Screenspace Effects

Introduction

- General idea:
  - Render all data necessary into textures
  - Process textures to calculate final image
- Achievable Effects:
  - Glow/Bloom
  - Depth of field
  - Distortions
  - High dynamic range compression (HDR)
  - Edge detection
  - Cartoon rendering
  - Lots more…

Hardware considerations

- Older hardware:
  - Multipass and Blending operators
  - Is costly and not very flexible
- Newer hardware:
  - Shaders render into up to 8 textures
  - Second pass maps textures to a quad in screenspace
    - Fragment shaders process textures

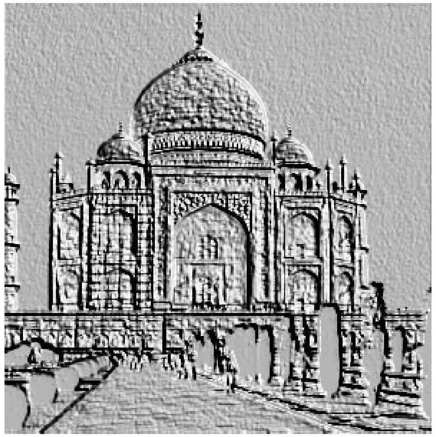
Standard Image Filters

- Image is filtered with 3x3 kernel:
  - Weighted texture lookups in adjacent texels
  - Edge detection through laplacian:

0	1	0
1	-4	1
0	1	0

  - Emboss filter:

2	0	0
0	-1	0
0	0	-1



Gaussian Filter

- Many effects based on gaussian filter
- 5x5 gaussian filter requires 25 texture lookups:

1	4	6	4	1
4	16	26	16	4
6	26	41	26	6
4	16	26	16	4
1	4	6	4	1

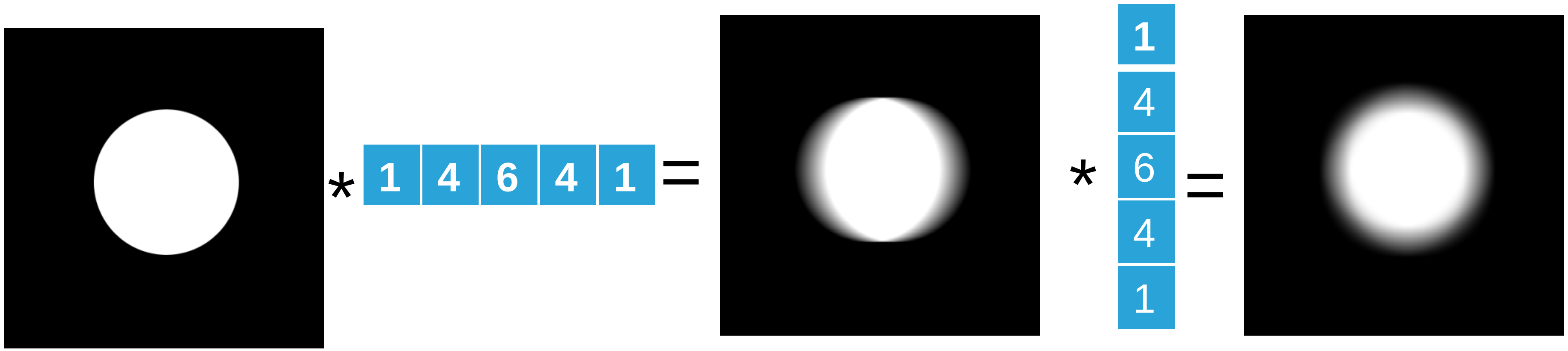
* 1/256

  - Too slow and too expensive

- But: Gauss is separable!

