# Shaders

Slide credit to Prof. Zwicker

# Today

- Shader programming

- Quick start
- Background
- Context

# A simple shader

## Goal: Produce RGBA

```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    fragColor = vec4(1.0,0.0,0.0,1.0);
}
```

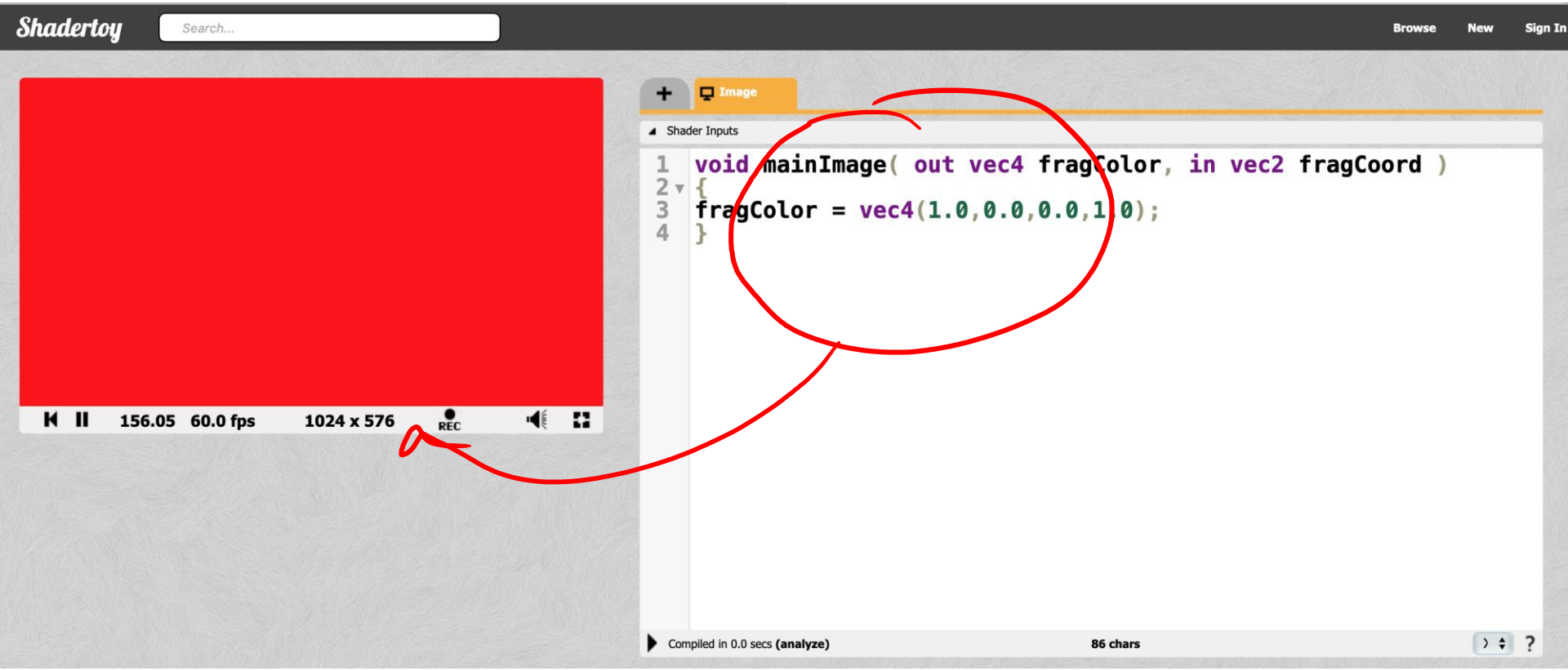
R G B A

Red color

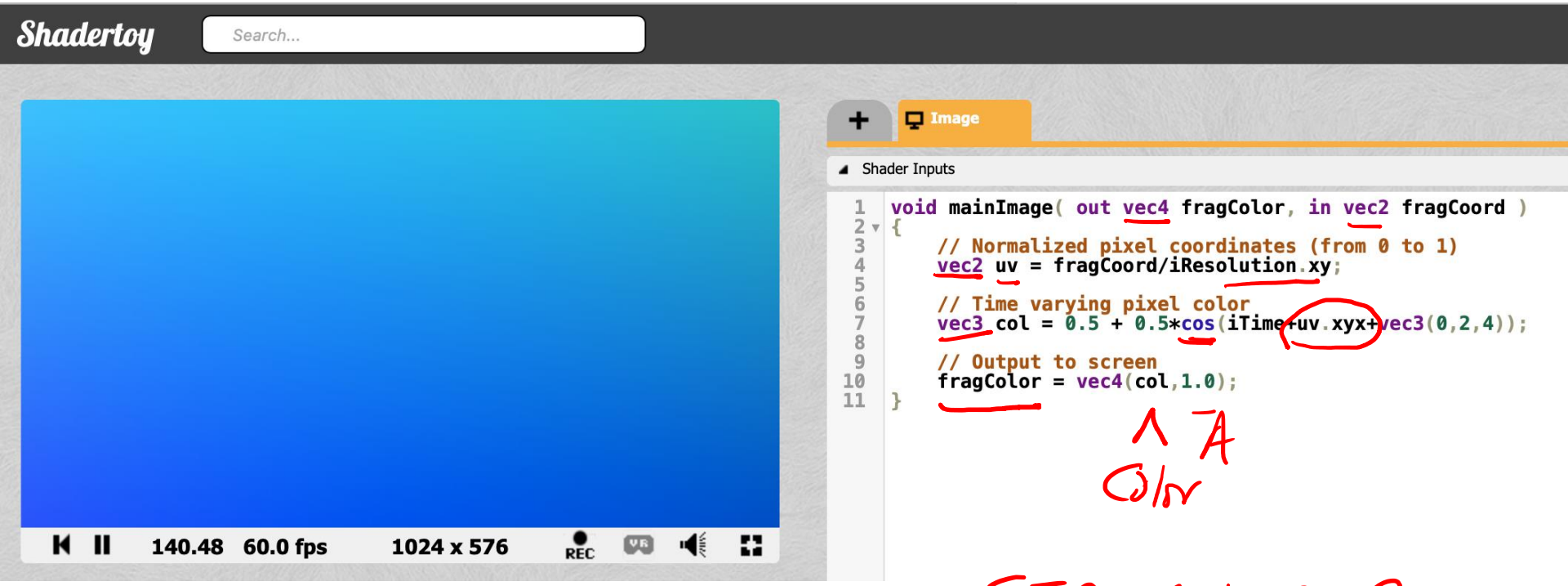
