NVISION 08
THE WORLD OF VISUAL COMPUTING

Does Your Software Scale ?
Multi-GPU Scaling for Large Data Visualization

Thomas Ruge, NVIDIA

NVIDIA.

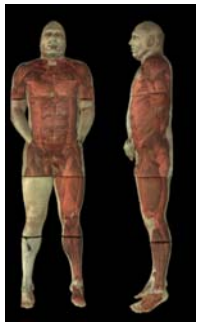
Agenda

- Multi-GPU Scaling for Large Data Visualization (Thomas Ruge, NVIDIA)
  - Multi GPU why ?
  - Different Multi-GPU rendering methods, focus on database decomposition
  - System Analysis of depth compositing and its impact on multi-GPU rendering
  - Performance results of multi-GPU rendering with NVSG and OSG
- NVIDIA's Multi-GPU SDK (Thomas Volk, NVIDIA)
  - How to use NVIDIA's MGPU SDK to get scalability in your application
- NVSG-Scale (Subu Krishnamoorthy, NVIDIA)
  - Multi-GPU implemented in NVSG
- Technical Demos (with Horn Thorolf Tonjum, Stormfjord)
- Q&A

Multi-GPU: Why ?

- Demand for Large Data visualization beyond capacity of 1 GPU
  - Boeing 777 (350 MTriangles) (between 4.2 – 29 GB of graphics data)
  - Visible Human 16 GB of 3D Textures

- Single GPU technology approaches cliff (like CPU's do)
  - Power consumption is growing
  - Cooling problems with high heat output
  - High production costs per chip of dies with large footprints
    G80 : 681 M transistors, 90nm, 484 mm2
    GT200: 1 bn transistors, 65 nm, 576 mm2



- Higher Performance than fastest single GPU on the market
  - MGPU can give you a time advantage, get tomorrow's single GPU performance today with a multi-GPU system

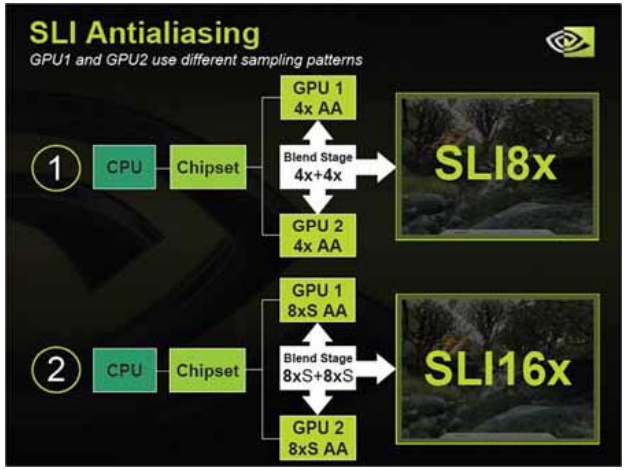
Multi-GPU: Goal

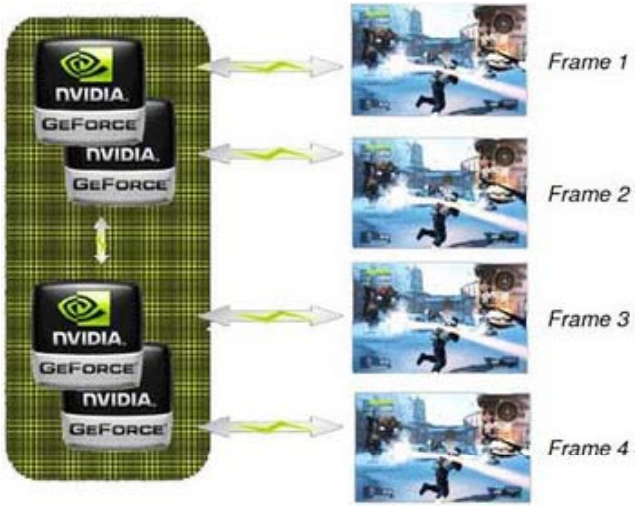
- Goal: Get linear scalability with number of GPU's in
  - Rendering performance
    - Triangle throughput
    - Fill rate
  - Data size
  - Image resolution (not topic of this presentation)

- Steps to get scalability:
  - Distribution of the workload/rendering tasks to all GPU's
  - Collection of rendering results from all GPU's
  - Assemble the rendering results in final image

Multi-GPU: Distributing the Load

- Pixel Decomposition (e.g. SLI Antialiasing)
  - Assigns different pixels or subpixels to each GPU
  - helps with fill-rate bound apps, good approach for AA
- Decomposition in time (e.g. SLI AFR)
  - Different GPUs render different frames
  - scales well for vertex-processing and fill rate bound applications



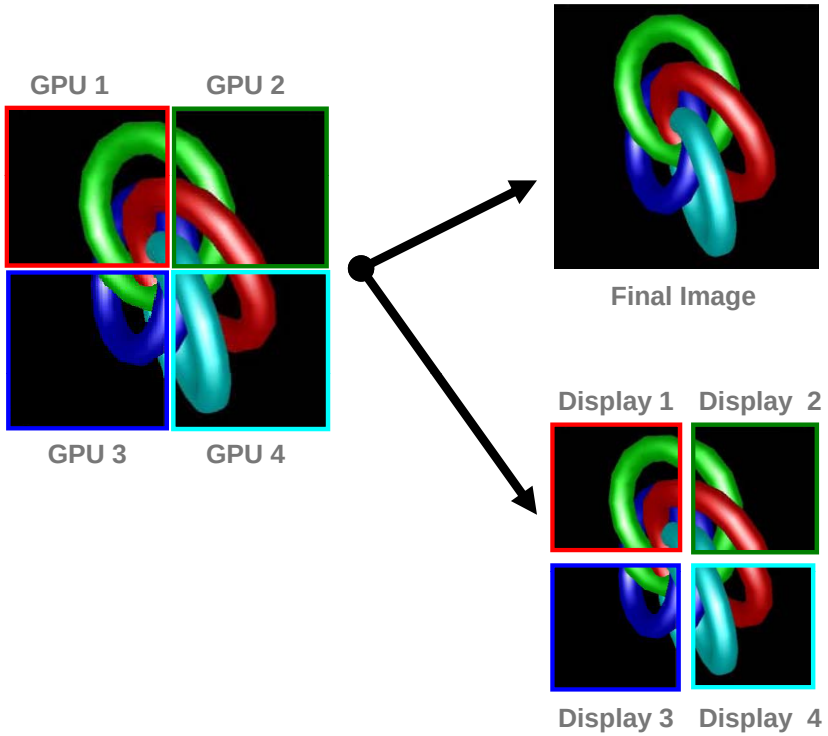


Multi-GPU: Screen Decompositing

- Sort-First or Screen Decomposition (e.g. SLI SFR or SLI Mosaic)
  - Different GPUs are rendering different portions of the screen
  - fill rate bound applications scale well
  - Good method for Displays with very high resolutions (e.g. mersive Displays or Sony 4k)

Compositing Schemes:

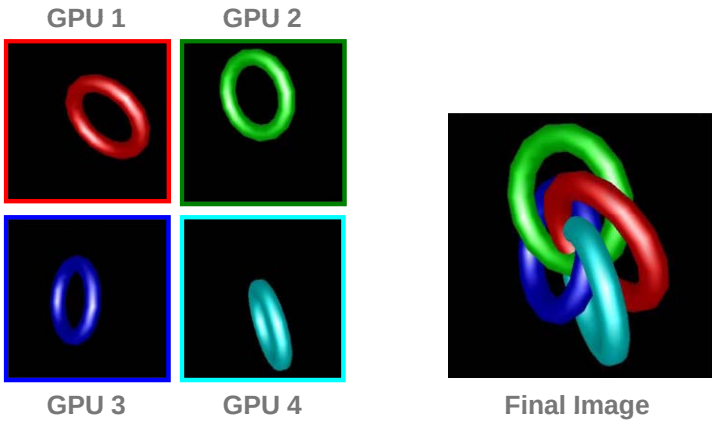
- None, use (Tiled walls or multi-input Displays)
- Simple stitching (e.g. SLI SFR)

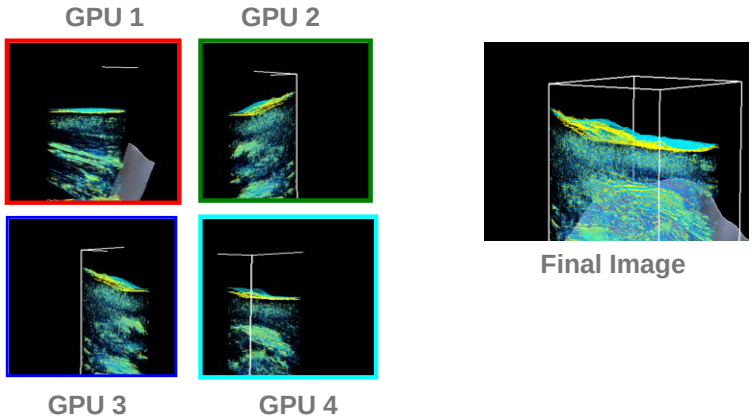


Multi-GPU: Database Decomposition

- Sort-Last or Database Decomposition
  - Different GPUs render different portions of the dataset
  - Scales well in Vertex processing, fill rate bound and graphics card memory bound applications
- Compositing Schemes
  - Depth or z-Buffer compositing (needs RGB and Depth Buffer from rendering GPU's
  - Alpha-compositing (needs only RGBA buffer)

=> Database decomposition is the method that gives us more graphics card memory





Compositing: Performance

All Decomposition Schemes require Compositing to create the final image

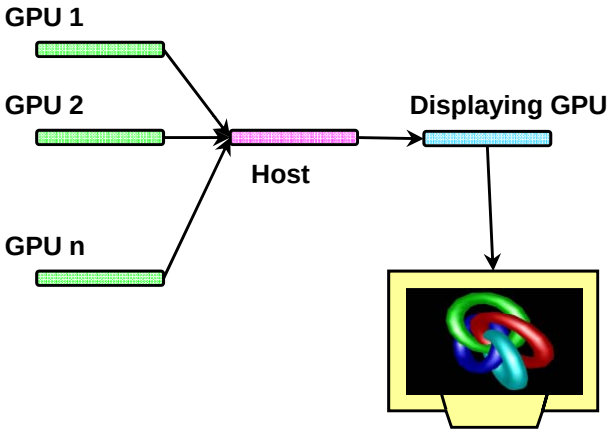
This can be done:

- with special purpose hardware
  - very fast, typically very low additional latency and performance impact
  - Tight to specific graphics hardware (can't mix/match different cards)
  - only subset of all possible composition schemes possible (e.g. SLI doesn't support z- or alpha-compositing)
  - no general purpose compositing hardware available (but several attempts in the past)
- with commodity hardware and some software
  - Adds additional latency, performance impact depends on final image size and compositing mode
  - not very graphic card dependent (mix/match possible)
  - can cover all known compositing schemes
  - New HW developments (e.g. PCIe Gen 2) reduces the impact of SW-compositing

