Deferred Lighting / Shadows / Materials

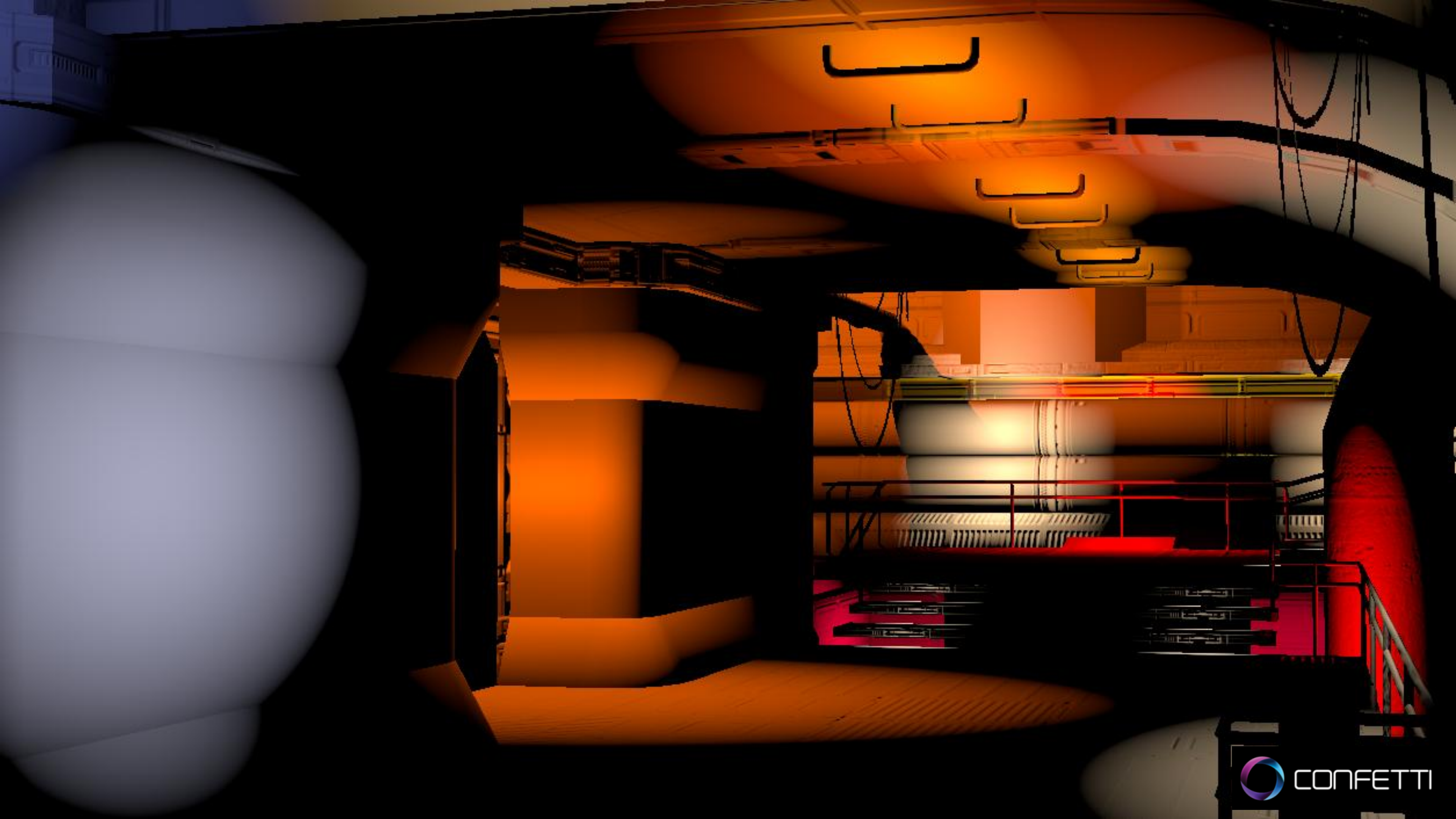
FMX May 6th, 2011

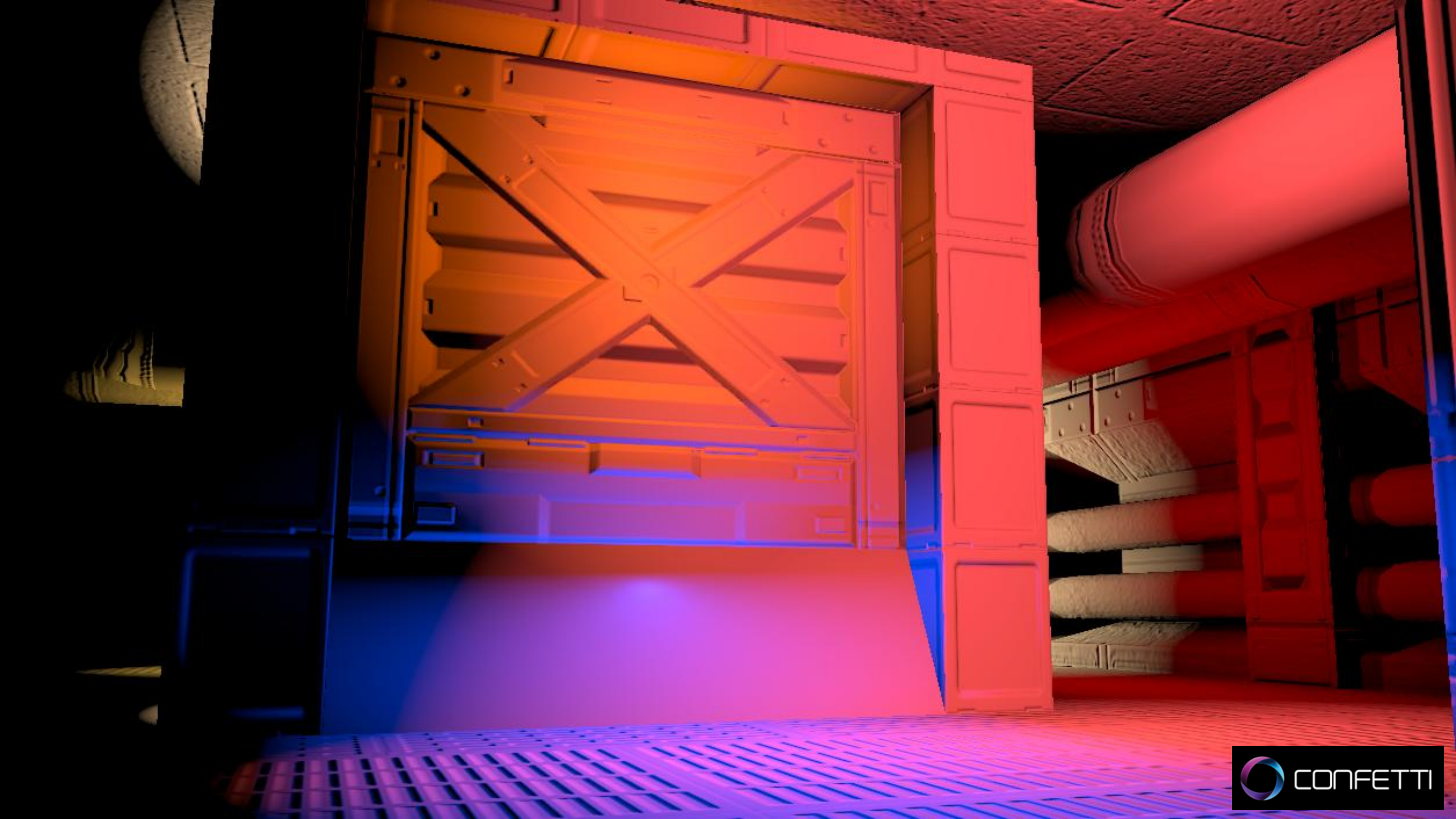
by Wolfgang Engel

CONFETTI

RawK®


CONFETTI


CONFETTI


CONFETTI


CONFETTI


CONFETTI

Agenda

- Deferred Lighting
- Ellipsoidal Light Shadows
- Screen-Space Material System

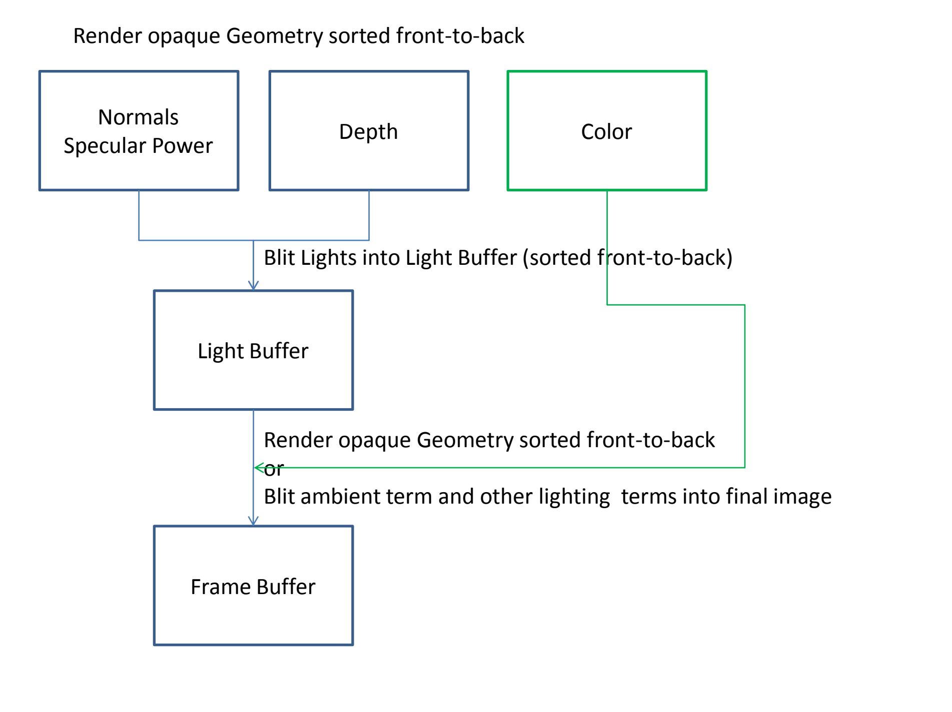
Agenda

- … all this is rendered real-time on an integrated graphics chip
  < 33ms per frame ☺
- RawK shows a setup with lots of lights and shadows, it mimics typical Movie lighting and shadowing patterns
- Nothing is pre-calculated … to
  - avoid streaming
  - be destructible

Deferred Lighting

- Instead of rendering lights while rendering an object, we create a buffer that holds all data of the geometry in screen-space; so called (G)eometry-Buffer
- The G-Buffer is then used to light the scene as many times as necessary -> e.g. apply several hundred or thousands of lights
  -> detaches lighting from scene complexity

