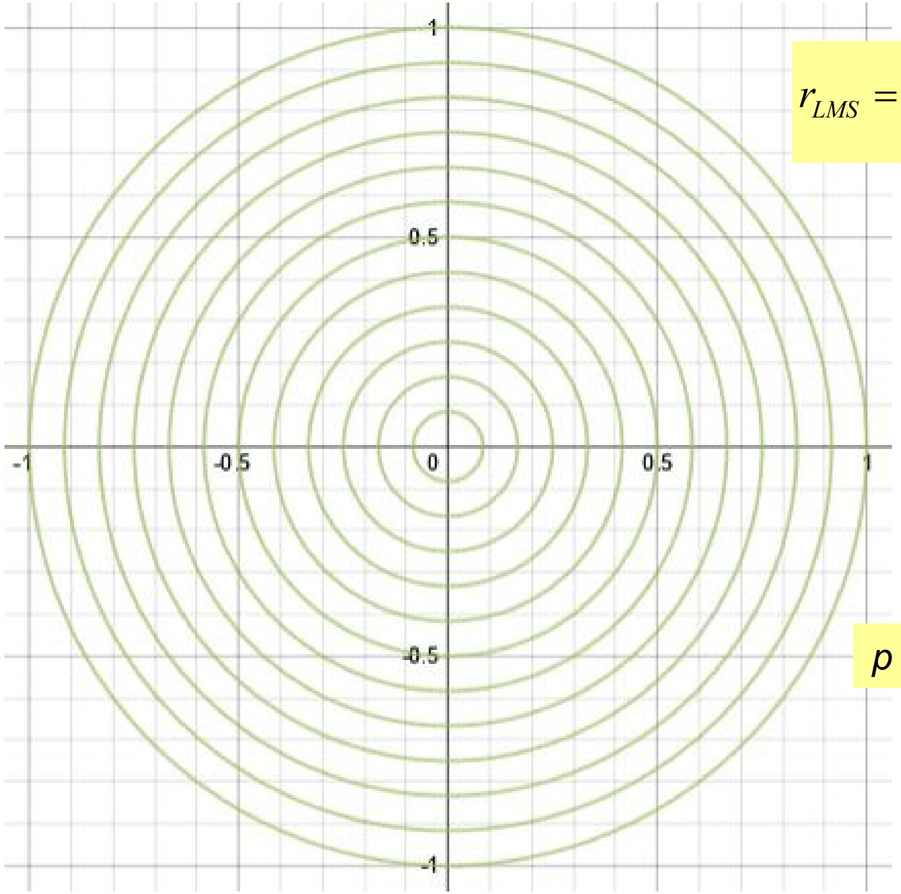
k_2 = 0.26



inverse lens warp view
(HMD screen)

Lens Matched Shading

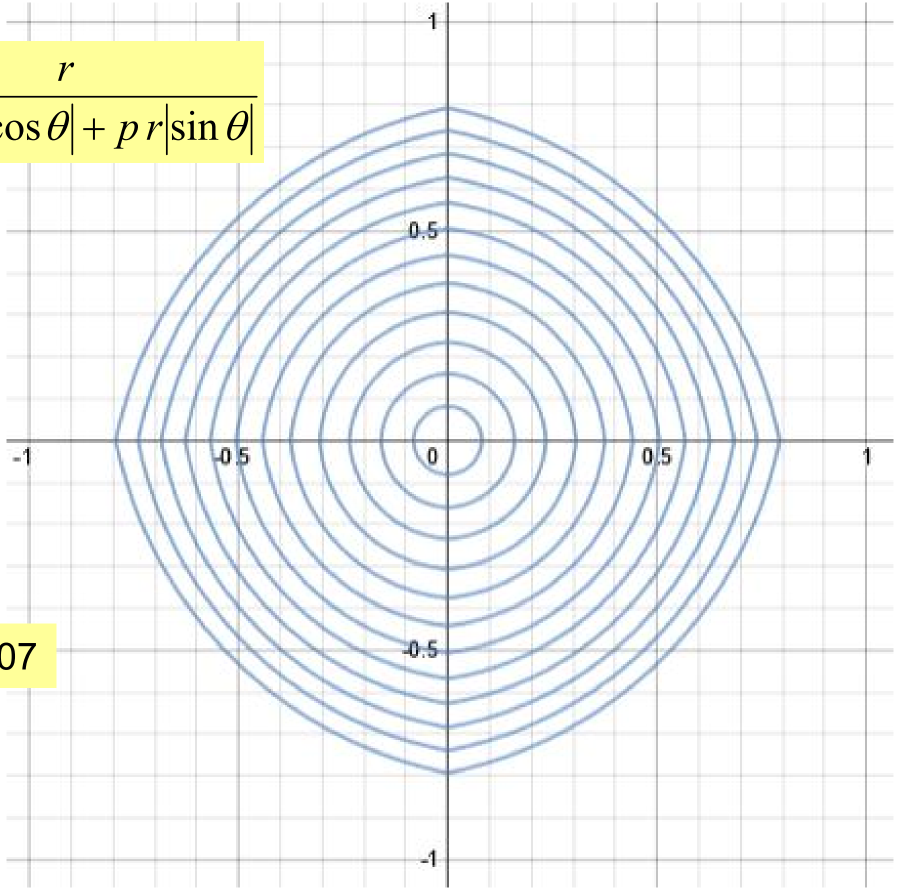
Concentric circles in pin hole camera view gets “projected” towards origin



pin hole camera view

$$r_{LMS} = \frac{r}{1 + p\,r\left|\cos\theta\right| + p\,r\left|\sin\theta\right|}$$

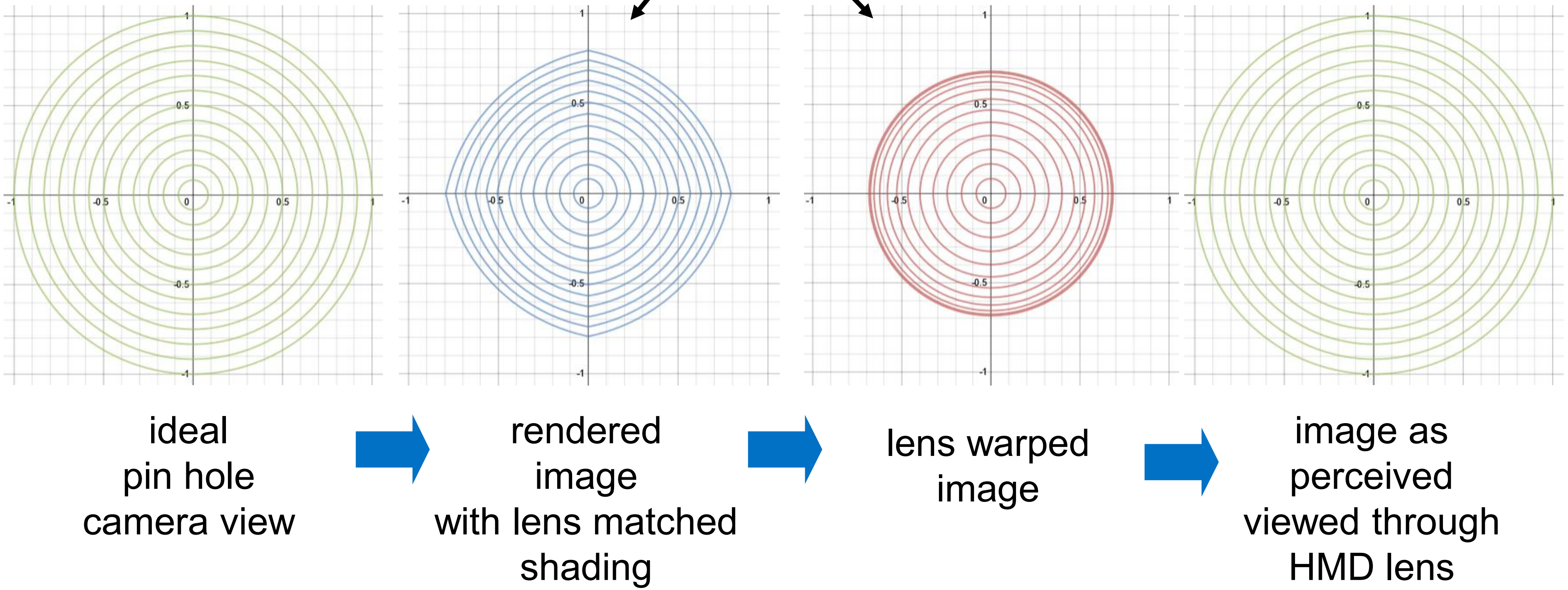
p = 0.26007



Lens Matched Shading
(rendered framebuffer image)

Complete Process of Lens Matched Shading

while different, these two images are "well matched" so warp between them minimizes pixel movement and resampling



ideal pin hole camera view

rendered image with lens matched shading

lens warped image

image as perceived viewed through HMD lens

What is Optimal Value for *p*?

A reasonable measure of optimality is root mean square error of difference between LMS and inverse lens warp radii over entire lens

So what p minimizes this integral for a particular lens's coefficients

$$\int_0^{2\pi}\int_0^1 \left(\frac{r}{1+k_1 r^2 + k_2 2r^4} - \frac{r}{1+p\,r\left|\cos\theta\right| + p\,r\left|\sin\theta\right|} \right)^2 r\,dr\,d\theta$$

When $k_1 = 0.22$ & $k_2 = 0.26$, optimal $p \approx 0.26007$

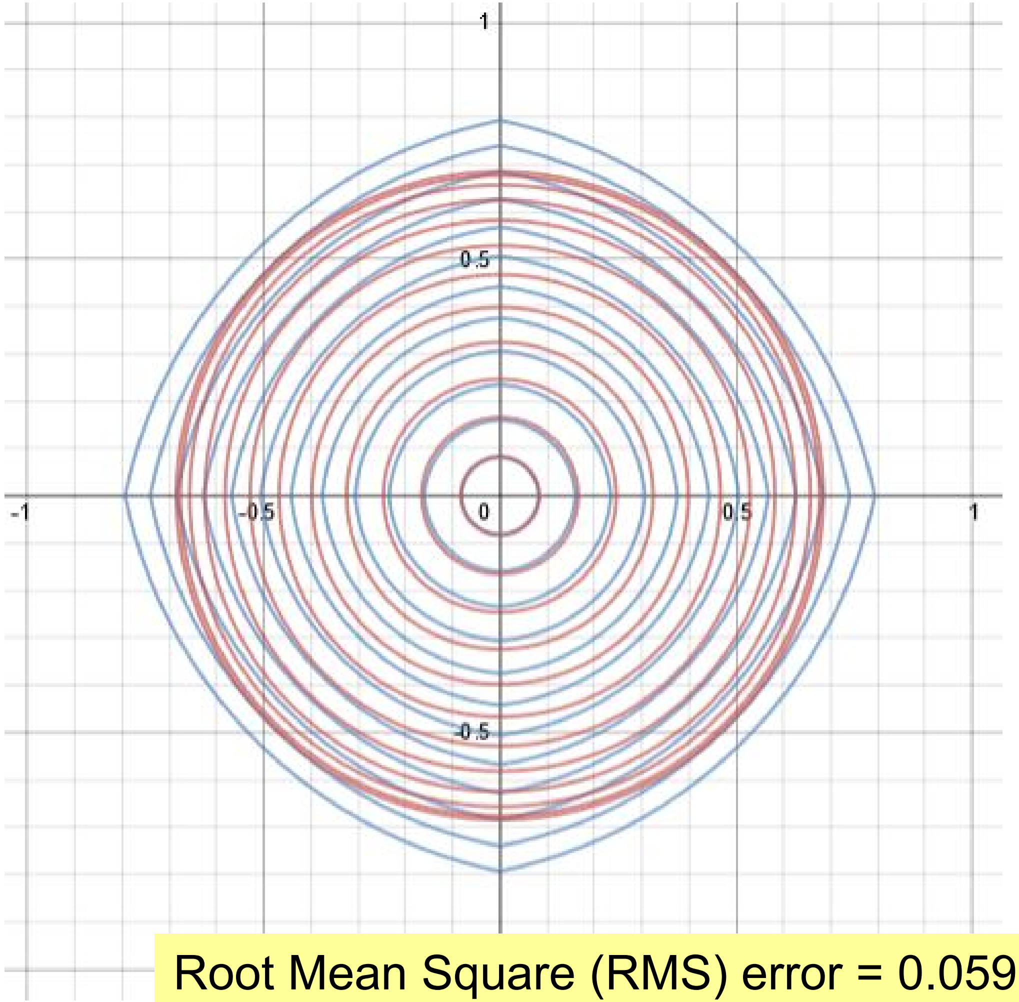
** Analysis assumes a Google Cardboard-type device; Oculus has asymmetric visible screen region*

