ENGINEERING DATASETS ON HMDS
WITH ESI VIRTUAL REALITY TECHNOLOGY AND NVIDIA VRWORKS

Andreas Mank, Ingo Esser
8 MAY 2017, 2:00 PM – 2:50 PM – Room 230A
1. Locality
2. Simplicity
3. Temporality
DATA SET

• 155 866 993 Triangles
• 200 526 Nodes
• Stereo
SCENIX (KEPLER)

- 2 FPS
- NVIDIA QUADRO K6000
Scalability

SceniX

![Graph showing scalability comparison between Kepler and Maxwell across K5000, M5000, and M6000 models]

- Talent2
- B
- C

Kepler
Maxwell
“ESI Rendering Innovations with NVIDIA DesignWorks™”

Andreas Mank Team, Leader Visualization, ESI Group
Markus Tavenrath, Senior Developer Technology Engineer, NVIDIA

— Source: GTC 2016, s6306
Scalability

SceniX vs. NVPro

<table>
<thead>
<tr>
<th>GPU</th>
<th>SceniX</th>
<th>NVPro</th>
</tr>
</thead>
<tbody>
<tr>
<td>K5000</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>M5000</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>M6000</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

- Kepler
- Maxwell

SceniX is 6.5 times faster than NVPro.
NVPRO-PIPELINE (MAXWELL)

- 13 FPS
- NVIDIA QUADRO M6000
- Duration: 1 Year

11.7%
VRWorks - VRSLI

https://developer.nvidia.com/vrworks/graphics/vrsli
VRWorks - VRSLI

GPU0

GPU1

Left

Right

Render Right Eye

Render Left Eye

Output to HMD
VRWorks - VRSLI

Initialization

typedef void ( GLAPIENTRY * PFNGLGPUCOPYIMAGESUBDATANVXPROC )( ... );
typedef void ( GLAPIENTRY * PFNGLGPUINTERLOCKNVXPROC )( void );

PFNGLGPUCOPYIMAGESUBDATANVXPROC glLGPUCopyImageSubDataNVX( nullptr );
PFNGLGPUINTERLOCKNVXPROC glLGPUInterlockNVX( nullptr );

if ( logicalGpuCount < physicalGpuCount ) {
    putenv( "GL_NVX_LINKED_GPU_MULTICAST=1" );

    // CREATE CONTEXT HERE!
}

GLint numExtensions; glGetIntegerv( GL_NUM_EXTENSIONS, &numExtensions );
for ( GLint i = 0; i < numExtensions && !found; ++i ) {
    std::string name( ( const char* )glGetStringi( GL_EXTENSIONS, I ) );
    if ( name == "GL_NVX_linked_gpu_multicast" ) {
        glLGPUCopyImageSubDataNVX = ( PFNGLGPUCOPYIMAGESUBDATANVXPROC )wglGetProcAddress( "glLGPUCopyImageSubDataNVX" );
        glLGPUInterlockNVX = ( PFNGLGPUINTERLOCKNVXPROC )wglGetProcAddress( "glLGPUInterlockNVX" );
        break;
    }
}

VRWorks - VRSLI

Rendering

1. cameraBuffersUpload( projectionMatrix[ EYE_LEFT ], viewMatrix[ EYE_LEFT ], inverseViewMatrix[ EYE_LEFT ], 0 );
cameraBuffersUpload( projectionMatrix[ EYE_RIGHT ], viewMatrix[ EYE_RIGHT ], inverseViewMatrix[ EYE_RIGHT ], 1 );

2. glBindFramebuffer( GL_DRAW_FRAMEBUFFER, fbos[ EYE_LEFT ].fboId);
   // RENDER HERE!

3. glLGPUInterlockNVX();
glLGPUCopyImageSubDataNVX(
    1 // source GPU
   , GPUMASK_0 // destination GPU mask
   , fbos[EYE_LEFT].colorAttachmentTextureId // source name
   , GL_TEXTURE_2D // source target
   , 0, 0, 0, 0 // source: level, X, Y, Z
   , fbos[EYE_RIGHT].colorAttachmentTextureId // destination name
   , GL_TEXTURE_2D // destination target
   , 0, 0, 0, 0 // destination: level, X, Y, Z
   , width
   , height
   , depth
);
glLGPUInterlockNVX();
VRSLI (MAXWELL)

- 21 FPS
- 2 x NVIDIA QUADRO M6000
- Duration: 1 Week
VRSLI (PASCAL)

- 32 FPS
- 2 x NVIDIA QUADRO P6000
VRWorks – Single Pass Stereo


by courtesy of NVIDIA
VRWorks – Single Pass Stereo

Vertex Shader

1. #extension GL_NV_viewport_array2: require
   #extension GL_NV_stereo_view_rendering: require

   layout( secondary_view_offset=1 ) out highp int gl_Layer;

   ...

