High Dynamic Range

Allowing for greater colour ranges

Programming – Computer Graphics

Contents

- Low Dynamic Range
- High Dynamic Range
- Tone Mapping
- Bloom

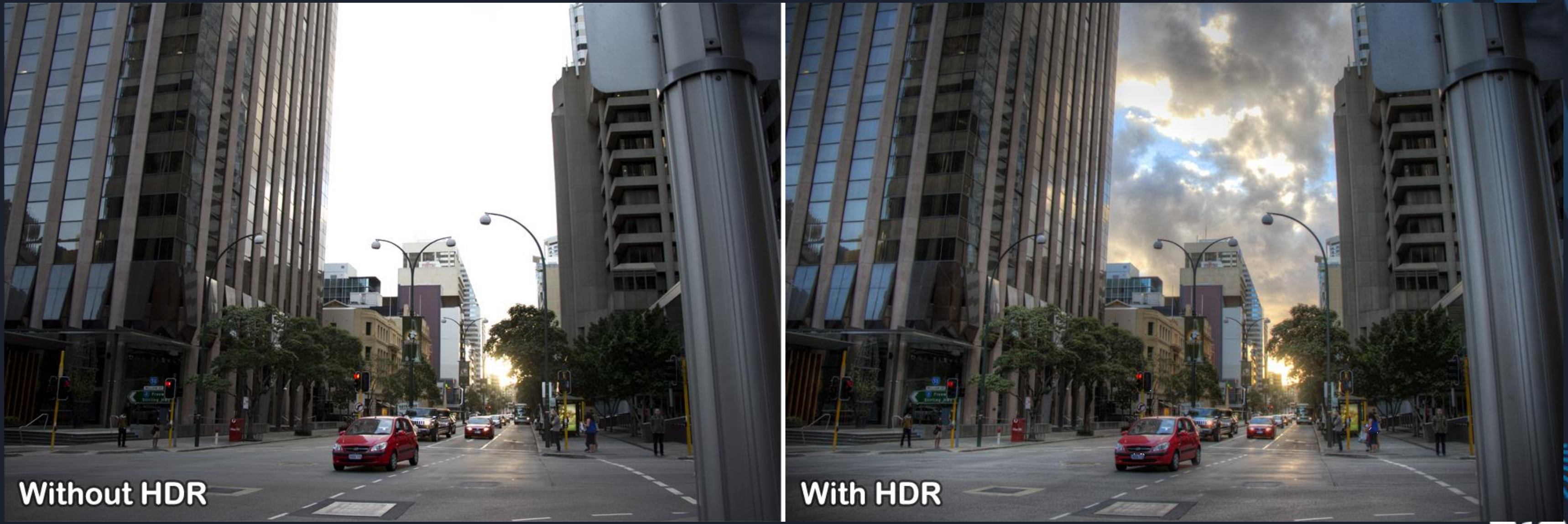
Low Dynamic Range

- Traditionally virtual colour is represented by red, green and blue colour channels, each using 8-bits of precision
  - A total of 256 shades for each colour channel
  - This gives us 16,777,216 different colour values, which is actually low!
  - Gamma correction and contrast ratios on computer monitors also mean dark colours usually can’t be displayed correctly
  - This is called a Low Dynamic Range (LDR)

- In reality light has a far greater range
  - Just because a colour cannot be observed within a dark area doesn’t mean that colour isn’t there
  - Standing in a bright area looking into a dark room you may not see much in the room, but if you were in the dark room you would see the details inside it but have a hard time seeing out into the bright area

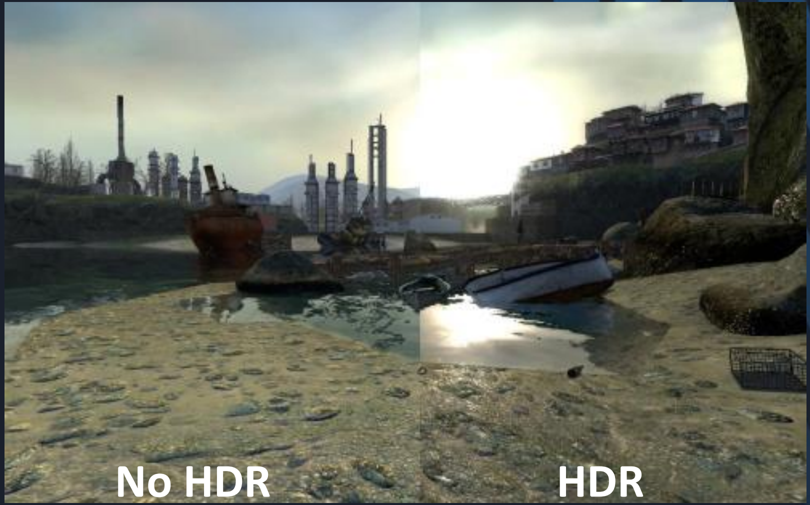
High Dynamic Range

- High Dynamic Range (HDR) represents lighting ranges that may not usually be seen but do still exist when the exposure is adjusted



