Hannes Kaufmann

3D Graphics Hardware

Hannes Kaufmann

Interactive Media Systems Group (IMS)

Institute of Software Technology and Interactive Systems

Thanks to Dieter Schmalstieg and Michael Wimmer for providing slides/images/diagrams

Hannes
Kaufmann

Motivation

- VR/AR environment = Hardware setup + VR Software Framework + Application
- Detailled knowledge is needed about
  - Hardware: Input Devices & Tracking, Output Devices, 3D Graphics
  - Software: Standards, Toolkits, VR frameworks
  - Human Factors: Usability, Evaluations, Psychological Factors (Perception,...)

Hannes Kaufmann

3D Graphics Hardware -Development

- Incredible development boost of consumer cards in previous ~20 years
- Development driven by game industry
- PC graphics surpassed workstations (~2001)

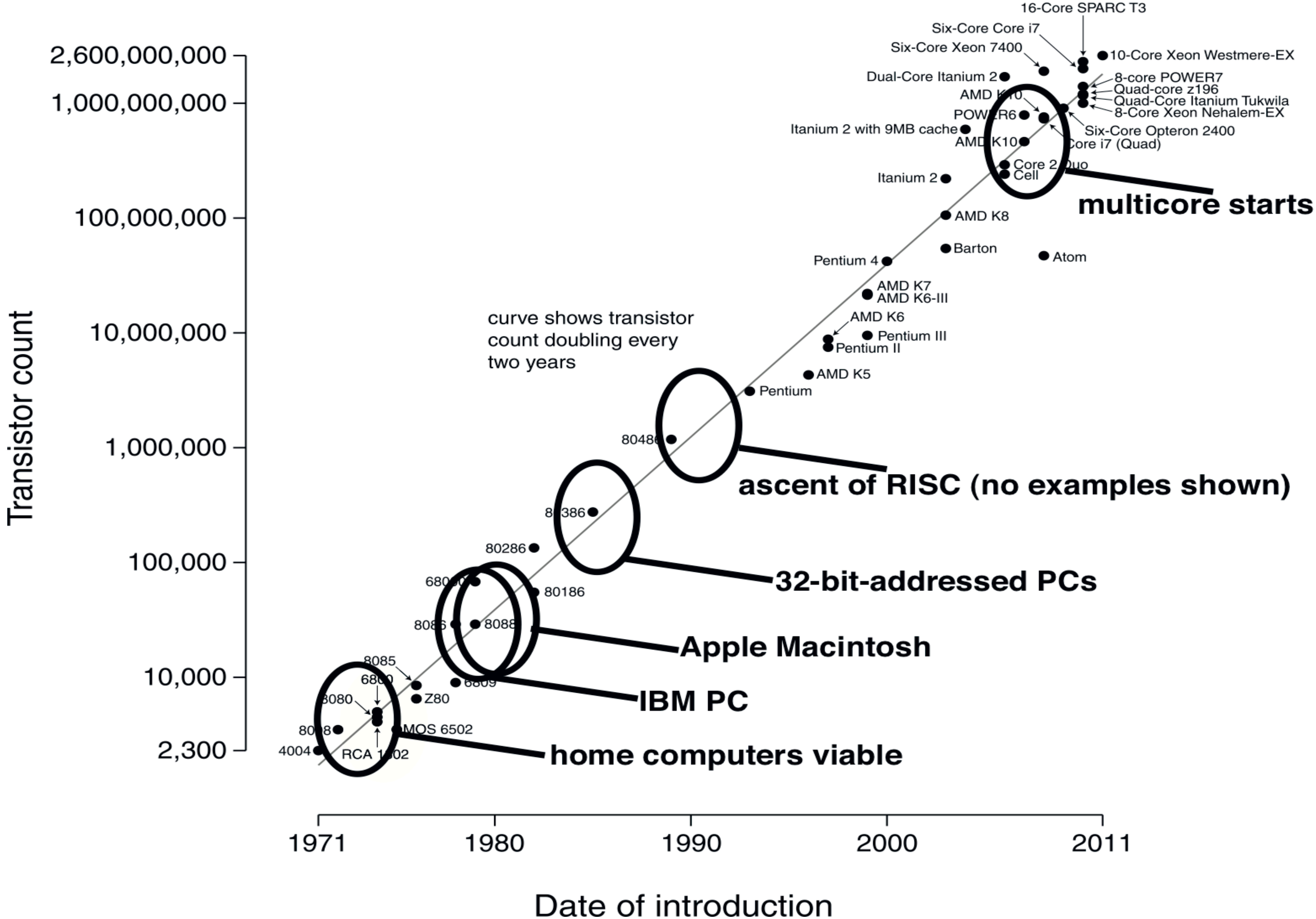
Consumer Graphics – History

- Up to 1995
  - 2D only (S3, Cirrus Logic, Tseng Labs, Trident)
- 1996 3DFX Vodoo (first real 3D card); Introduction of DX3
- 1997 Triangle rendering (… DX5)
- 1999 Multi-Pipe, Multitexture (…DX7)
- 2000 Transform and lighting (…DX8)
- 2001 Programmable shaders
  - PCs surpass workstations
- 2002 Full floating point
- 2004 Full looping and conditionals
- 2006/07 Geometry/Primitive shaders (DX10, OpenGL 2.1)
- 2007/08 CUDA (Nvidia) - GPU General Purpose Computing
- 2009 DX11: Multithreaded rend., Compute shaders, Tessel.
- 2018 Ray-Tracing Cores (RTX) & DirectX Raytracing (in DX12)

Moore‘s Law

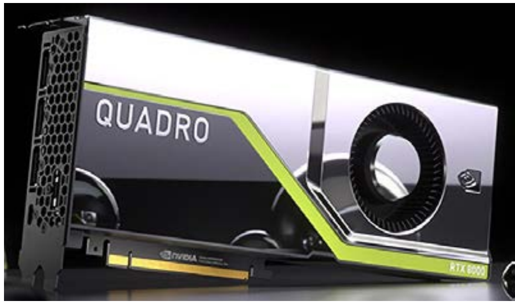
- Gordon Moore, Intel co-founder, 1965
- Exponential growth in number of transistors
- Doubles every 18 months
  - yearly growth: factor 1.6
  - Slow development since 2002 (2.8GHz available since December 2002);
  - But increase in number of cores - currently 18-core CPUs
- Moore‘s Law is coming to an end

Microprocessor Transistor Counts 1971-2011 & Moore's Law



Hannes
Kaufmann

Multi-Core Graphics



NVIDIA Quadro RTX 8000
4608 cores, 576 Tensor cores, 10 GigaRays/sec,
1730 Mhz, 48 GB GDDR6
~15944 GFLOPs (single prec.)

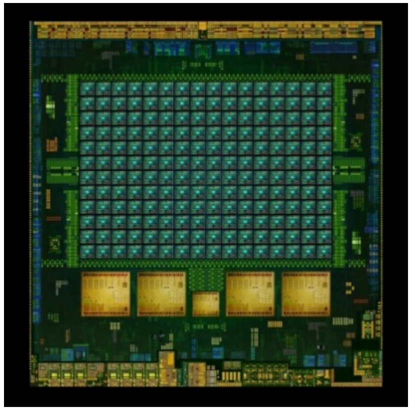
- Faster than the fastest Supercomputer in 2001
- Almost Moore’s law squared (^1.5-2.0)
- In the past performance doubled every **9-12 months** – not anymore but still fast development
- Used in HPC parallel computers (CUDA, Tesla)
  - Molecular dynamics, climate simulations, fluid dynamics
  - …everything highly parallel computable
- Speedup 10-100x compared to standard processors

And it goes on and on….

- Performance increase expected to continue within the next few years
  - Smaller chip production processes possible (currently 12nm for graphics cards, 10nm CPUs)
  - Multiple graphics cards or GPUs in a PC
  - Multi-core GPUs
- General purpose computing on GPUs
  - OpenCL
  - CUDA / www.gpgpu.org

Hannes Kaufmann

Mobile ARM Graphics Chips for Smartphones/Tablets



Example: Tegra X1 (2015)

- Processor: 4 ARM Cortex-A57 + 4 A53 cores, 20nm
- **NVIDIA Maxwell 256-core GPU** @ 1GHz supporting GPU computing – **CUDA**, **DirectX 12, OpenGL 4.5, OpenGL ES 3.1**
- Video output 4K x 2K @60 Hz, 1080p @120 Hz
- 4k H.265 video decode

Hannes Kaufmann

What are the benefits in VR/AR?

Which features are needed?

3D Card High End Model

- nVidia Quadro RTX 8000 (~ € 10.000.-)
- 48 GB GDDR6 RAM
- Based on Turing Architecture (Geforce RTX 2080)
- 4608 CUDA cores, 576 Tensor cores
- 672 GB/s Bandwidth
- optimized OpenGL drivers (comp. to consumer card)
- 16K x 16K texture resolution
- DX12, Shader Model 6.1, OpenGL 4.6, Vulkan 1.1.78

Some Relevant Features (for VR)

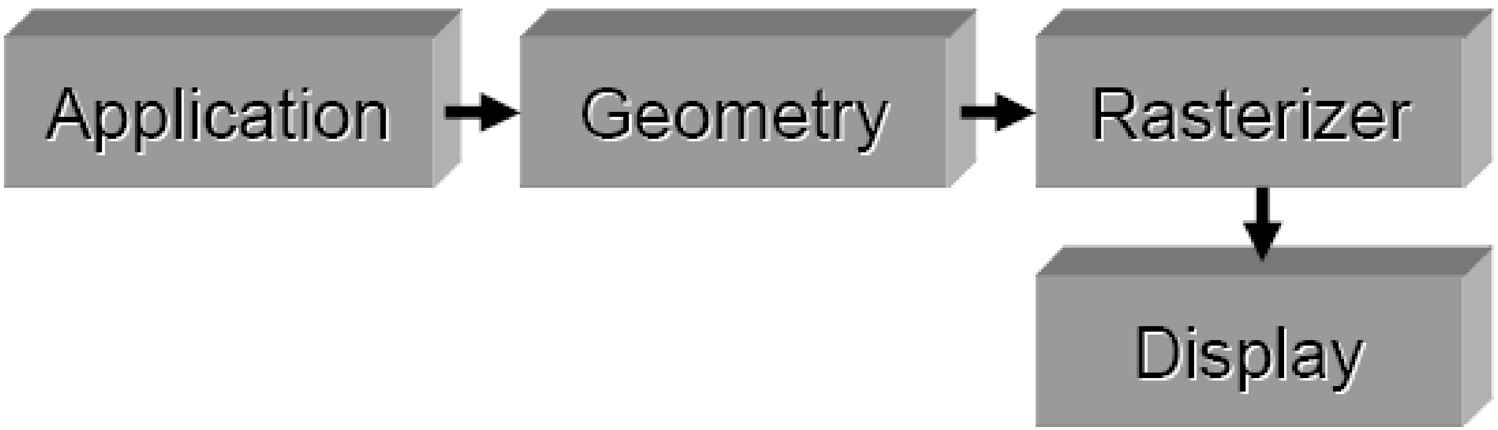
- Memory size: 48 GB
- 4 DisplayPorts 1.4 (8K@60Hz or 4K@120Hz), 1 VirtualLink (1 connector for VR)
- OpenGL quad-buffered stereo (optional 3-pin sync connector); 3D Vision Pro
- NVLink Technology
- Nvidia Mosaic: 2-8 displays
- Fast 3D Texture transfer; HW 3D Window clipping
- Quadro-Sync (optional) with Framelock and Genlock
- HDR technology, 30-bit color, SDI output option
- Quality: 64 x Full-Scene Antialiasing (FSAA), ...

Hannes
Kaufmann

Explanations
&
Back to the Basics

3D Graphics
Basics

The Graphics Pipeline(s)

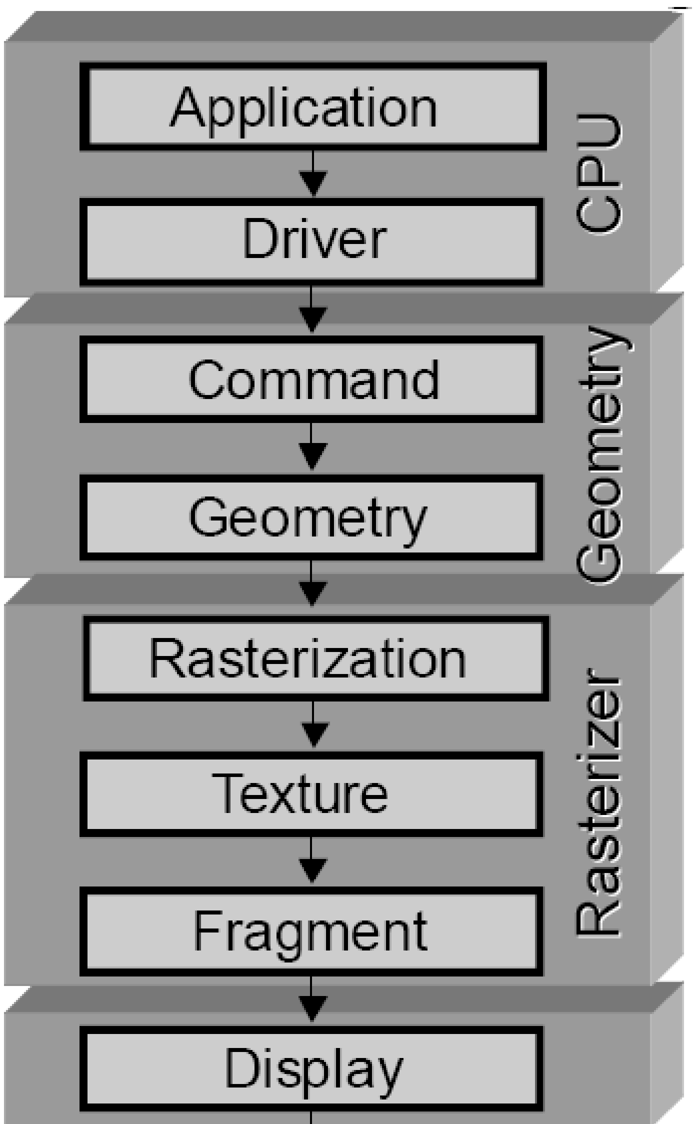


What for ?

Understanding the rendering pipeline is the key to real-time rendering!

