Practical Particle Lighting

tobias.persson@bitsquid.se

bitsquid

GAME DEVELOPERS CONFERENCE
SAN FRANCISCO, CA
MARCH 5-9, 2012
EXPO DATES: MARCH 7-9
2012

Overview

- Introduction
- Basic Particle Lighting
- Improvements
- Conclusions

Introduction: Bitsquid

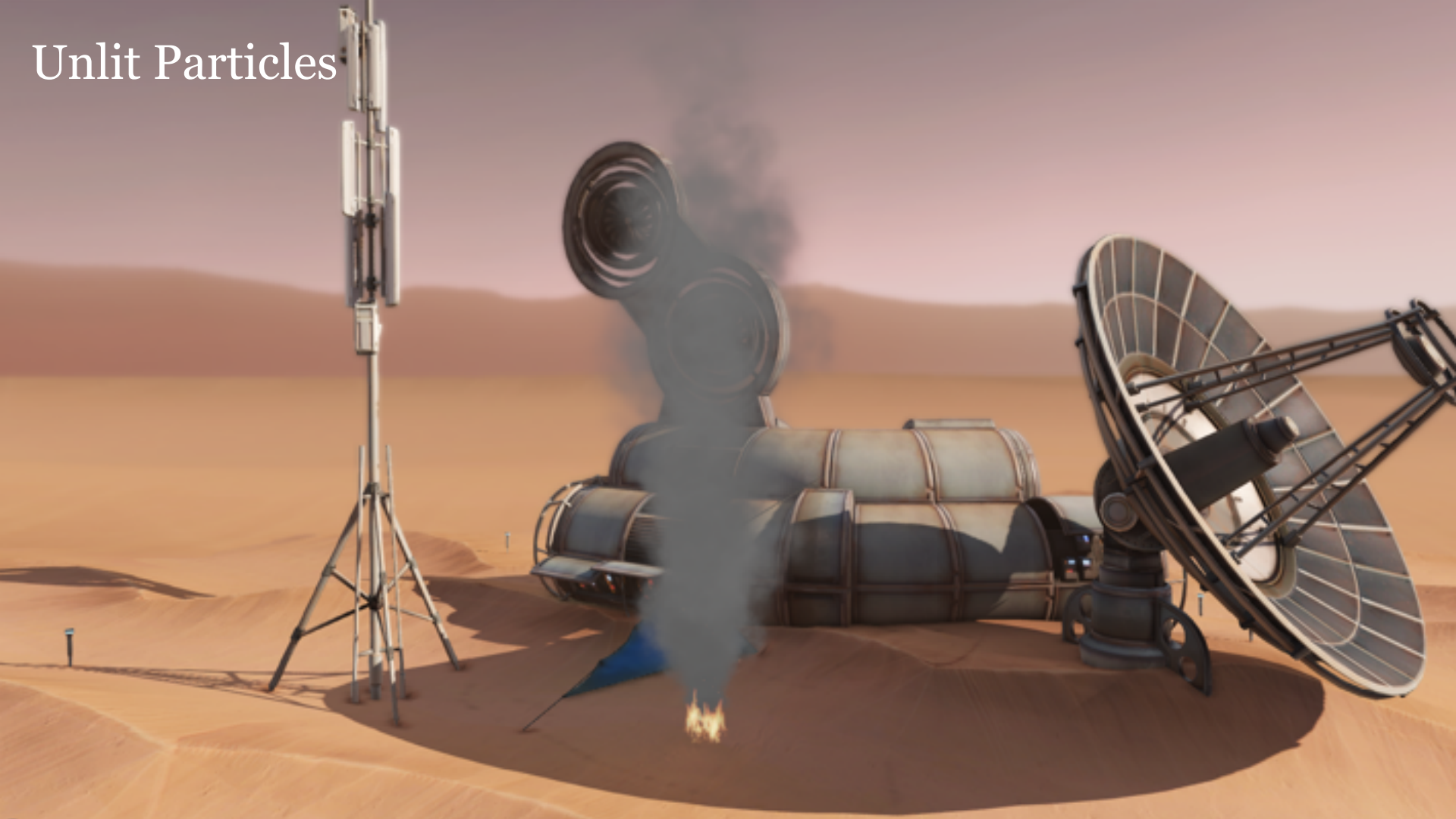
- High-end game engine for licensing
- Currently powering 10 titles in production
  - Team sizes between 15 and 40 developers
- Techniques presented used in one announced title so far
  - “War of the Roses”, Fatshark / Paradox Interactive

“War of the Roses”
Courtesy of Fatshark and Paradox Interactive

Introduction: Particle Lighting

- a.k.a. Billboard lighting
- Focus on making billboards fit in with the environment
  - Must support dynamic local lights as well as global lighting environment
- Cheap enough to be used on all non-emissive particles
  - Keep PS work to a minimum, push bulk cost to earlier stage (VS or DS or even off GPU on some arch.)

