Edge Detection

Whitepaper

Filename	:	Edge Detection.Whitepaper
Version	:	PowerVR SDK REL_16.2@4266559a External Issue
Issue Date	:	12 Oct 2016
Author	:	Imagination Technologies Limited

Contents

List of Figures

1. Introduction

Edge detection is a process used in computer graphics to determine the borders between different objects or areas in an image. The main uses of edge detection are in computer vision and image processing, generally to help locate individual objects. There are many edge detection techniques available, ranging from those that will run in real-time to complex post processing methods only suitable for offline computation.

This paper details a simple technique used to detect edges on a mobile platform, as implemented in the EdgeDetection Training Course, as well as discussing some alternative approaches. The purpose of detecting edges in this case is simply as a visual effect rather than gaining an understanding of the image's contents.

2. Techniques Used

There are many different ways of detecting edges in a 3D scene and, with the exception of simple normal-based edge detection, they all require an initial render of the scene to be stored as a 2D texture for post processing in a second render. Post processing techniques are covered in depth in the White Paper entitled “Post Processing Effects” that is packaged along with our SDK.

For the EdgeDetection Training Course it was concluded that the ‘object based’ edge detection was the most optimal approach for PowerVR hardware, however, there are other techniques that could have been used to find edges, as detailed next.

2.1. Colour Based

Colour based edge detection is one of the more standard methods of detecting edges and the only one available for a traditional 2D image (Figure 1). This technique essentially looks for steep changes in the colour between areas of pixels, where an edge exists at the boundary between these areas.

As well as locating edges between objects, colour based edge detection will detect internal edges where parts of an object are different colours. This sort of effect can make a scene look quite cartoon-like by highlighting all areas of differing colour in a scene. However, this also means edges will not be detected if two objects are a similar colour, which can cause missing lines in the effect. This technique can also cause problems when textures are used that have a lot of variation, often causing excessive line detection between many segments. If edges are highlighted with a solid coloured line, as in the training course, the texture can become lost in the highlighting.

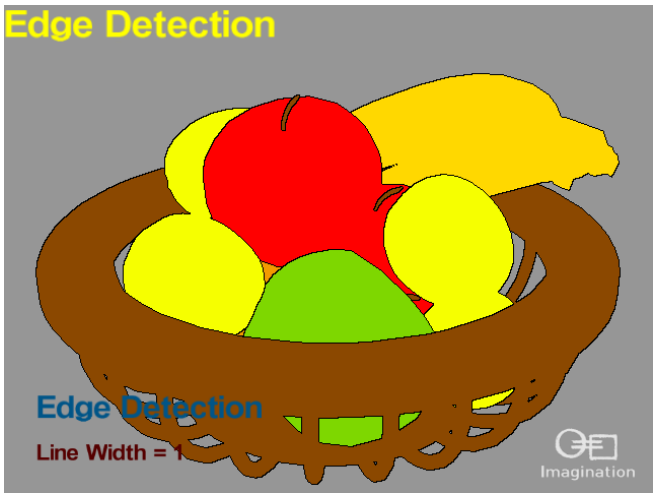

Figure 1. Colour based edge detection

2.2. Depth Based

Using depth to determine edges is a more standard approach in 3D scenes as the actual colour is ignored, which means textures and lighting do not affect the outcome (Figure 2). However, it works in a similar way to colour detection as steep changes in values within the depth buffer are used to detect edges. This technique can detect some internal edges, such as where a part of some object juts out towards the camera. This technique generally presents fewer visual artefacts than colour based edge detection.

The main problem with this method is that edges with little depth variation, such as where a wall meets with the ground, are not detected as there is no change in depth value, which is a fairly frequent occurrence in many 3D scenes. To get around this, normal-based edge detection is often employed to “mop-up” the remaining edges that the depth based approach have missed.