A System Analysis of Compositing with commodity hardware

System Analysis of database decomposition with depth compositing

- Goal:
  - Understand the impact of depth compositing on multi-GPU rendering
  - Means to estimate performance benefits
  - Valid on today's HW
- Assumptions:
  - single shared memory system (no clusters)
  - only one displaying GPU (no powerwalls)
  - buffer transports go through host memory (no specific HW tricks)
- Tasks:
  - Distribute workload to rendering GPUs
  - Download resulting 2D-buffers (from GPUs framebuffer) to host memory
  - Upload 2D-buffers to displaying GPU
  - composite 2D-buffers to final image



Compositing with 2 GPUs

Start Analysis with 2 GPUs

System description:

- 2 rendering GPUs, 1 displaying GPU
- Optimization: displaying GPU is also a rendering GPU
  => reduced number of buffer transports

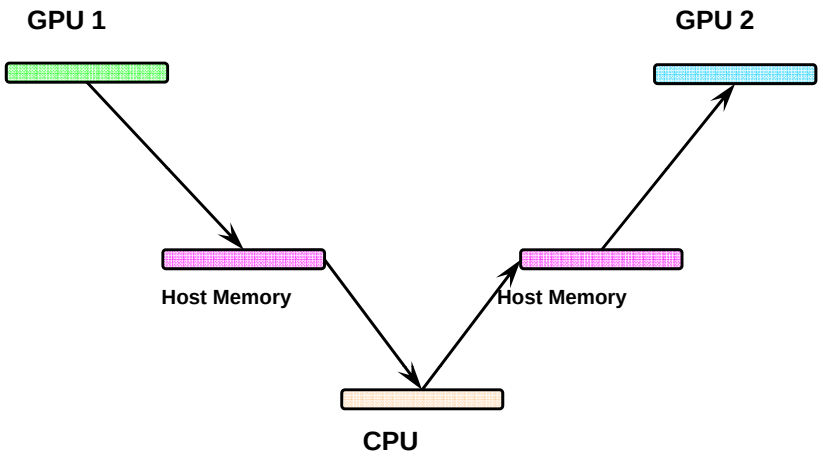
Depth compositing needs RGB and Z-buffer

E.g. 1920x1200 pixels :

$s_x = 1920$, $s_y = 1200$, $bpp = 8$ (BGRA+DEPTH_STENCIL)

- 18,432,000 Bytes per frame buffer
- 73,728,000 Bytes transported through system per frame
- @60 Hz = 4,423,680,000 > 4 GB per second !

=> High load on system Bus

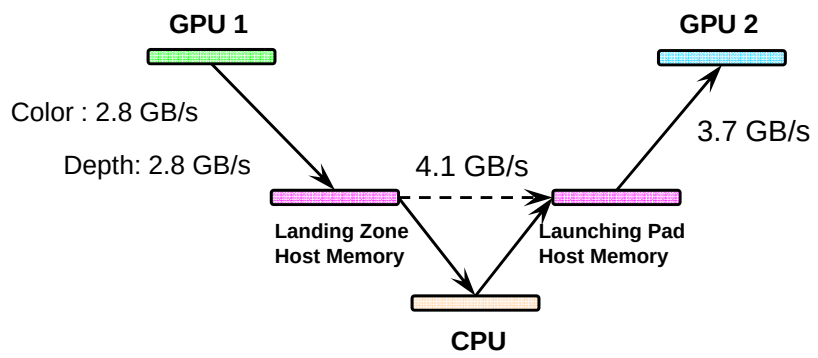


Depth Compositing Performance on 790i Ultra

System: 790i Ultra, 2 FX 3700 (PCIe 2.0)

Specs relevant for Compositing:

- Buffer downloads: 2.8 GB/s ⇔ 360 ms/GB
- host memory copy: 4.1 GB/s ⇔ 240 ms/GB
- Buffer uploads: 3.7 GB/s ⇔ 270 ms/GB
- Execution time of the actual compositing (fragment shader) is negligible



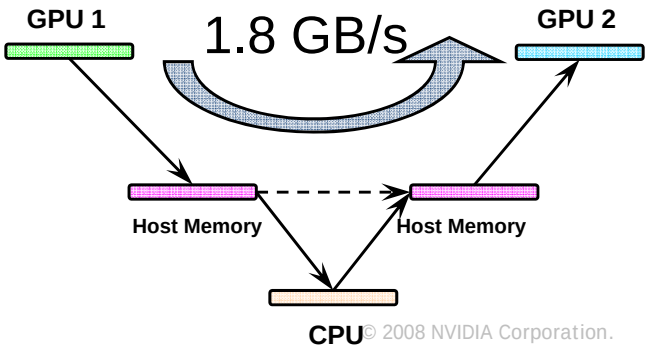
Depth Compositing Performance

Time to depth-composite 1 MPixel (worst – best case)

buffer size = $s_x \cdot s_y \cdot bpp$ = 1024·1024·8 = 8 MByte

- Compositing performance for sequential execution: 870 ms/GB ⇔ 143 fps
  worst case (no concurrency) $t_{total} = t_{dn} + t_{mc} + t_{up}$
- Compositing performance for parallel execution: 360 ms/GB ⇔ 350 fps
  best case (full concurrency) $t_{total} = max(t_{dn}, t_{mc}, t_{up})$
- Compositing performance measured: 550 ms/GB ⇔ 227 fps
  (some concurrency)

t_{dc} = Time to download BGRA buffer [s/GB]
t_{dc} = Time to download DEPTH_STENCIL buffer [s/GB]
$t_{dn} = (t_{dc} + t_{dc})/2$
t_{mc} = Time to copy from host mem to host mem [s/GB]
t_{up} = Time to upload [s/GB]

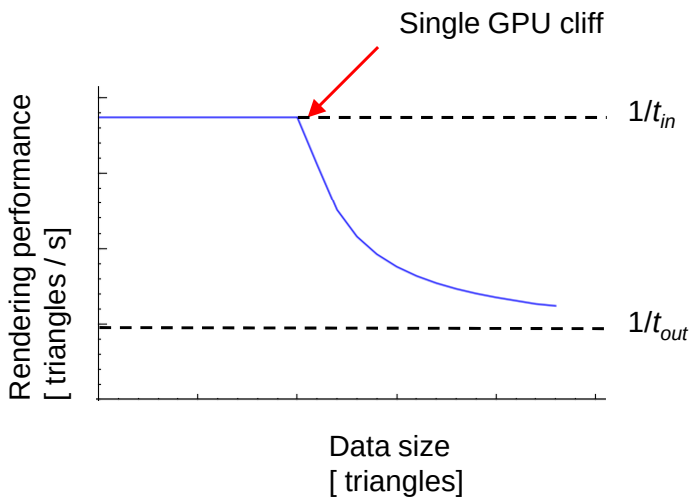


Single GPU Performance: Triangle throughput and datasize

Rendering performance measured in triangles/sec depending on data size on graphics card with limited memory

$$P(n) = \frac{n}{n_0 \cdot t_{in} + (n - n0) \cdot t_{out}}$$

$P(n)$ = Triangle performance [1/s]
n = number of triangles
n_0 = number of triangles in memory
t_{in} = rendering time 1 triangle in memory [s]
t_{out} = rendering time 1 triangle out of memory [s]



Example, Rendering performance on Quadro FX 5600:

bytes per triangle = 3 vertices(36 bytes) + 3 normals(36 bytes) = 72 bytes / triangle

NVSG-test program renders individual triangles in VBO's:

- p_{in} = 143 Mtriangles/s for in-memory data => t_{in} = 7 ns
- p_{out} = 25 Mtriangles / s for out-of memory data => t_{out} = 40 ns
- n_0 = 20 Mtriangles

Multi GPU Rendering: 2 GPUs

- **Compositing impact is known**
- **Rendering performance on single GPU is known**

Let's estimate what performance gain we can expect over a growing dataset

Assumptions:

- Execution of rendering and compositing happens sequentially (will be changed in the future):
- Rendering happens concurrently w/o overhead on both GPU's
- Fixed buffer size = 1k x 1k = 1Mpixel => $t_{compositing}$ = 4.4 ms

single GPU total time per frame:

- $t_{total,1\ GPU} = t_{render}(n) = n_0 \cdot t_{in} + (n-n_0) \cdot t_{out}$

dual GPU total time per frame:

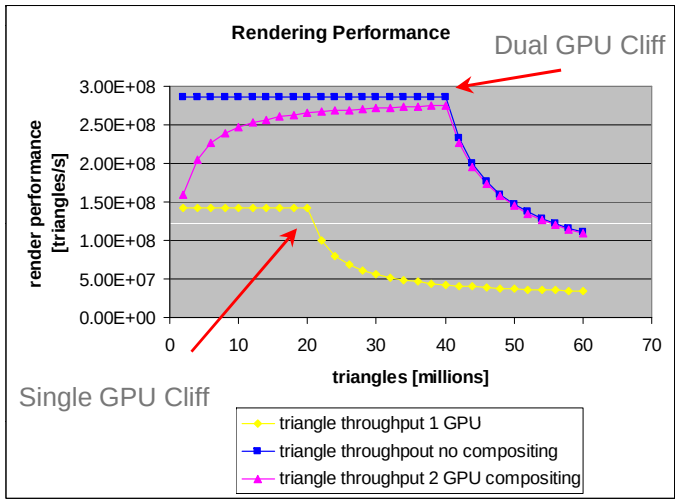
- $t_{total,\ 2\ GPU} = t_{render}(n/2) + t_{compositing}$

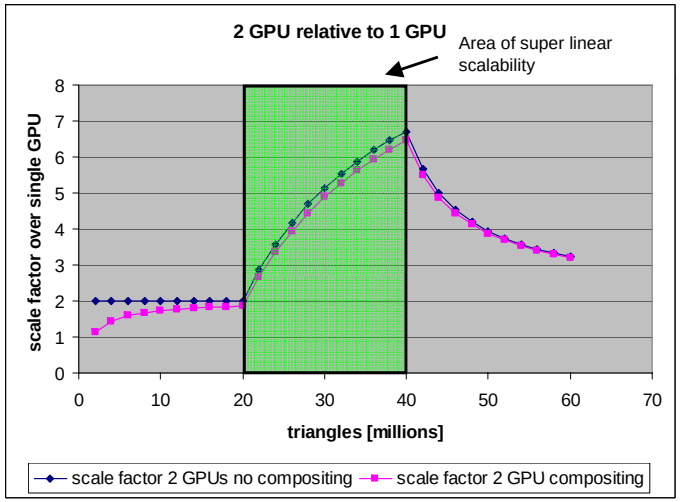
⇒ **performance gain** $s_{2GPU} = t_{total,1\ GPU} / t_{total,2\ GPU}$

peak performance to be expected at $n = 2\ n_0$:
$s(n=2 \cdot n_0) = (n_0 \cdot t_{in} + n_0 \cdot t_{out}) / (n_0 \cdot t_{in} + t_c) \Leftrightarrow$
$(t_{in} + t_{out}) / (t_{in} + t_c / n_0)$
improve peak performance by:

