Where Hacking Meets Demoscene

Abusing Kiddie-Hardware for the Greater Good
Breakpoint 2008

Felix “tmbinc” Domke <tmbinc@elitedvb.net>

- True 64bit architecture
- 3 Dual-Thread cores, 3.2 Ghz each
- ½ GB RAM
- Fast DVD-drive
- Large and fas DD
- Full-H t
- ort
- oad
- with embedded
- ed memory and Linux support

The perfect demo machine?

starting at $199!

available NOW!

The perfect demoscene machine?

- (Official) developers get “development kits” for big money (and big NDAs)
  - Those systems execute “unsigned” (or self-signed) code
  - Full toolchain/libs/docs included.

- On Retail machines, security mechanisms ensure that only “signed” code can be executed
- Gaming Consoles vendors are afraid of
  - piracy
  - cheating
  - non-licensed developers
- So they don’t let you use their products (except for very limited exceptions: commercial games).

Why do they spoil our fun?!

- Hardware is often subsidized

  (you don’t really believe $199 MSRP is enough for such a hardware, don’t you?)

- Money comes from the games.
- Vendors have absolutely no interest in selling hardware for demosceners

“Just some obfuscation”?

- State-of-the-art security schemes, including memory encryption, memory hashing, secure bootloaders, e-Fuses / on-chip non-volatile memory, on-chip RAM/ROM, NX, hypervisors.
- Quite a lot of effort these days will be put into security.

Should that stop us?

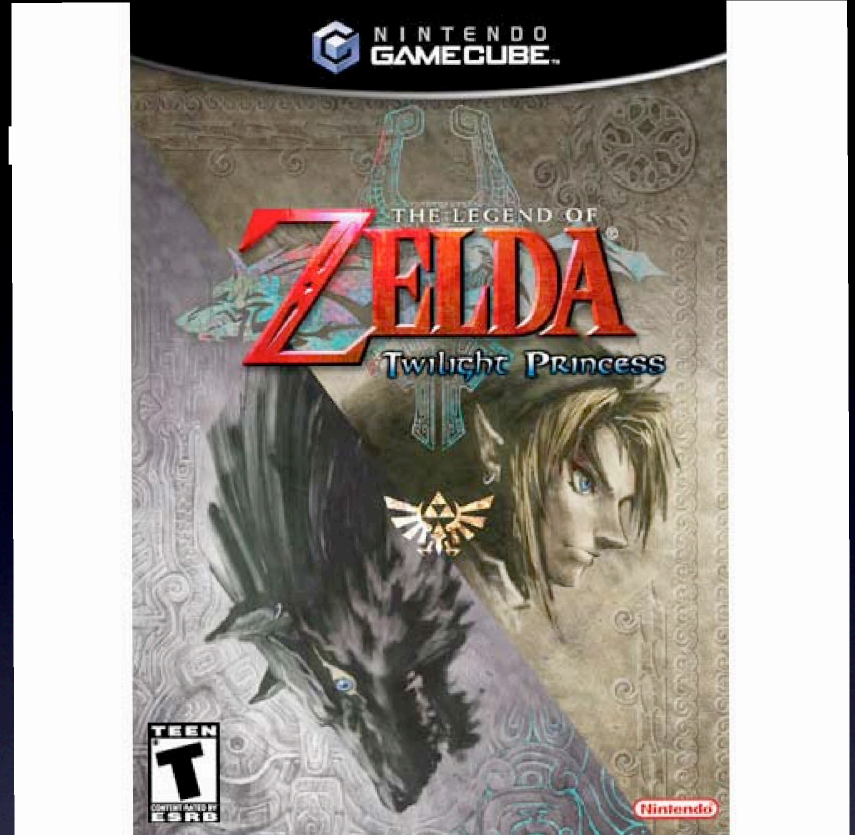
- Of course not! (Since when do we listen to vendors?!)
- Hackers so far always found ways around their security systems, allowing own code to be executed on retail systems.

Nintendo Wii

- “Gamecube 1.5”, but still:
- 729 MHz PowerPC CPU (IBM 750CL)
- EDTV (480p) graphics
- 24MB+64MB RAM
- Bluetooth, WiFi, USB, Gamecube legacy interfaces
- 250€ MSRP

Savegame Exploit

- Easiest way:
  - called “Twilight Hack” by Team Twiizers.
  - Uses “Zelda: Twilight Princess”, a top-selling first-party game.
  - Allows you to execute own, self-written code in Wii-mode.



