Real - Time Rendering

Graphics pipeline

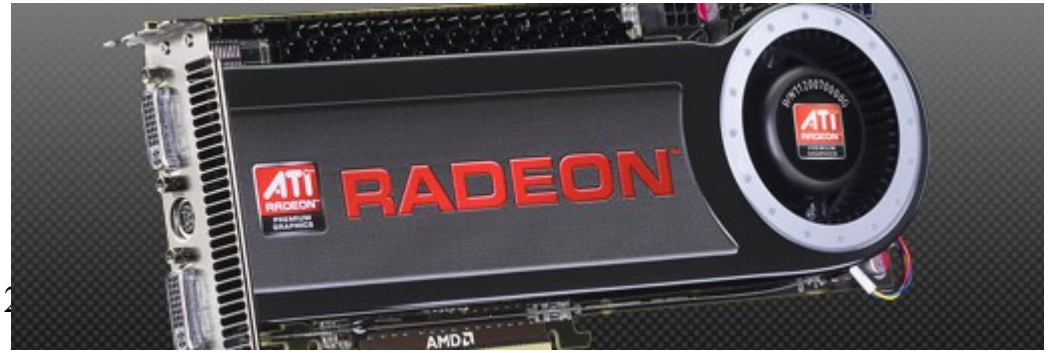
Michal Červeňanský
Juraj Starinský

Overview

- History of Graphics HW
- Rendering pipeline
- Shaders
- Debugging

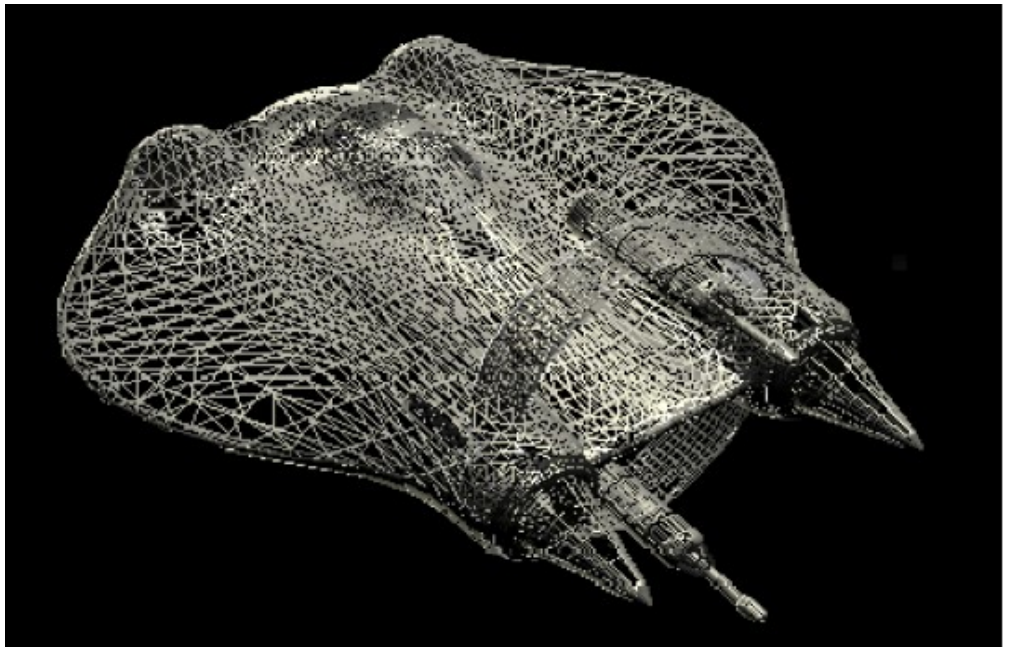
History of Graphics HW

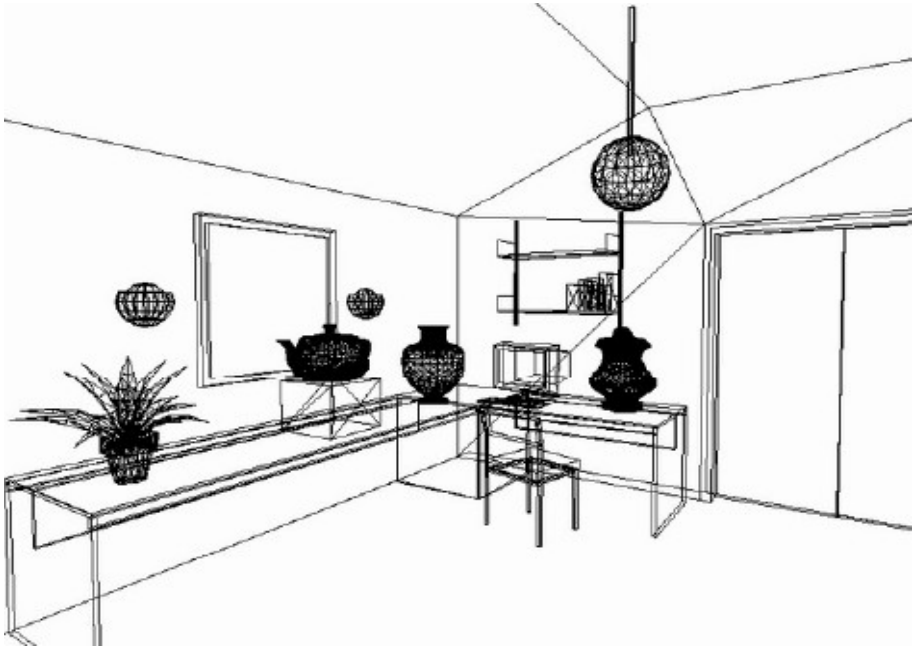
- First generation
- Second generation
- Third generation
- Fourth generation
- Fifth generation



First generation

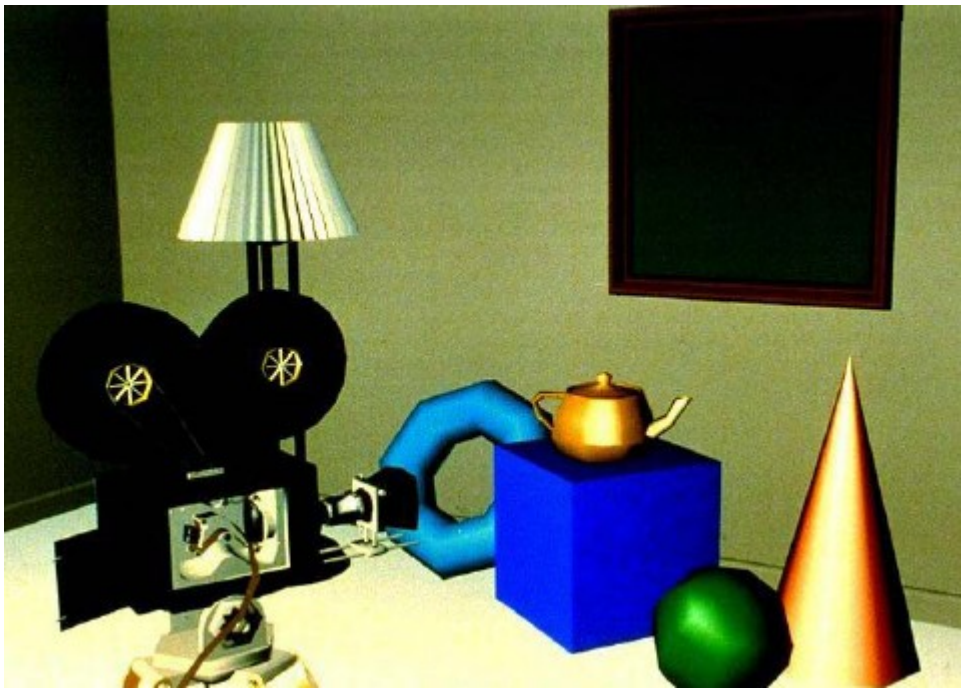
- Vertex - transform, clip, project
- Pixel - color interpolation of lines
- Frame buffer - overwrite
- When - prior to 1987