2. mat4 model2view_left = world2view * model2world;
   vec4 pos_ec_left = model2view * vec4( position, 1. );
   gl_Position = view2clip * pos_ec_left;

3. mat4 model2view_right = world2view_right * model2world;
   vec4 pos_ec_right = model2view_right * vec4( position, 1. );
   gl.SecondaryPositionNV = view2clip_right * pos_ec_right;

4. gl_Layer = 0;
   glViewportMask[0] = 1;
   gl.SecondaryViewportMaskNV[0] = 2;

   ...

Copyright © ESI Group, 2016. All rights reserved.
VRWorks – Single Pass Stereo

Initialization and Rendering

```c
GLint numExtensions; glGetIntegerv( GL_NUM_EXTENSIONS, &numExtensions );
for ( GLint i = 0; i < numExtensions && !found; ++i ) {
    std::string name(( const char* )glGetStringi( GL_EXTENSIONS, I ));
    if( name == "GL_NV_stereo_view_rendering" ) {
        // EXTENSION IS AVAILABLE!
        break;
    }
}
```

```
... cameraBuffersUpload( "view2clip", projectionMatrix[EYE_LEFT].getData(), 16 * sizeof( float ), 0 );
cameraBuffersUpload( "world2view", viewMatrix[EYE_LEFT].getData(), 16 * sizeof( float ), 0 );
```

```
cameraBuffersUpload( "view2clip_right", projectionMatrix[EYE_RIGHT].getData(), 16 * sizeof( float ), 0 );
cameraBuffersUpload( "world2view_right", viewMatrix[EYE_RIGHT].getData(), 16 * sizeof( float ), 0 );
```

```
glViewportIndexedf( 0, 0, 0, width, height );
glViewportIndexedf( 1, width, 0, width, height );
```

// RENDER HERE!
```
SINGLE PASS STEREO (PASCAL)

- 32 FPS
- NVIDIA QUADRO P6000
- Duration: 1 Week
"VRWorks (VRSLI/SPS) increases performance by 30%-80%, if the rendering performance is below 90FPS."

<table>
<thead>
<tr>
<th>Scene</th>
<th>VRSLI</th>
<th>SPS*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Nodes</td>
<td>#Objects</td>
</tr>
<tr>
<td></td>
<td>11.633</td>
<td>11.675</td>
</tr>
<tr>
<td></td>
<td>37.400</td>
<td>41,483,929</td>
</tr>
<tr>
<td></td>
<td>396</td>
<td>31,627,787</td>
</tr>
<tr>
<td></td>
<td>24.032</td>
<td>22,187,160</td>
</tr>
<tr>
<td></td>
<td>1.940</td>
<td>6,025,615</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5,980,277</td>
</tr>
<tr>
<td></td>
<td>2.055</td>
<td>11,353,007</td>
</tr>
<tr>
<td></td>
<td>28.897</td>
<td>81,877,312</td>
</tr>
<tr>
<td></td>
<td>15.138</td>
<td>37,805,318</td>
</tr>
<tr>
<td></td>
<td>32.322</td>
<td>61,232,257</td>
</tr>
<tr>
<td></td>
<td>214.802</td>
<td>85,922</td>
</tr>
<tr>
<td></td>
<td>6.757</td>
<td>26,711,554</td>
</tr>
<tr>
<td></td>
<td>65.027</td>
<td>48,890</td>
</tr>
<tr>
<td></td>
<td>156.811</td>
<td>117,252</td>
</tr>
<tr>
<td></td>
<td>94.869</td>
<td>47,075</td>
</tr>
</tbody>
</table>

* SPS: NVIDIA's Single Pass Stereo

< 90 FPS  > 90 FPS
“GPU-DRIVEN RENDERING”

Christoph Kubisch, Senior Developer Technology Engineer, NVIDIA
Pierre Boudier, Quadro Software Architect, NVIDIA
— Source: GTC 2016, s6138

“OPENGL SCENE-RENDERING TECHNIQUES”

Christoph Kubisch, Senior Developer Technology Engineer, NVIDIA
— Source: SIGGRAPH 2014, SG4117
GPU Occlusion Culling

https://github.com/nvpro-samples/gl_occlusion_culling

• Render color and depth
  ▪ Enable color/depth writes

• Clear visibility buffer

• Raster „invisible“ bounding boxes
  ▪ Disable color/depth writes
  ▪ Geometry shader to create the three visible box sides
  ▪ Depth buffer discards occluded fragments (earlyZ …)
  ▪ Fragment shader writes output: visible[0] = 1

• Set object visibility
  ▪ Visibility buffer CPU readback
  ▪ Set visibility mask