- Insights into how things work
  - Understanding algorithms
- Insights into how fast things work
  - Performance

„Historical“ Fixed Graphics Pipeline



Purpose: Convert Scene to Pixel Data

Fixed processing of scene

Geometry Stage:

- Input: Primitives
- Output: 2D window coordinates

Rasterization Stage:

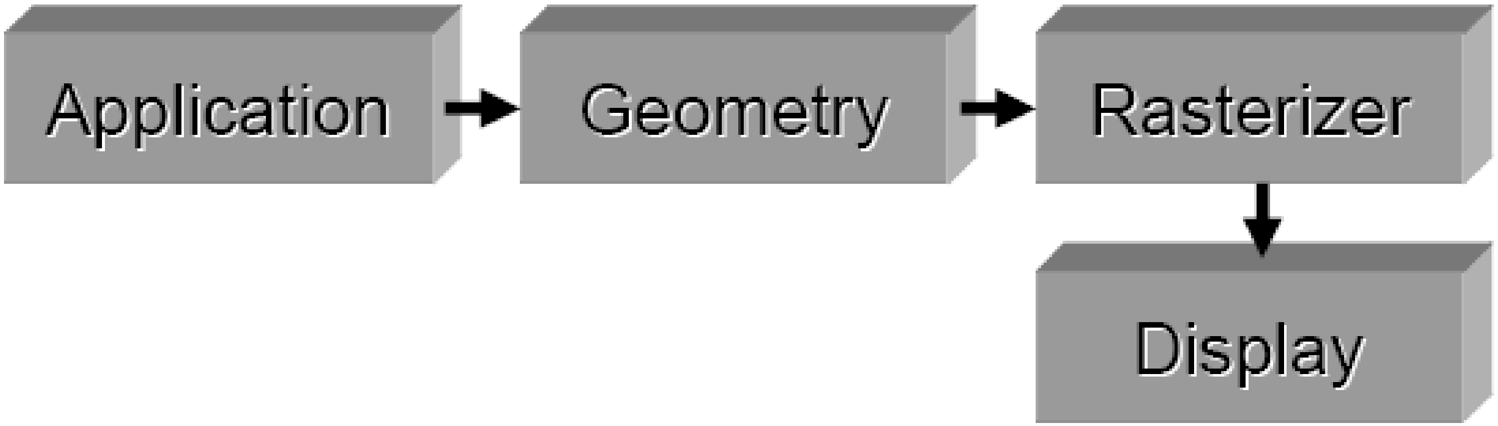
- Input: 2D window coordinates
- Output: Pixels
- Fragment: “pixel”, but with additional info (alpha, depth, stencil, …)

Nowadays every part of the pipeline is hardware accelerated !

Hannes
Kaufmann

3D Graphics
Basics

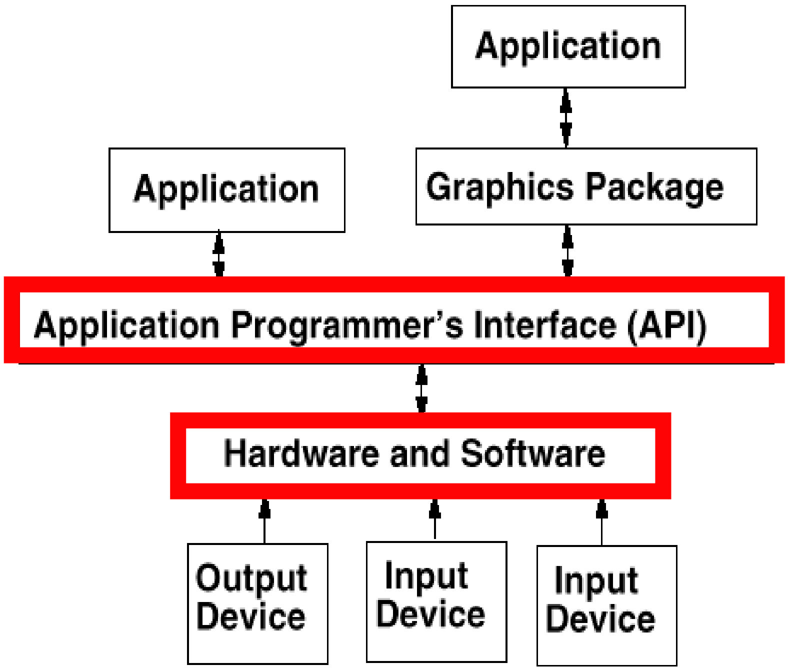
The Stages



(1) Application Stage: 3D Graphics Programming

3D Application Programmer's Interfaces (APIs)

- Access to Hardware
- Standards:
  - OpenGL, Direct3D
- Language: C, C++ (mostly)
- Higher Level APIs based on OpenGL, Direct3D
  - Game Engines
  - Scene Graph APIs:
    - OpenInventor, Java3D
    - OpenSceneGraph, Performer,...