Second generation

- Vertex - lighting calculations
- Pixel - depth interpolation, triangles
- Frame buffer - depth buffer, color blending
- Dates - 1987-1992



Third generation

- Vertex - texture coordinate transformation
- Pixel - texture coordinate interpolation

  - texture evaluation and filtering
- Dates - 1992-2000

Fourth generation

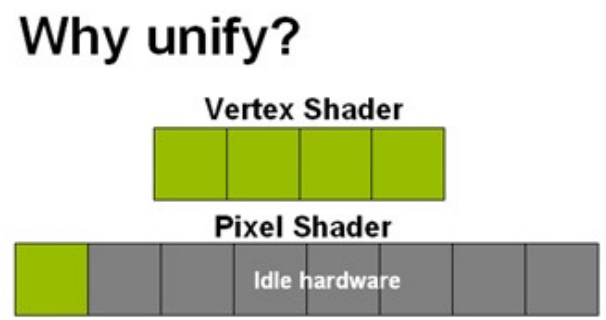
- Programmable shading
  - Vertex, Pixel, Geometry
- Multi GPU – SLI, Crossfire
- Full floating point
- GPGPU

Fifth generation

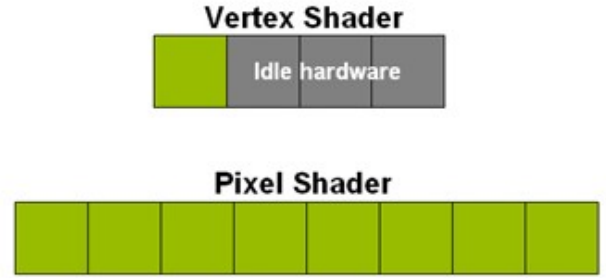
- Multi cores
- Merging CPU and GPU
  - IBM Cell (8+1cores)
  - AMD Fusion (CPU+GPU)
  - Intel Larabee – canceled
- Programming
  - Tesselation shader
  - DX11 (Compute shader)
  - CUDA, AMD Brook
  - OpenCL

Unified architecture

- (Non)Unified architecture
  - Sw – input parameters, functionality
  - Hw – stream processors



Heavy Geometry
Workload Perf = 4



Heavy Pixel
Workload Perf = 8

Heavy Geometry
Workload Perf =12

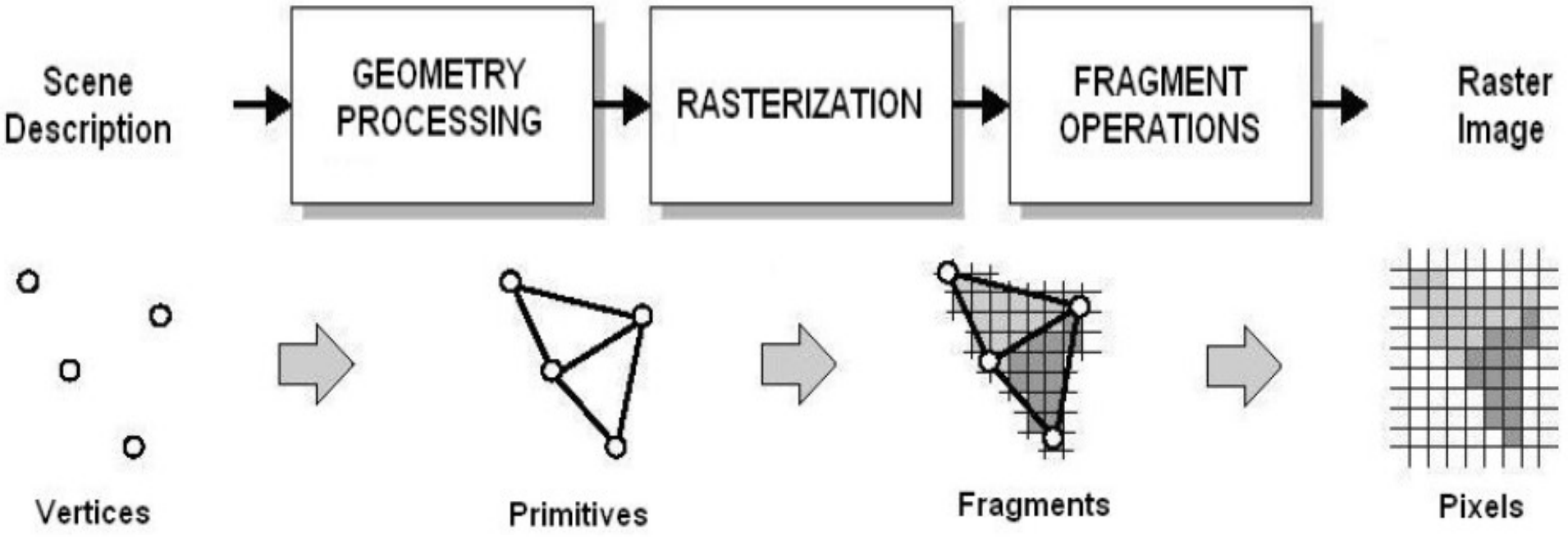
Heavy Pixel
Workload Perf = 12

Overview

- History of Graphics HW
- Rendering pipeline
  - Application stage
  - Geometry stage
  - Rasterization stage
  - Fragment tests
  - HW pipeline
  - Unified architecture
- Shaders
- Debugging