High Dynamic Range

- In computer graphics HDR can be achieved by using Post-Processing
  - Colour is stored in higher precision values, such as a 32-bit float per-channel rather than 8-bit for the render target's buffers
  - Colour and lighting values are allowed to far exceed the 0-to-1 range
  - Using Tone Mapping to adjust the HDR value to be within a LDR range that a computer monitor can display correctly

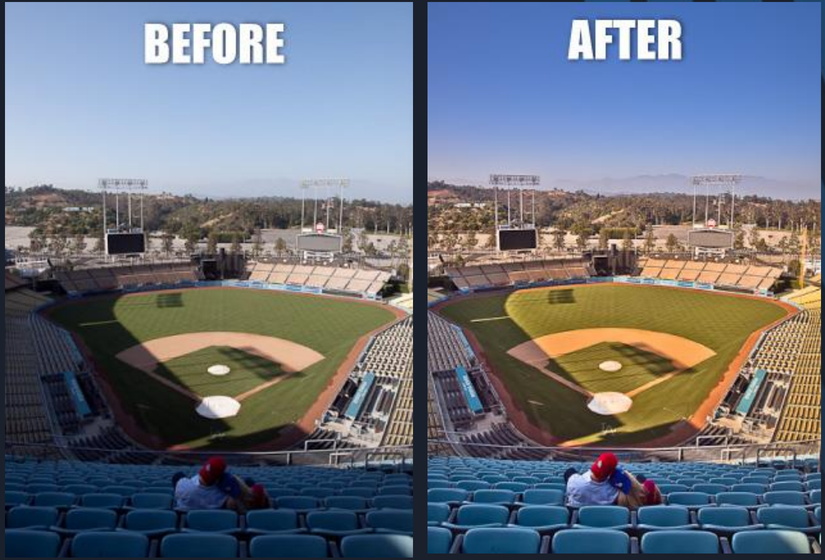


- Tone Mapping operators decide the final pixel colour

Tone Mapping

- Tone Mapping is a method for adjusting the colour ranges within an image using a function called an operator
  - It commonly uses an Exposure level and a mathematical curve to modify the pixel colours
  - Exposure can be calculated locally on each pixel based on its luminance, or globally based on the luminance of the whole image, or set as a constant

- Different Tone Mapping operators work in different ways
  - Each have different curves that control the precision given to darker or brighter colours

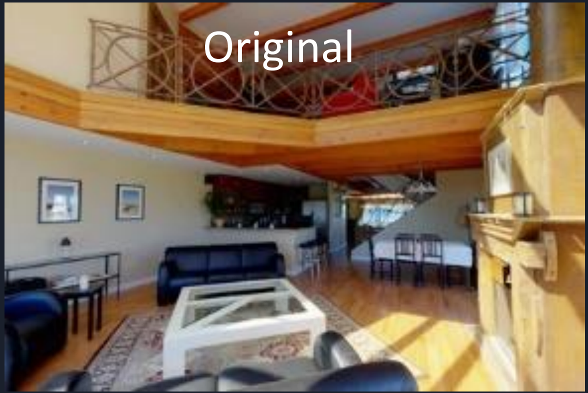


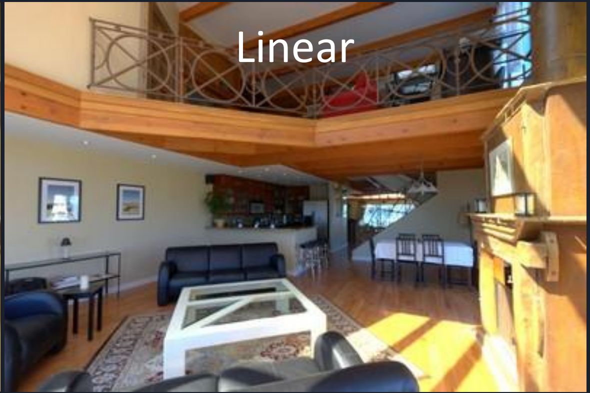
Tone Mapping

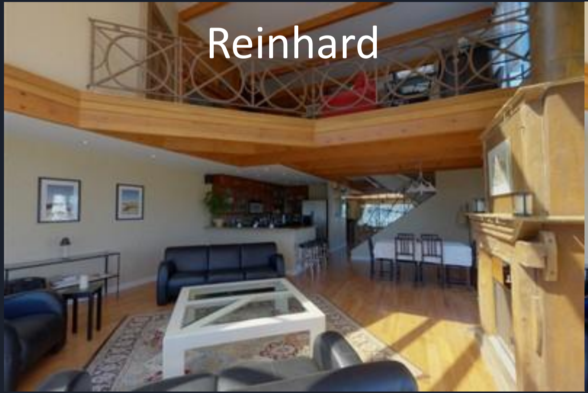
- Tone Mapping also tries to adjust for a monitor’s gamma and contrast ratio