OpenGL – Hello World

```
#include <GL/glut.h>

void display(void) {
    glClear (GL_COLOR_BUFFER_BIT);
    /* draw white polygon (rectangle) with
    corners at (0.25, 0.25, 0.0)
    and (0.75, 0.75, 0.0) */
    glColor3f (1.0, 1.0, 1.0);
    glBegin(GL_POLYGON);
    glVertex3f (0.25, 0.25, 0.0);
    glVertex3f (0.75, 0.25, 0.0);
    glVertex3f (0.75, 0.75, 0.0);
    glVertex3f (0.25, 0.75, 0.0);
    glEnd();
    glFlush ();
}
```

```
void init (void) {
    glClearColor (0.0, 0.0, 0.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE
    GLUT_RGB);
    glutInitWindowSize (250, 250);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("hello");
    init ();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

OpenGL Geometric Primitives

All geometric primitives are specified by vertices

(1) Application Stage

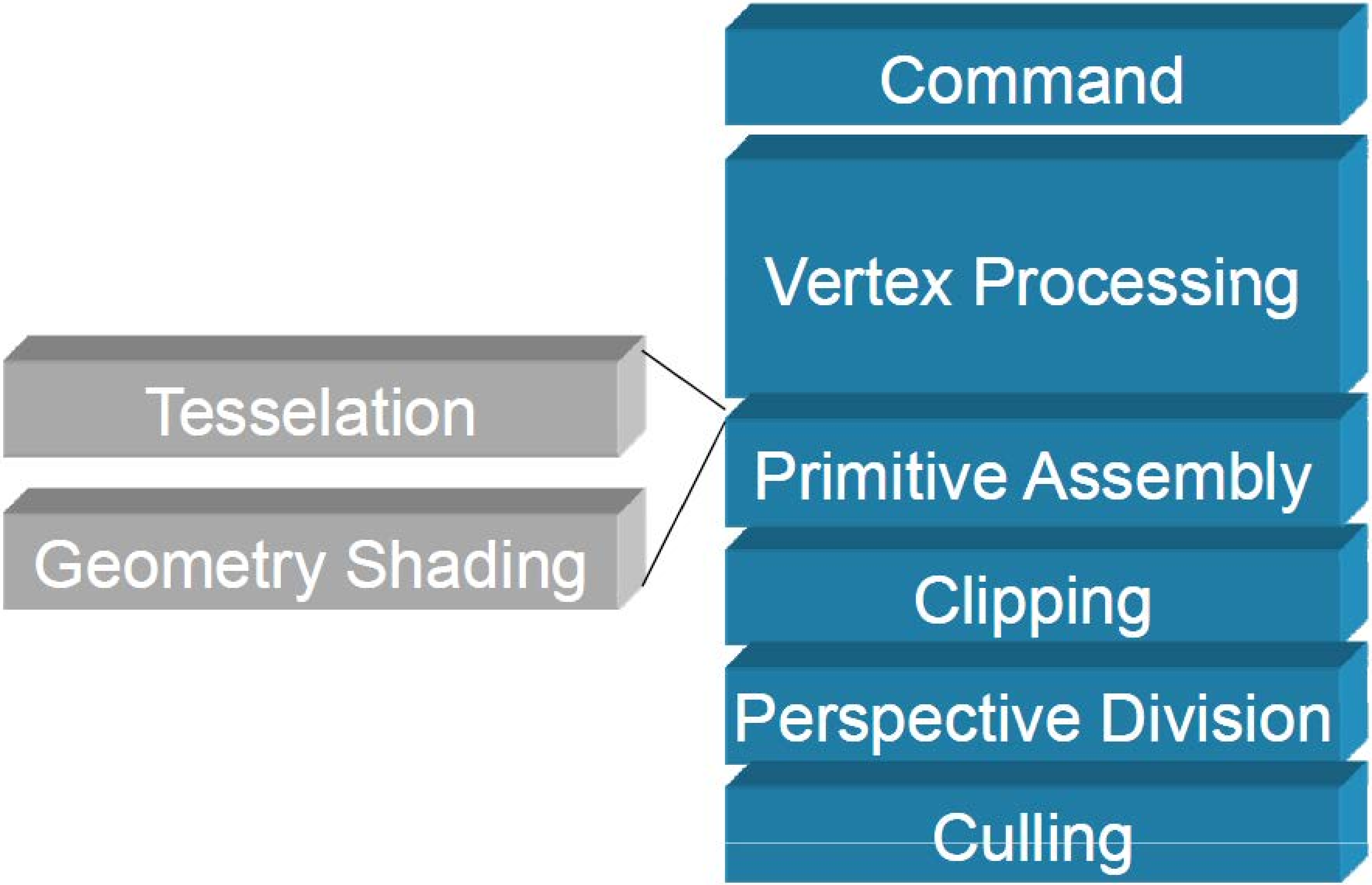
- Generate database (Scene description)
  - Usually only once
  - Load from disk
  - Build acceleration / optimization structures
    - Lots of optimizations possible: Build hierarchy, Level of Details, Culling Techniques, Impostors,…
- Simulation (Animation, AI, Physics)
- Input event handlers
- Modify data structures
- Database traversal
- Primitive generation
- Shaders (vertex, geometry, fragment)

Hannes
Kaufmann

Graphics Driver

- Command interpretation/translation
  - Host commands ⟶ GPU commands
- Handle data transfer
- Memory management
- Emulation of missing features (e.g. full OpenGL 4.5 support)

(2) Geometry Stage

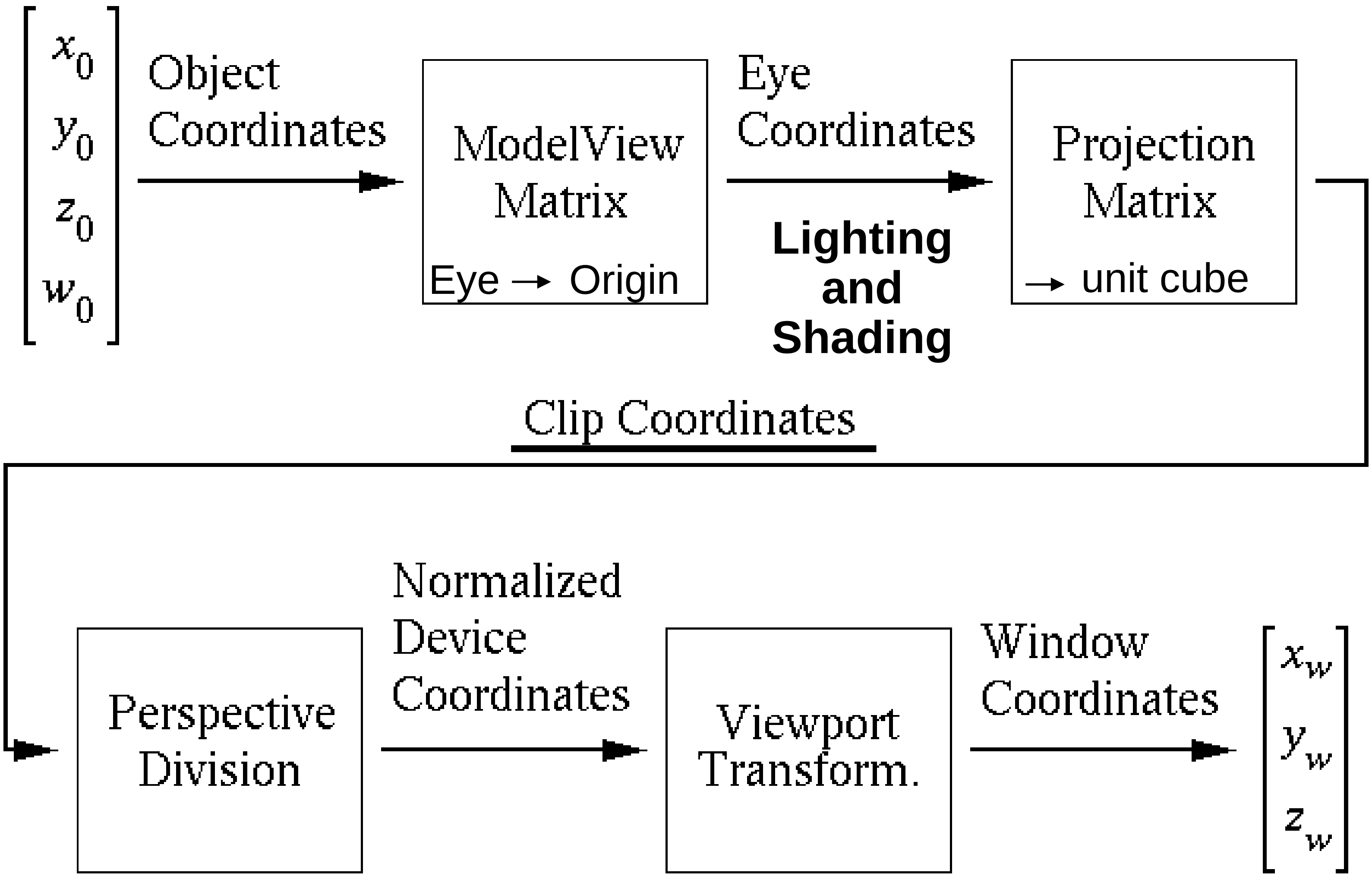


Hannes
Kaufmann

Command

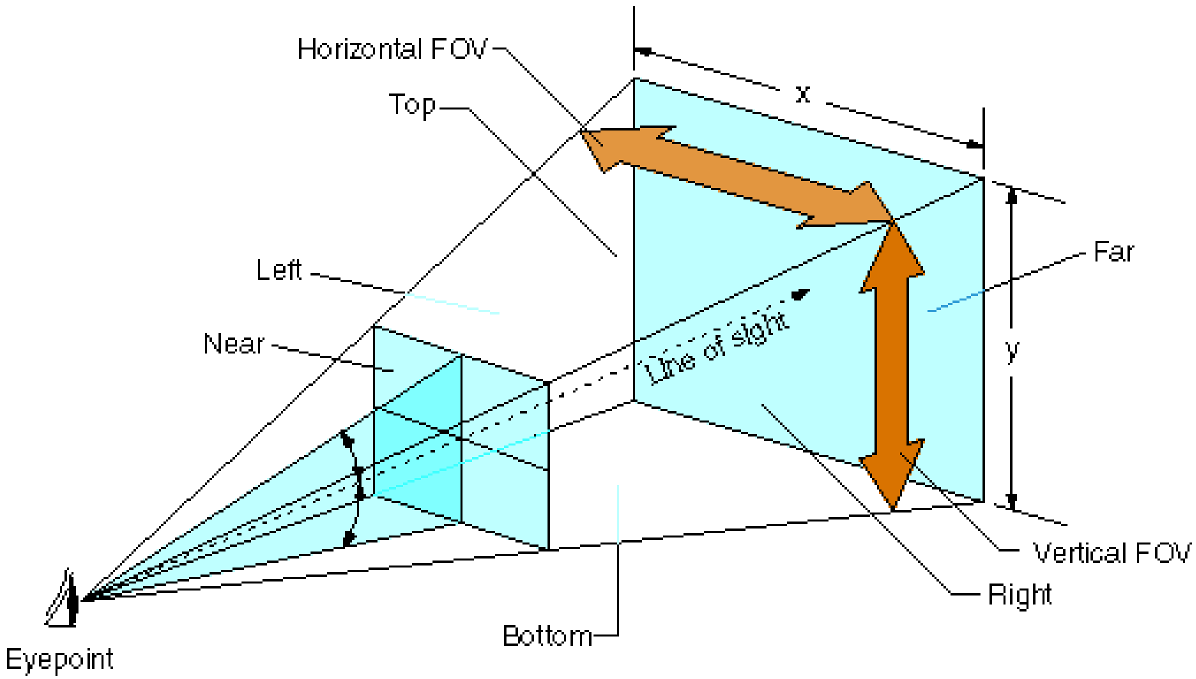
- Command buffering
- Command interpretation
- Unpack and perform format conversion „Input Assembler“

Vertex Processing: Old Geometry Stage



Hannes
Kaufmann

Viewing Frustum



$$\text{Aspect Ratio} = \frac{y}{x} = \frac{\tan(\text{vertical FOV}/2)}{\tan(\text{horizontal FOV}/2)}$$

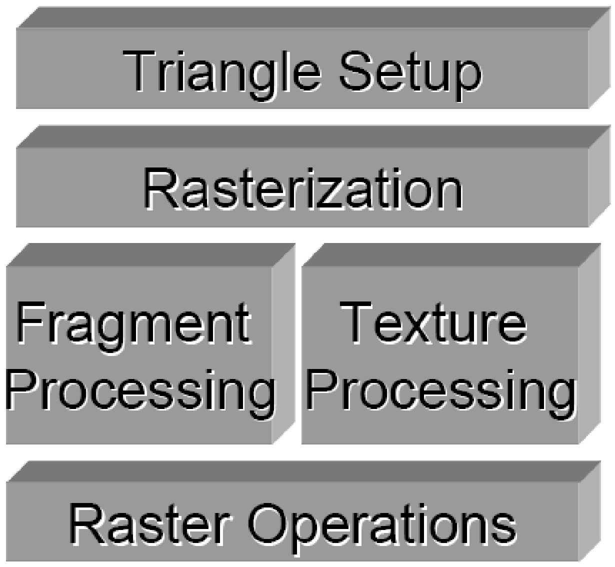
Hannes
Kaufmann

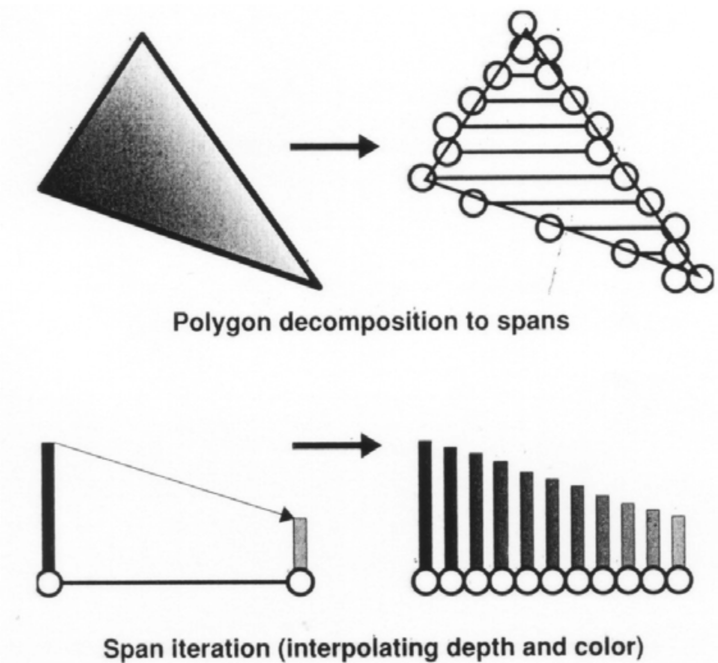
Vertex Processing

- Fixed function pipeline:
  - User has to provide matrices, the rest happens automatically
- Programmable pipeline:
  - User has to provide matrices/other data to shader
  - Shader Code transforms vertex explicitly
    - We can do whatever we want with the vertex!

(3) Rasterization Stage

- Input: 2D Geometric Primitives (Points, Lines, Polys, Bitmaps)
- Primitives needed!
- 1st step output: Fragments (Pixel-Coord. + Color + Depth + Texture-Coord.)
- Polygons are decomposed (various methods)





(3) Rasterization Stage

- Per-Fragment Operations
- Pixel Ownership Test (Window visible?)

Buffers:

- Frame Buffer (Color + Alpha channel)
- Depth Buffer Test (z-Buffer)
- Stencil Buffer
- Accumulation Buffer
- P-Buffer (aux. color buffer -> direct rendering)

Rasterizer/Display Stage

- Framebuffer pixel format: RGBA vs. indexed (colormap)
- Bits: 32, 24 (true color) 16, 15 (high color), 8
- Double buffering, Triple Buffering
- For Stereo: Quad buffer
