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

Spielekonsolenprogrammierung

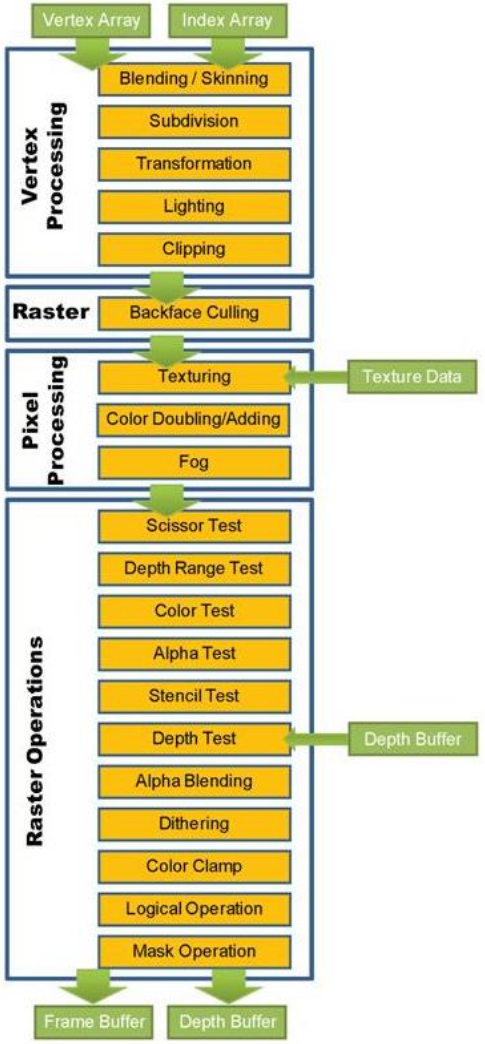
Graphikhardware

Übersicht

- Erläuterung der Graphik Hardware
- Initialisierung der Graphik
- Kernrendering
- Aufbau eines Miniprogrammes

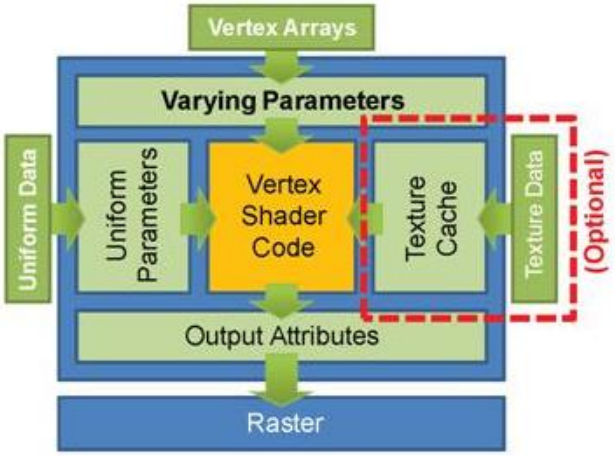
Graphik Hardware

- Grundlegende Idee:
  - Alte Hardware:
    - Fixed Function Pipeline
    - Wurde auf der PSP verwendet
    - Aus der CG Vorlesung bekannt

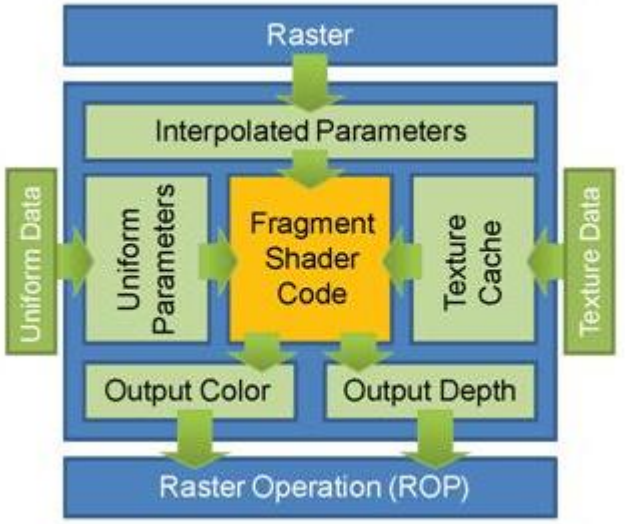


Graphik Hardware

Vertex shader



Pixel / Fragment Shader



PS Vita Spezialität: Backbuffer Farbe kann im Pixelshader ausgelesen werden.