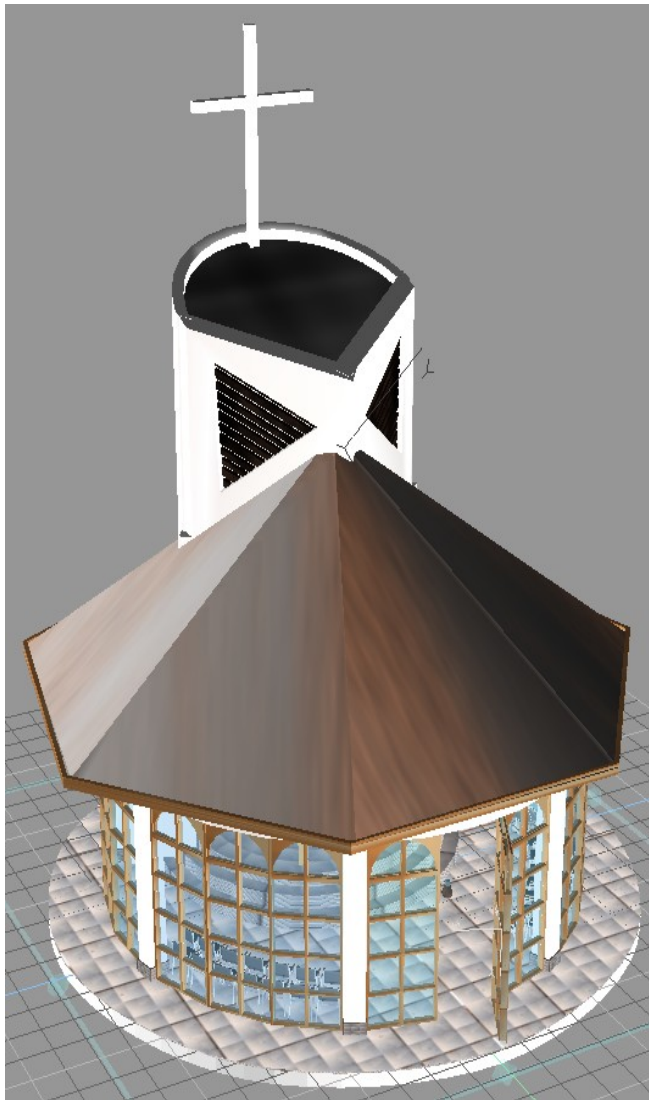
Rendering pipeline

- Application – user settings, scene description
- Geometry – transformations, clipping, projection
- Rasterization – texturing, computations
- Fragments – tests, blending



Application stage

- Scene description
- User full control
- Scene complexity – LOD, clipping
- Scene dynamics (physics)
- Handling – mouse, keyboard

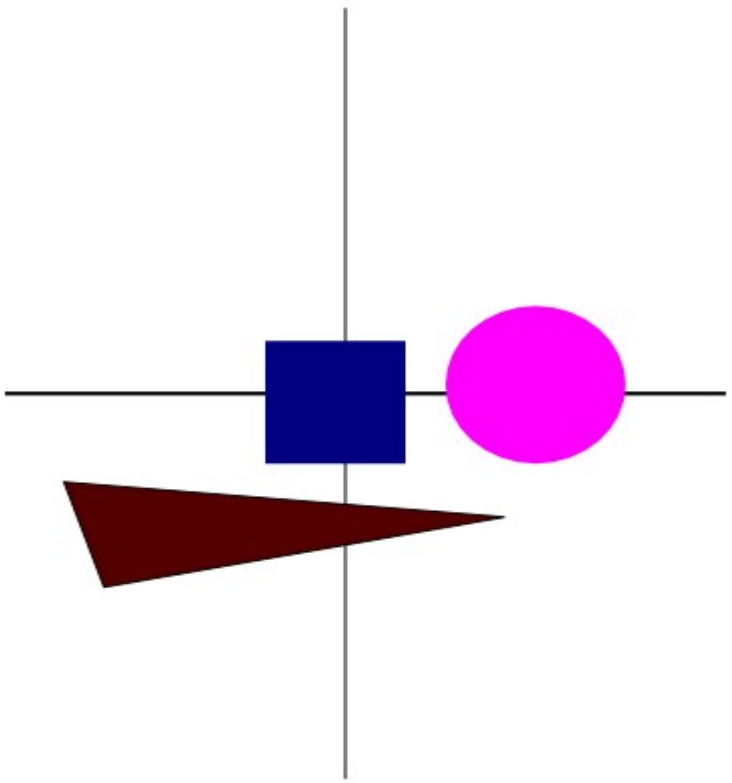


Geometry stage

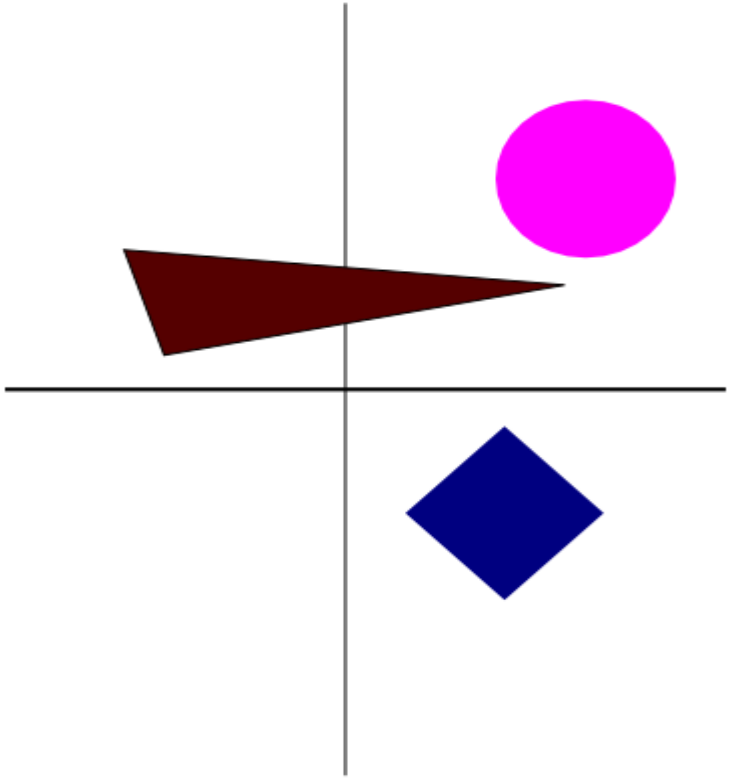
- Per-primitive operations
  - Model & view transform
  - Lightning & shading
  - Projection
  - Clipping
  - Screen mapping

Geometry stage (2)

