AMD SHADOWFX

ALEX KHARLAMOV

TABLE OF CONTENTS

- ShadowFX Techniques
- ShadowFX Options
- Integration guide

AMD

SHADOWFX TECHNIQUES

TRIVIA AND ALGORITHMS DESCRIPTION

SHADOWFX TECHNIQUES

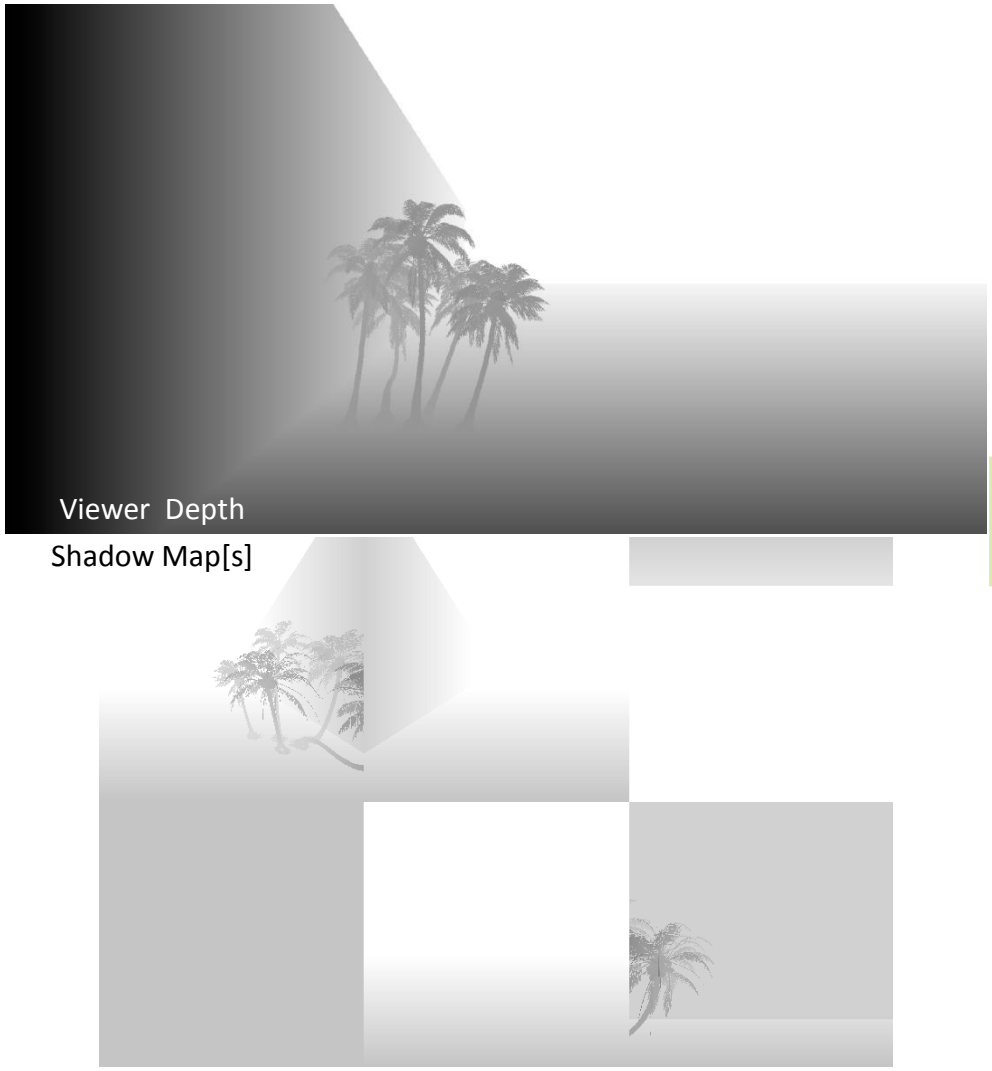
AMD

- ShadowFX implements highly optimized GCN shadow filtering
  - A full screen pass that returns a deferred shadow mask
- Currently ShadowFX supports the following filtering kernels:
  - Uniform Smooth Shadows (http://developer.amd.com/wordpress/media/2012/10/Isidoro-ShadowMapping.pdf)
  - Contact Hardening Shadows (http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/TakingAdvantageofDirect3D10.pps)
  - Shader permutations for different filter sizes: 7x7, 9x9, 11x11, 13x13, 15x15

- ShadowFX sample can also serve as a sandbox for shadow filtering technique development and comparison