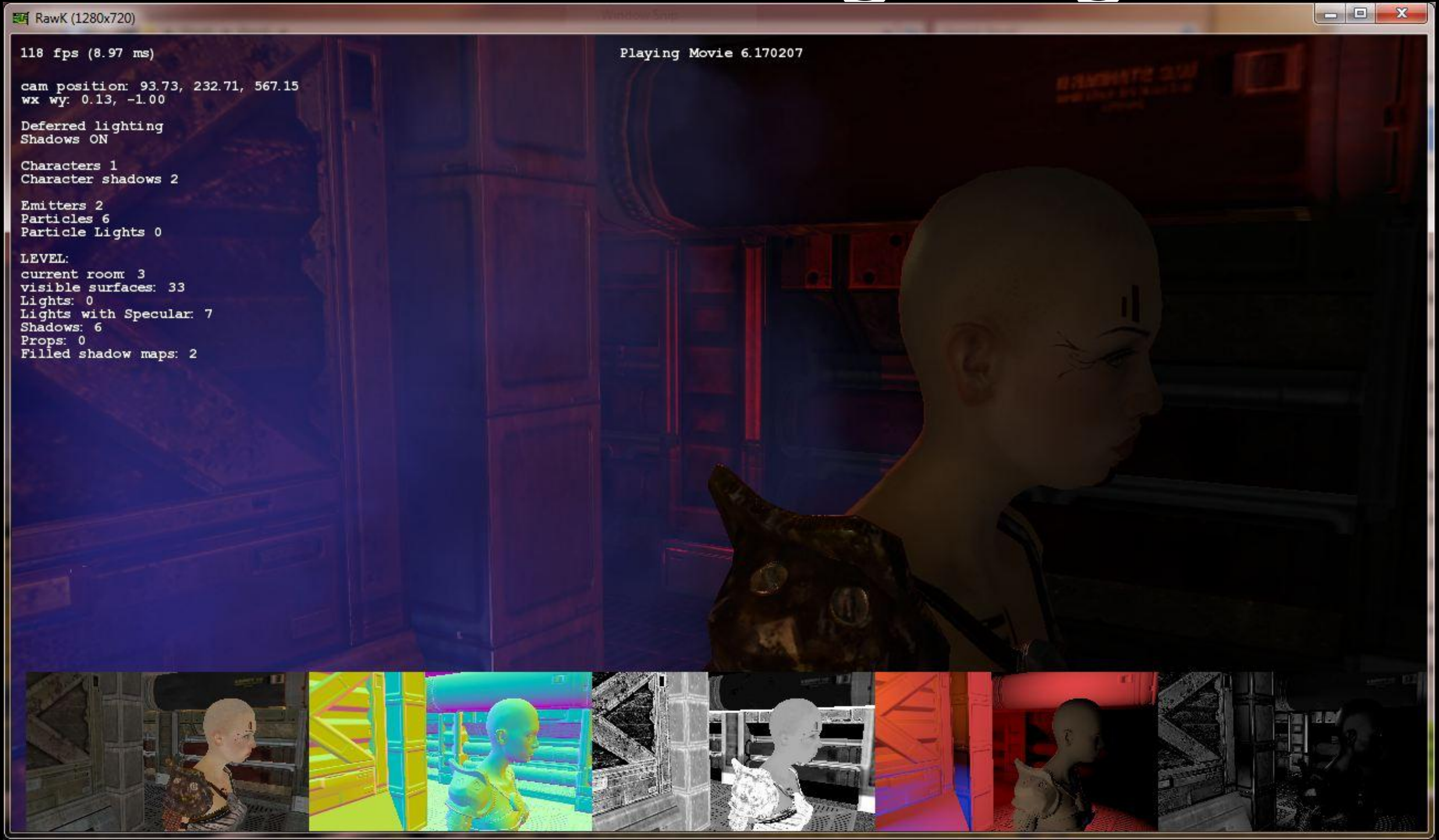
Deferred Lighting



Deferred Lighting

- Geometry pass: fill up normal + spec. power and depth buffer and a color buffer for the ambient pass [Engel09]
- Lighting pass: store light properties in light buffer
- Ambient + Resolve pass: fetch light buffer use its content as diffuse and specular content and add the ambient term while resolving into the main buffer
- Similar to S.T.A.L.K.E.R: Clear Sky [Lobanchikov]

Deferred Lighting



Deferred Lighting

- Light Properties that are stored in light buffer

$$I = A + \sum_i Att_i (N.L_i * LightColor_i * D_{MaterialColor} * D_{Intensity} + (N.H_i)^n * S_{MaterialColor} * S_{Intensity})$$

- Light buffer layout

Channel 1: $\sum_i N.L_i * D_{\mathrm{Red}} * Att_i$

Channel 2: $\sum_i N.L_i * D_{\mathrm{Green}} * Att_i$

Channel 3: $\sum_i N.L_i * D_{\mathrm{Blue}} * Att_i$

Channel 4: $\sum_i lum(N.L_i * (N.H_i)^n * Att_i)$

- $D_{red/green/blue}$ is the light color

Deferred Lighting

- Specular stored as luminance [Engel09][Deferred Lighting specular term]
- Reconstructed with diffuse chromacity

$$chromaticity = \frac{\sum_i (N.L_i * D_{RGB} * Att_i)}{\sum_i lum(N.L_i * D_{RGB} * Att_i) + \varepsilon}$$

$$ApproxSpecular = chromaticity * \sum_i lum(N.L_i * (N.H_i)^n * Att_i)$$

Deferred Lighting

- Memory Bandwidth Optimization (Sandy Bridge supports DirectX 10, / 10.1)
  - Light bounds calculated in Geometry Shader
    - GS bounding box: construct bounding box around light in the geometry shader
    - Render only what is in this box

Deferred Lighting

- Common challenges
  - Alpha blended objects can’t be rendered into the depth buffer -> Deferred Lighting won’t work here
    -> might need separate lighting system for those
    -> or possible solution: order-independent transparency
  - Anti-Aliasing with hardware MSAA is more challenging
    -> possible solution: use MLAA ~ do your own AA as a PostFX in screen-space
  - Material diversity … -> Screen-Space Material System

Ellipsoidal Light Shadows

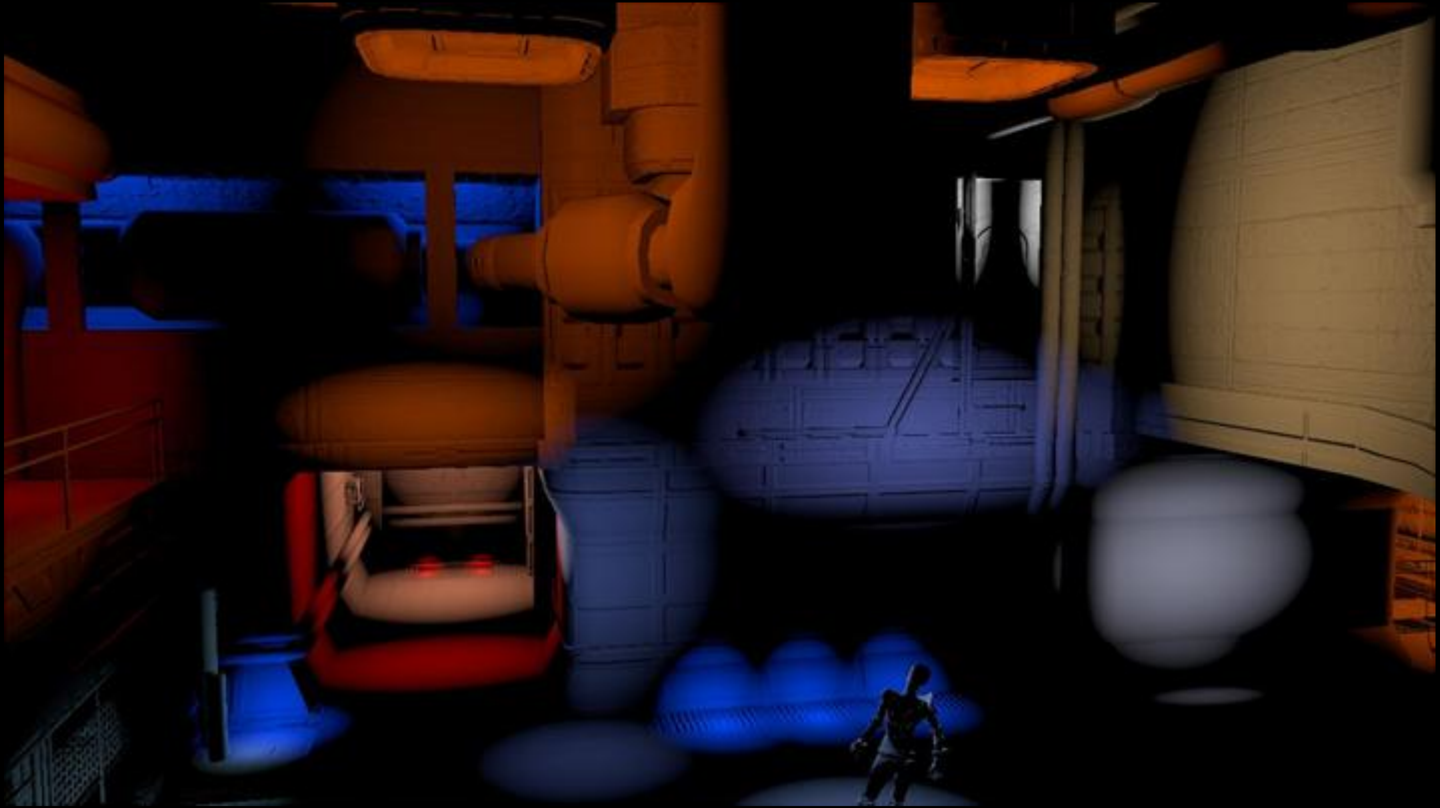
- In a game: many shadows to consider
  - Cloud shadows: just projected down
  - Character self-shadowing : those are optional shadows with their own frustum that is just around the bounding boxes
  - Sun shadows: Cascaded Shadow Maps
  - Shadows from point, spot and other light types