- Model & view transform



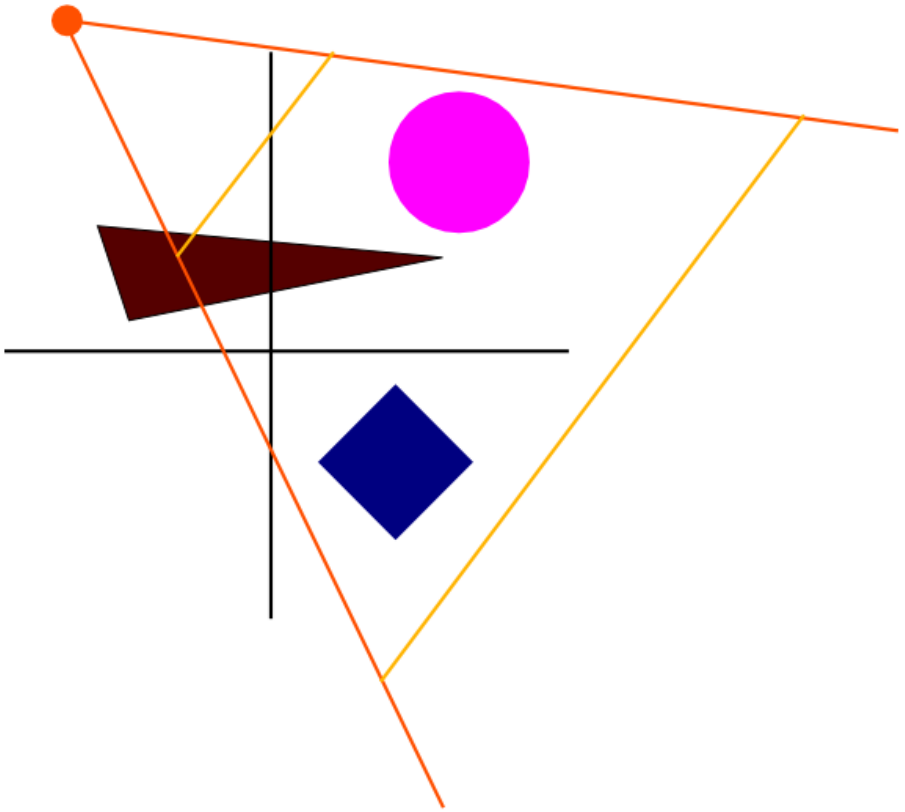
Model space



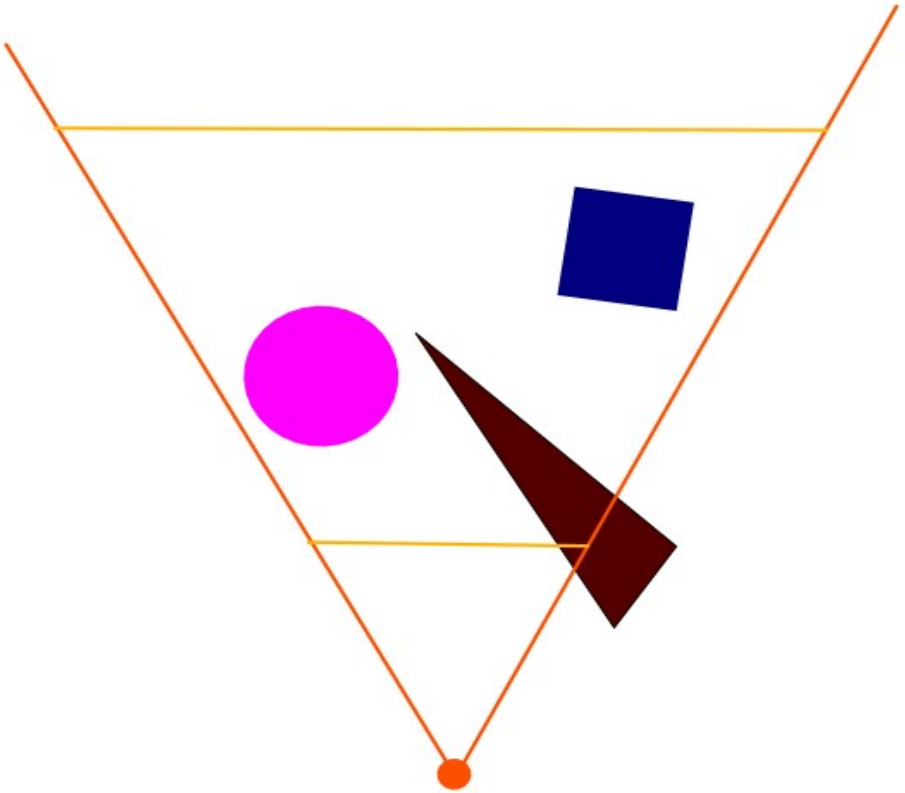
World space

Geometry stage (3)

- Model & view transform



World space



Camera space

Geometry stage (4)

