Post-processing

Full-screen effects

Programming – Computer Graphics

Contents

- What is Post-Processing?
- Kernels
- Various effects
  - Sharpen
  - Edge Detection
  - Blurs
  - Depth of Field
  - Radial Blur
  - Motion Blur

What is Post-Processing?

- The most common use of Render Targets is Post-Processing
  - Post-Processing usually consists of drawing the entire scene to a off-screen Frame Buffer
  - Once complete, a final render call is made, using the Back Buffer as the Render Target, drawing a single full-screen quad with the Render Target applied to it as a texture

- Doing this gives us access to all the pixels around the current pixel being drawn, via texture sampling
  - Knowing information about surrounding pixels gives us the ability to:
    - Blur
    - Sharpen
    - Detect edges
    - Completely change the scene's hue
    - Distort
    - Many other full-screen effects

Kernel Filters

- When sampling an image we usually sample a single texel
- We can of course sample more than one, and with Post-Processing we have access to all the “visible” pixels that were rendered to a Render Target
- One way to sample multiple texels is with a kernel filter
  - A kernel filter works by using a sampling matrix
  - The sampling matrix has weighting values that are multiplied by sampled texels around the current texel, and the current pixel is the sum of those weighted texels
- Let’s elaborate…

Kernels

The current texel

0	-2 / 3	0
-2 / 3	11 / 3	-2 / 3
0	-2 / 3	0

- The following is an example 3x3 sharpen kernel
  - Sharpen consists of adjusting the contrast around pixels
- When sampling texels in an image at pixel (x,y) we would multiply it by 11 / 3, then:
  - Sample texel (x+1, y) and multiply it by -2 / 3, adding to the previous sample
  - Sample texel (x-1, y) and multiply it by -2 / 3, adding to the previous sample
  - Sample texel (x, y+1) and multiply it by -2 / 3, adding to the previous sample
  - Sample texel (x, y-1) and multiply it by -2 / 3, adding to the previous sample
- And that would be how we use a simple kernel!

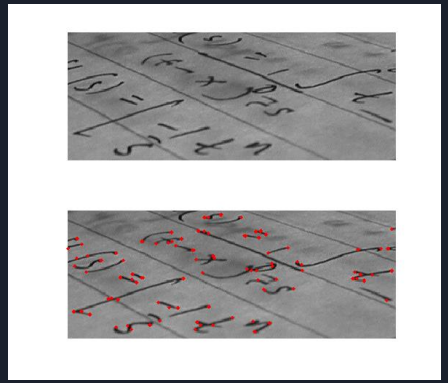
```
#version 410
// fragment shader example
in vec2 texCoord;
out vec4 fragColour;

uniform sampler2D screenTexture;

void main( ) {
     vec2 texelSize = 1.0f / textureSize( screenTexture, 0 ).xy;

     fragColour = texture( screenTexture, texCoord ) * (11 / 3);
     fragColour += texture( screenTexture, texCoord + vec2( 0, texelSize.y ) ) * (-2 / 3);
     fragColour += texture( screenTexture, texCoord - vec2( 0, texelSize.y ) ) * (-2 / 3);
     fragColour += texture( screenTexture, texCoord + vec2( texelSize.x, 0 ) ) * (-2 / 3);
     fragColour += texture( screenTexture, texCoord - vec2( texelSize.x, 0 ) ) * (-2 / 3);
     fragColour.a = 1.0f;
}
```

Sharpen

Original Sharpened

Edge Detection

- Edge Detection is the term given to finding changes in an image
  - Usually colour or brightness / contrast changes
  - In games we also can detect changes in depth, surface normals, or other chosen properties
- Edges can add interesting visual flare to games
- There are various Kernels for detecting colour and brightness changes
  - For normals and depth we can usually check surrounding texels for changes in normal direction / depth distance
- We can also detect corners with some algorithms, useful for shape recognition



Edge Detection in Games

Sobel Operator

- One common Kernel used for Edge Detection with colour is Sobel
  - It detects gradients in the image
- It is made by two kernels that work together, detecting gradients in each axis