- reduce t_c (e.g. optimize buffer transports)
- increase n_0 (e.g. pack more triangles in card like tristrips instead individual triangles)

- Theoretical peak max $s_{max} = (t_{in} + t_{out}) / t_{in}$
- applied to FX5600 s_{max} = (7ns +40ns)/7ns = 6.7





Multi-GPU Rendering: 4 GPUs

Generalize 2-GPU formula to k GPUs:

Assumption:
buffer transports time grows linearly with number of GPUs

k GPUs total time per frame:

- $t_{total,\ k\ GPU} = t_{render}(n/k) + (k\text{-}1)\cdot t_{compositing}$

$\Rightarrow$ performance gain $s_{kGPU} = t_{total,1\ GPU} \ / \ t_{total,k\ GPU}$

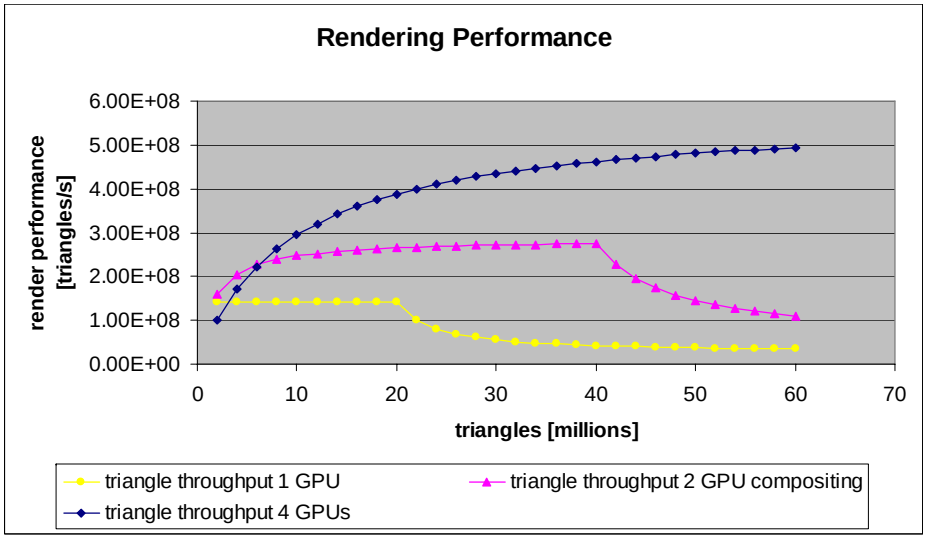
peak performance to be expected at $n = k\ n_0$:
$s(n{=}k\cdot n_0) = (n_0\cdot t_{in} + (k\text{-}1)\cdot n_0\cdot t_{out})/(n_0\cdot t_{in}+(k\text{-}1)\cdot t_c) \Leftrightarrow$

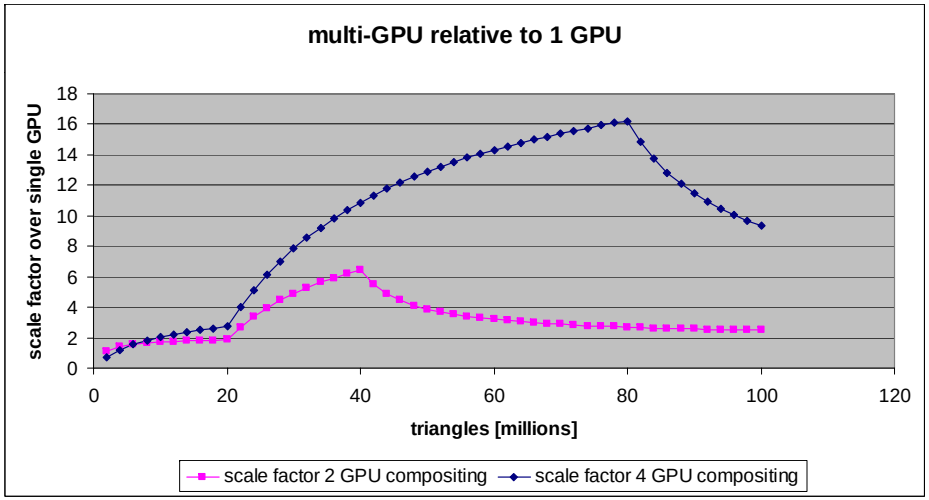
$s_{max} = (t_{in} + (k\text{-}1)\cdot t_{out})/(t_{in}+(k\text{-}1)\cdot t_c/n_0)$

- Theoretical peak max $s_{max} = (t_{in} + (k\text{-}1)\cdot t_{out})/t_{in}$
  $= 1 + (k\text{-}1)\cdot t_{out}/t_{in}$

E.g. FX 5600

- k = 4 FX 5600 $s_{max} = (7ns + 3\cdot 40ns)/7ns = 18.1$
- k = 8 FX 5600 $s_{max} = (7ns + 7\cdot 40ns)/7ns = 41$





Frame Rates

Application classes by frame rate

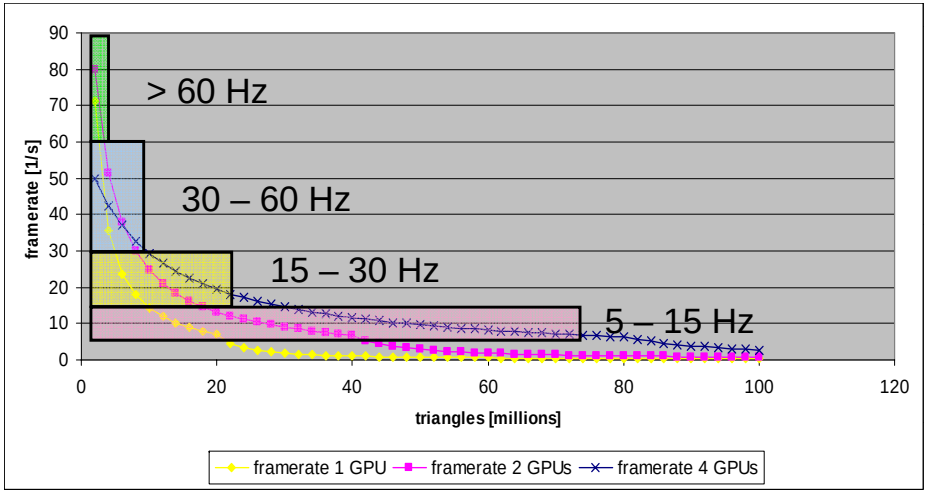
>= 60 Hz, Visual Simulation (e.g. professional flight simulators)

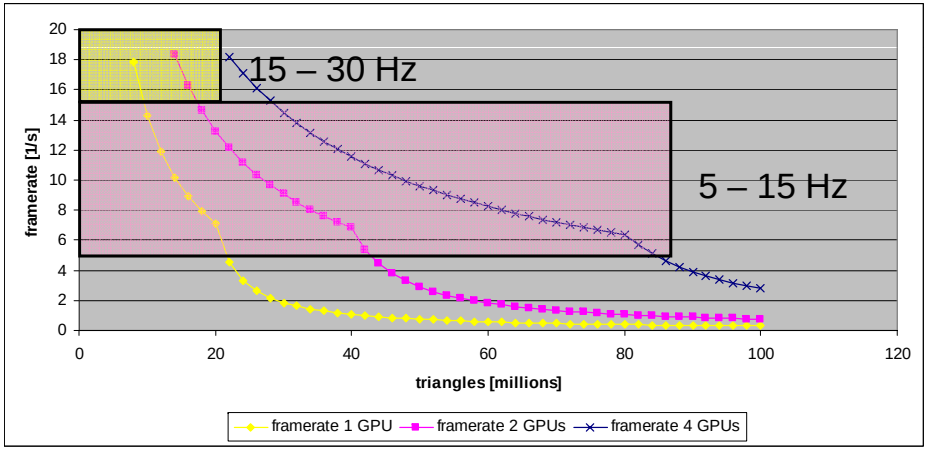
>= 30 Hz, < 60 Hz, Entertainment (gaming)

>= 15 Hz, < 30 Hz VR, Design Review

>= 5 Hz, < 15 Hz Modeling, Large CAD Data Visualization, Seismic Interpretation,…

Best scalability for Large Models (low impact by compositor)
=> very good fit for Large Data visualization like Seismic interpretation





Results System Analysis

System Analysis shows:

- Performance scales with number of GPUs

=> Higher framerate

- Graphics card memory scales linearly with number of GPUs

=> Larger models

- Small models don't scale very well

=> Multi-GPU rendering (database decomposition) not useful for small models

- Peak scalability far beyond linear (Larger Cache effect)

Multi-GPU rendering for Large models works !

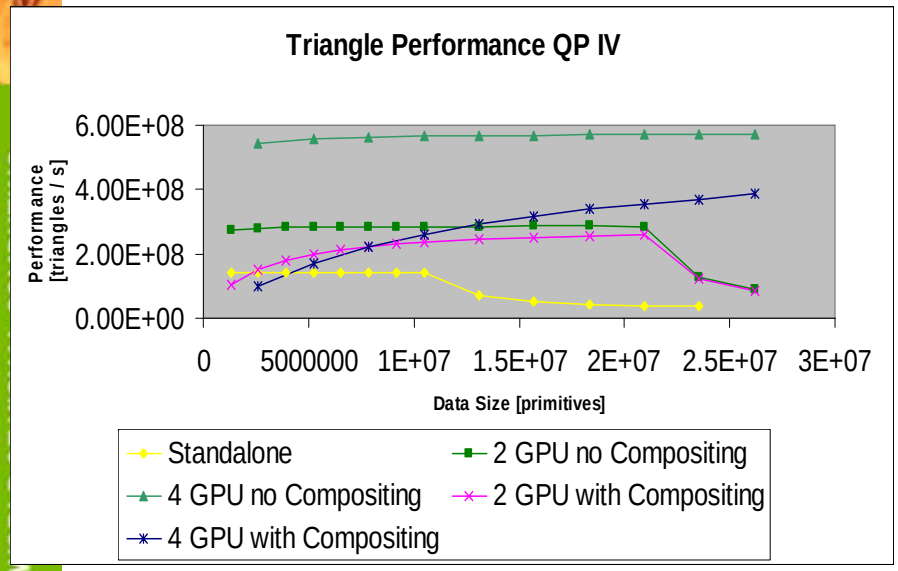
Results Multi GPU rendering: OSG

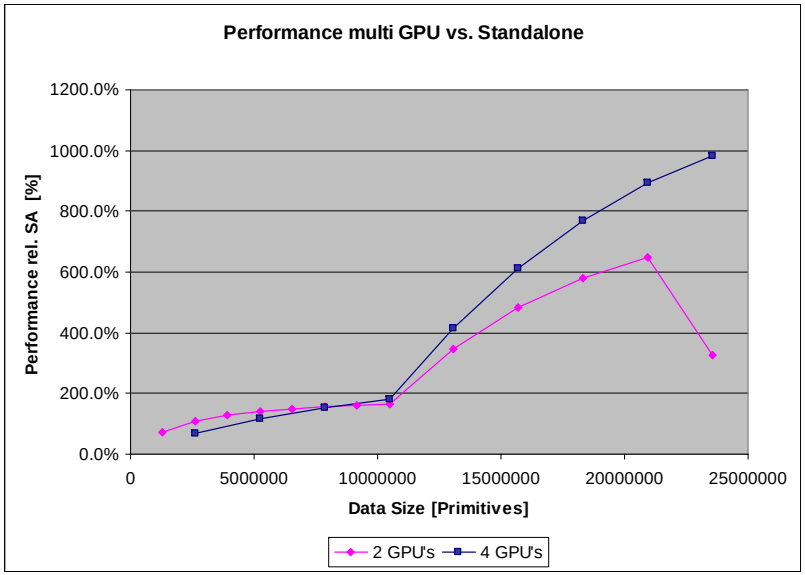
OSG = Open Scene Graph

Open Source Scene Graph widely used in Academia, Simulation, Oil&Gas and other industries