Gaussian Filter

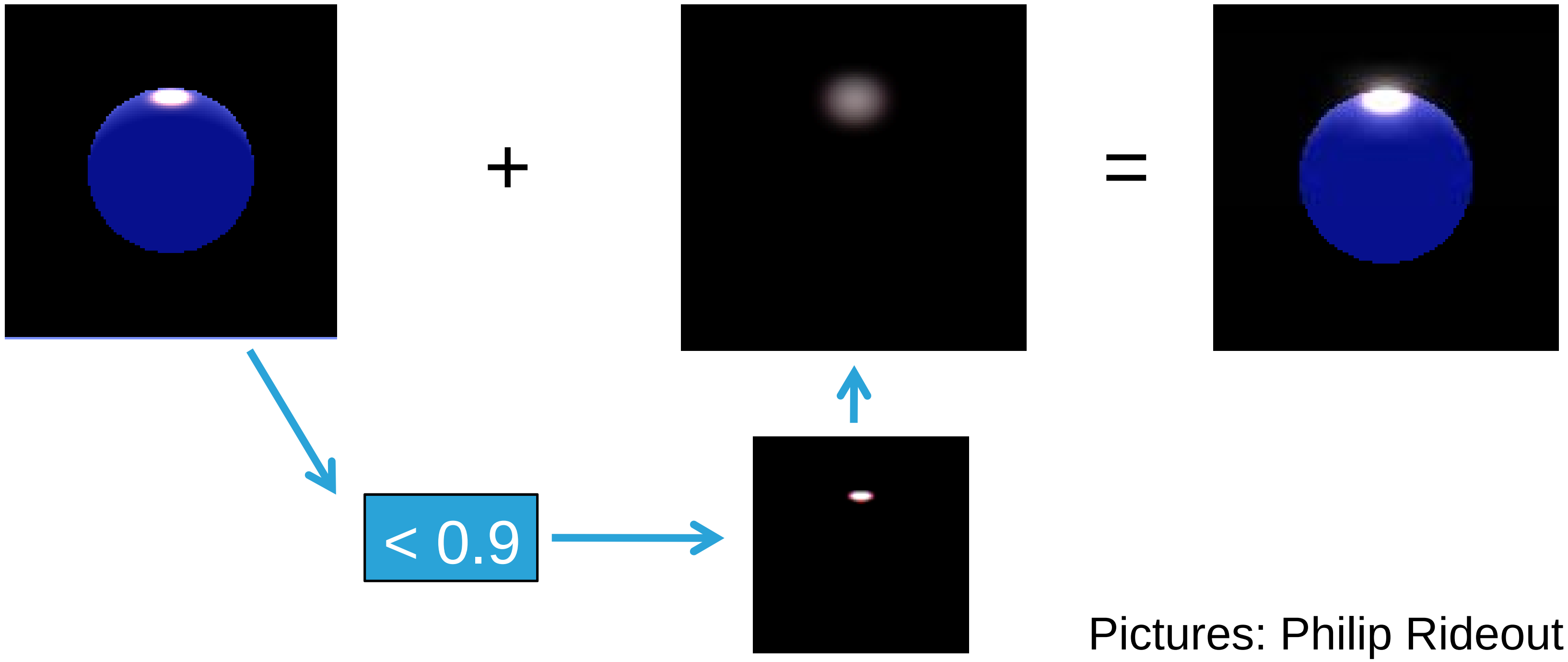
- Separate 5x5 filter into 2 passes
- Perform 5x1 filter in u
- Followed by 1x5 filter in v



- Lookups can be formulated to use linear filtering
  - 5x1 filter with 3 lookups

Bloom

- Modify rendered texture intensities before gaussian filtering
  - Clamp or glowing object only pass
  - Exponential weight
- Add filtered image to original image



Pictures: Philip Rideout

Bloom

- Bloom usually applied to downsampled render textures
  - 2x or 4x downsampled
    - Effectively increases kernel size
  - But: Sharp highlights are lost
  - Combination of differently downsampled and filtered render textures possible
    - Allows high controllability of bloom

- Filter in *u* and *v* and separate addition leads to star effect

Bloom

Picture: Oblivion

Bloom remarks

- Disguises aliasing artifacts
- Works best for shiny materials and sun/sky
  - Only render sun and sky to blur pass
  - Only render specular term to blur pass
- A little bit overused these days
  - Use sparsely for most effect
- Can smudge out a scene too much
  - Contrast and sharp features are lost (fairytale look)

Bloom remarks

- Extreme example

Picture: Zelda Twilight Princess

Motion Blur

- Keep previous frames as textures
  - Blend weighted frames to final result

- Calculate camera space speed of each pixel or object in texture
- Blur along motion vector
  - Harder to implement, but looks very good
  - Faster than blending

Motion Blur Example

Picture: Crysis (Object Based Motion Blur)

Other filters

- Use precomputed noise maps
  - Modulate Color with noise:
    - TV snow emulation
  - Modulate texture coordinates:
    - glass refractions
    - TV distortions
    - Warping
  - Remap intensity:
    - Heat vision
    - Eye adaptation

Demo

OGRE Demo

HDR Rendering

- Up to now, parameters are chosen so that the result is [0..1]
- Real world:
  - Dynamic Range is about 1:100 000
    - 1: dark at night
    - 100 000: direct sunlight
  - Eye adapts to light intensities
- Current hardware allows to calculate everything in floating point precision and range
  - Use lights/environment maps with intensities of high dynamic range

