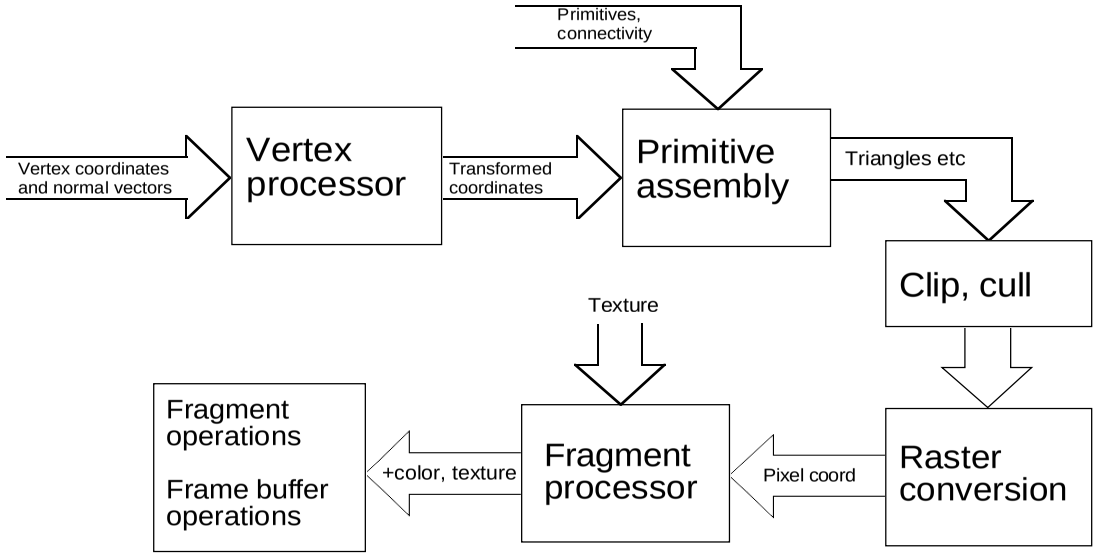
Lecture 7

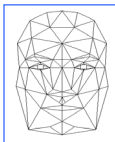
Programmable shaders

The OpenGL Shading Language

The 3D pipeline in the GPU

Low-level operations from vertices to pixel data

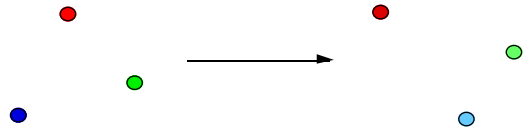


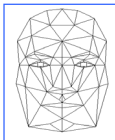


The vertex processor

The vertex processor handles the following tasks:

- **Vertex transformation (from model cordinates to screen coordinates)**
- **Transformation of normal vectors**
- **Generation of texture coordinates**
- **Transformation of texture coordinates**
- **Lighting calculations**
- **Material parameters**



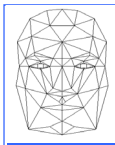


Primitive assembly

Assembly of primitives

“Primitive” not as in simple but as in geometrical primitives

Transformed coordinates are collected into structures for each triangle, quad etc.



Clipping and culling

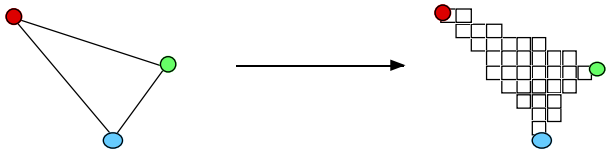
Primitives are clipped to screen borders. Backface culling is performed.

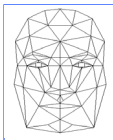
Note that texture coordinates also needs clipping (as well as any other data that is interpolated between vertices).

Raster conversion

Polygon rendering, convert polygons to pixel coordinates

Creates “fragments”. Note that they do not have any colors yet!

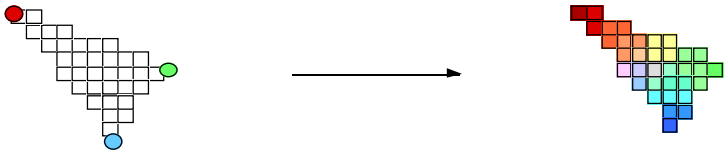




The fragment processor

From pixel coordinates and interpolated data for color, texture etc, calculate a color for the fragment.