We extended OSG to render in multiple threads (one thread per GPU) and added the MGPU SDK

In this test OSG used around 140 bytes per triangle



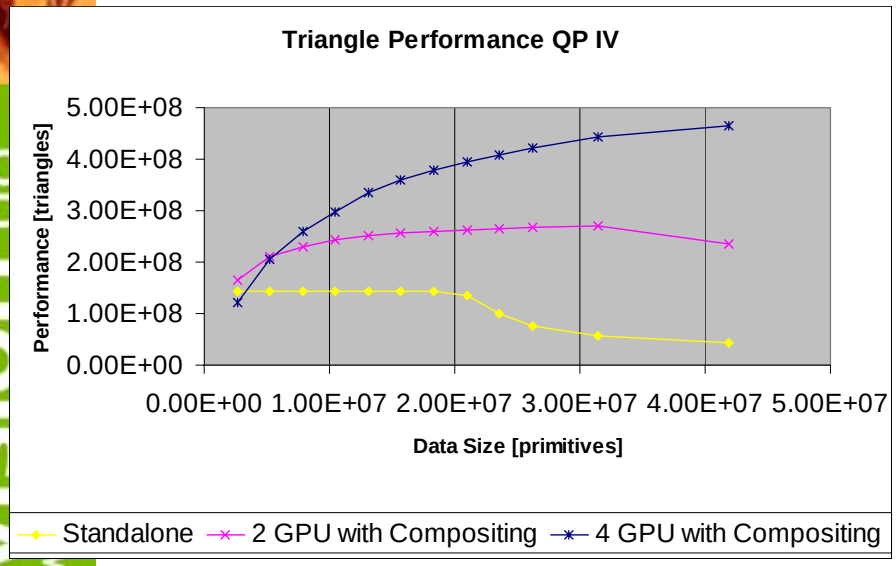


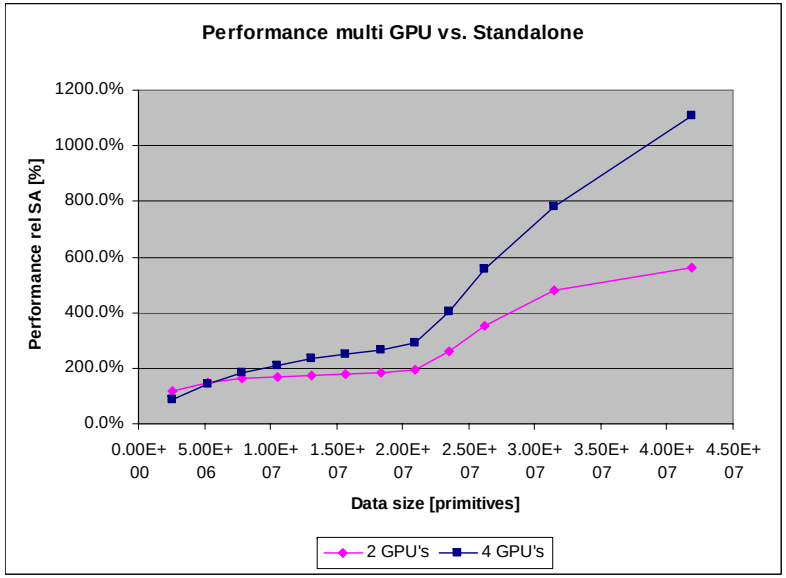
Results Multi GPU rendering: NVSG

NVSG = NVIDIA's Scene Graph

Very efficient Scene Graph used in automotive, simulation and academia

NVSG used 72 bytes per triangle





NVISION 08
THE WORLD OF VISUAL COMPUTING

MGPU – SDK Introduction

Thomas Volk

NVIDIA.

Agenda

- Scope of the SDK
- SDK features
- Programming Guide for the SDK

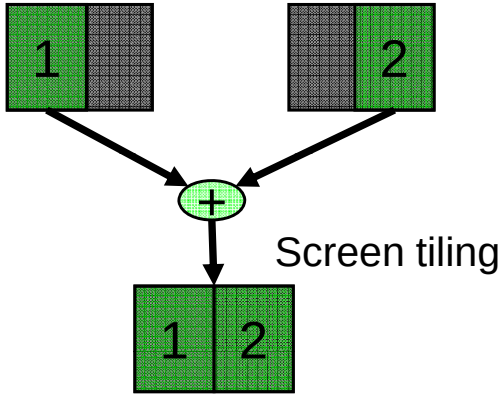
Scope of the MGPU SDK

- Multi-GPU set-up and addressing
  - Affinity context/screen per GPU
  - SDK utility class for context handling and off-screen rendering
- Application parallelism
  - Multi-threading of rendering loop: already solved in lots of scene graph implementations
  - Multi-process and out of core/cluster solutions: highly specialized solutions already available
- Load decomposition and balancing: needs to be addressed in SG for sort last (see NVSGScale)
  - Sample/reference implementations
  - Utility classes
- Image compositing with huge data transfer (5GB/s) and low latency: SDK compositor

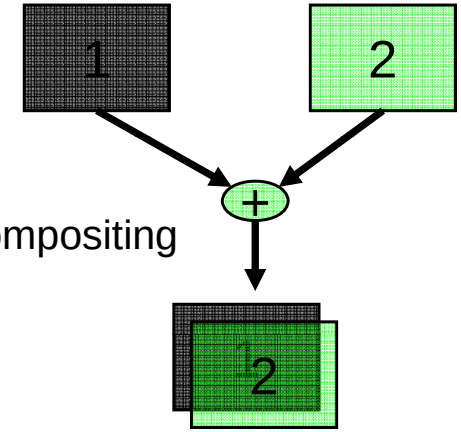
SDK features

- OpenGL based
- Professional applications
- QuadroPlex only
  - With current systems up to 2-8 GPUs, e.g. 2xD2s, 2xD4s
  - Up to 16 GB of addressable video memory
- Platforms: win32/64, linux 64
- In comparison to SLI non-transparent to application
- Utility functions to create MGPU aware GL contexts and drawables
  - Platform independent
  - Stereo
  - AA
- Image compositor for sort first and sort last based applications
  - Easy to use abstract interface for compositing
  - Compositor implementation based on latest technologies, no migration effort for applications (next gen hardware will provide faster transport)
  - Multi-threaded, shared memory
  - Configurable: 1-1, n-1, hierarchical, hybrid
  - Screen tiling, alpha and depth based compositing