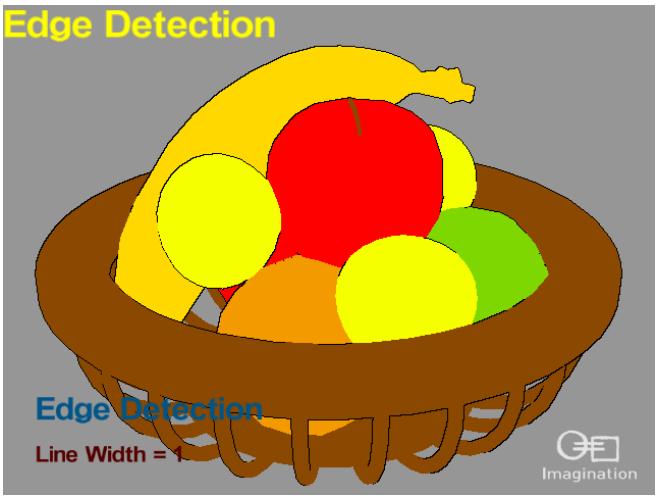

Figure 2. Depth based edge detection

2.3. Normal Based

There are two types of normal based edge detection (Figure 3): the first is a cheap but rough pre-processing technique which is demonstrated in the “Mouse” demo in the PowerVR SDK; the second is a post processing technique which produces a cleaner result.

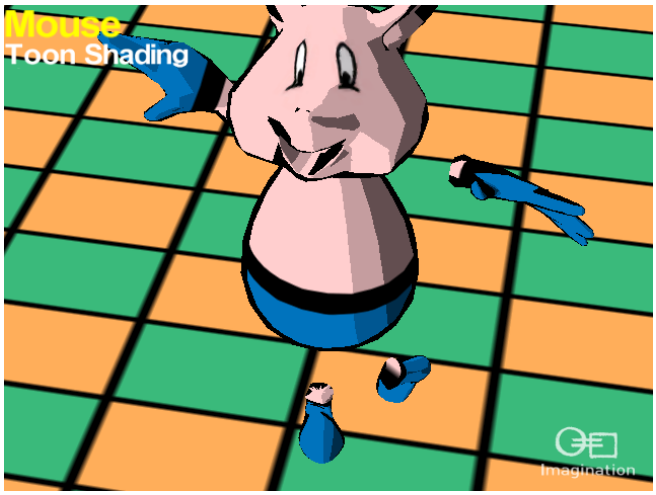

Figure 3. Normal based edge detection

2.3.1. Pre-Processing

This simple approach checks for any normal that is at roughly right angles to the camera, which should draw an edge wherever a front-facing polygon meets with a back-facing polygon. It is quite a rough technique but will provide simple and very fast edge detection. This technique is a very rough one and can have problems, but its speed means it is a good choice for some simpler scenes.

2.3.2. Post-Processing

Another method looks for sudden changes in the normal values at various pixel locations. By looking at pixels that neighbour others with a very different normal, edges between objects that meet up can be found which will, otherwise, not be found with depth-based detection. This method will detect most edges but misses the ones that depth-detection will pick up instead, such as between two objects with similar or identical normals, but at different depths in the scene. Therefore, it is ideally used in tandem with depth based edge detection.

2.4. Object Based

Object based edge detection is novel in that it reverses the standard use of edge detection (Figure 4). Generally, edges are found first and used to locate objects, not the other way around. However, in a 3D scene the objects are known at the start, edges are not. To find the edges from this data, each object is assigned a unique ID and then drawn with this data preserved for each pixel. By then checking wherever the ID value differs from a neighbouring pixel, without requiring any sort of threshold, an edge can be found along the silhouette of the object.

This method will perfectly highlight the edges of any object that is assigned an ID as long as the ID values remain unique. This is, therefore, one of the most accurate single techniques available. However, internal edges that would be detected by other methods will be completely missed, although this can be worked around by manually assigning a different ID value to portions of an object if necessary. Because of the accuracy of edge detection from this data, this technique was chosen for the EdgeDetection Training Course. Other methods had been tried but they all proved more expensive to use and less reliable. A combination of normal and depth techniques would be more accurate, but would be too resource-intensive for current generation mobile platforms.

