GPU Programming

Introduction

Thorsten Grosch

TU Clausthal

Welcome…

- …to the first lecture GPU Programming!

- Today
  - Organization
  - Overview of the lecture
  - Look back the GPU history
  - Look into some libraries

The Team

Lectures:
Prof. Dr. Thorsten Grosch

Exercises:
M. Sc. Johannes Jendersie

About this lecture

- Programming the Graphics Hardware

- Basically two sections
  - First half: Improved Rendering (OpenGL 4.4 Pipeline, Shader programming)
  - Second half: General Purpose Computations on the GPU with OpenGL, parallel programming

  - Lecture: Monday, 10:30 – 12:00, D5-101 Multimedia Room
  - Exercises: Monday, 9:00 – 10:30, IfI Room 116

Prerequisites

- Computer Graphics (Computergrafik 1)
  - Rasterization, Transformations/Projections, Vertex Array Objects, Textures, Vertex/Fragment Shader…
- Programming
  - C++
- Current graphics card required (OpenGL 4.4)
    - IfI Room 116: 7 Computers with NV 680 GPU

Web

- Slides, Exercises, Programs are available from StudIP

Exercises

- One exercise each week
  - Essential for understanding GPU programming
  - See StudIP information
  - Mainly programming exercises, only a few theoretical questions
  - First exercise on Monday 8.11.
  - Present your solution one week later (Mondays 9:00–10:30) in room 116
  - Teams of two students
- 7 computers with NV 680 graphics cards in room 116
  - Exercises Monday 9:00–10:30
  - Windows, MS Visual Studio

Exercises

- 50% of the exercise points are required for the exam
  - Oral exam
  - Exam date(s) will be defined in a few weeks
  - Questions about lecture **and** exercises
- Bonus points if more than 80%

- Most of the information required for the exercises are in the slides, but not everything
  - Search additional information in the web
  - Make sure that your solution is compatible with OpenGL 4.4

Which Hardware is required ?

- Graphics card that supports OpenGL 4.4, at least one of these should work:
  - NVidia GeForce 440+
  - AMD Radeon HD 5450+
  - Intel HD Graphics 5300+ (Broadwell)

  - See here for a list:

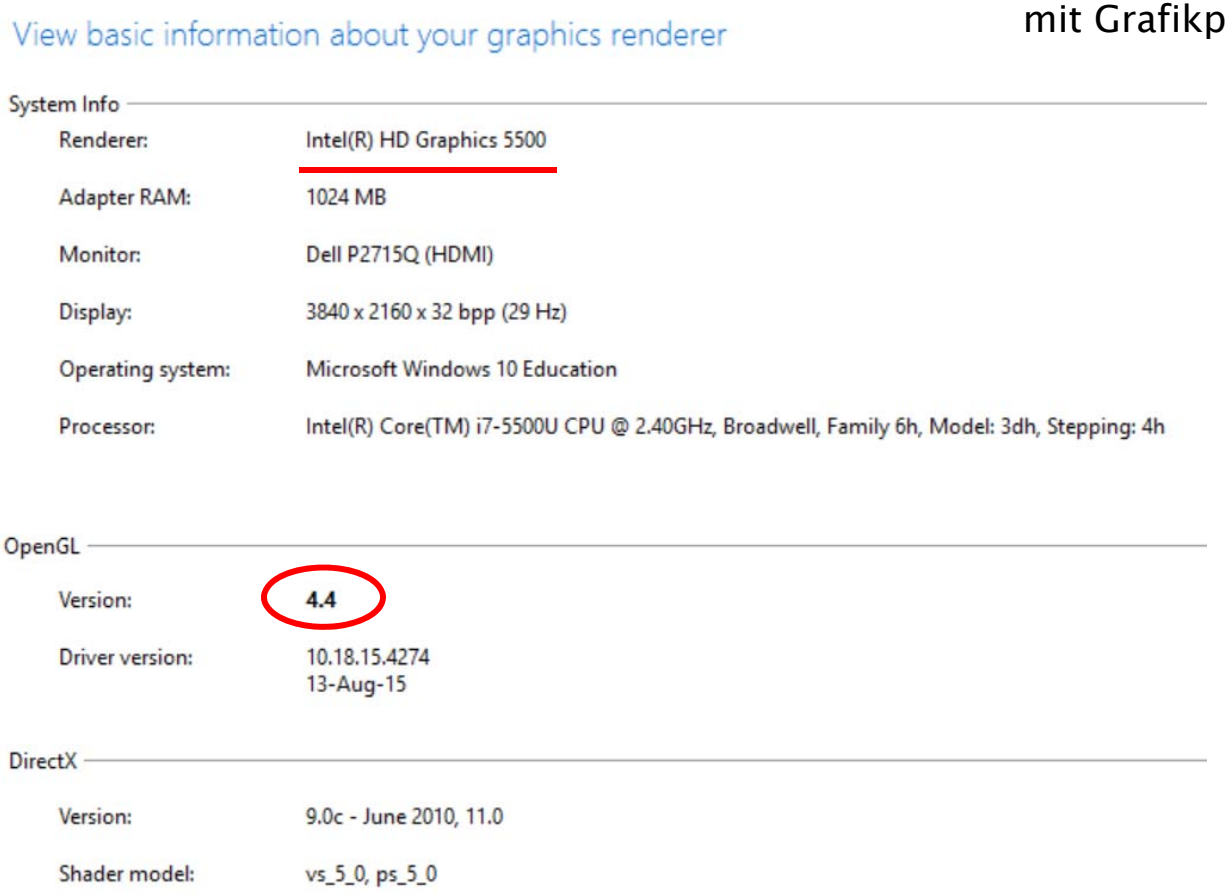
    http://opengl.gpuinfo.org/

- Or use the GL Extensions Viewer to check which OpenGL Version is supported by your hardware

  www.realtech-vr.com/glview/
- Check also that recent graphics drivers are installed

OpenGL Viewer Output

Hint:
Some laptops have
two graphics cards/chips
Use right mouse button →
mit Grafikprozessor ausführen

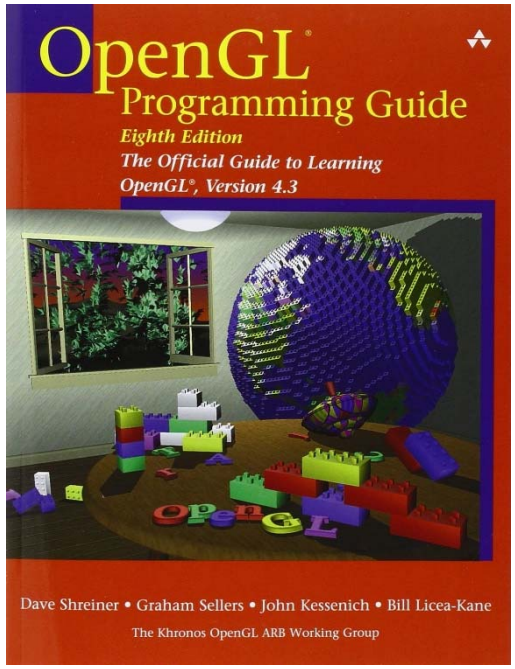


Hardware for Exercises

- IfI Room 116
  - 7 machines with current graphics hardware
  - Visual Studio 2015 installed

- Own computer/laptop
  - Install Visual Studio Community Edition
  - Install all the required libraries → see StudIP for instructions in a few days

Literature

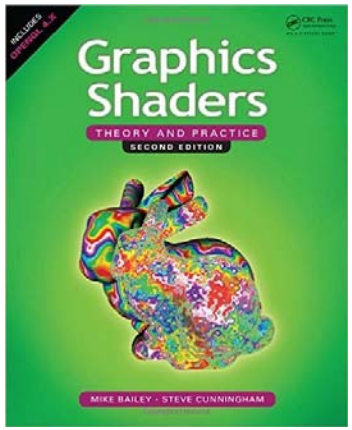


OpenGL Programming Guide
8th Edition
Dave Shreiner et al.
Addison Wesley

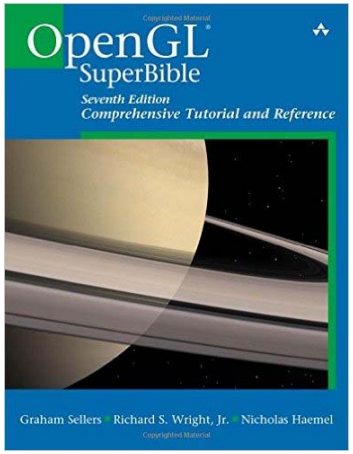
- The main source of information:
  - The „Red Book“
  - We use the 8th editon for this lecture
  - Available in our library
- Main web source
  - www.opengl.org
  - https://www.khronos.org/files/opengl44-quick-reference-card.pdf