- Lighting, projection, clipping

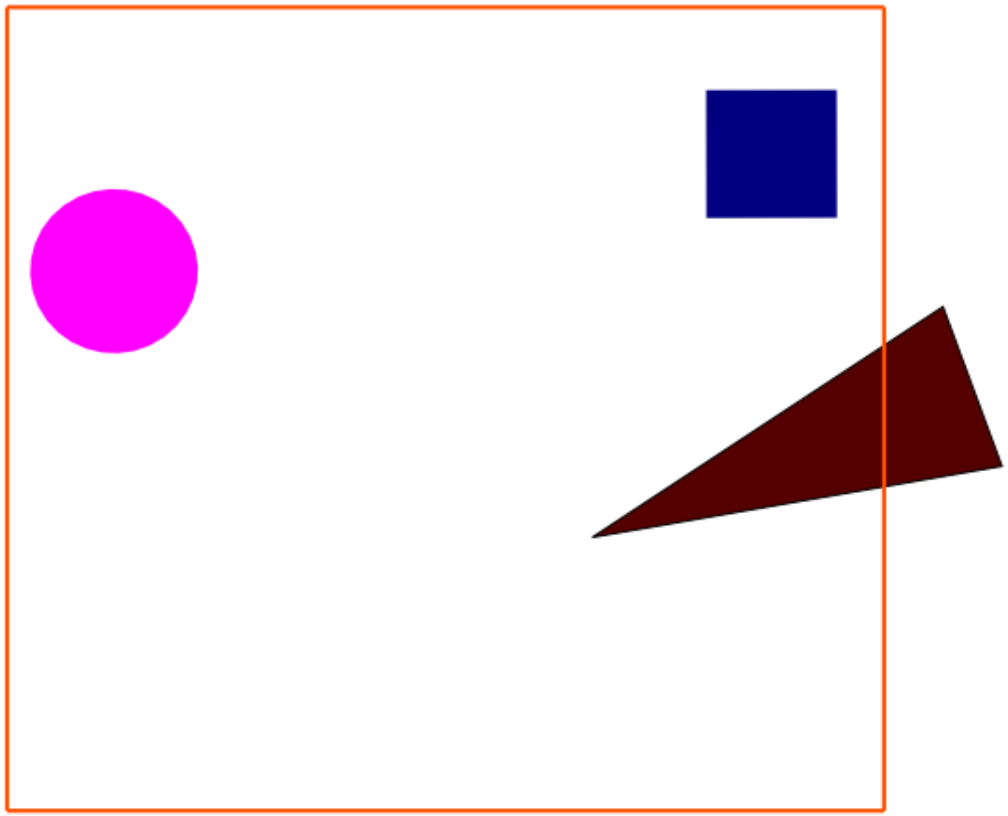


Image space

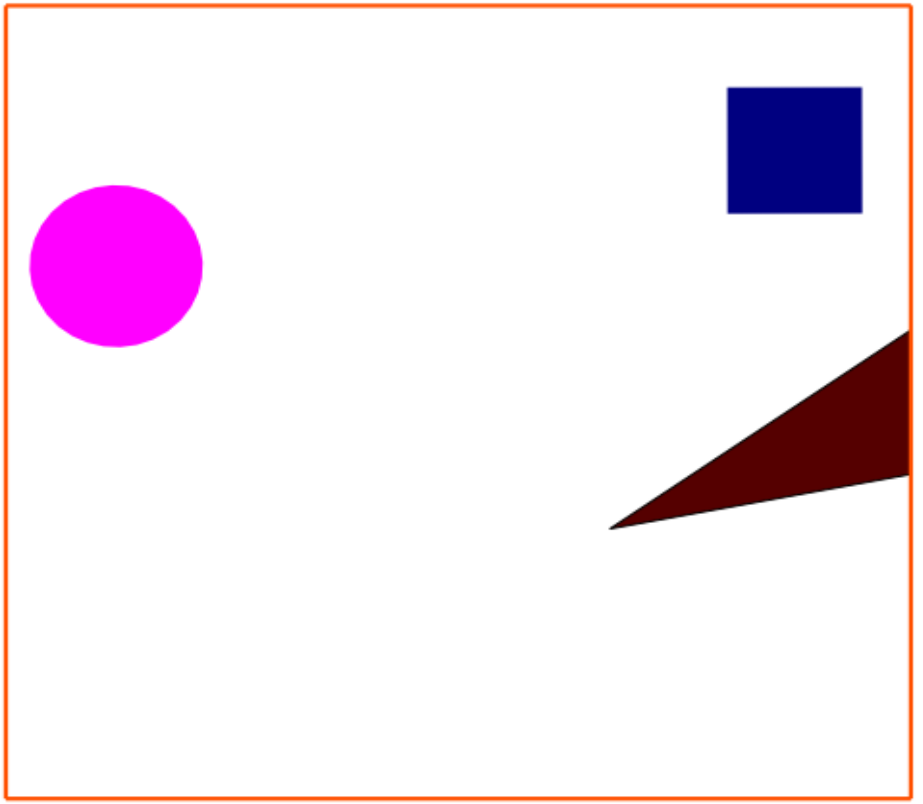


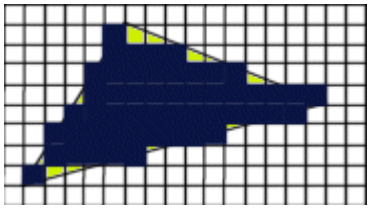
Image space

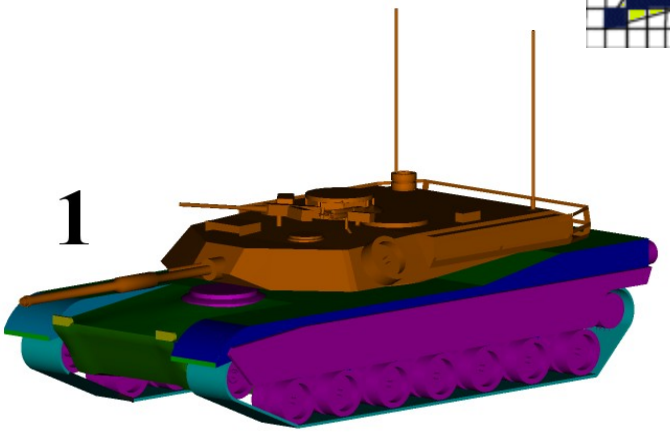
Geometry stage (5)

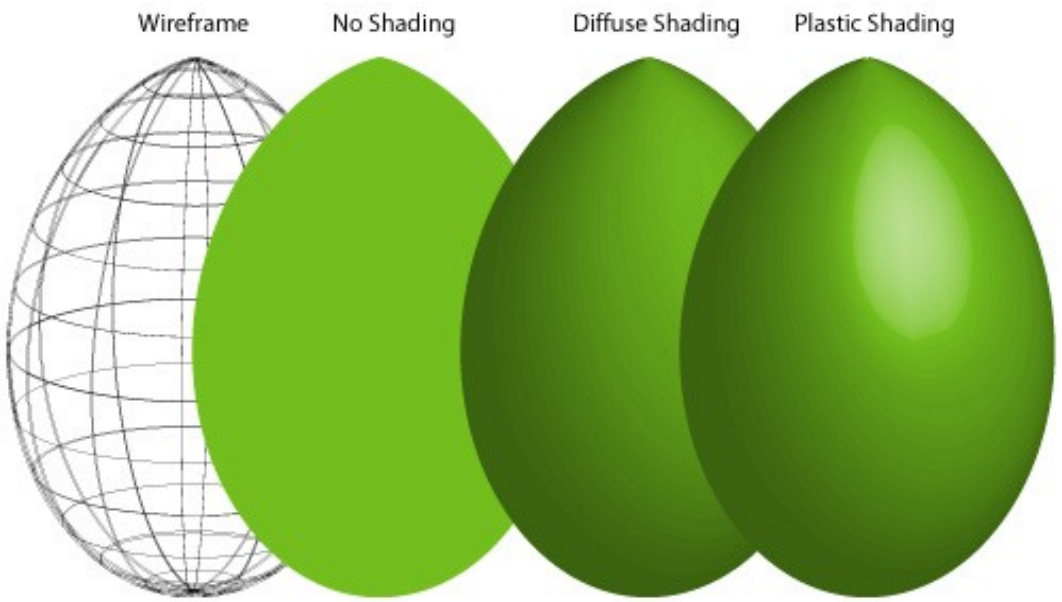
- Mapping

Rasterization stage

- Per-fragment operations
  - Attribute interpolation
  - Color mapping
  - (Multi)texturing
  - Computations

