Brandon Kiesling

Engine Development 2

Research Project

November 18, 2008

Depth of Field Rendering Techniques.

Depth of field refers to the range of distances within in a scene that appear acceptably within focus. Any objects outside of this range is out of focus, or blurry. This is a common effect in cinematography or photography, used to direct the viewer's attention to a specific object in a scene, or to clearly separate the foreground from a background. For example, pulling focus in a movie can direct a viewer to look at different places in a scene, following the point of focus as it moves with a scene. (Barsky, 2007) Traditional computer graphics are rendered through what is referred to as the pinhole camera model, which is ever y single object in the scene, regardless of size, distance to the camera or any other variable, are rendered perfectly in focus. However, a real camera lens controls the depth of field in a scene via distance, the f/stop of the lens and the lens' focal length. Replicating depth of field effects is referred to as rendering within the *thin lens* model, as opposed to the pinhole camera. The depth of field effect itself arises from the physical effects of lenses. A point of light in an image is considered to be in focus if it is projected through the lens on to the film (or retina) as a single point. The range of distance at which this occurs is called the *plane in focus*. Anything outside this range will project to a region on the film; known as the *circle of confusion.* (Demers, Depth of Field: A Survey of Techniques, 2004). Because real time computer graphics use the pinhole camera model (as true lens refraction would require expensive ray tracing), depth of field effects must be approximated.

Overview of Rendering Methods

Frustum Jittering

A very simple method of rendering DOF can be affected through the use of the Accumulation buffer. The accumulation buffer holds RGBA color data and is typically used for accumulating a series of images into a composite image. The method here is to draw the scene multiple times, each successive render will jitter the viewing frustum around a common point, such that each frustum when combined produces a plane of focus. (Sheiner, Woo, Neider, & Davis, 2007) Advantages of this method are ease of implementation, and this is the only real-time method that delivers "true" depth of field, correct visibility and correct shading from a lens camera, as opposed to the pinhole camera. (Demers, Depth of Field: A Survey of Techniques, 2004) However in order for the blur effect to be realistic many passes must be made. If too few passes are rendered ghosting or copies of the objects are visible in the most blurred regions of the objects, as can be seen in figure 2. The number of passes required is proportional to the area of the circle of confusion, that is the strength of the out of focus effect, to believably deliver a DOF effect for a strong circle of confusion, the number of rendering passes may be prohibitively high.

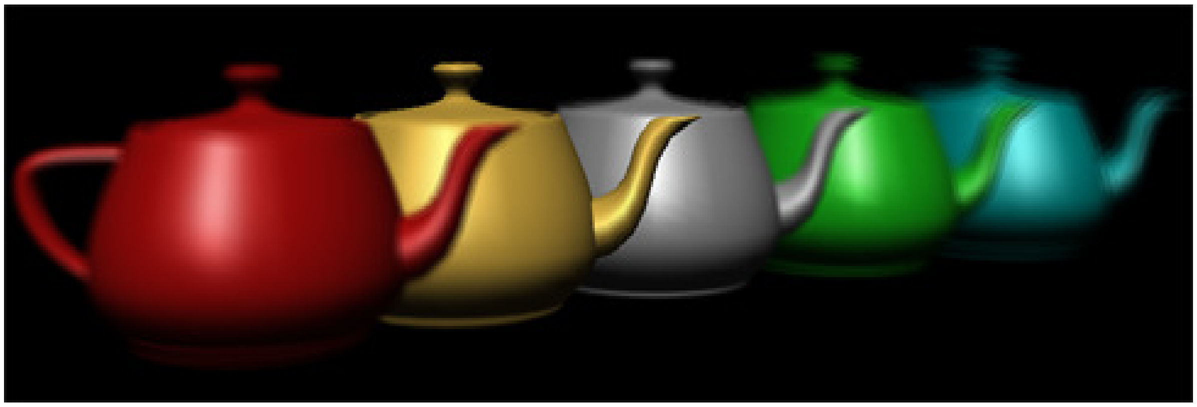

Figure 1: Depth of field effect using frustum jittering. Note the gold teapot is the focused object.