Matched Overlap of Lens Matched Shading and Lens Warped Image

k_1 = 0.22

k_2 = 0.26

p = 0.26007



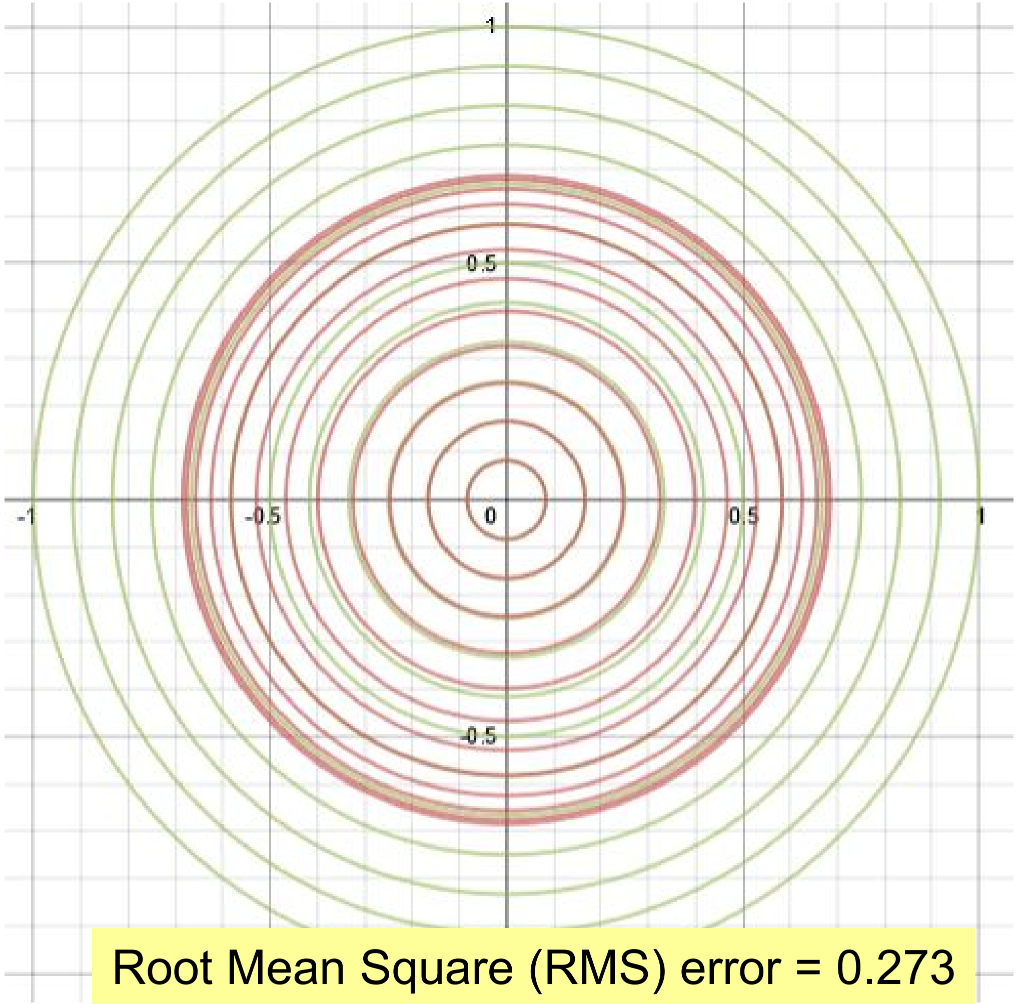
Root Mean Square (RMS) error = 0.0598

Much Worse Overlap of Conventional Projection and Lens Warped Image

$k_1 = 0.22$

$k_2 = 0.26$

$p = 0$



Root Mean Square (RMS) error = 0.273

Advantages of Lens Matched Shading

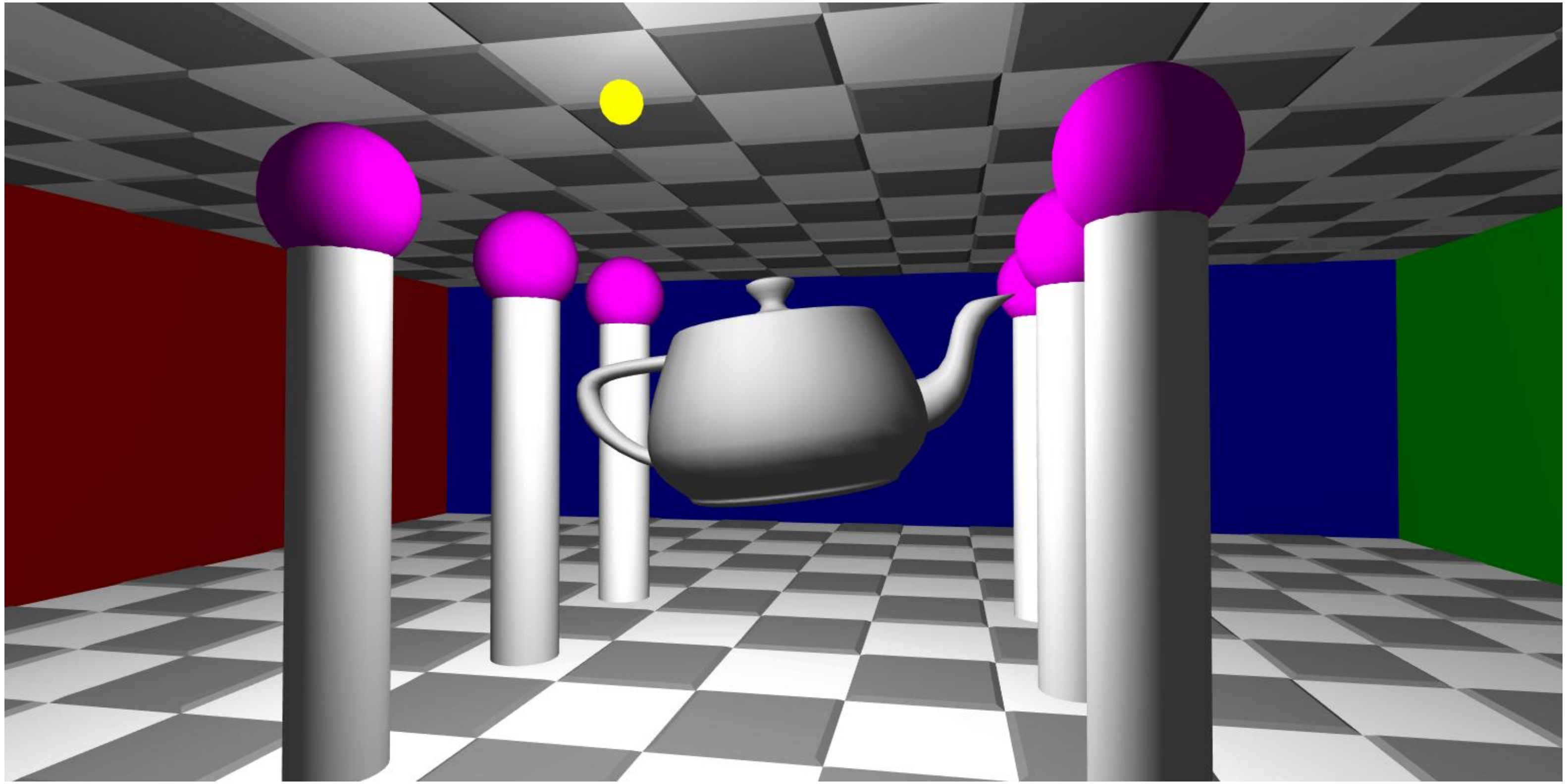
- What is rendered by GPU is closer (*less error*) to what the HMD needs to display than conventional “pin hole” camera rendering

- Means less resampling error
  - There’s still a non-linear re-warping necessary
  - However the “pixel movement” for the warp is greatly reduced

- Another advantage: fewer pixels need be rendered for same wide field of view

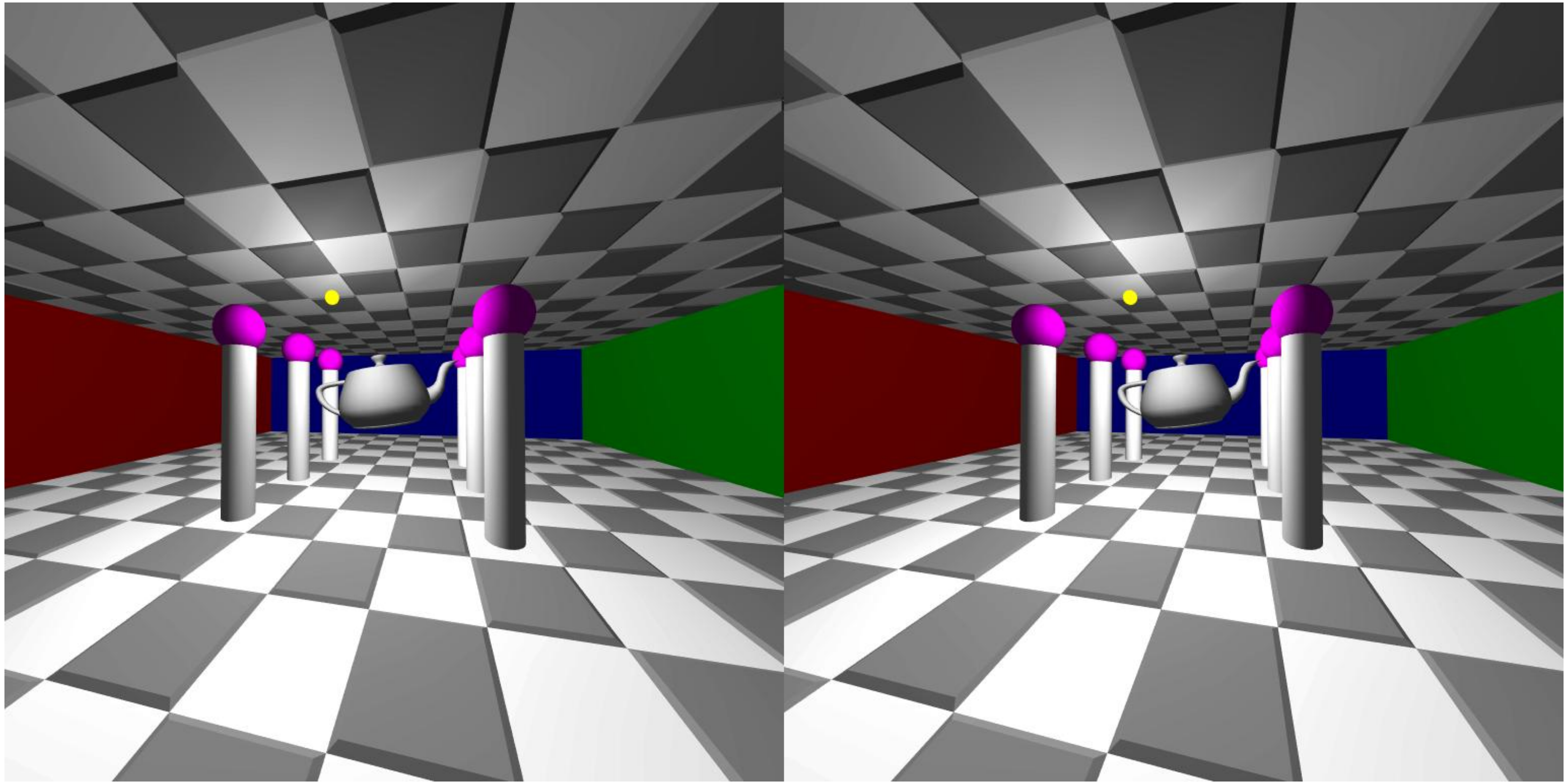
- Also want application to render left & right views with LMS in a single efficient rendering pass