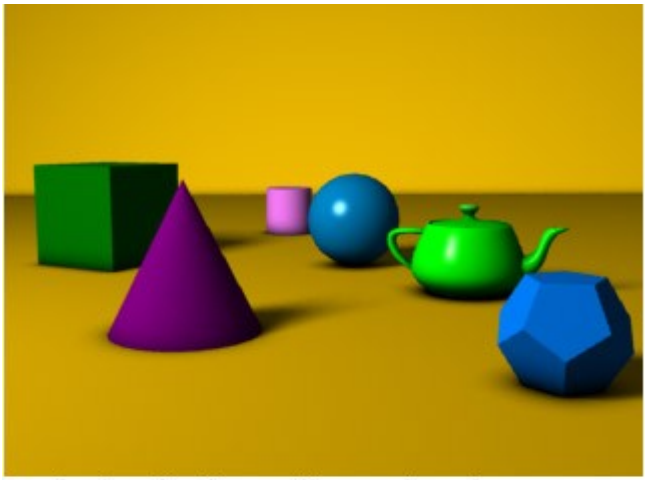






Fragment tests

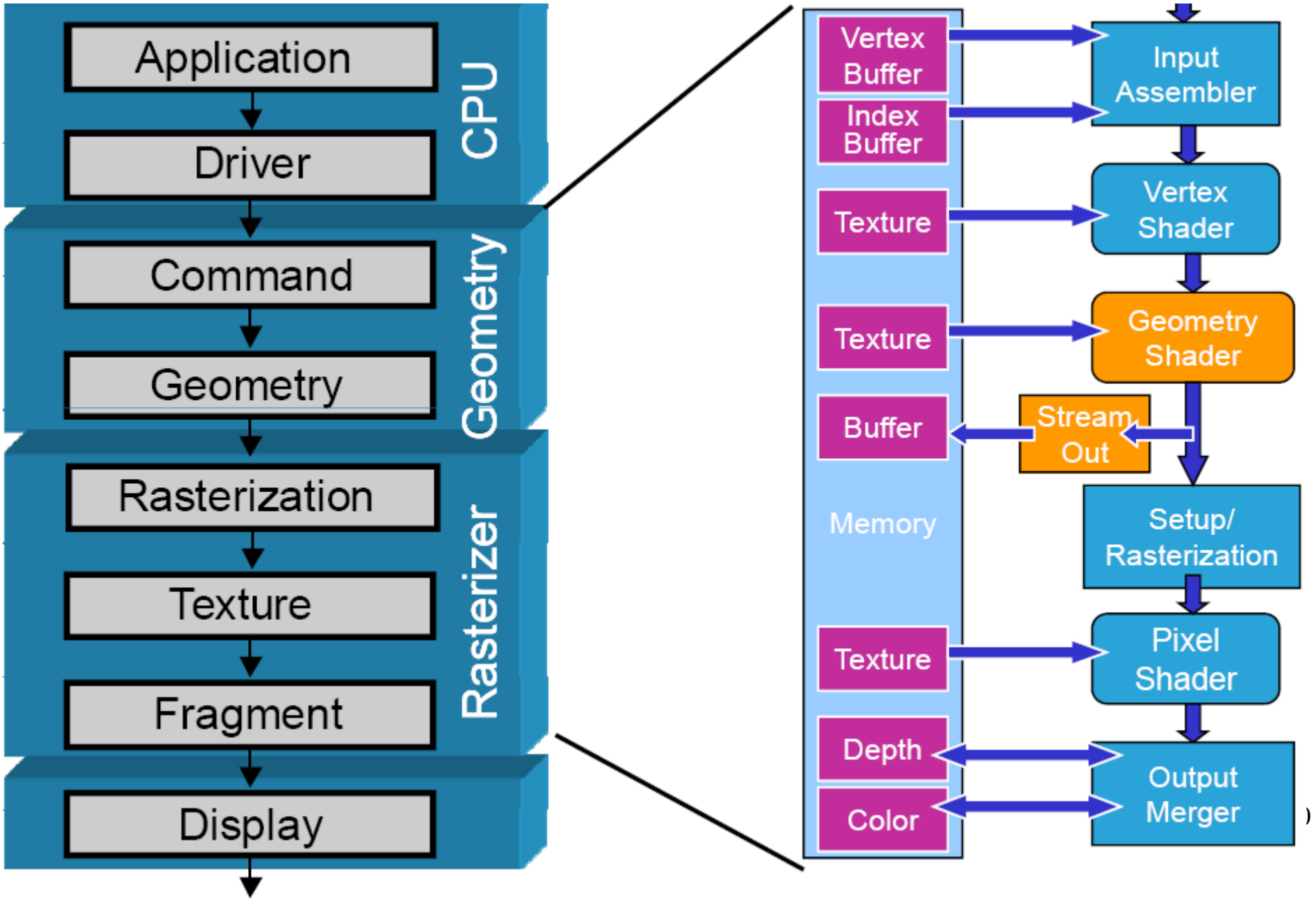
- Ownership
- Scissor test
- Alpha test
- Stencil test
- Depth test
- Alpha blending
- Logical operations



