

# DEPTH OF FIELD ALGORITHMS FOR MORE REALISTIC SIMULATIONS

DANIEL RHODES, RICHARD CANT, DAVID AL-DABASS

*School of Computing & Technology  
The Nottingham Trent University  
Nottingham NG1 4BU  
[richard.cant@ntu.ac.uk](mailto:richard.cant@ntu.ac.uk)*

**Abstract:** Investigations into the depth of field phenomenon and currently existing real-time graphical simulations of it lead to the conclusion that the current solutions did not provide a sufficiently high level of accuracy to convincingly portray the depth of field phenomenon. In particular these techniques do not provide support for the see through effect, which is defined as being able to see a sharp object viewed through a de-focused object. We investigate the possibilities of implementing a technique that does support the see-through effect using currently available low cost graphics hardware and APIs.

**Keywords:** Algorithms for depth of field, see-through effect, animation.

## 1. INTRODUCTION

Computer graphics hardware for the home market has developed in gigantic leaps and bounds over the past few years. This is due largely to the push towards “*Cinematic Computing*” [NVIDIA, 2003a] which grows in momentum on an almost daily basis.

The ongoing quest for realism is leading to the requirement for more and more advanced techniques in computer graphics systems. One such technique is the depth of field phenomenon, which has yet to be modelled with any great degree of accuracy in real-time computer graphics.

Most computer graphics systems take the easy route and ignore depth of field altogether, opting instead to use the pinhole camera model which produces completely sharp images. By not taking into account depth of field these systems are limiting how realistic they can be, as without depth of field any computer graphics system no matter how advanced will look synthetic. This is quite simply because we, as humans are used to seeing this effect all the time through our eyes. If we're to obtain truly realistic images all aspect of the eye need to be taken into account, even those which may be described as a deficiency of our natural optical system

Systems that choose to ignore depth of field are starving themselves of an excellent method for providing depth cues and diverting the viewers' attention to areas of importance. For example Hollywood films have utilised the depth of field phenomenon to great effect over the years, using it to divert the viewers' attention to important aspects of the scene. If done properly this is not picked up on consciously and provides a very powerful special effect in the Hollywood arsenal.

A more accurate implementation of the depth of field phenomenon would not necessarily be of use throughout computer graphics but it would provide much needed realism for many different types of simulators and perhaps even games, where the quest for greater realism is constantly gaining momentum.

Possible uses for such an implementation of depth of field include military simulators where depth cues are vitally important, such as flight simulators. Over the long term this would potentially provide much more realistic simulations and hence better training conditions. This would of course have the effect of lessening the jump between simulator and real life which would provide numerous benefits, particularly from a military perspective as complex training scenarios could be worked on from the safety of the base.

## 2. WHAT IS DEPTH OF FIELD?

The depth of field phenomena is something experienced perhaps unknowingly by everyone on a daily basis. When the eye focuses on a particular object the objects around it are perceived as increasingly more blurred the further they are away from the object of focus. This also applies to other lens based optical systems such as photography. The actual range in which the eye can see completely sharp objects is relatively small. In-fact the eye is actually rather poor at distinguishing the detail of objects not directly on its focal plane. Most of what the eye sees is non-sharp and a large amount of image enhancement is performed by the brain to get to the final image we see. For an example of how much work the brain actually does after the eye sends its images we can examine the human eyes blind spots, which few people realise exist. Each eye has a blind spot where the optic nerve meets the back of the eyeball, the brain fills in these blind spots utilising image data from the surrounding area. For a practical example see Serendip, 2003. This emphasises the point that the eye can only see clearly directly in its plane of view, hence all the surrounding objects will appear blurred. Figure 2.1 shows an example of Depth of Field in an optical system.

As observed by Rokita [1996] Depth of Field is a direct result of the process of accommodation; which is one of many depth cues that influence the way humans perceive their surroundings.