SHADOWFX TECHNIQUES

SHADOWFX IMPLEMENTS A DEFERRED SHADOW MASK FILTERING



Full screen pass

AMD

SHADOWFX OPTIONS

SHADOWFX OPTIONS

SHADER PERMUTATIONS

- Can process up to 6 shadow maps per full screen pass
- Shadow maps can be arranged in
  - Texture Atlas
    - No UV clamping is done to avoid filtering across different atlas regions
  - Texture Array
- Supports different shadow casting types:
  - Union – a set of shadow casters which contribute to the same deferred shadow mask
    - Filtering shader will process all active lights (and potentially execute filtering up to 6 times)
    - Filtered results are blended with a MIN function
  - Cascade – a set of shadow maps that form different LOD levels of a cascaded shadow map
    - Filtering shader will early out after it finds the nearest cascade that contains the filtered point
  - Cube – a set of shadow maps that form different faces of a cube shadow map
    - Filtering shader will calculate the exact shadow map face and only execute filtering once

SHADOWFX OPTIONS

AMD

- The world space position of a shadow'ed pixel can be used for light transformation directly or:
  - Displaced by a normal fetched form Gbuffer
  - Displaced by a normal calculated from depth buffer
- Shadow filtering texture fetch instructions:
  - Gather4
    - AMD preferred option that results in better performance
  - PCF
- Application can setup up stencil buffer and stencil test
  - Stencil reduces full screen pass to only execute in areas that overlap with shadow caster volume
- Most of the options are orthogonal to each other
  - A high number of shader permutations

AMD

SHADOWFX TECHNIQUE

INTEGRATION GUIDE

INTEGRATION

MINIMUM AMOUNT OF CODE FOR INITIAL INTEGRATION

- Declare, initialize

```
AMD::ShadowFX_Desc g_ShadowsDesc;
g_ShadowsDesc.m_pDevice = pd3dDevice;
AMD::ShadowFX_Initialize(g_ShadowsDesc);
```