Other Literature

Real-time Rendering
3rd Edition
T. Möller, E. Haines, N. Hofmann
AK Peters



Graphics Shaders
2nd Edition
M. Bailey, S. Cunningham
AK Peters



OpenGL SuperBible
7th Edition
G. Sellers et al.
Addison Wesley

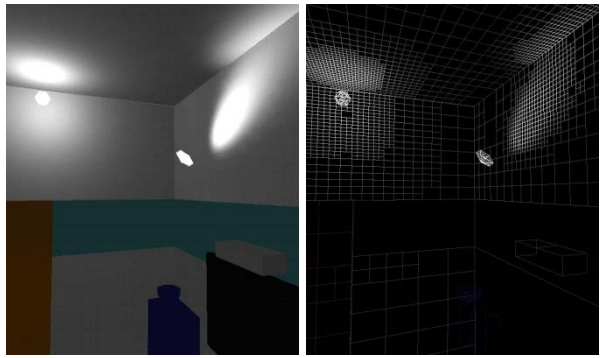
GPU Gems 1 – 3
ShaderX 1 – 7
GPU Pro 1 – 7

Who am I

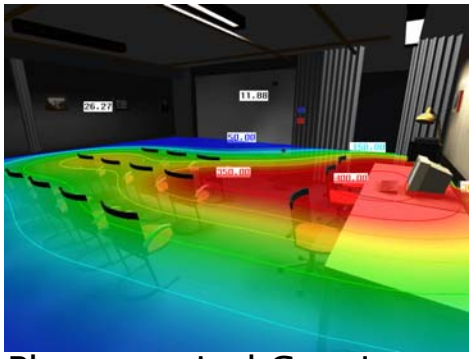
- Thorsten Grosch
- Since September 2015 Professor for Graphical Data Processing and Multimedia at TU Clausthal
  - Study Informatics TU Darmstadt
  - Fraunhofer IGD
    - Global Illumination (Radiosity)
  - University of Koblenz Landau
    - creating Computer Graphics lectures
    - Dissertation
      - Augmented Reality with consistent Illumination
  - Post-Doc at MPI Informatik Saarbrücken
  - Juniorprofessor for Computational Visualistics in Magdeburg

IGD Darmstadt

- Study Informatics TU Darmstadt
- Fraunhofer IGD Darmstadt
- Radiosity
  - Finite-Element method for global Illumination
  - Physically correct for diffuse Environments



Refinement



Photometrical Consistency

Illumination inside a Car

University of Koblenz-Landau

- 2002 change to Uni Koblenz together with Prof. Dr. Stefan Müller
- Creation of new Computer Graphics …
  - Lectures
    - Computer Graphics 1 + Exercises
    - Computer Graphics 2 + Exercises
    - Photorealistic Computer Graphics + Exercises
  - Seminars, Study/Diploma Theses, Praktika, …
- Dissertation
  - Augmented Image Synthesis

Augmented Image

Original Photograph

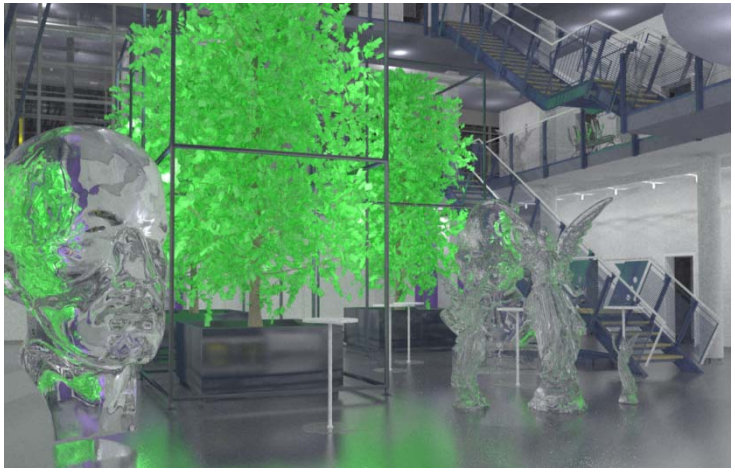
MPI Informatik

- Post-Doc
- Research topic
  - Real-time Global Illumination on the GPU
- Projects
  - Coherent Shadow Maps
  - Imperfect Shadow Maps
  - Screen-Space Directional Occlusion

MPI Informatik

Imperfect Shadow Maps: Ritschel, Grosch, Kim, Seidel, Dachsbacher, Kautz

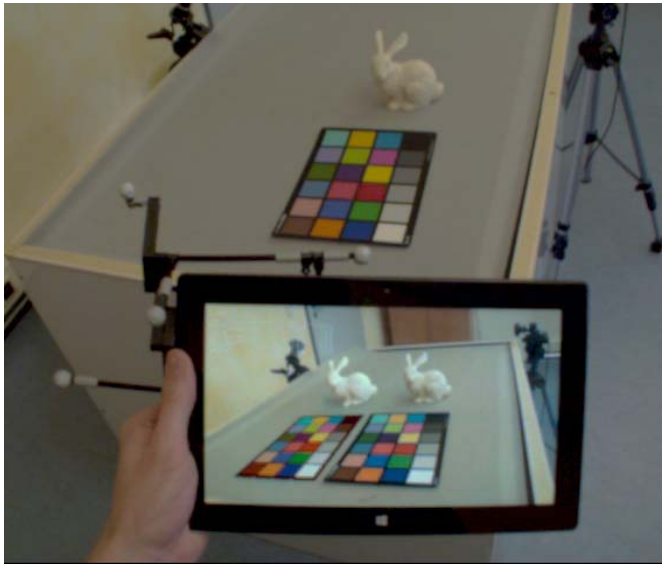
Current Research



~ 90 Million Triangles

- Global Illumination of large scenes on the GPU
  - Distributed Out-of-Core Stochastic Progressive Photon Mapping

- Interacting with Photorealistic Augmented Reality
  - Distributed Illumination on a PC with GPU and a mobile Device



~ 20 Bilder / Sek. (Surface RT)

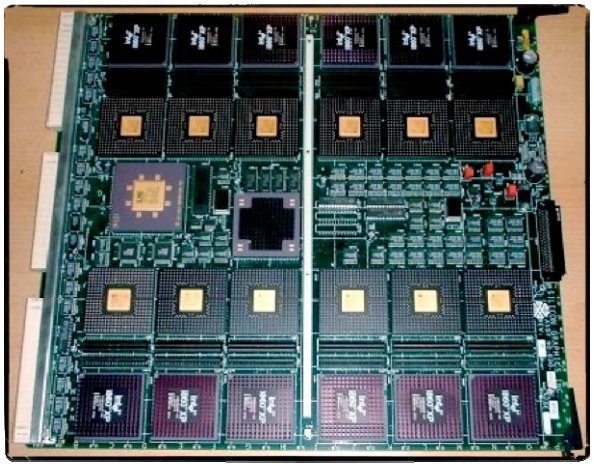
GPU History

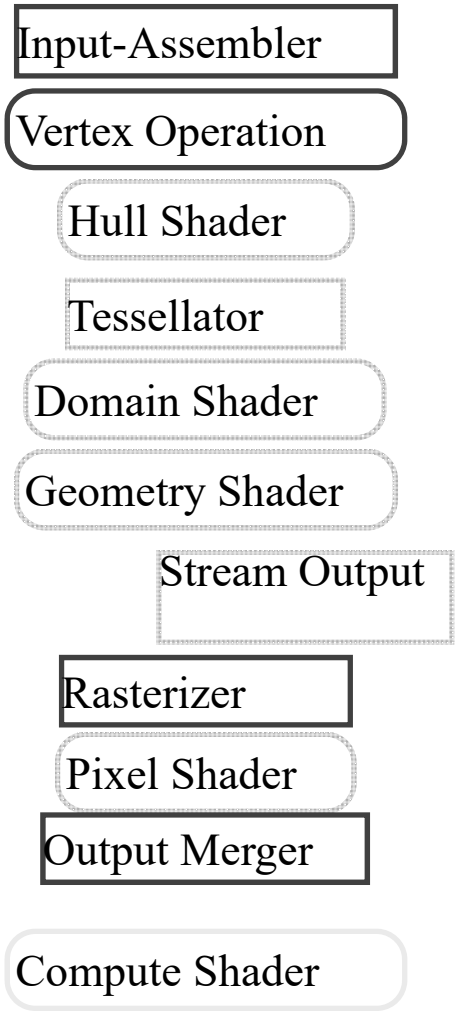
Thanks to John McLaughlin

TU Clausthal

Early Graphics Chips

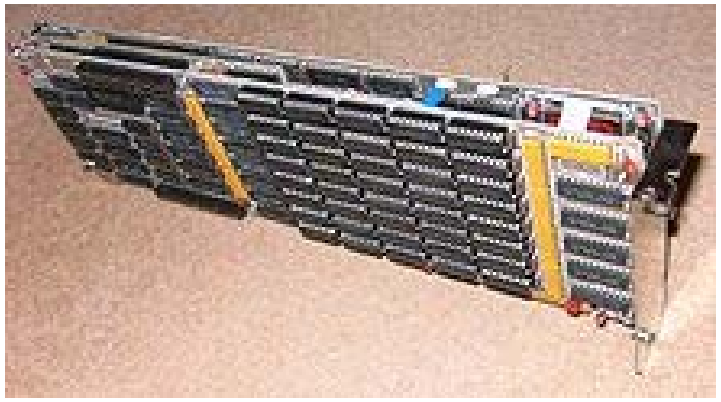
- ANTIC (Atari 8-bit) – ca. 1979
  - Text and graphics
- Geometry Engine – SGI, ca. 1982
  - Matrix transformations
  - Clipping
  - Mapping to output
  - SGI founded (Jim Clark)