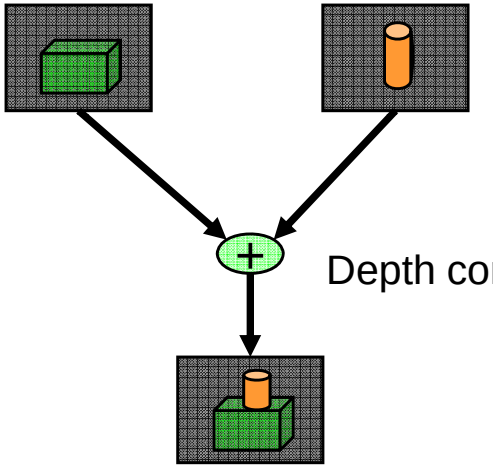
Compositor Types and Configurations



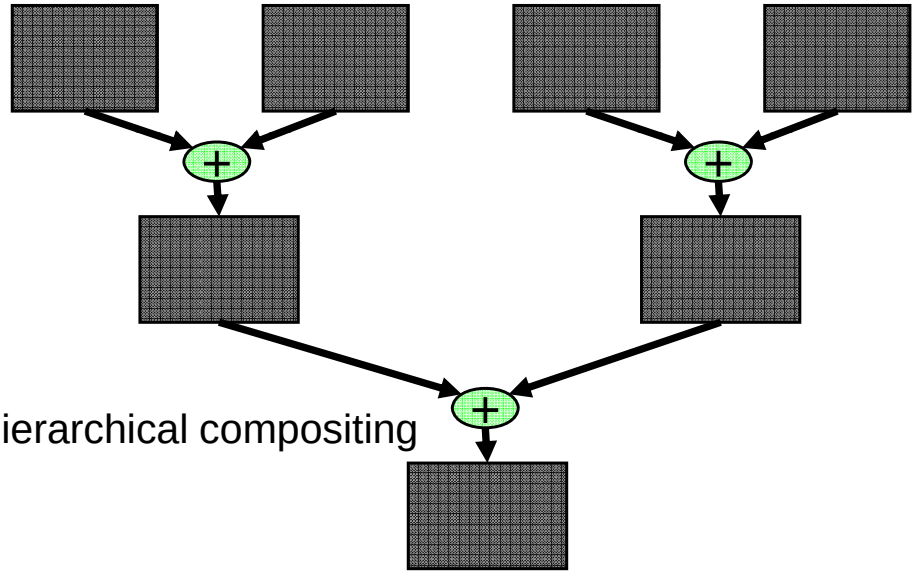
Screen tiling



Alpha compositing

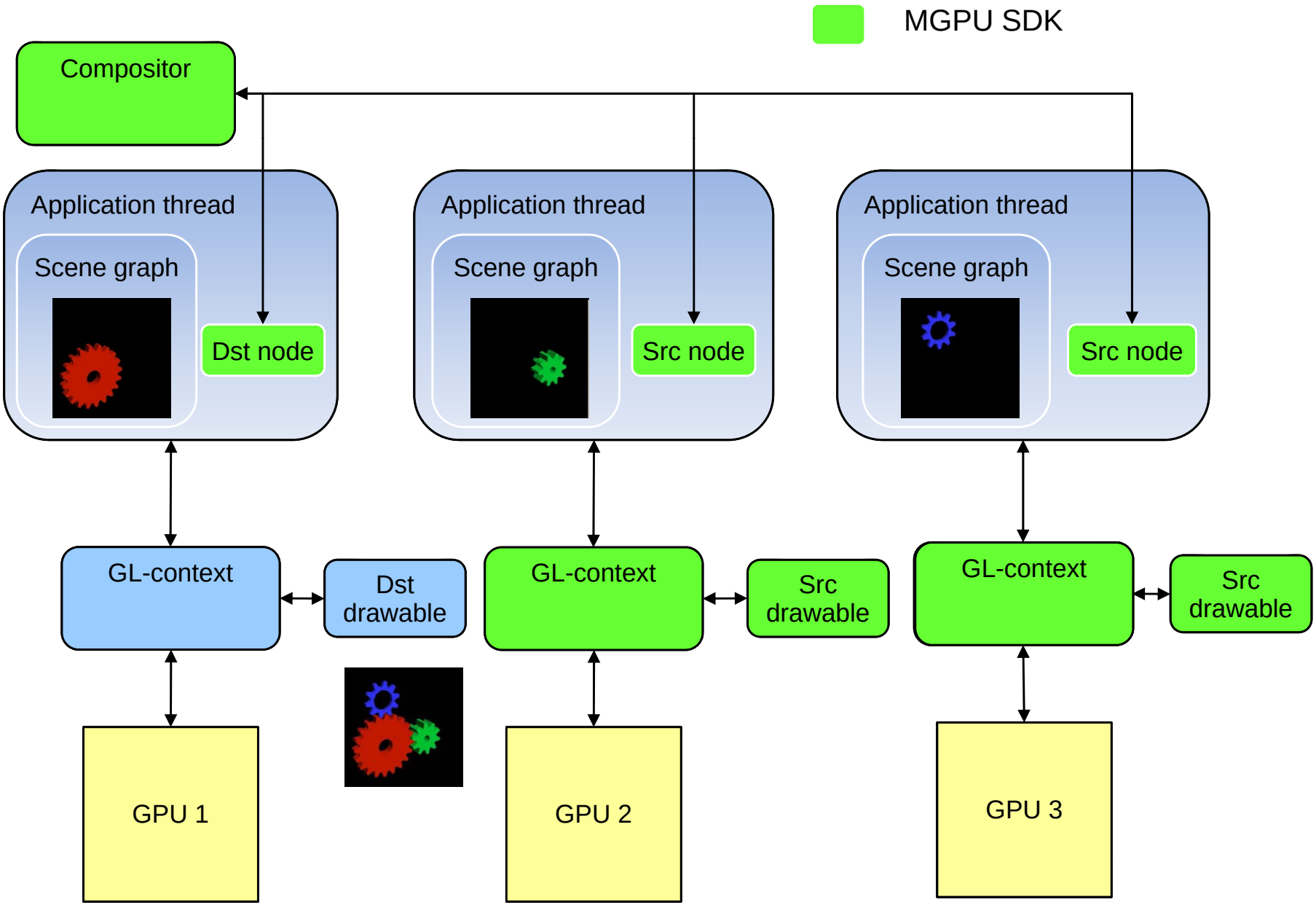


Depth compositing

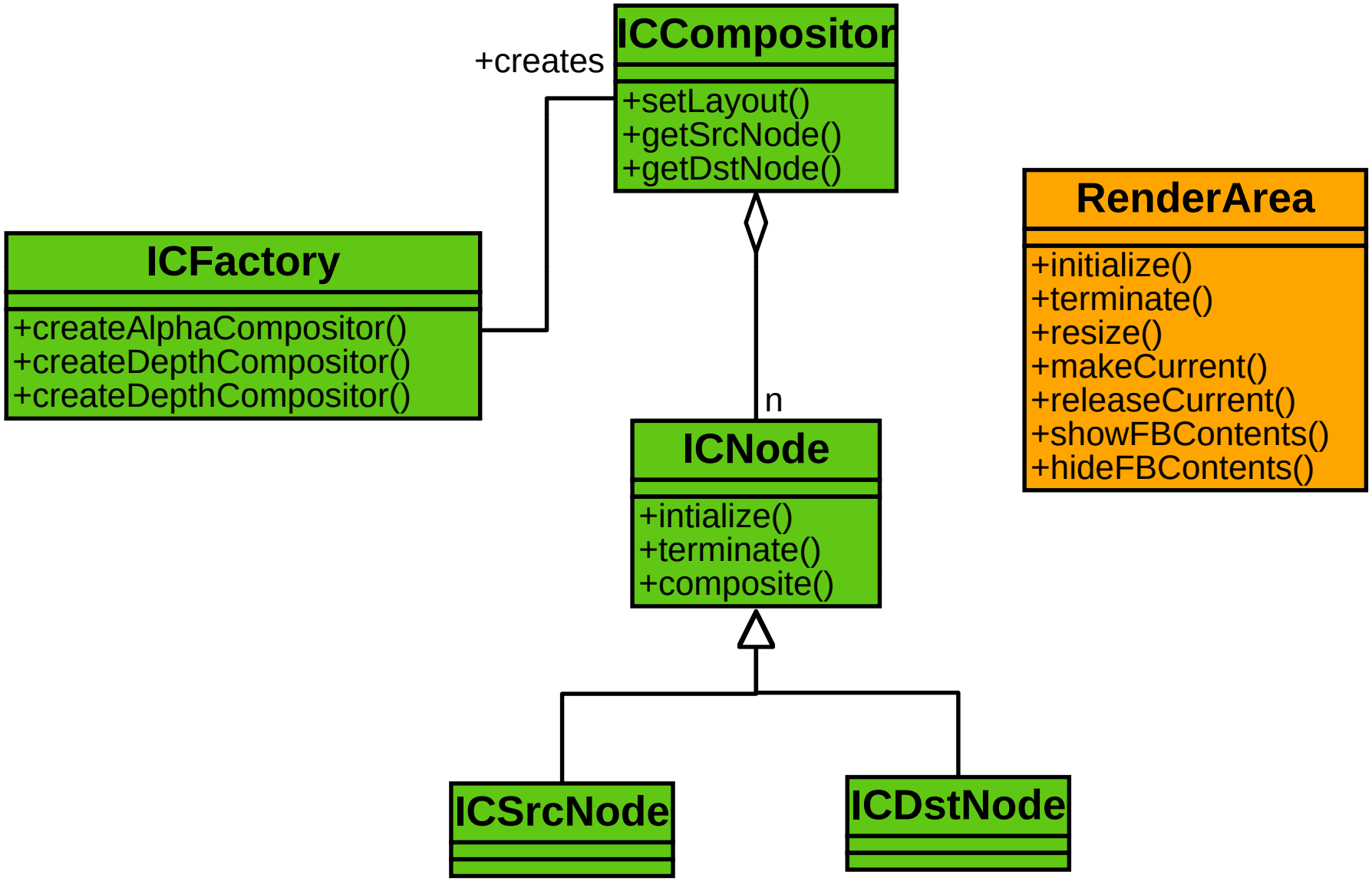


Hierarchical compositing

System Overview



SDK Classes



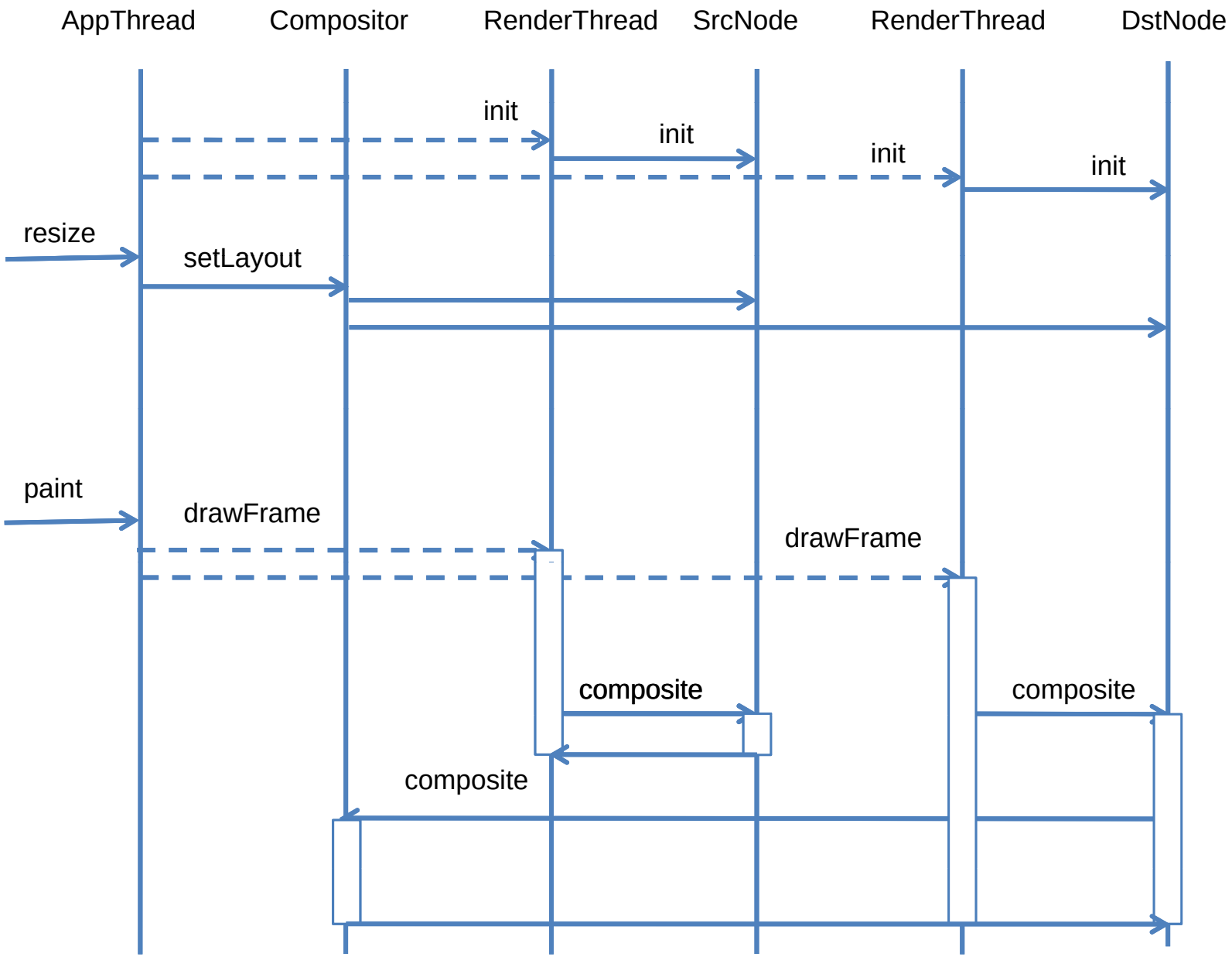
Code Example

```
//init in control/main thread
ICDepthCompositor* c = CompositorFactory::getInstance()->getDepthCompositor();
c->setLayout(IC_ALLNODES, IC_RECT(width, height));
ICNode* myCompositorNode[MAX_NODES];
myCompositorNode[0] = c->getSrcNode(idx);
```