- Setup viewer and light cameras, d3d11 resources

- Release

```
AMD::ShadowFX_Release(g_ShadowsDesc);
```

```
g_ShadowsDesc.m_Execution = AMD::SHADOWFX_EXECUTION_UNION;
g_ShadowsDesc.m_NormalOption = AMD::SHADOWFX_NORMAL_OPTION_NONE;
g_ShadowsDesc.m_TapType = AMD::SHADOWFX_TAP_TYPE_FIXED;
g_ShadowsDesc.m_Filtering = AMD::SHADOWFX_FILTERING_CONTACT;
g_ShadowsDesc.m_TextureFetch = AMD::SHADOWFX_TEXTURE_FETCH_GATHER4;
g_ShadowsDesc.m_FilterSize = AMD::SHADOWFX_FILTER_SIZE_7;
g_ShadowsDesc.m_TextureType = AMD::SHADOWFX_TEXTURE_2D_ARRAY;

float2 backbufferDim((float)g_Width, (float)g_Height);

memcpy(&g_ShadowsDesc.m_Viewer.m_View, &g_ViewerData.m_View, sizeof(g_ShadowsDesc.m_Viewer.m_View));
memcpy(&g_ShadowsDesc.m_Viewer.m_Projection, &g_ViewerData.m_Projection, sizeof(g_ShadowsDesc.m_Viewer.m_Projection));
memcpy(&g_ShadowsDesc.m_Viewer.m_ViewProjection, &g_ViewerData.m_ViewProjection, sizeof(g_ShadowsDesc.m_Viewer.m_ViewProjection));
memcpy(&g_ShadowsDesc.m_Viewer.m_View_Inv, &g_ViewerData.m_ViewInv, sizeof(g_ShadowsDesc.m_Viewer.m_View_Inv));
memcpy(&g_ShadowsDesc.m_Viewer.m_Projection_Inv, &g_ViewerData.m_ProjectionInv, sizeof(g_ShadowsDesc.m_Viewer.m_Projection_Inv));
memcpy(&g_ShadowsDesc.m_Viewer.m_ViewProjection_Inv, &g_ViewerData.m_ViewProjectionInv, sizeof(g_ShadowsDesc.m_Viewer.m_ViewProjection_Inv));
memcpy(&g_ShadowsDesc.m_Viewer.m_Position, &g_ViewerData.m_Position, sizeof(g_ShadowsDesc.m_Viewer.m_Position));
memcpy(&g_ShadowsDesc.m_Viewer.m_Direction, &g_ViewerData.m_Direction, sizeof(g_ShadowsDesc.m_Viewer.m_Direction));
memcpy(&g_ShadowsDesc.m_Viewer.m_Up, &g_ViewerData.m_Up, sizeof(g_ShadowsDesc.m_Viewer.m_Up));
memcpy(&g_ShadowsDesc.m_Viewer.m_Color, &g_ViewerData.m_Color, sizeof(g_ShadowsDesc.m_Viewer.m_Color));
memcpy(&g_ShadowsDesc.m_DepthSize, &backbufferDim, sizeof(g_ShadowsDesc.m_DepthSize));

g_ShadowsDesc.m_ActiveLightCount = CUBE_FACE_COUNT;
for (int i = 0; i < CUBE_FACE_COUNT; i++)
{
    float2 shadowDim(g_ShadowMapSize, g_ShadowMapSize);
    float4 shadowRegion(0.0f, 0.0f, 0.0f, 0.0f);
    if (g_ShadowsDesc.m_TextureType == AMD::SHADOWFX_TEXTURE_2D_ARRAY)
    {
        shadowRegion.x = 0.0f;
        shadowRegion.z = 1.0f;
        shadowRegion.y = 0.0f;
        shadowRegion.w = 1.0f;
        g_ShadowsDesc.m_ArraySlice[i] = i;
    }

    memcpy(&g_ShadowsDesc.m_ShadowSize[i], &shadowDim, sizeof(g_ShadowsDesc.m_ShadowSize[i]));
    memcpy(&g_ShadowsDesc.m_ShadowRegion[i], &shadowRegion, sizeof(g_ShadowsDesc.m_ShadowRegion[i]));
    memcpy(&g_ShadowsDesc.m_Light[i].m_View, &g_LightData[i].m_View, sizeof(g_ShadowsDesc.m_Light[i].m_View));
    memcpy(&g_ShadowsDesc.m_Light[i].m_Projection, &g_LightData[i].m_Projection, sizeof(g_ShadowsDesc.m_Light[i].m_Projection));
    memcpy(&g_ShadowsDesc.m_Light[i].m_ViewProjection, &g_LightData[i].m_ViewProjection, sizeof(g_ShadowsDesc.m_Light[i].m_ViewProjection));
    memcpy(&g_ShadowsDesc.m_Light[i].m_View_Inv, &g_LightData[i].m_ViewInv, sizeof(g_ShadowsDesc.m_Light[i].m_View_Inv));
    memcpy(&g_ShadowsDesc.m_Light[i].m_Projection_Inv, &g_LightData[i].m_ProjectionInv, sizeof(g_ShadowsDesc.m_Light[i].m_Projection_Inv));
    memcpy(&g_ShadowsDesc.m_Light[i].m_ViewProjection_Inv, &g_LightData[i].m_ViewProjectionInv, sizeof(g_ShadowsDesc.m_Light[i].m_ViewProjection_Inv));
    memcpy(&g_ShadowsDesc.m_Light[i].m_Position, &g_LightData[i].m_Position, sizeof(g_ShadowsDesc.m_Light[i].m_Position));
    memcpy(&g_ShadowsDesc.m_Light[i].m_Up, &g_LightData[i].m_Up, sizeof(g_ShadowsDesc.m_Light[i].m_Up));
    memcpy(&g_ShadowsDesc.m_Light[i].m_Direction, &g_LightData[i].m_Direction, sizeof(g_ShadowsDesc.m_Light[i].m_Direction));
    g_ShadowsDesc.m_Light[i].m_Aspect = g_LightCamera.GetAspect();
    g_ShadowsDesc.m_Light[i].m_Fov = g_LightCamera.GetFOV();
    g_ShadowsDesc.m_Light[i].m_FarPlane = g_LightCamera.GetFarClip();
    g_ShadowsDesc.m_Light[i].m_NearPlane = g_LightCamera.GetNearClip();
    g_ShadowsDesc.m_SunArea[i] = g_SunSize * g_SunSize; // for the filtering we actually need squared size
    g_ShadowsDesc.m_DepthTestOffset[i] = g_DepthTestOffset;
    g_ShadowsDesc.m_NormalOffsetScale[i] = g_NormalOffsetScale;
}

g_ShadowsDesc.m_pContext = pd3dContext;
g_ShadowsDesc.m_pDevice = pd3dDevice;
g_ShadowsDesc.m_pDepthSRV = g_AppDepth._srv;
g_ShadowsDesc.m_pNormalSRV = g_AppNormal._srv;
g_ShadowsDesc.m_pOutputRTV = g_ShadowMask._rtv;
g_ShadowsDesc.m_pShadowSRV = g_ShadowMap._srv;
g_ShadowsDesc.m_OutputChannels = 1;
g_ShadowsDesc.m_ReferenceDSS = 0;
g_ShadowsDesc.m_pOutputDSS = NULL;
g_ShadowsDesc.m_pOutputDSV = NULL;

AMD::ShadowFX_Render(g_ShadowsDesc);
```

DISCLAIMER & ATTRIBUTION

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION