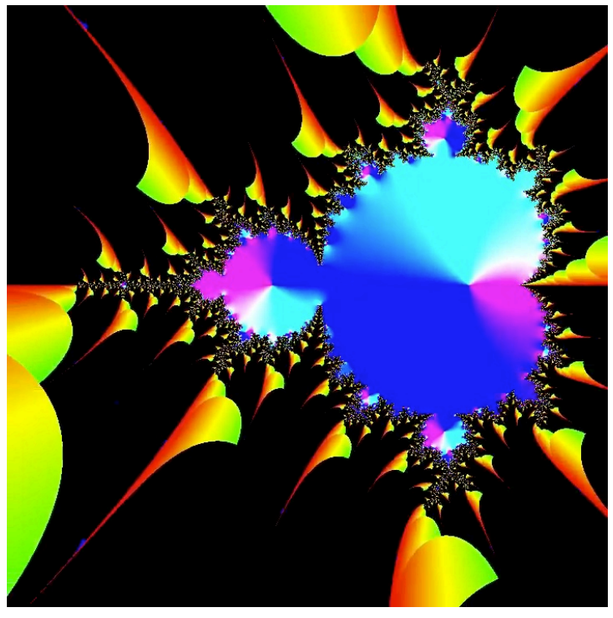
TSBK 07

Computer Graphics

Ingemar Ragnemalm, ISY



Lecture 4
3D graphics part 2

Today's topics:

Graphics pipeline and shaders
GLSL

Object representation (intro)

Visible surface detection (intro):
Z-buffer
Back-face culling

Transformations in 2D and 3D with homogenous coordinates

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

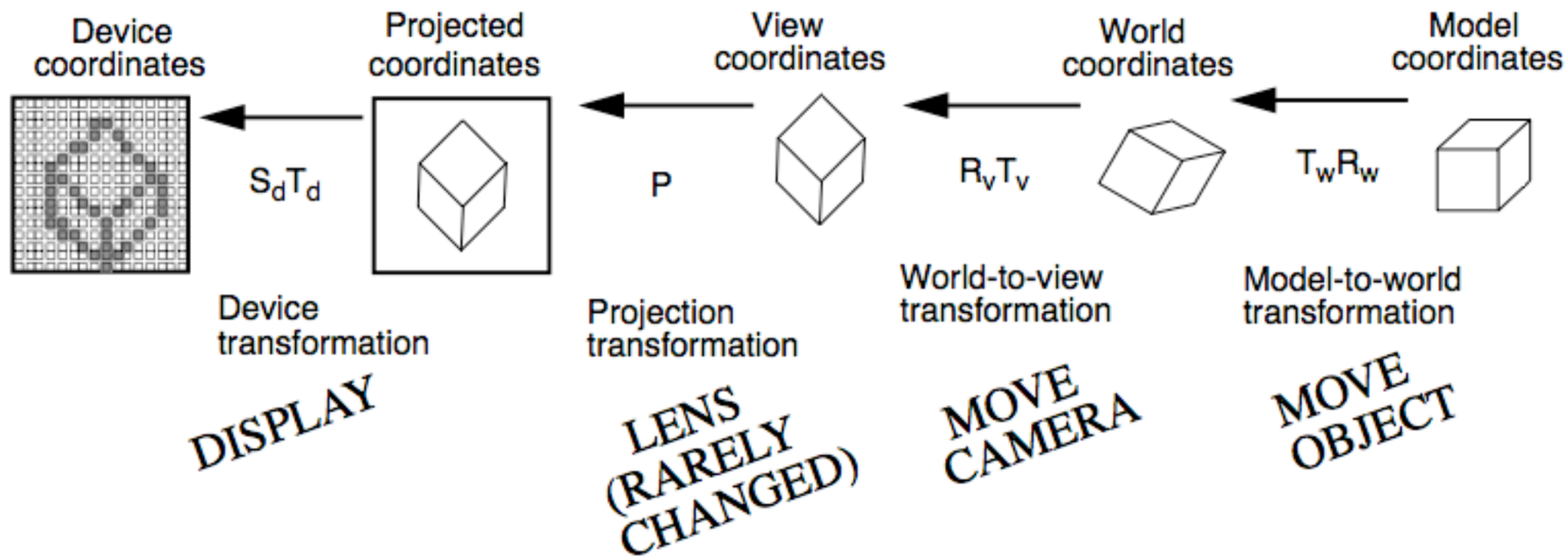
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformation pipeline

