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:
    
    \[
    \begin{array}{ccc}
    0 & 1 & 0 \\
    1 & -4 & 1 \\
    0 & 1 & 0 \\
    \end{array}
    \]

- Emboss filter:
  
  \[
  \begin{array}{ccc}
  2 & 0 & 0 \\
  0 & -1 & 0 \\
  0 & 0 & -1 \\
  \end{array}
  \]
Gaussian Filter

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

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>26</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>26</td>
<td>41</td>
<td>26</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>26</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

- 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 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
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 (faireytale 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
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
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

- Scene
- Down-sampled
- Luminance
- Star
- Glare
- Bright
- Result
Reinhard’s operator

\[ L_{scaled} = \frac{a \cdot L_w}{\bar{L}_w} \]

- **Key** \( a \)
- **Average luminance** \( \bar{L}_w \)
- **Pixel luminance** \( L_w \)

**Original**

\[ Color = \frac{L_{scaled}}{1 + L_{scaled}} \]

**Modified**

\[ Color = \frac{L_{scaled} \cdot \left(1 + \frac{L_{scaled}}{L_{white}}\right)}{1 + L_{scaled}} \]

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)
Reinhard’s operator
Logarithmic mapping

\[ L_d = \frac{\log_x (L_w + 1)}{\log_x (L_{\text{max}} + 1)} \]

Improvement: Adaptive logarithmic mapping

\( L_{\text{max}} \) causes heavy changes of the output color when moving through the scene

\( \rightarrow \) Modifications necessary
Adaptive logarithmic mapping: [Drago 03]
Comparison

Linear

Reinhard’s

Reinhard’s Mod

Adaptive Log
OGRE Beach Demo
(this time HDR part)

Author: Christian Luksch

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

Albedo

Depth

Diffuse lighting

Specular reflection

Normal

Final image
Picture: S.T.A.L.K.E.R.
Deferred Shading

Pros:
- Perfect batching (no object dependence)
- Many small lights are just as cheap as 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 analyzes 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!

Vienna University of Technology
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)