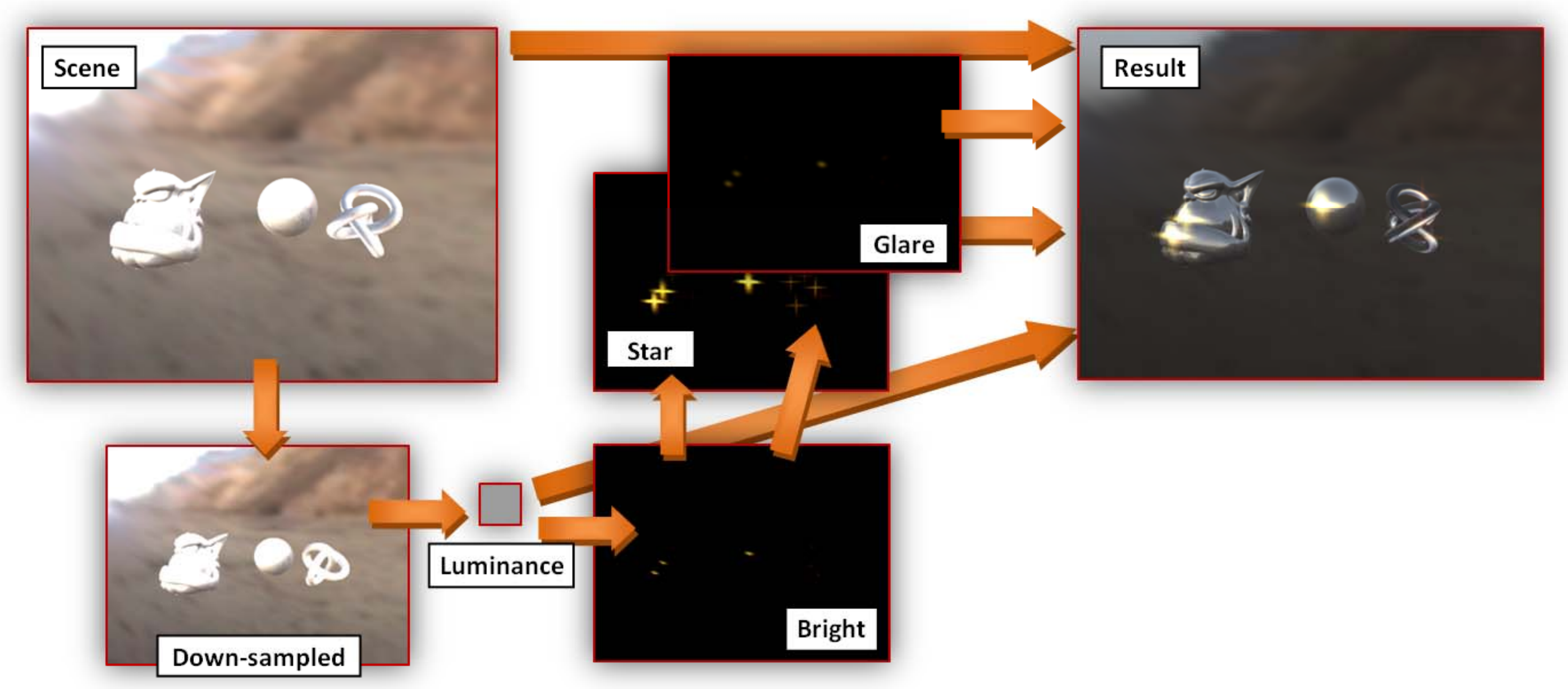
HDR rendering

- But: we cannot display a HDR image!
- Solution: Remap HDR intensities to low dynamic range:
- **Tone mapping**
  - Imitates human perception
  - Can mimic time delayed eye adaptation
  - Can mimic color desaturation
  - Can imitate photographic effects
    - Over exposure
    - Glares

HDR Rendering

- Tone mapping requires information about the intensities of the HDR image
  - Extract average/maximum luminance through downsampling
    - Hardware MIPmap generation
    - Or through a series of fragment shaders
- Naturally combines with bloom filter

HDR Processing Overview



Picture: Christian Luksch

Tone mapping Operators (1)