Early Graphics Chips

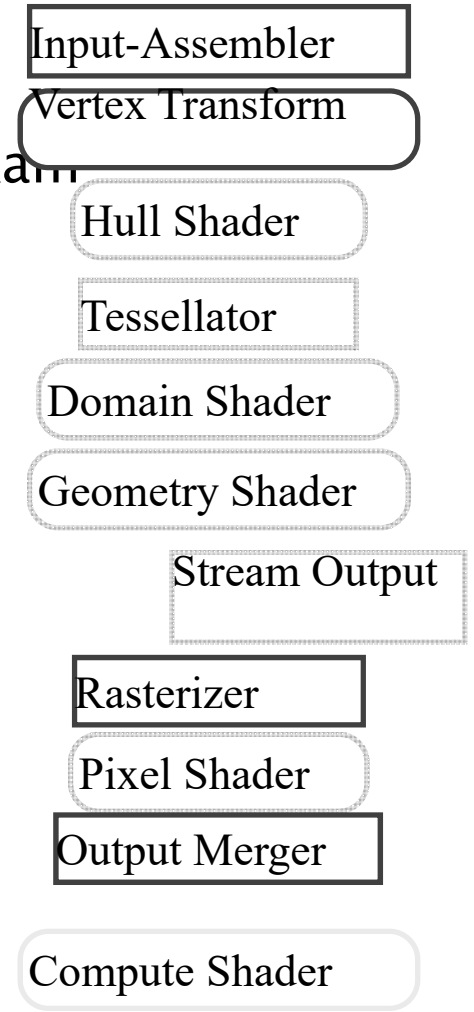
- Professional Graphics Controller – IBM ~1984
  - 640*480; 256 colors (palette) at 60FPS
  - 320kb RAM, Intel 8088 (8Mhz)
  - 3D Rotation and image clipping
  - $4290, quite cheap



Input-Assembler
Vertex Transform
Hull Shader
Tessellator
Domain Shader
Geometry Shader
Stream Output
Rasterizer
Pixel Shader
Output Merger
Compute Shader

Early Graphics Chips

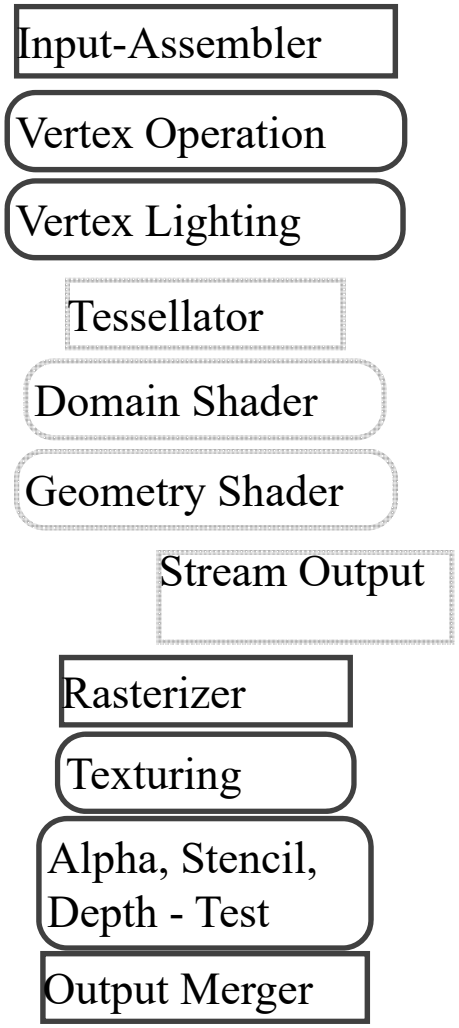
- SGI Iris 1400 (1984)
  - 3D Workstation, UNIX-Variant
  - Motorola 68010 CPU, 10Mhz, 1.5 MB Ram
  - 1024x1024, 256 colors
  - ca. 60.000 $



Early Graphics Chips

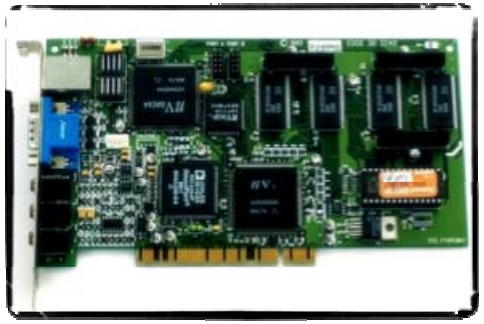
- SGI RealityEngine 1992
  - >1 mio. triangles / second
  - 100.000 $
- IRIS GL opened by SGI
  - Becomes known as OpenGL (ca. 1992)
  - cross-platform graphics development for the first time

TU Clausthal

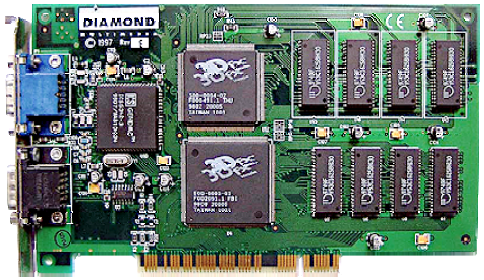


Early Graphics Chips

- NV1 – Nvidia 1995
  - Does not support the 'new' DirectX
- Voodoo – 3dfx 1996, ca. 300$
  - First pure 3D accelerator
  - 16-Bit, Texture Filter, Z-Buffer
  - 800x600, ca. 1 mio triangles / second
- Glide API (3dfx) preferred by game developers
  - OpenGL has performance problems (driver)



Nv1 - Nvidia



Voodoo - 3dfx

Input-Assembler

Projektion

Hull Shader

Tessellator

Domain Shader

Geometry Shader

Stream Output

Rasterizer

Texturing

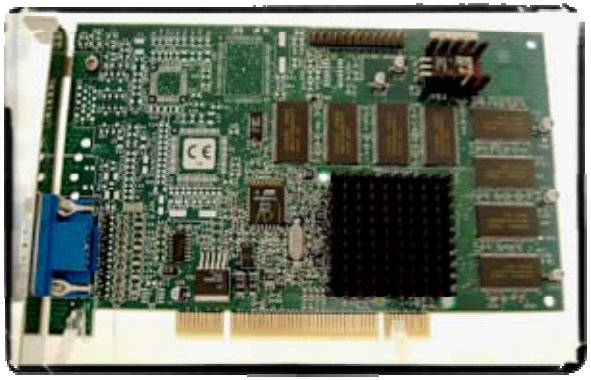
Alpha, Stencil, Depth - Test

Output Merger

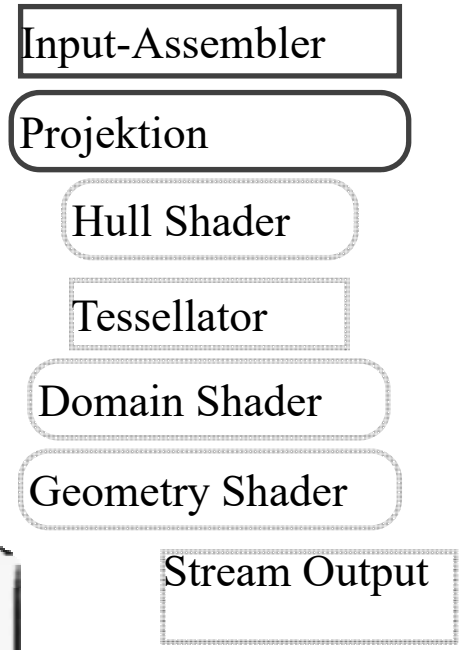
Early Graphics Chips

- Voodoo 2/3 – ca. 1998
  - Only limited AGP support
  - no 32-Bit
  - Textures max. 256x256
  - Max. 16MB Ram
  - In the beginning no 2D graphics
- Riva TNT2 – Nvidia März 1999
  - Nvidia replaces 3dfx dominance (2000)
  - 32bit, Z-Buffer, stable drivers