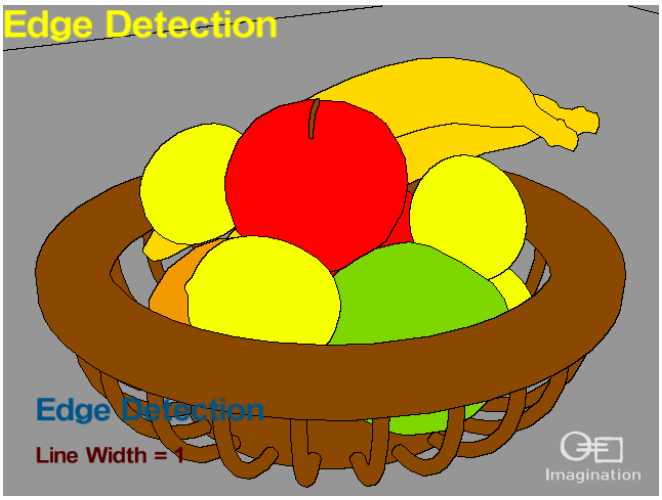

Figure 4. Object ID based edge detection

3. Implementation

To implement the edge detection, a method similar to using a convolution matrix is used (as described in the post processing recommendations document). Firstly, the entire scene is rendered into a texture, which is then drawn to a flat plane covering the entire screen. This plane is processed using a post-process fragment shader which looks at each texel from the first render and checks the ID values of neighbouring texels for differences. When differences are found, the fragment shader recognises this as an edge and is highlighted according to the user's choice.

3.1. Repurposing the Alpha Channel

To implement the object based detection technique, a unique ID for each object needs to be stored. There are two options for doing this: either store it in a parallel texture or store it in the texture that is already being drawn to. Storing this information in the rendered texture provides the best performance, as the number of render passes is kept to a minimum (one to render the scene to a texture and a second to perform post processing) and the post processing render pass only has to read from a single texture. The best place to store the information for the Training Course was in the alpha channel, as this leaves the RGB channels free to store colour information. By putting the IDs into a single channel, the number of available IDs is limited by its precision. For example, a typical 8-bit channel will allow only 256 different ID values, which can cause problems with highly complex scenes that contain more objects than this. This can be somewhat countered by intelligent allocation of IDs but this could slow down the program.

Although there are significant benefits to using the alpha channel in this way, there are some drawbacks. A major problem with using the alpha channel is that transparency is effectively disabled. There are ways of packing the ID and alpha into the same channel to circumvent this, but without the availability of bit manipulation in GLSL ES, this approach can significantly reduce precision. Despite this drawback, the technique works very well and allows accurate edge detection at a solid frame rate, even on lower end PowerVR graphics cores. The limitations of this approach are in most cases fairly insignificant and are largely outweighed by the gains.

3.2. Simplifying the Checks

Texture lookups are expensive, so limiting the number of these is a good way to minimise memory bandwidth.

A traditional matrix check such as convolution will look at several surrounding texels to get a result. Typically this is 4, 8 or even more lookups, as well as the lookup for the central texel. Diagonals can be used for high accuracy, but a sufficient result can be achieved by simply checking four directions (up, down, left and right). Additionally, looking in all four directions is not necessary as the boundary between fragments that are vertical neighbours is checked twice, as are horizontally neighbouring fragments. For this reason, the EdgeDetection Training Course only looks at texels above and to the right of the fragment being processed, which results in edges being drawn at the bottom left of objects. As this test is performed for all objects in the scene, the result is still convincing and is less intensive than performing the two additional texture lookups that four directions would require.

When performing colour or depth edge detection, a check often needs to be done over multiple pixels in each direction using a threshold to account for shallower gradients where a single pixel step may not be enough to find an edge. The object ID approach avoids the need for a threshold, as the difference between the ID values of neighbouring fragments is instantly recognisable.

With all of the simplifications mentioned above, each fragment only needs to look up just 3 texels, taking colour data from just one and the object ID in the alpha channel from all three. This results in an edge detection effect that can be run at interactive frame rates with ease, even on the lower powered PowerVR graphics cores.

4. Future Work

Although the object based edge detection technique used in the training course is a robust approach, the limitation of only being able to store information in the RGBA channels of the rendered texture in OpenGL ES 2.0 prevents additional information being passed to the post processing render. With the introduction of Multiple Render Targets (MRTs) in future graphics APIs it will be possible to write to additional channels so that more information can be given to the post processing render pass. This would not only allow alpha information to be used during post processing with the object based technique, but would also allow other techniques to be combined with this one without introducing additional render passes.

5. Contact Details

For further support, visit our forum:
http://forum.imgtec.com

Or file a ticket in our support system:
https://pvrsupport.imgtec.com

To learn more about our PowerVR Graphics SDK and Insider programme, please visit:
http://www.powervrinsider.com

For general enquiries, please visit our website:
http://imgtec.com/corporate/contactus.asp