Single-eye Scene



Simple 3D scene

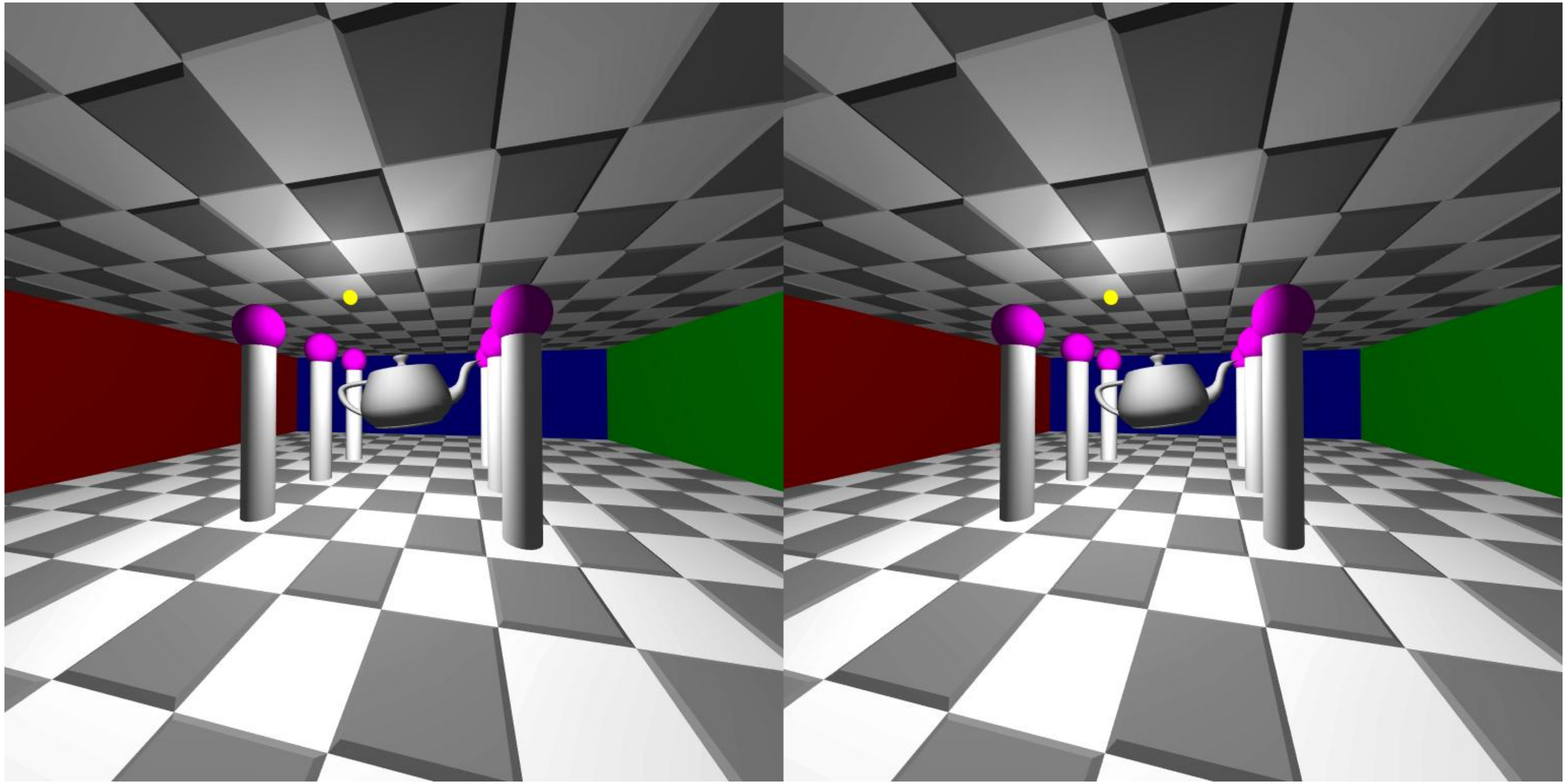
Stereo Views of Same Scene



Left and **Right** eye view of same simple scene

Two scenes are slightly different if compared

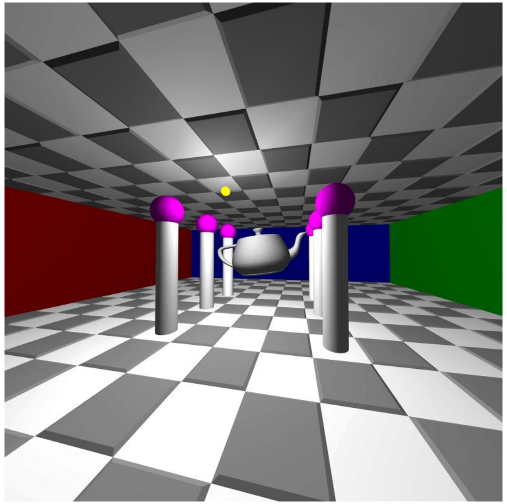
Swapped Stereo Views



Right and **Left** (swapped) eye view of same simple scene

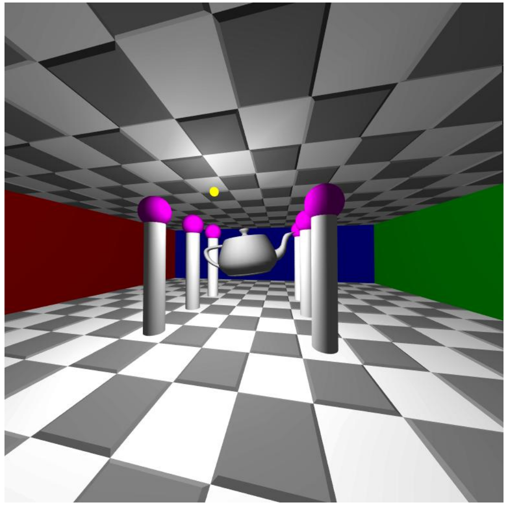
Two scenes are slightly different if compared