Ellipsoidal Light Shadows

- We only focus on shadows from a light type that we call Ellipsoidal Light
- It is similar to a point light but can have different attenuation values in three directions
- For example the area that the light affects can be ellipsoidal

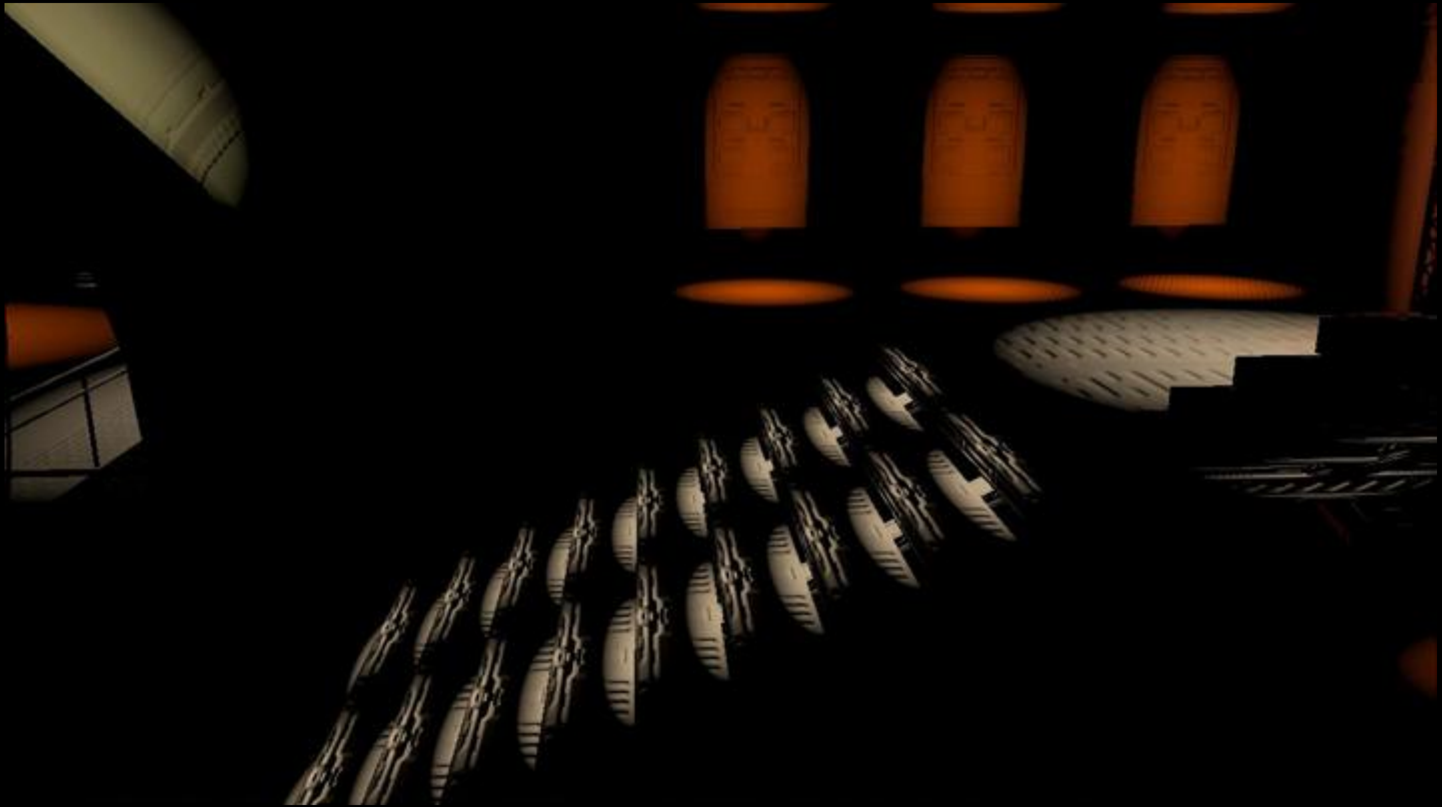
Ellipsoidal Light Shadows

Ellipsoidal Light Shadows

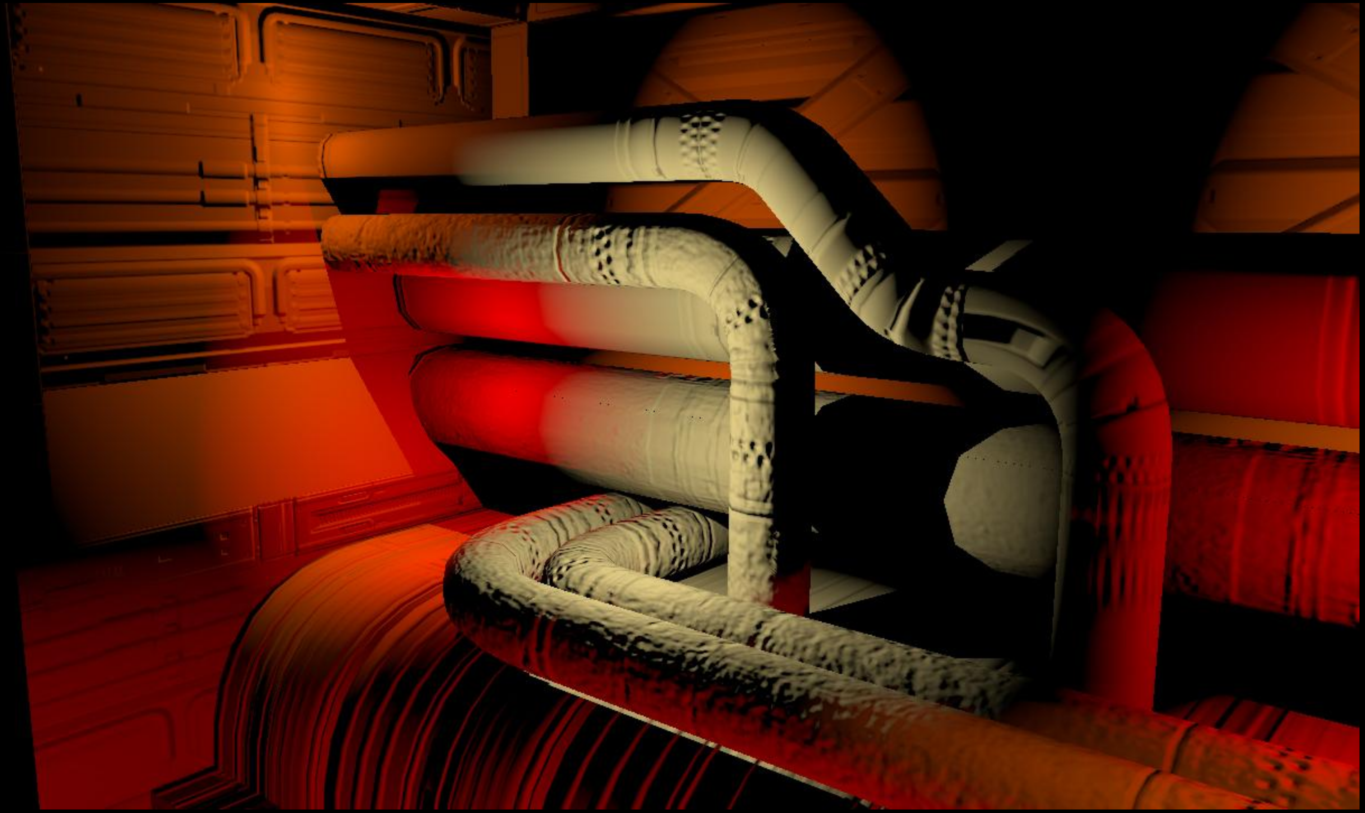


Ellipsoidal Light Shadows

Ellipsoidal Light Shadows



Ellipsoidal Light Shadows



Ellipsoidal Light Shadows

Ellipsoidal Light Shadows

- Mostly four areas of challenge [Engel11]
  - Shadow Rendering
  - Shadow Caching
  - Shadow Bias value
  - Softening the Penumbra

Ellipsoidal Light Shadows

- Shadow Rendering
- Cube shadow maps
  -> more even error distribution than Dual-Paraboloid Shadow Maps
- DirectX 10+ geometry shader helps to render into cube shadow maps in one pass

Ellipsoidal Light Shadows

- Geometry shader optimization:
  - Using geometry shader might be expensive
  - Typical code for this on next slide

Ellipsoidal Light Shadows

```
// Loop over cube faces
[unroll]
for (int i = 0; i < 6; i++)
{
	// Translate the view projection matrix to the position of the light
	float4x4 pViewProjArray = viewProjArray[i];

	//
	// translate
	//
	// access the row HLSL[row][column]
	pViewProjArray[0].w += dot(pViewProjArray[0].xyz, -In[0].lightpos.xyz);
	pViewProjArray[1].w += dot(pViewProjArray[1].xyz, -In[0].lightpos.xyz);
	pViewProjArray[2].w += dot(pViewProjArray[2].xyz, -In[0].lightpos.xyz);
	pViewProjArray[3].w += dot(pViewProjArray[3].xyz, -In[0].lightpos.xyz);

	float4 pos[3];
	pos[0] = mul(pViewProjArray, float4(In[0].position.xyz, 1.0));
	pos[1] = mul(pViewProjArray, float4(In[1].position.xyz, 1.0));
	pos[2] = mul(pViewProjArray, float4(In[2].position.xyz, 1.0));

	// Use frustum culling to improve performance
	float4 t0 = saturate(pos[0].xyxy * float4(-1, -1, 1, 1) - pos[0].w);
	float4 t1 = saturate(pos[1].xyxy * float4(-1, -1, 1, 1) - pos[1].w);
	float4 t2 = saturate(pos[2].xyxy * float4(-1, -1, 1, 1) - pos[2].w);
	float4 t = t0 * t1 * t2;

	[branch]
	if (!any(t))
	{
	// Use backface culling to improve performance
	float2 d0 = pos[1].xy * pos[0].w - pos[0].xy * pos[1].w;
	float2 d1 = pos[2].xy * pos[0].w - pos[0].xy * pos[2].w;

	[branch]
	if (d1.x * d0.y > d0.x * d1.y || min(min(pos[0].w, pos[1].w), pos[2].w) < 0.0)
	{
		Out.face = i;

		[unroll]
		for (int k = 0; k < 3; k++)
		{
			Out.position = pos[k];
			Stream.Append(Out);
		}
		Stream.RestartStrip();
	}
	}
}
```

Ellipsoidal Light Shadows

- This code does:
  - Cube map projection