Figure 2.1: Depth of Field in an optical system

Figure 2.2 shows a visual example of how the depth of field effect is created, the sharp image is created where the rays of light focus directly onto the photoreceptor; this could be the film (in the case of photography) or the retina (in the case of the human eye). In the case where the rays of light focus in front of or behind (i.e. the image is de-focused) the photoreceptors blur circles are created. Blur circles are best explained by taking into account a single point of light as in Figure 2.2.

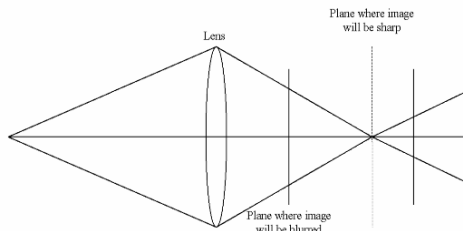


Figure 2.2: How the depth of field effect occurs

Taking Figure 2.2 as an example; if the rays from the light source are in focus then the rays will converge on the view plane to create a sharp image, for example in the eye the in-focus rays will converge on the retina. However if they're not in focus then the rays will converge either in front of or behind the sharp image plane, in the case of the eye this would mean the defocused light rays are spread over an area which is dependant on the sources distance from the focal plane. The brain fills in any missing information from the defocused light which creates a blur circle. Obviously in the real world there is much more than a single point light source, so many thousands of blur circles will appear in any one particular optical image.

This effect is taken advantage of extensively in the worlds of film and photography, for example the recent film 'The Lord of the Rings' makes heavy use of Depth of Field along with various other techniques to draw the viewers' attention to the important aspects of a scene. Obviously this could also be used to the same effect in things such as computer games. However one of the main uses of the depth of field phenomenon for computer graphics images would be to add realism to Virtual Reality systems such as flight training simulators.



Figure 2.3: An example of a computer generated image without depth of field

The problem with most current computer rendered images is that they're based on the pin-hole camera model. Meaning that each image is potentially infinity sharp, obviously taking into account practical limitations. As shown by the image in Figure 2.3 taken from Quake 3. In Figure 2.3 it can be reasonably assumed that the point of focus should be somewhere around the door at the end of the corridor, this would mean that the gun should be partially blurred. So some method is needed to create more realistic computer generated images by inclusion of a depth of field effect.

### 3. EXISTING SOLUTIONS

There are currently a number of solutions to this problem all of which have shortcomings:

#### 3.1 Blurring by multiple viewpoints

This is by far the simplest approach to solving the depth of field problem and also currently the most successful for real-time applications, providing moderate results and supporting the fabled see through effect. The effect is achieved by creating the image from multiple discrete viewpoints as shown in Figure 3.1. The image from each viewpoint is added to an accumulation buffer where the final image is built up.

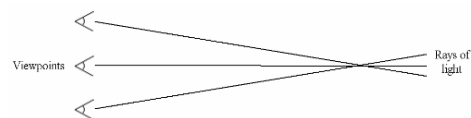


Figure 3.1: Sampling multiple viewpoints

As can be seen in Figure 3.2 this method creates the unwanted effect of multiple images being clearly distinguishable, this is due to the fact that not enough sample have been used. The simple solution to this is to increase the number of samples, however; increasing the level of blurring slightly drastically increases the number of samples that are needed to disguise help this artefact. The number of samples required even for a small level of blurring is potentially very large, particularly with more complex images. Rendering the same scene multiple times is also potentially very time consuming, so a large number of samples will obviously lead to a big performance hit in your application.

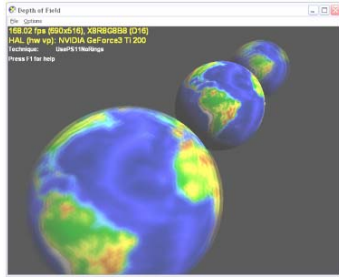


Figure 3.2: Depth of field by multisampling [Microsoft, 2003]

There is another problem with this method that is apparent from figure 3.2; particularly when compared with figure 3.3. It can be seen that the effect created via multiple images is not actually blurring of the image and is perhaps better described as adding fuzziness to the image.

