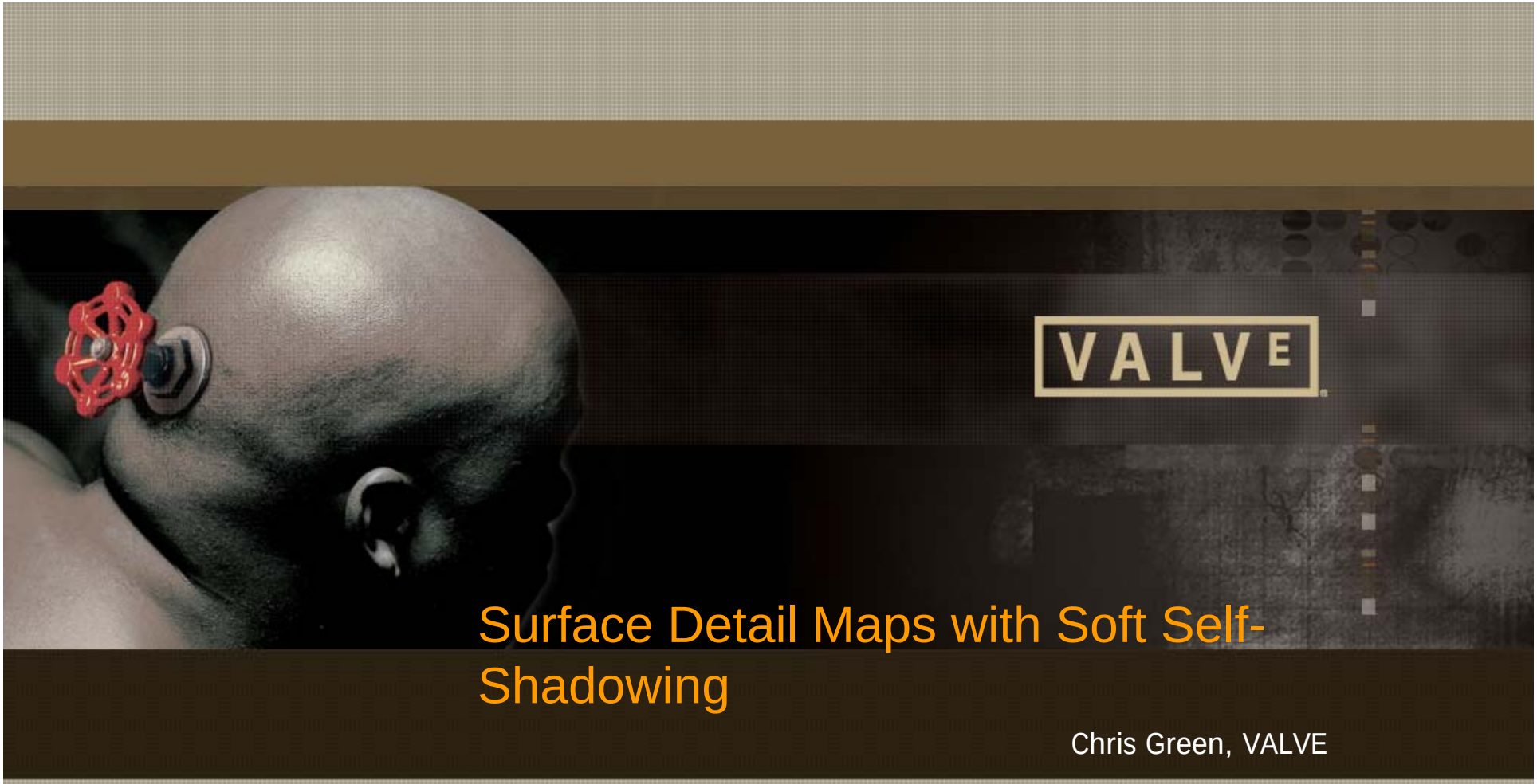


Surface Detail Maps with Soft Self-Shadowing

Chris Green, VALVE

VALVE

Contribution

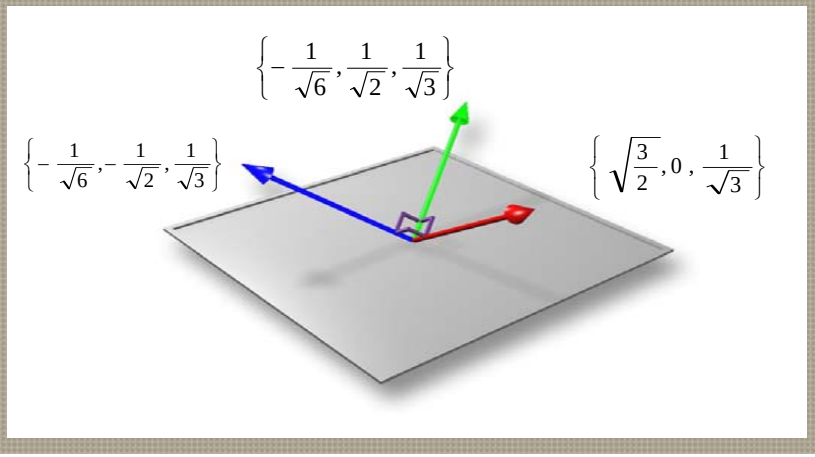
A replacement format for surface bumpmaps

- Encodes surface orientation, ambient occlusion, self-shadowing, and albedo modulation in one 3-component texture.
- Fits seamlessly into the Source engine's radiosity lightmap system.
- Renders faster than our ordinary bumpmap format.