Model coordinates
World coordinates
View coordinates
Projected coordinates
Device coordinates



Projection by using the bottom row with homogenous coordinates

$$\begin{bmatrix} 2n/(r-l) & 0 & A & 0 \\ 0 & 2n/(t-b) & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

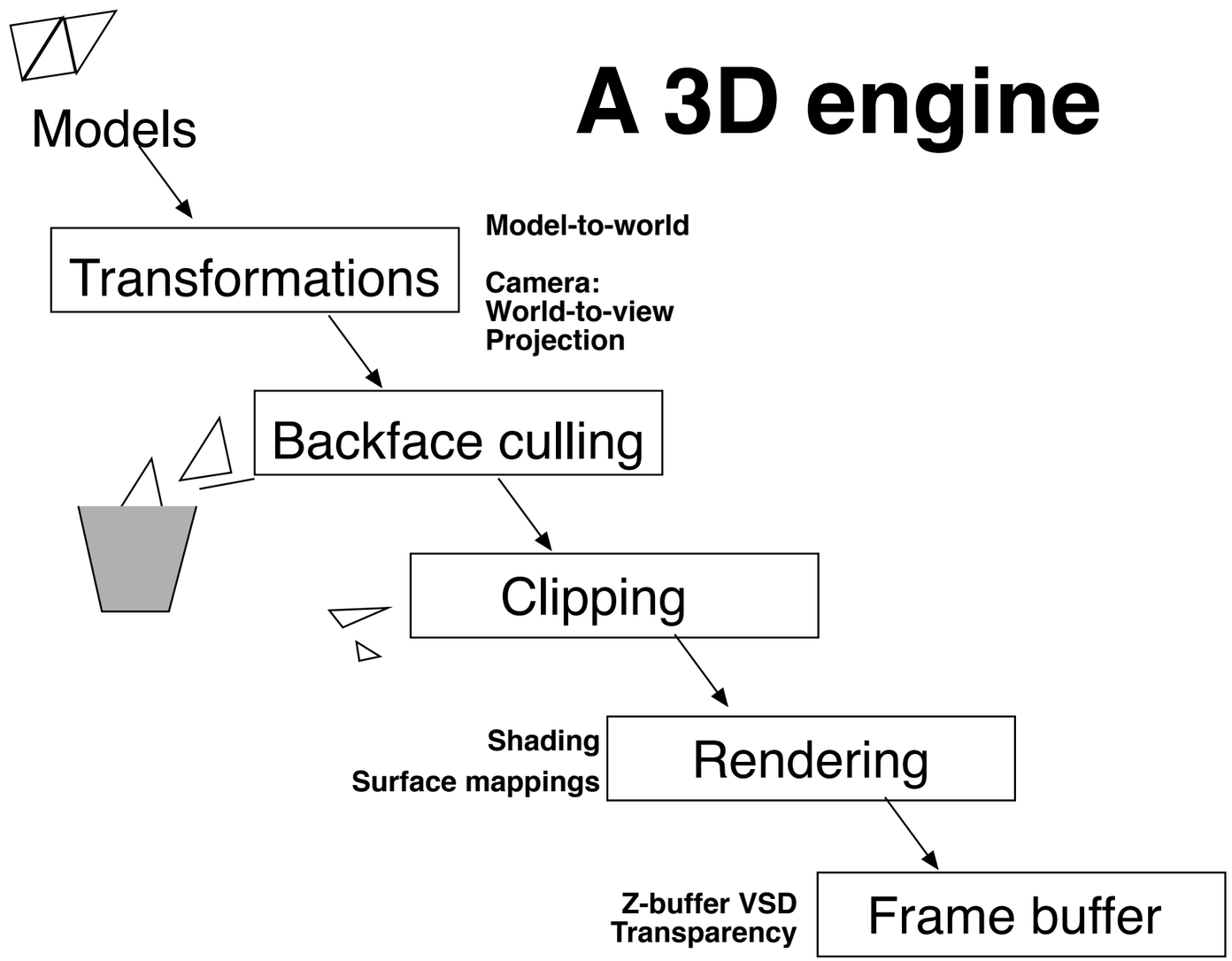
A, B usually zero

C = -(f+n)/(f-n)