Unlit Particles



Lit Particles

Vertex Lighting

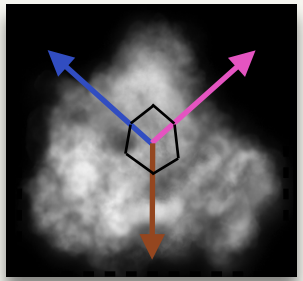
- Super cheap
  - Calc incoming light per-vertex in VS (or on CPU)
  - Modulate with particle color in PS
- Solves problem with fitting into rest of the scene
- Better than nothing but looks very flat
  - No sense of incoming light direction biggest problem
- Can we do better?

Resurrecting an old friend: HL2-basis

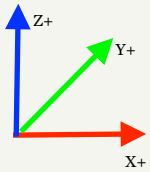
- Project lighting environment to HL2-basis[1]
- Align HL2-basis vectors with billboard (i.e view space)

$$\left(-\frac{1}{\sqrt{2}} \quad -\frac{1}{\sqrt{3}} \quad \frac{1}{\sqrt{6}}\right)$$

$$\left(\frac{1}{\sqrt{2}} \quad -\frac{1}{\sqrt{3}} \quad \frac{1}{\sqrt{6}}\right)$$



$$\left(0 \quad -\frac{1}{\sqrt{3}} \quad -\sqrt{\frac{2}{3}}\right)$$



Lighting using HL2-basis

- In VS: For all light sources affecting the billboard vertex - accumulate incoming light

```
void accumulate_lighting(Output o, half3 light_dir, half3 light_col, half atten) {
     light_col *= atten;
     half3 weights = saturate(dot(light_dir, hl2_basis0),
                    dot(light_dir, hl2_basis1),
                    dot(light_dir, hl2_basis2));
     o.basis_col0 += light_col * weights.x;
     o.basis_col1 += light_col * weights.y;
     o.basis_col2 += light_col * weights.z;
}
```

Lighting using HL2-basis

- To evaluate per pixel lighting in the PS we need some form of normal
  - Allow VFX artist to provide a normal map
    - Extra texture lookup + tangent space transform
    - Consider encoding normal in same texture as diffuse
  - For low-frequency content (smoke & dust etc) some simple curvature approximation is enough, we use:

```
// billboard_normal == -view_direction
half3 n = lerp(billboard_normal, normalize(corner-center), curvature_amount);
// rotate it to view space
n = mul(normalize(n), (float3x3)view);
```