Riva TNT2 - Nvidia

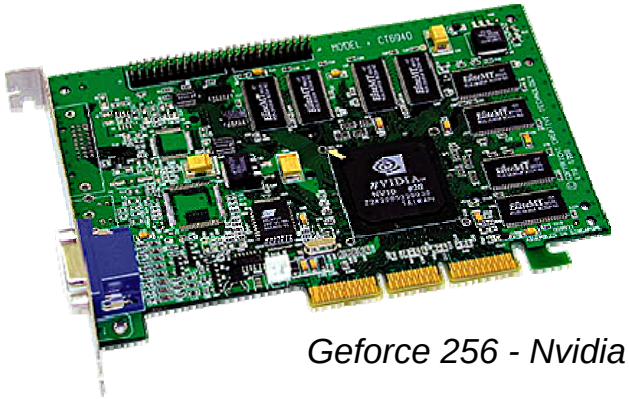


Voodoo 3 - 3dfx

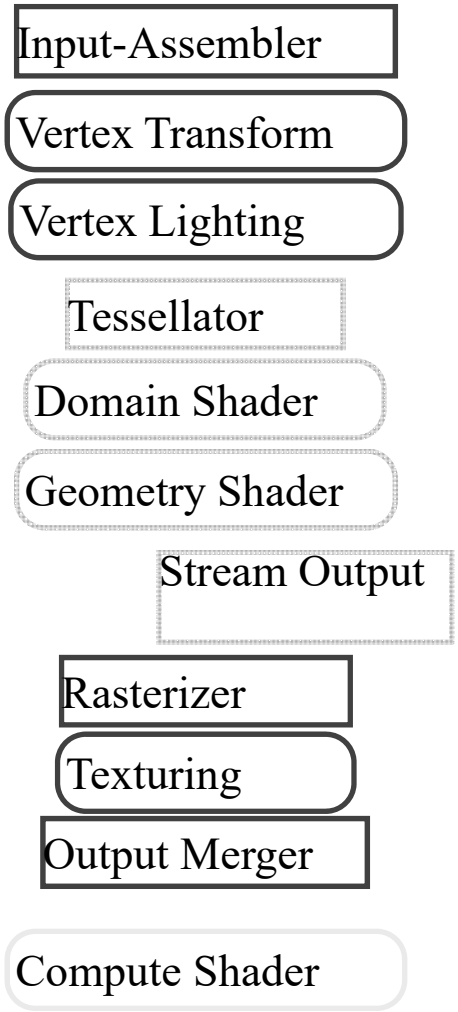


First GPU

- Geforce 256 (NV10) – Sept. 1999
  - Nvidia defines the "GPU"
  - Hardware Transformation and Lighting (Hardware TnL)
  - Previous versions of Vertex and Pixel Shader units
  - 15Mio Polygons / second
  - 480Mio Pixel / second
  - Up to128MB Ram

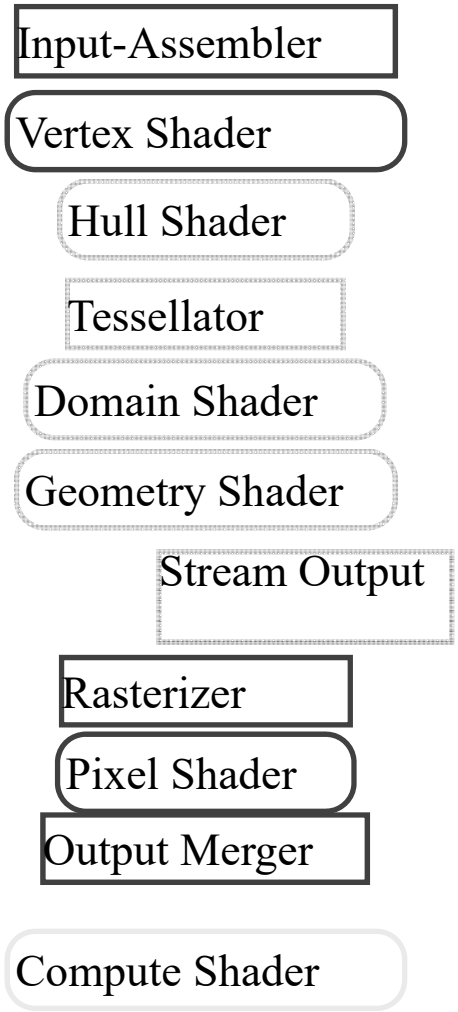


Geforce 256 - Nvidia



Programmable Hardware

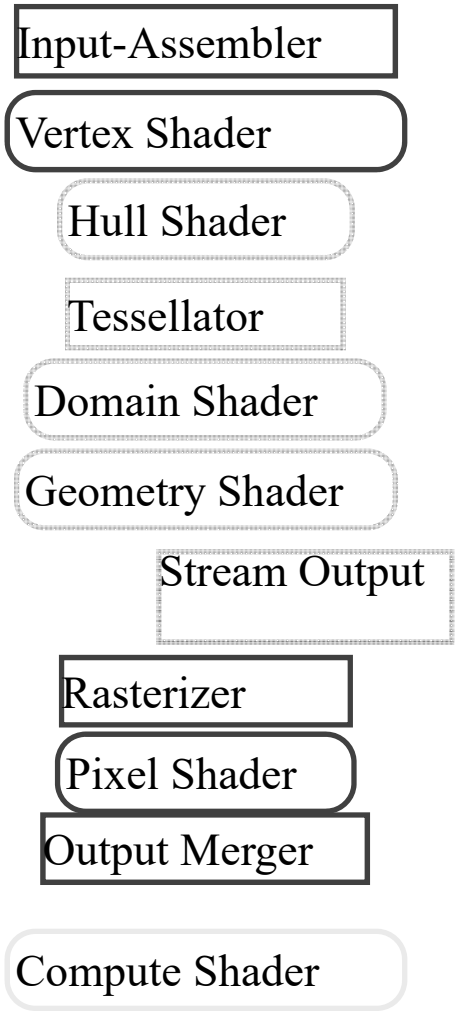
- SGI ignores 3D cards for gamers
  - cheap ATI and Nvidia cards for professional development
  - Financial problems at SGI, today they only work in the area of servers
- ATI and Nvidia do longer orient at the SGI-defined pipeline
  - Programmable Shader



Programmable Hardware

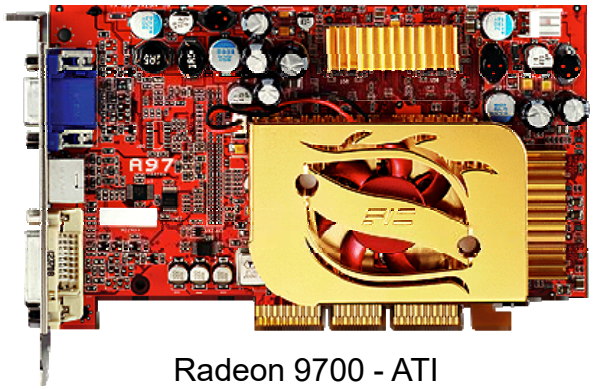
- Geforce 3 (NV20) – 2001
  - first vertex and pixel shader programming
  - Developers can implement features directly for the first time
- DirectX8 replaces OpenGL dominance (Gaming)
  - Fort the first time newer features than OpenGL
  - first Vertex and Pixel Shader support

Geforce 3 - Nvidia

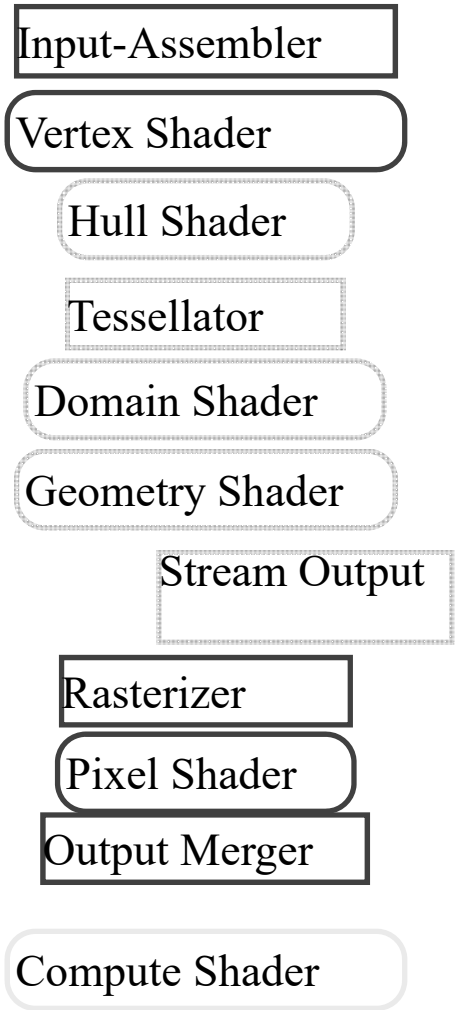


Programmable Hardware

- ATI Radeon 9700 (R300) – 2002
  - first Direct3D 9.0 card
  - Replace Nvidia Geforce4 dominance
- DirectX9 – 2002
  - HLSL (High-Level Shader Programming)
  - OpenGL introduces GLSL as alternative in 2004
  - Shader Model 2.0
  - DirectX dominates the games market