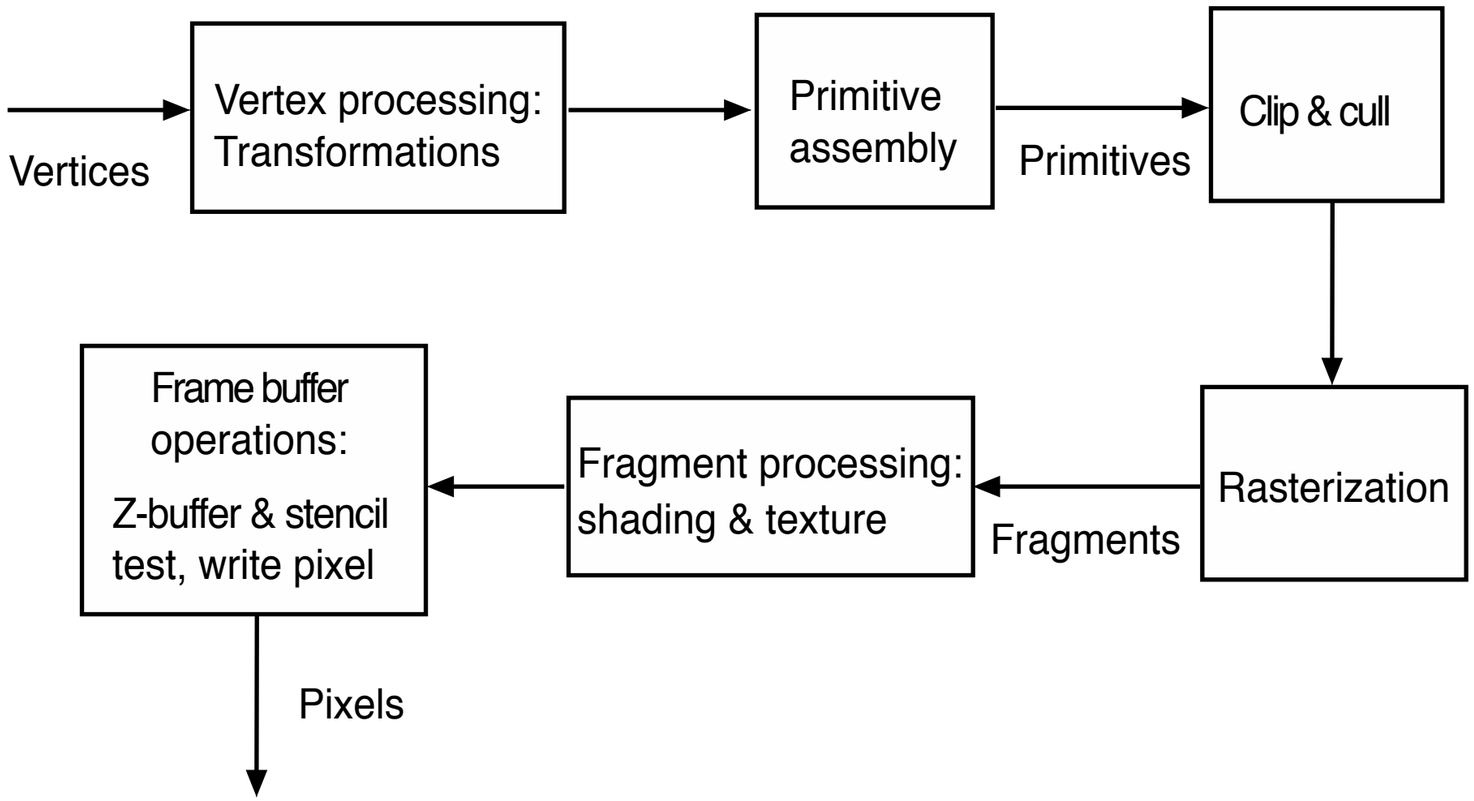
D = -2f*n/(f-n)

Normalized coordinates and viewing frustum

A 3D engine


Models
Transformations
Model-to-world
Camera:
World-to-view
Projection
Backface culling
Clipping
Shading
Surface mappings
Rendering
Z-buffer VSD
Transparency
Frame buffer

The OpenGL pipeline



So what is a shader?