Shadertoy version

https://gamedevelopment.tutsplus.com/tutorials/a-beginners-guide-to-coding-graphics-shaders--cms-23313

# Shadertoy - online editor



# Shadertoy - dynamic




STD SHADER NEW

UV is vec2 x,y fields

UV.XYX Swizzling

# Standard shading model

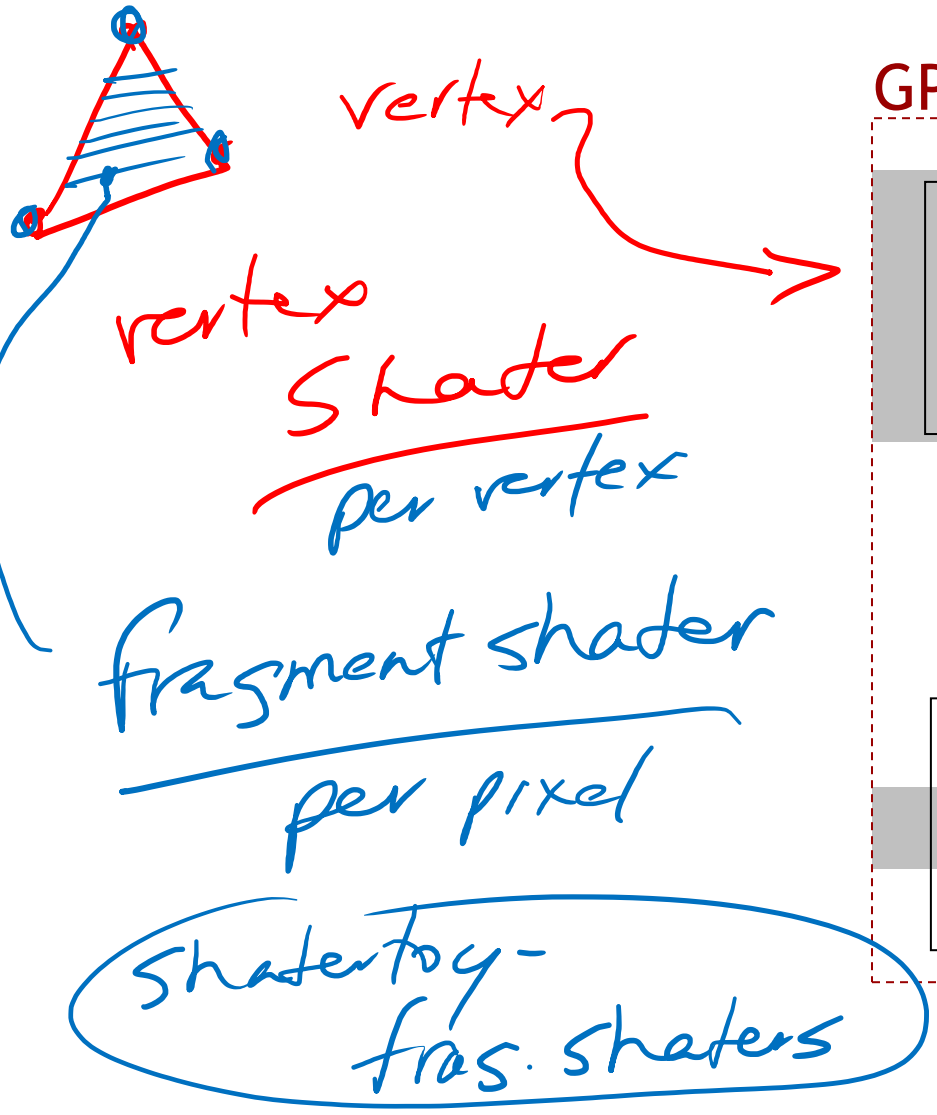
- Blinn model with several light sources $i$