Radeon 9700 - ATI



Programmable Hardware

- Nvidia Geforce 8 Series – 2006
  - First consumer GPU with Direct3D 10 support
  - unified shaders introduction
    - Vertex and Pixel Shader were separate before
    - Thread Architecture → CUDA
  - Shader Model 4.0
    - Geometry Shader

Geforce 8800 GTX - Nvidia

Input-Assembler

Vertex Shader

Hull Shader

Tessellator

Domain Shader

Geometry Shader

Stream Output

Rasterizer

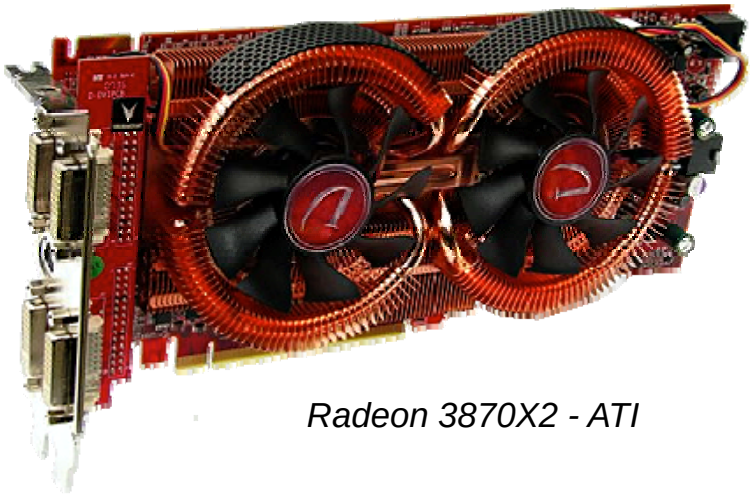
Pixel Shader

Output Merger

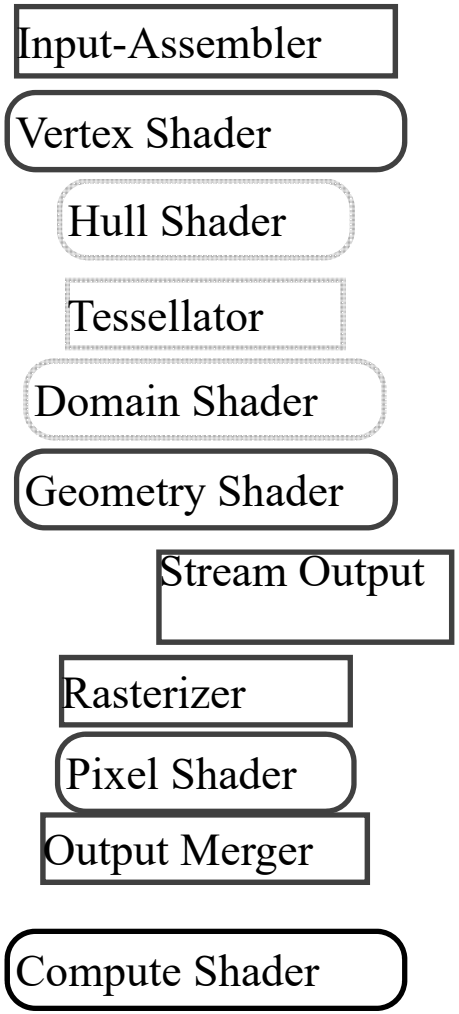
Compute Shader

Programmable Hardware

- Ati Radeon R600 – 2007
  - Based on Xenos GPU (Xbox 360)
  - Unified Shader Architecture
  - DirectX10, Shader Model 4.0, OpenGL 3.0

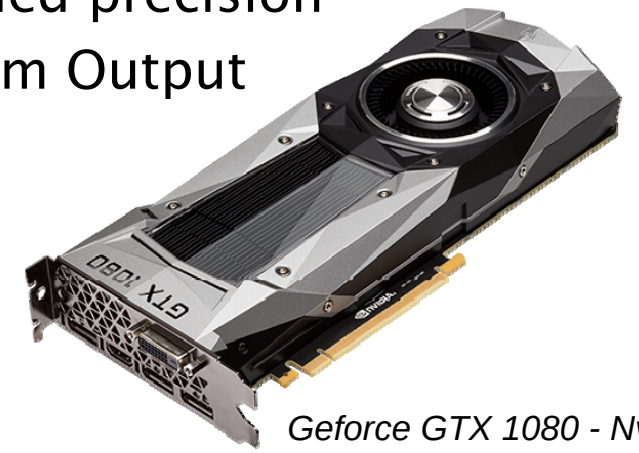


Radeon 3870X2 - ATI

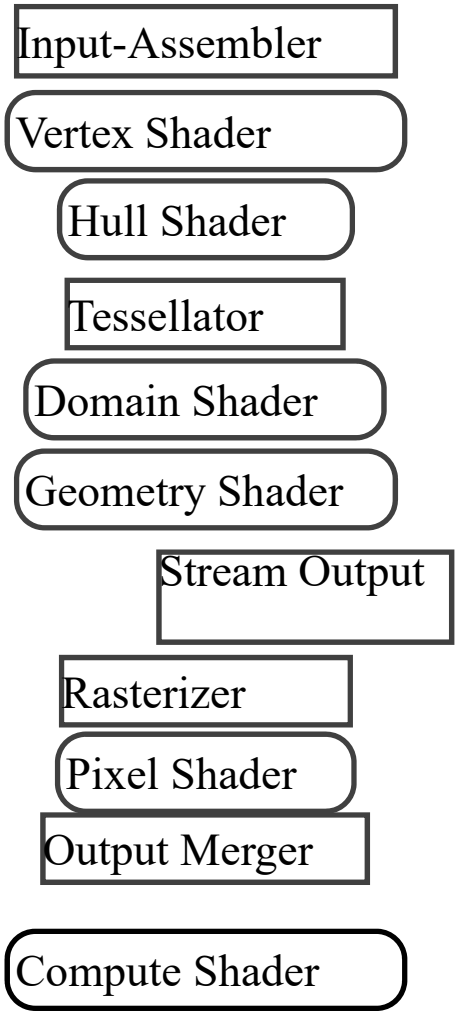


Today

- DirectX 12
  - Object-Orientierted Programming in the Shader
  - Tessellation
  - Multi-Threading
- GPGPU – “Compute Shader”
  - Arbitrary computation
  - Defined precision
  - Stream Output



Geforce GTX 1080 - Nvidia

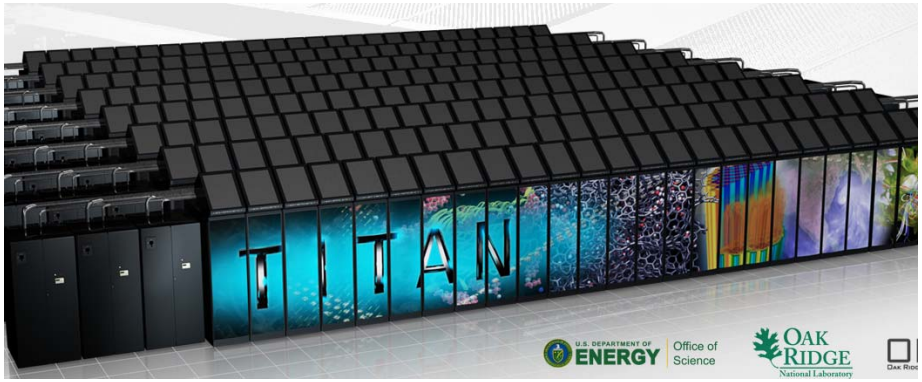


Multi-GPU Systems

- Link several GPUs together
  - e.g. NVIDIA SLI (Scalable Link Interface)
    - Alternate/Split Frame Rendering
  - Tesla

GPU Cluster

- Titan
  - Oak Ridge National Laboratory (ORNL), Tennessee, USA
  - 18.688 NVIDIA Tesla K20 GPUs
  - 299.008 Opteron Cores
  - 710 Terabytes
  - 20 Petaflops ($20 \cdot 10^{15}$ FLOPS)



- Currently number 3 in the top ten of Super computers
  - Simulations for climate change, energy, materials, …
- Current top: Sunway TaihuLight (China)
- (June 2016)