- Reinhard's operator

$$L_{scaled} = \frac{a \cdot L_W}{\bar{L}_W}$$

a … Key
$\bar{L}_W$ … Average luminance
L_W … Pixel luminace

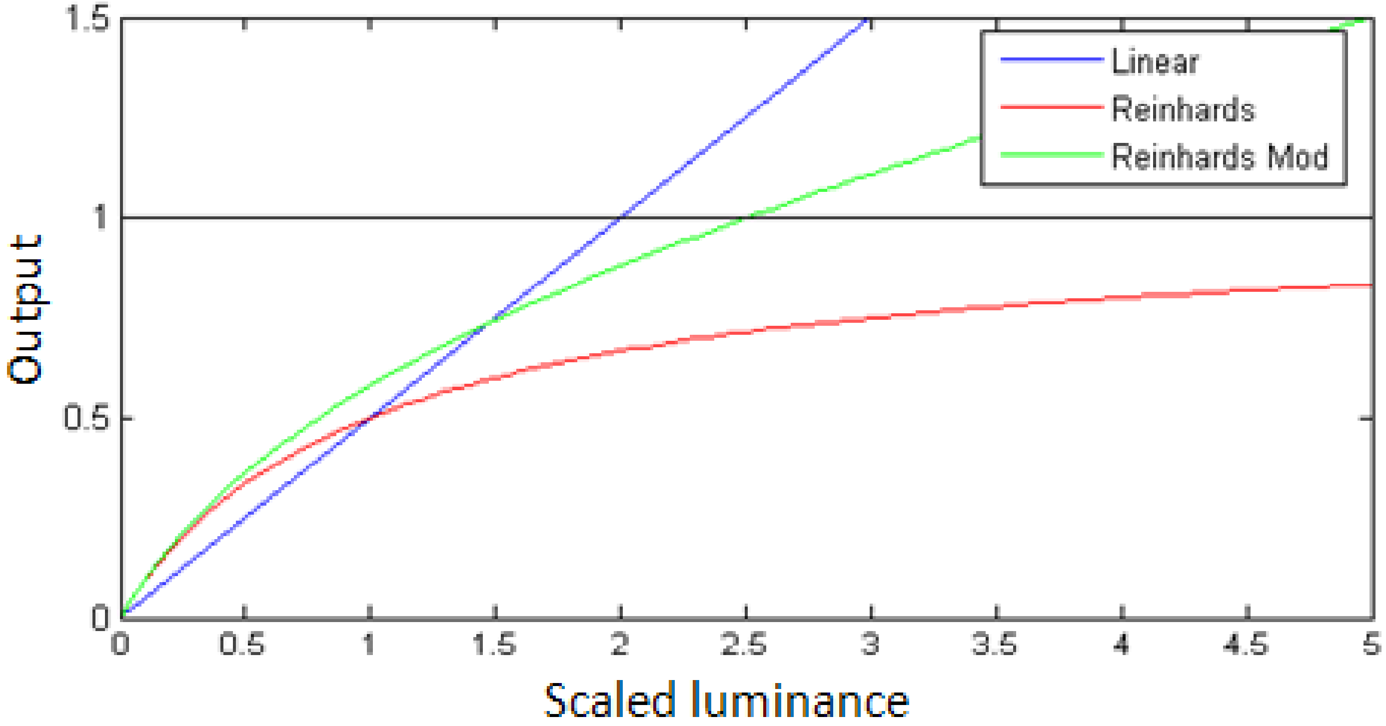
- Original $$Color = \frac{L_{scaled}}{1 + L_{scaled}}$$

- Modified $$Color = \frac{L_{scaled} \cdot \left(1 + \frac{L_{scaled}}{L_{white}^2}\right)}{1 + L_{scaled}}$$

  - Key *a is* set by user or some predefined curve $a(l_a)$ dependent on average luminance l_a
  - Calculations need to be done in linear color space! (floating point buffers, see perception issues)

Tone mapping Operators (2)

- Reinhard’s operator



Picture: Christian Luksch

Tone mapping Operators (3)