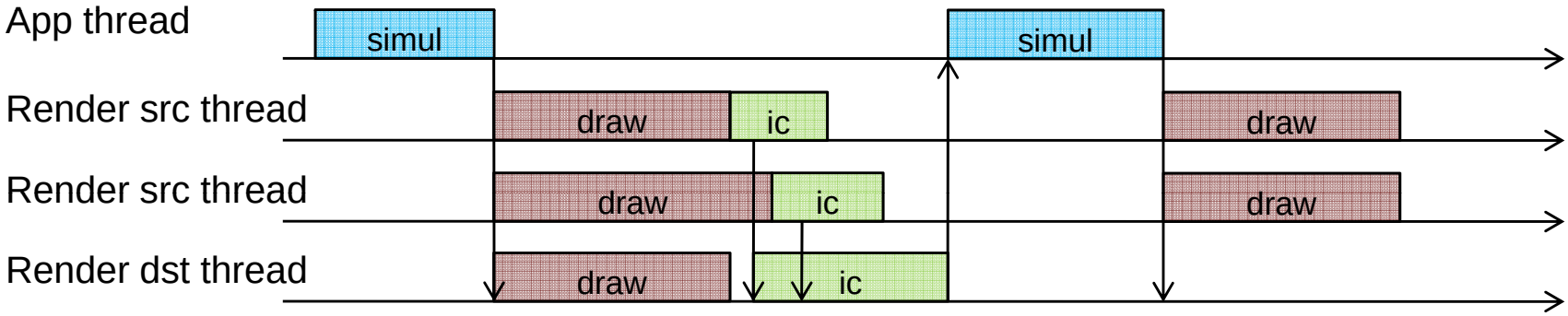
```
//init in every thread and context
//create affinity context and drawable;
RenderArea ra;
ra->initialize();
ra->makeCurrent();

//initialize GL resources of compositor
myCompositornode[i]->initialize();
```

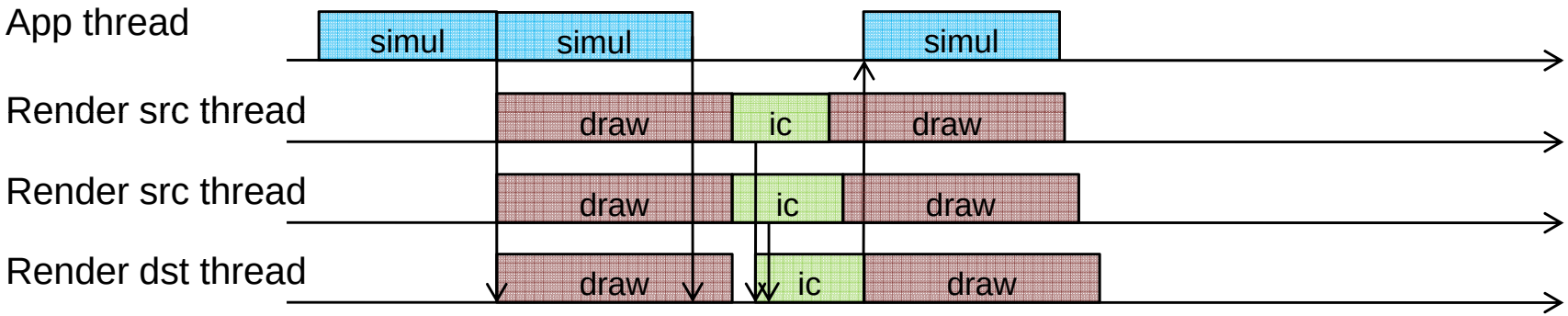
```
//in every render-thread
void drawFrame(int myID)
{
    drawPartialScene(myID);

    //trigger image composition
    myCompositornode[myID]->composite();
}
```

Sample Sequence Diagram



MGPU Threading





Wrap up

- Current status
  - Beta release next week
  - No stereo and AA, linux only for beta
  - Freely available
  - Release mid Oct
- Future features
  - Better hardware support
  - Access to frame-buffers
  - Utility classes for load balancing
  - Performance feedback
  - Extensible shader compositor
  - Additional color formats

NVISION 08
THE WORLD OF VISUAL COMPUTING

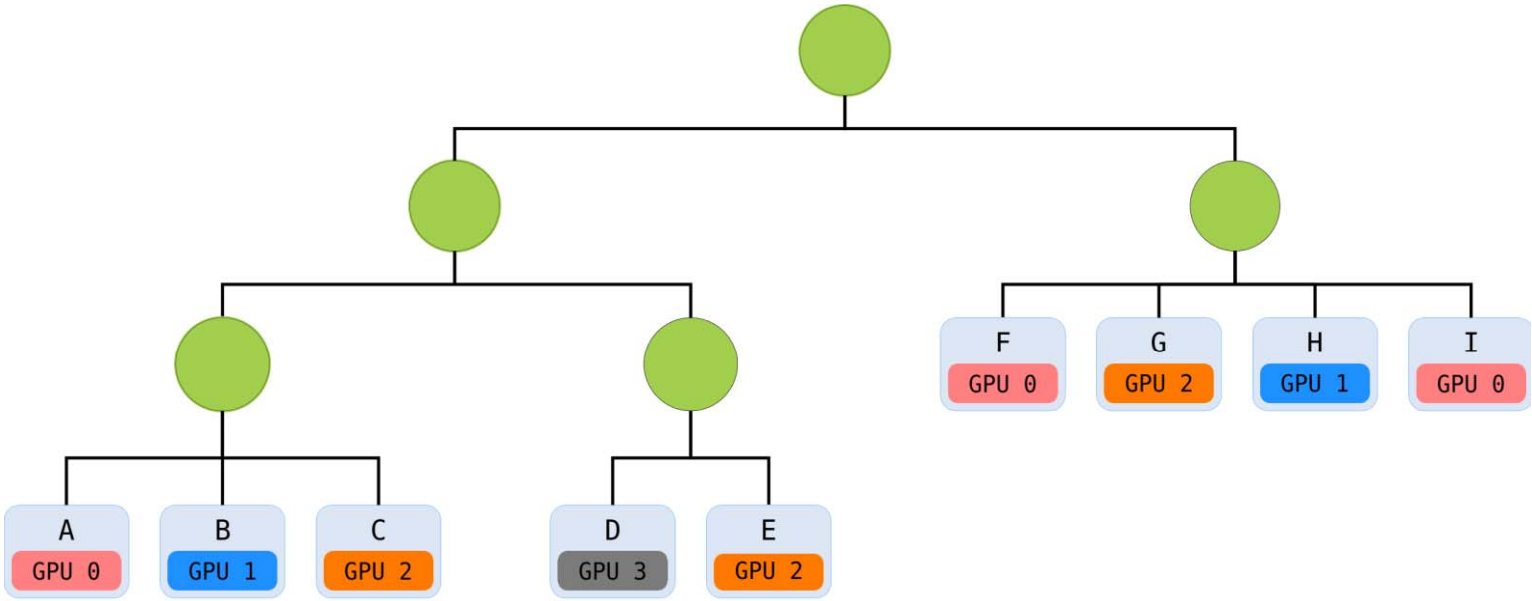
Multi-GPU Rendering with NVIDIA Scene Graph

The World's Fastest Scene Graph On Steroids

Subu Krishnamoorthy

NVIDIA.

Distributing Scene Objects



- Audience – specifies which GPUs are responsible for processing a scene graph object
- DistributionTraverser – assigns Audiences to balance load on each GPU
- GLDistributedTraverser – respects Audiences and prunes traversal when its GPU is excluded