$$c = \sum_i c_{l_i} \left( k_d \left( \mathbf{L}_i \cdot \mathbf{n} \right) + k_s \left( \mathbf{h}_i \cdot \mathbf{n} \right)^s \right) + k_a c_a$$

diffuse specular ambient

**How is this implemented on the graphics processor (GPU)? Shader programming!**

# Programmable pipeline




GPU

Scene data

Vertex processing, modeling and viewing transformation

Projection

Rasterization, fragment processing, visibility

Image

# GPU architecture (2006)

DATA PARALLELISM

## Pipeline

GPU

Scene data

Vertex processing, modeling and viewing transformation

Projection

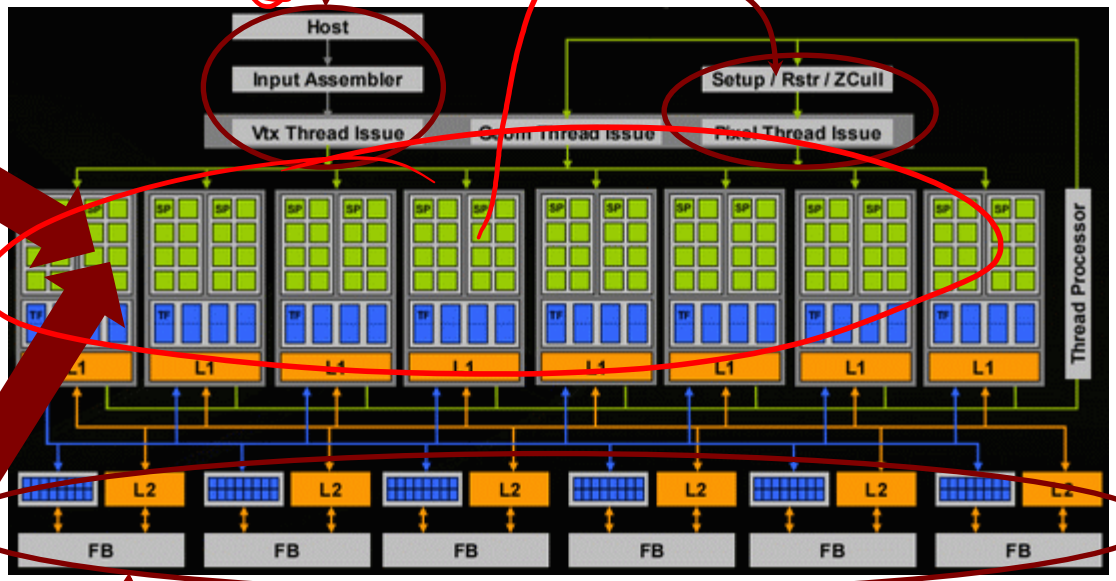
Rasterization, fragment processing, visibility

