FACHHOCHSCHULE **TRIER**
Hochschule für Technik, Wirtschaft und Gestaltung
University of Applied Sciences
Informatik

# Spielekonsolenprogrammierung

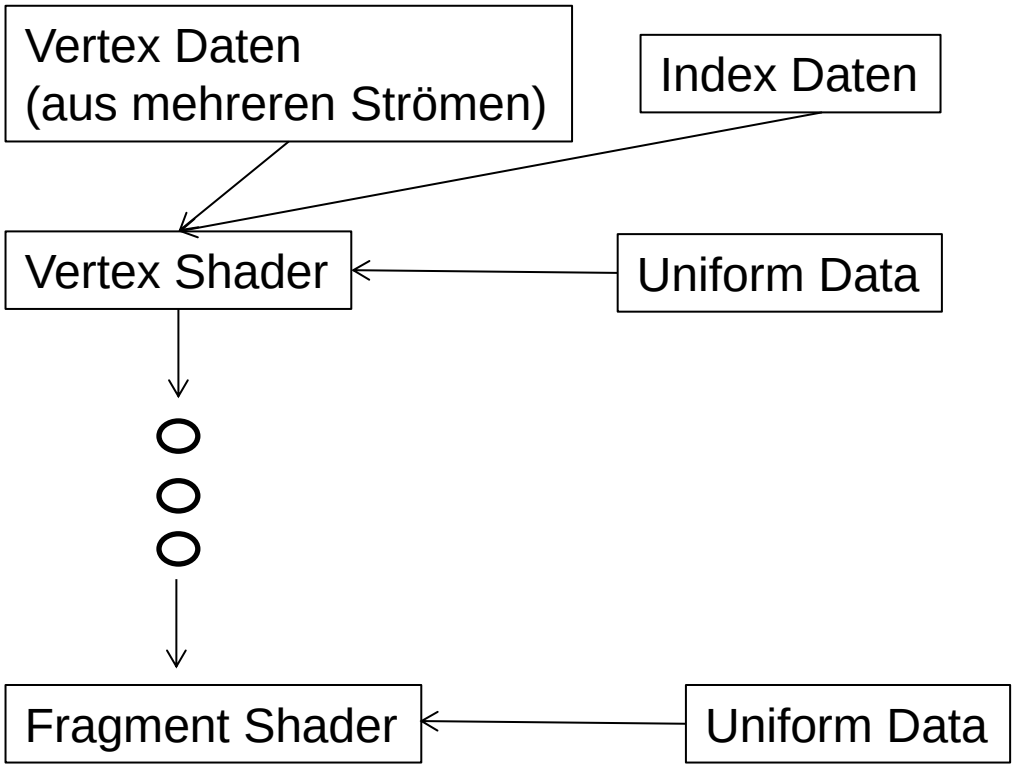
## Shaderprogrammierung, Vertices und ein Würfel

# Übersicht



- Zusammenfassung der Idee
- Beschreibung eines Vertexaufbaus
- Input Parameter für einen Fragmentshader
- Shader Programmierung in CG
- Setzen der Renderstates für den Tiefentest
- Zeichnen eines Würfels
- Zeichnen eines Würfels mit Beleuchtung

# Zusammenfassung der Idee



# Vertexaufbau



- Vertices können beliebige Daten enthalten
  (Position muss nicht enthalten sein)

- Vertices können aus verschiedenen Strömen zusammengemischt werden (B.: Beleuchtungsinformation)
- Vertex Streams werden vor dem Zeichnen (**sceGxmSetVertexStream**) gesetzt.

# Vertexaufbau



- Streams geben an mit (**SceGxmVertexStream** )

- Mit welchem Datentyp wird der Stream indiziert (16/32 bit, instanced / normal).
- Wie groß ist die Datenportion je Vertex im Stream (stride)
- Vertexaufbau (nächste Folie)
- Beides wird benutzt in **sceGxmShaderPatcherCreateVertexProgram**

# Vertexaufbau

- Vertex Struktur (wird Array von gebildet um einen Vertex zu beschreiben :

```
#include <gxm/structs.h>
typedef struct SceGxmVertexAttribute {
    uint16_t streamIndex;
    uint16_t offset;
    uint8_t format;
    uint8_t componentCount;
    uint16_t regIndex;
} SceGxmVertexAttribute;
```

Zu welchem Stream?