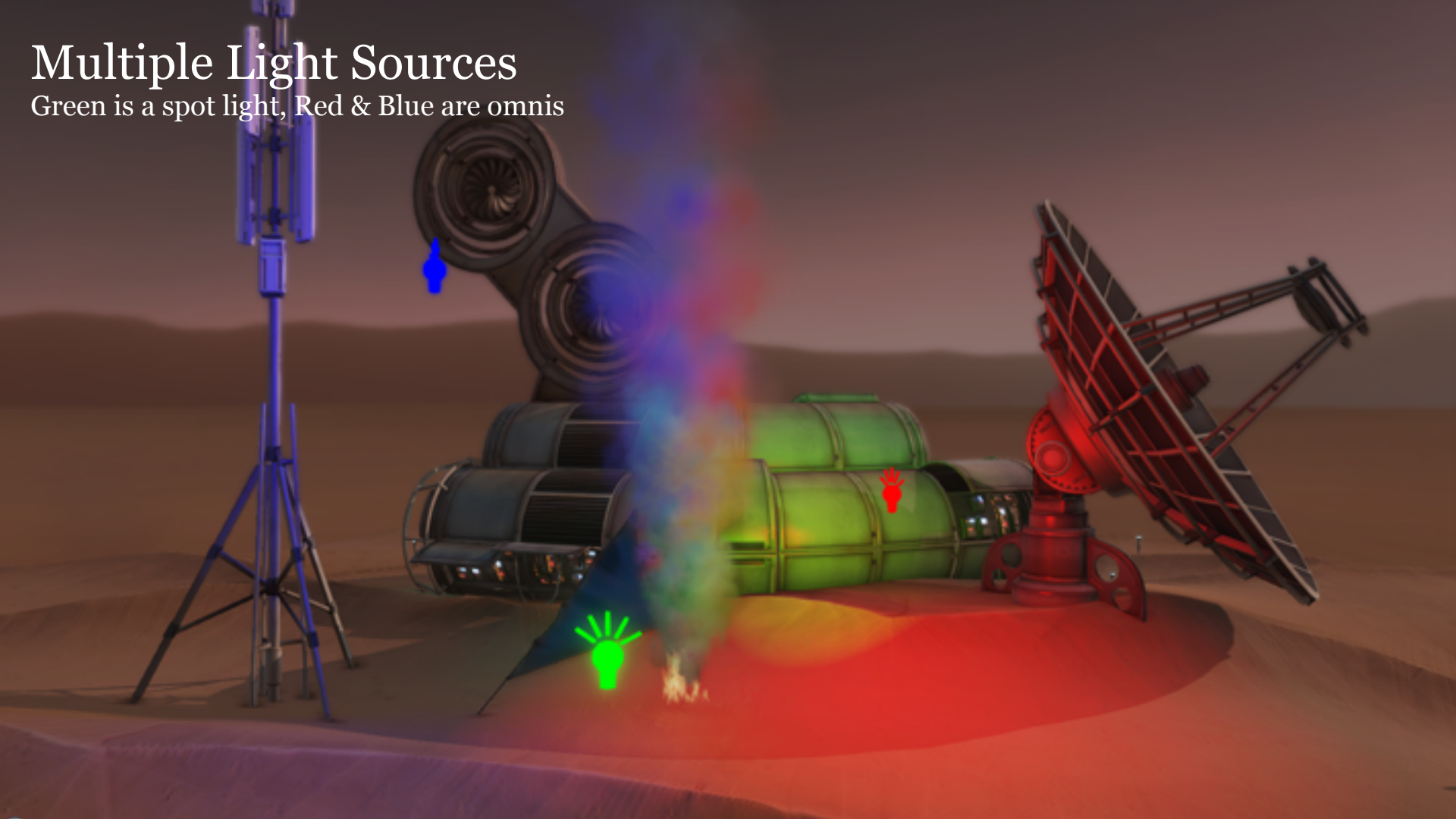
Lighting using HL2-basis

- Evaluating the incoming per-pixel light is simple

```
half3 n = normalize(i.normal);
half3 w = saturate(dot(n, hl2_basis0), dot(n, hl2_basis1), dot(n, hl2_basis2));
half3 diffuse_light = i.basis_col0 * w.x + i.basis_col1 * w.y + i.basis_col2 * w.z;
```

Multiple Light Sources

Green is a spot light, Red & Blue are omnis



View space HL2-basis

- Gives good enough indication of light direction
- Cheap
- Decently compact representation (3xfloat3)

Improvements

- Shadow receiving
- Increasing light sampling frequency (DX11)
- Shadow casting
- Quick note on self-shadowing techniques

Shadow Receiving

- Shadow map look-up in VS
  - Requires hardware with fast VS texture reads
- Recycling of shadow map RTs can cause problems
  - You might not have them around by the time you render the billboards
    - Deferred CSM: Render biggest slice last and let it cover entire frustum
      - In 16:9 you are probably almost doing that already
    - However low-res shadow map is fine since sample frequency is per-vertex
      - Consider keeping low-res versions of your shadow maps around

Sampling shadow map in vertex shader

Increasing light sampling frequency

- On DX11 HW we can use tessellation to increase sampling frequency of the shadow map
  - And rest of the lighting environment if desired
    - More precise capturing of light attenuation
- Simple to implement
  - Push VS light accumulation code down to DS
  - HS main is just a simple pass-through shader
    - Patch constant function is not though..

