Shaders in Eve Online

Páll Ragnar Pálsson
EVE Online
EVERYONE

Y U NO PLAY EVE!?
• Trinity
  • First released 2003
  • Proprietary graphics engine
  • DirectX 9 (DX11 on its way)
  • Shader Model 3 (4 & 5 in development)
  • HLSL
<table>
<thead>
<tr>
<th>Granny Real32 Position</th>
<th>Granny Uint8 BoneWeights</th>
<th>Granny Uint8 BoneIndices</th>
<th>Granny Real16 Normal</th>
<th>Granny Real16 Tangent</th>
<th>Granny Real16 Binormal</th>
<th>Granny Real16 TextureCoordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-26.500799 85.897544 -21.859985</td>
<td>0 0 0 0</td>
<td>-0.322265 0.751953 -0.574707 0</td>
<td>-0.056732 0.590820 0.804687 0</td>
<td>0.944824 0.291992 -0.147705 0</td>
<td>0.246093 0.079345</td>
</tr>
<tr>
<td>1</td>
<td>-28.500797 87.186868 -15.219329</td>
<td>0 0 0 0</td>
<td>-0.390869 0.619140 -0.681152 0</td>
<td>-0.225463 0.652832 0.723144 0</td>
<td>0.892578 0.436279 -0.115722 0</td>
<td>0.258544 0.079711</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Granny Real32 Position</th>
<th>Granny Uint8 BoneWeights</th>
<th>Granny Uint8 BoneIndices</th>
<th>Granny Real16 Normal</th>
<th>Granny Real16 Tangent</th>
<th>Granny Real16 Binormal</th>
<th>Granny Real16 TextureCoordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>9692</td>
<td>-49.182678 70.347160 -62.935657</td>
<td>0 0 0 0</td>
<td>-0.805684 0.496337 0.323242 0</td>
<td>0.211914 0.751464 -0.625000 0</td>
<td>-0.553222 -0.435058 -0.710449 0</td>
<td>0.422119 0.091247</td>
</tr>
<tr>
<td>9693</td>
<td>-41.317810 77.452629 -66.020416</td>
<td>0 0 0 0</td>
<td>-0.293212 0.897949 0.328369 0</td>
<td>0.101501 0.370849 -0.923339 0</td>
<td>-0.950683 -0.237426 -0.199951 0</td>
<td>0.435058 0.104003</td>
</tr>
</tbody>
</table>
Into this
HL SL code structure

• Matrices, Variables and Structures
• Vertex Shaders
• Pixel shaders
• Techniques
• Functions
• Matrices, Variables and Structures
• Vertex Shaders
• Pixel shaders
• Techniques
• Functions
Matrices, Variables and Structures

//<variable name> : SEMANTIC FROM APP

// world space
float4x4 wMatrix : World;

// World View Projection
float4x4 wvpMatrix : WorldViewProjection;

// contains camera position
float4x4 vitMatrix : ViewInverseTranspose;
// Vertex shader input

struct SimpleAppVtx
{
    float3 pos : POSITION;
    float3 normal : NORMAL;
    float3 tangent : TANGENT;
    float3 bitangent : BINORMAL;
    float2 texcoord : TEXCOORD;
};
// Vertex shader output / Pixel shader input

struct SimpleShaderVtx
{
    float4 posClip : POSITION;
    float2 texcoord0 : TEXCOORD0;
};
struct StandardShaderVtx
{
    float4 posClip : POSITION;
    float4 texcoord01 : TEXCOORD0;
    float3 normalWorld : TEXCOORD1;
    float3 tangentWorld : TEXCOORD2;
    float3 bitangentWorld : TEXCOORD3;
    float4 eyeDirWorld : TEXCOORD4; /// 4th component is the fogcoefficient (0 = full fog, 1 = no fog)
    float3 posWorld : TEXCOORD5;
#if SHADOW_ENABLED
    float4 posShadowView : TEXCOORD7;
    float4 posShadowViewProjection : TEXCOORD8;
#endif
};
struct BaseLightingParams
{
    float3 normalWorld;
    float diffuse;
    float specularMaskedBrightness;
    float specularDot;
    float3 pointLights;
    float3 reflectDir;
};
Matrices, Variables and Structures

```csharp
float4 ReflectionFactors
{
    string Group = "Material";
    bool SasUiVisible = true;
    string UIWidget = "VectorMixed";
    string Component1 = "Add";
    string Component2 = "Multiply";
    string Component3 = "Reflection Saturation";
    string Component4 = "Strength in shadow";
    = { 1.0, 1.0, 0.0, 1.0 };
}
```
HLSL code structure