Image Difference of Two Views



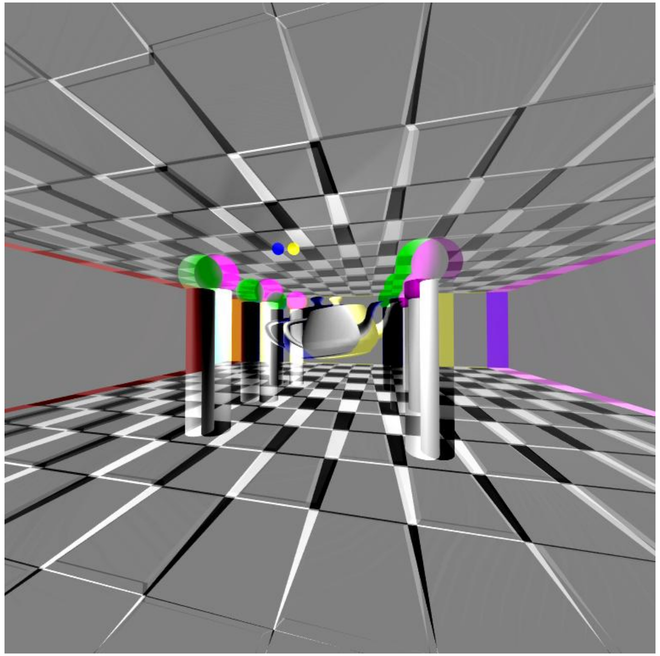
Left eye view

–



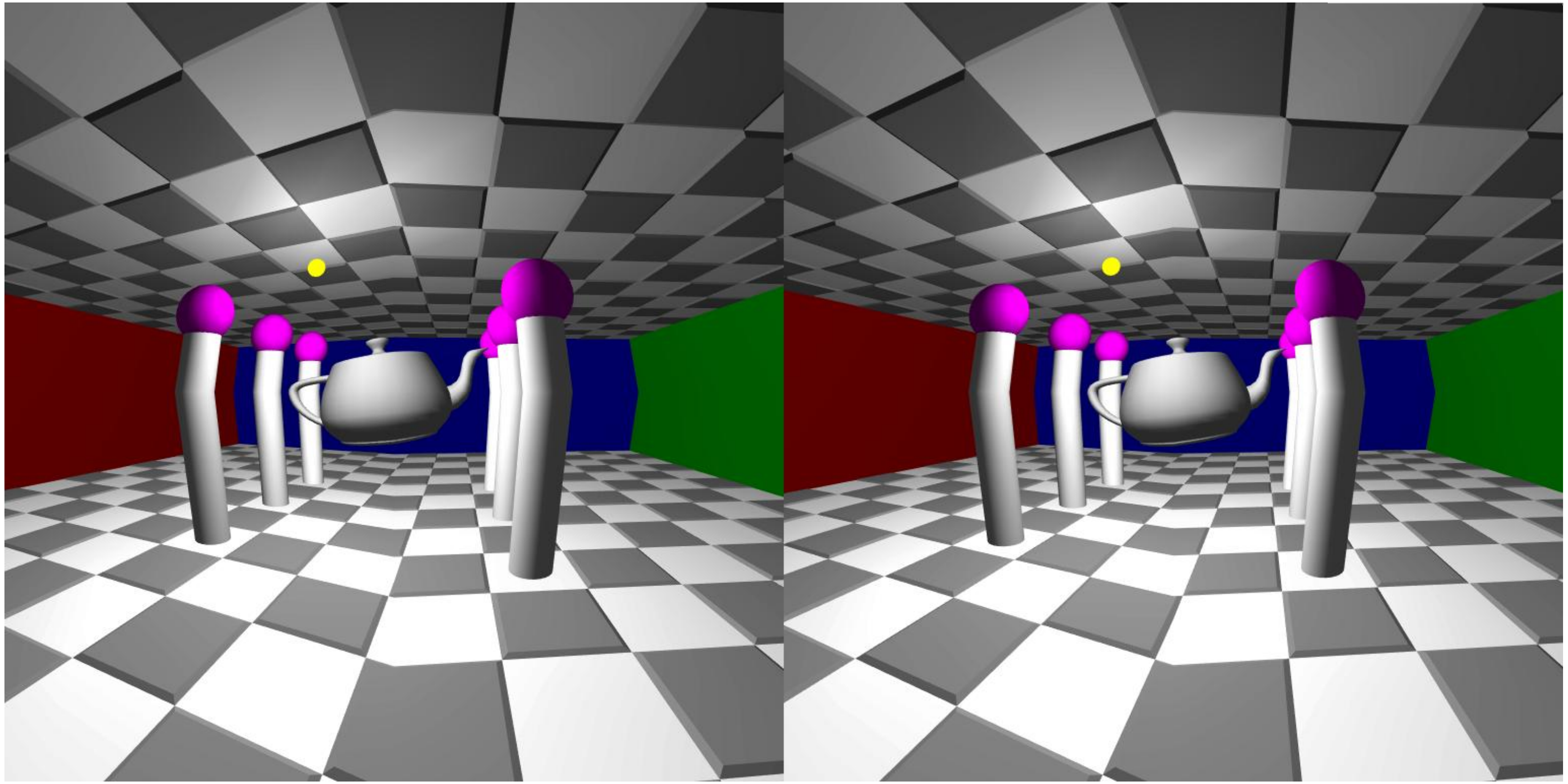
Right eye view

+ 0.5 =



Clamped difference image

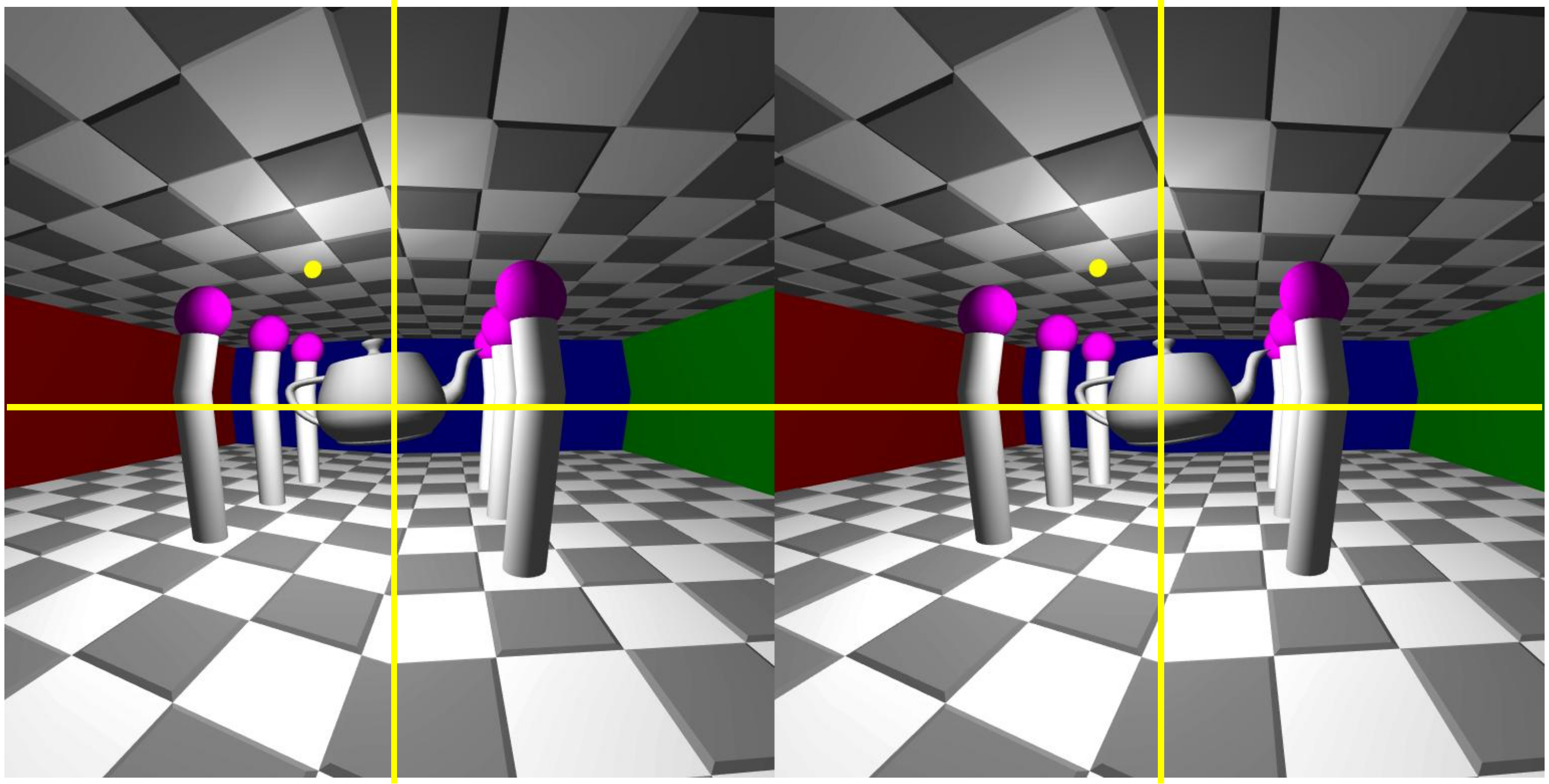
Lens Matched Shading



Same left & right eye view but rendered with w scaling

Lens Matched Shading Quadrants

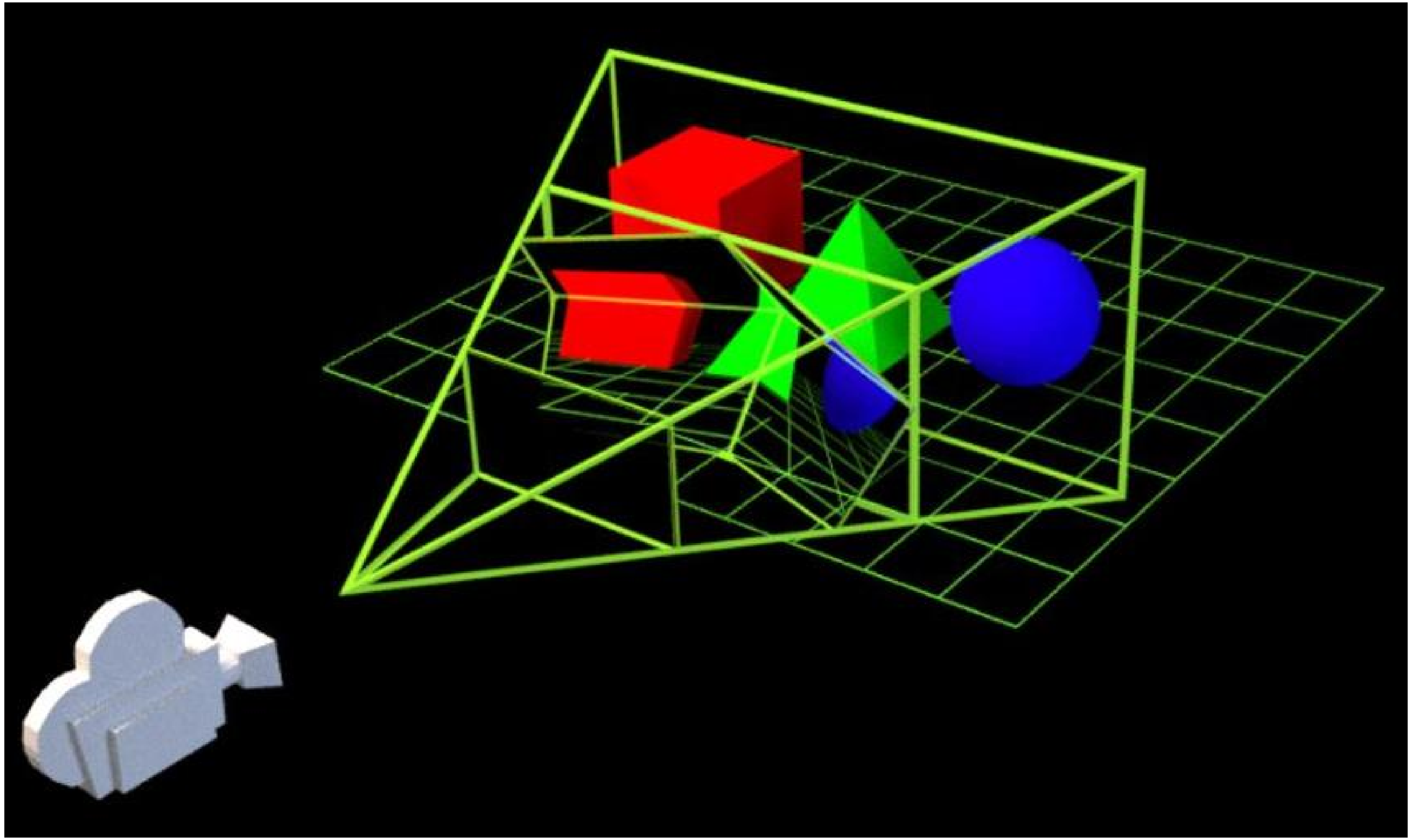
LENS MATCHED SHADING



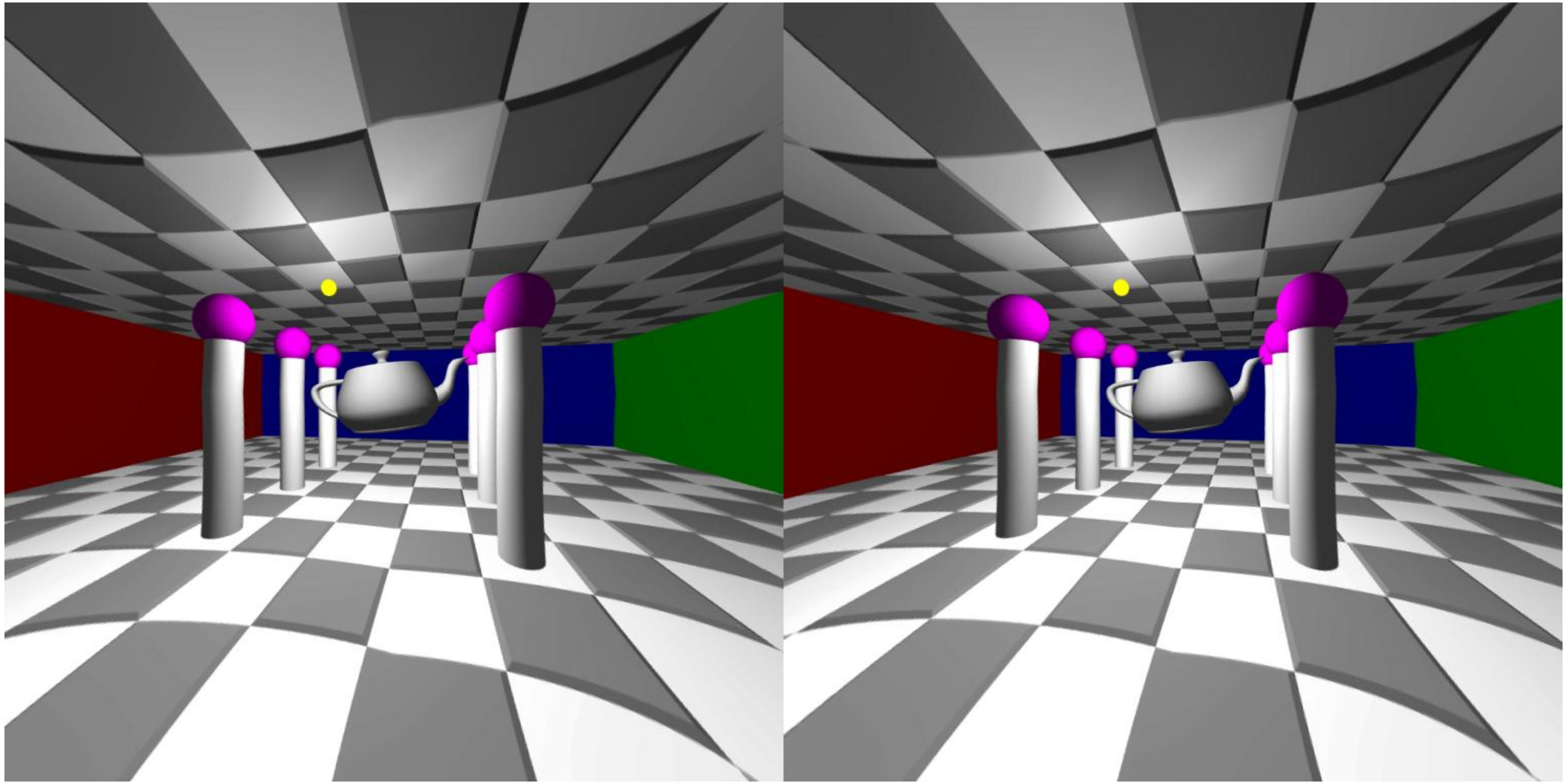
Same left & right eye view but rendered with w scaling

Each quadrant gets different projection to “tilt to center”

Visualization of Lens Matched Shading Rendering

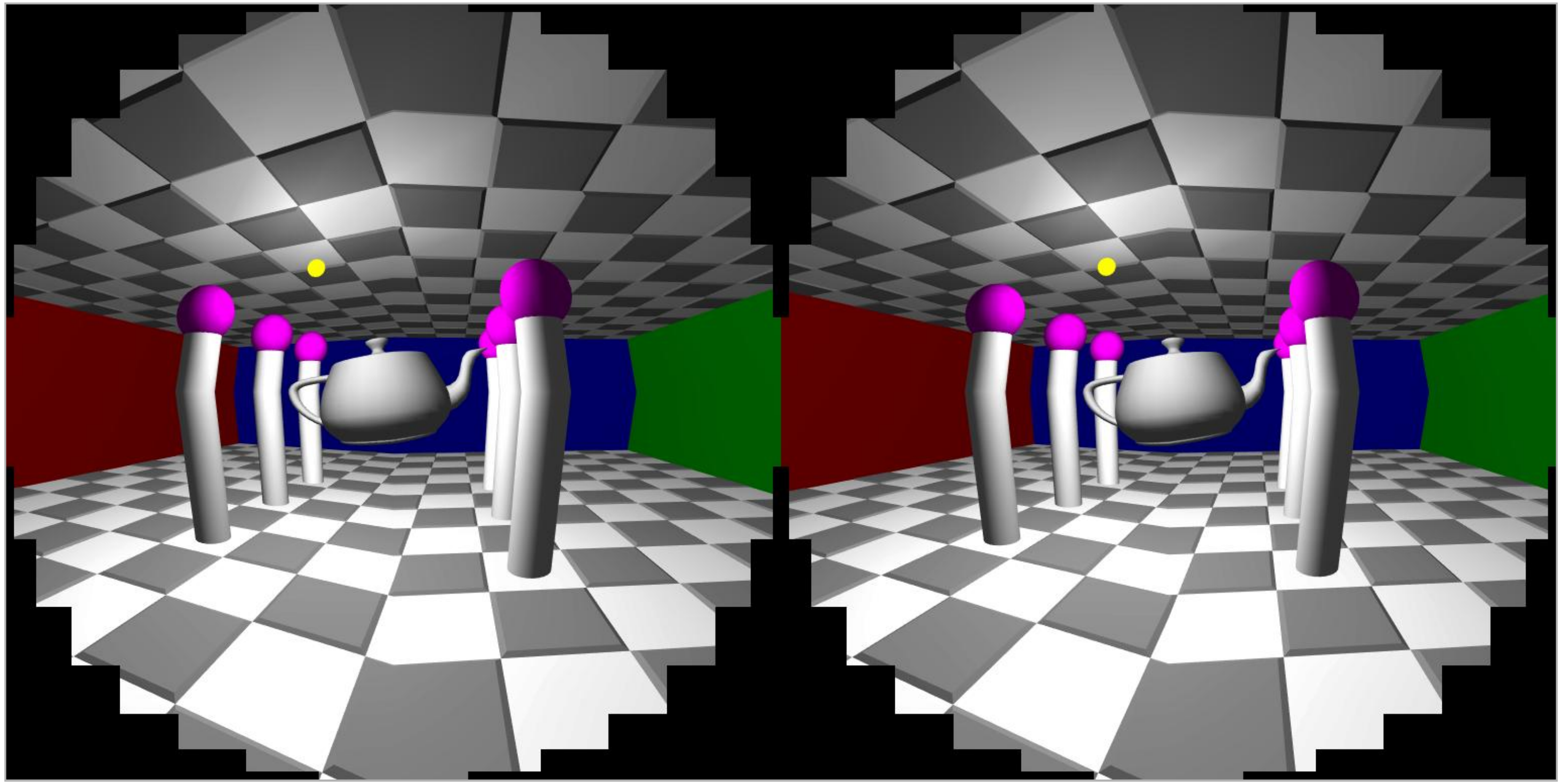


Warped Lens Matched Shaped



Warped version of lens shading to match HMD lens

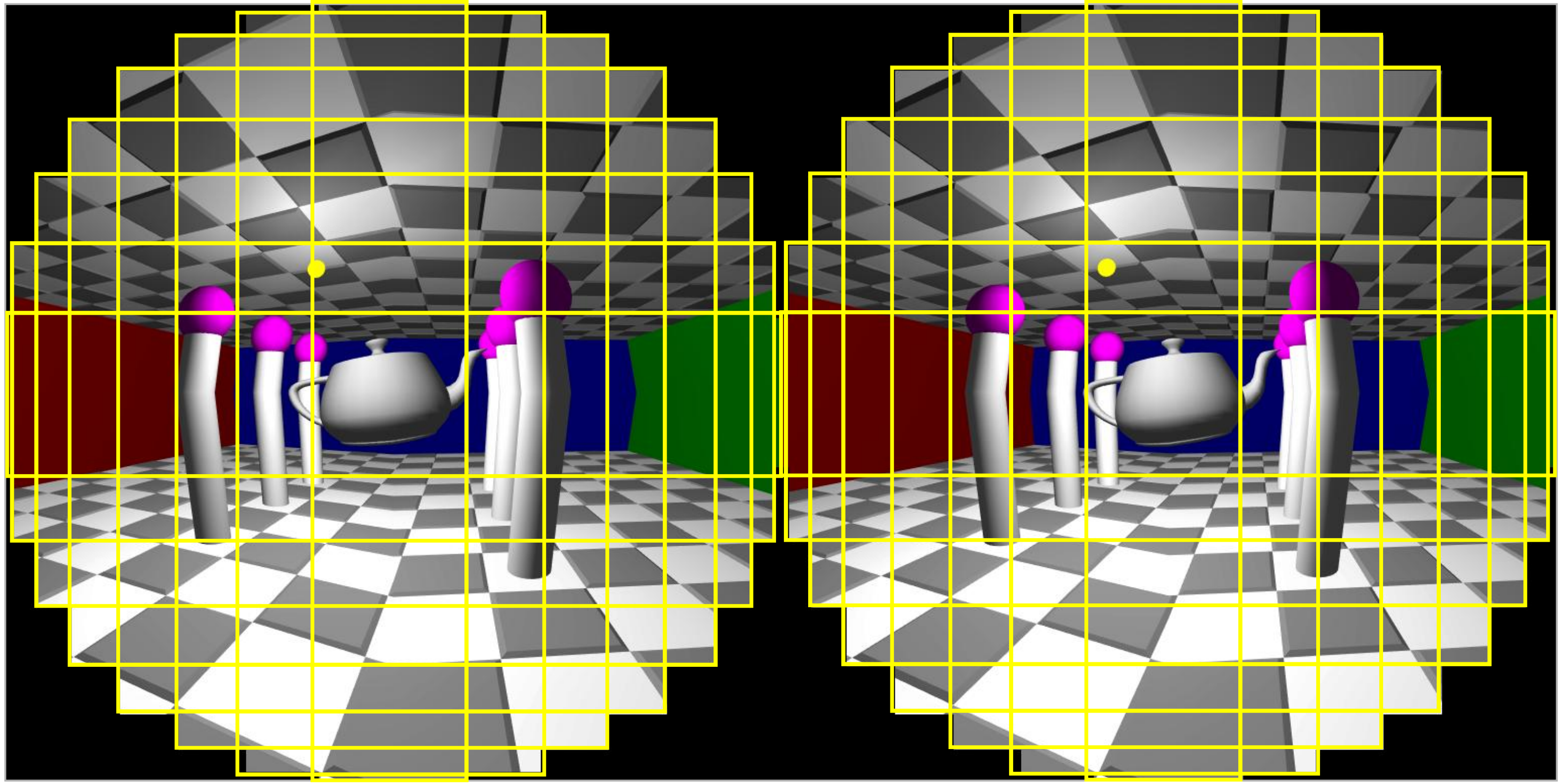
Lens Matched Shading with Window Rectangle Testing