Lecture Overview

TU Clausthal

Basics

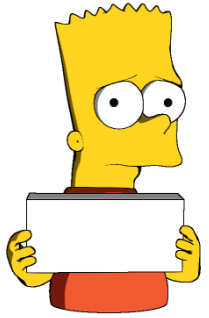
- The Rendering Pipeline
  - The way from vertices to pixels
- Mathematics
  - Transformations
  - Projections
  - Viewport
- Textures
- Buffer Objects
  - Vertex Buffer Object
- We will repeat several things from Computer Graphics 1 in the next 2-3 weeks

Advanced OpenGL Programming

- Blending
- Logic Operations
- z Buffer
- Stencil Buffer
- Rasterizer Options
- Sampler States

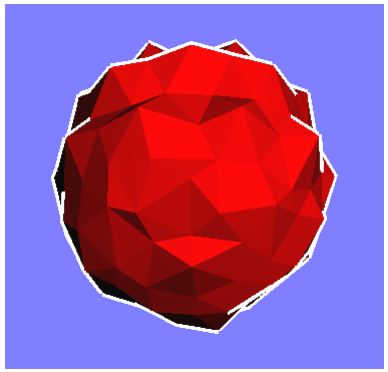
Shader Programming

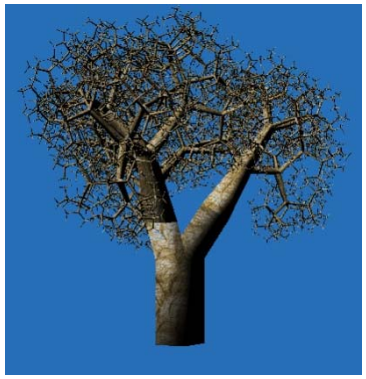
- Vertex – und Fragment Programs
- OpenGL Shading Language (GLSL)
- Multi-Pass Rendering
- Multiple Render Targets
- Geometry Buffer (G Buffer, Deep Framebuffer)



Shader Programming

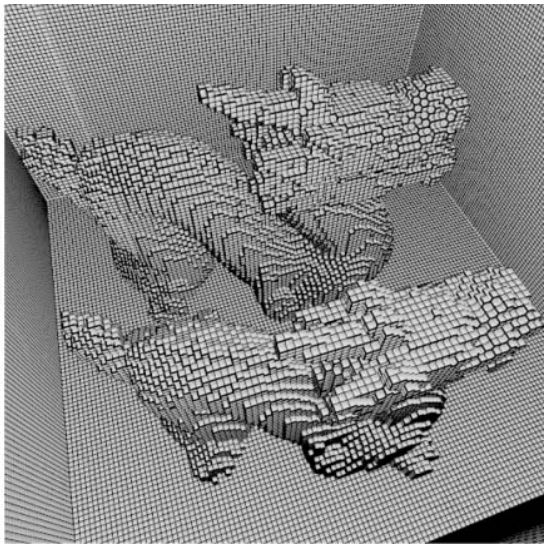
- Geometry Shader
- Transform Feedback
- Tessellation Shader





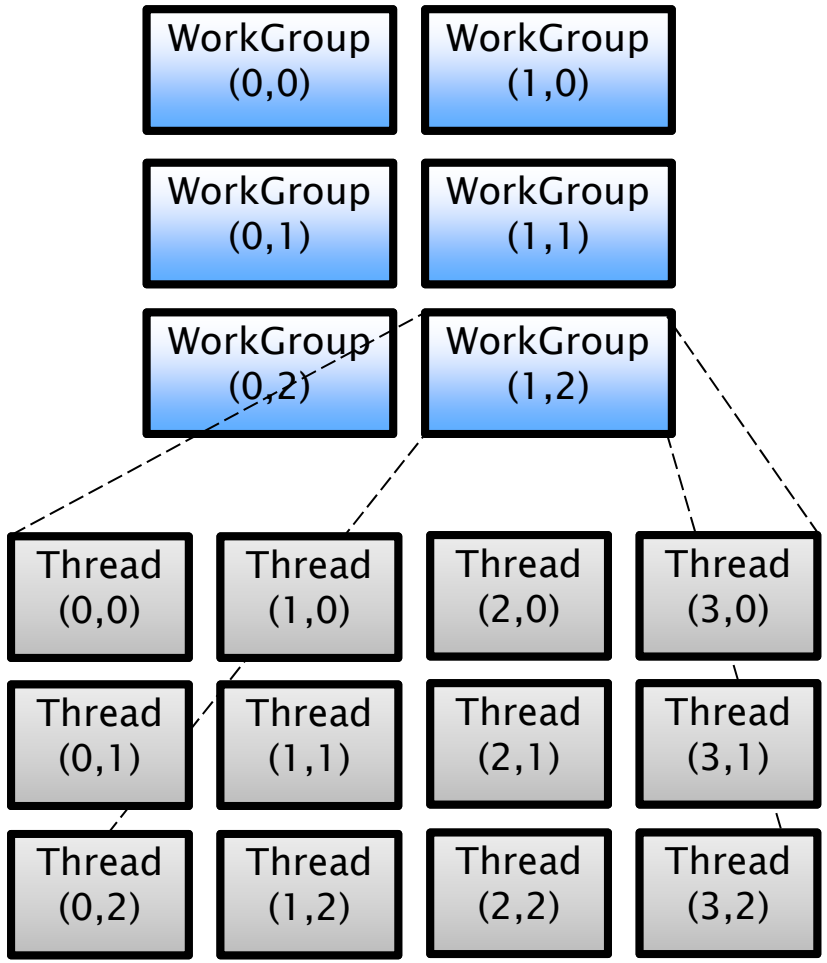
General Purpose GPU

- Use the GPU not only for Rendering
  - E.g. Voxelization



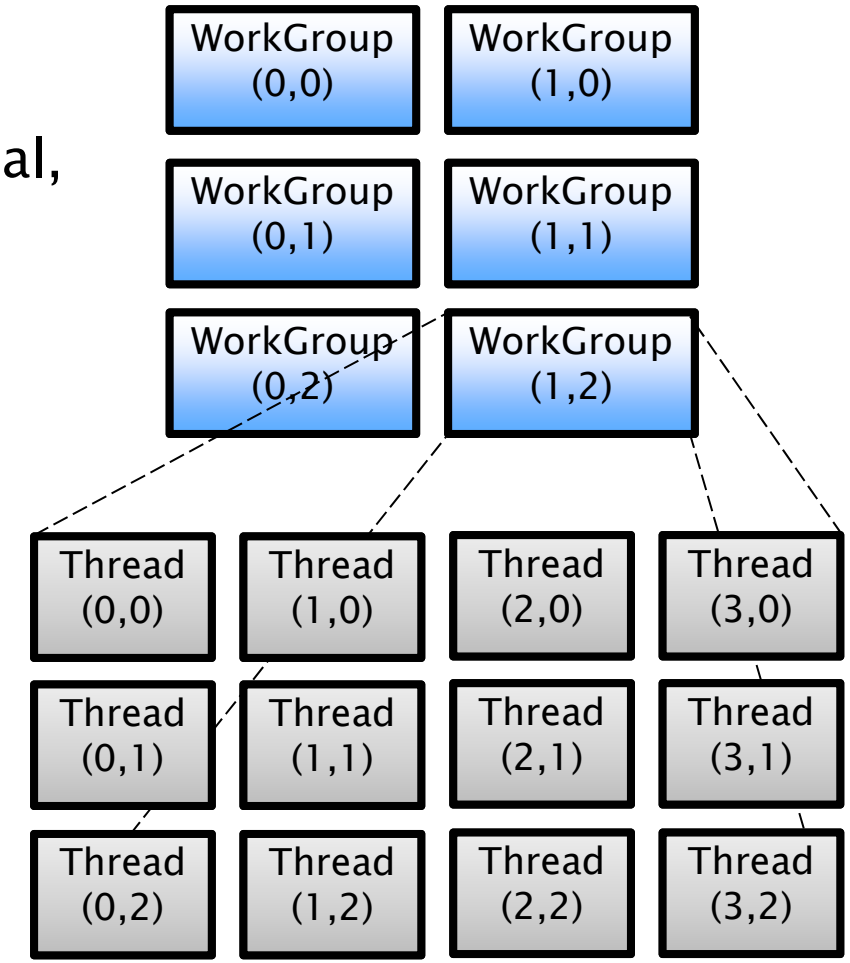
Parallel Programming with Compute Shaders

- GPU = Parallel processor
- Programming model
  - Grid, WorkGroup, Thread
- Basics
  - Functions and Variables
  - Kernel Functions
  - „Hello World“ Program
  - e.g. Add two large vectors in parallel, dot product, image filtering, …