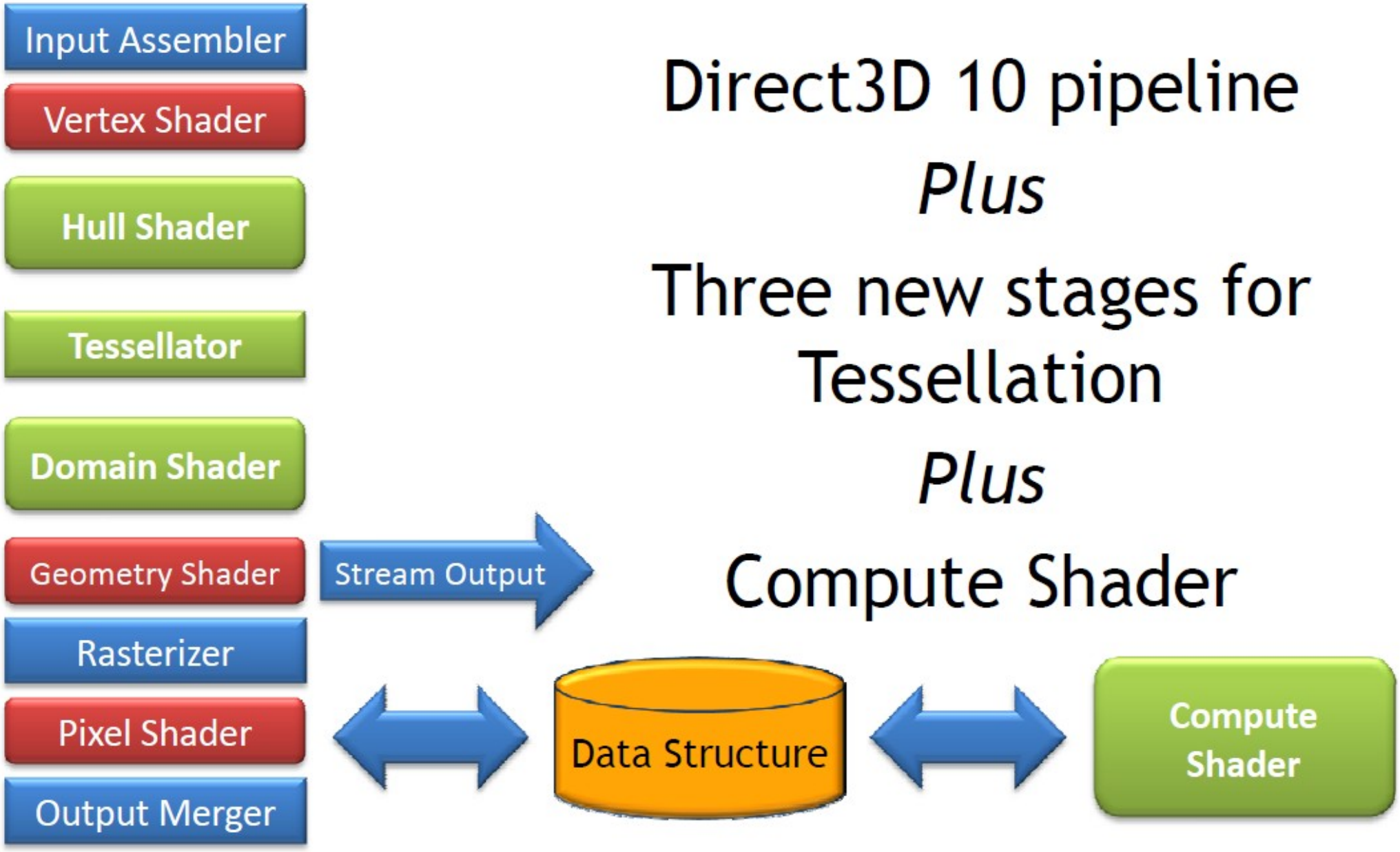
A simple three dimensional scene

Z-buffer representation

DirectX10 pipeline



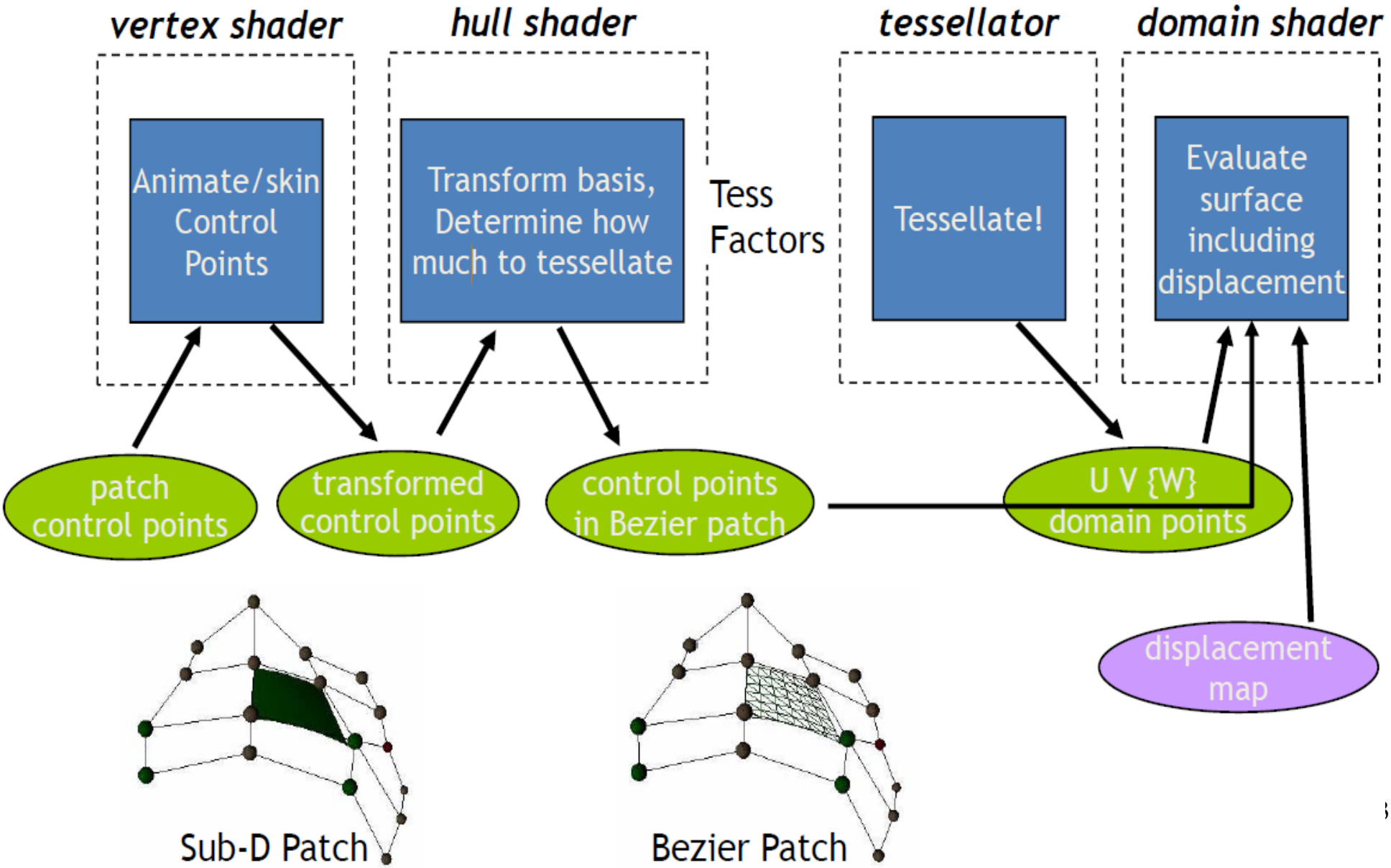
DirectX11 pipeline



Geometry shader

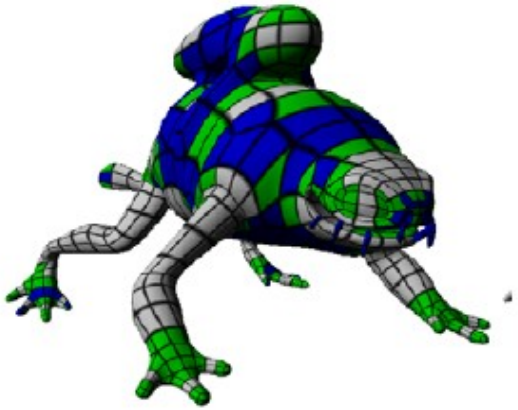
- Calculations on a primitive
- Access to neighbour primitives
- Generation new primitives (triangles)
- Limited output (1024 32-bit values)
- Applications
  - Extrusion operations (fur rendering)
  - Shadow volume generation
  - Render to cubemap

Tesselation



Tesselation 2

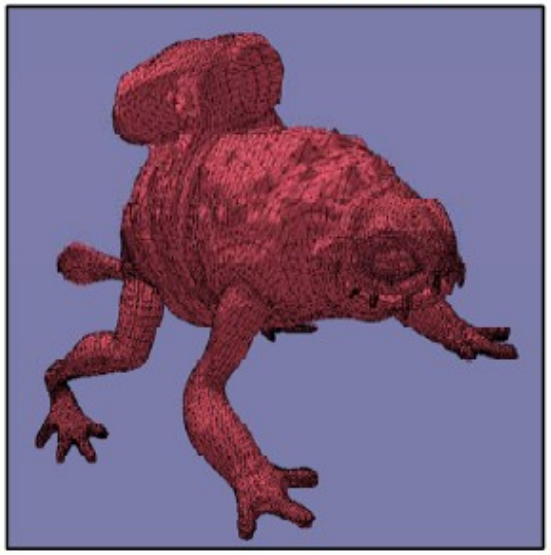
Sub-D Modeling



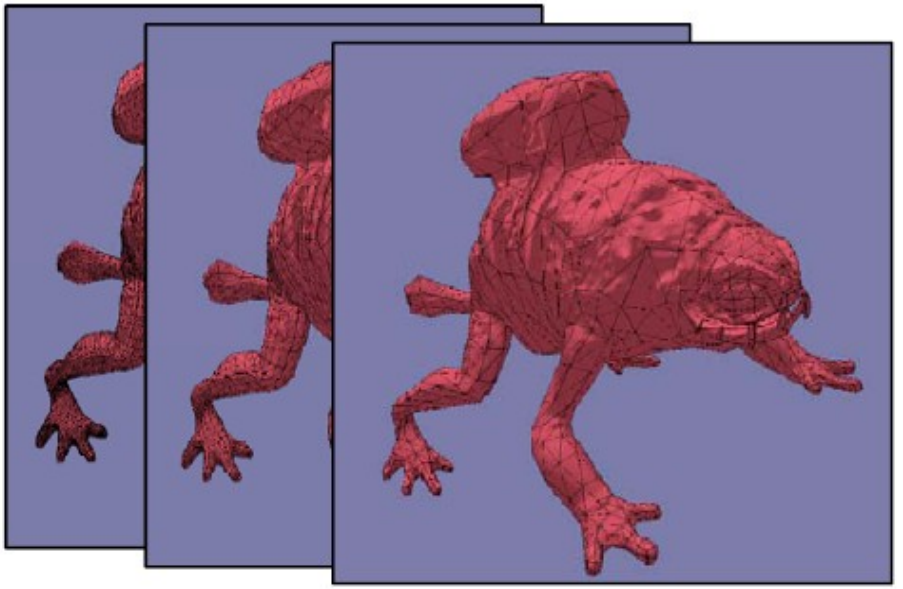
Animation

Displacement Map

Polygon Mesh



Generate LODs



Overview

- History of Graphics HW
- Rendering pipeline
- Shaders
  - Introduction
  - Languages (Cg, glsl)
- Debugging

Introduction

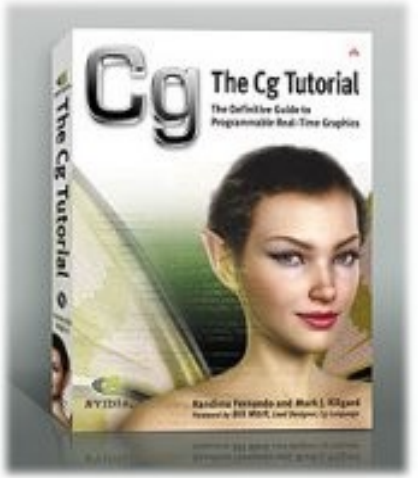
- Types – Vertex, Fragment, Geometry
- Parallel processing
- Input
  - textures, matrices, variables
  - uniform, varying
- Output
  - vertex (attributes), color (depth), primitives