  - Triangle <-> Frustum culling
  - Triangle backface culling
  - Replicates triangles –if needed- in all six directions

Ellipsoidal Light Shadows

- If hardware doesn’t offer a very performant geometry shader
  -> move projection into the vertex shader

Ellipsoidal Light Shadows

```
[Vertex shader]

float4x4 viewProjArray[6];
float3 LightPos;

GsIn main(VsIn In)
{
	GsIn Out;

	float3 position = In.position - LightPos;

	[unroll]
	for (int i=0; i<3; ++i)
	{
	Out.position[i] = mul(viewProjArray[i*2], float4(position.xyz, 1.0));
	Out.extraZ[i] = mul(viewProjArray[i*2+1], float4(position.xyz, 1.0)).z;
	}

	return Out;
}
```

Ellipsoidal Light Shadows

```
//----------------------------------------------------------------------------
[Geometry shader]

#define POSITIVE_X 0
#define NEGATIVE_X 1
#define POSITIVE_Y 2
#define NEGATIVE_Y 3
#define POSITIVE_Z 4
#define NEGATIVE_Z 5

float4 UnpackPositionForFace(GsIn data, int face)
{
	float4 res = data.position[face/2];

	[flatten]
	if (face%2)
	{
		res.w = -res.w;
		res.z = data.extraZ[face/2];
		[flatten]
		if (face==NEGATIVE_Y)
			res.y = -res.y;
		else
			res.x = -res.x;
	}
	return res;
}
```

Ellipsoidal Light Shadows

```
[maxvertexcount(18)]
void main(triangle GsIn In[3], inout TriangleStream<PsIn> Stream)
{
	PsIn Out;

	// Loop over cube faces
	[unroll]
	for (int i = 0; i < 6; i++)
	{
		float4 pos[3];
		pos[0] = UnpackPositionForFace(In[0], i);
		pos[1] = UnpackPositionForFace(In[1], i);
		pos[2] = UnpackPositionForFace(In[2], i);

		// Use frustum culling to improve performance
		float4 t0 = saturate(pos[0].xyxy * float4(-1, -1, 1, 1) - pos[0].w);
		float4 t1 = saturate(pos[1].xyxy * float4(-1, -1, 1, 1) - pos[1].w);
		float4 t2 = saturate(pos[2].xyxy * float4(-1, -1, 1, 1) - pos[2].w);
		float4 t = t0 * t1 * t2;

		[branch]
		if (!any(t))
		{
			// Use backface culling to improve performance
			float2 d0 = pos[1].xy * pos[0].w - pos[0].xy * pos[1].w;
			float2 d1 = pos[2].xy * pos[0].w - pos[0].xy * pos[2].w;

			[branch]
			if (d1.x * d0.y > d0.x * d1.y || min(min(pos[0].w, pos[1].w), pos[2].w) < 0.0)
			{
				Out.face = i;

				[unroll]
				for (int k = 0; k < 3; k++)
				{
					Out.position = pos[k];
					Stream.Append(Out);
				}
				Stream.RestartStrip();
			}
		}
	}
}
```

Ellipsoidal Light Shadows

- On some hardware platforms with a huge amount of shadows, shadow caching is a challenge
- Use 16-bit cube shadow maps for memory
- Caching Parameters:
  - Distance from shadow to camera
  - Size of shadow on screen
  - Is there a moving object in the area of the light / shadow ?