Same Lens Matched Shading but with EXT_window_rectangles

Nothing in black corners is shaded or even rasterized

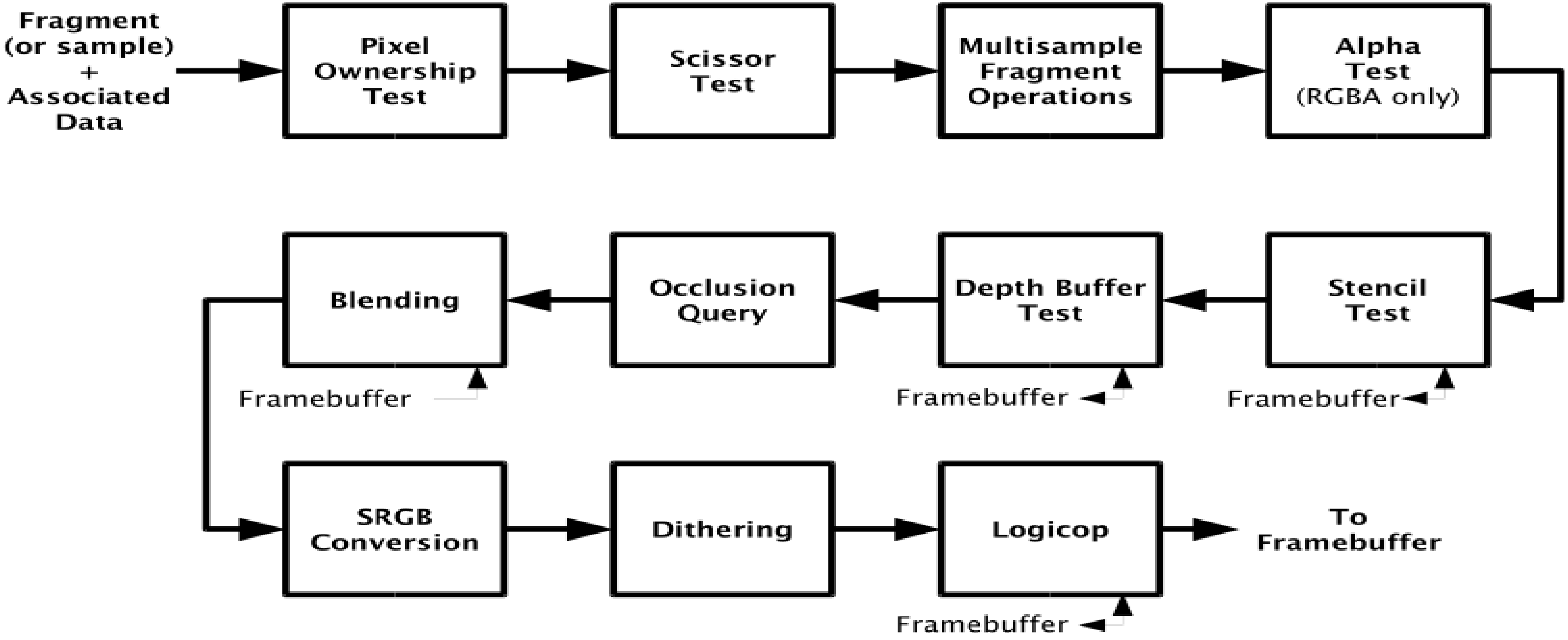
Lens Matched Shading with Window Rectangle Testing



Nothing in black corners is shaded or even rasterized

Yellow lines show overlaid 8 <u>inclusive</u> window rectangles
Same 8 window rectangles "shared" by each view's texture array layer

Standard OpenGL Per-fragment Operations



Fragment (or sample) + Associated Data

Pixel Ownership Test

Scissor Test

Multisample Fragment Operations

Alpha Test (RGBA only)

Stencil Test

Depth Buffer Test

Occlusion Query

Blending

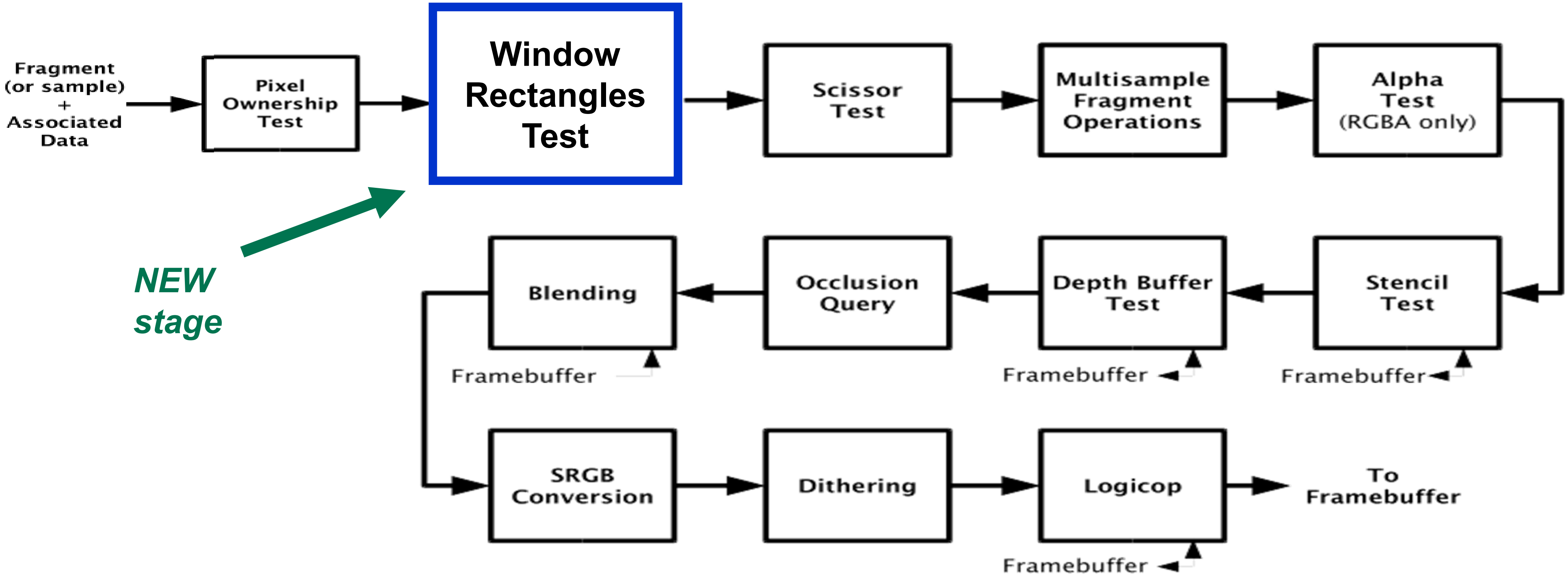
SRGB Conversion

Dithering

Logicop

To Framebuffer

NEW Window Rectangles Test in Per-fragment Operations

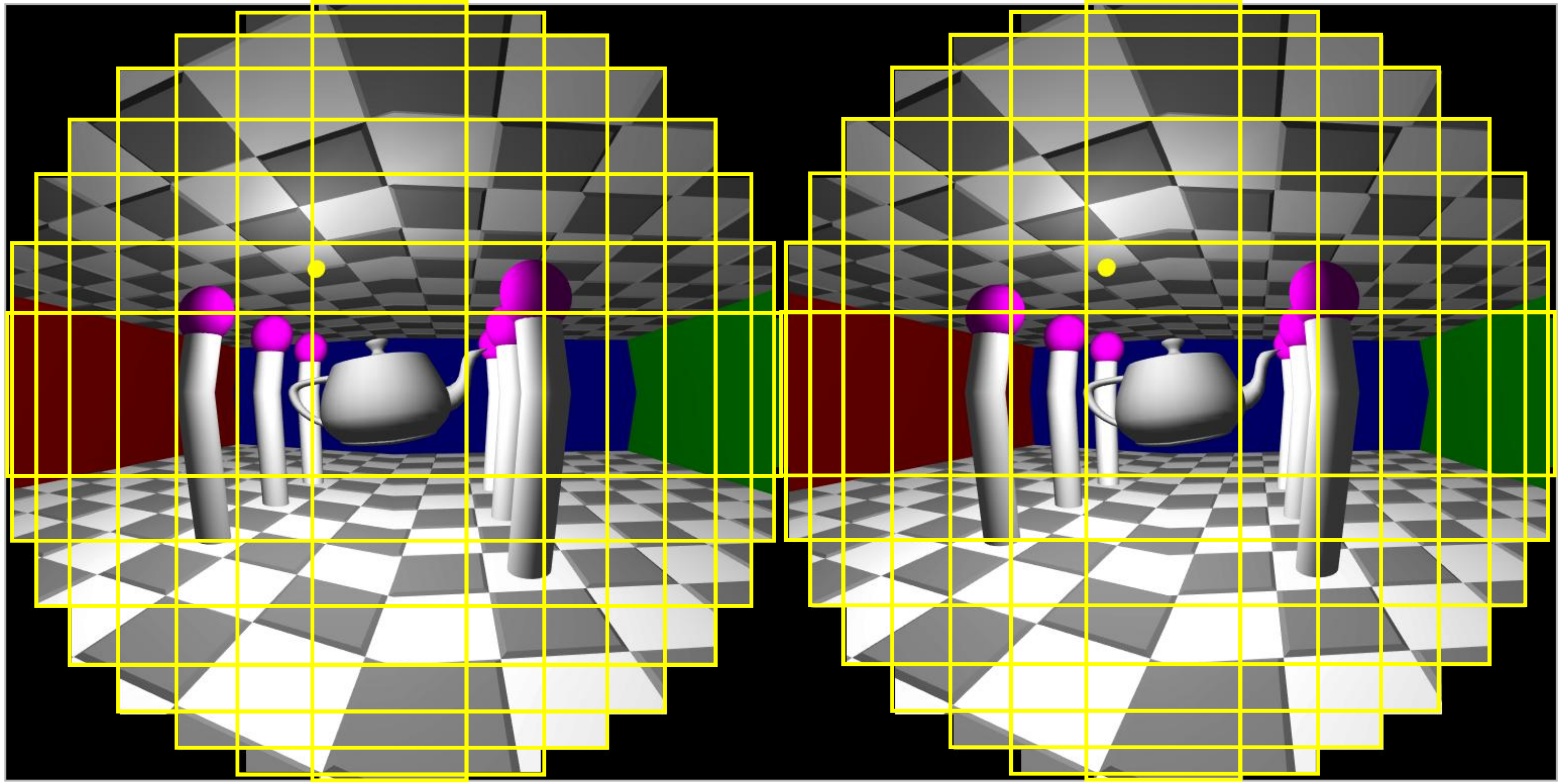


Straightforward API

Multi-vendor **EXT_window_rectangles** Extension

- glWindowRectanglesEXT(GLenum *mode*, GLsizei *count*, const GLint *rects*[]);
  - *mode* can be either GL_INCLUSIVE_EXT or GL_EXCLUSIVE_EXT
  - *count* can be from 0 to maximum number of supported window rectangles
    - Must be at least 4 (for AMD hardware)
    - NVIDIA hardware supports 8
  - Rectangles allowed to overlap and/or disjoint
    - Each rectangle is (*x*,*y*,*width*,*height*)
    - *width* & *height* must be non-negative
- Initial state
  - GL_EXCLUSIVE_NV with zero rectangles
  - Excluding rendering from zero rectangles means nothing is discarded by window rectangles test