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

- Feature Liste:

- 4 Unified Shader Cores
- Tile Based Deferred Rendering (jedes Tile hat eigenen Speicher), Hidden Surface Removal
- 2D Texturen bis 4096*4096, Cubemaps, alle Texturfilter, Fließkommatexturen, div. Kompressionsformate
- Rendertargets: MSAA (Anti-Aliasing), Fließkomma render targets, auch bis 4096*4096

Graphik Hardware

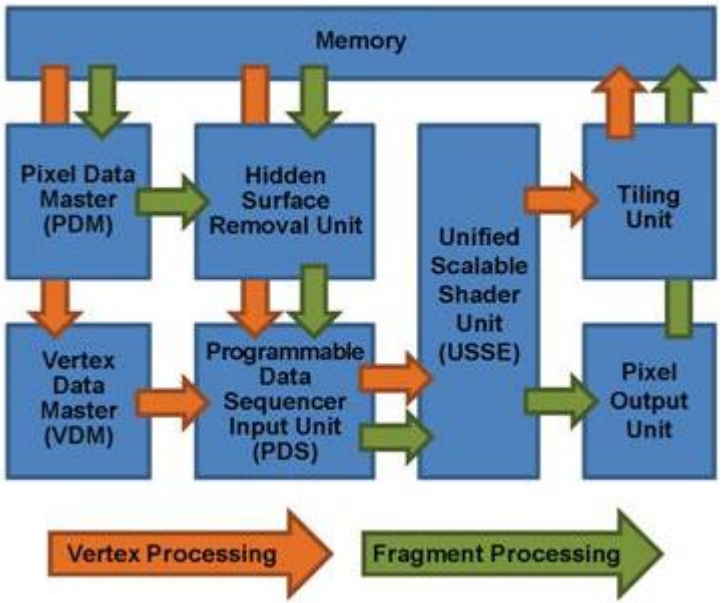
- Datenfluss durch das System:

VDM: Verarbeitet Zeichenbefehlssequenz der mit Lib Erzeugt wird

PDM: Steuert Rasterisigung bekommt Informationen der Tiling Unit

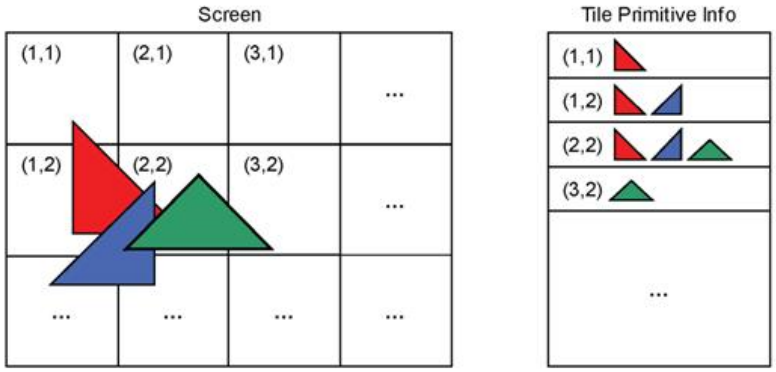
PDS: Steuereinheit der Shader unit

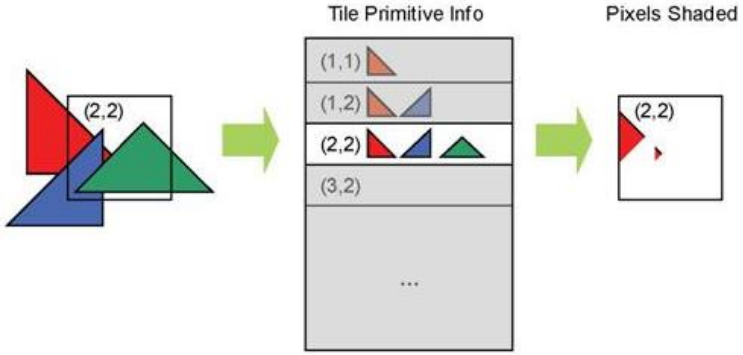
Tiling Unit: Aufteilung der Dreiecke in Tiles



Graphik Hardware

- Tile Based Rendering





Hidden surface removal beim Opaque rendering

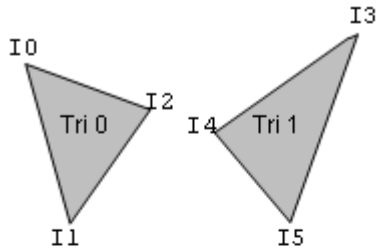
Cache freundliche Struktur

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

- Die Primitive, die gezeichnet werden können sind alle indiziert
  - Point List
  - Line List
  - Triangle List
  - Triangle Strip
  - Triangle Fan
  - Triangle Edge List

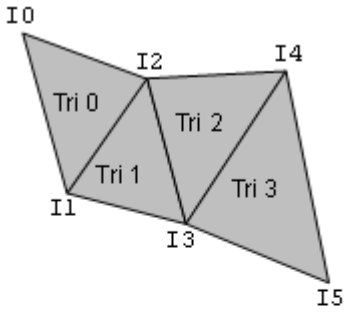
Graphik Hardware

- Triangle List



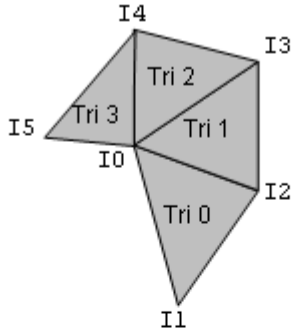
16-bit or 32-bit Value
Index 0 (Tri 0)
Index 1 (Tri 0)
Index 2 (Tri 0)
Index 3 (Tri 1)
Index 4 (Tri 1)
Index 5 (Tri 1)
...

- Triangle Strip



16-bit or 32-bit Value
Index 0 (Tri 0)
Index 1 (Tri 0, Tri 1)
Index 2 (Tri 0, Tri 1, Tri 2)
Index 3 (Tri 1, Tri 2, Tri 3)
Index 4 (Tri 2, Tri 3)
Index 5 (Tri 3)
...