Distribution Scheme

- Opaque objects

  Least loaded GPU included, others excluded

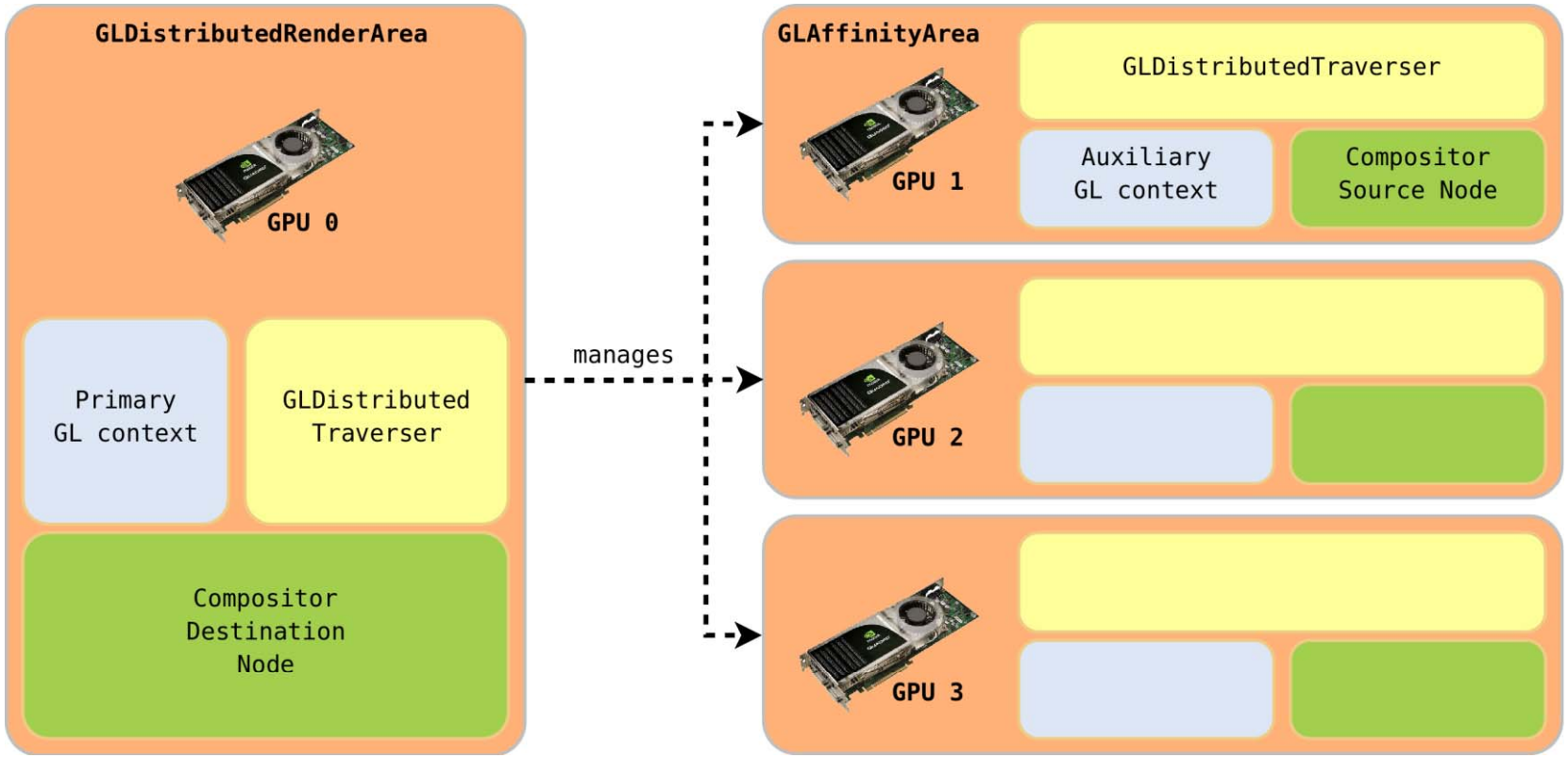
- Translucent and overlay objects

  Primary GPU included, others excluded

- Implies use of Depth Compositor

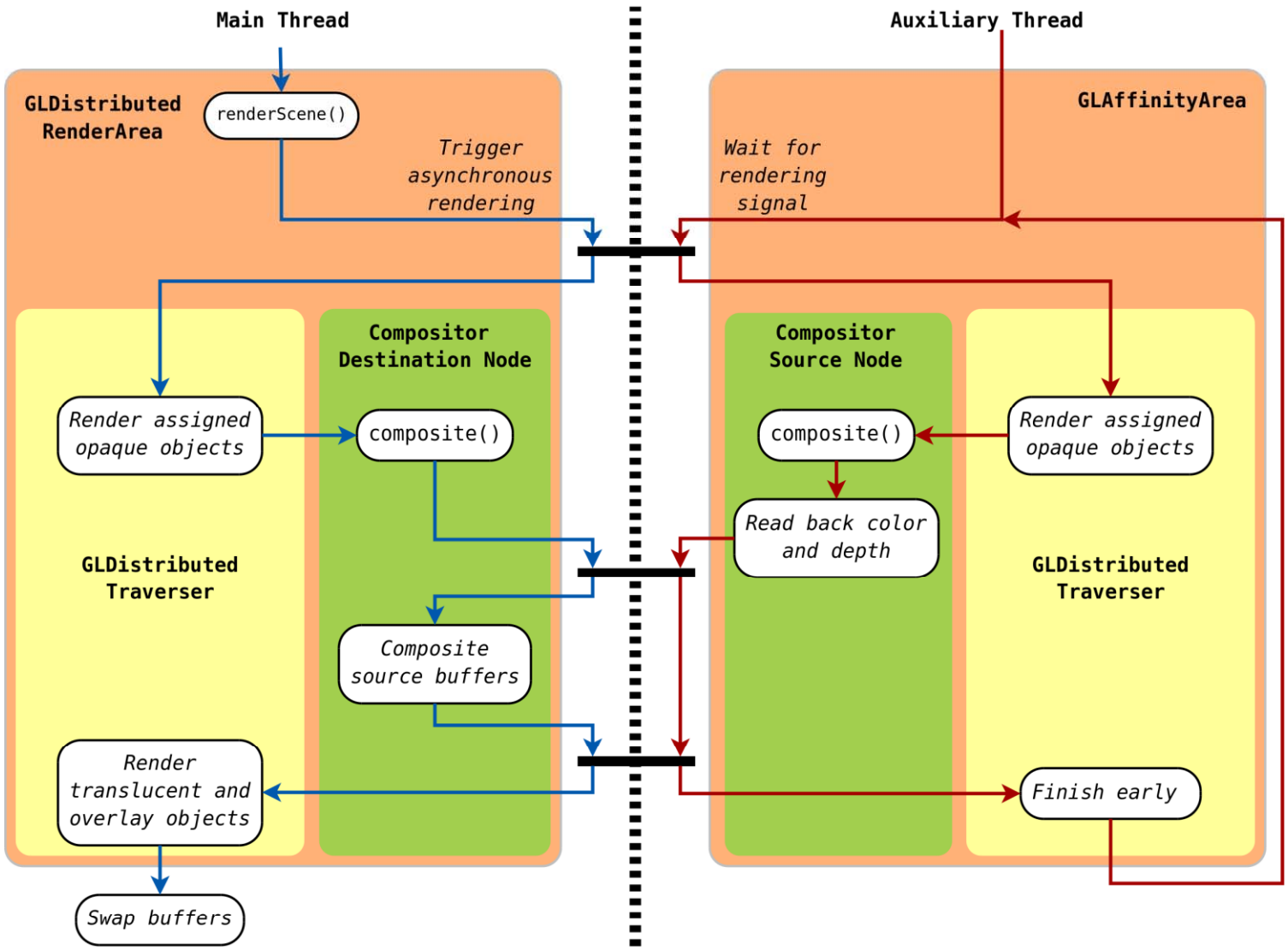
  Image composited before translucent and overlay objects are drawn

The Components



- GLAffinityArea creates a thread that drives an associated GLDistributedTraverser
- GLDistributedRenderArea manages the collection of N-1 GLAffinityAreas

Parallel Rendering



Ease of Use

- Derive client RenderArea from GLDistributedRenderArea

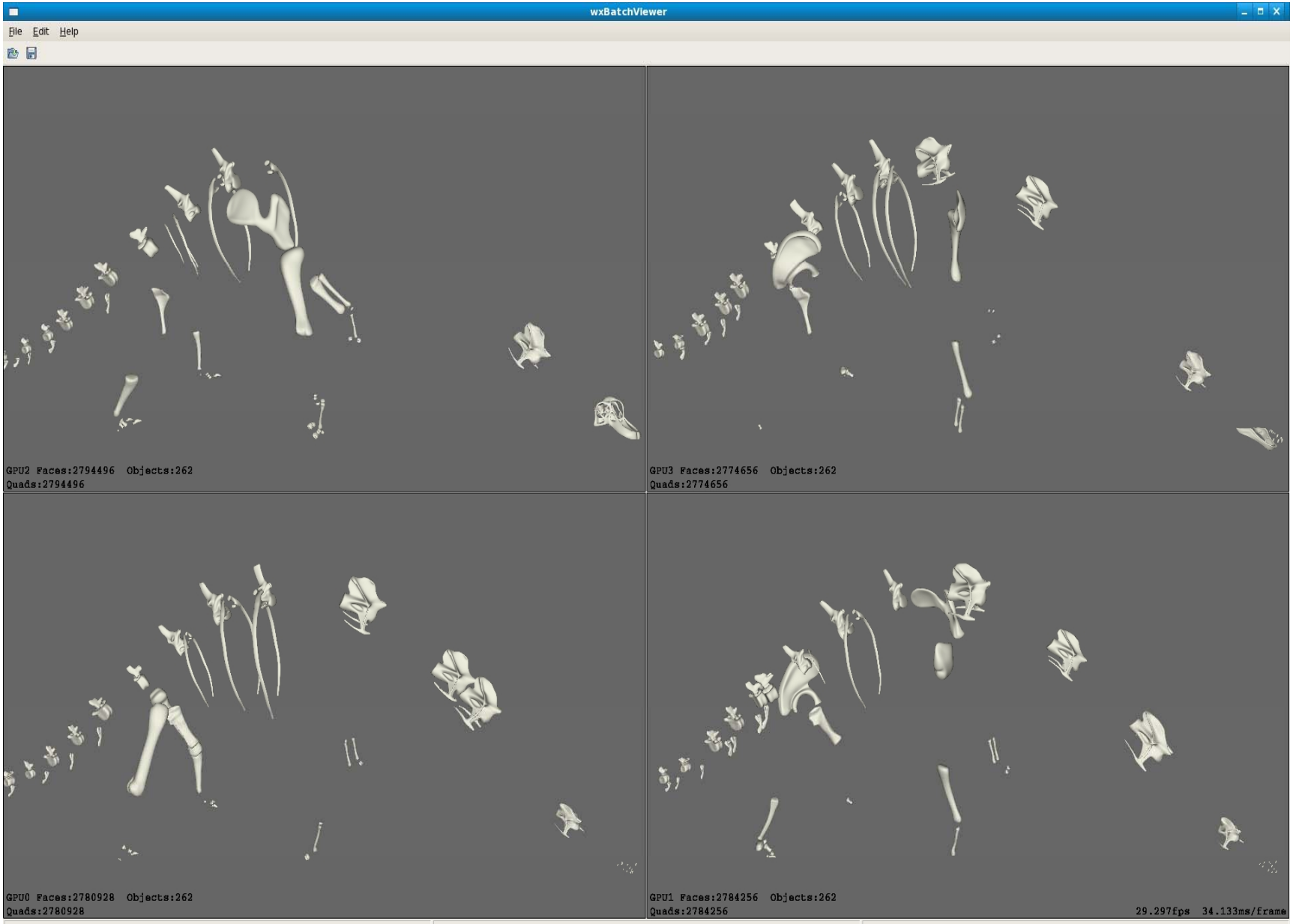
  (instead of nvui::RenderArea)

```
class wxGLDistributedRenderArea : public wxGLCanvas,
                                  public nvui::GLDistributedRenderArea
{
  ...
};
```