Anzahl der Bytes vom Start

Format nächste Seite

Komponenten 1-4

Verwendung im Vertexprogramm (CG)

# Attribute Type

Komponenten:

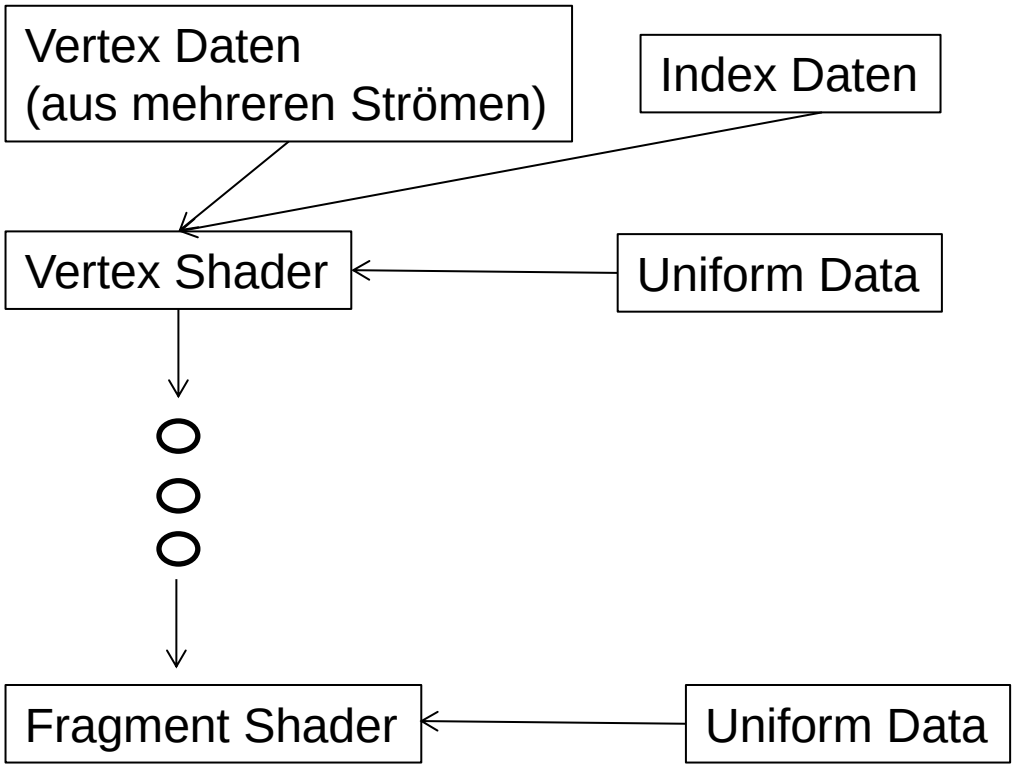
| Macro | Value | Description |
|---|---|---|
| SCE_GXM_ATTRIBUTE_FORMAT_U8 | N/A | 8-bit unsigned integer |
| SCE_GXM_ATTRIBUTE_FORMAT_S8 | N/A | 8-bit signed integer |
| SCE_GXM_ATTRIBUTE_FORMAT_U16 | N/A | 16-bit unsigned integer |
| SCE_GXM_ATTRIBUTE_FORMAT_S16 | N/A | 16-bit signed integer |
| SCE_GXM_ATTRIBUTE_FORMAT_U8N | N/A | 8-bit unsigned integer normalized to [0,1] range |
| SCE_GXM_ATTRIBUTE_FORMAT_S8N | N/A | 8-bit signed integer normalized to [-1,1] range |
| SCE_GXM_ATTRIBUTE_FORMAT_U16N | N/A | 16-bit unsigned integer normalized to [0,1] range |
| SCE_GXM_ATTRIBUTE_FORMAT_S16N | N/A | 16-bit signed integer normalized to [-1,1] range |
| SCE_GXM_ATTRIBUTE_FORMAT_F16 | N/A | 16-bit half precision floating point |
| SCE_GXM_ATTRIBUTE_FORMAT_F32 | N/A | 32-bit single precision floating point |
| SCE_GXM_ATTRIBUTE_FORMAT_UNTYPED | N/A | 32-bit untyped data for use with offline vertex unpack |