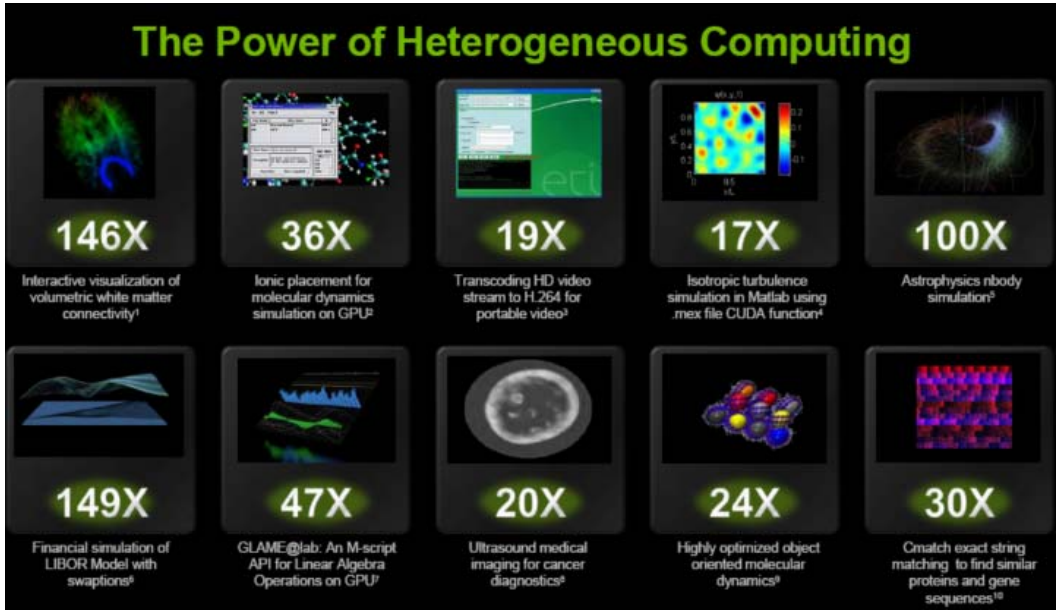
Parallel Programming with Compute Shaders

- Memory management
  - Texture, Image, Shared, Shader Storage Buffer, Local, Register
- Threads and Warps
- Thread Synchronisation
  - Barrier
- Atomic Operations



Parallel Physical Simulations

- Parallel solving of differential equations
  - N-Body
  - Deformations
  - Heat distribution
- Global Illumination
- Can often be parallelized



Source: NVIDIA

Parallel Programming Techniques

- Parallel Sorting
  - Bitonic Merge Sort
- Parallel Memory Access
- Scan

Approaching Zero API Overhead

- Many draw calls with low overhead!

- Instancing
- Indirect Draw
- Uniform Ring Buffers
  - Persistent Mapping
- Bindless Textures

- Questions about exam

Vulkan Overview

- Low level API
  - First Triangle OpenGL ~300 LoC vs Vulkan ~2500 LoC
  - Almost nothing done by the graphics driver

- Differences and benefits:
  - Backbuffer control
  - Pipelines
  - Memory management
  - Binding model
  - Synchronization

OpenGL History

Version	Year	Description
1.0	1992	**First Publication**
1.1	1997	Vertex Arrays, Texture Objects, Polygon Offset
1.2	1998	3D Textures, Pixel formats, MipMaps
1.2.1	1998	ARB Extensions, ARB Multitexture
1.3	2001	Compressed Textures, Cube Maps, Multi-Texturing
1.4	2002	Depth Textures, automatic MipMaps, Fog coordinates
1.5	2003	**Buffer Objects**, Occlusion Queries
2.0	2004	**OpenGL Shading Language, Multiple Render Targets,** non-power-of-2 Textures, Point Sprites, Vertex Texture Fetch
2.1	2006	Pixel Buffer Objects, sRGB Textures, OpenGL Shading Language 1.2

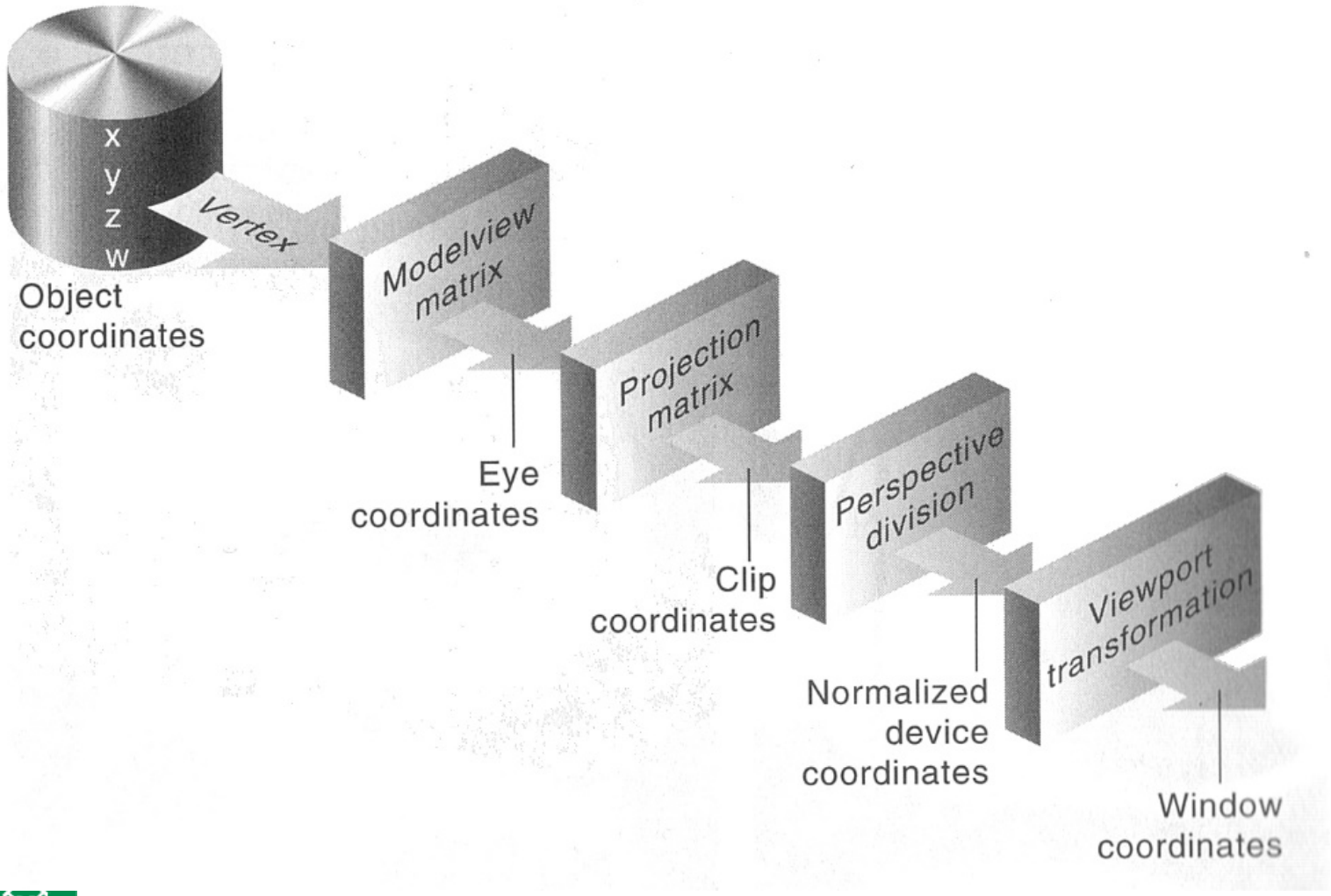
Version	Year	Description
3.0	2008	OpenGL Shading Language 1.3, **Removal of the Fixed-Function-Pipeline (begin/end, T&L,…)**, Similarities to DirectX
3.1	2009	Compatibility extension for old OpenGL , **Uniform Buffer Objects**, Primitive Restart, Instancing, CopyBuffer, Texture Buffer Objects
3.2	2009	**Geometry Shader**, Sync and Fence, OpenGL Shading Language 1.5
3.3	2010	Several ARB Extensions, OpenGL Shading Language 3.3
4.0	2010	**OpenCL, Tessellation**, OpenGL Shading Language 4.0
4.1	2010	Binary Shader Programs, double precision for Vertex Shader, OpenGL Shading Language 4.1
4.3	2012	**Compute Shader**
4.4	2013	Several OpenGL-Objects Shading Language 4.4