Increasing light sampling frequency

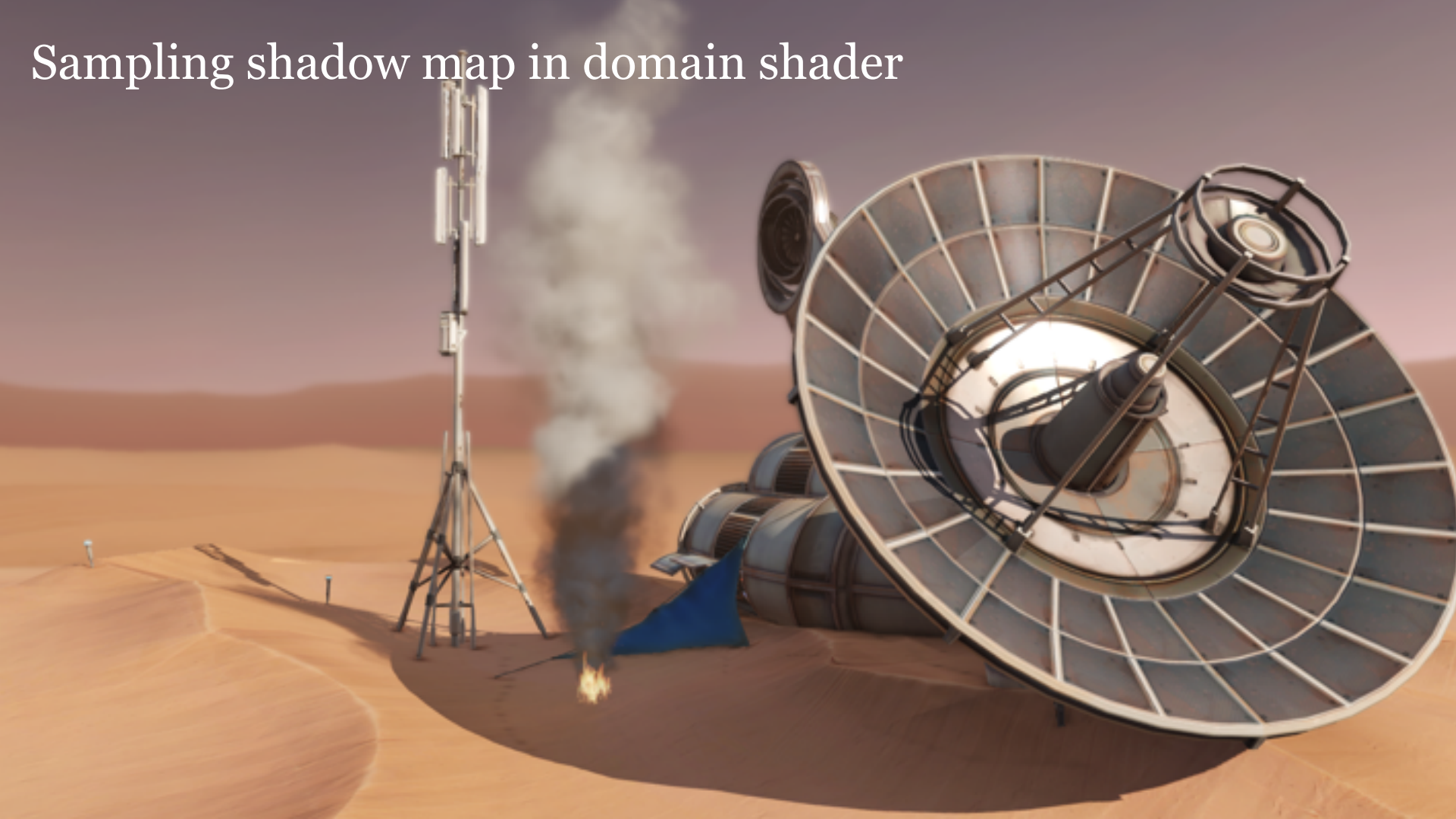
- Be careful not to over-tessellate in the distance
  - LOD metric that strives for constant screen space sized triangles

```
const float wanted_tri_size = 16; // 16x16 pixel triangles

// p0, p1, p2 are our patch corners in screen space pixel coordinates
float3 edge_tess_factors = float3(
   length(p2-p1) / wanted_tri_size,
   length(p2-p0) / wanted_tri_size,
   length(p1-p0) / wanted_tri_size);

float inside_tessellation = max(edge_tess_factors.x, max(edge_tess_factors.y,
   edge_tess_factors.z);
```