- Logarithmic mapping

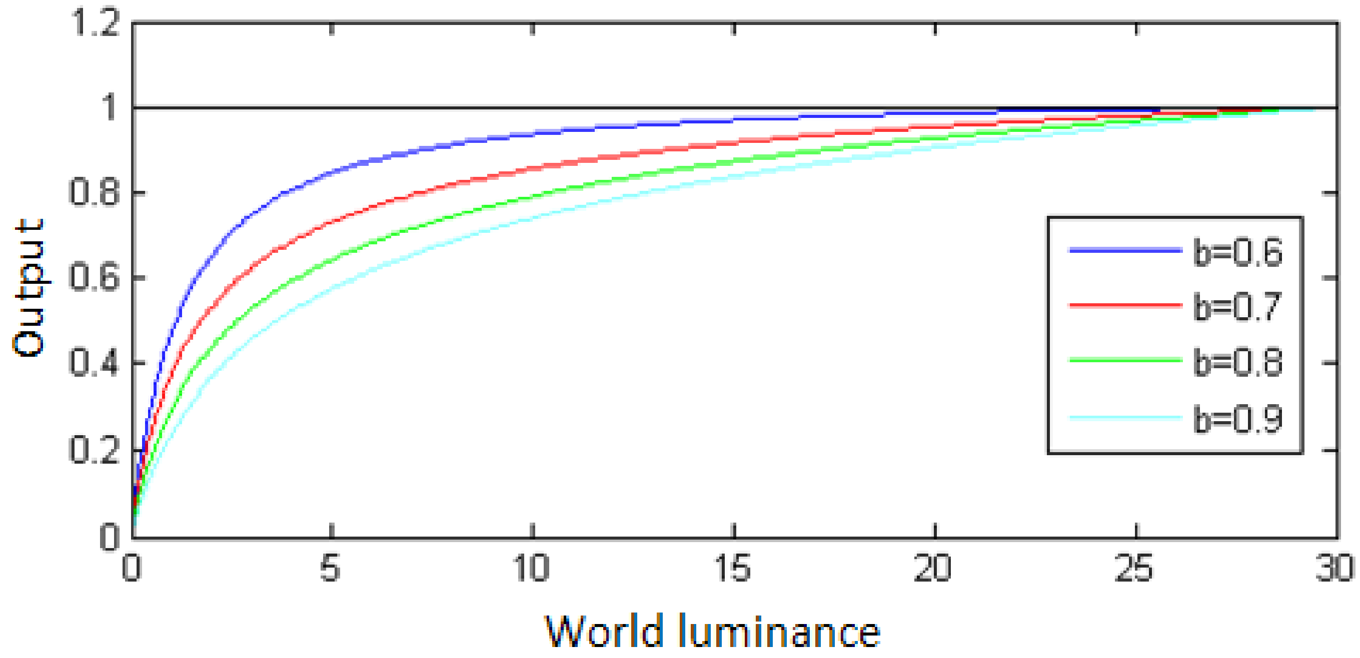
$$L_d = \frac{\log_x(L_w + 1)}{\log_x(L_{max} + 1)}$$

  - Improvement: Adaptive logarithmic mapping
  - L_{max} causes heavy changes of the output color when moving through the scene

    → Modifications necessary

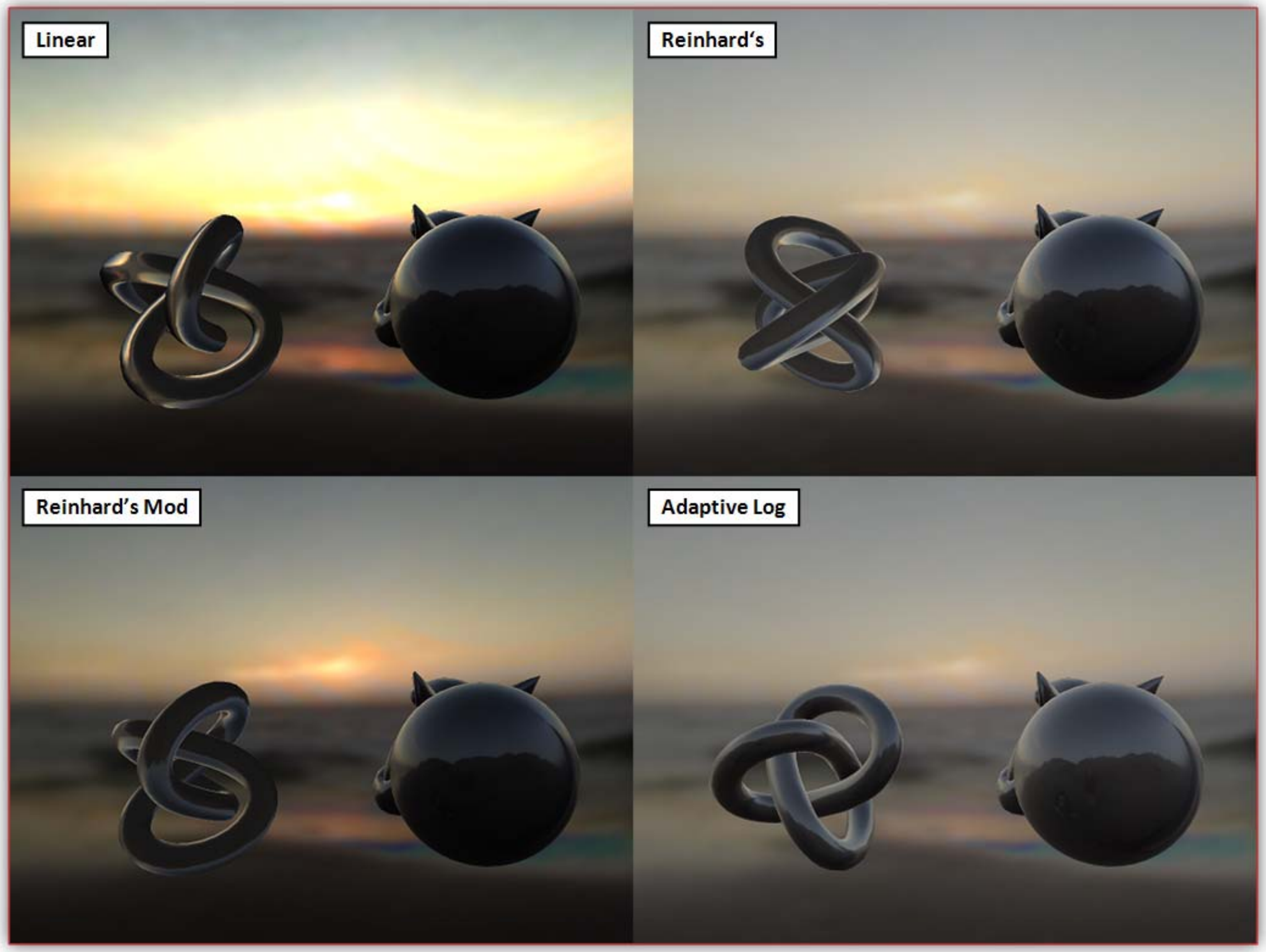
Tone mapping Operators (4)