- Per-window video mode (e.g. stereo, mono)
- **Display**: frame buffer -> screen

„Modern“ Programmable Graphics Pipeline

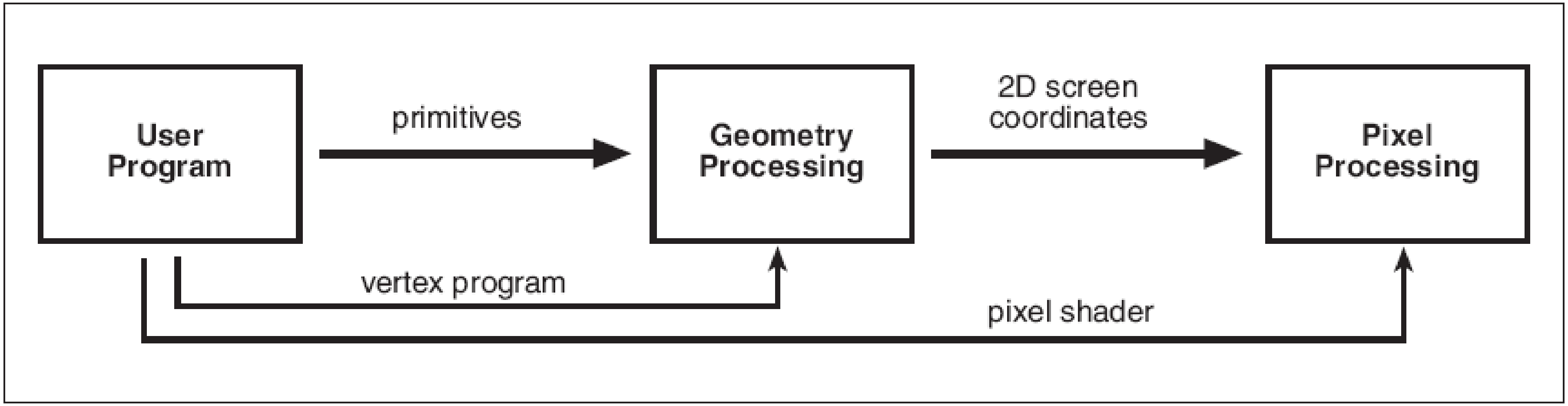


Figure 17.5. The programmable graphics hardware pipeline. The user program supplies primitives, vertex programs, and fragment programs to the hardware.

- Vertex Shader integrated in „old“ Geometry Stage
  - Allows per vertex transformations e.g. warping
- Fragment/Pixel Shader integrated in „old“ Rasterization Stage
  - Fragment: „pixel“ with additional information (alpha, depth, stencil,…)
  - Allows e.g. per pixel lighting,….

Vertex and Fragment Shaders

Hannes Ka...

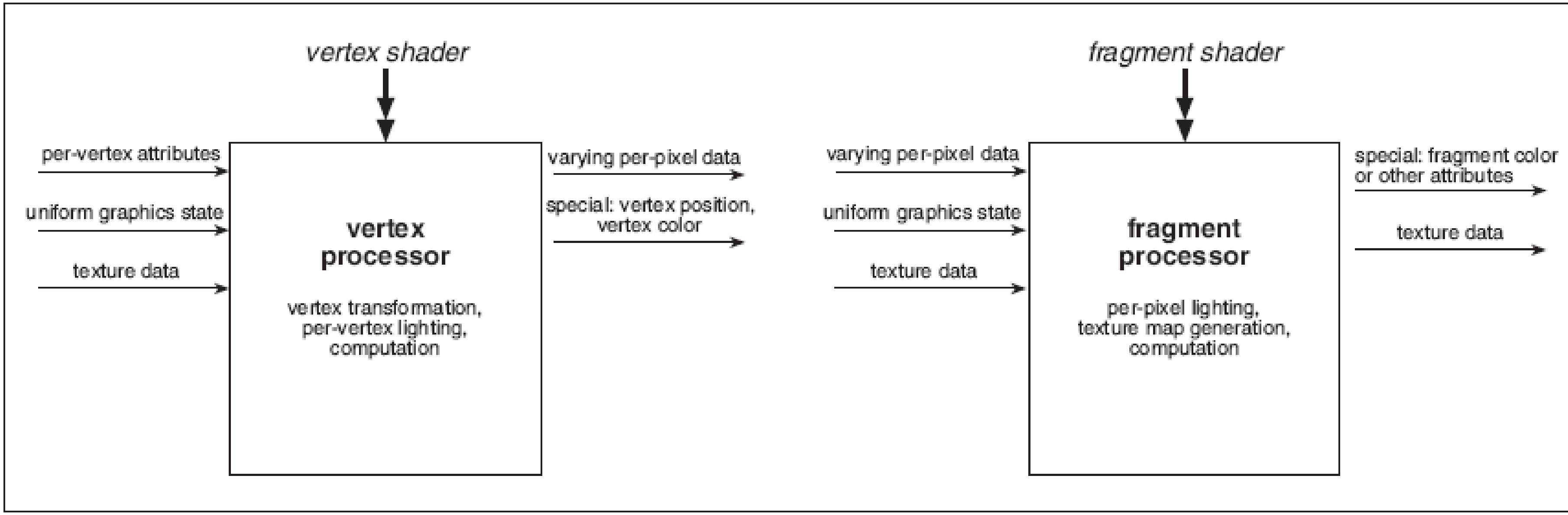
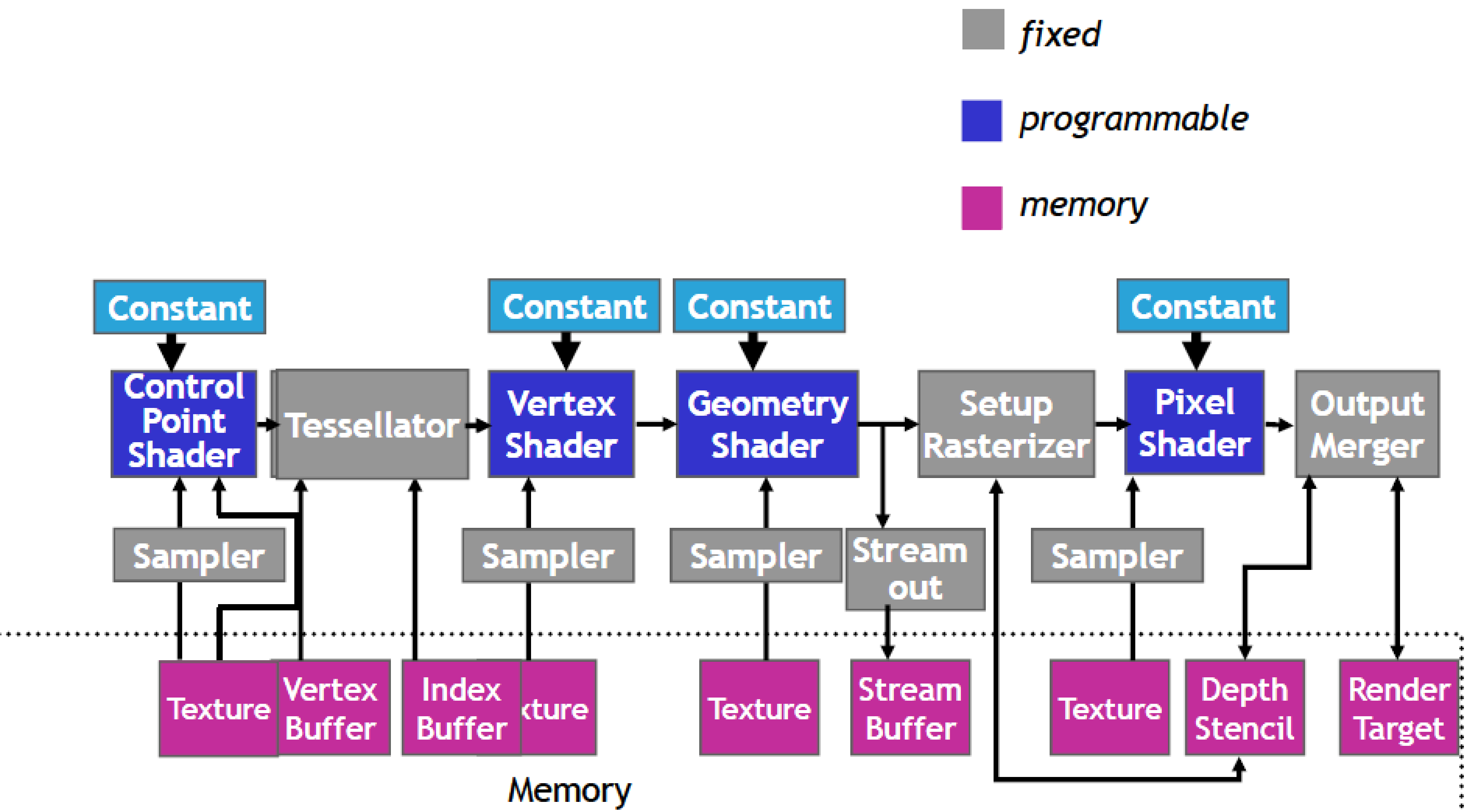


Figure 17.6. The execution model for shader programs. Input, such as per-vertex attributes, graphics state-related uniform variables, varying data, and texture maps are provided to vertex and fragment programs within the shader processor. Shaders output special variables used in later parts of the graphics pipeline.

- Various Shading Languages
  - ARB - GPU assembly language (optimized)
  - GLSL (Open GL Shading Language – in OpenGL 2.0)
  - HLSL (High Level Shading Language – Microsoft)
  - CG (Nvidia) – used in Unity

Current OpenGL 4.x / DirectX 11 Architecture



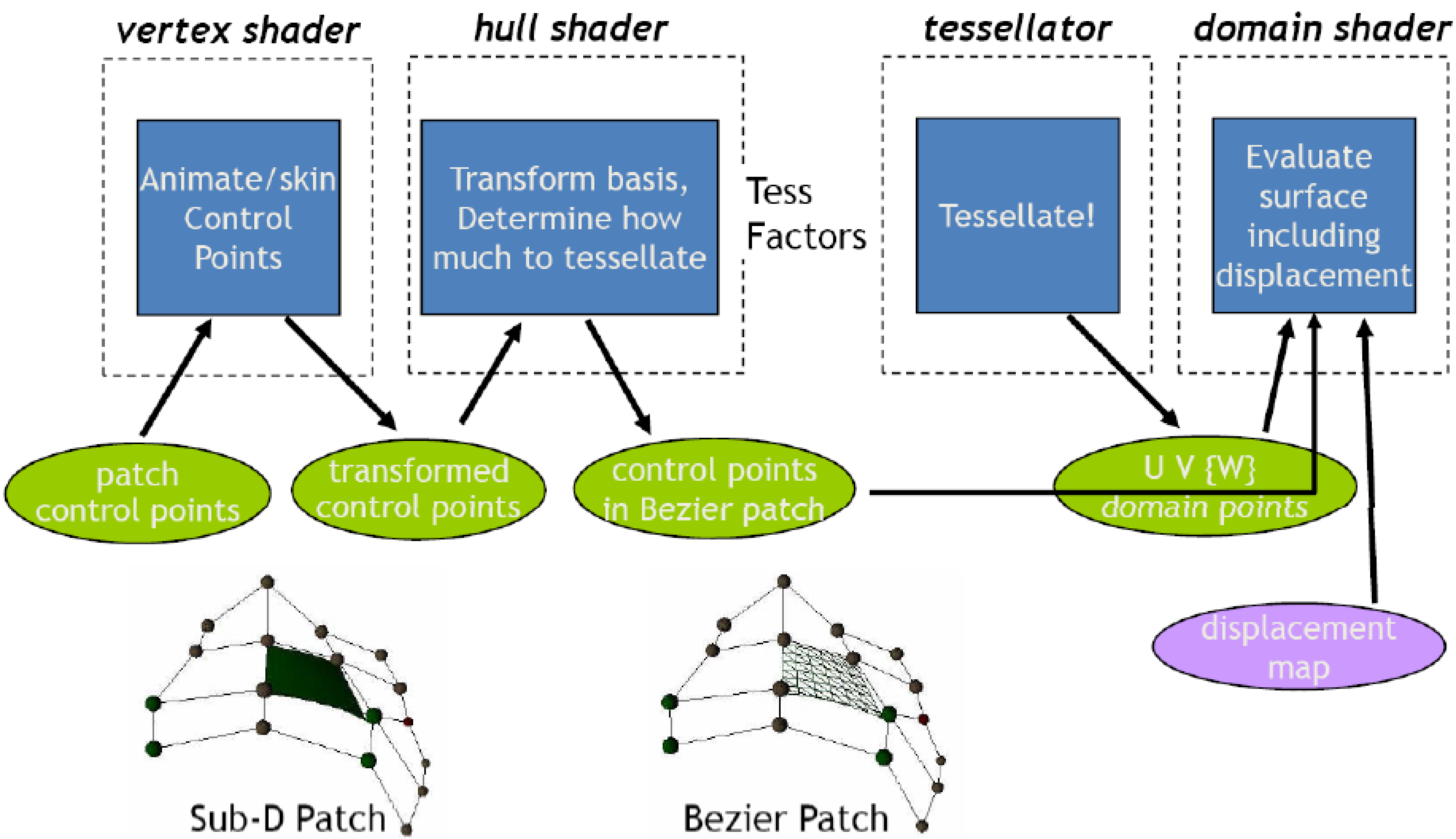
Hannes
Kaufmann

Tesselation

- If just triangles, nothing needs to be done, otherwise:
  - Evaluation of polynomials for curved surfaces
  - Create vertices (tesselation)
- OpenGL 4/ DirectX11 specifies this in hardware
  - 3 new shader stages!
  - Still not trivial (special algorithms required)
- https://www.nvidia.com/content/siggraph/Rollin_Oster_OpenGL_CUDA.pdf

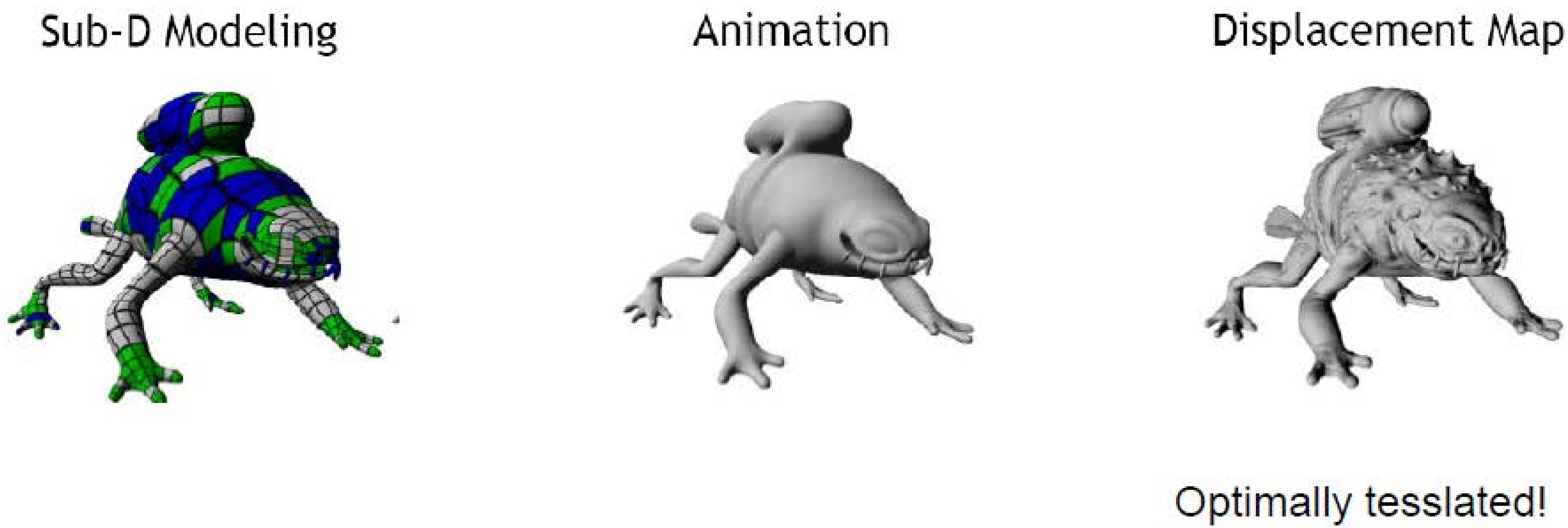
Hannes Kaufmann

DirectX 11 Tesselation



Hannes
Kaufmann

Tesselation Example & Displacement Map



Hannes
Kaufmann

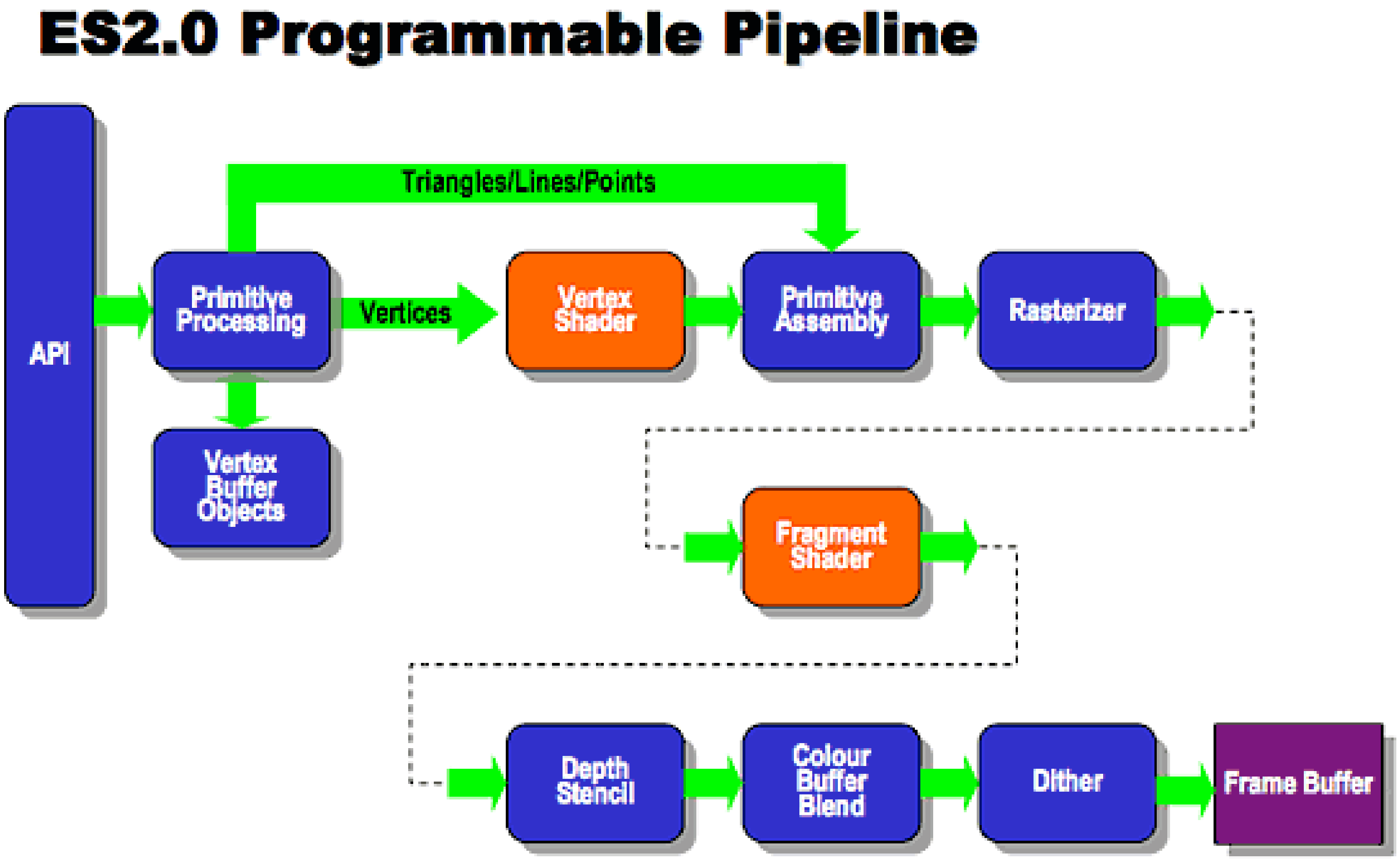
Mobile Graphics: OpenGL ES

OpenGL ES Momentum

- **The leading 3D rendering API for mobile and embedded devices**
  - Based on desktop OpenGL – but optimized for mobile / handheld devices
  - Removes redundancy & rarely used features - adds mobile-friendly data types
  - The power of OpenGL distilled into a much smaller package
- **OpenGL ES adopted by every major handset OS**
  - Pervasive mobile 3D is evolving fast
- **OpenGL ES has become the most widely deployed 3D API**
  - Used in diverse applications, devices and markets
  - Mobile phones, games consoles, personal navigation devices, personal media players automotive systems, settop boxes

Hannes Kaufmann

OpenGL ES 2.0



2014: OpenGL ES 3.x

- 32bit Floating point support
- Texture compression, 3D textures
- Multiple apps can share 3D hardware

Current State

- High bandwidth interconnect of CPU and GPU
  - CPU and streaming units working together
  - Nvidia NVLink
- Heterogeneous architectures
  - CPU and GPU on one chip (especially mobile chips)
  - GPU is treated as a parallel streaming PU