Sampling shadow map in domain shader

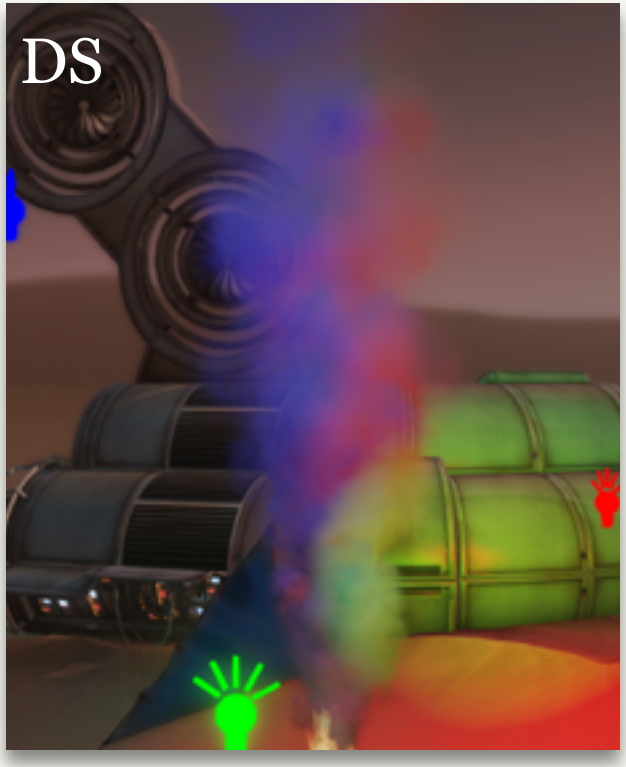


Quality Comparison

Evaluating lighting in domain shader

Green is a spot light, Red & Blue are omnis

Quality Comparison



Performance Comparison

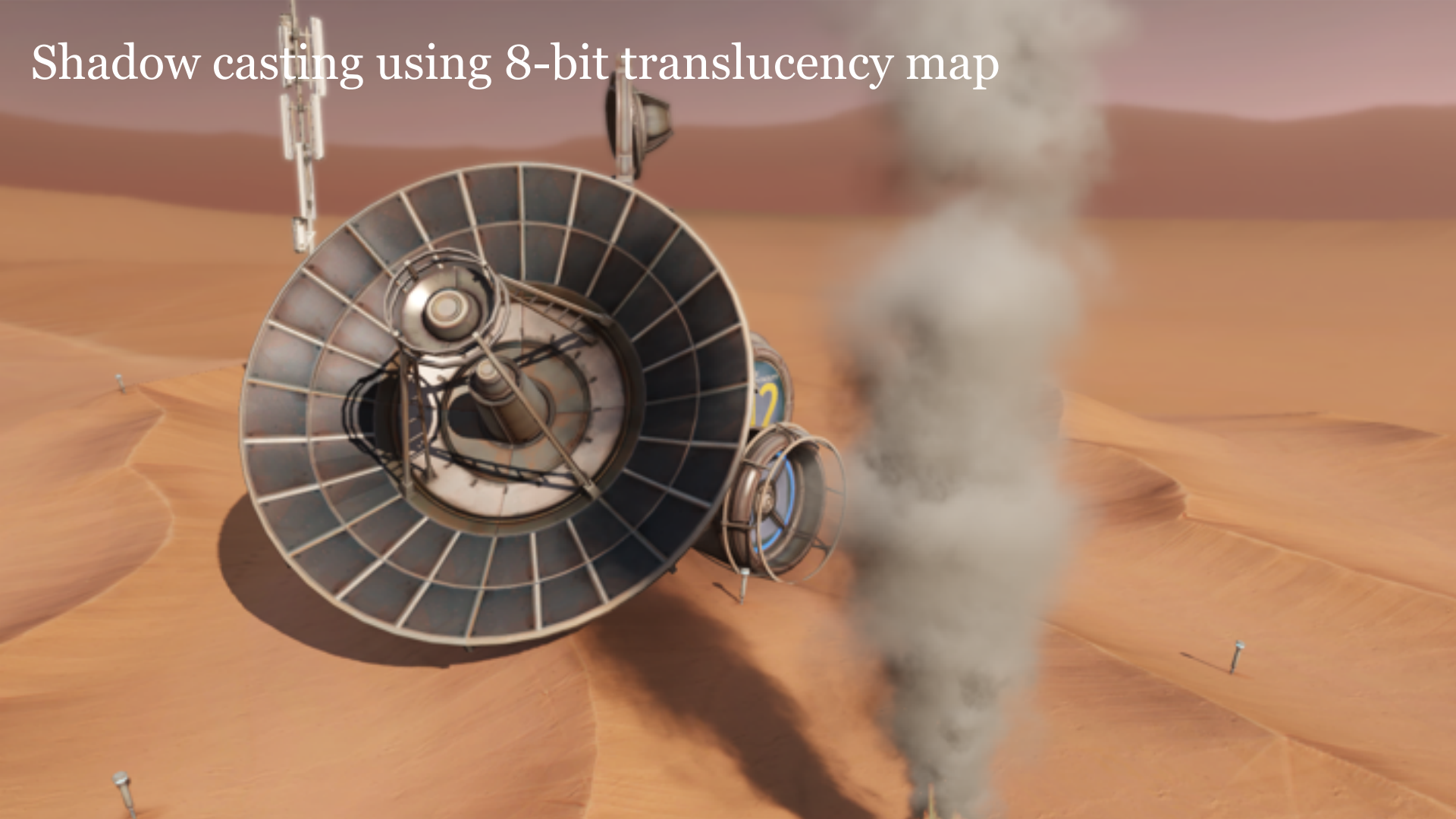
- Timings done using D3D11_QUERY_TIMESTAMP