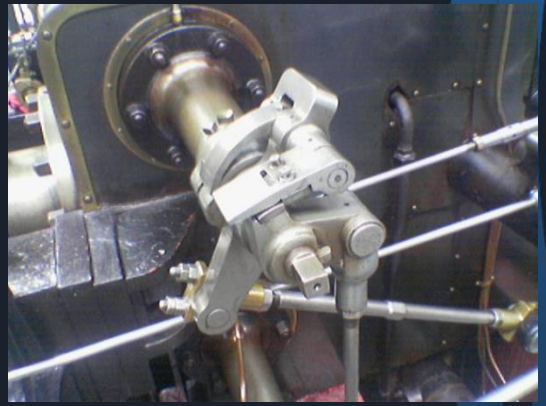
-1	0	1
-2	0	2
-1	0	1

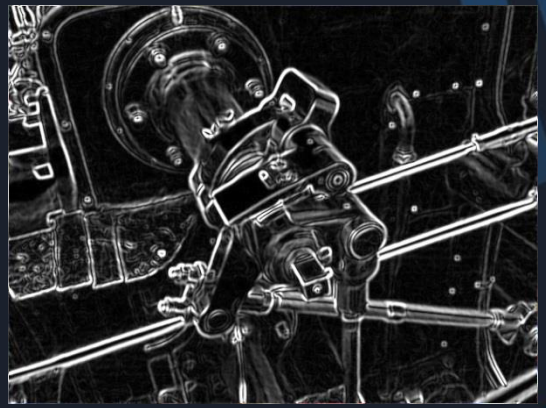
X

-1	-2	-1
0	0	0
1	2	1

Y

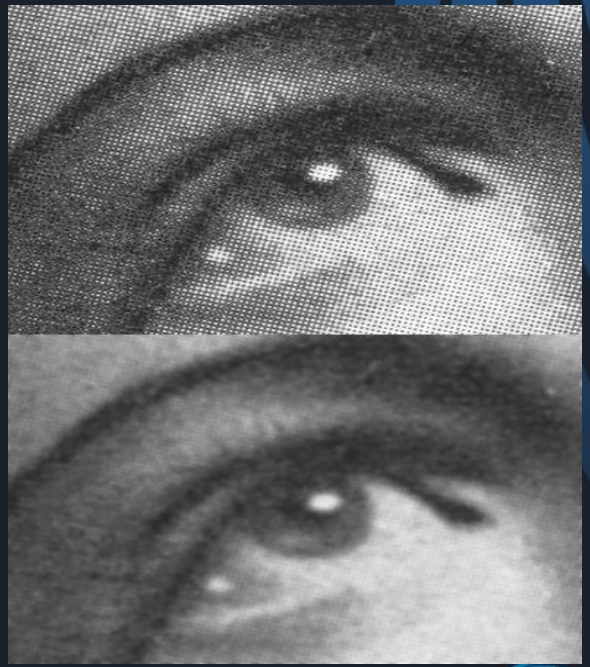
$$pixel = \sqrt{X^2 + Y^2}$$





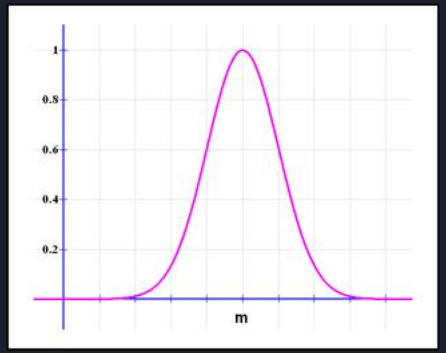
Blurs

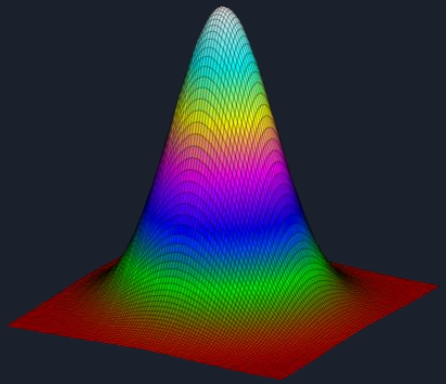
- A simple use of Kernels is basic blurring
- To blur we can sample multiple texels and give each texel in the kernel a sample weight, combing them together based on their weights to “blur” the pixel colours together:
  - Box Blur is a simple kernel where the 4 neighbouring pixels (up, down, left, right) are sampled and averaged to determine the current pixel’s colour
    - Can cause “boxy” artefacts
  - Gaussian Blur is a popular method due to its lack of boxy artefacts



Gaussian Blur

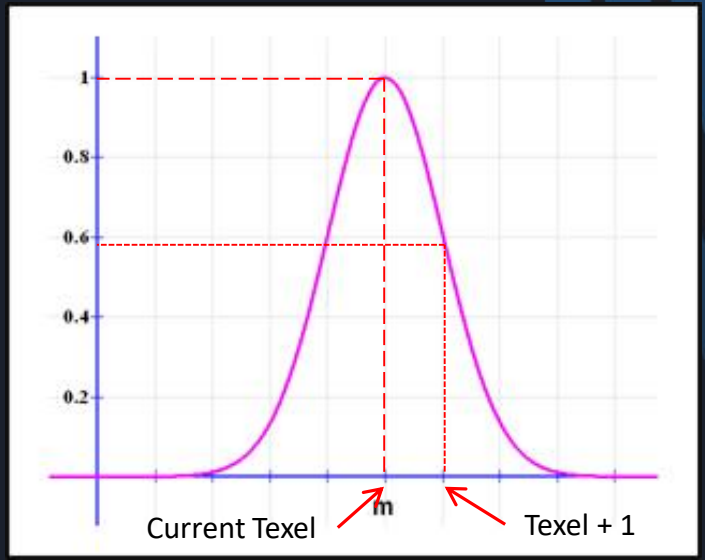
- Gaussian Blurring is a technique that smooths an image using a Gaussian Function
  - Gaussian Functions represent a Bell Curve
  - Can have various sized radius
- Gaussian creates a smooth even blur that reduces noise
  - Useful for reducing noise in an image before applying Edge Detection





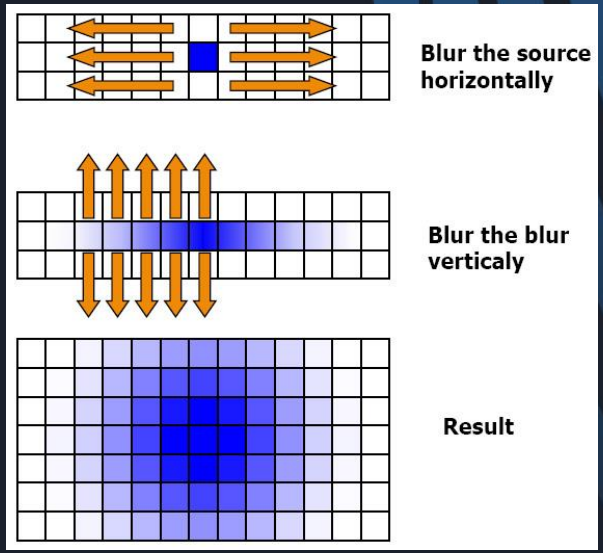
Gaussian Blur

- In essence a Gaussian works as follows
  - The vertical axis represents weighted values
  - The horizontal axis represents texels to sample
  - In the example the current texel represents the peak of the curve
  - The texel is sampled and multiplied by its matching weighted value
  - Surrounding texels are sampled and multiplied by their weight
  - The texels are all combined to create the blurred texel



Kernel Size Problems

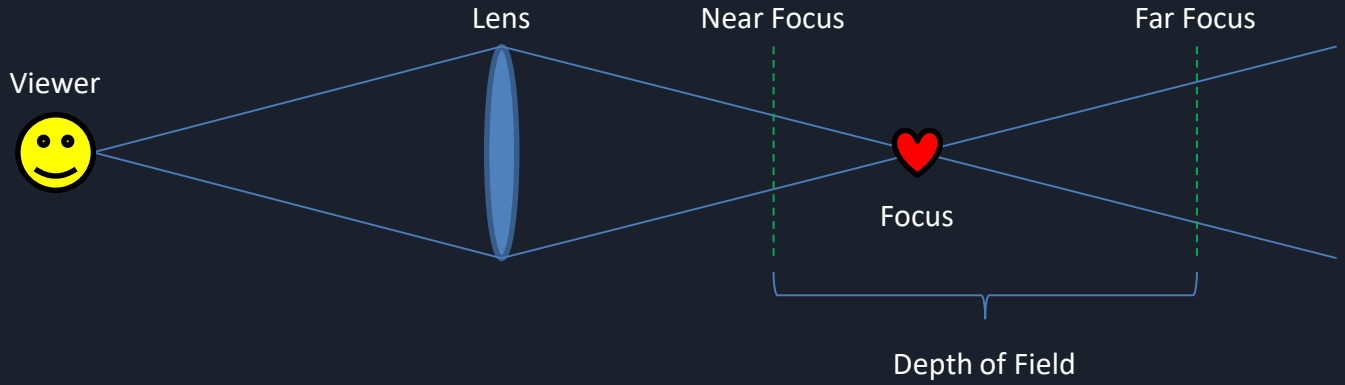
- Larger Kernels give a better blur but this requires many samples per texel
  - A 9x9 Kernel on a 1920x1080 screen requires 167,961,600 texture samples, 81 per texel!
- Gaussian Blurs are able to be separated into separate axis
  - Blur the image horizontally or vertically first
  - Blur the blurred image in the other axis
- The resulting image is the same but instead of 9x9 samples per texel it is now 9+9, 18 instead of 81!