```glsl
// GLSL fragment shader
// from ARB_shader_image_load_store
layout( early_fragment_tests ) in;

buffer visibilityBuffer {
  int visibility[]; // cleared to 0
};

flat in int objectId; // unique per box

void main() {
  visibility[objectId] = 1;
  // no atomics required (32-bit write)
}
```

by courtesy of NVIDIA
GPU CULLING (PASCAL)

- 94 FPS
- NVIDIA QUADRO P6000
- Duration: 2 Weeks
GPU OCCLUSION CULLING (PASCAL)
# NVIDIA VRWORKS
Comprehensive SDK for VR Developers

<table>
<thead>
<tr>
<th>GRAPHICS</th>
<th>HEADSET</th>
<th>TOUCH &amp; PHYSICS</th>
<th>AUDIO</th>
<th>VIDEO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lens Matched Shading</td>
<td>Context Priority</td>
<td>PhysX</td>
<td>VRWorks Audio</td>
<td>360 Video</td>
</tr>
<tr>
<td>Single Pass Stereo</td>
<td>Direct Mode</td>
<td>Warp &amp; Blend</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MultiRes Shading</td>
<td>Front Buffer Rendering</td>
<td>Synchronization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VR SLI</td>
<td></td>
<td>GPU Affinity</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>GPU Direct for Video</td>
<td></td>
</tr>
</tbody>
</table>
### NVIDIA VRWORKS

Comprehensive SDK for VR Developers

<table>
<thead>
<tr>
<th>GRAPHICS</th>
<th>HEADSET</th>
<th>TOUCH &amp; PHYSICS</th>
<th>AUDIO</th>
<th>VIDEO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lens Matched Shading</td>
<td>Context Priority</td>
<td>PhysX</td>
<td>VRWorks Audio</td>
<td>VRWorks 360</td>
</tr>
<tr>
<td>Single Pass Stereo</td>
<td>Direct Mode</td>
<td></td>
<td></td>
<td>360 Video</td>
</tr>
<tr>
<td>Multires Shading</td>
<td>Front Buffer Rendering</td>
<td>WARP &amp; BLEND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VR SLI</td>
<td></td>
<td>Synchronization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>GPU Affinity</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
VR PERFORMANCE DEMANDS

Fragment load, vertex load, frame time

124 M fragments / sec  ~3.6x  457 M fragments / sec
N Vertices @ 60Hz  3x  2N Vertices @ 90Hz

60 Hz

90 Hz
VR REQUIREMENTS - ANTI-ALIASING

No Anti-Aliasing

Spacial aliasing: High
Temporal aliasing: High
VR REQUIREMENTS - ANTI-ALIASING

8 x multisampling

Spacial aliasing: Low
Temporal aliasing: High
VR REQUIREMENTS - ANTI-ALIASING

8 x multisampling
4 x supersampling
Spacial aliasing: Low
Temporal aliasing: Low
# NVIDIA VRWORKS
Comprehensive SDK for VR Developers

<table>
<thead>
<tr>
<th>GRAPHICS</th>
<th>HEADSET</th>
<th>TOUCH &amp; PHYSICS</th>
<th>AUDIO</th>
<th>VIDEO</th>
</tr>
</thead>
<tbody>
<tr>
<td>LENSTMATCHED SHADING</td>
<td>CONTEXT PRIORITY</td>