### 3.2 Ray Tracing

Blurring by ray tracing is a form of blurring by multiple viewpoints; the major difference is that ray tracing allows varying of the sample points pixel by pixel. The most realistic depth of field effects can be obtained via ray tracing techniques, where rays of light are traced from the viewpoint to the light source. The calculations involved in ray-tracing make use of the actual physics of light. Figure 2.6 show an example of a ray traced image taken from Pixar's film Monsters Inc. Ray tracing does produce correct results, however; with current technology it is impossible to perform such complex calculations in real time. As such ray tracing is only of use for pre-processed scenes such as the one shown in Figure 3.3.



Figure 3.3: Depth of field by ray tracing

### 3.3 Blurring dependant on depth

The basic idea of these systems is to create a blurring effect dependant on depth (usually the value of  $Z$ ). Examples of this type of system include work by Snyder and Lengyal [1998], Rokita [1996] and also Potmesil and Chakravarty [1981].

NVIDIA® Corporation have implemented a variation of the Potmesil and Chakravarty method [NVIDIA, 2003b], this can be seen in Figure 3.4.

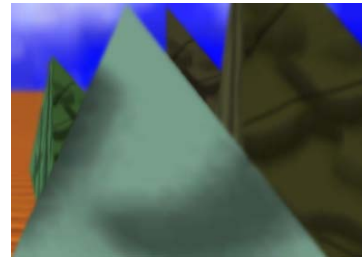


Figure 3.4: NVIDIA Artefacts [NVIDIA, 2003b]

There are problems with the method employed by NVIDIA. For example some of the objects within the scene appear to have an aura surrounding them; this is because of the lack of support for the see through effect. This is a common problem which plagues the Rokita, Potmesil and Chakravarty and similar methods. This artefact can be seen more clearly in ATI's [ATI, 2003] Depth of Field demonstration, shown in Figure 3.5 with the focus plane on the chequered wall. This uses virtually the same method as the NVIDIA example with the exception that pixel shader version 2.0 is preferred over the version 1.1 used by NVIDIA. This of course has the disadvantage of limiting the number of people able to view this demo to those with high-end graphics cards that support pixel shader 2.0. However it does mean that a much higher level of blurring is achieved due to the extra instructions and registers available.

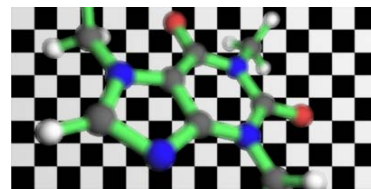


Figure 3.5: ATI artefacts [ATI, 2003]

Snyder and Lengyel (1998) however get around this problem via the use of a layering system. Assuming that the objects in question truly are on separate layers the see through effect is supported. However Snyder and Lengyel's system does have two major drawbacks. Firstly the choice of which objects go on each layer depends on hidden surface removal considerations rather than depth, hence correct ordering for use with depth of field cannot be guaranteed. Also in order to be able to use Snyder and Lengyel's system a non standard method for hidden surface removal must be used. This causes a huge problem as it would require an entirely new type of graphics rendering system and is simply not practical for such a specialised requirement.

## 4. THE ALGORITHM

The first important aspect is the inclusion of a layering system similar to that proposed by Snyder and Lengyal [1998]. However for our purposes the layers are determined directly by depth and not by hidden surface removal considerations as in Snyder and Lengyal's system. This method guarantees that two objects rendered on the same level will have a similar level of blurring,

which was not the case with Snyder and Lengyal's system.

Each pixel within the system consists of x, y and z values (assumed to be in the form 1/z) along with their associated colour values as per a standard z-buffer based system. However unlike a normal z-buffer based system more than just the winning pixel contributions from the depth test must be retained. This is because these values are necessary if the 'see through' effect is to be supported. These must be stored in a depth of field A-buffer structure. Schilling and Staßer [1993] provide a description of a suitable A-buffer however the purpose in that case was quite different, they set out to solve the HSE (Hidden Surface Elimination) problem on the sub-pixel level.