- Adaptive logarithmic mapping: [Drago 03]



Picture: Christian Luksch

Comparison



OGRE Beach Demo

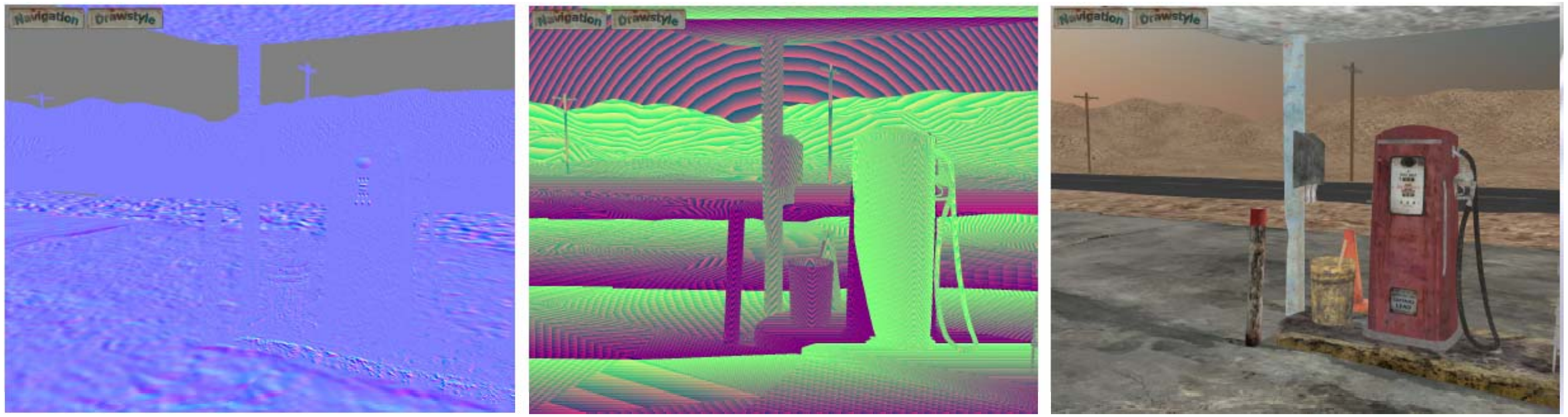
(this time HDR part)