<td>PHYSX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SINGLE PASS STEREO</td>
<td></td>
<td></td>
<td>VRWORKS AUDIO</td>
<td>VRWORKS 360 VIDEO</td>
</tr>
<tr>
<td>MULTires SHADING</td>
<td>DIRECT MODE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VR SLI</td>
<td>FRONT BUFFER RENDERING</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>WARP &amp; BLEND</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SYNCHRONIZATION</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>GPU AFFINITY</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>360</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GPUDIRECT FOR VIDEO</td>
</tr>
</tbody>
</table>
SINGLE PASS STEREO

Traditional Rendering

Render eyes separately

Doubles CPU and GPU load
Single Pass Stereo uses Simultaneous Multi-Projection architecture

Draw geometry only once

Vertex/Geometry stage runs once
Outputs two positions for left/right

Only rasterization is performed per-view
SINGLE PASS STereo

OpenGL

In OpenGL via GL_NV_stereo_view_rendering

Create texture array for rendering left and right eye simultaneously

No other changes needed, shaders perform SPS
SINGLE PASS STEREO
Vertex Shader

Calculate projection space position

\[ \text{proj\_pos} = \text{proj} \times \text{view} \times \text{model} \times \text{inPosition}; \]

Output both positions via different builtin variables, only x component may differ

\[ \text{gl\_Position} = \text{proj\_pos} + \text{vec4(\text{offset, 0, 0, 0})}; \]
\[ \text{gl\_SecondaryPositionNV} = \text{proj\_pos} - \text{vec4(\text{offset, 0, 0, 0})}; \]

Use declaration and value of \text{gl\_Layer} to route output to layers 0 and 1 of tex array

\[ \text{layout(secondary\_view\_offset=1) out highp int gl\_Layer}; \]
\[ \text{gl\_Layer} = 0; \]
Single Pass Stereo brings benefits in geometry bound scenarios.

Heavy fragment shaders will reduce scaling.
NVIDIA VRWORKS
Comprehensive SDK for VR Developers

**GRAPHICS**
- Lens Matched Shading
- Single Pass Stereo
- MultiRes Shading
- VR SLI

**HEADSET**
- Context Priority
- Direct Mode
- Front Buffer Rendering

**TOUCH & PHYSICS**
- PhysX

**AUDIO**
- VRWORKS Audio

**VIDEO**
- Warp & Blend
- Synchronization
- GPU Affinity
- GPUDirect for Video
HMD OPTICS
Countering Lens Distortion

Displayed Image → Optics → User’s View
HMD RENDERING

Oversampling near the borders

Rendered Image

Displayed Image
LENS MATCHED SHADING

\[ w' = w + Ax + By \]
LENS MATCHED SHADING

Four Viewports

Original Image

LMS Image
In OpenGL via `GL_NV_clip_space_w_scaling` extension

Set up four viewports, rendering full resolution

Set scissors to each quadrant