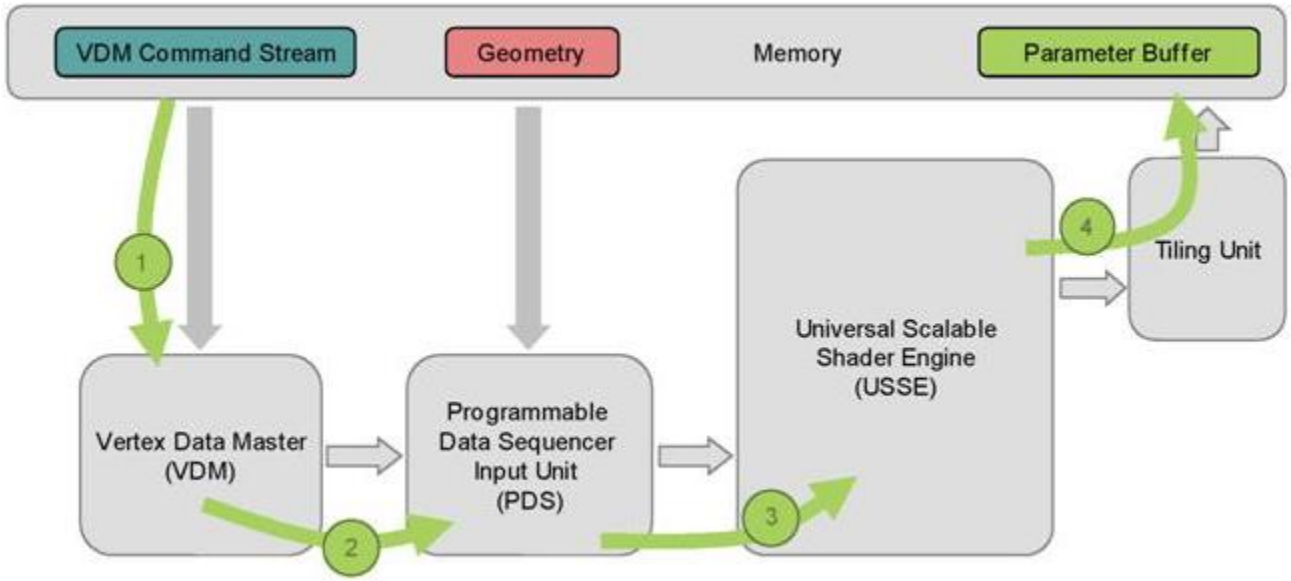
- Triangle Fan



16-bit or 32-bit Value
Index 0 (Tri 0, Tri 1, Tri 2, Tri 3)
Index 1 (Tri 0)
Index 2 (Tri 0, Tri 1)
Index 3 (Tri 1, Tri 2)
Index 4 (Tri 2, Tri 3)
Index 5 (Tri 3)
...

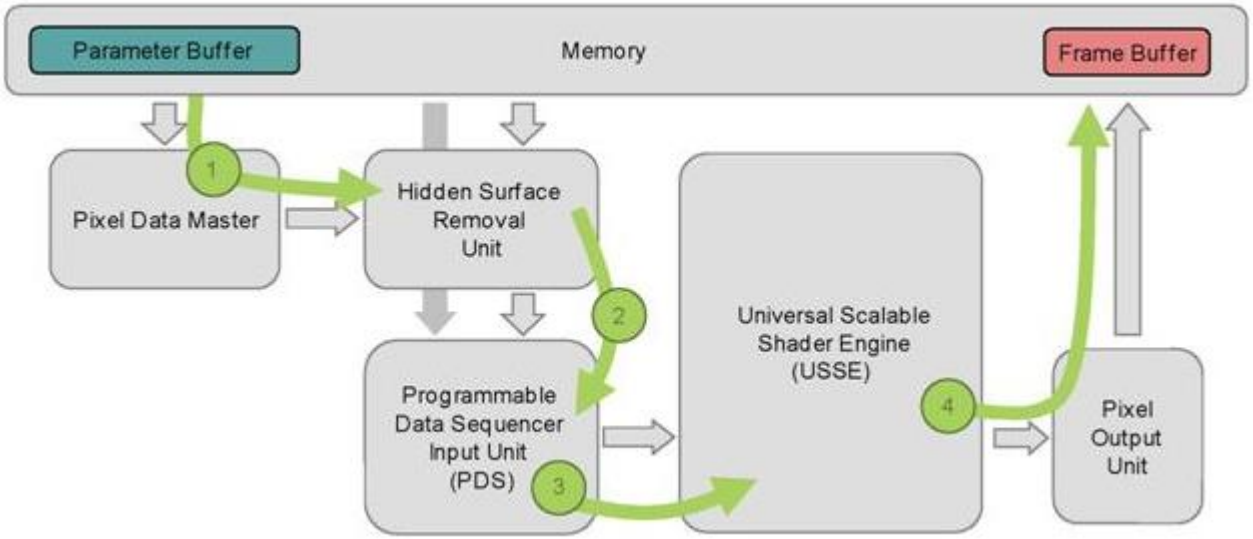
Graphik Hardware

- Vertex Processing:



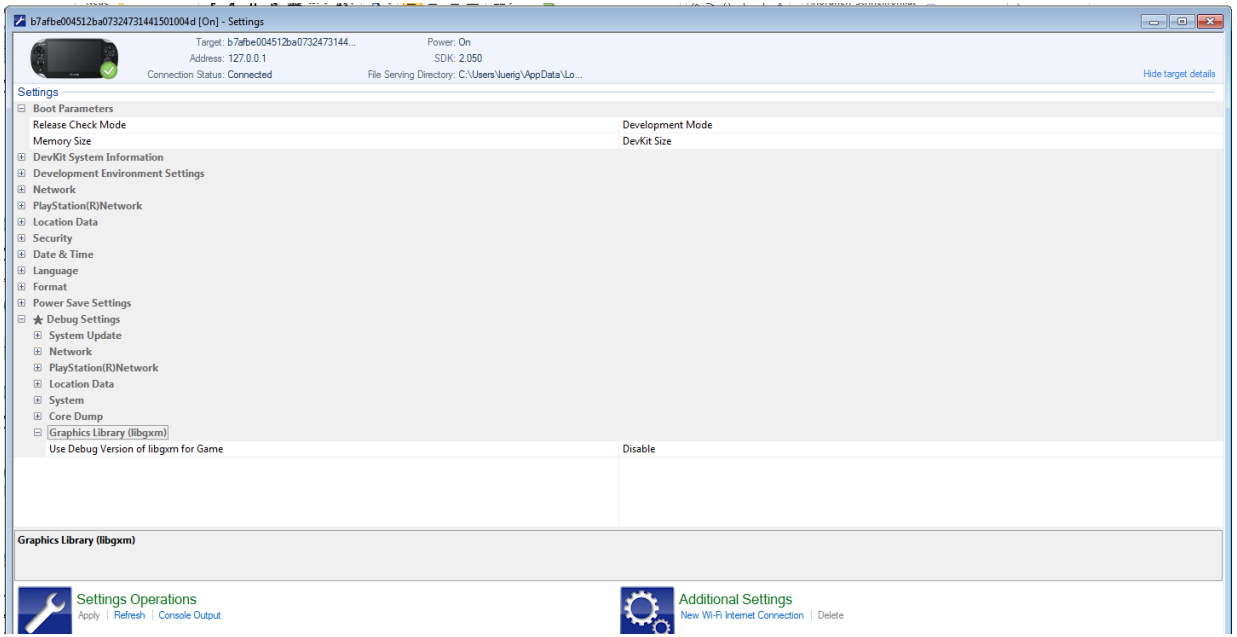
Graphik Hardware

- Pixel Data



Graphik Lib

- Graphik Programmierung erfolgt mit der libgxm
- Lib kann auf dem Dev-Kit in einen Debug Modus geschaltet werden:



Graphik Lib

- Initialisierung und das Framework der Lib ist etwas umständlich

1. Initialisierung der Library im wesentlichen mit Angabe des Vblank callbacks (sceGxmInitialize())
2. Erzeugung eines Renderkontextes (sceGxmCreateContext())
   - Mit seiner Hilfe kann eine Szene auf ein Rendertarget abgebildet werden
   - Verschiedene Speicher müssen der Lib zur Verfügung gestellt werden
   - Default Werte vorhanden aber etwas black art.
   - Viele der Speicherarten müssen für die Benutzung durch die GPU freigegeben werden (**sceGxmMapMemory**).