Schilling and Staßer state that the major difference between the A-buffer and a traditional z-buffer is that where a z-buffer only retains one item per-pixel the A-buffer retains a list of pixel contributions.

Obviously all this retention of extra information can potentially cause performance problems. In order to combat this culling can be performed to remove redundant data. For example any winning pixel contribution from behind the focus plane can be ignored, similarly any contributions at a similar depth to the current winning pixel (but obviously still behind it) can be ignored. These culls could potentially be beneficial for a hardware implementation of the algorithm.

The contents of the A-buffer are then blurred to varying degrees dependant on their depth values relative to the focal plane of the system. The in-focus part of the image can be determined by the formula in Figure 4.1. The resultant in-focus image will be at a distance  $v$  where the object in question is at a distance  $u$  from a lens with a focal length  $f$ .

$$\frac{1}{f} = \frac{1}{u} + \frac{1}{v} \quad C = \frac{|v - p|}{v} a$$

Figure 4.1: Equation 1

Figure 4.2: Equation 2

This is fine for the simple case where the system is focused on this object, however if the system is not focused on this object more elements need to be taken into account. The image plane of an out of focus object will be a distance  $p$  and the degree of blurring is dependant on the circle of confusion. The circle of confusion can be defined as "The image of a point source that appears as a circle of finite diameter because of defocusing or the aberrations inherent in an optical system" [Melles Griot, 2003]. In terms of our system the size of the circle of confusion can be determined by the paths of the rays of light which will pass through the edges of our 'lens' aperture and converge on the focal point. As shown by the formula in Figure 4.2 if we take the aperture  $a$ , we can calculate the size of the circle of confusion  $C$ .

This is probably best understood visually; Figure 4.3 shows a visual representation of the two preceding

formulae.  $C$  and  $p$  refer to the case where the image plane is closer to the aperture than the focal plane and where  $C'$  and  $p'$  refer to the case where the image plane is further away than the focal plane. The same equation as shown in Figure 4.2 applies to both cases.

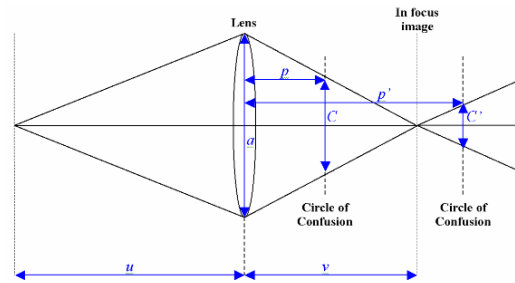


Figure 4.3: Calculating the Circle of confusion

This blurring can be achieved via the use of two mip map style sets known as the b-buffers. One set is used for the areas of the image in front of the focal plane and one for the areas behind, this can be seen in Figure 4.4. The b-buffers start at the screen resolution (level 0) and finish at a resolution consistent with the maximum level of blur required by the scene (level  $n$ ).

The advantages of this method is that the different resolutions can be easily obtained by splatting [see Watt, 2000, p389] the pixels to a lower resolution and also a high level of blurring can be obtained with relatively little processing required. Extra b-buffers cost little time and memory so a higher level of blur can be obtained relatively cheaply.

