Lecture 2

Shaders, GLSL and GPGPU

Is it interesting to do GPU computing with graphics APIs today?

Information Coding / Computer Graphics, ISY, LiTH

Lecture overview

- Why care about shaders for computing?
- Shaders for graphics
- GLSL
- Computing with shaders

Lecture questions

1) What kind of shaders is most interesting for GPU computing? (What part of the pipeline?)

2) What geometry is usually used for shader-based GPU computing?

3) What is ping-ponging? Suggest an example where it is useful.

Why is classic GPGPU interesting?

- Highly suited to all problems dealing with images, computer vision, image coding etc
- Parallelization ”comes natural”, you can’t avoid it and good speedups are likely. Fewer pitfalls.
- Highly optimized (for graphics performance).
- Compatibility is vastly superior!
- Very much easier to install!

They say classic GPGPU is obsolete?

- Marketing hype!
- Mature technology - publishing opportunities limited.
- Remapping to images awkward for some problems.
- Limited access to local memory
- Double support hardware dependent extension

”Any given program, when running, is obsolete”

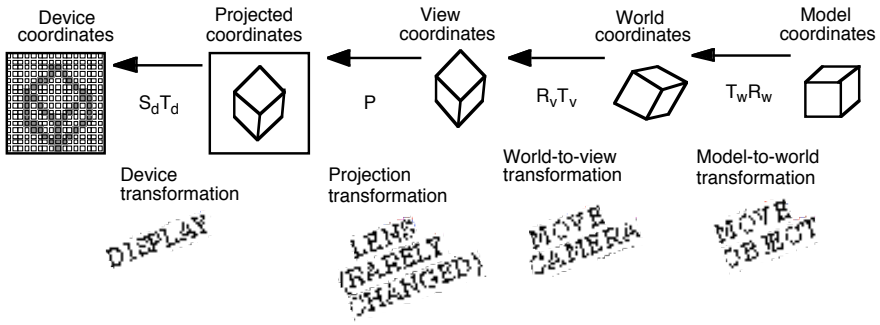
Information Coding / Computer Graphics, ISY, LiTH

Shaders and GLSL

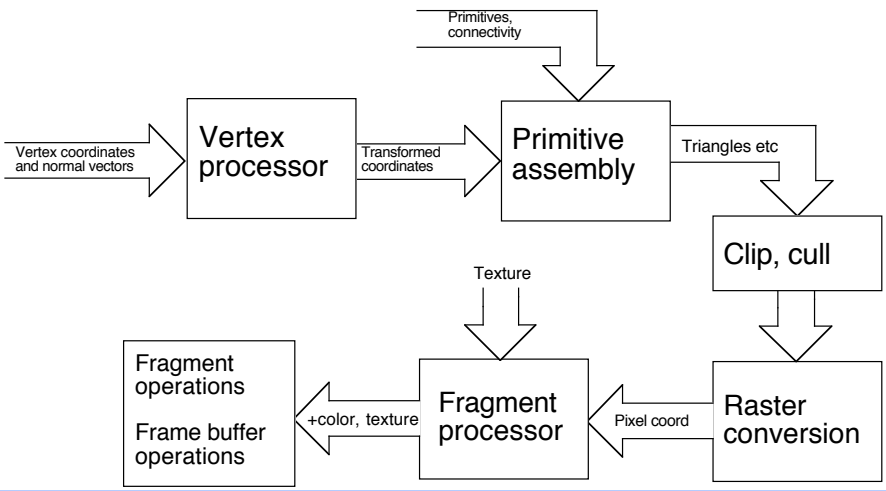
Let’s have a look at how it works!