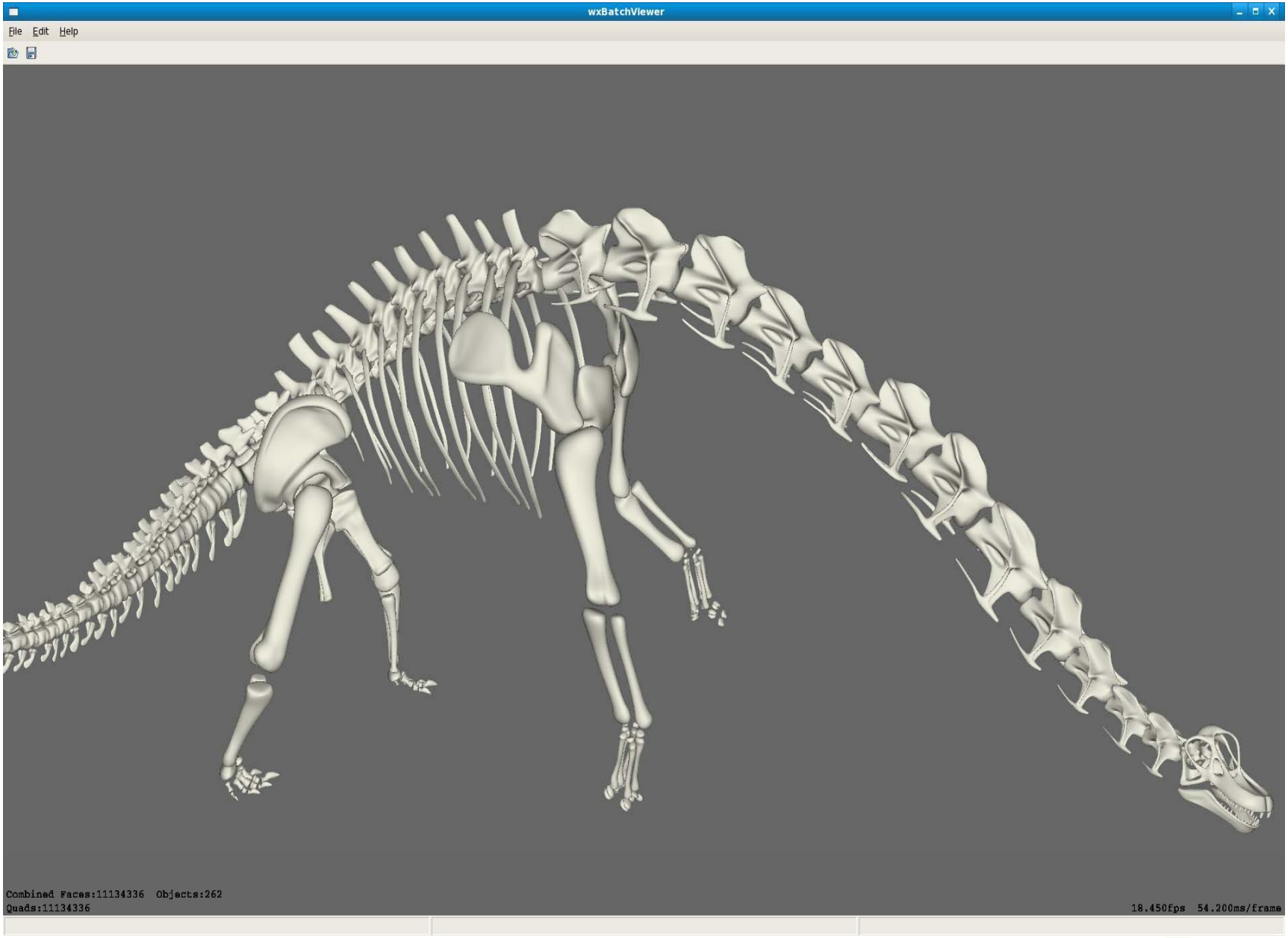
- Override GLDistributedRenderArea::init()
  - Create Primary GL context (affinity context) and make it current
  - Invoke base implementation

```
bool wxGLDistributedRenderArea::init(nvui::RenderArea *shareArea)
{
  ...
  m_glContext = new wxGLAffinityContext(this, shareArea);
  wxGLCanvas::SetCurrent(*m_glContext);
  ...
  GLDistributedRenderArea::init(shareArea);
  ...
}
```

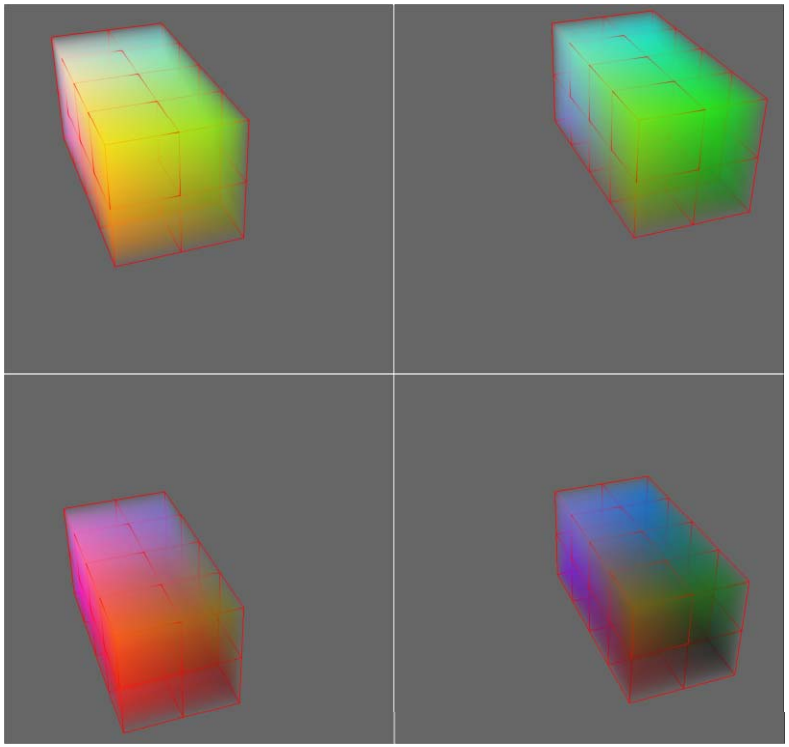
Opaque Object Distribution

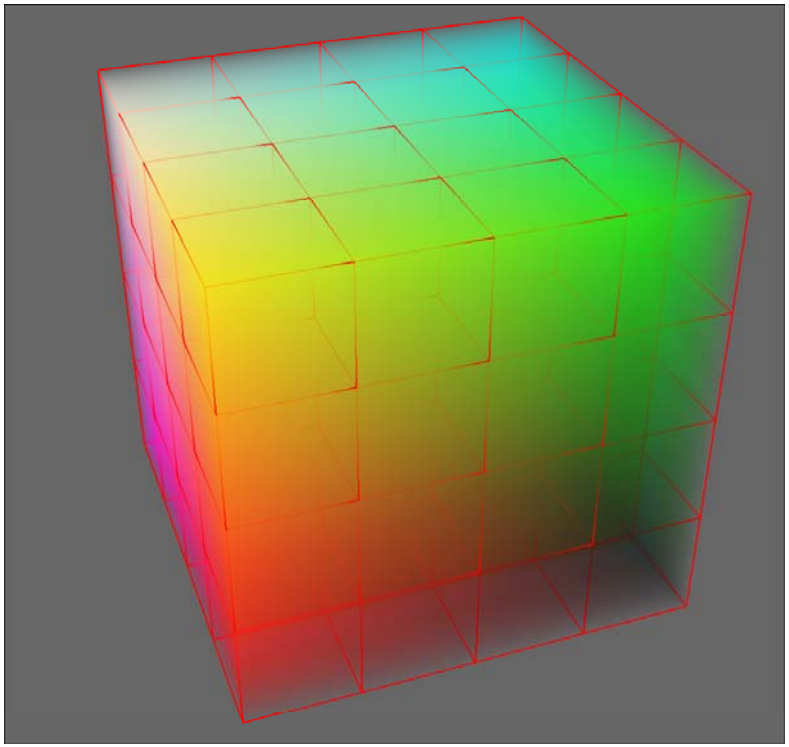


Depth Composited Image



Spatial Distribution





- Audience assigned based object bounding box
- Implies use of Alpha Compositor

Summary

- Powerful new feature of NVSG
- Visualize large models at interactive frame rates

NVISION 08
THE WORLD OF VISUAL COMPUTING

The Industrial GPU Revolution

Visual Computing In The Oil Sector, Horn Thorolf Tonjum

NVIDIA.

Demos

Dinosaurs in the Museum

Model characteristics:

- Model designed in Maya, AutoDesk
- 217 Million Triangles

Rendering Hardware:

- HP xw8600 with 32 GB System Memory
- 2xD2's (4xGT200 with 4 GB FB memory) with 16 GB total FB memory

Rendering Performance:

1 GPU = 0.7 fps

2 GPU's = 3 ½ fps = 5 times faster than 1 GPU

4 GPU's = 6 fps = 8 ½ times faster than 1 GPU

=> Total max. Triangle throughput = 1.3 GTri/s

Multi-GPU rendering of Kristin.

Model characteristics:

- Kristin is an off-shore platform in the North Sea
- Model designed in MicroStation, Bentley
- 230 Million Triangles (individual triangles, no triangle strips)

Rendering Hardware:

- HP xw8600 with 32 GB System Memory
- 2xD2's (4xGT200 with 4 GB FB memory) with 16 GB total FB memory

Rendering Performance:

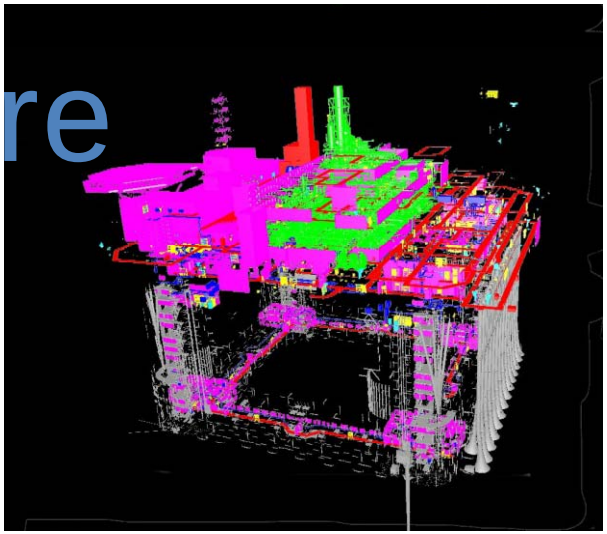
1 GPU = 0.4 fps

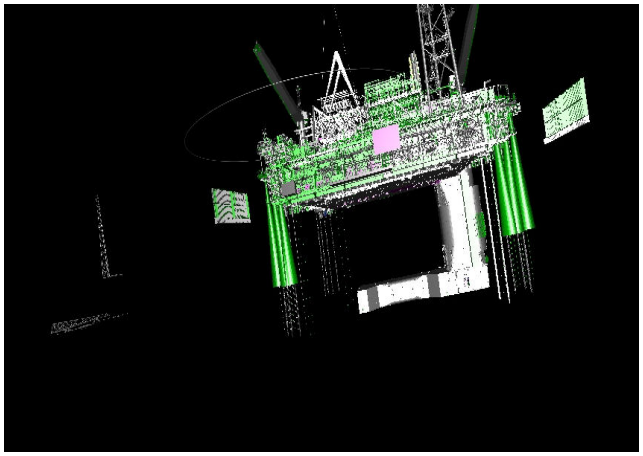
2 GPU's = 2 ½ fps = 6 times faster than 1 GPU

4 GPU's = 4 ½ fps = 11 times faster than 1 GPU

=> Total max. Triangle throughput = 1 GTri/s

With the friendly permission of stormfjord





Summary

- System analysis of multi-GPU rendering
- How to do multi-GPU rendering with NVIDIA's Multi-GPU SDK
- Ease of use of multi-GPU rendering with NVSG
- Large CAD data visualization in the Oil&Gas industry

=> Database decomposition together with depth compositing is a strong tool to give you tomorrows graphics performance today

The End

Questions ?