Image

## GPU Architecture

Data

SAME SHADER




NVidia NV80 (GeForce 8800 Series)

128 functional units, "stream processors"

http://arstechnica.com/news.ars/post/20061108-8182.html

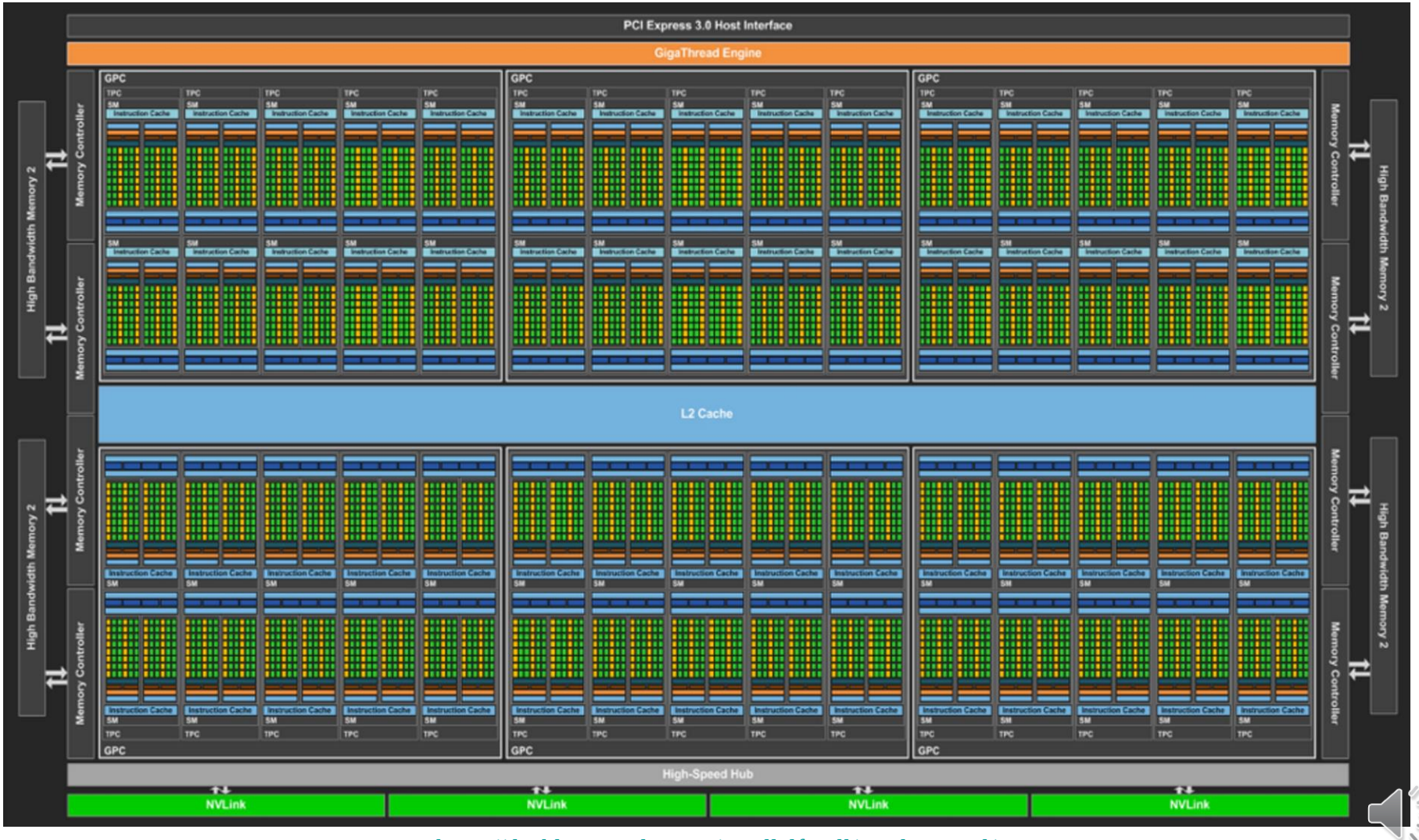
# GPU architecture (2014)

- Similar, but more processors (2048 ■ )


http://hexus.net/tech/reviews/graphics/74849-nvidia-geforce-gtx-980-28nm-maxwell/

