A shader unit

Architecture, OpenGL-specific aspects, simulator implemented using SystemC, adaptions for embedded systems

Philipp Klaus Krause

January 22, 2008

Gliederung

1. Introduction
2. Architecture
3. Implementing OpenGL
4. Adaptions for use in embedded systems

Gliederung

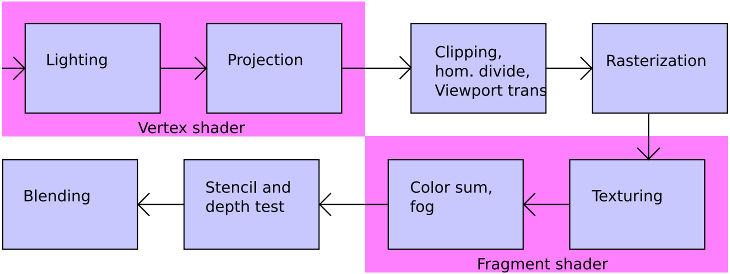
1 Introduction

2 Architecture

3 Implementing OpenGL

4 Adaptions for use in embedded systems

OpenGL pipeline



Lighting

Projection

Vertex shader

Clipping, hom. divide, Viewport trans

Rasterization

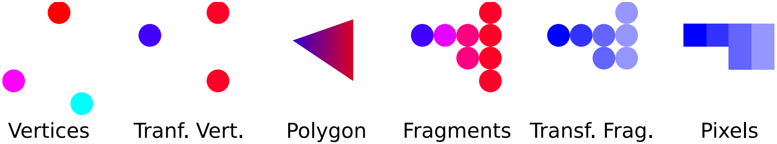
Texturing

Color sum, fog

Fragment shader

Stencil and depth test

Blending



Shaders

- Replace parts of the graphics pipeline to gain flexibility
- Written in special-purpose languages like GLSL
- Executed on special-purpose, programmable processors called shader units
- Unified shaders: Shader units can be dynamically assigned to different shader types
- Different architectures in use

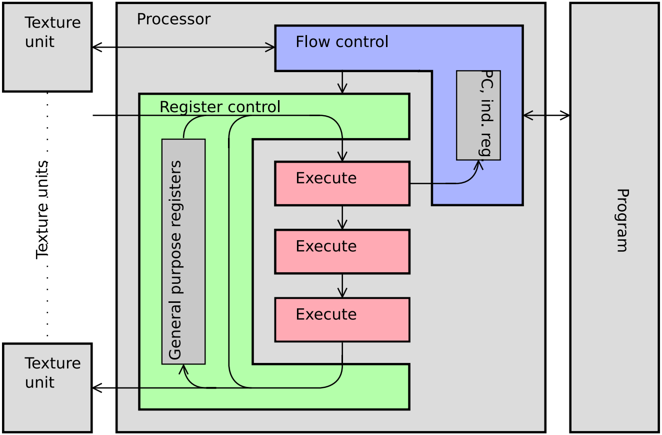
Goals

- Architecture of a shader unit (for OpenGL shaders)
- Selected aspects of hardware implementation
- Adaptions for embedded systems
- Cycle-accurate simulator written in SystemC
- Assembler

Gliederung

1. Introduction
2. Architecture
3. Implementing OpenGL
4. Adaptions for use in embedded systems

Processor and parts of its environment



Execution environment

- (Up to) 240 general-purpose registers
- (Up to) 14 texture units
- 2 index registers
- Single address space for general-purpose registers, texture units, indirect addressing
- 64K program address space
- Program counter

General-purpose registers

- 128 bit wide, treated as 4-component vectors by most instructions
- Data is passed from and to the rest of the graphics pipeline through these

Instructions

Typical instructions:

- One or two source operands
- One destination operand
- Destination write mask to selectively write components of destination operand

Example: Double the first and third component of register 1

```
add 1.xz, 1, 1
```

Dot products

- Common in shaders
- Can be used for other tasks: Matrix multiplication, Taylor series
- Map well to a pipeline that has three execution stages

Gliederung

1. Introduction
2. Architecture
3. Implementing OpenGL
4. Adaptions for use in embedded systems

OpenGL

- Drivers generate shaders for fixed-function pipeline
- Legacy pseudo-assembly languages' functionality is a subset of GLSL functionality
- GLSL has many built-in functions

Example - cosinus

$$\cos(c) = \sum_{i=0}^{\infty} (-1)^n \frac{x^{2i}}{(2i)!} \tag{1}$$

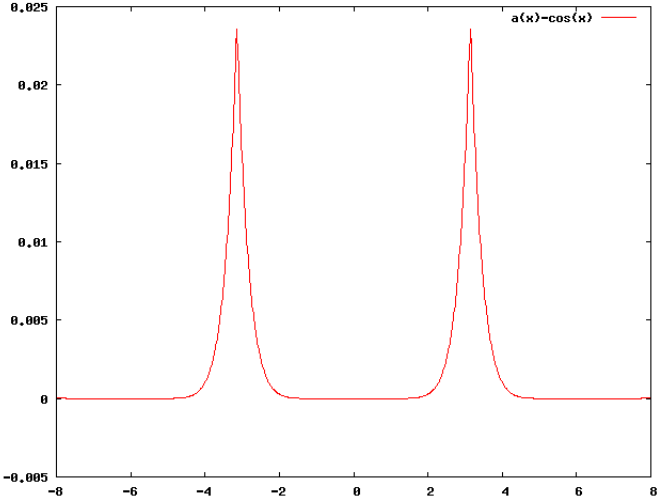
$$\cos(x) = t(g(\frac{x}{2\pi} + \frac{1}{2}) - \frac{1}{2}) \tag{2}$$

$$g(x) = x - \lfloor x \rfloor \tag{3}$$

$$t(x) = \sum_{i=0}^{\infty} (-1)^n \frac{(2\pi x)^{2i}}{(2i)!} \approx \sum_{i=0}^{4} (-1)^n \frac{(2\pi x)^{2i}}{(2i)!} = s(x) \tag{4}$$

$$\cos(x) \approx a(x) := s(g(\frac{x}{2\pi} + \frac{1}{2}) - \frac{1}{2}) \tag{5}$$

Cosinus approximation error



Gliederung

1 Introduction

2 Architecture

3 Implementing OpenGL

4 Adaptions for use in embedded systems

OpenGL ES

- OpenGL for embedded systems
- Removes legacy functionality
- Removes highend features

Architecture

- Often no hardware-accelerated vertex processing
- Reduced (half) precision is sufficient for fragment shaders
- 64 bit wide general-purpose registers
- Indirect adressing and integer support not mandatory

Texturing

Moving functionality to shaders results in simpler texture units

- Texture filtering (texture units support bilinear filtering, shader does trilinear filtering)
- Calculation of level-of-detail paramater from texture coordinates derivative's