Version Comparison

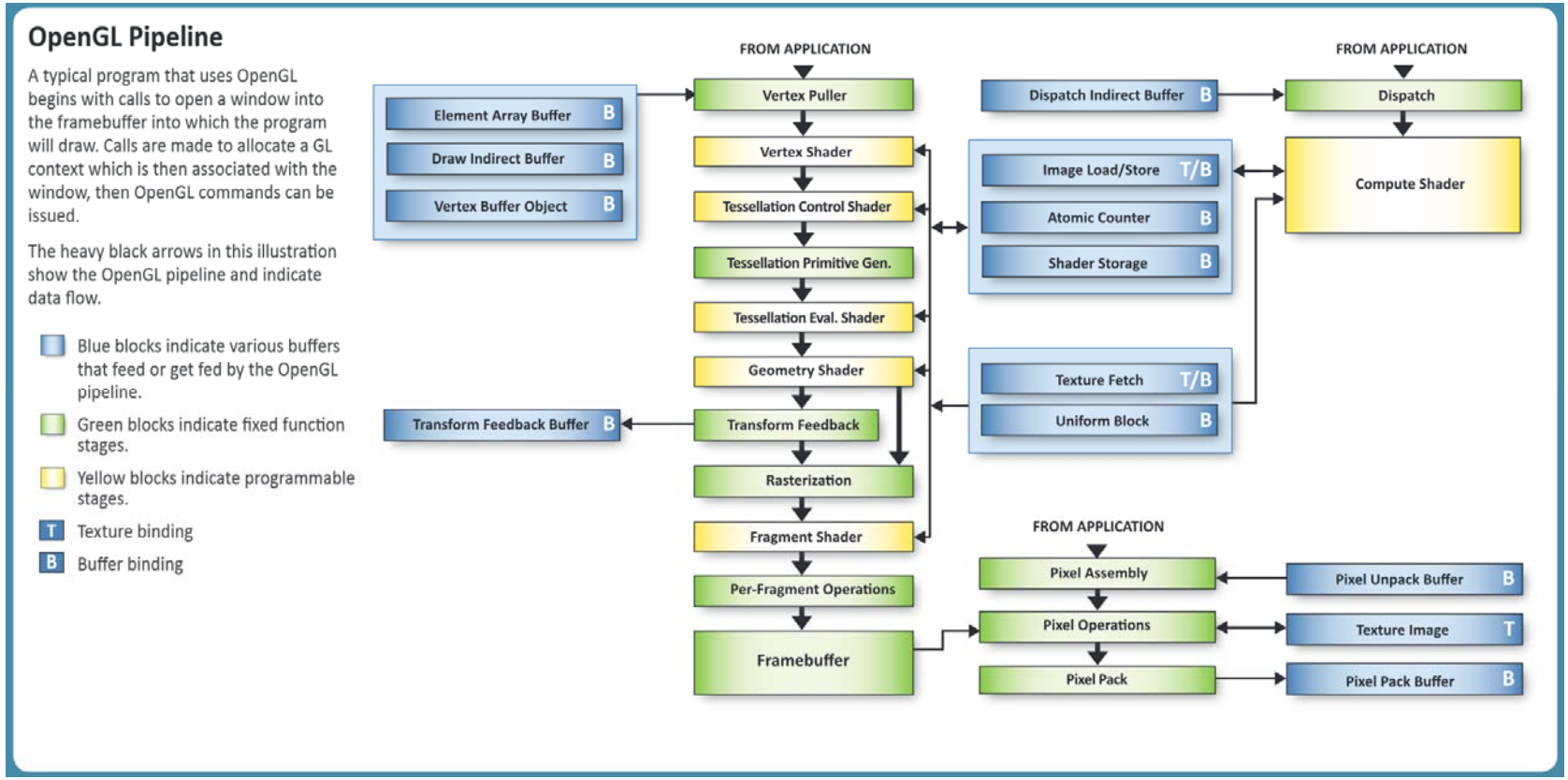
- OpenGL 1.0 (1992)
- Easy to understand for a beginner
- Only a few lines of code for the first triangle on screen
- Theory and practice can be shown side-by-side
- Fixed-function-pipeline
  - Transformation, projection, texturing, illumination
- Difficult to tweak for general purpose tasks
- Slow for large geometry due to memory transfer CPU→ GPU

- OpenGL 4.4 (2013)
- Difficult to understand for a beginner
- Circular dependencies, no good starting point → framework required
- Hundrets of lines of code for the first triangle on screen
- Additional libraries required for matrix/vector calculations
- Several programmable shader stages → flexible
  - Only few fixed pipeline stages
- Works well for general purpose tasks
- Fast for large geometry

OpenGL 1.0 Pipeline



OpenGL 4.4 Overview



Additional Libraries

- Showing a simple OpenGL program is difficult with modern OpenGL

- Therefore, we do not look into OpenGL today, but we look into some additional libraries that we need
  - glm (Vector/Matrix mathematics)
  - GLFW (Open a window, mouse/keyboard input)
  - GLAD (activate extensions for modern OpenGL)

glm: OpenGL Mathematics

- Vectors and matrices are often required in Computer Graphics
- This is why we use **glm** (OpenGL Mathematics), which is a separate Library (actually only header files) for Matrix/Vector calculations
- By operator overloading we can use vectors and matrices similar to scalar types like int and float
- Identical syntax to GLSL shaders

glm Examples

- with namespace

```
#include <glm/glm.hpp>

glm::vec2 a( 1.0, 1.0);
glm::vec2 b( 0.0, 1.0);
glm::vec2 c;

c = a + b;
```

- without namespace

```
#include <glm/glm.hpp>

using namespace glm;

vec2 a( 1.0, 1.0);
vec2 b( 0.0, 1.0);
vec2 c;

c = a + b;
```

GLFW

- We need additional functionality to open a window and for user input from keyboard and mouse
  - OpenGL does not support this, so we use an additional library for these tasks
- Several options
  - FreeGLUT (used in lecture „Computergrafik 1“ and Red Book)
    - OK, but quite old (original GLUT is from 1997)
  - Qt (lots of interfaces, but very large…)
  - SDL (Simple direct media layer, especially for game developers)
  - We decided to use GLFW (http://www.glfw.org/, OpenGL FrameWork
    - similar to FreeGLUT, but more recent
    - supports other versions of OpenGL, like OpenGL ES

GLFW Create a Window

```
#include <GLFW/glfw3.h>

int main(void)
{
    GLFWwindow* window;

    /* Initialize the library */
    if (!glfwInit())
        return -1;

    /* Create a windowed mode window and its OpenGL context */
    window = glfwCreateWindow(640, 480, "Hello World", NULL, NULL);
    if (!window)
    {
        glfwTerminate();
        return -1;
    }
```

GLFW Version 3 Header

A pointer for the window we will create

Init GLFW
Must be done before GLAD init

Create a window of size 640x480 with title „Hello World“

Finish if window creation failed

GLFW Create a Window

```
/* Make the window's context current */
glfwMakeContextCurrent(window);

/* Loop until the user closes the window */
while (!glfwWindowShouldClose(window))
{
    /* Render here */
    glClear(GL_COLOR_BUFFER_BIT);

    /* Swap front and back buffers */
    glfwSwapBuffers(window);

    /* Poll for and process events */
    glfwPollEvents();
}

glfwTerminate();
return 0;
}
```

The context (OpenGL state)

The main loop (FreeGLUT uses a display function for this)

Any draw commands go here

Bring back buffer to front (no flickering)

Optional polling of mouse/keyboard events

Call this before the application ends

Keyboard Callback Function

- A callback function can be used for keyboard input, the function can be registered using

```
glfwSetKeyCallback(window, key_callback)
```

```
void key_callback(window, key, scancode, action, mods)
{
  if (key == GLFW_KEY_E && action == GLFW_PRESS) {
    /* code for key e pressed */
  }
}

// Action = {GLFW_PRESS, GLFW_REPEAT, GLFW_RELEASE}
```

Mouse Callback Function

- A callback function can be used for mouse input, the function can be registered using

```
glfwSetCursorPosCallback(window, cursor_pos_callback);
```

```
static void cursor_pos_callback(window, xpos, ypos)
{
  // this function is called when the mouse is moved
  // the mouse pointer position is (xpos, ypos)
}
```

GLAD

- OpenGL is a specification, the commands do not work automatically
- We need a library to „load“ the OpenGL commands: **GLAD**
  - Without using such a library, many OpenGL commands are unknown or invalid function pointers
  - Additionally, we can activate the supported extensions of our graphics card easily
  - GLAD is an alternative to GLEW (we use in Comutergrafik 1)

- **GLAD = Multi-Language GL/GLES/EGL/GLX/WGL Loader-Generator based on the official specs.**

GLAD Init

```
#include <glad/glad.h>
#include <GLFW/glfw3.h>

int main(void)
{
    GLFWwindow* window;
    glfwInit();
    window = glfwCreateWindow(640, 480, "Hello World", NULL, NULL);
    …
    if (!gladLoadGLLoader((GLADloaderproc) glfwGetProcAddress)) {
      std::cout << „failed to init OpenGL context“;
      return -1;
    }
}
```

Include GLAD header before GLFW header

Init GLAD after GLFW

That´s all for today

- Next week we repeat the basic mathematics we need for OpenGL