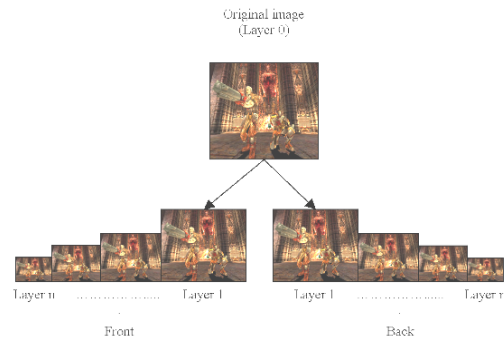


Figure 4.4: B-buffers

The reason two sets of b-buffers are required is that pixel contributions in-front of the focal plane will have a different priority to those behind the focal plain when the final image is generated by matting the contents of the b-buffers. Any z information can be discarded at this stage however information on pixel occupancy (alpha) is now required as the original high resolution pixels will only partially cover the lower resolution pixels generated in the b-buffers. Figure 4.5 shows an example of the occlusion problem.

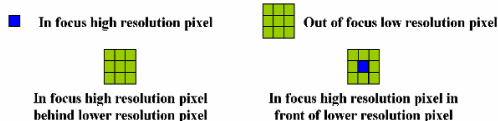


Figure 4.5: Occupancy

The final stage is to recombine the images into our final image. The amount of processing required for this stage can be greatly reduced by using a hierarchical method such as a Gaussian Pyramid. By splatting each layer onto the layer directly above rather than attempting to splat the lowest resolution directly to the highest resolution a lot of processing can be saved.

## 5. A WORKING SOLUTION?

The question still remains; is all this possible on current commercially available graphics hardware and API's? Currently on commercially available fixed function style graphics boards the answer is simply no, not without the addition of the algorithm into the hardware itself. So is there an alternative that allows us to potentially utilise today's hardware (= NVIDIA GeForce3 for our purposes)?

SIGGRAPH '99 included a panel discussion about the future of graphics hardware [Dempski, 2002]. The general consensus of this discussion was that the programmer should have a greater level of control over what happens inside graphics hardware. Most of today's graphics cards have hard wired functionality for lighting, texture mapping etc. what this discussion found was the need for some method to open up the graphics hardware and to allow the programmer to create their own non-standard effects such as anisotropic lighting or cartoon rendering.

In DirectX the 'programmable pipeline' (or more specifically vertex and pixel shaders) provides the interface for this greater flexibility with regards to what can be done with the hardware; for example while the DirectX fixed function pipeline doesn't support Phong shading it can be implemented using a combination of Vertex and Pixel shaders on supporting hardware (e.g. NVIDIA GeForce3 / ATI Radeon9700).

Although the programmable pipeline does offer a much greater degree of flexibility it does have the disadvantage that it is slower than the fixed function pipeline. Traditionally anything that even approached this sort of flexibility needed to be run largely on the CPU (Central Processing Unit) rather than the GPU (Graphics Processing Unit, also known as the VPU or Visual Processing Unit) which has severe performance penalties for graphical applications where lots of fast floating point arithmetic is required. For example the vertex shader hardware on the NVIDIA GeForce3 graphics card (now over two years old) is a SIMD (Single Instruction Multiple Data) FPU (Floating Point Unit) and is considerably faster than even most powerful of the modern Intel Pentium 4 series for floating point arithmetic (at the time of writing), which is the main operation in any graphical system.

## 6. DEPTH OF FIELD DEVELOPMENT

So in theory we have a means to produce a working version of the algorithm running on hardware, but will it work in reality?

**6.1 Depth testing:** Within Direct3D the programmer has three Depth testing options Z-Buffer, W-Buffer and no depth testing. Standard depth testing is performed once per frame and is largely managed by DirectX. The Depth of field problem requires a Z-Buffer solution as outlined in section 4; this is not inline with the standard single pass approach to Z-Buffering but instead requires multiple passes to separate each layer.

While there are various settings that can be changed within Direct3D depth testing (for example to set a Z-Bias) there is no direct way to allow discarded Z values to be retained as is required to build up the multiple layers as outlined in section 4.

The proposed solution is to use "Depth Peeling" [NVIDIA, 2003b], this is an area currently undergoing a large amount of research at NVIDIA. The basic idea of this is that each pass across the scene allows us to get a level deeper into the image. The levels can be thought of as levels of depth; level 0 is the standard Z-Buffer test. Level 1 is the result that the same standard Z-Buffer would provide if level 0 were not part of the image. Level 2 is the result that the Z-Buffer would provide if level 0 and level 1 were not part of the image and so on as can be seen in Figure 6.1 & 6.2.