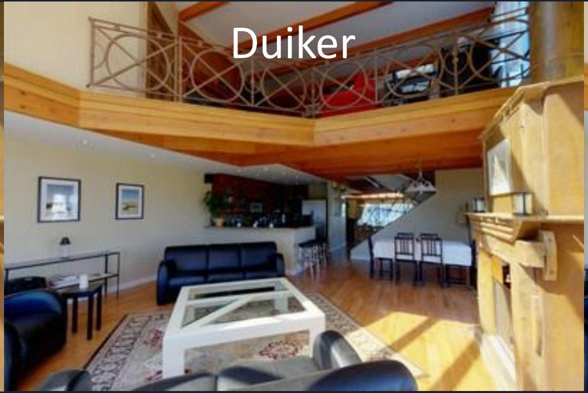
- There are many common Tone Mapping operators
  - Linear
  - Reinhard
  - Duiker
  - Hejl & Burgess-Dawson

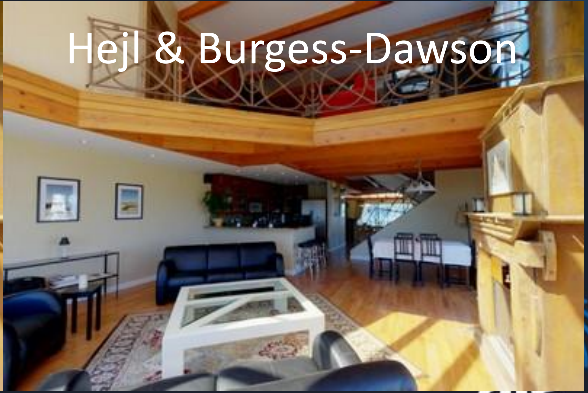
Tone Mapping











Tone Mapping Example - Linear

- The following is an example of the Linear Tone Map that applies an exposure adjustment and then adjusts for the monitors gamma
  - Gamma Correction

```
#version 440

in vec2 texCoord;

uniform sampler2D fullscreenImage;
uniform float exposure;

out vec4 fragColour;

void main() {
    // sample image
    vec3 colour = texture( fullscreenImage, texCoord ).rgb;
    // apply exposure
    colour *= exposure;
    // adjust for monitor gamma and output
    fragColour = vec4( pow( colour, 1 / 2.2f ), 1 );
}
```

Tone Mapping Example - Reinhard

- Reinhard was a common implementation that added an extra step

```
#version 440

in vec2 texCoord;

uniform sampler2D fullscreenImage;
uniform float exposure;

out vec4 fragColour;

void main() {
    // sample image
    vec3 colour = texture( fullscreenImage, texCoord ).rgb;
    // apply exposure
    colour *= exposure;
    colour = colour / (1 + colour);
    // adjust for monitor gamma and output
    fragColour = vec4( pow( colour, 1 / 2.2f ), 1 );
}
```

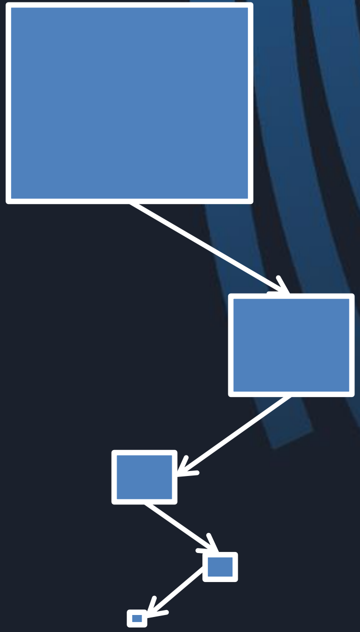
Tone Mapping Example – Hejl & Burgess-Dawson

- Another technique, but this time does not require Gamma Correction!

```
#version 440

in vec2 texCoord;

uniform sampler2D fullscreenImage;
uniform float exposure;

out vec4 fragColour;

void main() {
    // sample image
    vec3 colour = texture( fullscreenImage, texCoord ).rgb;
    // apply exposure
    colour *= exposure;

    // Hejl & Burgess-Dawson
    vec3 x = max( 0, colour - 0.004f );
    colour = (x * (6.2f * x + 0.5f)) / (x * (6.2f * x + 1.7f) + 0.06f);

    // does not require gamma adjustment!
    fragColour = vec4( colour, 1 );
}
```