Sample Frequency	*Time (ms)*	*Time (ms)*
Vertex Shader	0.32	2.83
Domain Shader (32pix)	0.72	4.1
Domain Shader (16pix)	0.82	5.3
Pixel Shader	2.6	20.2

Shadow Casting

- Casting shadows onto opaque geometry is straightforward
- We use same technique as Crysis2 [2]
  - Render particles back-front, accumulate translucency (blended particle alpha) in single channel 8-bit RT
  - Use opaque shadow map as DST with depth test enabled to avoid back-projection
  - CSM: Render shadow casting particles for each cascade
  - Combine with shadow intensity from opaque shadow map
    - Needs matching filter kernels

Shadow casting using 8-bit translucency map



Shadow casting using 8-bit translucency map

Quick note on Self-Shadowing

- Lots of research in this area
  - Fourier Shadow Mapping [4], “Half-angle Slice Rendering” [5], Opacity Shadow Maps [6] + variations
  - None of them scalable enough to use in large-scale in-game scenarios
    - We have a large area to cover with high depth complexity
      - Need “CSM-style” solution
  - Perfect for cut-scenes and contained environments though

Conclusions

- Most important: make your particles fit in with the lighting environment
- Simple techniques takes you a long way, vertex lighting from key light better than nothing
- HW tessellation is usable for more stuff than displacement mapping

Thanks!

- Philip Klevestav for letting me use his sci-fi environment
  - http://www.philipk.net
- Bitsquid team, Iain Cantly, Jon Jansen, Miguel Sainz, Nicolas Thibieroz, Yury Uralsky for great feedback!

Questions?

- More Bitsquid @ GDC2012
  - “Cutting the Pipe: Achieving Sub-Second Iteration Times” Wednesday 11:00, Room 3022, Niklas Frykholm
  - “Flexible Rendering for Multiple Platforms” Thursday 2:30, Room 2011, Tobias Persson
- Contact
  - tobias.persson@bitsquid.se / @tobias_persson
  - slides -> http://www.bitsquid.se

Resources

- [1] Half-Life 2 / Valve Source Shading
  - http://www2.ati.com/developer/gdc/D3DTutorial10_Half-Life2_Shading.pdf
- [2] Secrets of CryENGINE 3 Graphics Technology
  - http://advances.realtimerendering.com/s2011/index.html
- [3] Fourier Opacity Mapping
  - http://www.sci.utah.edu/~bavoil/research/shadows/FourierOpacityMapping_I3D2010.pdf
- [4] Volumetric Particle Shadows
  - http://www.naic.edu/~phil/hardware/nvidia/doc/src/smokeParticles/doc/smokeParticles.pdf
- [5] Opacity Shadow Mapping, Kim and Neumann (2001)

– Bonus Slides

Billboard back-lighting

- Cheap man's light scattering
- In Vertex-/Domain Shader:
  - During light accumulation, also calculate incoming light for backside of billboard
    - Single direction: -billboard normal
    - If light casts particle shadows - modulate light attenuation with translucency map
- In Pixel Shader:
  - Modulate back-lighting with inverse opacity value (1-alpha) multiplied by some artist-tweakable translucency value


Back-lighting disabled

Back-lighting with artist-tweaked attenuation

Back-lighting with attenuation from translucency map

Notice the rim-lighting