Ellipsoidal Light Shadows

- Based on those parameters a cube shadow map is updated or not
- Storing 100 256x256x6 16-bit cube maps is about 75 Mb
- If still to much, caching needs to be restricted by distance and then maps are moved in and out into a linked list

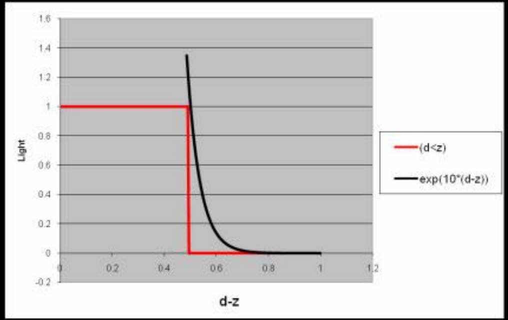
Ellipsoidal Light Shadows

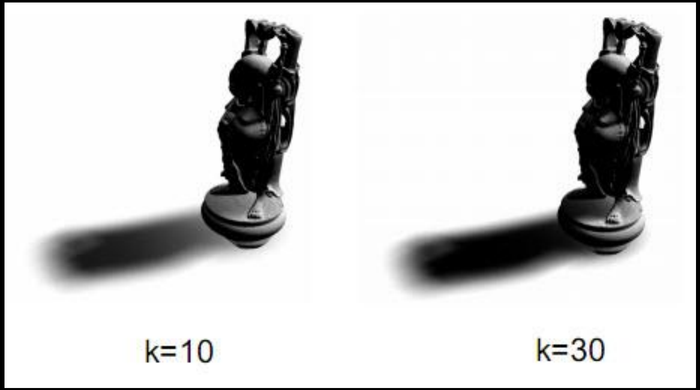
- Shadow Bias Value
- As long as the shadow map comparison for cube shadow maps is binary -> 0 / 1 the depth bias value won’t be correct for all six directions
- Replace binary comparison with Exponential Shadow maps comparison

Ellipsoidal Light Shadows

- Exponential Shadow Maps [Salvi]

```
float depth = tex2D(ShadowSampler, pos.xy).x;
shadow = saturate(2.0 - exp((pos.z - depth) * k));
```





- Approximate step function (z-d> 0) by

```
exp(k*(z-d)) = exp(k*z) * exp(-k*d)
```

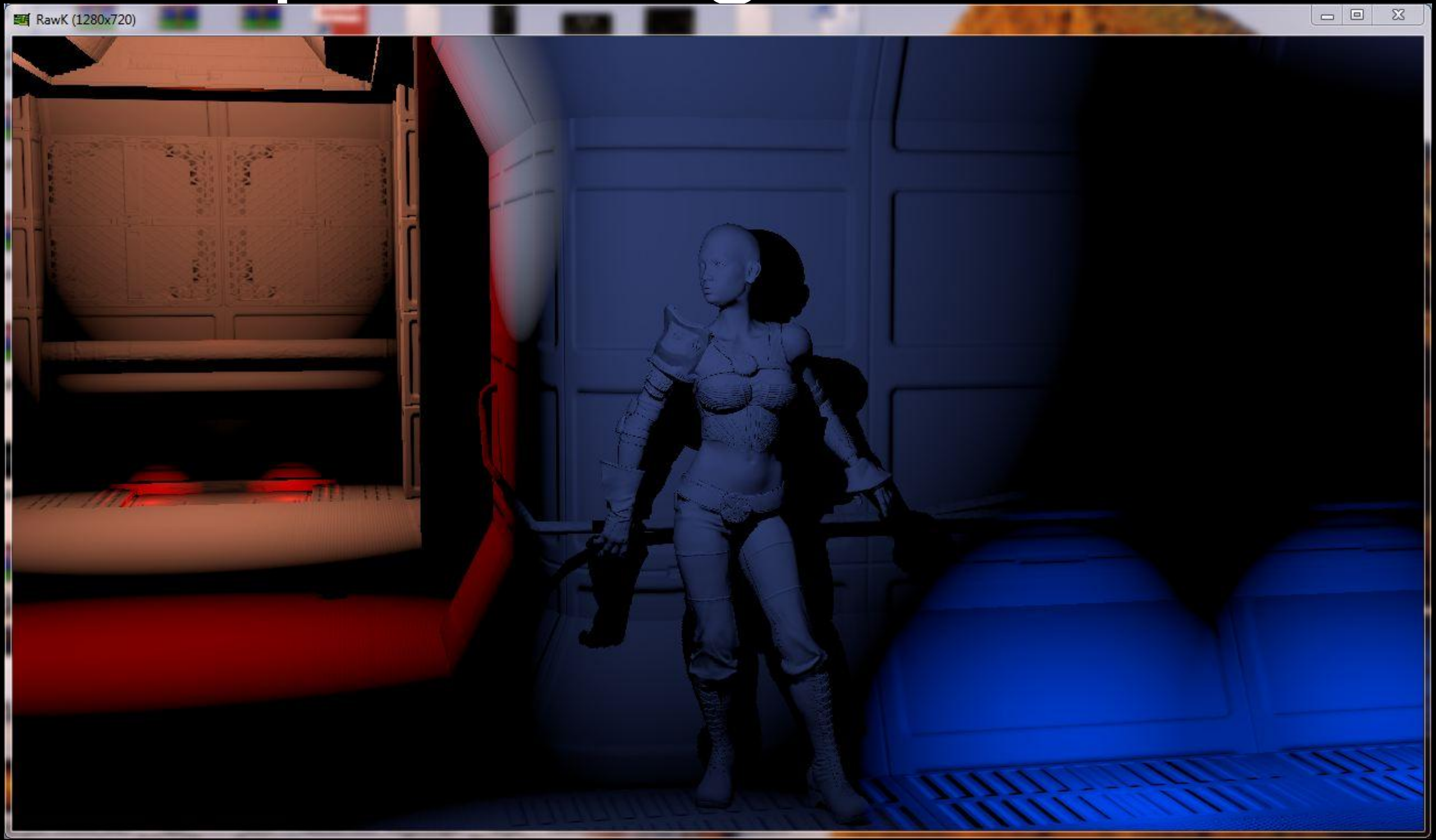
Good overview on the latest development in [Bavoil]