Lens Matched Shading with Window Rectangle Testing

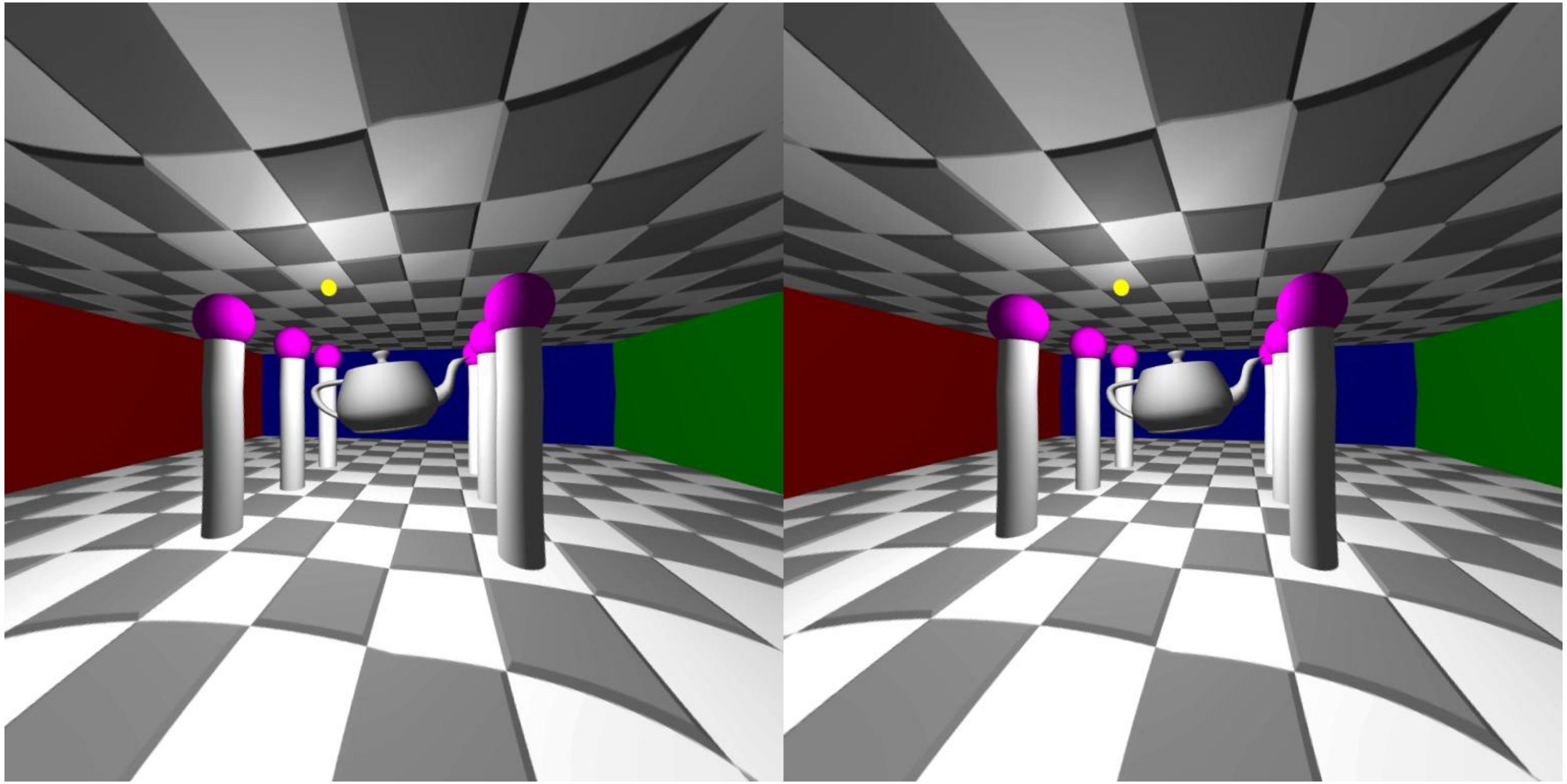


Nothing in black corners is shaded or even rasterized

Yellow lines show overlaid 8 <u>inclusive</u> window rectangles
Same 8 window rectangles “shared” by each view’s texture array layer

Warped Lens Matched Shading

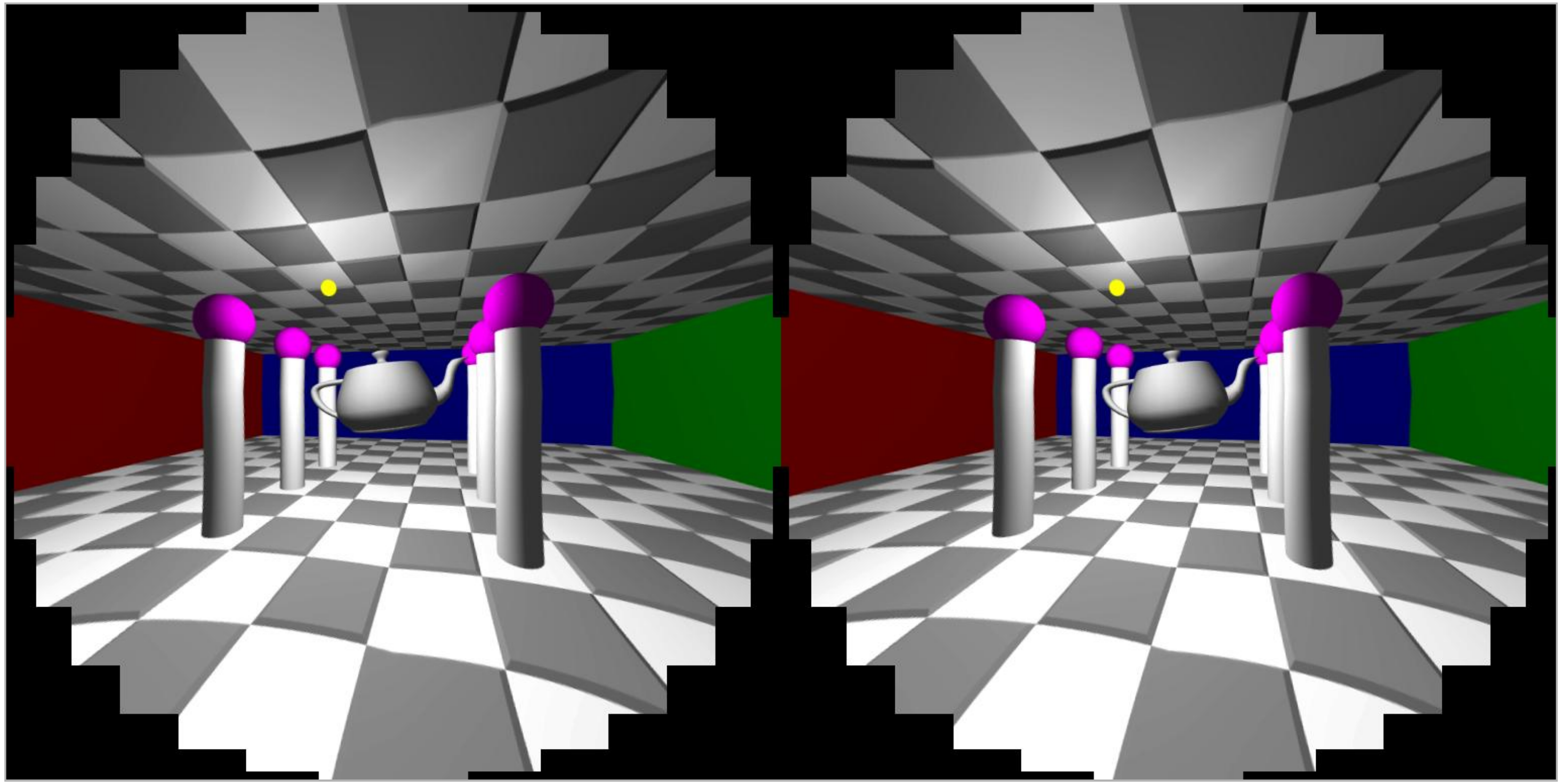
with Window Rectangle Testing during Rendering



Identical as “Lens Matched Shading” despite corners not being rasterized because corners don’t contribute to warped version

Warped Lens Matched Shading

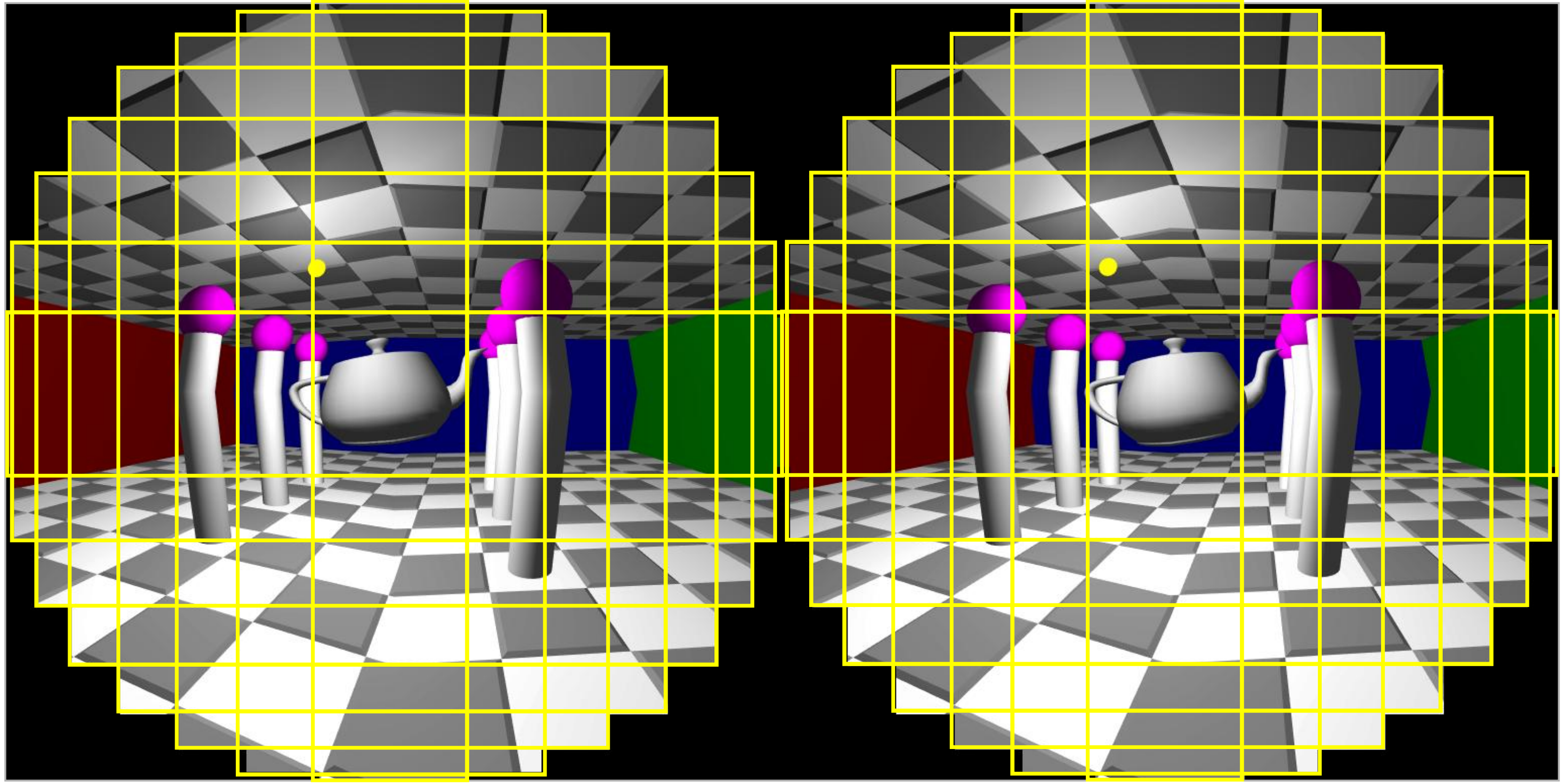
with Win. Rect. Testing during Rendering & Warping



Same prior image, but warp now uses window rectangles

Avoids wasting time warping corners not visible through lens

Visualizing Warp Window Rectangles

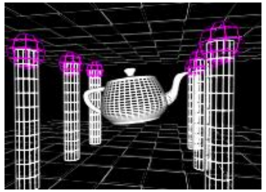


Point: Window rectangle testing used TWICE
#1 during Lens Matched Shading rendering pass
#2 during warping pass

VR Rendering Pipeline

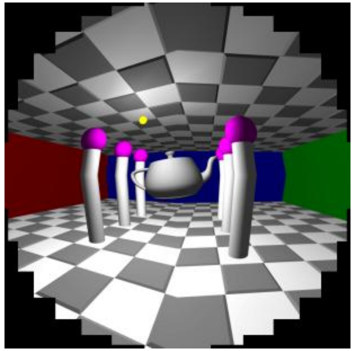
Pascal does all this efficiently in a single rendering pass!