# GPU architecture (2016)

- Similar, but even more processors (3840 ■)




https://devblogs.nvidia.com/parallelforall/inside-pascal/

# Still fixed functionality (2014)

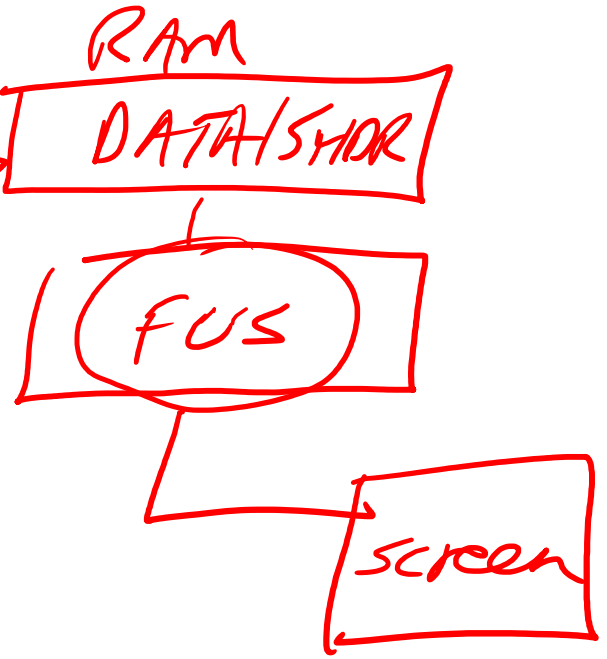
- “Hardcoded in hardware”
- Projective division
- Rasterization
  - I.e., determine which pixels lie inside triangle
  - Vertex attribute interpolation (color, texture coords.)
- Access to framebuffer
  - Z-buffering
  - Texture filtering
  - Framebuffer blending

# CPU vs. GPU

- CPU

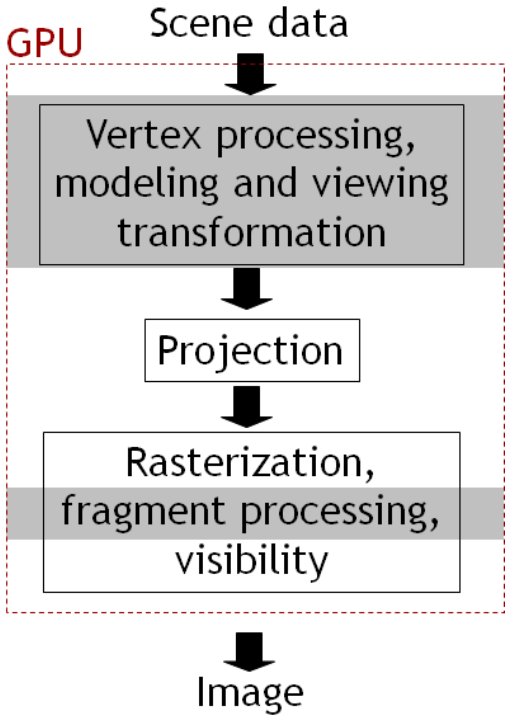
JAVA PROGRAM
-> HAS SHADERS AS TEXT
-> COMPILES
-> LOADS TO GPU
-> ASSOCIATES DATA
-> LOADS TO GPU
-> INITIATES GO!

- GPU

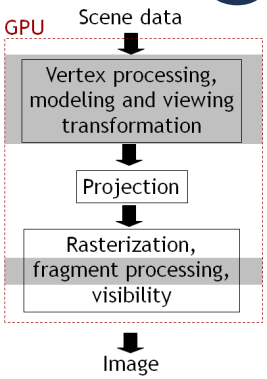


# Fragment programs

- Executed once for every fragment
  - Or: “Every fragment is processed by same fragment program that is currently active”
- Implements functionality for
  - Output color to framebuffer
  - Texturing
  - Per-pixel shading
  - Bump mapping
  - Shadows
  - Etc.



# Fragment programs




**Fragment data storage classifier `in`**

Interpolated vertex attributes, additional fragment attributes

From rasterizer

**Uniform parameters storage classifier `uniform`**

OpenGL state, application specified parameters

**Fragment program**

To fixed framebuffer access functionality (z-buffering, etc.)