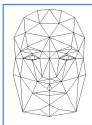
Transformations

performed by 4x4 matrix multiplications

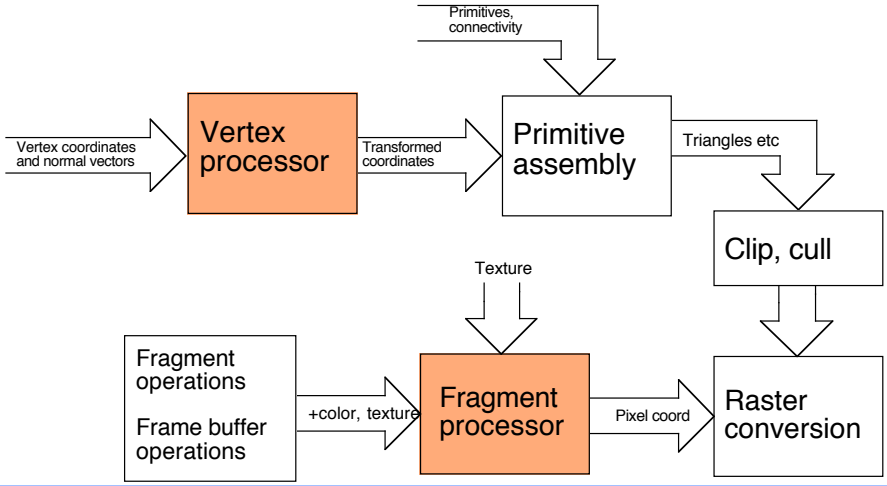


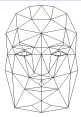
The OpenGL pipeline



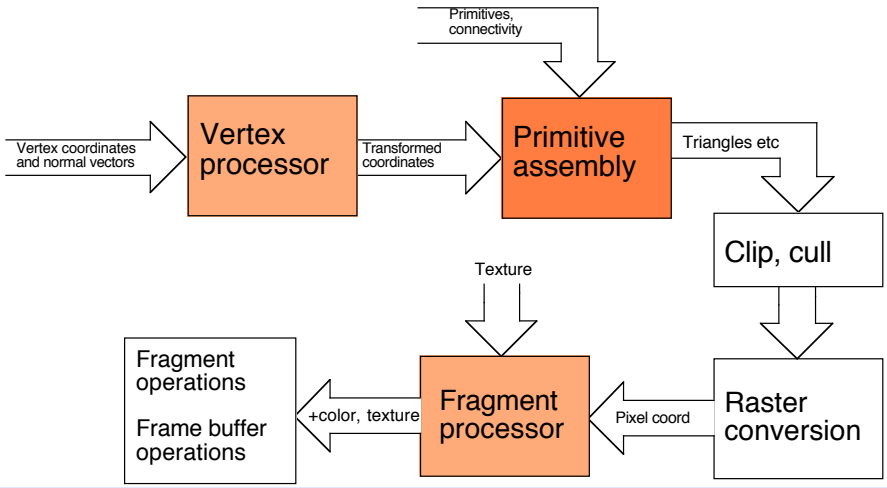


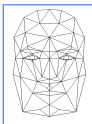
Out of these, two are programmable!





Actually, three (on advanced GPUs) - but we focus on the two first





Shader programs

Program snippets that are executed per vertex or per fragment, on the GPU!

Two programs cooperate, one vertex program and one fragment program.

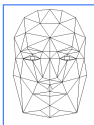
"Shader" implies that the goal is lighting, but that is only one of the goals!.

Vertex transform
Vertexcolor, vertex-level lighting

Can be done in a vertex shader

Texturing
Color and light per pixel

Can be done in a fragment shader



Vertex shader

Replaces the fixed functionality of the vertex processor.

It can:
- **transform vertices, normals and texture coordinates**
- **generate texture coordinates**
- **calculate lighting per vertex**
- **set values for interpolation for use in a fragment shader**

It knows nothing about:
- **Perspective, viewport**
- **Frustum**
- **Primitives (!)**
- **Culling**

Fragment shader
(a.k.a pixel shader)

Replaces the fixed functionality of the fragment processor.

It can:
- set the fragment color
- get color values from textures
- calculate fog and other color calculations
- use any kind of interpolated data from the vertices

It can not
- change the fragment coordinates
- write into textures
- affect stencil, scissor, alpha, depth...

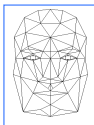
Shader languages

Four different:

Assembly language: Old solution, being phased out, no longer updated.

Cg: “C for graphics”, NVidia
HLSL: “High-level shading language”, Microsoft
GLSL: “OpenGL shading language”

Choice depends on flatform and needs (and taste).
GLSL superior for compatibility!



Typical shader examples in graphics

Vertex manipulation

Lighting calculations

Multitexturing

Bump mapping