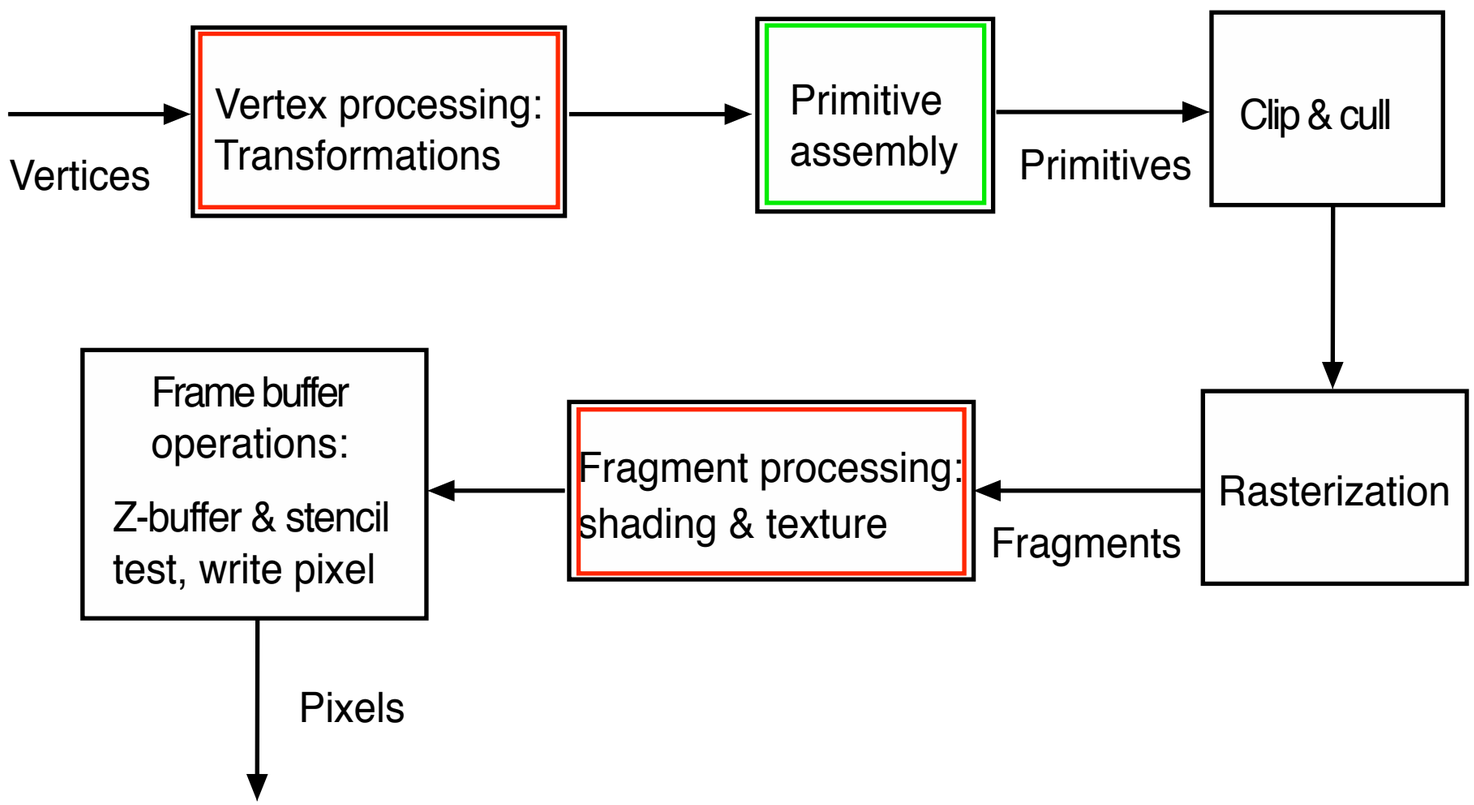
- **Small program (computing kernels)**

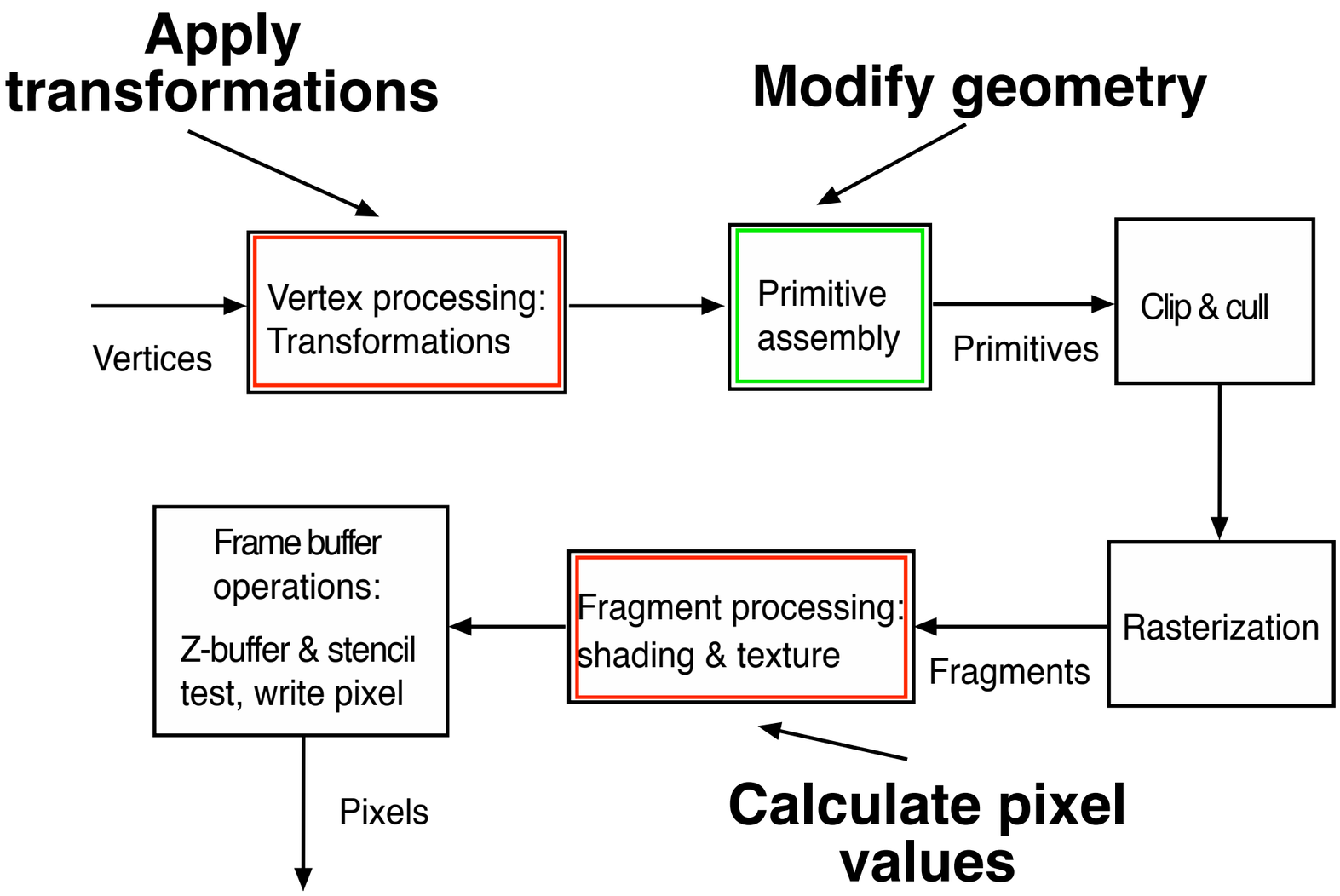
- **Performs caclulations for a specific stage in the graphics pipeline**

- **Runs natively on the GPU**

- **CPU orders rendering -> active shaders are invoked as part of the rendering**

Stages that may or must have shaders




Apply transformations
Modify geometry
Vertex processing: Transformations
Vertices
Primitive assembly
Primitives
Clip & cull
Rasterization
Fragments
Fragment processing: shading & texture
Frame buffer operations:
Z-buffer & stencil test, write pixel
Pixels
Calculate pixel values

More on shaders

- **Load from source and compile at runtime (e.g. at program launch or scene changes)**