8 viewports, 1 pass

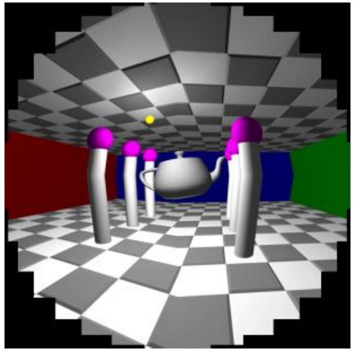


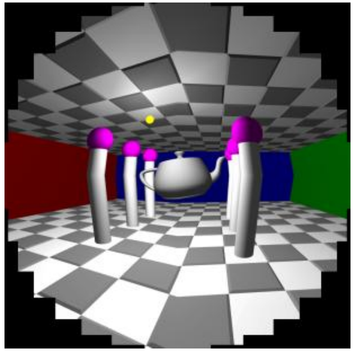
Scene

LMS Left Eye View

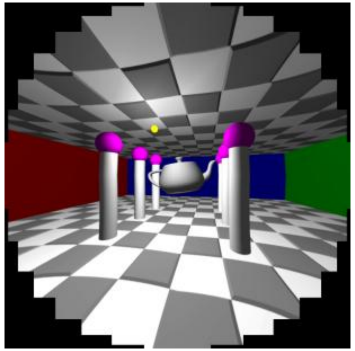


Warped Left Eye View





LMS Right Eye View



Warped Right Eye View

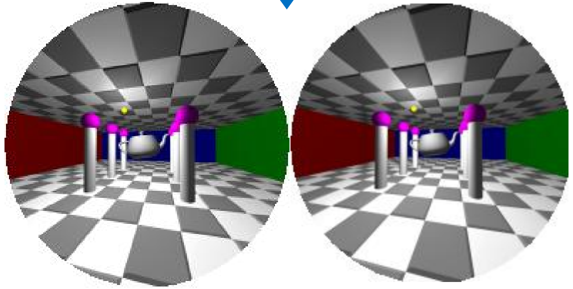
Single Rendering Pass

Single Pass Stereo +
Lens Matched Shading +
Window Rectangle Testing

Drawn with Single Triangle

Fragment Shader Warping
Window Rectangle Testing

Displayed within HMD



Perception to user is linear rendering

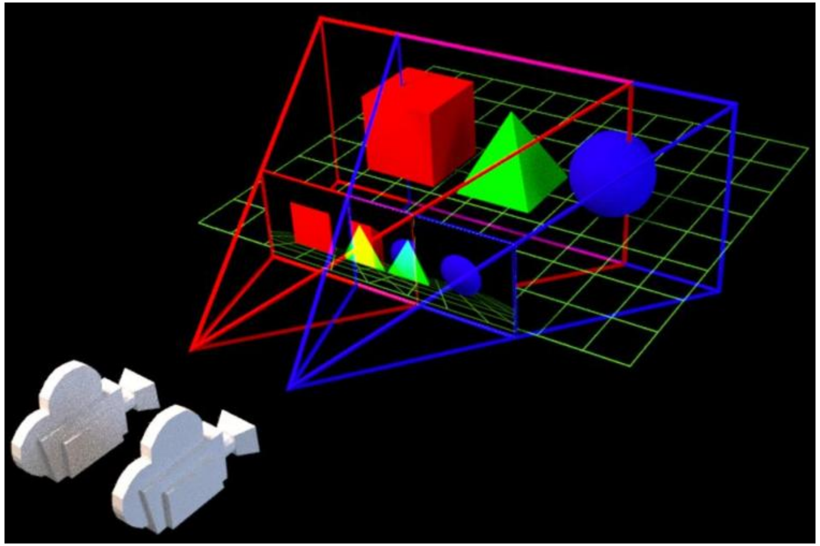
HMD lens “undoes” warping to provide a perceived wide field-of-view

OpenGL Extensions Used in LMS VR Pipeline

Pascal’s **NV_stereo_view_rendering** Extension

SINGLE PASS STEREO

- Allows vertex shader to output two clip-space positions
  - (x_1,y,z,w) and (x_2,y,z,w)
  - Results in TWO primitives
    one for left eye & one for right eye
- New GLSL built-ins
  - gl_SecondaryPositionNV
    - Like gl_Position but for “second eye’s view”
  - gl_SecondaryViewportMaskNV[]
    - Like gl_ViewportMaskNV[] but for “second eye’s view”
- Also can steer primitives to different texture array slices
  - layout(secondary_view_offset = 1) int gl_Layer;



OpenGL Extensions Used in LMS VR Pipeline

Pascal's **NV_clip_space_w_scaling** Extension

Adds a new set of state to viewport array elements

Viewport array state

	x_v y_v w_v h_v	n,f	x_s y_s	w_s h_s e_s	$x_{sw}y_{sw}z_{sw}w_{ws}$	A,B
0	0 0 1024 1024	0,1	0,0,	512,512,1	x+,y+,z+,w+	−0.26,−0.26
1	0 0 1024 1024	0,1	512,0,	512,512,1	y+,z+,x+,w+	+0.26,−02.6
2	0 0 1024 1024	0,1	512,0,	512,512,1	z+,x+,y+,w+	−0.26,−0.26
3	0 0 1024 1024	0,1	512,512,	512,512,1	z+,x+,y+,w+	+0.26,+0.26
…	…					
15						

Four quadrants for Lens Matched Shading

standard viewport array state

swizzle state

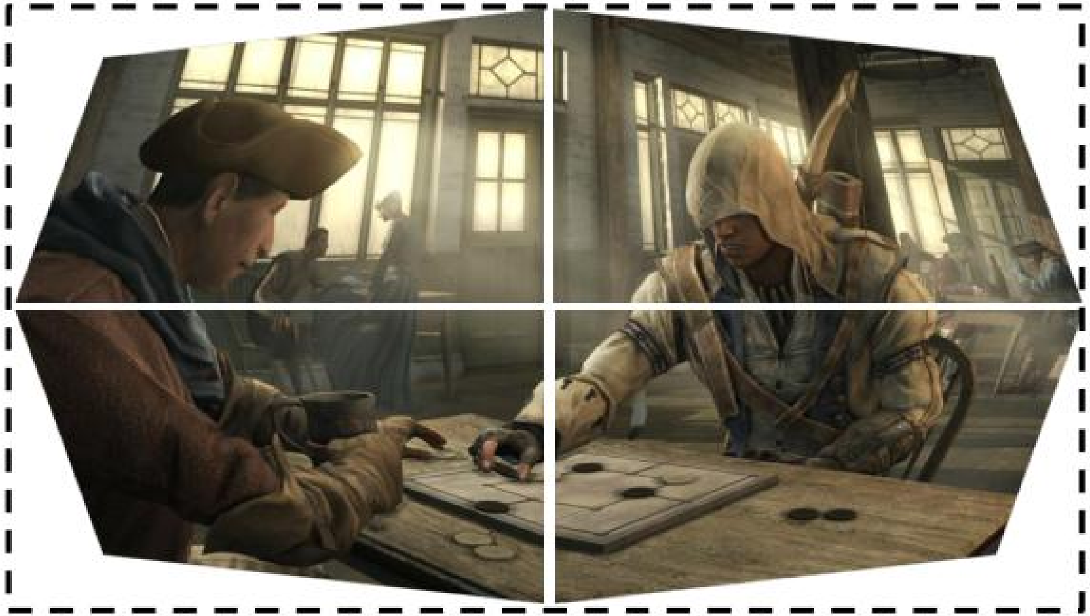
NEW w scaling

Each viewport index can recompute clip space as $w = w + A\ x + B\ y$

Example Lens Matched Shading Rendered Image

A=−0.2, B=+0.2

A=+0.2, B=+0.2



A=−0.2, B=−0.2

A=+0.2, B=−0.2

Example image

More Information on NVIDIA Virtual Reality GPU Support

Get the VRWORKS 2.0 SDK

Growing Software Development Kit (SDK) for Virtual Reality

Focus on GPU efficiency

Whitepapers and sample code

Both OpenGL and Direct3D supported

https://developer.nvidia.com/vrworks

LENS MATCHED SHADING

SINGLE PASS STEREO

Still More Pascal OpenGL Extensions

Pascal's non-Virtual Reality Enhancements

NVX_blend_equation_advanced_multi_draw_buffers

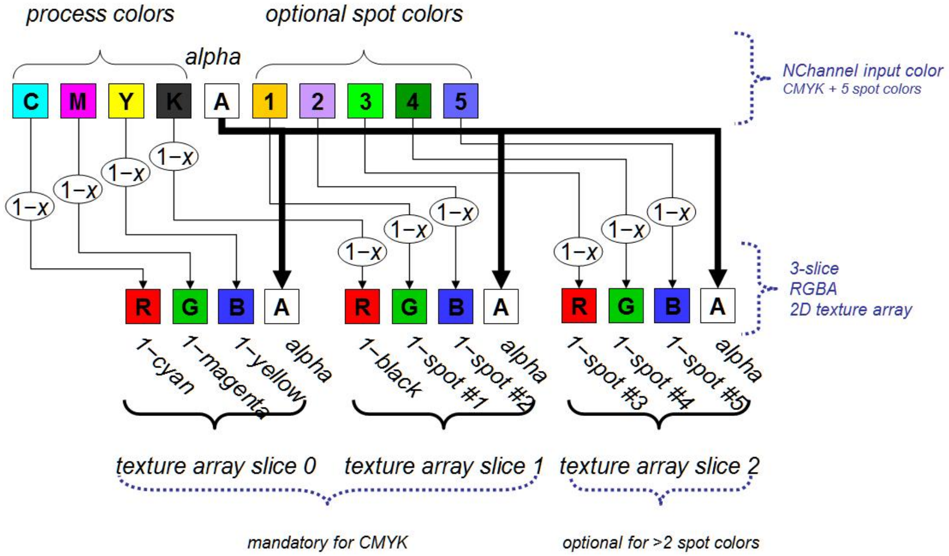
- No API, simply relaxes error restriction so advanced blend modes from KHR_blend_equation_advanced & NV_blend_equation_advanced work with more than 1 color attachment
- Important for CMYK rendering

NV_conservative_raster_pre_snap_triangles

- More Conservative Rasterization control
- Allows conservative rendering dilation prior to sub-pixel snapping

NV_shader_atomic_float64

- Atomic shader operations on double-precision values



CYMK color space rendering with multiple color attachments

OpenGL extension exposing Khronos intermediate language for parallel compute and graphics

Khronos standard extension **ARB_gl_spirv**

New standard Khronos extension for OpenGL

- *Just announced!* July 22, 2016

Allows compiled SPIR-V code to be passed directly to OpenGL driver

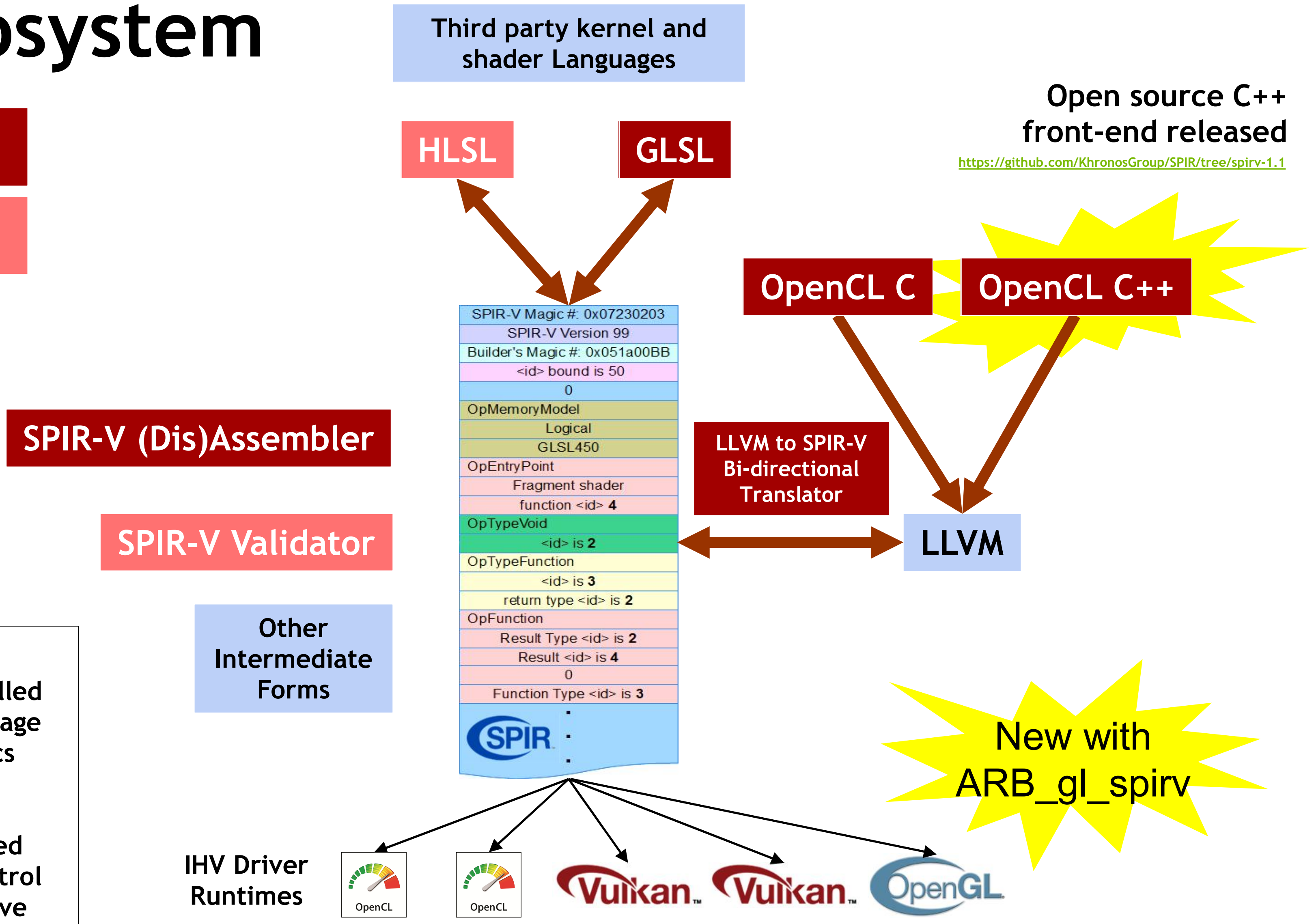
- Accepts SPIR-V output from open source Glslang Khronos Reference compiler
  - https://github.com/KhronosGroup/glslang
- Other compilers can target SPIR-V too

SPIR™ + OpenGL®

SPIR-V Ecosystem

Khronos has open sourced these tools and translators

Khronos plans to open source these tools soon



Third party kernel and shader Languages

HLSL

GLSL

Open source C++ front-end released

https://github.com/KhronosGroup/SPIR/tree/spirv-1.1

OpenCL C

OpenCL C++

SPIR-V (Dis)Assembler

SPIR-V Validator

Other Intermediate Forms

LLVM to SPIR-V Bi-directional Translator

LLVM

SPIR-V Magic #: 0x07230203
SPIR-V Version 99
Builder's Magic #: 0x051a00BB
<id> bound is 50
0
OpMemoryModel
Logical
GLSL450
OpEntryPoint
Fragment shader
function <id> **4**
OpTypeVoid
<id> is **2**
OpTypeFunction
<id> is **3**
return type <id> is **2**
OpFunction
Result Type <id> is **2**
Result <id> is **4**
0
Function Type <id> is **3**
.
.
.