- Matrices, Variables and Structures
- **Vertex Shaders**
- Pixel shaders
- Functions
- Techniques
StandardShaderVtx StandardVS( StandardAppVtx inVtx, float4x4 world )
{
    StandardShaderVtx outVtx = (StandardShaderVtx)0;

    // position
    float4 posWorld = mul( float4( inVtx.pos, 1.0f ), world );
    outVtx.posClip = mul( posWorld, PerFrameVS.ViewProjectionMat );
    // texcoord
    outVtx.texcoord01 = inVtx.tex.xyyy;
    float4x4 worldInverseTranspose = PerObjectVS.WorldInverseTransposeMat;
    outVtx.normalWorld = mul( inVtx.normal, worldInverseTranspose );
    outVtx.tangentWorld = mul( inVtx.tangent, worldInverseTranspose );
    outVtx.bitangentWorld = mul( inVtx.bitangent, worldInverseTranspose );
    float3 eyePosWorld = ExtractEyePosFromViewMatrix( PerFrameVS.ViewInverseTransposeMat );
    float3 pos2eye = eyePosWorld - posWorld.xyz;
    float4 eyeDirWorld;
    outVtx.eyeDirWorld = float4( normalize( pos2eye ), 0.0f );
    outVtx.posWorld = posWorld;
    ProjectPositionToShadow( posWorld, outVtx.posShadowView, outVtx.posShadowViewProjection );
    return outVtx;
}
HLSL code structure

• Matrices, Variables and Structures
• Vertex Shaders
• **Pixel shaders**
• Functions
• Techniques
float4 SinglePS( StandardShaderVtx inVtx ) : COLOR
{
    BaseLightingParams params = CalcBaseLightingParams( inVtx );
    float4 diffuseMap = tex2D( DiffuseMapSampler, inVtx.texcoord01.xy );
    return ColorWithFog( inVtx, CalcBaseLighting( params, diffuseMap, MaterialDiffuseColor, MaterialAmbientFactor, MaterialSpecularFactors, MaterialSpecularCurve ) );
}
HLSL code structure

• Matrices, Variables and Structures
• Vertex Shaders
• Pixel shaders
• Functions
• Techniques
BaseLightingParams CalcBaseLightingParams( StandardShaderVtx inVtx )
{
    BaseLightingParams ret;
    ret.normalWorld = CalcNormalFromMap( inVtx );
    ret.pointLights = CalcPointLightsColor( inVtx.posWorld, inVtx.normalWorld );
    float2 brightness = CalcBrightnessFromShadowMap( inVtx );
    // diffuse
    ret.diffuse = saturate( 2.0f * dot( ret.normalWorld, PerFramePS.Sun.DirWorld ) - 0.2f );
    ret.diffuse *= brightness.x;
    // specular
    float3 halfDirWorld = normalize( PerFramePS.Sun.DirWorld + inVtx.eyeDirWorld.xyz );
    ret.specularDot = saturate( dot( ret.normalWorld, halfDirWorld ) );
    float specularMask = GetSpecularStrengthFromMap( inVtx.texcoord01.xy );
    ret.specularMaskedBrightness = specularMask * brightness.x;
    // reflection (Need to negate the reflect function since it is implemented all backwards!)
    float3 reflectDirWorld = -reflect( inVtx.eyeDirWorld, ret.normalWorld );
    ret.reflectDir = mul( float4( reflectDirWorld, 1.0 ), ReflectionMapTransform );
    return ret;
}
HLSL code structure

- Matrices, Variables and Structures
- Vertex Shaders
- Pixel shaders
- Functions
- Techniques
technique Main
{
    pass P0
    {
        // We can put device state changes here,
        // But we try not to do them per-shader
        // Explained later in the talk.