Depth of Field

- Depth of Field is a camera blur that is caused by lens and aperture size, and distance of the viewer to the lens and from the lens to the target object
  - Virtual cameras typically represent a Pin-hole Camera and do not have a lens, so do not have a Depth of Field

- The Depth of Field is the distance from the closest in-focus object to the farthest in-focus object
  - There is generally less focal distance between the near focus and the in-focus object

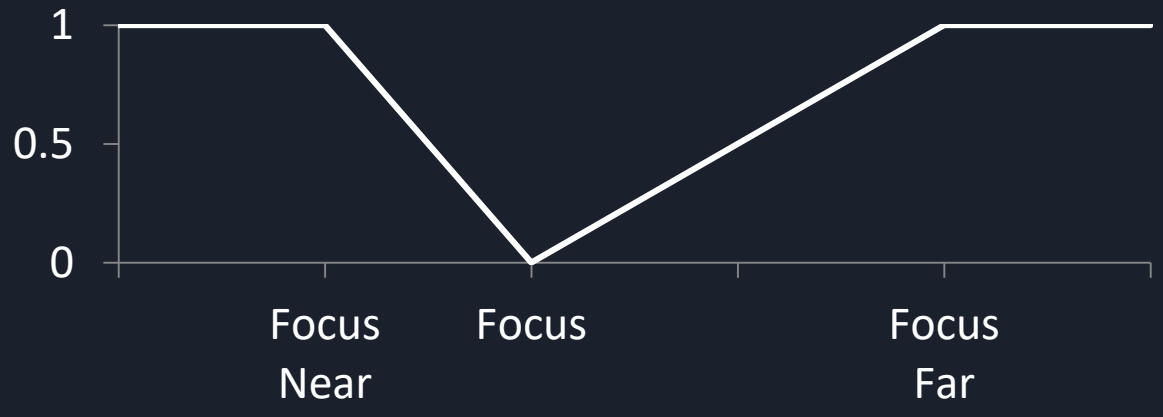


Depth of Field

Depth of Field in Games

- We can easily mimic the Depth of Field if we know the depth of the scene
- One simple way is to use 3 tweakable variables and calculate the blur strength at any given pixel
  - Focus Depth, Near Focus and Far Focus
- We can use this strength value to interpolate between a non-blurred version of the scene and a blurred version of the scene
  - The blurred version can be calculated using a blur kernel, such as a Gaussian

Depth of Field in Games

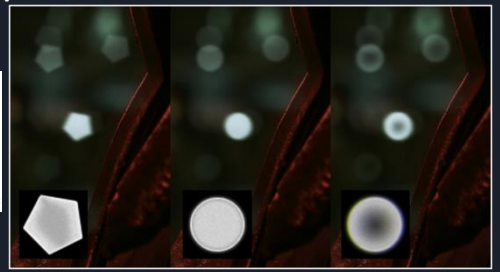



```
uniform float focus;
uniform float focusNear;
uniform float focusFar;

float calculateBlurStrength( float depth ) {

     float f;
     if ( depth < focus )
          f = (depth - focus) / (focus - focusNear);
     else
          f = (depth - focus) / (focusFar - focus);
     return min( 1, abs( f ) );
}
```

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

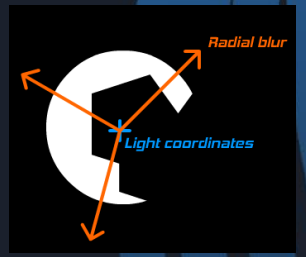
Depth of Field and Bokeh

- Bokeh is a term for an artefact that occurs in Depth of Field, based on the size and shape of the lens and aperture
  - The blur takes on the shape of the aperture
- Bokeh is showing up more in games as it is thought to give a pleasing visual effect
  - One way it is achieved in games is by detecting individual bright pixels with a bright-pass filter, then drawing a small coloured quad at the pixel's location



Radial Blur

- One form of blur that does not use a kernel is a Radial Blur
- Pixels are mixed with pixels sampled towards a coordinate in the image, rather than pixels surrounding the current pixel
  - The central point is usually the center of the screen
- Blurring from a point not at the center of the screen is useful for other effects, such as “God Rays”
  - Pixels are combined with pixels along a vector towards the origin of the light



Motion Blur

- Motion Blur attempts to mimic the blur from fast moving objects
- There are two ways programmers usually achieve this
  - The current back buffer is blended with the previous frame's back buffer
    - This gives poor results but can be acceptable
  - A frame buffer containing pixel velocities is used to control blurring pixels in a set direction with a set strength

Summary

- Shaders are used to achieve many varied visual results, from complex realistically lit scenes to humorous cartoon mayhem
- Post-Processing gives us yet another chance to customise the look of our games to suit our needs
- Many different types of kernels can be used to manipulate our final images
  - Many of which can be sourced from image editing programs like photoshop!

Further Reading

- Wolff, D, 2013, *OpenGL 4 Shading Language Cookbook*, 2nd Edition, PACKT Publishing

- Akenine-Möller, T, Haines, E, 2008, *Real-Time Rendering*, 3rd Edition, A.K. Peters