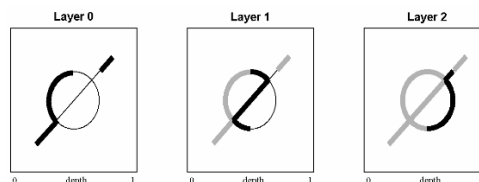


Figure 6.1 Depth Peeling [NVIDIA, 2003b]

This is made possible by the use of multiple depth tests on a single pass. On the first pass Z-buffering occurs effectively as normal, however on subsequent passes the winning pixel contributions from the previous pass are used to discard anything from a previous layer by performing the exact inverse of a standard depth test and setting the comparison value in such a way as to remove the previous winning layer.

Once previous layers have been discarded by the first test the second test kicks in and performs Z-Buffering as normal.

So where does this extra Z-Buffer come from? Shadow mapping plays a surprising role in providing a second depth buffer. Basically shadow mapping is a form of depth testing; there are in fact very few differences between a standard depth test and shadow mapping. The first difference is that shadow mapping sets colour values rather than discarding pixels.

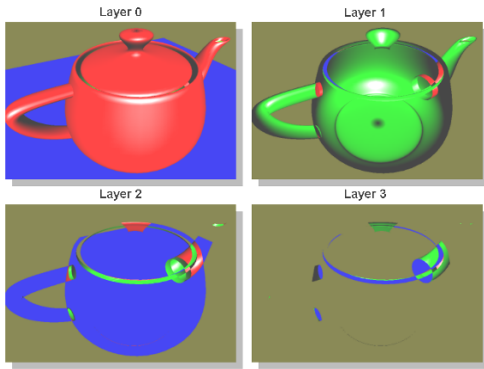


Figure 6.2: Depth Peeling in Action [NVIDIA 2003b]

However this can be worked around by setting the results of the shadow mapping to the alpha channel. Then the alpha test can be used to actually discard the relevant pixels. The second difference is that unlike Z-Buffering the shadow mapping test is not tied to the camera position and as such must be explicitly set to the camera position to allow it to be used as a depth test.

### 6.2 B-Buffers

Modern Graphics cards have the ability to use textures of any size not just the standard multiples of 2. This coupled with the fact that Direct3D has the ability to render to textures means that the levels of the b-buffer can be stored as textures of varying resolutions.

Our original idea was to create the layers as described in section 1 by abusing the D3D texture system and having the D3D mip mapping system create our levels for us. This however proved impossible due to the fact that although D3D will create the mip map levels it manages the selection of those levels automatically by considering standards texturing techniques. Obviously for our purposes we require full control over the level selection as our use of the mip levels would be far from standard. The obvious solution is to use some form of filter to lower the resolution and to store the results as textures. This is made possible by pixel shaders as they operate on the pixel level which is exactly what is required of such a filter.

There is however a potential problem with this solution; the limits on texture addressing and texture blending forced on us by pixel shader 1.1 on the GeForce3 platform mean that even if enough instructions are available to implement the method then only a relatively small change in resolution should be possible.

### 6.3 Pixel Obscuration and Occlusion

Pixel shaders provide the facility to perform texture blending. So as the layers will be stored as textures pixel shaders are ideal to help calculate the final pixel colours taking into account occlusion and occupancy to allow the see through effect. This is a situation where standard texturing facilities could not be used; as this would be using standard texture blending techniques which would not take into account the pixel obscuration problem as discussed in section 1.

Ideally this would be done as the A-buffer is processed; however due to the use of depth peeling the alpha value will be already in use during the processing of the layers and so cannot be used for these calculations.