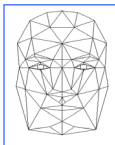
- **Shading**
- **Texturing**
- **Fog**
- **Color calculations**



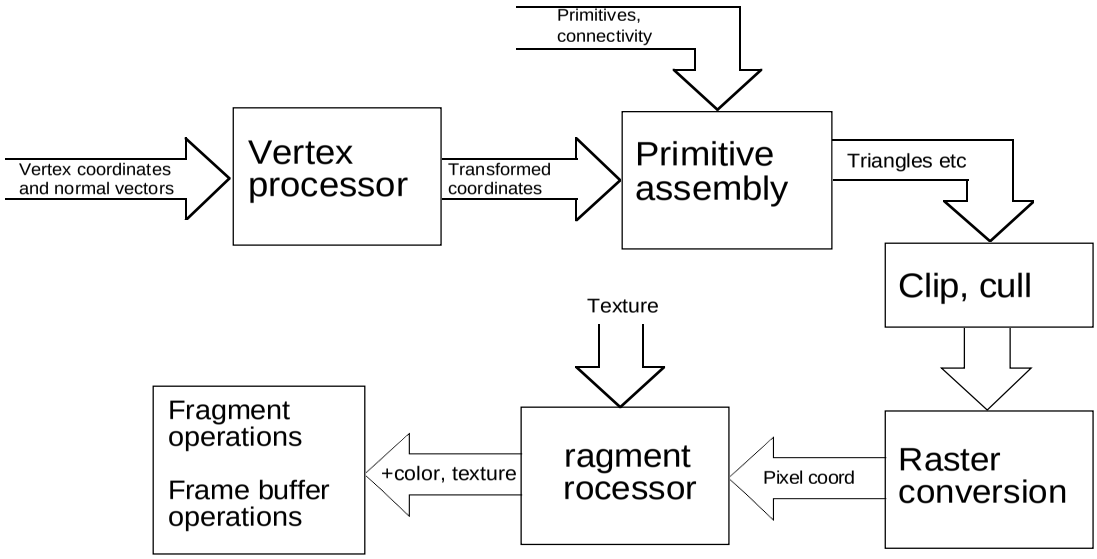
Fragmentoperationer

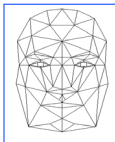
Final operations before the fragment is written to a frame buffer pixel

- **Stencil test**
- **Z-buffer test**
- **The blend function (glBlendFunc mm)**
- **The alpha function (glAlphaFunc)**



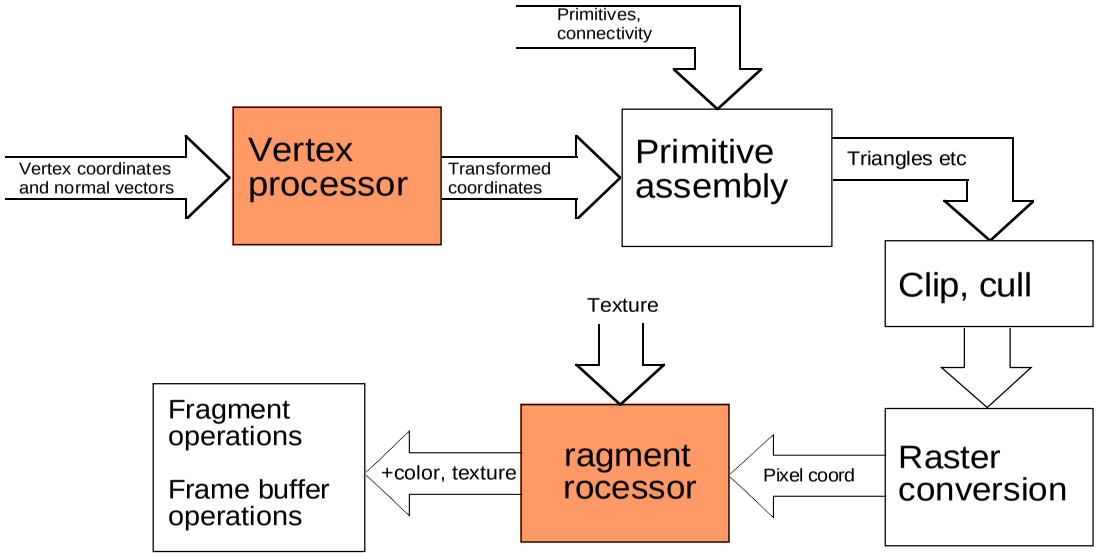
Out of these, two are programmable!

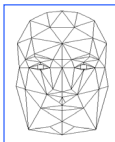




Information Coding / Computer Graphics, ISY, LiTH

Out of these, two are programmable!





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!.

ertex transform
ertexcolor, vertex-level lighting

Can be done in a vertex shader

exturing
olor and light per pixel

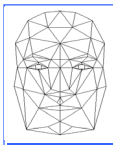
Can be done in a fragment shader

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)

eplaces the fixed functionality of the fragment rocessor.

t 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

t 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”

Choce depends on flatform and needs (and aste).