- Whole pipeline is fully programmable (GPU computing)

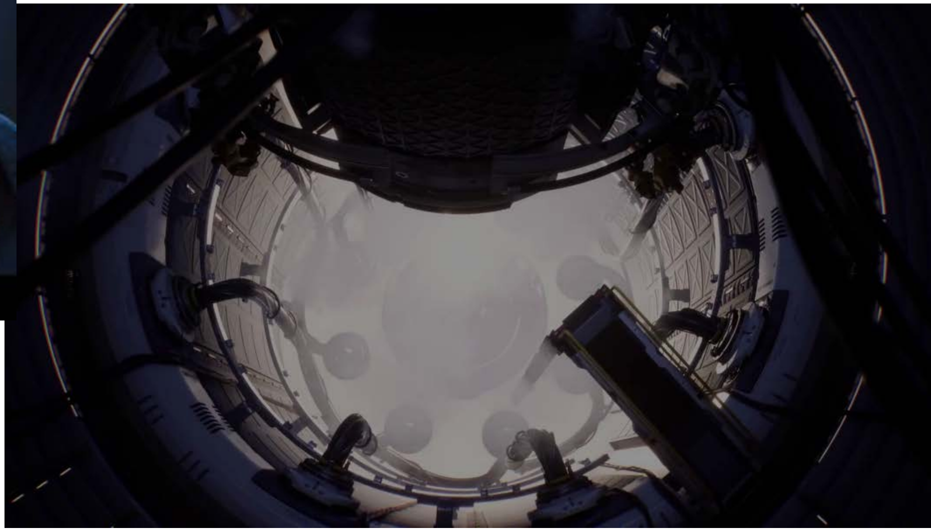
- **Good-bye to the one way graphics pipeline!**

Hannes
Kaufmann

Architectural Addition: Real Time Ray Tracing

- 2018: Nvidia RTX / DirectX Raytracing (DXR)

Star Wars – 1 Quadro RTX 8000



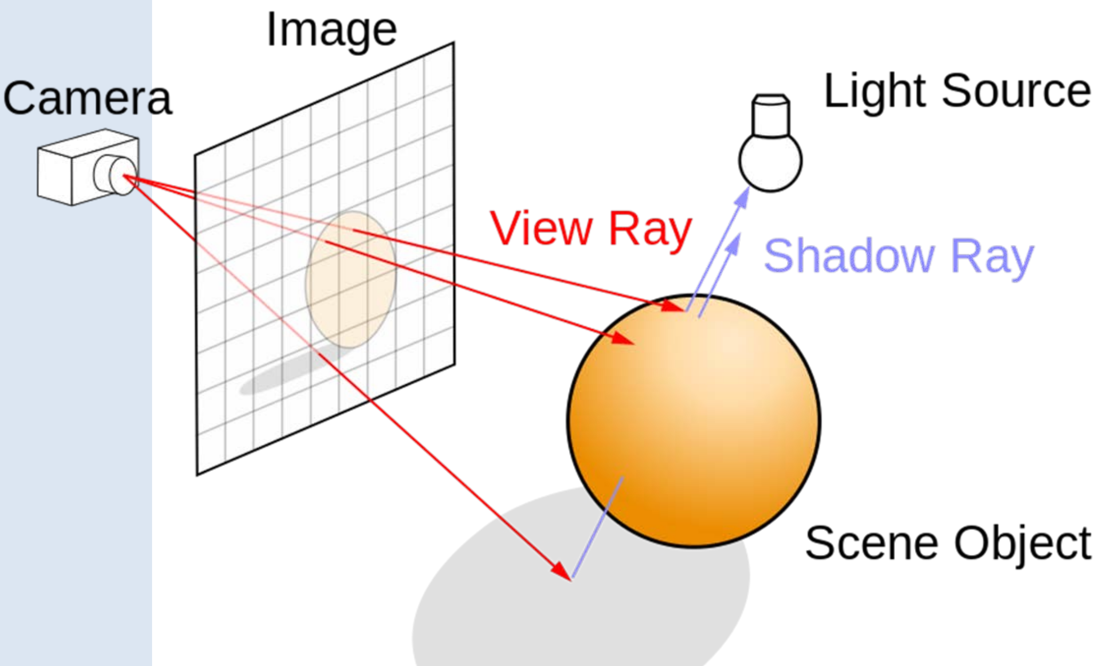
Project Sol – 1 Quadro RTX 6000

Nvidia Geforce RTX 2080Ti

- 12nm chip (TU102)
- 14.2 TFLOPS single precision math
- 4352 CUDA cores
- 544 Tensor cores
- 68 RT (RayTracing) cores

Hannes
Kaufmann

Ray Tracing Principle



1. Construction of the camera/eye rays
2. Intersection with the scene objects
3. Shading
4. Reflection/refraction directions
5. Recursion

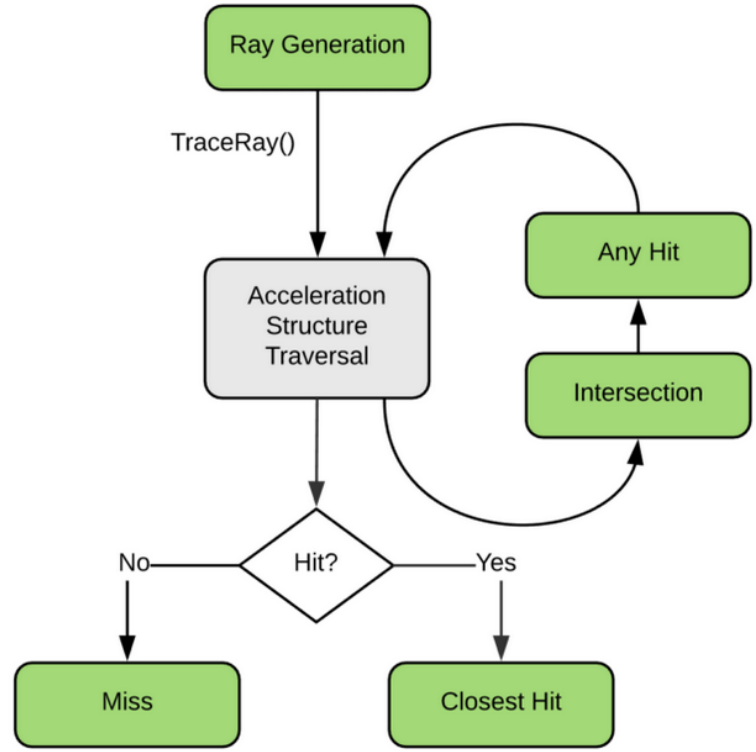
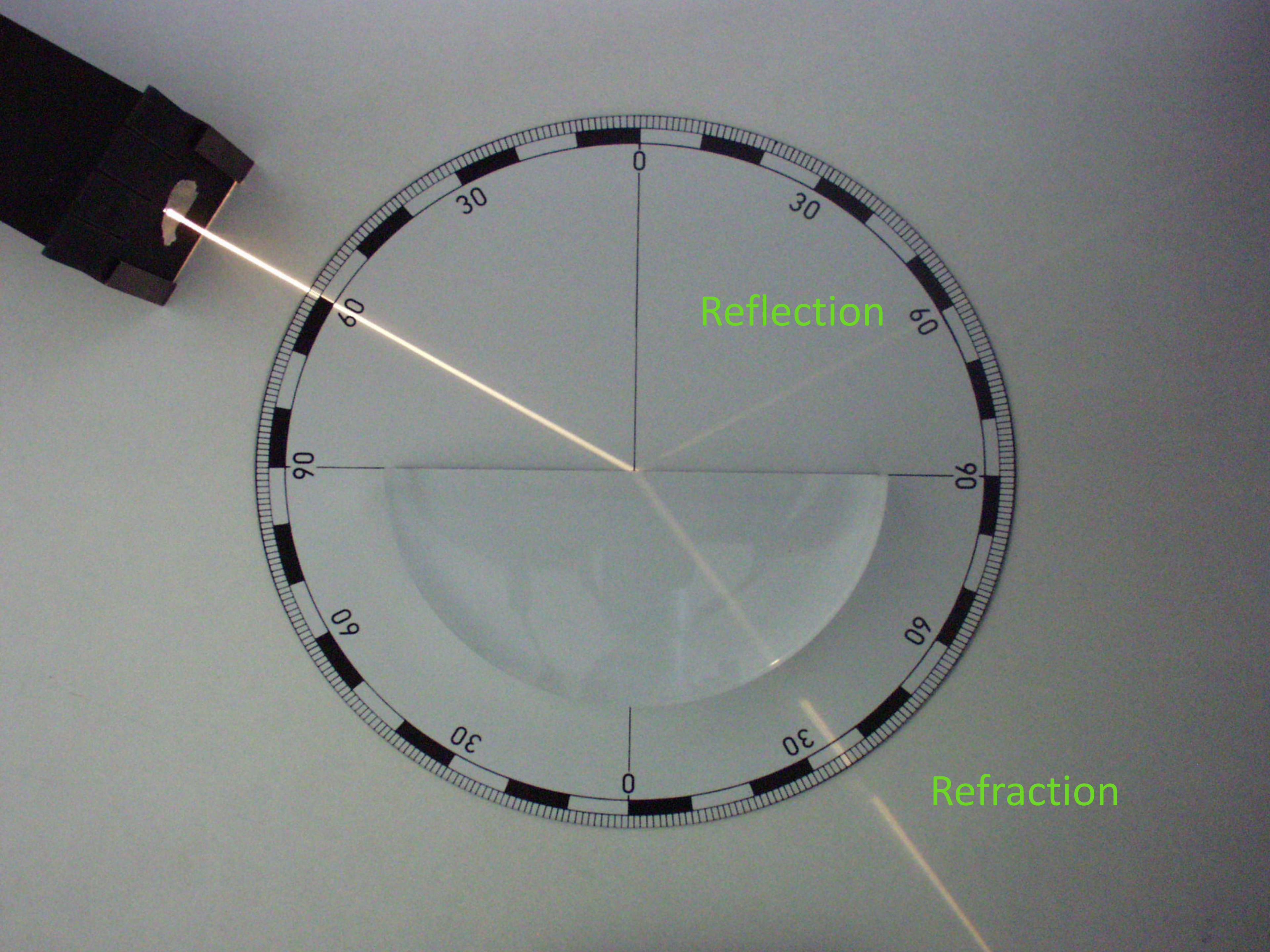


Figure 1. The ray tracing pipeline

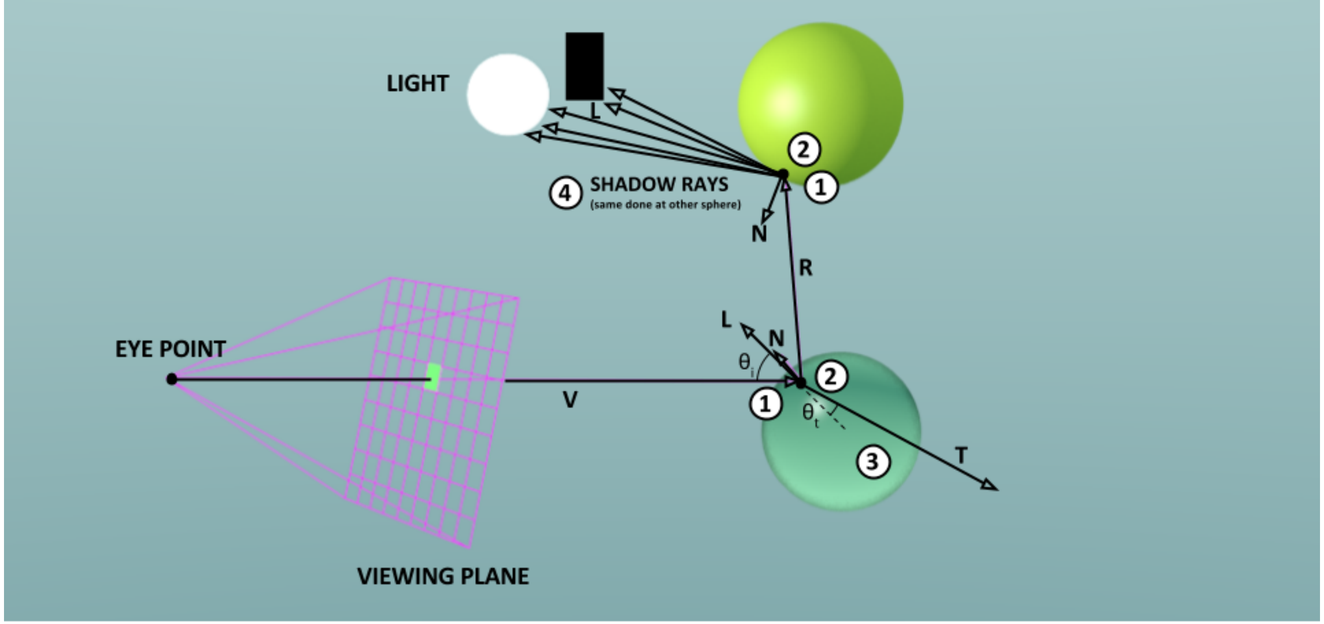


Reflection
Refraction

Hannes
Kaufmann

RAY TRACING

(for one pixel up to first bounce)



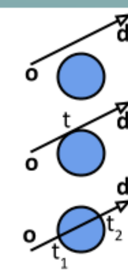
1 Sphere equation: $(\vec{p}-\vec{c})\cdot(\vec{p}-\vec{c}) = r^2$

Ray equation: $\vec{r}(t) = \vec{o} + t\vec{d}$

Intersection:

$$\left(\vec{o}+t\vec{d}-\vec{c}\right)\cdot\left(\vec{o}+t\vec{d}-\vec{c}\right) = r^2$$

$$t^2\left(\vec{d}\cdot\vec{d}\right) + 2(\vec{o}-\vec{c})\,t\vec{d} + (\vec{o}-\vec{c})\cdot(\vec{o}-\vec{c}) - r^2 = 0$$



2 **Illuminiation Equation (Blinn–Phong) with recursive Transmitted and Reflected Intensity:**

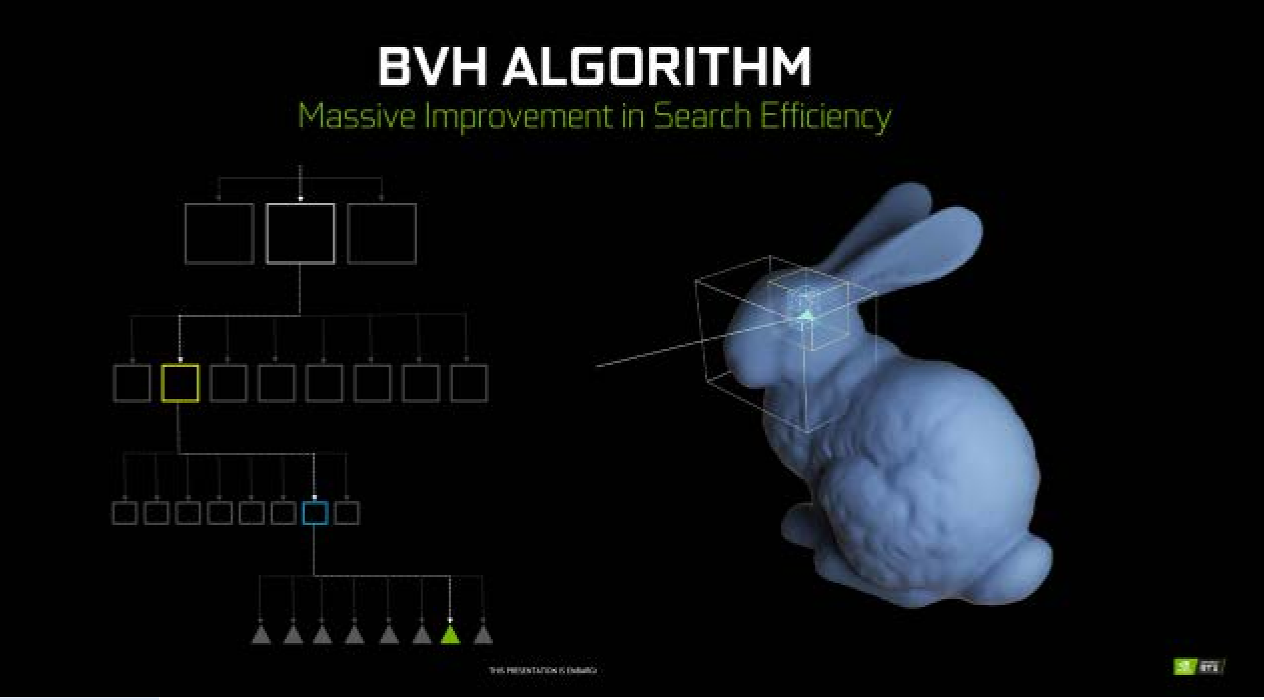
$$I = k_a I_a + I_i\left(k_d\left(\vec{L}\cdot\vec{N}\right) + k_s\left(\vec{V}\cdot\vec{R}\right)^n\right) + \underbrace{k_t I_t + k_r I_r}_{recursion}$$

3 Snell's law: $\frac{\sin\theta_1}{\sin\theta_2} = \frac{v_1}{v_2} = \frac{n_2}{n_1}$ $\quad n_{air}\sin\theta_i = n_{glass}\sin\theta_t$

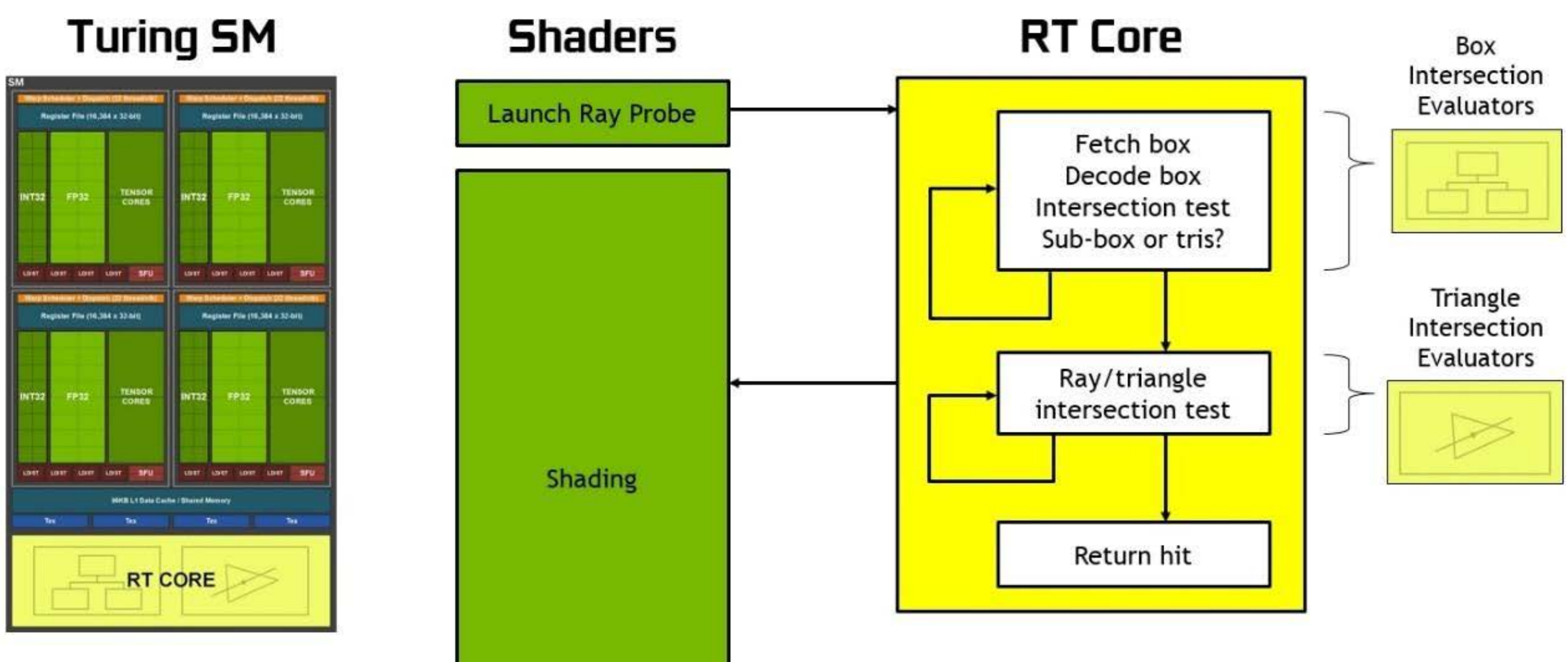
refraction coefficients:
$n_{air} = 1$, $n_{glass} = 1.5$

4 **Area Light Simulation:** $I_{light}\dfrac{\#\,(\text{visible shadow rays})}{\#\,(\text{all shadow rays})}$

Nvidia RT cores



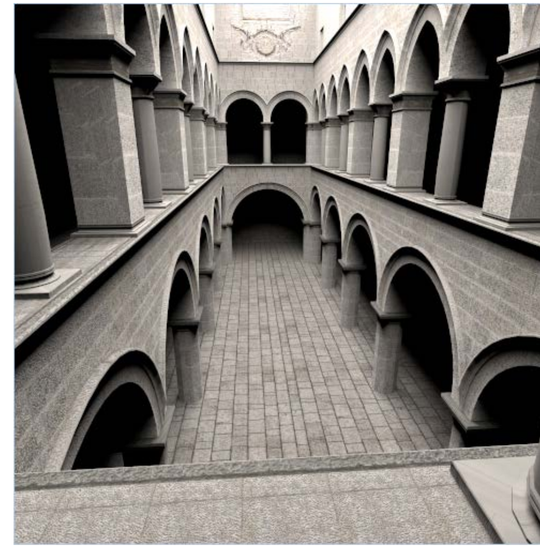
ware Emulation



Ray Tracing De-Noising with Deep Learning Networks

4 samples per pixel at 512x512 resolution

Our algorithm (Davletaliyev, Kan): 0,48sec



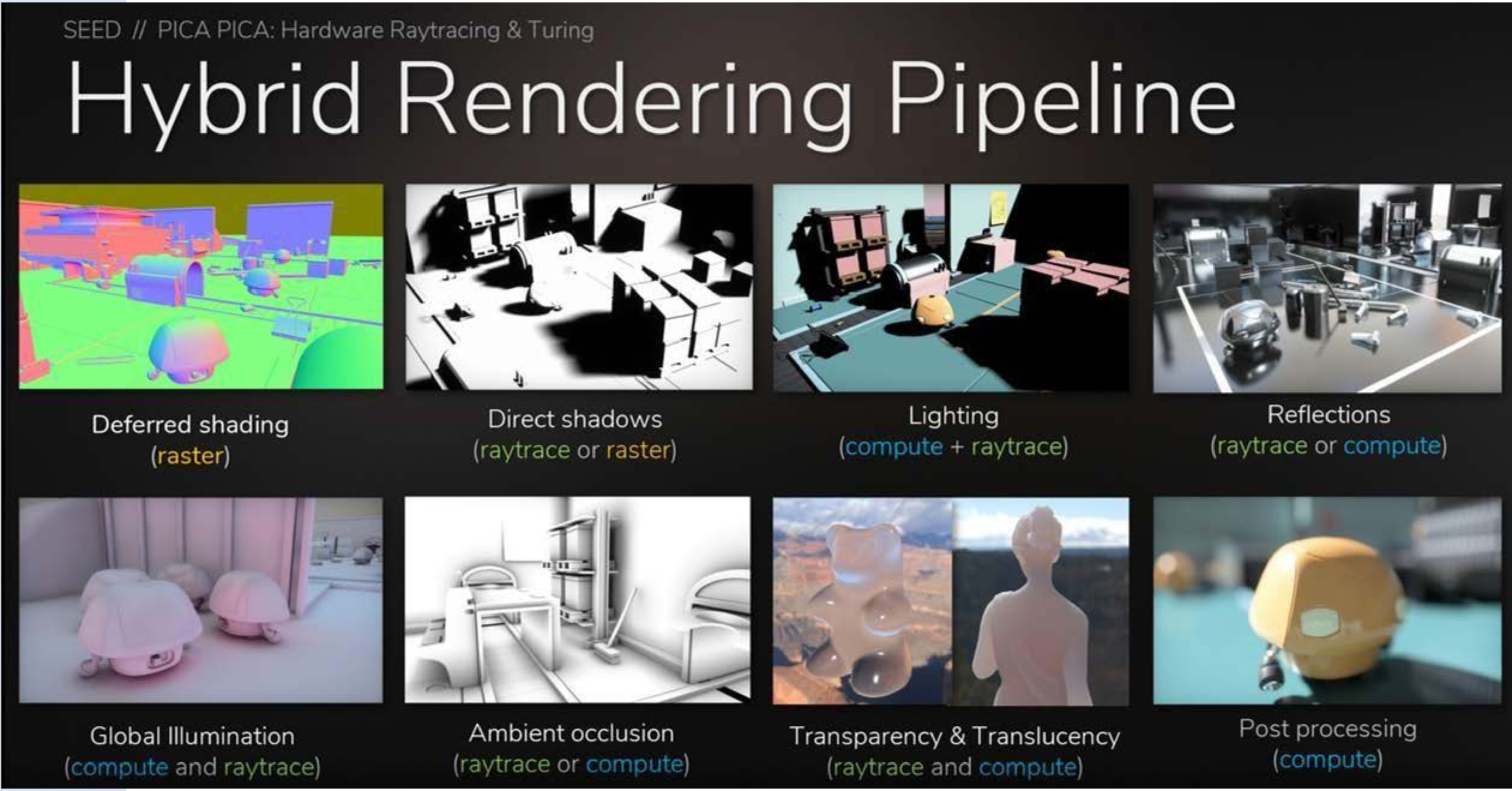
Ground Truth

Nvidia: 1 ray per pixel

Nvidia: 1 ray per pixel – denoised. Real-time !

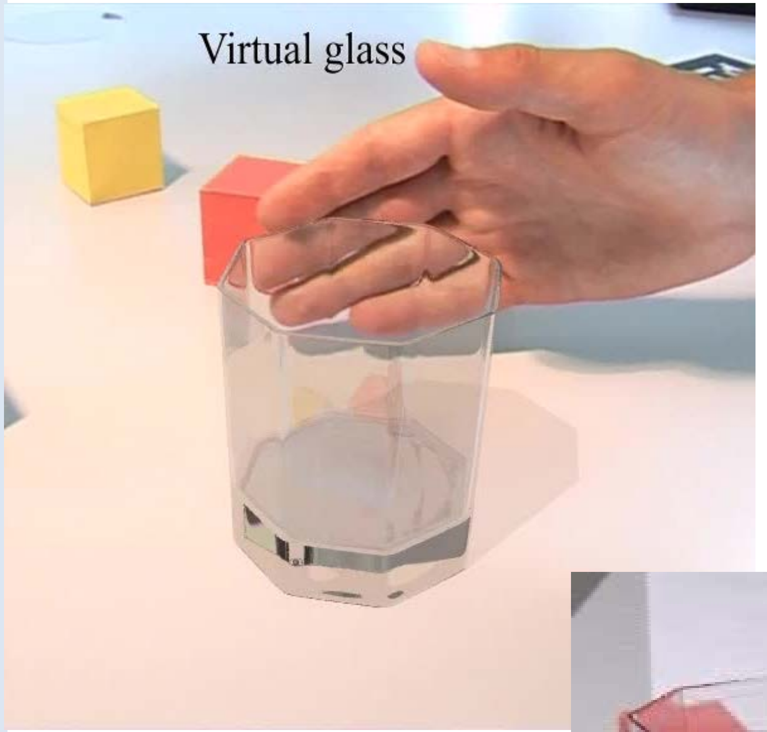
Ground Truth

Hannes
Kaufmann



Hannes
Kaufmann

Real-time Ray Tracing in AR



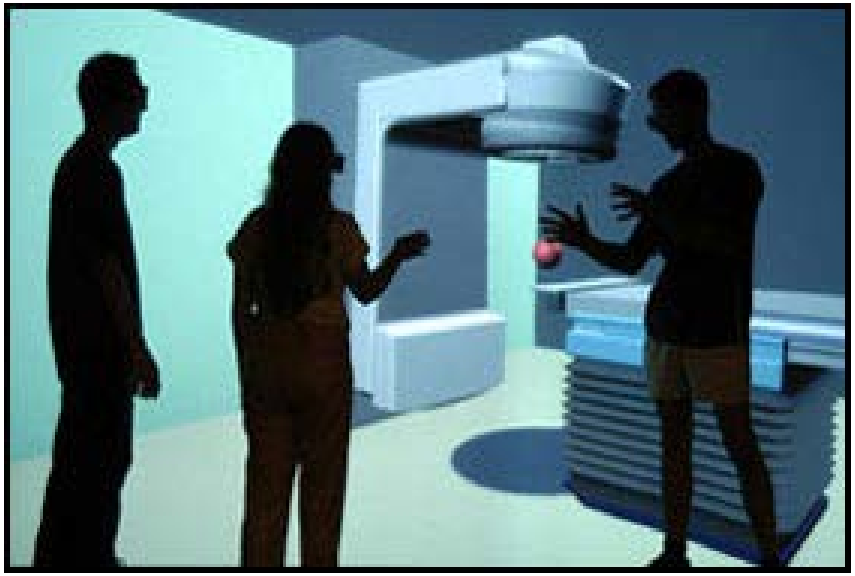
Work by Peter Kan
2012-2014

Hannes
Kaufmann

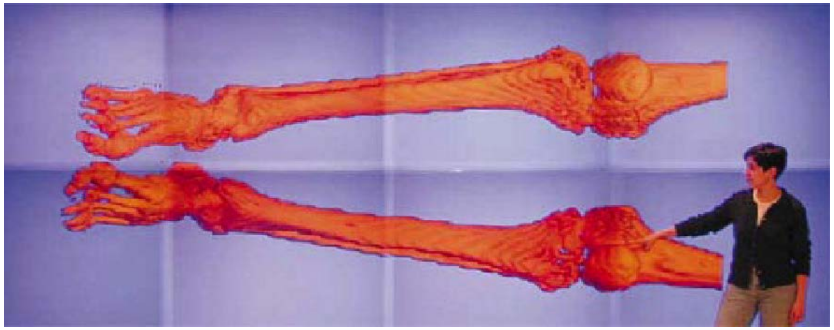
Long term future

- We have…
  - Very high fill rates polygon rates
  - Lots of textures
  - Almost full programmability
  - Few limits (program lengths, memory bandwidth)
  - Real-time ray tracing
- We want (and will get)…
  - Flexible geometry specification
  - Full, easy programmability
  - Higher performance
- Convergence of film rendering and real-time rendering imminent