Ellipsoidal Light Shadows

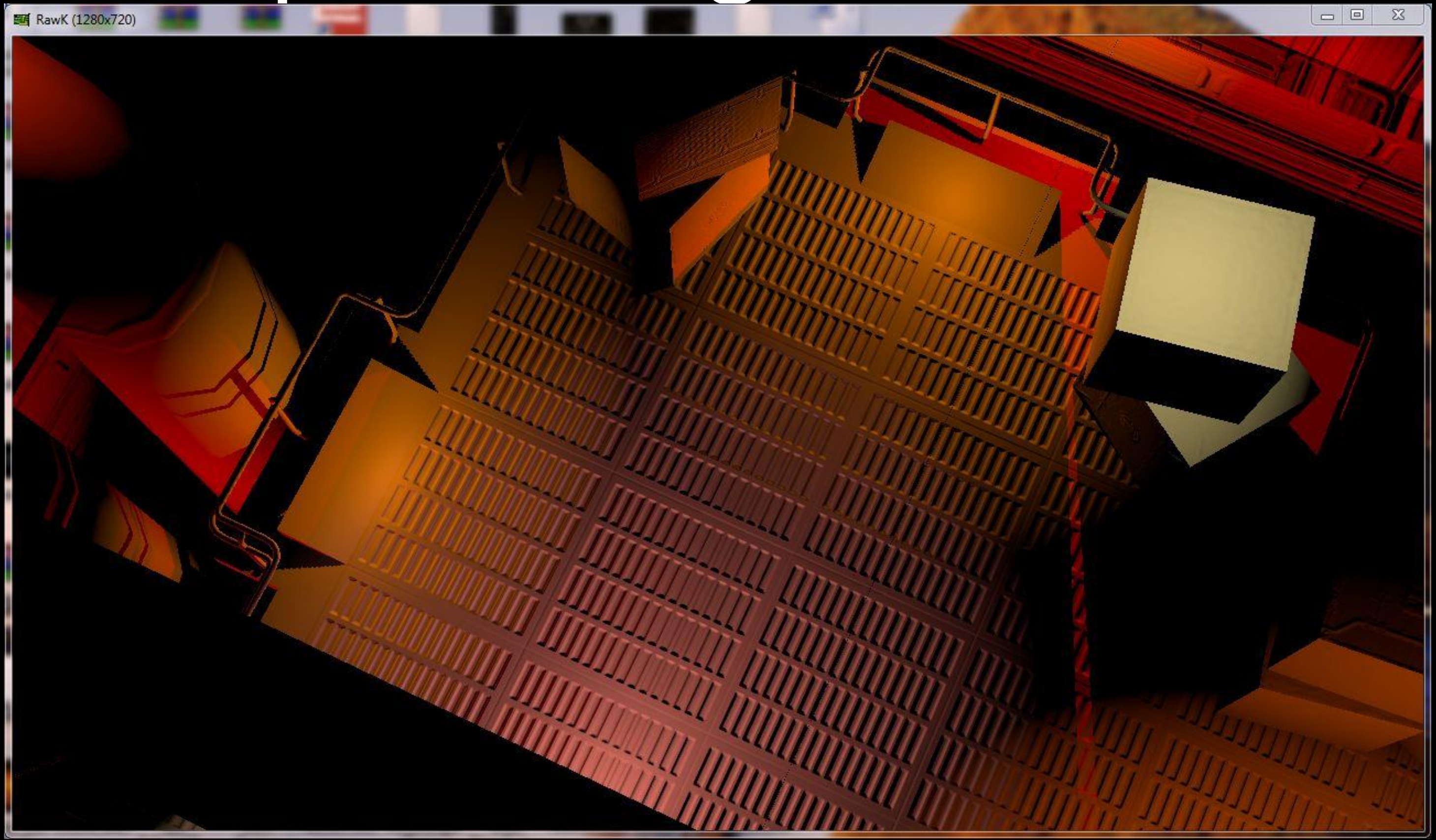
- Softening Penumbra
  -> ESM will soften the penumbra
  -> we don’t use any other softening

Ellipsoidal Light Shadows

Ellipsoidal Light Shadows



Ellipsoidal Light Shadows



Screen-Space Material System

- Instead of applying materials per-object, apply them in screen-space
- Pays off when many objects use the same material like skin
- Just do it for the two most expensive materials or the ones that should have the highest quality. For example:
  - Hair
  - Skin

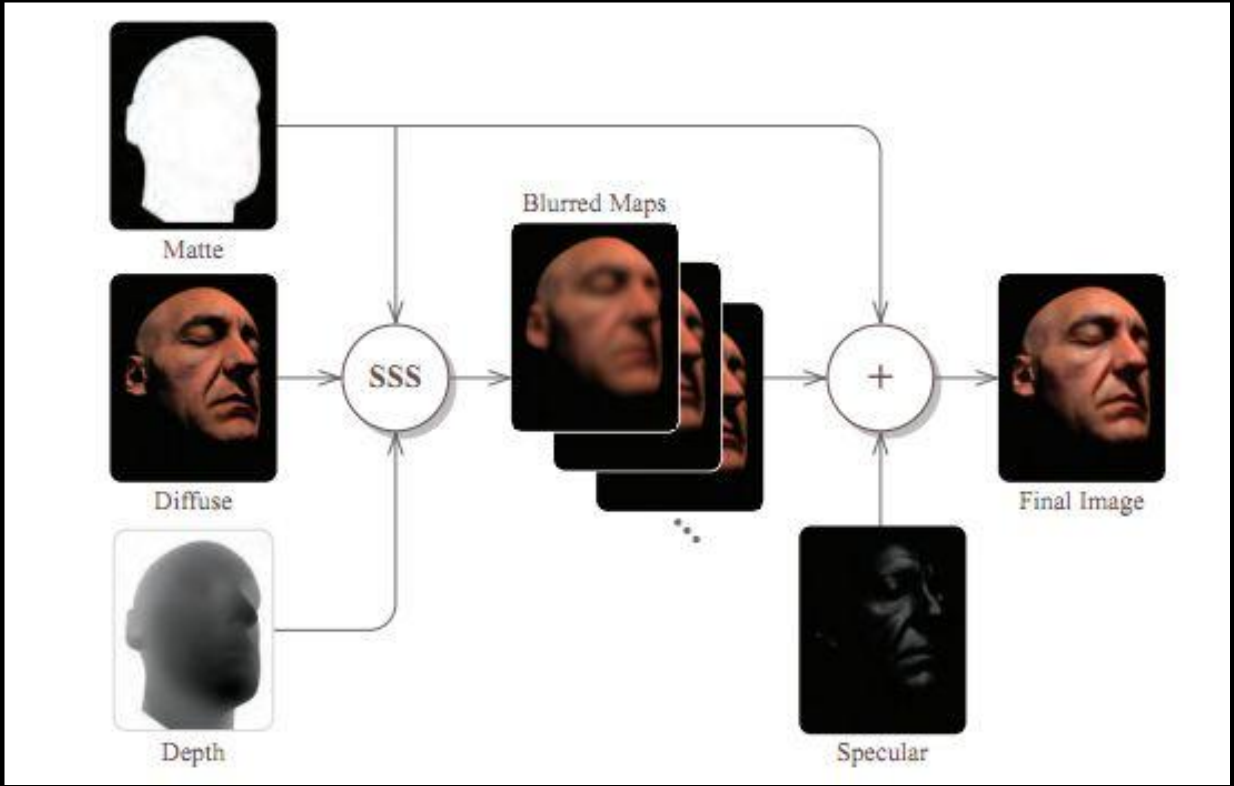
Screen-Space Material System

Screen-Space Material System

Screen-Space Material System

- [Jimenez]:
  - Render skin irradiance (diffuse * light) and mark stencil for areas with skin using material ID
  - Sub-Surface Scattering Screen-Space Filter Kernel (SSSSS)
    Blur using weights to simulate sub-surface scattering
    - Three different screen-space kernels
      - Single bidimensional convolution similar to [Hable]
      - Six 1D Gaussian blurs [d'Eon]
      - Bloom
  - Perform specular calculation in same pass as final blur pass