Graphik Lib

- Anzugebene Werte

Puffer	Beschreibung
VDM Ring Buffer	Puffer für Zeichenbefehle (mehrere 100kb)
Vertex Ring Buffer	Puffer für Vertex Uniform Data und PDS Programme (ein paar MB)
Fragment Ring Buffer	Puffer für Fragment Uniform Data und PDS Programme (ein paar MB)
Fragment USSE Ring Buffer	Dynamisch erzeugter Fragment USSE Code (ein paar 10kb)

Graphik Lib

- Anlegen mehrerer Displaybuffer und eines Tiefen/Stencil Buffers
  - sceGxmColorSurfaceInit Speicher muss in CDRAM sein
  - sceGxmDepthStencilSurfaceInit muss in normalem Speicher angelegt sein
  - Bei beiden Befehlen müssen diverse Speicheralignment Aspekte beachtet werden.
- Displaybuffer stellen Front und Backbuffer (mindestens 2 müssen vorhanden sein)

- Auch ein einfaches Löschen des Bildschirmes muss als Rendering durchgeführt werden
- Kann durch Zeichnen eines Dreieckes geschehen

- Zu diesem Zweck muss vorbereitet werden:
  1. Shader Patcher (bindet einen Shader)
  2. Vertex Stream muss definiert und an ShaderPatcher gebunden werden
  3. Geometrie (Vertex und Index Information muss aufgebaut werden).

Graphik Lib

- Shader für das Clear sind denkbar einfach (und müssen mit CG Compiler übersetzt werden, Datei: Benutzerdefiniertes Build Tool).

Vertex

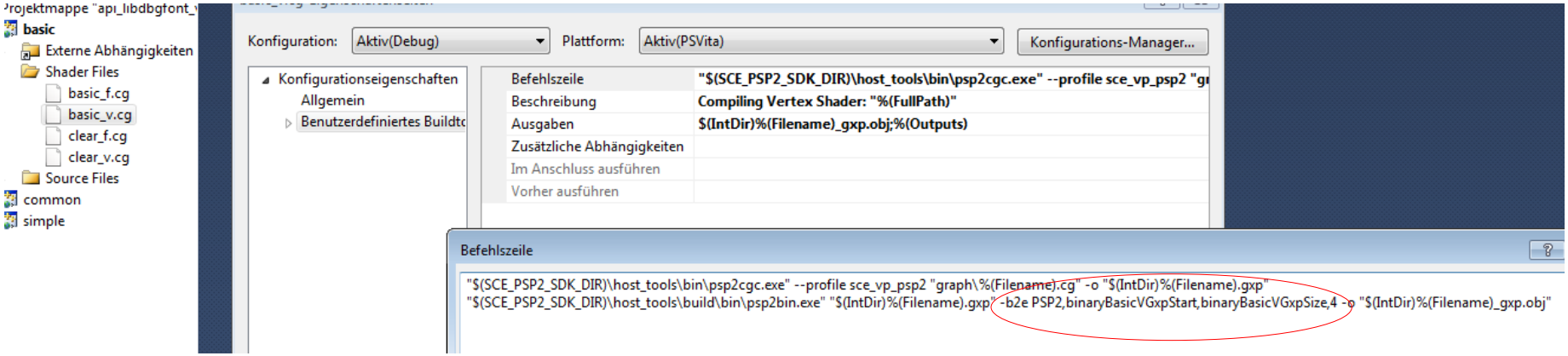
```
float4 main(
float2 aPosition
) : POSITION
{
return float4(aPosition, 1.f, 1.f);
}
```

Fragment

```
float4 main() : COLOR
{
return 0.f;
}
```

Graphik Lib

- Compilieren der Shader (in CG)



- Shader werden vom Linker gebunden und können referenziert werden.

```
extern const SceGxmProgram binaryBasicVGxpStart;
extern const SceGxmProgram binaryBasicFGxpStart;
```

Graphik Lib

- Anbindung des Shaders geschieht mit der Shader Patcher API (**sceGxmShaderPatcherCreate**)
- Diese braucht entweder vorkonfigurierten Speicher oder einen Speichermanager für
  - verschiedene USSE Codes
  - Parameterbereiche
- Meist werden vorkonfigurierte Speicherbereiche verwendet.

Graphik Lib

- Shader Patcher API
  - Binde Vertex and Fragment programme (**sceGxmShaderPatcherRegisterProgram**)
  - Shader parameters können erfragt werden (**sceGxmProgramFindParameterByName**)
  - Vertexshader mit Vertexstromdefinitionen können erzeugt werden (**sceGxmShaderPatcherCreateVertexProgram**)
  - Ebenso können Fragmentshader gebunden werden (**sceGxmShaderPatcherCreateFragmentProgram**)

Graphik Lib

- Weiterhin werden Synchronisatiosobjekte für die Framebuffer benötigt (**sceGxmSyncObjectCreate**)

- Grundlegende Idee:
  - Umschalten der Buffer (Front Back) geschieht über Interrupt
  - Ein zu schnelles Zeichnen wird über die Synchronisationsobjekte vermieden

Graphik Lib

- Grundsätzlicher Ablauf des Zeichenvorganges

```
/* start rendering to the render target */
sceGxmBeginScene( s_context, 0, s_renderTarget, NULL, NULL, s_displayBufferSync[s_displayBackBufferIndex],
&s_displaySurface[s_displayBackBufferIndex], &s_depthSurface );

/* set clear shaders */
sceGxmSetVertexProgram( s_context, s_clearVertexProgram );
sceGxmSetFragmentProgram( s_context, s_clearFragmentProgram );

/* draw ther clear triangle */
sceGxmSetVertexStream( s_context, 0, s_clearVertices );
sceGxmDraw( s_context, SCE_GXM_PRIMITIVE_TRIANGLES, SCE_GXM_INDEX_FORMAT_U16, s_clearIndices, 3 );

/* stop rendering to the render target */
sceGxmEndScene( s_context, NULL, NULL );
```

- Der Callback am Ende des Renderns wird so angefordert:

```
/* PA heartbeat to notify end of frame */
sceGxmPadHeartbeat( &s_displaySurface[s_displayBackBufferIndex], s_displayBufferSync[s_displayBackBufferIndex] );

/* queue the display swap for this frame */
DisplayData displayData;
displayData.address = s_displayBufferData[s_displayBackBufferIndex];

/* front buffer is OLD buffer, back buffer is NEW buffer */
sceGxmDisplayQueueAddEntry( s_displayBufferSync[s_displayFrontBufferIndex],
                    s_displayBufferSync[s_displayBackBufferIndex], &displayData );

/* update buffer indices */
s_displayFrontBufferIndex = s_displayBackBufferIndex;
s_displayBackBufferIndex = (s_displayBackBufferIndex + 1) % DISPLAY_BUFFER_COUNT;
```

Graphik Lib

- Der Callback selber sieht so aus:

```
SceDisplayFrameBuf framebuf;

/* cast the parameters back */
const DisplayData *displayData = (const DisplayData *)callbackData;


/* wwap to the new buffer on the next VSYNC */
memset(&framebuf, 0x00, sizeof(SceDisplayFrameBuf));
framebuf.size        = sizeof(SceDisplayFrameBuf);
framebuf.base        = displayData->address;
framebuf.pitch       = DISPLAY_STRIDE_IN_PIXELS;
framebuf.pixelformat = DISPLAY_PIXEL_FORMAT;
framebuf.width       = DISPLAY_WIDTH;
framebuf.height      = DISPLAY_HEIGHT;
int returnCode = sceDisplaySetFrameBuf( &framebuf, SCE_DISPLAY_UPDATETIMING_NEXTVSYNC );
SCE_DBG_ALWAYS_ASSERT( returnCode == SCE_OK );

/* block this callback until the swap has occurred and the old buffer is no longer displayed */
returnCode = sceDisplayWaitVblankStart();
```

Graphik Lib

- Aufbau eines Miniprogrammes
- Schauen wir uns mal ein Kernprogramm an in dem alles eingebaut ist…
- Boilerplate Code ist der Aufwendigste ☺

Zusammenfassung

- Erläuterung der Graphik Hardware
- Initialisierung der Graphik
- Kernrendering
- Aufbau eines Miniprogrammes