Languages

- Assembler
- Cg – C for graphics
  - Nvidia
  - HLSL (DirectX)
- glsl – OpenGL shading language
  - OpenGL standard
  - ATI, Nvidia

Cg

- Nvidia extensions
- Various profiles
- Compiler, runtime libraries



- Examples – Cg Toolkit

Cg – vertex

```
void  main( float4 position : POSITION,
            uniform float4x4 matrixModelProj,
            out float4 oPos     : POSITION)
{
  oPos = mul(matrixModelProj, position);
}
```

```
struct input_data
{
  float4 position   : POSITION;
};

struct output_data
{
  float4 position   : POSITION;
};

output_data main(input_data IN)
{
  output_data OUT;
  OUT.position  = IN.position;
  return OUT;
}
```

Cg – fragment

```
void main( float2 tc : TEXCOORD0,
           uniform sampler2D textureIn,
           out float4 oColor : COLOR )
{
  oColor = tex2D(textureIn, tc);
}
```

```
struct input_data{
  float2 tc : TEXCOORD0;
};
struct output_data{
  float4 color  : COLOR;
};

output_data main(input_data IN,
          uniform sampler2D textureIn )
{
  output_data OUT;
  OUT.color = tex2D(textureIn, IN.tc);
  return OUT;
}
```

glsl

- glsl v 1.30
- Based on C
- Quick reference
  - http://www.opengl.org/sdk/libs/OpenSceneGraph/glsl_quickref.pdf
- Example
  - http://www.lighthouse3d.com/opengl/glsl/index.php?intro
  - Basic example (demonstration)

Error handling

- OpenGL errors
- Cg, glsl errors
- More errors

```
bool CheckCgError(CGcontext context_in, unsigned int progName_in, const char* msg_in
{
  CGerror error;
  const char *string = cgGetLastErrorString(&error);

  if (error != CG_NO_ERROR){
      LOG(string<<" Program( "<<progName_in<<") "<<msg_in);

      if (error == CG_COMPILER_ERROR)
          LOG(cgGetLastListing(context_in));

      return false;
  }

  return true;
}
```

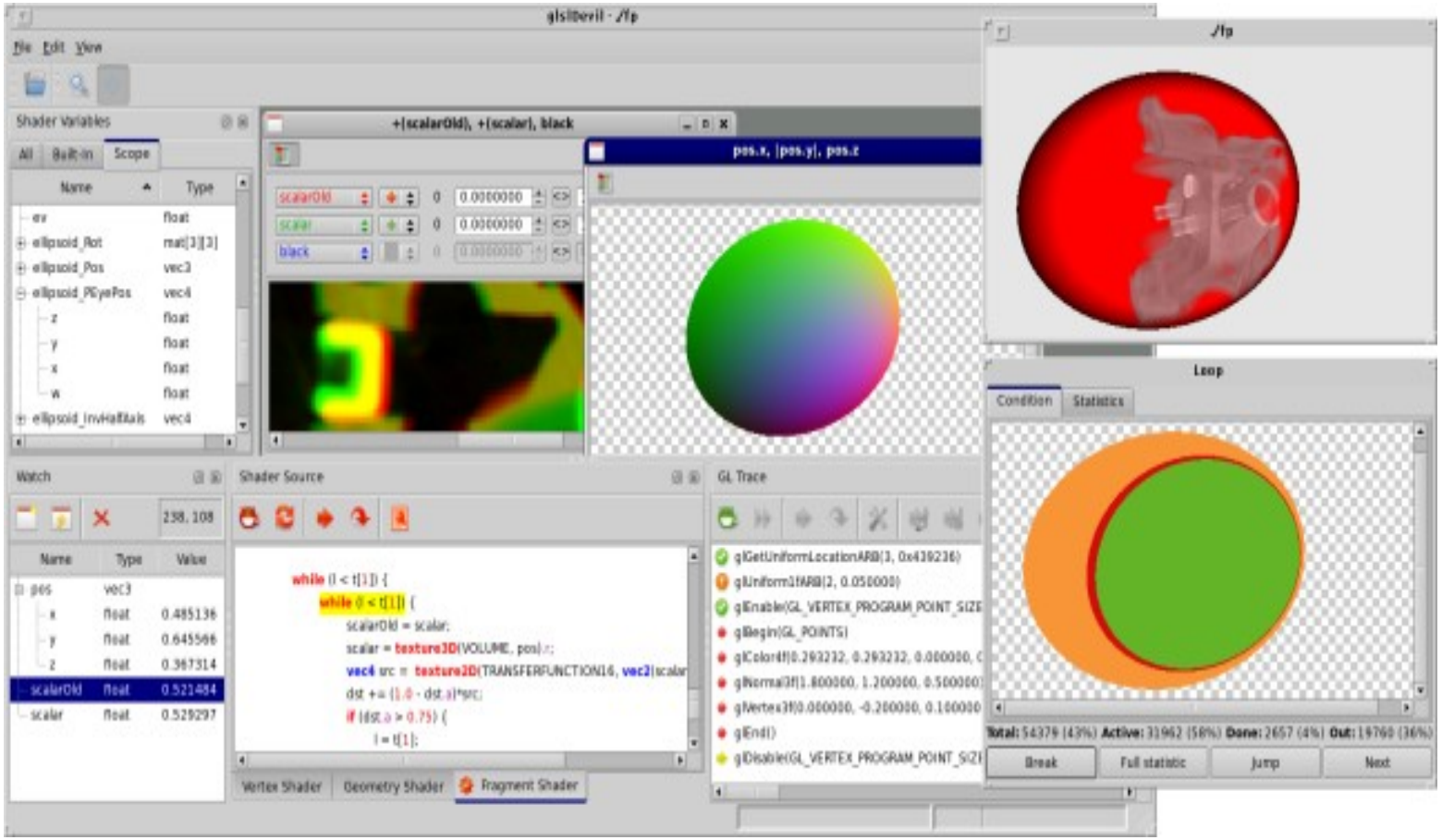
```
bool CheckGLError(const char *msg_in)
{
  GLint error = glGetError();

  if(error == GL_NO_ERROR )
    return true;

  do{
    fprintf(stderr, "OpenGL ERROR: %s - %s (%d)\n", msg_in, (char*)gluErrorString(error), error);
    error = glGetError();
  }while(error!= GL_NO_ERROR);

  return false;
}
```

Debugging

- Error handling
- Software tools
  - gDEBugger
  - Fx Composer (+ Shader Debugger)
  - glslDevil
  - …
- Own techniques – hard but working ;)

Debuggers

- Demonstration (glslDevil)



References

- Real-time rendering, Akenine-Möller, Haines
- http://www.cg.tuwien.ac.at/courses/Realtime/
- www.nvidia.com
- http://www.caligari.com/ - TrueSpace
- http://graphicssoft.about.com
- http://en.wikipedia.org/
- www.gamedev.net
- http://informatik.uibk.ac.at/teaching/ss2008/seminar/gpu.pdf
- http://www.nvidia.com/content/nvision2008/tech_presentations/Game_Developer_Track/NVISION08-Direct3D_11_Overview.pdf

Questions ?