**Output storage classifier `out`**

Fragment color, depth

# “Hello world” fragment program

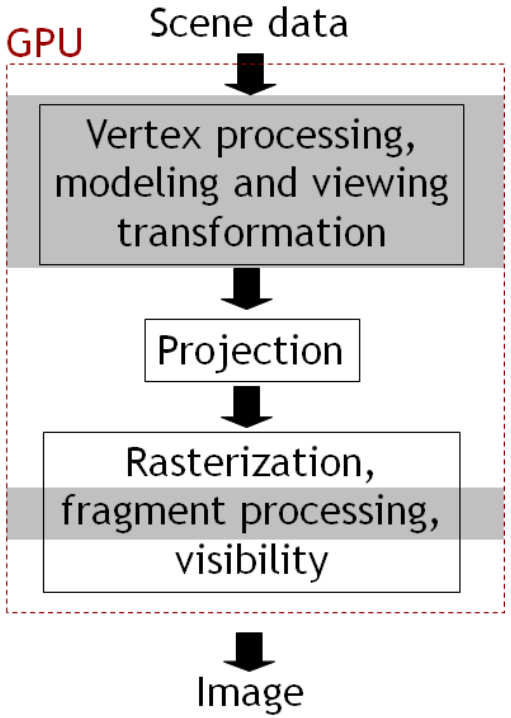
- `main()` function is executed for every fragment - OpenGL GLSL
- Draws everything in bluish color

NOT SHADERTOY

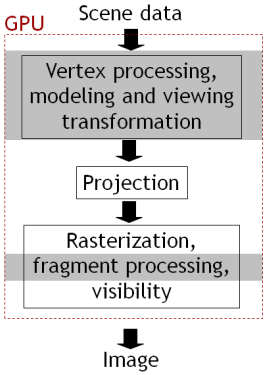
```
out vec4 fragColor;

void main()
{
   fragColor = vec4(0.4,0.4,0.8,1.0);
}
```

# Vertex programs

- Executed once for every vertex
  - Or: “every vertex is processed by same vertex program that is currently active”
- Implements functionality for
  - Modelview, projection transformation (required!)
  - Per-vertex shading
- Vertex shader often used for animation
  - Characters
  - Particle systems



# Vertex programs




**Vertices with attributes**
**storage classifier `in`**
Coordinates in object space, additional vertex attributes

From application

**Uniform parameters**
**storage classifier `uniform`**
OpenGL state, application specified parameters

Vertex program

To rasterizer

frag. shader

**Output**
**storage classifier `out`**
Transformed vertices, processed vertex attributes

# “Hello world” vertex program

- `main()` function is executed for every vertex
- Three storage classifiers: **`in`**, **`out`**, **`uniform`**

vertex x,y,z,w

```
in vec4 position;          // position, vertex attribute
uniform mat4 projection;   // projection matrix, set by host (Java)
uniform mat4 modelview;    // modelview matrix, set by  host (Java)

void main()
{
  gl_Position =            // required, predefined output variable
    projection *           // apply projection matrix
    modelview *            // apply modelview matrix
    position;              // vertex position
}
```

# Creating shaders in OpenGL

- You can switch between different shaders during runtime of your application
  - Setup several shaders as shown before
  - Call `glUseProgram(s)` whenever you want to render using a certain shader `s`
  - Shader is active until you call `glUseProgram` with a different shader
- In `jrtr`, this functionality is encapsulated in the `Shader` class

# “Hello world” fragment program

- `main()` function is executed for every fragment
- Draws everything in bluish color

```
out vec4 fragColor;

void main()
{
   fragColor = vec4(0.4,0.4,0.8,1.0);
}
```

# GLSL built in functions and data types

- See OpenGL/GLSL quick reference card
  http://www.khronos.org/files/opengl-quick-reference-card.pdf
- Matrices, vectors, textures
- Matrix, vector operations
- Trigonometric functions
- Geometric functions on vectors
- Texture lookup

vector ops

# Summary

- Shader programs specify functionality of parts of the rendering pipeline
- Written in special shading language (GLSL in OpenGL)
- Sequence of OpenGL calls to compile/activate shaders
- Several types of shaders, discussed here:
  - Vertex shaders
  - Fragment shaders

# GLSL main features

- Similar to C, with specialties
- Most important: `in`, `out`, `uniform` storage classifiers
- Parameters of shader (`uniform` variables) passed from host application via specific API calls
- Built in vector data types, vector operations
- No pointers, classes, inheritance, etc.

# Tutorials and documentation

- OpenGL and GLSL specifications
  http://www.opengl.org/documentation/specs/
- OpenGL/GLSL quick reference card
  http://www.khronos.org/files/opengl-quick-reference-card.pdf
- Learn from example code and use the Ilias forum!