- Wii
- Zelda: Twilight Princess
- SD-Card
- That's it!

Wii, Security?

- Security scheme is interesting, but totally broken once you disassemble it.
- Their RSA implementation is broken.
- No protection against stack overflows in games (savegame exploits FTW).
- “Security processor” firmware has many obvious, exploitable bugs (once you’re in, you own it. Completely.).

“Twilight Hack”

- Savegame exploit: Giving Link’s horse an overlong name crashes the game.
- Exploiting that can run code.
- You can restore savegames from SD card.
- Running Z:TP after that will run the exploit.
- So, let’s do that!

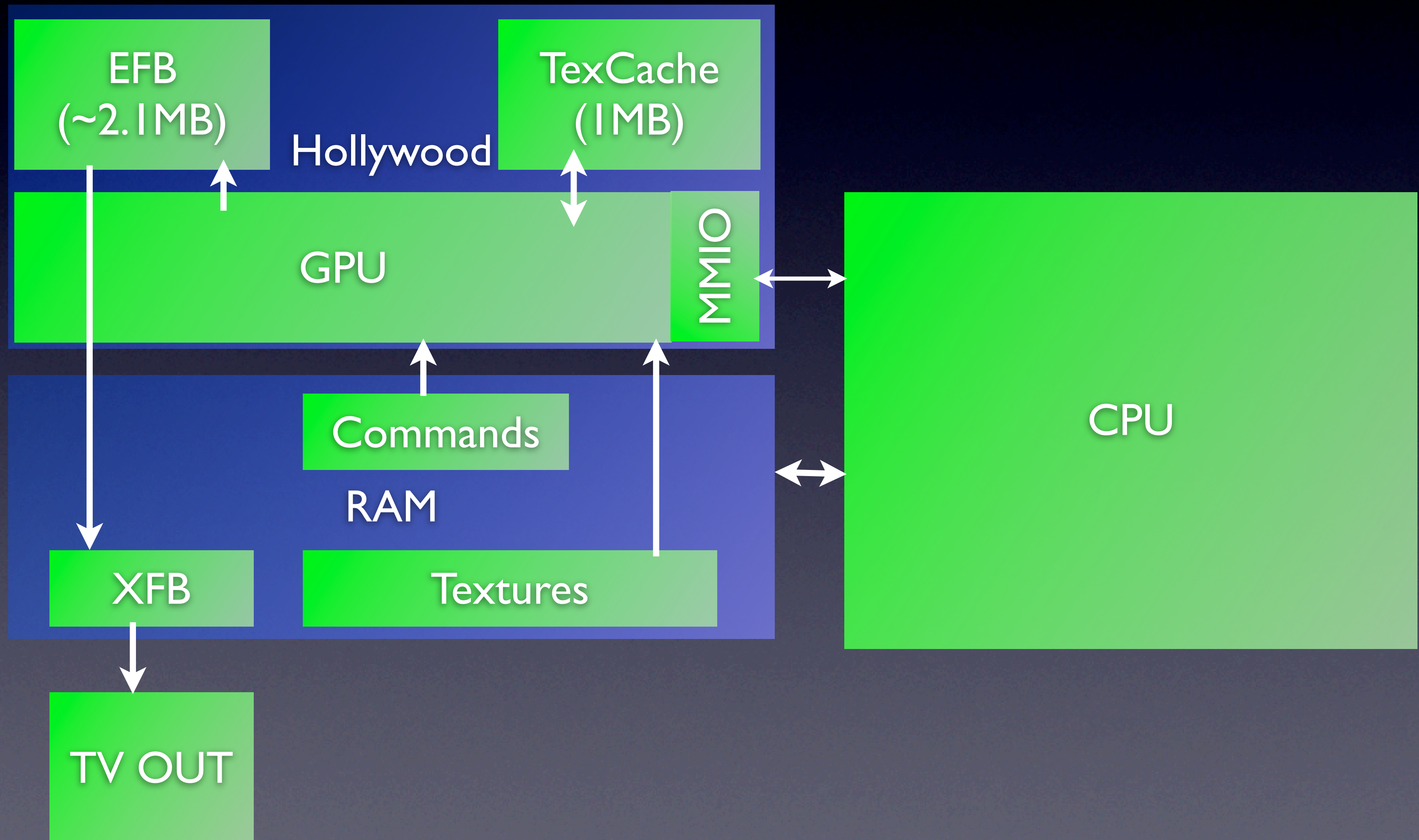
How to write Code?

