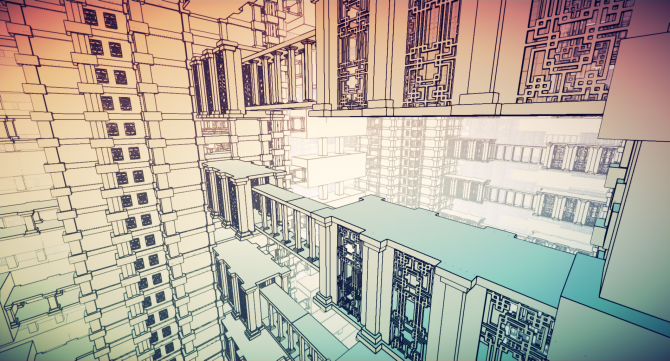
**Edge Detection Shader**

**Overview**

This shader detects and draw edges around a mesh, with configurable thickness and choice of colour. It was inspired by the work of William Chyr on the game Manifold Garden.



The concept is an image-effect process, meaning the calculation and rendering are happening after the scene camera gets the info from the texture it is rendering, creating a post-processing effect.

The camera is then running through a shader and passing information to it. To achieve this effect, the camera sends a *NormalDepth* map. This contains the *Normal,* which way every pixel on materials are facing, and the *Depth*, the distance from the camera the pixel is at. From there, each pixel can run through a function that will:

– Sample surrounding pixels in a cross pattern  
– Combine depth and normal values to form a colour, create an average of it  
– Compare the values of the new combined colour in surrounding pixels

If values are close, it’s not an edge, else it is, and we apply the colour set in parameter.

**Camera Script**

The script on the camera is straightforward.

In the Start function, we make sure the camera mode is set to get the NormalDepth map needed, with

|  |
| --- |
| gameObject.GetComponent<Camera>().depthTextureMode = DepthTextureMode.DepthNormals; |

The function *OnRenderImage* (RenderTexture source, RenderTexture destination) is call after that. It allows you to modify the final image by processing it with shader based filters. The incoming image is source render texture. The result should end up in destination.

Inside we use Graphics.Blit, which copies source texture into destination render texture with a shader, that will be applied through a material, set as a public Material.

Last step, in the Camera Inspector, drag a newly created material that is using our custom shader.

**Shader**

The Shader is of type **Vertex and Fragment** and uses four properties:

* \_**MainTex**, Access the main texture, the one rendered by the camera
* \_**Threshold**, the difference comparison after the sample. The smaller it is, the less difference is needed to be considered an edge
* \_**ExtraSample**, Change the distance of pixels sampled, essentially detecting an edge further, thus creating a thicker edge
* \_**EdgeColor**, Final colour given to the pixel if it is an edge.

The shader will then decode each fragment normal and depth, then each of them will pass through a function:

float4 GetPixelValue(in float2 uv)

This return a colour value as a float4. where R, G, and B are the normals of the pixel, and A is the float depth.

The following frag function will get the pixel value through the precedent function then run the four pixels in a cross shape pattern through the same function, calculate an average, and compare it to the original value.

If it is an edge, we colour it to the color of \_**EdgeColor,** otherwise it stays at the original colour.

More information detailed inside the ***EdgeDetection.Shader***, in the *Shader* folder.

**Test Scene**

The asset folder contains a scene Folder, where the main scene is stored. It is configured to run the shader in a scene containing some coloured squares and a sphere.

To set the shader in a different scene or project, the **EdgeEffect** script needs to be added to the camera, and the **EdgeDetectionMaterial** needs to be be dragged into the **Shader Mat** line in the inspector.

To access the shader parameters, click on the **EdgeDetectionMaterial** in the asset list. The inspector holds the value for the parameters. The *Main Texture* does not need to be changed and must be left empty, as it is filled through the camera. *Threshold* works best at a low value, usually around 0.01. *Thickness* works in increment of 0.5, keep it between 1 and 2 for best results, as a thickness of 3 and over will create artefact in the detection. Finally, *Edge Colour* is self-explanatory.