VR/AR and the Need for Extreme Graphics Power: Examples



Mechanical visualization
CAVE, SGI Onyx
(8 CPUs, 6 outputs)



Princeton Display Wall
3x8 projectors, 24 PC cluster

HMD setups for
larger groups

Hannes Kaufmann

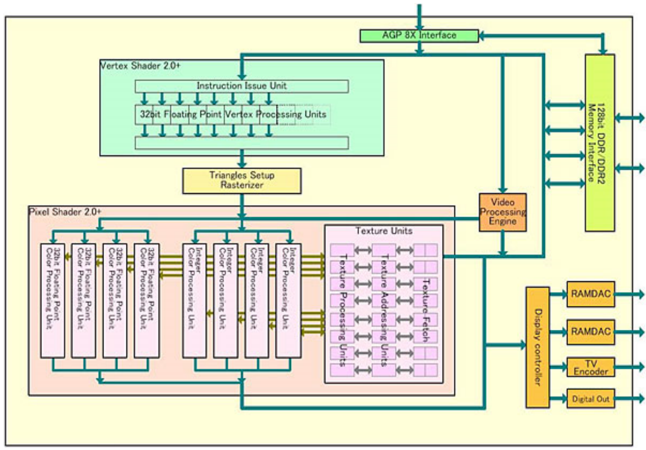
Parallel Graphics Hardware

Overcome bottleneck by parallel computation

Types of parallel graphics:

1. On-chip / on a graphics board (standard)
2. Multiple boards (former: graphics supercomputer)
   Multiple boards with multi GPUs (1+2)
3. PC cluster:
   - Offline Rendering: Standard network – Distributed Environment
   - Realtime Rendering: PC cluster with special hardware

Multiple Graphic Pipelines



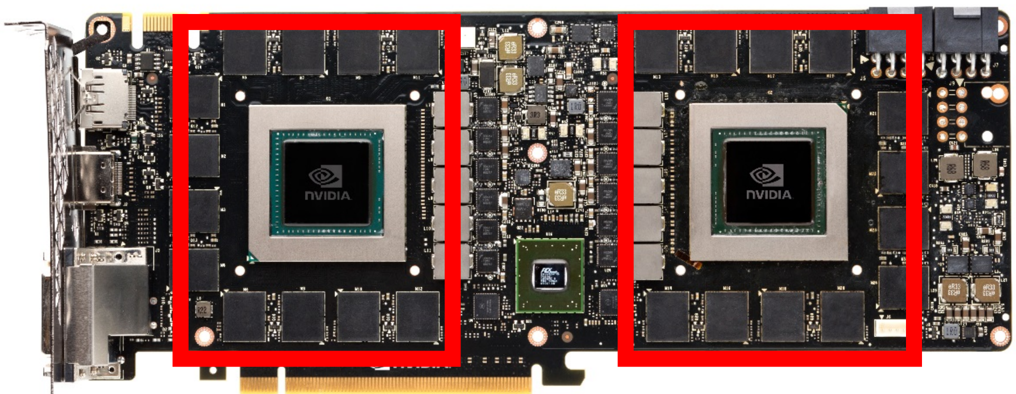
- Pipelines fully in HW
- Multiple independent pipelines can be parallelized
- NVIDIA / AMD
  - CUDA cores / streaming processors

- Modern GPUs process more than 3800 Pipelines in parallel (Unified architecture)
  e.g. Nvidia Quadro P6000

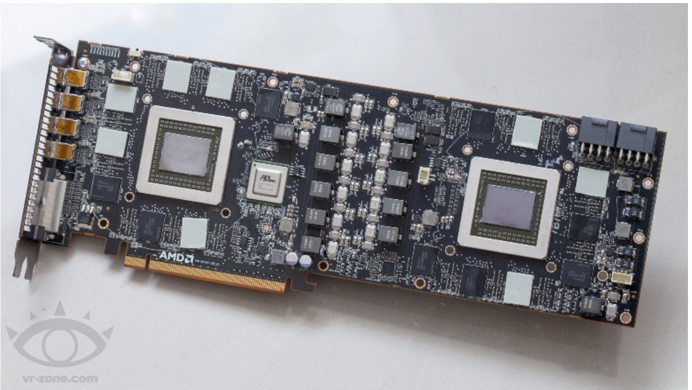
Parallel On-Board

Examples:

- Nvidia Geforce GTX TITAN Z (2014)
  - 2 x 2880 cores



- AMD Radeon R9 295X2 – 2x2816 cores



Hannes
Kaufmann

Parallel Graphics Hardware

Types of parallel graphics:

1. On-chip / on a graphics board (standard)
2. **Multiple boards (former: graphics supercomputer)**
   Multiple boards with multi GPUs (1+2)
3. PC cluster:
   – Offline Rendering: Standard network – Distributed Environment
   – Realtime Rendering: PC cluster with special hardware

Multiple Graphics Boards

Parallel graphics rendering:

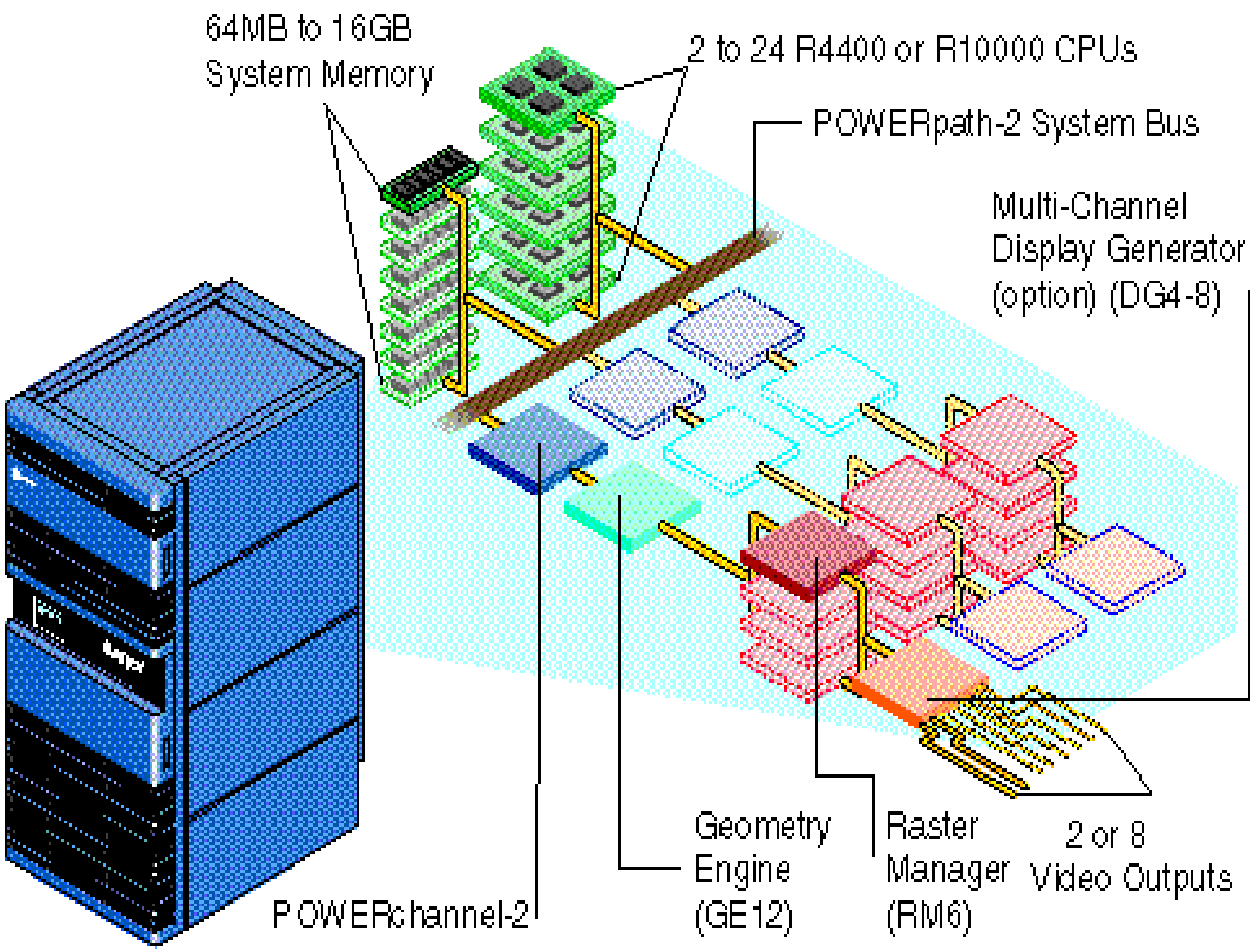
- Graphics „Supercomputer“
- PC with SLI or CrossFire

Different:

Multiple display support - (not) synchronized:

- PC with multiple unconnected cards
  - Nvidia Mosaic

Graphics Supercomputer



SGI Onyx with
Infinite Reality 3

SGI Onyx 3000 & Infinite Reality 4

G-Brick:

- 4 RasterManager Boards
- 1.3 Gpixel/s/Pipeline
  - (8 subsample/full scene/AA)
- 1 GB Texturspeicher
- 10 GB Framebuffer
- 192 GB/s Bandbreite
- Kombination bis zu 16 IR4

Nvidia Multi-GPU solutions

- Connected via PCIe to PC
- 2-8 GPUs
- 12 GB Frame Buffer per GPU
- 2 to 8 Dual-Link Digital Display Connectors
- Genlock/Frame Lock

Hannes
Kaufmann

Supercomputer – Application Areas

- Theme Parks (DisneyQuest – CyperSpace Mountain)
- Flight Simulators
- Military Applications
- CAVEs / Large setups

Barco RP-360
Flight Simulator (Video)

Graphics Supercomputer

- Multiple CPUs
- Multiple Geometry Engines
- Multiple Rasterization Engines
- Genlocking
- Multiple Pipes (=graphics cards)
- Multiple Channels (=display outputs)
- Highly configurable
- Now used: standard Nvidia/ATI graphics chips
- On PC: Scalable Link Interface (Nvidia) or CrossFireX (ATI) for PCI Express

Hannes
Kaufmann

Parallel Graphics Hardware

(A) Computing the same (high resolution) image

(B) Computing multiple images –
Multiple outputs

Hannes
Kaufmann

Basic Problems of Parallel Rendering

Vertex and Pixel Load Balancing:

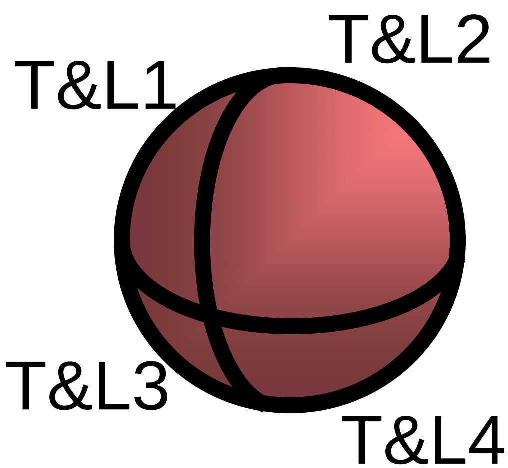
- Problem with parallel rendering
  - Load balancing of vertices
    → 3D (object space) problem
  - Load balancing of pixel (rasterizers)
    → 2D (screen space) problem

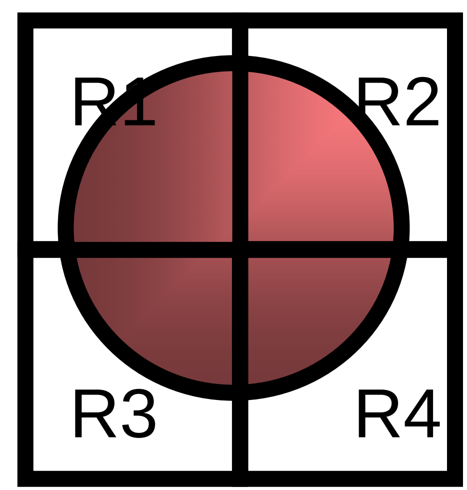
Parallel Rendering as Sorting

- Parallel Geometry Stage
  - Cut 3D model into pieces with equal number of vertices
  - Assign one piece to one T&L unit
- Parallel Rasterization
  - Cut destination image into tiles
  - Assign (triangles contained in) one tile to one rasterizer

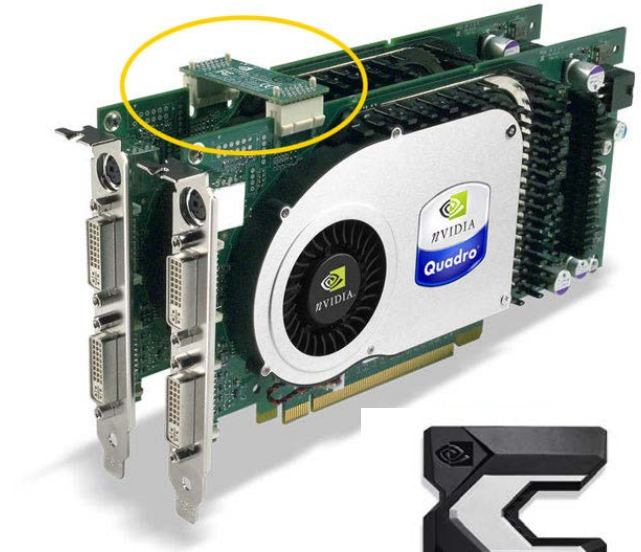
→ Need to SORT transformed 2D triangles

- Shared common memory





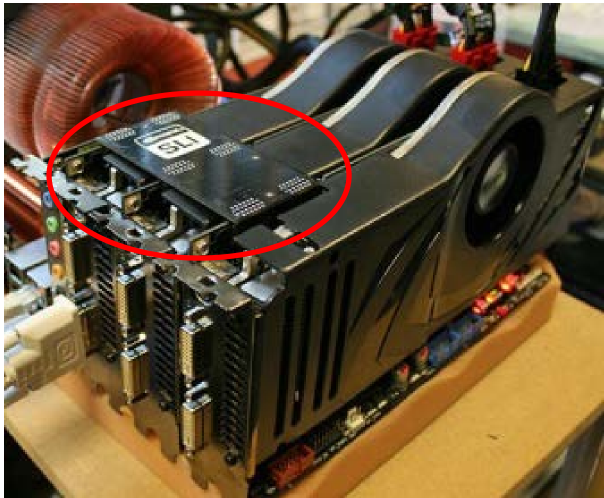
SLI / NVLink (Nvidia)



- Scalable Link Interface

3 Modes:

- Split Frame Rendering (SFR) - Scissors: Splits each frame and sends half the load to each of the graphics cards
- Alternate Frame Rendering (AFR):
  Frame 1 – Card 1, Frame 2 – Card 2, alternating
- VR SLI: Right/Left frame computed on Card 1/Card2 in parallel



- PCIe cards are connected by a bridge – very fast data transfer
- Optimal performance increase: 1,8 max.

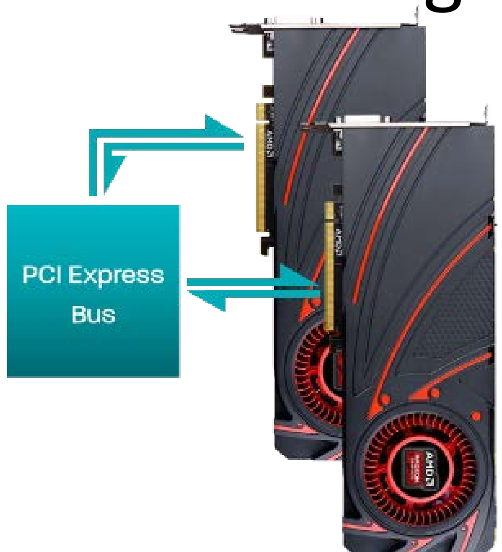
CrossFireX (AMD) / XDMA

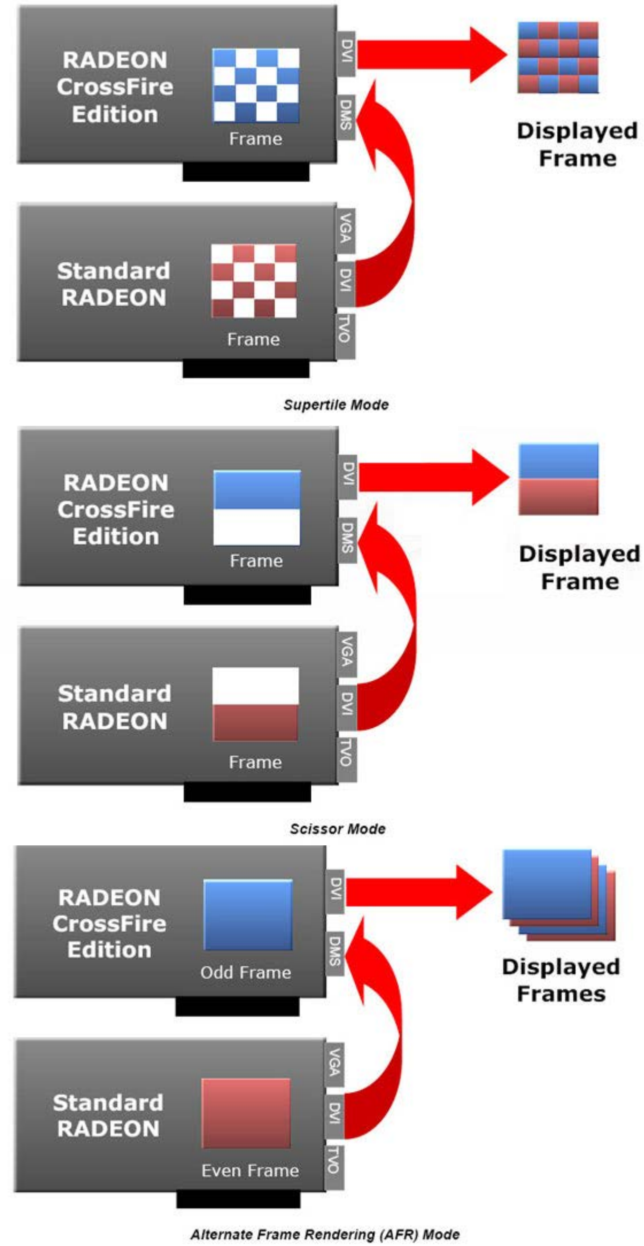
3 Modes:

- Supertiling
- Scissors
- Alternate Frame Rendering

Additional AA Mode

No external Bridge needed!



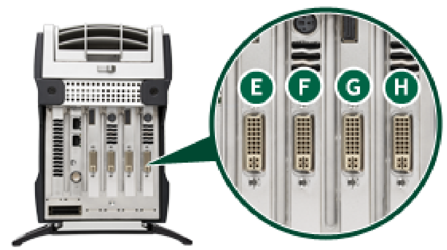


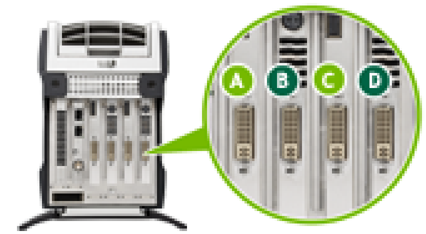
SLI / CrossFire

- Mainboards need to support it
- Connection via separate bridge (PCIe communication) – only for Nvidia.
- AMD used XDMA transfer over PCIe
- CrossFire SuperTiling efficient
- CrossFireX more flexible (supports multiple displays and connection of different AMD cards)
- 2-4 cards can be connected
- VR SLI Mode: Parallel left/right image

Nvidia Mosaic

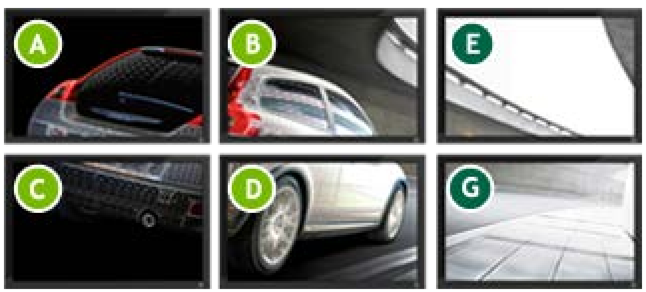
multiple display configurations with Quadro cards





4-Display Connections for 2-Display Passive

2x3 Configuration



Left Eye

Right Eye

2x4 Configuration

Dual Host Interface Card required to run dual systems.

Hannes
Kaufmann

Parallel Graphics Hardware

Types of parallel graphics:

1. On-chip / on a graphics board (standard)
2. Multiple boards (former: graphics supercomputer)
   Multiple boards with multi GPUs (1+2)
3. PC cluster:
   - **Offline Rendering: Standard network – Distributed Environment**
   - Realtime Rendering: PC cluster with special hardware

Parallel Cluster Rendering (1)

- PC Cluster
  - Off-the-shelf hardware
  - Network (LAN)
  - Cheap
  - Scalable
- Distributed Software System

Parallel Cluster Rendering (2)

- power of cluster ≥ power of supercomputer
- Price of cluster << price of supercomputer
- BUT: problems of cluster
  - How to make cluster PCs work together
  - On a single image
    (or consistent set of images)

→ Parallel Execution of Rendering !

→ Cluster synchronisation (genlocking) !

Cluster Synchronisation

3

Q: How to synchronize multiple displays?

(1) Simple: PC + Multiple graphic outputs