```c
glScissorArray(0, 4, scissors);
```

W scaling parameters

```c
glViewportPositionWScaleNV(i, Wx, Wy);
```
LENS MATCHED SHADING

Shaders

gl_ViewportMask[0] controls broadcasting of vertices and primitives

Inefficient - set mask in vertex shader

\[
gl_ViewportMask[0] = 15;
\]

More efficient - filter in pass through geometry shader

Determine quadrant(s) for each primitive

Set bit(s) in \( gl_ViewportMask[0] \)
LENS MATCHED SHADING
Scaling and Unscaling

HMD runtime can't consume warped images yet, need to unscale before submit.

\[
scale = \frac{1}{1 - w_x P'_x - w_y P'_y}
\]

\[
P' = scale \times P
\]

\[
unscale = \frac{1}{1 + w_x P_x + w_y P_y}
\]

\[
P = unscale \times P'
\]
LENS MATCHED SHADING

Extreme example, $W_x = 2.0$  $W_y = 2.0$
LENS MATCHED SHADING

Extreme example, Wx = 2.0  Wy = 2.0
LMS can improve performance of Raster / Fragment stage

Trade-off between quality and performance
**NVIDIA VRWORKS**
Comprehensive SDK for VR Developers

<table>
<thead>
<tr>
<th>GRAPHICS</th>
<th>HEADSET</th>
<th>TOUCH &amp; PHYSICS</th>
<th>AUDIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>LENS MATCHED SHADING</td>
<td>CONTEXT PRIORITY</td>
<td>PHYSX</td>
<td>VRWORKS AUDIO</td>
</tr>
<tr>
<td>SINGLE PASS STEREO</td>
<td>DIRECT MODE</td>
<td></td>
<td>VIDEO</td>
</tr>
<tr>
<td>MULTITRIP SHADING</td>
<td>FRONT BUFFER RENDERING</td>
<td>WARP &amp; BLENDE</td>
<td>VRWORKS 360 VIDEO</td>
</tr>
<tr>
<td>VR SLI</td>
<td></td>
<td>SYNCHRONIZATION</td>
<td>GPU AFFINITY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>GPUDIRECT FOR VIDEO</td>
</tr>
</tbody>
</table>
TYPICAL SLI AFR

GPUs render alternate frames

Latency

CPU: N - N+1

GPU 0: N_L - N_R

GPU 1: N+1_L - N+1_R

Display: N - N+1
VR SLI

Each GPU renders one eye - lower latency

- CPU
- GPU 0
- GPU 1
- Display

Latency
## HMD RENDERING

### Dual vs. single pass rendering

<table>
<thead>
<tr>
<th>Typical HMD rendering</th>
<th>VR SLI HMD rendering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prepare scene</td>
<td>Prepare scene</td>
</tr>
<tr>
<td>Upload left view data</td>
<td>Upload left view data to GPU0</td>
</tr>
<tr>
<td>Render scene</td>
<td>Upload right view data to GPU1</td>
</tr>
<tr>
<td>Upload right view data</td>
<td>Render scene on both GPUs</td>
</tr>
<tr>
<td>Render scene</td>
<td>Transfer texture</td>
</tr>
<tr>
<td>Submit to HMD</td>
<td>Submit to HMD</td>
</tr>
</tbody>
</table>
HMD RENDERING
VR SLI functionality

VR SLI HMD rendering

Prepare scene
Upload left view data to GPU0
Upload right view data to GPU1
Render scene on both GPUs
Transfer texture
Submit to HMD

← GL allocations & uploads are broadcast
← Separated data upload
← GL render calls are broadcast
← Efficient texture copies
VR SLI

Updates between NVX and NV extensions

Command & data broadcast
BufferSubData to specific GPU
CopyImageSubData & CopyBufferSubData
GPU-GPU Framebuffer Blit
Global barrier & directed sync functions
GPU Masks
Per-GPU sample locations
Per-GPU queries
VR SLI
Broadcast allocations & uploads

Left view data
Geometry
Parameters
Textures
Right view data

tex0
tex1
tex0
tex1
for( auto i = 0; i < 2; ++i )
{
    sceneData.viewMatrix = view[i];
    sceneData.viewProjMatrix = proj[i] * view[i];

    glMulticastBufferSubDataNV (
        1<<i,
        sceneUbo,
        0, sizeof(SceneData), &sceneData
    );
}
VR SLI
Broadcast render commands

Application sends draw commands only once
Commands are broadcast between GPUs

Render
VR SLI
Broadcast render commands

```c
glBindFramebuffer( ..., renderFBO );

glFramebufferTexture2D( ..., tex0, 0 );

render();
```

- tex0 on both GPUs
- render on both GPUs
Copy function allows direct copy between GPUs

Avoids CPU copy, transfer directly via PCIe

```c
glMulticastWaitSyncNV(0, GPUMASK_1);
glMulticastCopyImageSubDataNV(1, 1<<0, tex0, ..., tex1, ..., width, height, 1);
glMulticastWaitSyncNV(1, GPUMASK_0);
```

GPU 1 wait for GPU 0 (Target is ready)

GPU 0 wait for GPU 1 (Copy is done)

copy tex0 @ GPU 1 to tex1 @ GPU0
VR SLI covers a wide variety of workloads

Perfect load balancing between left/right eye and two GPUs

Copy overhead and view independent workloads limit scaling

Preprocessing

Geometric Pipeline

Rasterization Fragment Shader

Postprocessing
TRY IT OUT!

NVIDIA VRWorks SDK provides OpenGL, Direct3D & Vulkan samples

developer.nvidia.com/vrworks

Extensions

www.khronos.org/registry/OpenGL/extensions/NV/NV_stereo_view_rendering.txt
www.khronos.org/registry/OpenGL/extensions/NV/NV_clip_space_w_scaling.txt
www.khronos.org/registry/OpenGL/extensions/NV/NV_gpu_multicast.txt