IHV Driver Runtimes

New with ARB_gl_spirv

- •SPIR-V
- •Khronos defined and controlled cross-API intermediate language
- •Native support for graphics and parallel constructs
- •32-bit Word Stream
- •Extensible and easily parsed
- •Retains data object and control flow information for effective code generation and translation

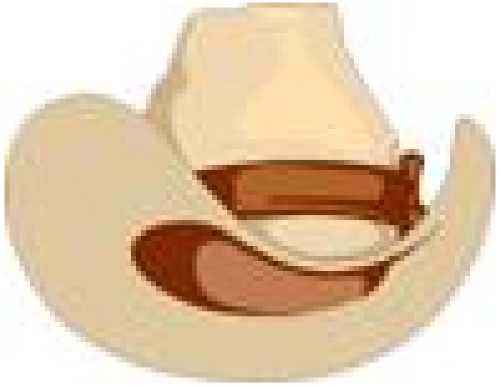
NVIDIA’s SIGGRAPH Driver Update

Developed driver with **ARB_gl_spirv** extension

- NVIDIA historically releases a “developer” driver at SIGGRAPH with support for all Khronos standard extensions announced at SIGGRAPH
  - This year too ☺
- Monday (July 25, 2016) NVIDIA will put out a new SIGGRAPH driver
  - ARB_gl_spirv
    - Major extension in terms of compiler infrastructure & shader support
  - EXT_window_rectangles
  - Updates to Pascal OpenGL extensions
  - For Windows and Linux operating systems

https://developer.nvidia.com/opengl-driver

GLEW Support Available NOW



GLEW = The OpenGL Extension Wrangler Library

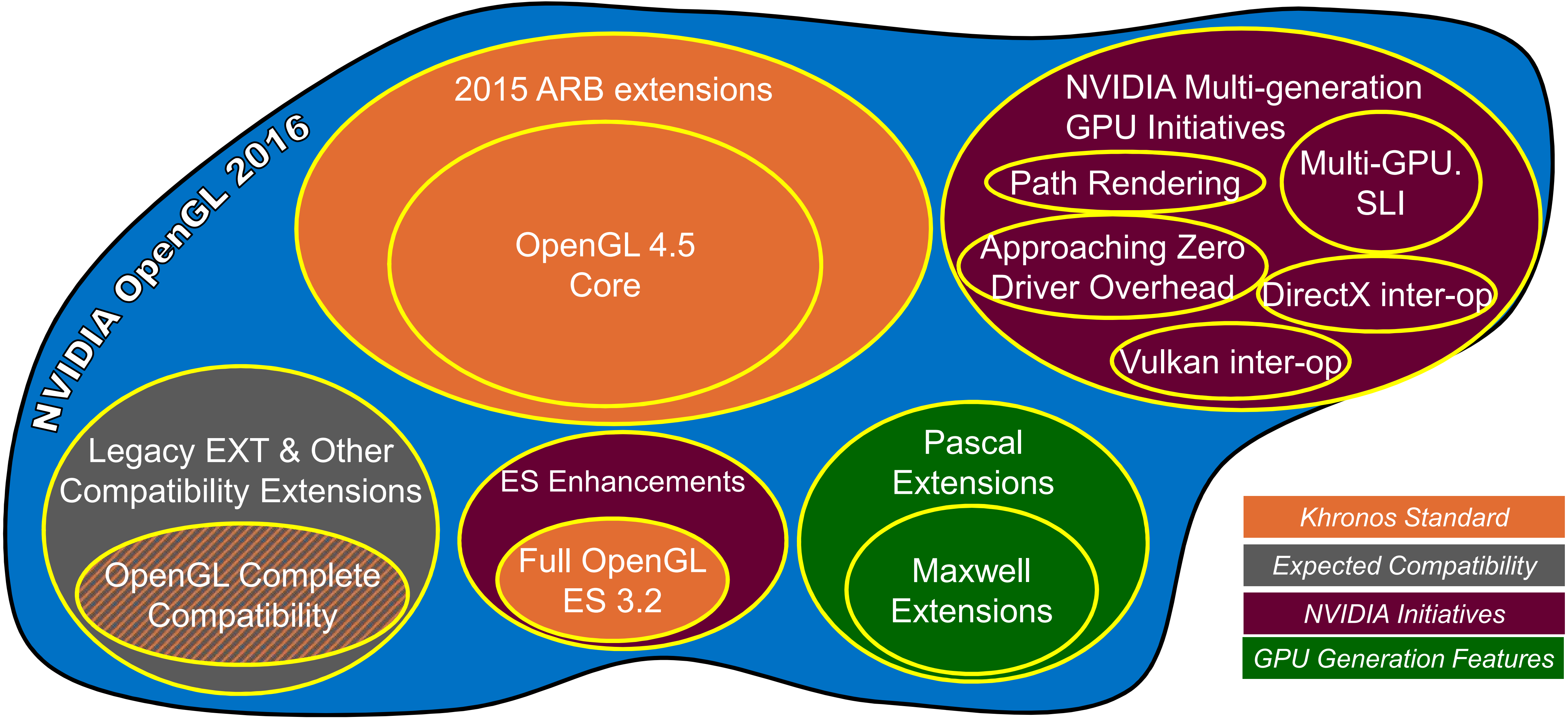
- Open source library
  - Pre-built distribution: http://glew.sourceforge.net/
  - Source code: https://github.com/nigels-com/glew
- Your one-stop-shop for API support for all OpenGL extension APIs

Just released GLEW 2.0 (July 2016) provides API support for

- ARB_gl_spirv
- EXT_window_rectangles
- All of NVIDIA's Maxwell extensions
- All of NVIDIA's Pascal extensions
- All other NVIDIA multi-GPU generation initiatives
  - **Examples:** NV_path_rendering, NV_command_list, NV_gpu_multicast

Thanks to Nigel Stewart, GLEW maintainer, for this

NVIDIA OpenGL in 2016 Provides OpenGL's Maximally Available Superset



Last Words

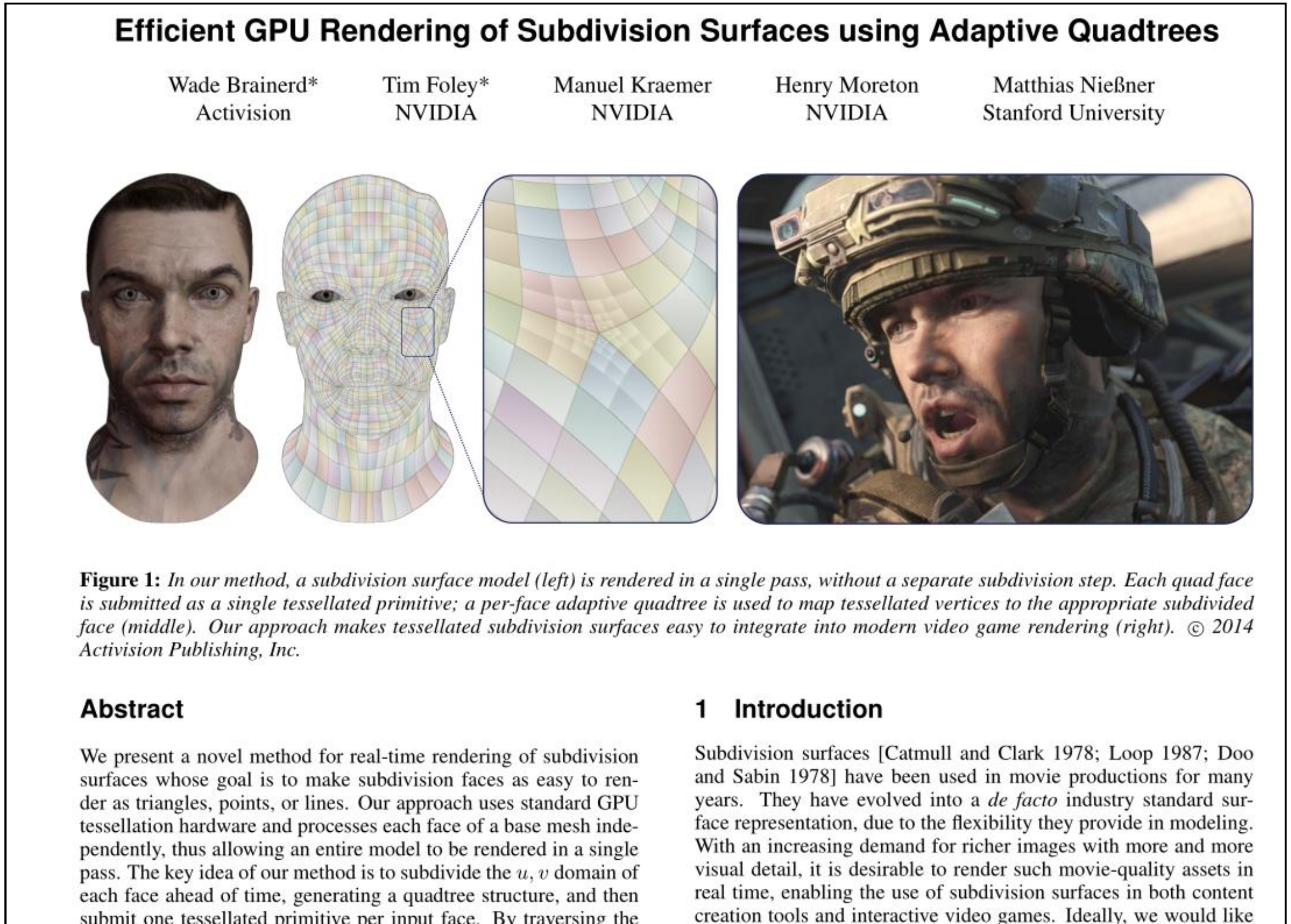
- Lots of new OpenGL features in NVIDIA's 2016 Driver
- Highlights
  - OpenGL 2015 Khronos standard extensions all supported by NVIDIA
  - Maxwell's features for
    - GPU Voxelization & Global Illumination
    - Vector Graphics
    - *And Pascal supports all these features too*
  - Pascal's features for efficient Virtual Reality rendering
  - NVIDIA supports new ARB_gl_spirv extension
    - Provides shader compilation inter-operability for Vulkan and OpenGL

SIGGRAPH Paper Using OpenGL to Check Out

- Harnesses OpenGL-based GPU tessellation

- Avoids the complex patch splitting in current OpenSubdiv approach

- Wednesday, July 27
- Ballroom C/D/E
- 3:45 to 5:55pm session



Efficient GPU Rendering of Subdivision Surfaces using Adaptive Quadtrees

Wade Brainerd* Activision
Tim Foley* NVIDIA
Manuel Kraemer NVIDIA
Henry Moreton NVIDIA
Matthias Nießner Stanford University

Figure 1: *In our method, a subdivision surface model (left) is rendered in a single pass, without a separate subdivision step. Each quad face is submitted as a single tessellated primitive; a per-face adaptive quadtree is used to map tessellated vertices to the appropriate subdivided face (middle). Our approach makes tessellated subdivision surfaces easy to integrate into modern video game rendering (right).* © *2014 Activision Publishing, Inc.*

Abstract

We present a novel method for real-time rendering of subdivision surfaces whose goal is to make subdivision faces as easy to render as triangles, points, or lines. Our approach uses standard GPU tessellation hardware and processes each face of a base mesh independently, thus allowing an entire model to be rendered in a single pass. The key idea of our method is to subdivide the u, v domain of each face ahead of time, generating a quadtree structure, and then submit one tessellated primitive per input face. By traversing the

1 Introduction

Subdivision surfaces [Catmull and Clark 1978; Loop 1987; Doo and Sabin 1978] have been used in movie productions for many years. They have evolved into a *de facto* industry standard surface representation, due to the flexibility they provide in modeling. With an increasing demand for richer images with more and more visual detail, it is desirable to render such movie-quality assets in real time, enabling the use of subdivision surfaces in both content creation tools and interactive video games. Ideally, we would like

NVIDIA