VALVE

Radiosity bumpmapping in the Source Engine

- An offline radiosity program generates lightmap data including direct lighting, indirect lighting, and shadows from static light sources.
- For bumpmapped surfaces, instead of storing one lighting value per luxel, 3 values are stored, representing the lighting for 3 different surface orientations.



VALVE

Radiosity bumpmapping in the Source Engine

- At rendering time, the actual surface normal is sampled from a bumpmap texture.
- The projection of this normal vector onto each of the 3 basis vectors is then used to blend between the 3 precomputed lighting values.

```
float3 dp;
dp.x = saturate( dot( normal, bumpBasis[0] ) );
dp.y = saturate( dot( normal, bumpBasis[1] ) );
dp.z = saturate( dot( normal, bumpBasis[2] ) );
dp *= dp;

diffuseLighting = dp.x * lightmapColor1 +
                  dp.y * lightmapColor2 +
                  dp.z * lightmapColor3
```

- **21 Multiplies, 12 Adds, 3 Saturates**

Simplifying the math by changing the bumpmap data

- The weights used for a given bumpmap texel are always the same!
- If we know that a bumpmap texture is going to be used on geometry in this way, we can just directly store the blend weights in the texture.
- Math becomes:

```
diffuseLighting = texel.x * lightmapColor1 +
                  texel.y * lightmapColor2 +
                  texel.z * lightmapColor3;
```

9 Multiplies, 6 adds. >2x reduction in operations.

Taking advantage of the representation

- Since we are now just storing the weights, there is no reason these weights need to sum to 1.0.
- If we store values which sum to <1, we'll achieve a darkening effect
  - If we multiply the weights by an ambient occlusion value, we get surface orientation and ambient occlusion encoded in 3 components, "for free".
  - We can store a brightness modulation factor in the normal map "for free". Allows use of low resolution color maps with high resolution normal maps and surface albedo modulation.

VALVE

Directional ambient occlusion

- If we scale the blend weights in the bumpmap texture nonuniformly, light coming from some directions will be attenuated differently.
- This can be used to encode a directional ambient occlusion term which encodes a self-shadowing component in the bumpmap, with no added shader instructions.

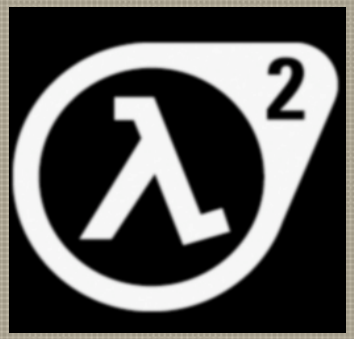
Generating the self-shadowing data

- Given an input height map or other geometric surface description, we can generate the self-shadowing information using a modified ambient occlusion generator:
  - Apply a bilateral smoothing filter to the height data to reduce stair-stepping.
  - Trace many rays from each surface point to see if they are blocked.
  - Each unblocked ray contributes to the 3 weight values based upon its dot product with the corresponding basis vectors.
  - The dot product is raised to a power, which controls the tradeoff between diffuse and sharp shadows.

Generating the self-shadowing data

- Execution time of the generator can be a burden:
  - If the input height map is 1024x1024, and we want to trace 256 rays per sample, that's 2^28 rays to trace.
  - The database being traced against is 2^21 triangles.
  - If a tiled texture is being generated, we need to either add multiple tiled copies of the height map to our ray-tracing database, or make the ray tracer know about tiling.
- We use optimized SIMD kd-tree-based ray-tracing code.
- The directional ambient occlusion generator is fully threaded.
- Still many minutes for 1024x1024 on a 4 processor system.
- Alternatively, the data can be generated in any rendering package which has good area light support, by placing red, green, and blue area lights in the scene which are lined up with the basis vectors.

Generating the self-shadowing data



Input height map

Generated normal map

Generated weights

Generated directional ambient occlusion

Rendering

- Diffuse light from static light sources is handled by multiplying the stored weight values by the 3 input lightmap pixel values.
- Diffuse light from dynamic light sources is modelled by projecting the dynamic light source direction into the local bump map basis, and using the stored weight values to modulate the incoming light. This provides self-shadowing from dynamic light sources.
- For specular lighting, a normal vector is derived by summing the basis vectors together, weighted by the bumpmap texture, and then normalizing. This provides something akin to a “bent” normal.

Rendering

Bumpmapped rendering, lit by the flashlight

Rendering

Ambient occlusion rendering

Rendering

Self-shadowed rendering

Rendering

Scene with static and dynamic lighting.

Other features

- Since our textures are just unsigned blend weights as opposed to sign normal vectors, ordinary image-processing tools (such as Photoshop filters) will work without modification and will produce meaningful, useful results.
- No special handling is needed for mip-mapping or mip-map generation.
- Multiple directional ambient-occlusion textures can be easily combined i.e. for detail texturing and texture blending.

Extensions

- Sample lighting in more than 3 directions. In particular, add a 4th channel representing light perpendicular to the surface. Going beyond 4 directions will allow better shadow sharpness.
- Diffuse inter-reflection could be simulated during the texture generation process.
- Apply to the generation process when mapping high resolution models onto low polygon meshes.
- Can be combined with relief mapping

Questions?

cgreen@valvesoftware.com

http://valvesoftware.com/publications.html