Another potential place to do this is when the images are re-integrated; this would be an addition to the blending pixel shader used to re-combine the layers. Unfortunately, there are simply not enough operations available to be able to do this. One way to get around this would be to perform several passes loading a slightly different pixel shader for each pass, however intermediate values would need to be stored. This raises another problem as the only logical place to do this is the alpha channel which is as mentioned already in use. Although even if it were not already in use the alpha channel doesn't really provide enough accuracy for these purposes.

### 6.4 Depth Peeling Code

To begin with Depth Peeling was implemented on its own, with the aim of integrating it with the blurring code (see section 6.5) at a later stage in the development process.

The use of depth peeling limits us to graphics hardware which supports shadow mapping within DirectX. This means that in the current market we're limited to the NVIDIA GeForce3 and GeForce4 series. This restricts us with regards to the programmable pipeline options (required later), although the differences between pixel shader 1.1 (GeForce3) and pixel shader 1.3 (GeForce4) are to a large extent negligible. The first real leap in pixel shader technology and the first increase in the number of allowed registers was pixel shader 1.4. Pixel shader 1.4 was proposed by ATI and is currently only supported by ATI hardware, as is shown by The Tech Report [2003].

### 6.5 Blur Code

Due to the problems described in section 4.2 arising from a lack of available pixel shader instructions / registers an exact implementation of the method from section 1 is not possible; on currently available hardware through DirectX. However as shown by NVIDIA [2003a] approximations of some of these things are possible. Hence the blurring method implemented is based on that shown by NVIDIA [2003a].

### 6.6 Putting it all Together

The final task for the development was to attempt to combine the depth peeling with the blurring method. This again posed a problem and again this is due to the same old problems caused in part by the limitations of pixel shader version 1.1.

Both methods push the limit of the number of instructions pixel shader 1.1 offers and combining the two would require a combination of some of the created pixel shaders due to the order of operations that would be required. This is simply not possible with the limits on texture addressing and texture blending put in place by pixel shader version 1.1.

The other problem with combining the two methods is that both require heavy use of the alpha channel and unfortunately the two separate uses are not compatible.

Using the alpha for circle of confusion information would prevent the storage of occupancy information, thus preventing the successful completion of depth peeling and visa versa.

## 8. CONCLUSIONS

The first interesting point is Depth Peeling; this is an area still under research by NVIDIA and as such can be expected to push current hardware to its limits. This is certainly the case as the results in section 5 show; a clear and dramatic drop in frame rate is witnessed when compared to the earlier tests within the same environment. The requirement that depth peeling will not operate without hardware shadow map support effectively killed any chance of coupling it with any form of blurring implementation that requires pixel shaders. This is due to the instruction restrictions of pixel shader 1.1 on the GeForce3. So once the mip map style idea was ruled out due to the limitations of DirectX and it was necessary to utilise pixel shaders to implement the blurring the chances of integrating depth peeling with a blur engine effectively died.

The second major consideration is the blurring code. Again hardware implementations within this area are rare as this is a relatively new field of development. This point can be shown by taking the case of the NVIDIA and ATI Depth of Field attempts both of which appeared well within the last year.

This method like depth peeling also shows poor performance on the GeForce3, as can be seen in the GeForce3 Depth of Field video. This inevitably leads to the conclusion that even if it were possible to combine depth peeling and the blurring code on the GeForce3 platform the potential reduction in frame rate would make it extremely impractical for use in real-time applications.

## 9. FUTURE WORK

The scope for future work in the field is quite large as much is currently not possible. Future hardware developments will undoubtedly lead to a solution similar to the one proposed in section 4 being possible.

Two such contenders to make this all possible is the new GeForceFX from NVIDIA along with ATI's 9800 series.

Instruction counts are liable to continue to rise at exponential rates as the battle for market domination between NVIDIA and ATI heats up; so as time passes it gets increasingly likely that the proposed method and much more besides will become possible in real time and on consumer level hardware.