Author: Christian Luksch

http://www.ogre3d.org/wiki/index.php/HDRlib

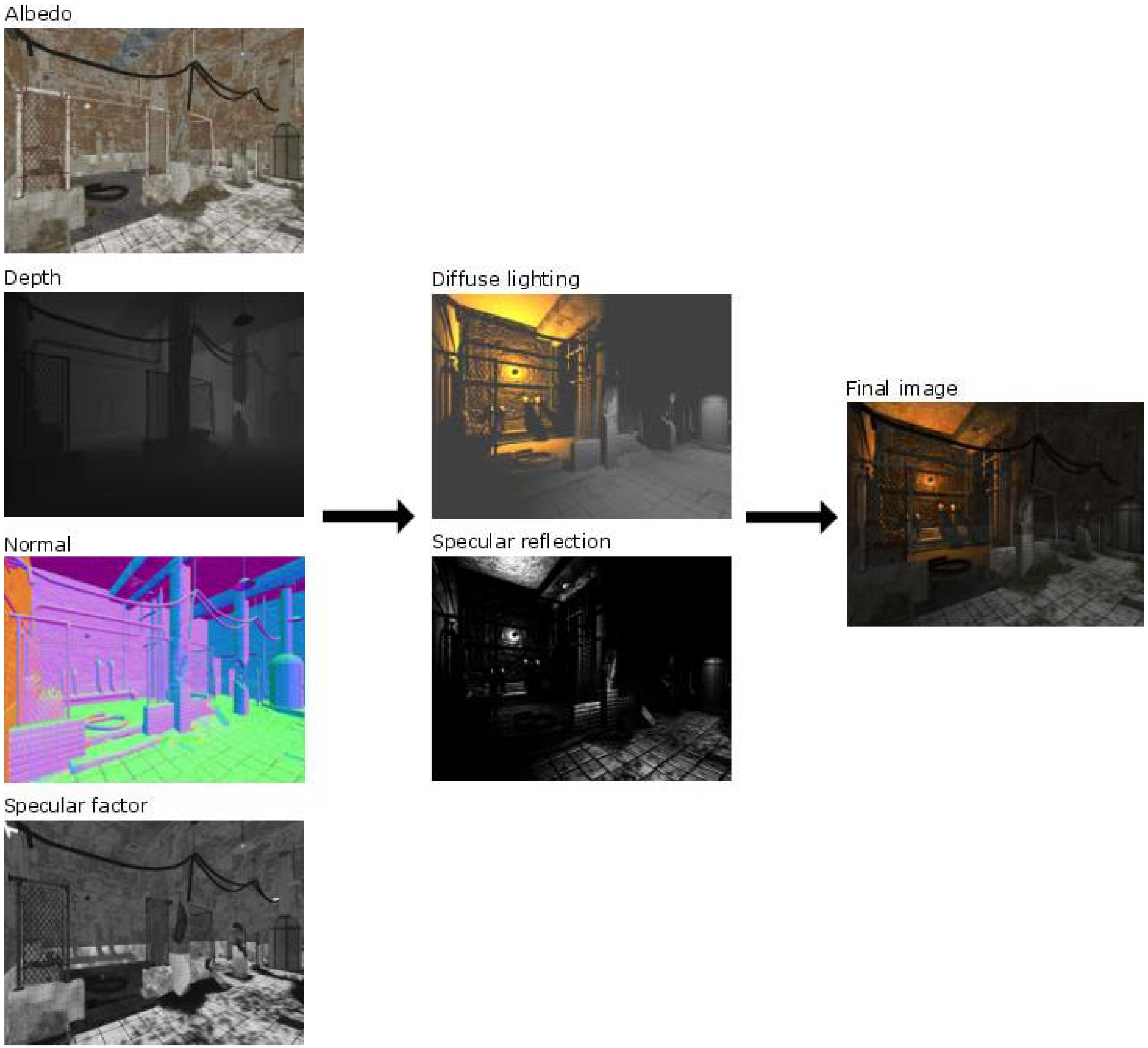
Deferred Shading

- General Idea: Treat lighting as a 2D postprocess
- Deferred Shading rendered textures:
  - Normals
  - Position
  - Diffuse color
  - Material parameters
- Execute lighting calculations using the textures as input



Picture: NVIDIA

Deferred Shading



Picture: Leadwerks

Deferred Shading

Picture: S.T.A.L.K.E.R.