        VertexShader = COMPILE_VS( StandardStaticVS );
        PixelShader  = COMPILE_PS( SinglePS );
    }
    pass P1
    ...
}

Textures
Specular map
Normal map
Object normals, tangents and bitangents
Wire frame
Shadow map
Shadows – shadow maps

- For each object receiving shadows
  - Create a frustum from the bounding info of the object
  - Use this frustum to gather list of objects casting shadows on this object
  - Render the shadow casters using the frustum
    - In Eve the sun is always assumed to be far away so we use orthogonal projection matrix
    - We override the pixel shader, but keep the vertex shader intact
  - Only store the depth, discard the color info
  - End up with one shadow map per receiver
During rendering
  - We project the pixel position into the light’s coordinate space using the light’s view and projection transforms
  - The depth in the camera’s frustum is the z value of the projected position
  - The x and y values are used to calculate the UVs needed to look up into the shadow map rendered earlier.
  - If the projected value is closer to the light than the value read from the texture the pixel in question is not behind a shadow caster, and is fully lit.
Trinity graphics engine

• Batches
  – Sending triangles to the GPU is fairly slow
  – A batch is a group of triangles that you can render with one draw call
  – We want to have few draw calls
  – We want to have few device state changes

• Trinity uses “areas” for this purpose
  – Each object in the scene has a list of different area lists
  – The most commonly used are opaque, decal, transparent and additive
Trinity graphics engine

• Areas
  – During rendering we gather the batches from each area type and render together
  – We only change render states between batches
  – Opaque batches are not sorted
    • Should they be?
  – Transparent batches are sorted and rendered back to front
  – Additive areas are not sorted
Not just ships
Not Just ships
Not just ships
Not just ships
But sometimes A LOT OF SHIPS
But sometimes A LOT OF SHIPS

RENDER
ALL THE SHIPS!
Depth effects – God rays
God rays

- Render depth to a texture
- Render a quad centered on the sun

- In the pixel shader:
  - Use VPOS to get screen UV
  - Walk form this screen position to the center of the quad
  - For each step, sample the depth map, with random jitter and compare the value to a threshold
  - Accumulate all “Hits” and find the occlusion ratio
  - Subtract the occlusion from the texture of the quad (a faint glow)
for (int i = 0; i < numSamples; ++i)
{
    float ratio = (float)i / (float)numSamples;
    float2 uv = lerp( screenTextureUV, spriteCenterScreen, ratio + noise);
    depth = GetDepthFromMirroredTexture( uv );
    occlusion += step(thresholdDepth, depth);
}
texColor = saturate( texColor - float4(occlusion.xxx, 0.0 ) );
Physical not always best

• We make games, not simulations
• Art director has the final say
• Artists want control
• If a cheap hack delivers almost the same results, then by all means hack away.
• **General Performance Tips**
  – Clear only when you must.
  – Minimize state changes and group the remaining state changes.
  – Use smaller textures, if you can do so.
  – Draw objects in your scene from front to back.
  – Use triangle strips instead of lists and fans. For optimal vertex cache performance, arrange strips to reuse triangle vertices sooner, rather than later.
  – Gracefully degrade special effects that require a disproportionate share of system resources.
Best practices

• General Performance Tips
  – Constantly test your application's performance.
  – Minimize vertex buffer switches.
  – Use static vertex buffers where possible.
  – Choose texture formats wisely.
  – Draw using indexed primitives. This can allow for more efficient vertex caching within hardware.
  – If the depth buffer format contains a stencil channel, always clear the depth and stencil channels at the same time.