Dynamic Exposure & Down-Sampling

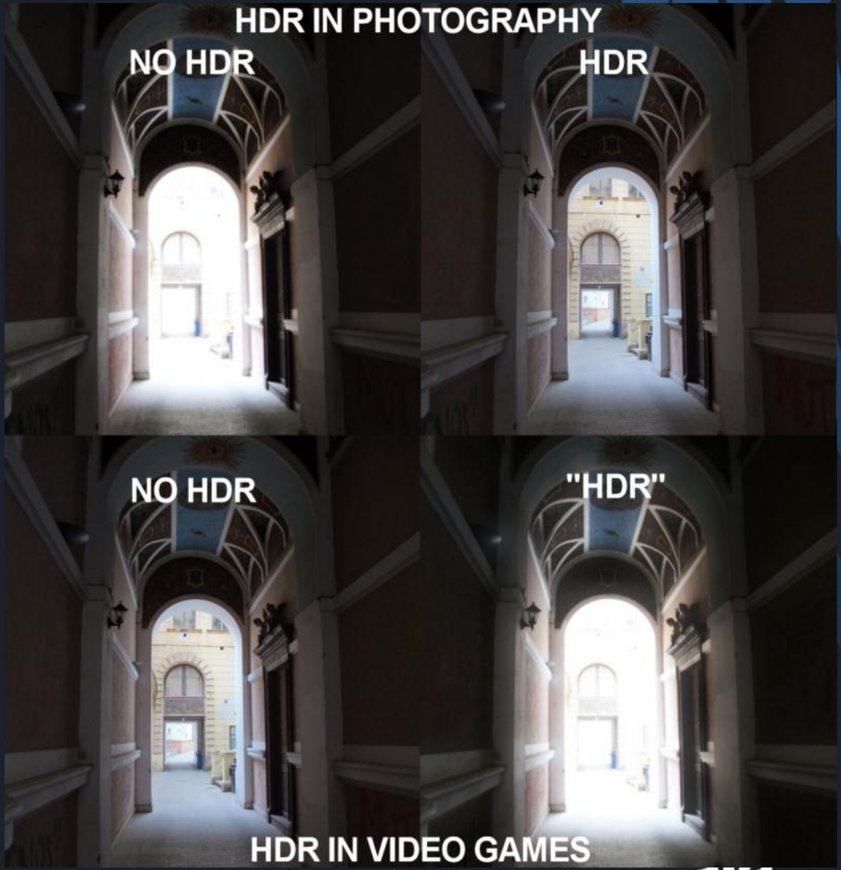
- Exposure depends on the brightness of the scene
- One way to dynamically calculate exposure in real time is:
  - Render the scene in HDR to a Render Target
  - Repeatedly down-sample the Render Target, halving its size each time and using filtering to blur the pixels together, until we have a Render Target with a single pixel
  - Calculate the luminance of this single pixel and use it as the exposure for all pixels
- This can be extremely slow
  - Can be done over multiple frames to increase performance or use compute shaders / GPGPU techniques



Real-Time HDR

- HDR is meant to allow us to see a greater range of colours
  - Contrast and exposure is adjusted to reveal colours that could not easily be seen

- In video games however typically HDR is confused with brightening a scene and making it bloom
  - This is incorrect…



Bloom

- Bloom is the bleeding of light from a light area into a darker area
- Only the bright pixels are written into a Render Target, then blurred, and finally added on top of the original image
  - A Gaussian Blur is commonly used for pleasing results
- A pixel's brightness, Luminance, can be calculated by performing a dot product between the pixel colour and the value (0.2125, 0.7154, 0.0721)

```
float luminance = dot( colour, vec3(0.2125f, 0.7154f, 0.0721f) );
```

Summary

- High Dynamic Range allows us to work with colour values that exceed the traditional 0.0-to-1.0 range
- Adjusting a pixel’s exposure and using a Tone Mapping operator allows us to see colours that we typically would not be able to see because of the lack of or too much light in a scene
- Real-time HDR commonly mistakes Bloom for HDR
  - Bloom can give an interesting visual flare to our scenes
  - Bloom is achieved typically through the use of a Gaussian Blur

Further Reading

- Wolff, D, 2013, *OpenGL 4 Shading Language Cookbook*, 2nd Edition, PACKT Publising
- Akenine-Möller, T, Haines, E & Hoffman, N, 2008, *Real-Time Rendering*, 3rd Edition, CRC Press
- Haemel, N, Sellers, G & Wright, R, 2014, *OpenGL SuperBible*, 6th Edition, Addison Wesley