## REFERENCES

ATI Technologies Inc. 2003. **Depth of Field** [online]. Ontario, Canada: ATI Technologies Inc. Available at: <[URL:http://www.ati.com/developer/samples/dx9/DepthOffield.html](http://www.ati.com/developer/samples/dx9/DepthOffield.html)> [Accessed 23<sup>rd</sup> March 2003].

Dempski, K. 2002. **Real-time rendering tips and techniques in DirectX**. 1<sup>st</sup> ed. USA: Premier Press.

Melles Griot Inc., 2003. **Circle of Confusion** [online]. Carlsbad, California: Melles Griot Inc. Available at: <[URL:http://www.mellesgriot.com/glossary/wordlist/glossarydetails.asp?wID=132](http://www.mellesgriot.com/glossary/wordlist/glossarydetails.asp?wID=132)> [Accessed 9<sup>th</sup> April 2003].

Microsoft® Corporation. 2003. **Depth of Field Sample** [online]. USA: Microsoft® Corporation. Available at: <[URL:http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9\\_c/directx/graphics/programmingguide/tutorial/sandsamplesandtoolsandtips/samples/depthoffield.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/graphics/programmingguide/tutorial/sandsamplesandtoolsandtips/samples/depthoffield.asp)> [Accessed 23<sup>rd</sup> March 2003].

NVIDIA® Corporation, 2003a. **GeForceFX** [online]. Santa Clara, USA: NVIDIA® Corporation. Available at: <[URL:http://www.nvidia.co.uk/view.asp?PAGE=geforcefx](http://www.nvidia.co.uk/view.asp?PAGE=geforcefx)> [Accessed 23<sup>rd</sup> March 2003].

NVIDIA® Corporation, 2003b. **Depth of Field** [online]. Santa Clara, USA: NVIDIA® Corporation. Available at: <[URL:http://developer.nvidia.com/view.asp?IO=depth\\_field](http://developer.nvidia.com/view.asp?IO=depth_field)> [Accessed 23<sup>rd</sup> March 2003].

NVIDIA® Corporation, 2003c. **Interactive Order-Independent Transparency** [online]. Santa Clara, USA: NVIDIA® Corporation. Available at: <[URL:http://www.nvidia.co.uk/docs/IO/1316/ATT/order\\_independent\\_transparency.pdf](http://www.nvidia.co.uk/docs/IO/1316/ATT/order_independent_transparency.pdf)> [Accessed 23<sup>rd</sup> March 2003].

Potmesil, M., Chakravarty, I. 1981. **A Lens and Aperture Camera Model for Synthetic Image Generation**. USA: Computer Graphics (Proceedings of SIGGRAPH 1981).

Rokita, P. 1996. **Generating Depth-of-Field Effects in Virtual Reality Applications**. (s.l.): IEEE Computer Graphics and Applications.

Serendip, 2003. **Seeing more than your eye does** [online]. USA: Bryn Mawr College. Available at: <[URL:http://serendip.brynmawr.edu/bb/blindspot1.html](http://serendip.brynmawr.edu/bb/blindspot1.html)> [Accessed 9<sup>th</sup> April 2003]

Snyder, J. Lengyel, J. 1998. **Visibility Sorting and Compositing without Splitting for Image Layer Decompositions**. USA: Microsoft® Corporation.

Schilling, A. Staßer, W. 1993. **EXACT: Algorithm and hardware architecture for an improved A-buffer**. USA: Computer Graphics (Proceedings of SIGGRAPH 1993).

The Tech Report. 2003. **Dissecting the 3DMark03 controversy** [online]. USA: The Tech Report. Available at: <[URL:http://www.tech-report.com/etc/2003q1/3dmark03-story/index.x?pg=3](http://www.tech-report.com/etc/2003q1/3dmark03-story/index.x?pg=3)> [Accessed 23<sup>rd</sup> March 2003].

Watt, A. 2000. **3D Computer Graphics**. Essex: Pearson Education Limited.