Typ muss nicht mit der Verwendung im Shader übereinstimmen (Konvertierung->Float)

# Uniform Data

- Uniform Data
  - wird für die Verarbeitung aller Vertices verwendet.
  - Register erfragen mit **sceGxmProgramFindParameterByName**
  - Speicher zum Hochladen erfragen mit **sceGxmReserveVertexDefaultUniformBuffer**
  - Befüllen des Speicher mit **sceGxmSetUniformDataF**

# Zustand


Vertex Daten
(aus mehreren Strömen)
Index Daten
Vertex Shader
Uniform Data
Fragment Shader
Uniform Data

# Datenübergabe

- Übergabe Vertex Shader -> Interpolator -> Fragmentshader ist nicht mehr so frei
- Angabe als „Semantik“ des Parameters

- Vertex Shader:

```
void main(
float3 aPosition,
float4 aColor,
float4 out vPosition : POSITION,
float4 out vColor : TEXCOORD0)
```

- Fragment shader:

```
float4 main(float4 vColor : TEXCOORD0)
{
return vColor;
}
```

# Semantik der Parameter

- Relevante Vertex output Semantiken:
  - POSITION: Position
  - TEXCOORD0-TEXCOORD9: Texturkoordinaten
  - COLOR0-COLOR1: Farbe (Byte interpolation)

- Fragment shader input semantiken:
  - WPOS: Windows position (Spez)
  - FRAGCOLOR: Existierende Backbuffer Farbe (Spez)
  - TEXCOORD0-TEXCOORD9: Texturkoordinaten
  - COLOR0-COLOR1: Farbe (Byte interpolation)

# CG Übersicht



- Ist sehr GLSL und C ähnlich
- Variablen werden bezeichnet mit
  - out : Ausgabeparameter
  - uniform: Uniform parameter

- Von allen Basistypen (int, float, half) gibt es Vektorvarianten (int2, int3, int4)
- Subkomponenten können angesprochen werden(bla.xy)
- Swizzling ist möglich (bla.yx)

- CG verwendet normalerweise Row Major Notierung!!

# Aufgabe 3D Würfel



- AUFGABE 1: Ziel ist es aus der Dreiecksaufgabe einen rotierenden 3D Würfel zu machen
- Im ersten Ansatz unbeleuchtet
- Jede Seite eigene Farbe

- Tiefentest ist normalerweise eingeschaltet
- Bauen Sie einen Würfel aus 4 Seiten auf
- Jede Seite hat 4 Farben
- Benutzen Sie die Matrixfunktionen lookAt, perspective, um die Matrix zu zeichnen!
- Machen Sie die Rotation vom linken Joystick abhängigs

# Aufgabe 3D Würfel



- Wir wollen der Würfel beleuchten
- Beleuchtung nach einfachem D‘Allembertschen Gesetz

- Codieren Sie noch eine Normale in den Vertex rein
- Die Normale muss rotiert werden
- Beleuchtungsrichtung kommt aus der Sichtquelle

# Aufgabe 3D Würfel



- Zu ändern:
  - Vertexformat muss angepasst werden
  - Normalen für die Vertices müssen erzeugt werden
  - Rotationsmatrix muss an uniform Variable gebunden und hochgeladen werden
  - Vertex Shader muss angepasst werden

- AUFGABE 2

# Zusammenfassung



- Zusammenfassung der Idee
- Beschreibung eines Vertexaufbaus
- Input Parameter für einen Fragmentshader
- Shader Programmierung in CG
- Setzen der Renderstates für den Tiefentest
- Zeichnen eines Würfels
- Zeichnen eines Würfels mit Beleuchtung