(2) Not so simple: Multiple workstations

Parallel Graphics Hardware

Types of parallel graphics:

1. On-chip / on a graphics board (standard)
2. Multiple boards (former: graphics supercomputer)
   Multiple boards with multi GPUs (1+2)
3. PC cluster:
   - Offline Rendering: Standard network – Distributed Environment
   - **Realtime Rendering: PC cluster with special hardware**

Example: CAVE

“Computer Assisted Virtual Environment” ™

- Has 3 to 6 large screens
- Puts user in a room for visual immersion
- Usually driven by a single or group of powerful graphics engines – nowadays usually PC cluster

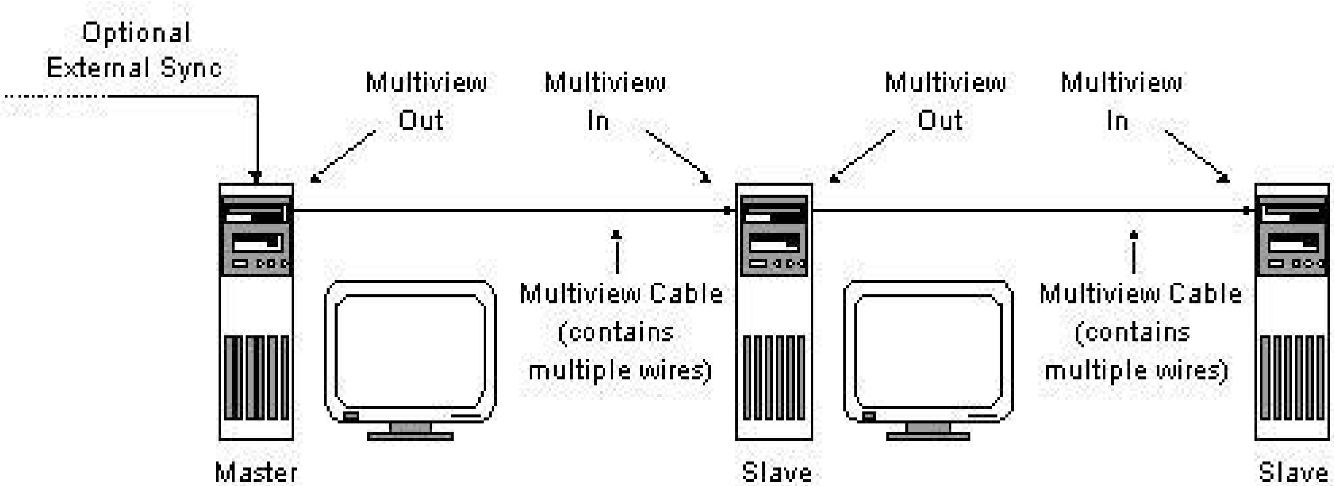
Hannes Kaufmann

Example: CAVE & Shuttering

Shutter Glasses

Hardware Synchronisation

Synchronizing multiple displays/workstations



Framelock:

Synchronizing frame buffer swap

- Begins redrawing at the same time

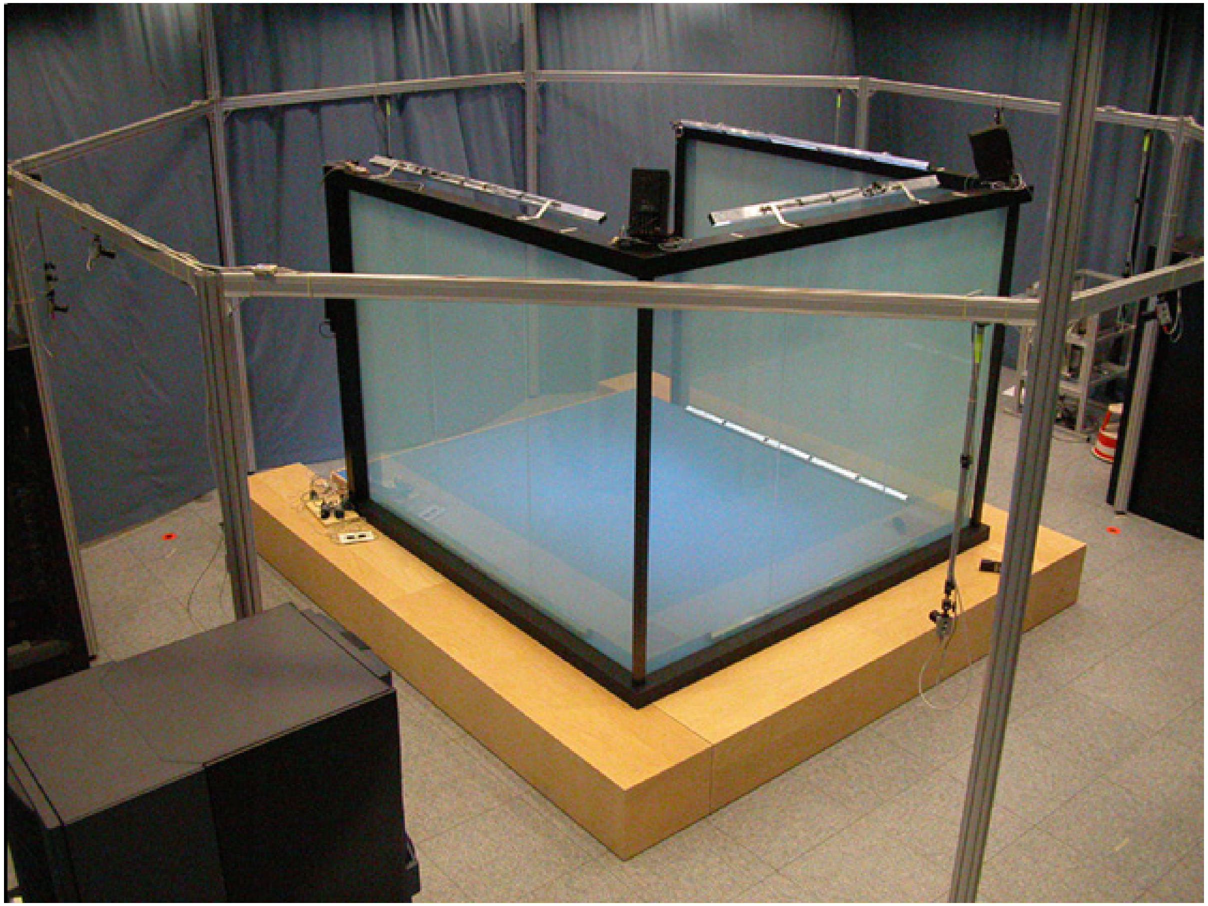
Genlock:

Exact synchronization of vertical synch (electron beam of CRT)

- Refreshes each pixel synchronously

Hannes
Kaufmann

Example: Blue-C



Hannes Kaufmann

3D Card High End Model

- nVidia Quadro RTX 8000 (~ € 10.000.-)
- 48 GB GDDR6 RAM
- Based on Turing Architecture (Geforce RTX 2080)
- 4608 CUDA cores, 576 Tensor cores
- 672 GB/s Bandwidth
- optimized OpenGL drivers (comp. to consumer card)
- 16K x 16K texture resolution
- DX12, Shader Model 6.1, OpenGL 4.6, Vulkan 1.1.78

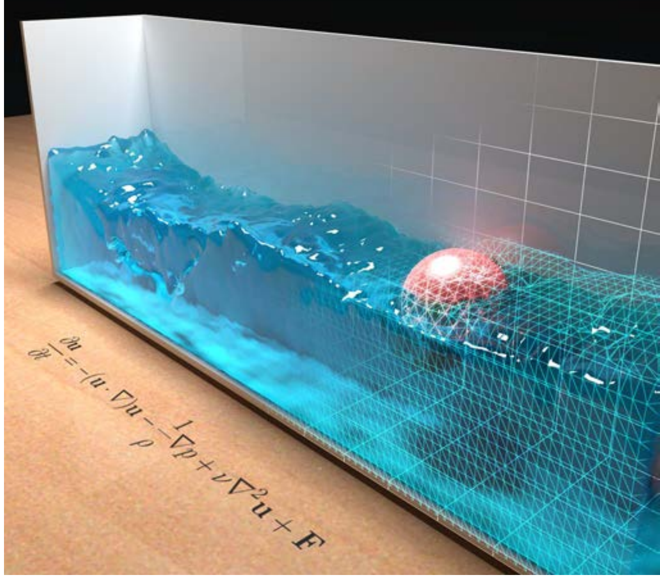
Hannes
Kaufmann

Some Relevant Features (for VR)

- Memory size: 48 GB
- 4 DisplayPorts 1.4 (8K@60Hz or 4K@120Hz), 1 VirtualLink (1 connector for VR)
- OpenGL quad-buffered stereo (optional 3-pin sync connector); 3D Vision Pro
- NVLink Technology
- Nvidia Mosaic: 2-8 displays
- Fast 3D Texture transfer; HW 3D Window clipping
- Quadro-Sync (optional) with Framelock and Genlock
- HDR technology, 30-bit color, SDI output option
- Quality: 64 x Full-Scene Antialiasing (FSAA), …

Hannes
Kaufmann

Physics Effects

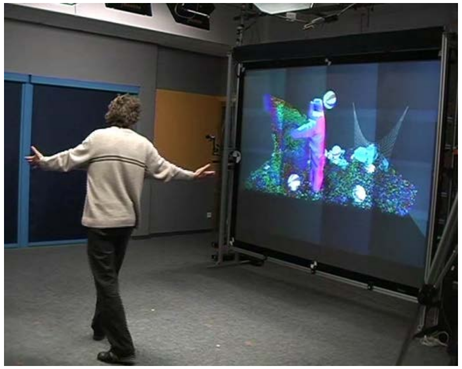


- Calculation on GPU
- Rigid Bodies, Joints
- Cloth, Particles, Fire, Fluids
- Puts higher rendering load on graphics card
  - SLI recommended

Hannes Kaufmann

Physics in VR

GRIMAGE Project



Incredible Machine

Microsoft Holodesk

General Purpose Computing

- Nvidia TESLA V100
  - „High Performance Computing“ / Deep Learning
  - No graphics card! No graphics output!
  - Programmed using CUDA
  - Additional GPU
  - 5120 CUDA cores
  - 640 Tensor Cores
  - 16 GB HBM2 RAM
  - CUDA C/C++/Fortran, OpenCL, DirectCompute Toolkits, ...

- Alternative: Intel Xeon Phi
  - x86 cores (72 Atom cores)

Nvidia GRID

- GPU Virtualization – sharing the GPU
- Low latency remote display
  - „Real time“ H.264 encoding
- Grid K2:
  - 2 Kepler GPUs, 3072 cores
  - 8GB RAM

Hannes
Kaufmann

Literatur

- Real-time Rendering
  Tomas Akenine-Möller, Eric Haines, and Naty Hoffman, 1045 pages, from A.K. Peters Ltd., 3rd edition, ISBN 978-1-56881-424-7, 2008
- http://www.realtimerendering.com/