Deferred Shading

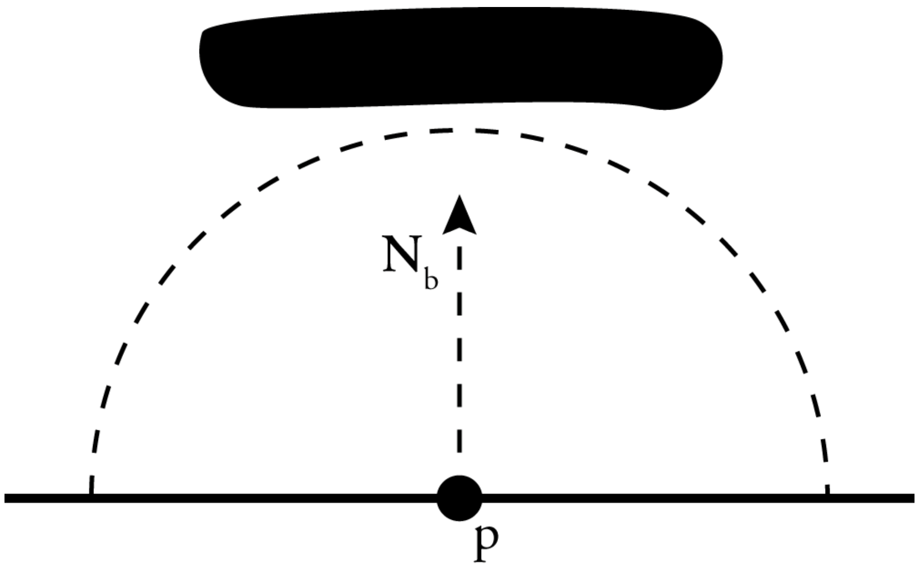
- Pros:
  - Perfect batching (no object dependence)
  - Many small lights are just as cheap a a few big ones (32 lights and up are no problem)
  - Combines well with screenspace effects
- Cons:
  - High bandwidth required
    - Not applicable on older hardware
  - Alpha blending hard to achieve
  - Hardware multisampling not available

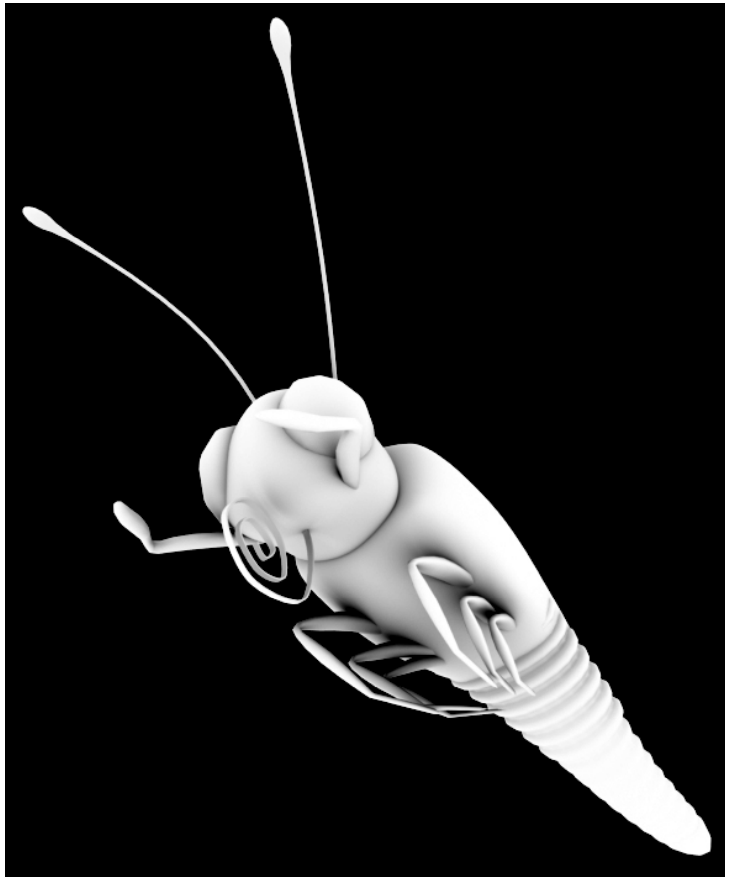
Deferred Shading

- Cons are diminishing on current hardware
  - Hardware features assist deferred shading (sample buffers)
  - High bandwidth and lots of RAM available
- Many state-of-the-art engines feature deferred shading
- Allows to approximate GI with high number of lights (including negative lights).

Ambient Occlusion (AO)

- Calculates the occlusion of each surface point to the surrounding.
  - No information of the surrounding is used





Screen Space Ambient Occlusion (SSAO)

- Newest hype in real-time graphics
- Popularized by Crysis (Crytek)
- Render textures needed:
  - Depth (as linear z-buffer) or world space position
  - Normals
- Approach:
  - Fragment analyses its surrounding
    - Fragment samples z-buffer around screen position to find occluders in surrounding
    - Simplest approach: depth difference of fragment and sample

Screen Space Ambient Occlusion (SSAO)

- Pros:
  - Independent from scene complexity
  - No preprocessing
  - Dynamic scenes
- Cons:
  - Not correct
  - Only evaluates what is seen
  - Only close range shadowing
  - Sampling artifacts (needs additional smoothing/blur)
- But noone cares about correctness in realtime graphics
- Very powerful method!

OGRE SSAO Demo

Screen Space Ambient Occlusion (SSAO)

- Many variations are available, differing in correctness/speed/filtering.
- Can be extended to include approximations of global illumination or image based lighting (Ritschel et al. 2009)