Screen-Space Material System



Overview of screen-space algorithm (courtesy of [Jimenz])

Screen-Space Material System

- Our approach -> only two passes:
  - Render skin irradiance (diffuse * light) and mark stencil for areas with skin using material ID
  - Sub-Surface Scattering Screen-Space Filter Kernel (SSSSS)
    Blur using weights to simulate sub-surface scattering
    - We use a 1 pass jittered sample approach [Hable]
      -Requires weighting each channel
  - Perform specular calculation in same pass as final blur pass
  - Bloom comes from generic PostFX bloom applied to the scene

Screen-Space Material System

- Mark skin in stencil buffer and render skin irradiance

```
float2 uv = In.texCoord.xy;
float4 diffuse = gDiffuseTx.SampleLevel( gFilter, uv, 0 );

if( diffuse.a < 0.99f )
{
  discard;
}

float3 light = gLightTx.SampleLevel( gFilter, uv, 0 ).rgb;
return float4( diffuse.rgb * light.rgb, 1.0f );
```

Screen-Space Material System

- Run the Sub-Surface Scattering Screen-Space Filter Kernel (SSSSS (S^5))
  - Calculate the depth gradient [Jimenez]

$$s_x = \frac{\alpha}{d(x, y) + \beta \cdot abs(\nabla_x d(x, y))},$$
$$s_y = \frac{\alpha}{d(x, y) + \beta \cdot abs(\nabla_y d(x, y))},$$

  - Large depth gradients reduce the size of the convolution filter kernel
    -> limits the effects of background pixels being convolved
  - Additionally scales based on distance to camera

Screen-Space Material System

- Run the Sub-Surface Scattering Screen-Space Filter Kernel (SSSSS (S^5))
  - Sample in a disc shape [Hable] -> single-pass bidimensional convolution at 13 jittered sample points
    - Direct reflection – one point
    - Mid-level scattering – six points
    - Wide red scattering – six points
  - Code on following slides

Screen-Space Material System

```
float2 pixelSize = 1.0f / dim;
const int samples = SAMPLE_COUNT;

float depth = depthTx.Sample( pointFilter, uv );

float2 s = float2( 0.0f, 0.0f );
float2 step = float2( 0.0f, 0.0f );

// empirical values
const float maxdd = 0.001f;
const float correction = 800.0f;        // following [Jimenez]
const float ssLevel = 20.0f;             // following [Jimenez]

// [Jimenez] depth gradients for scaling kernel + distance to camera
step.xy = ssLevel.xx * width.xx * pixelSize.xy;
float2 gradient = min( abs( float2( ddx( depth ), ddy( depth ) ) ), maxdd.xx );
s.xy = step.xy / (depth.xx + gradient.xy * correction.xx );
```

Screen-Space Material System

```
  float2 offset = s / samples;

  float3 color = float3( 0.0f, 0.0f, 0.0f );
  float4 middle = colorTx.Sample( pointFilter, uv );
  float3 totalWeight = 0.0f;

  [unroll]
  for( int i = 0; i < samples; ++i )
  {
    float2 pos = uv + blurJitteredSamples[ i ] * offset;     // sample points in a disc
    float4 sample = colorTx.Sample( bilinearFilter, pos );
    float  weight = sample.a;
    color += sample.rgb * weight.xxx * blurJitteredWeights[ i ].xyz; // jittered weights [Hable]
    totalWeight += weight.xxx * blurJitteredWeights[ i ].xyz;;
  }

  if( length(totalWeight) < 0.1f )
  {
    return middle;
  }
  else
  {
    return float4( color / totalWeight, middle.a );
  }
}
```

Screen-Space Material System

- Perform specular calculation in same pass as final blur pass
  -> see slide on specular for Deferred Lighting above
- For other materials than skin: use different stencil value to mark those … watch out for future research results

Acknowledgement

- Michael Alling did the work on the S^5 effect and the PostFX pipeline
- Igor Lobanchikov came up with the Geometry Shader optimization trick
- Jared Marsau implemented EmotionFX and Fmod and worked on the art and sound pipelines … including create sounds ☺
- Peter Santoki … we also call him Michael Bay

Interns

- Doing cool graphics research such as -> work on RawK II has started
    - New direct and indirect lighting algorithms
    - New soft shadow algorithms
    - Order-Independent Transparency
    - Hardware Tesselation
    - Etc.
- Send e-mail to wolf@conffx.com

Contact

- wolf@conffx.com
- www.conffx.com
- Click “like” at http://www.facebook.com/pages/Confetti-Special-Effects-Inc/159613387880?v=wall

References

- [Deferred Lighting specular term] Read the overview on different Deferred Lighting Approaches by Naty Hoffman: http://www.realtimerendering.com/blog/deferred-lighting-approaches/
- [d'Eon] Eugene d'Eon, David Luebke, and Eric Enderton, "Efficient Rendering of Human Skin", Eurographics 2007http://http.developer.nvidia.com/GPUGems3/gpugems3_ch14.html
- [Engel09] Wolfgang Engel, “Light Pre-Pass -Deferred Lighting: Latest Development-”, SIGGRAPH 2009, http://www.google.com/url?sa=t&source=web&cd=1&ved=0CBUQFjAA&url=http%3A%2F%2Fwww.bungie.net%2Fimages%2FInside%2Fpublications%2Fsiggraph%2FEngel%2FLightPrePass.ppt&ei=fFfCTfifKJGJhQfW1sixBQ&usg=AFQjCNGveO2pv79oiDkw-9h9FyK5wbihuA
- [Engel11] Wolfgang Engel, “Shadows - Thoughts on Ellipsoid Light Shadow Rendering”, http://altdevblogaday.org/2011/02/28/shadows-thoughts-on-ellipsoid-light-shadow-rendering/
- [Hable] John Hable, George Borshukov, and Jim Hejl, "Fast Skin Shading", ShaderX7, http://www.shaderx7.com/TOC.html
- [Jimenez] Jorge Jimenez, Veronica Sundstedt, and Deigo Gutierrez, "Screen-Space Perceptual Rendering of Human Skin", http://giga.cps.unizar.es/~diegog/ficheros/pdf_papers/TAP_Jimenez_LR.pdf
- [Lobanchikov] Igor A. Lobanchikov, “ GSC Game World‘s S.T.A.L.K.E.R : Clear Sky – a showcase for Direct3D 10.0/1”, http://developer.amd.com/gpu_assets/01GDC09AD3DDStalkerClearSky210309.ppt