- Using the official SDK?
  - Illegal!
  - Fully functional.
  - ...but soooo boring.

Free Software FTW!

- libOGC
  - open-source library
  - originally written for the gamecube
    - (but also supporting Wii-specific features)
  - comes together with a precompiled cross-toolchain...
  - ... and sample code, like that spinning cube.

The Wii Architecture



Hello, Cube!

- Rendering first happen into the internal GPU RAM (eFB)
- Data then gets copied into main RAM (XFB), then output to the TV
- For details, look at the "acube" libogc demo.

- http://sourceforge.net/projects/devkitpro/

Xbox 360

- A beast:
  - 3 SMT-cores with 3.2 GHz each
  - 512 MB RAM
  - GPU:
    - Superset of Shader Model 3.0
    - Very strong fill rate, thanks to embedded RAM (again)

So, let’s hack it, and have some fun.

- Quite advanced security: essentially forbits that any user (=game) software can write executable code into memory
- Microsoft - yes - really learned their lessons from original Xbox.
- Buffer overflows in a game doesn’t help at all (keywords: encrypted memory, NX, Hypervisor)
- To make a long story short: Yes, there is a hack.

“King Kong Shader Hack”

- You need:
  - An Xbox 360 with a specific kernel version,
  - a modified DVD-ROM Firmware,
  - a modified game.

“King Kong Shader Hack”

- You gain:
  - Arbitrary code execution in hypervisor context (something that’s not possible with an official SDK)
  - Fully functional Linux, if you want
  - Full hardware access

(DEMO)

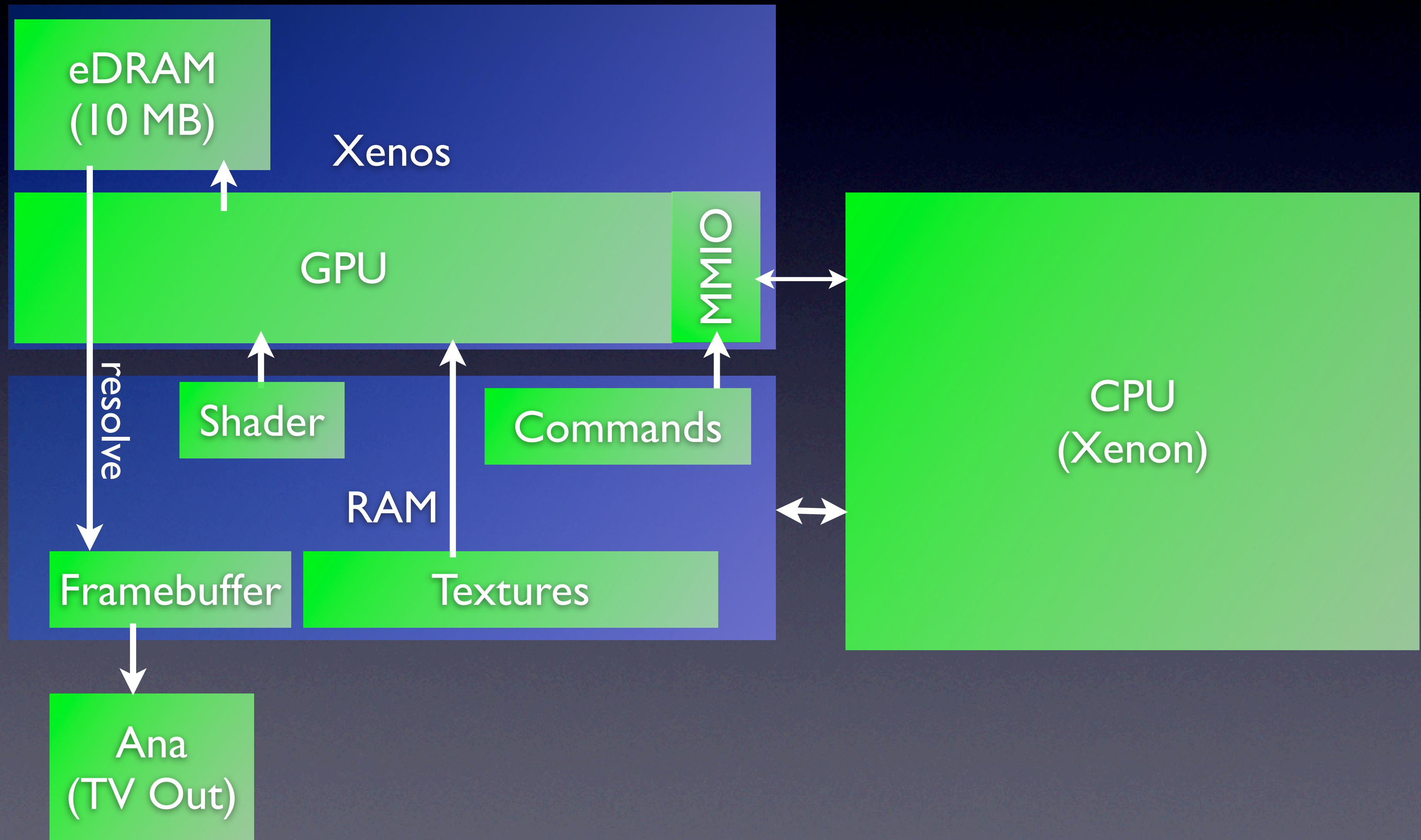
Hello, world!

- Let’s write a winnerdemo!
- Let’s display a spinning cube!

Hello, Cube!

- We have a GPU driver based on reverse-engineering.
- GPU doesn't have fixed-function pipeline.
- We need to compile shaders.
- Microsoft offers XNA ("Use C# code to write games"), which includes a shader compiler we can use.

Xbox 360 GPU architecture



Xenos Features

- eDRAM (10MB) is “intelligent” RAM which handles Blending, Z-Compare, AA and Format-Conversion
  - Huge bandwidth
  - No textures etc.
  - Rendering happens directly into eDRAM
  - “Resolve”-Operation copies into FB

Xenos Features

- SM 3.0+
- Memory Export: Shaders can write to memory (multipass-geometry)
- 32-bit float textures, 16-bit float framebuffer (per channel, of course), if you want

Hello, Cube!

- Init Hardware (Map Memory, upload Microcodes)
- Load Shader
- For each frame:
  - Setup Material (Renderstates, Textures, Shaders)
  - Draw Primitive
  - “Resolve” eDRAM into Framebuffer

Our Xe_-API

- GPU was developed with Direct3D in mind.
- Hardware more or less matches what D3D exposes to the user (most enum even match exactly!)
- Direct3D-like API, just a bit more low-level.
- No unneccessary indirections / wrappers.
- So things get very very complicated, right?
- I’ll try it anyway.

Initialize Hardware

```
struct XenosDevice _xe, *xe;

  xe = &_xe;
    /* initialize the GPU */
  Xe_Init(xe);
```

- Library will memory map MMIO registers and physical memory to use
- Hardware register will be re-initialized (they were left in an uncertain state after the KK-exploit)

Define Render Target

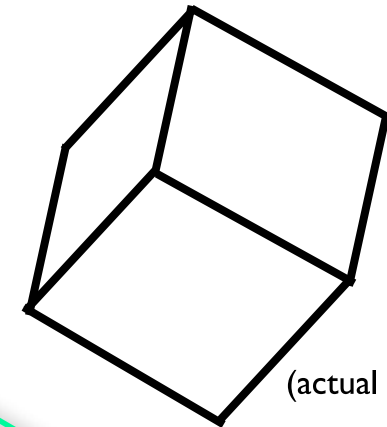
```
/* create a render target (the framebuffer) */
struct XenosSurface *fb = Xe_GetFramebufferSurface(xe);
Xe_SetRenderTarget(xe, fb);
```

- Basically just some pointers are set.
- Viewport Scaling is set.

- Now, let's display something!

```
    /* let's define a vertex buffer format */
static const struct XenosVBFFormat vbf =
{
   5, {
       {XE_USAGE_POSITION, 0, XE_TYPE_FLOAT3},
       {XE_USAGE_NORMAL,   0, XE_TYPE_FLOAT3},
       {XE_USAGE_TANGENT,  0, XE_TYPE_FLOAT3},
       {XE_USAGE_COLOR,    0, XE_TYPE_UBYTE4},
       {XE_USAGE_TEXCOORD, 0, XE_TYPE_FLOAT2}
   }
};
```

Our Cube!




```
    /* a cube */
float cube[] = {
    //       POSITION               |            NORMAL             |           TANGENT             |   COL   |    U        V    |
    -0.5000 , -0.5000 , -0.5000 , +0.0000 , +0.0000 , -1.0000 , +1.0000 , +0.0000 , +0.0000 , +0.0000 , +1.0000 , +1.0000,
    -0.5000 , +0.5000 , -0.5000 , +0.0000 , +0.0000 , -1.0000 , +1.0000 , +0.0000 , +0.0000 , +0.0000 , +1.0000 , +0.0000,
    +0.5000 , +0.5000 , -0.5000 , +0.0000 , +0.0000 , -1.0000 , +1.0000 , +0.0000 , +0.0000 , +0.0000 , +2.0000 , +0.0000,
    +0.5000 , -0.5000 , -0.5000 , +0.0000 , +0.0000 , -1.0000 , +1.0000 , +0.0000 , +0.0000 , +0.0000 , +2.0000 , +1.0000,
                        ...
    +0.5000 , -0.5000 , +0.5000 , +0.0000 , -1.0000 , +0.0000 , -1.0000 , +0.0000 , +0.0000 , +0.0000 , +0.0000 , +0.0000,
    -0.5000 , -0.5000 , +0.5000 , +0.0000 , -1.0000 , +0.0000 , -1.0000 , +0.0000 , +0.0000 , +0.0000 , +1.0000 , +0.0000,
    -0.5000 , -0.5000 , -0.5000 , +0.0000 , -1.0000 , +0.0000 , -1.0000 , +0.0000 , +0.0000 , +0.0000 , +1.0000 , +1.0000,
};
```

```
unsigned short cube_indices[] = { 0, 1, 2, 0, 2, 3, 4, 5, 6,
4, 6, 7, 8, 9, 10, 8, 10, 11, 12, 13, 14, 12, 14, 15, 16, 17,
18, 16, 18, 19, 20, 21, 22, 20, 22, 23};
```

Put Geometry into Physical Memory

```
  /* create and fill vertex buffer */
struct XenosVertexBuffer *vb = Xe_CreateVertexBuffer(xe,
   sizeof(cube));
void *v = Xe_VB_Lock(xe, vb, 0, sizeof(cube), 0);
memcpy(v, cube, sizeof(cube));
Xe_VB_Unlock(xe, vb);

  /* create and fill index buffer */
struct XenosIndexBuffer *ib = Xe_CreateIndexBuffer(xe,
   sizeof(cube_indices), XE_FMT_INDEX16);
unsigned short *i = Xe_IB_Lock(xe, ib, 0, sizeof(cube_indices), 0);
memcpy(i, cube_indices, sizeof(cube_indices));
Xe_IB_Unlock(xe, ib);
```

Load Shaders

```
  /* load pixel shader */
struct XenosShader *sh_ps, *sh_vs;
sh_ps = Xe_LoadShader(xe, "ps.psu");
Xe_InstantiateShader(xe, sh_ps, 0);

  /* load vertex shader */
sh_vs = Xe_LoadShader(xe, "vs.vsu");
Xe_InstantiateShader(xe, sh_vs, 0);
Xe_ShaderApplyVFetchPatches(xe, sh_vs, 0, &vbf);
```

Refers to the used Vertex Buffer Format

Vertex Shader

```
float4x4 modelView: register (c0);
float4x3 modelWorld: register (c4);

struct Input
{
   float4 vPos: POSITION;
   float3 vNormal: NORMAL;
   float4 vUV: TEXCOORD0;
};

struct Output
{
   float4 oPos: POSITION;
   float3 oNormal: NORMAL;
   float4 oUV: TEXCOORD0;
}
```

```
Output main(Input input)
{
   Output output;
   output.oPos =
     mul(transpose(modelView), input.vPos);
   output.oNormal =
     mul(transpose(modelWorld), input.vNormal);
   output.oUV = input.vUV;
   return output;
}
```

Transform Position and Normal,
Passthru UV

Pixel Shader

```
float4 lightDirection: register(c0);

struct Input
{
   float3 oNormal: NORMAL;
   float4 oUV: TEXCOORD0;
};

sampler s;

float4 main(Input input): COLOR
{
   float4 tex = tex2D(s, input.oUV);
   return dot(input.oNormal, lightDirection) * tex;
}
```

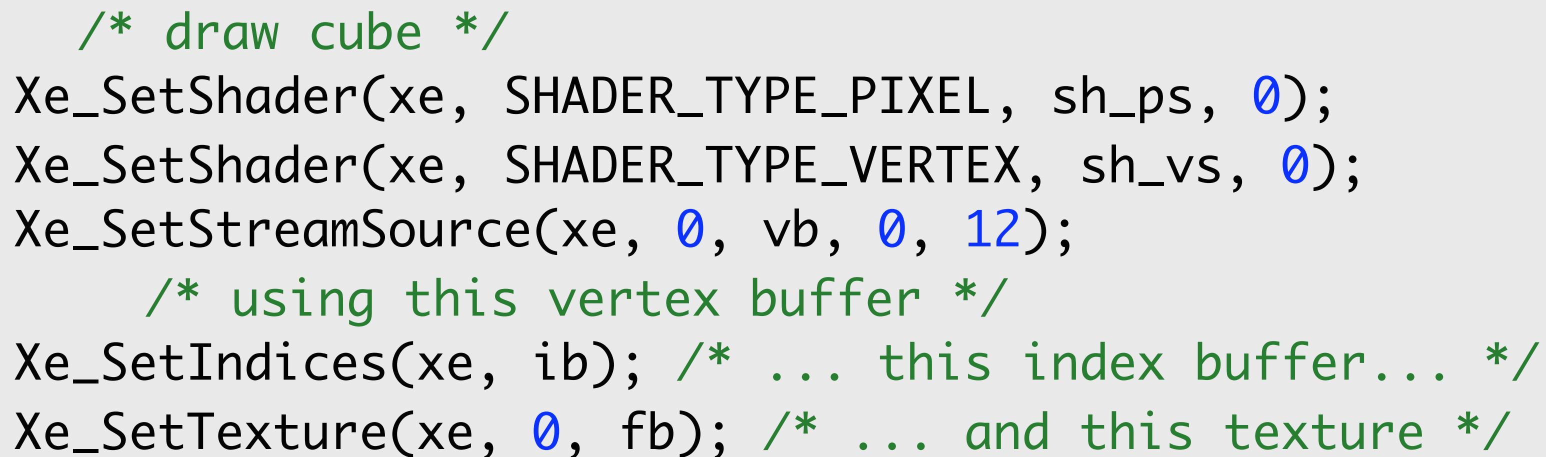
Multiply directional light with texture

Render a Frame!

```
/* begin a new frame, i.e. reset all
   renderstates to the default */
Xe_InvalidateState(xe);
```

```
/* load some model-view matrix */
glLoadIdentity();
glPushMatrix();
glTranslate(0, 0, -3);
glRotate(f / 100.0, .5, .1, 1);
M_LoadMV(xe, 0); // load model view matrix to VS constant 0
M_LoadMW(xe, 4); // load (fake) model world matrix to VS constant 4
```

```
/* set the light direction for the pixel shader */
float lightDirection[] = {0, 0, -1, 0};
Xe_SetPixelShaderConstantF(xe, 0, lightDirection, 1);
```



That's the last framebuffer...

Why not.

```
int max_vertices = sizeof(cube)/(sizeof(*cube)*12);
int nr_primitives = sizeof(cube_indices)/sizeof(*cube_indices) / 3;
Xe_DrawIndexedPrimitive(xe, XE_PRIMTYPE_TRIANGLELIST, 0, 0,
    max_vertices, 0, nr_primitives);
```

Resolve into Rendertarget

```
/* clear to white */
Xe_SetClearColor(xe, ~0);

/* resolve (and clear) */
Xe_Resolve(xe);

/* wait for render finish */
Xe_Sync(xe);
```

(Demo)

Thank you!

- Questions??
- http://debugmo.de/ (will also contain these slides, the Xbox GPU library and other more or less interesting things)
- http://free60.org/ for Xbox360-related stuff
- http://www.wiibrew.org/ for Wii stuff

Felix "tmbinc" Domke - Breakpoint 2008

(and thanks to **bushing** for doing the video for me)