- **Written in the language GLSL (OpenGL Shader Language)**

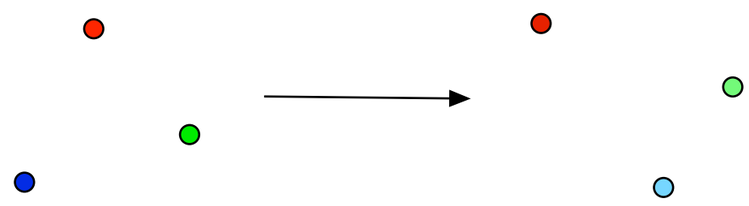
- **Runs in parallel for many data items at once**

- **Fast and flexible!**

The vertex processor/vertex shader

The vertex processor handles the following tasks:

- **Vertex transformation (model to projected)**
- **Transformation of normal vectors**
- **Generation/transformation of texture coordinates**
- **Lighting calculations (if not in fragment shader)**
- **Material parameters**
- **Send interpolated data to fragment shader**



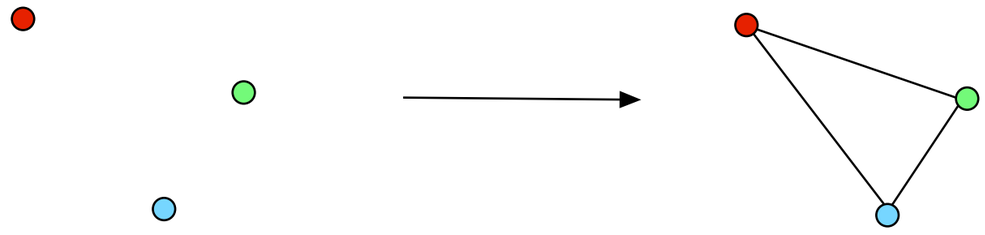
Primitive assembly

Assembly of primitives

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

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

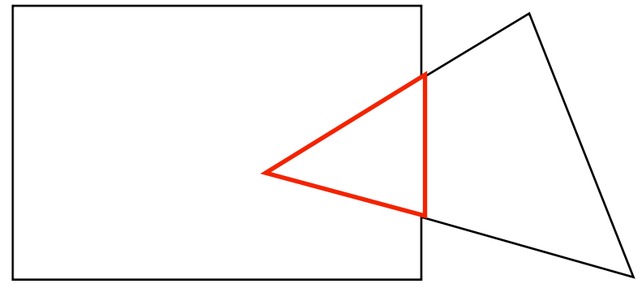
Geometry shader can alter geometry. (Optional.)



Clipping and culling

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

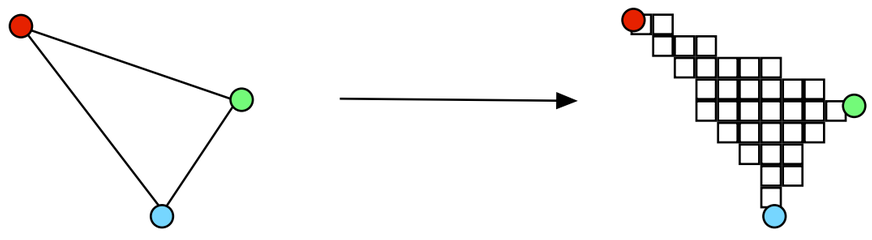
Note that texture coordinates also needs clipping (as well as any other data that is interpolated between vertices).



Raster conversion

Polygon rendering, convert polygons to pixel coordinates

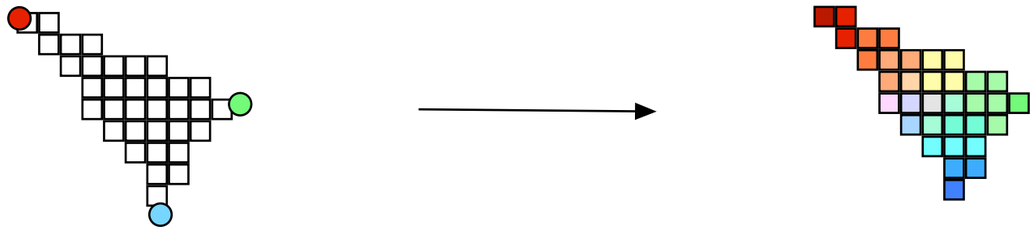
Creates “fragments”. Note that they do not have any colors yet!



The fragment processor/fragment shader

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

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



Frame buffer operations

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

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