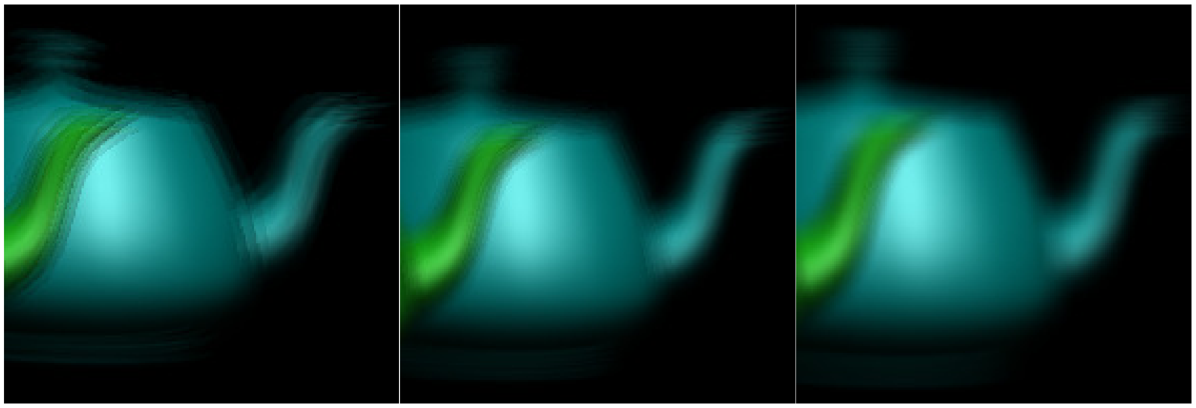

Figure 2: Blurring Artifacts in 8-pass, 24-pass and finally 66-pass rendering. Node the banding artifacts persist until a very high number of rendering passes.

Algorithm

accPrerspective() is used to jitter the frustum, but keep it aligned towards a plane of focus, objects outside of the plane of focus will become blurred due to parallax.

```
typedef struct
{
	GLfloat x, y;
} jitter_point;

/* 8 jitter points predefined data*/
jitter_point j8[] = {}

	glGetIntegerv (GL_VIEWPORT, viewport);
	glClear(GL_ACCUM_BUFFER_BIT);

	for (jitter = 0; jitter < 8; jitter++)
	{
		glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
		accPerspective (45.0,
			(GLdouble) viewport[2]/(GLdouble) viewport[3],
			1.0, 15.0, 0.0, 0.0,
			0.33*j8[jitter].x, 0.33*j8[jitter].y, 5.0);
```

```
            /* Scene rendering goes here */
            glAccum (GL_ACCUM, 0.125);
        }
        glAccum (GL_RETURN, 1.0);
        glFlush();
}
```

Layered Depth of Field

Layered depth of field refers to a technique where objects are sorted into layers that do not overlap based on depth. Then an arbitrary blurring factor can be applied to the scene to approximate the depth of field effect. This is a straightforward method as each layer of depth is rendered individually though the methods both to separate the layers and to blur the image can vary in accuracy and complexity. This approach however can lead to some interesting problems when rendering objects that span multiple layers of depth, or span the entire scene, such as a floor or wall. Splitting these objects into multiple layers can also introduce an artificial seam in the object where the blurring factor abruptly changes, and also can cause objects not normally visible to get blurred into the object incorrectly. Because of these issues this method should only be used where the scene can be constructed specifically to avoid these issues, or that scene-specific workarounds can be implemented. Several published methods of depth of field blurring are a variation on this particular effect, many of which opt to use a hardware texture filtering function such as linear interpolation as opposed to potentially expensive software or pixel-shader convolution effects. This approach results in a lower quality blur effect, but is many times faster in operation due to hardware acceleration. (Henriksson, 2002)

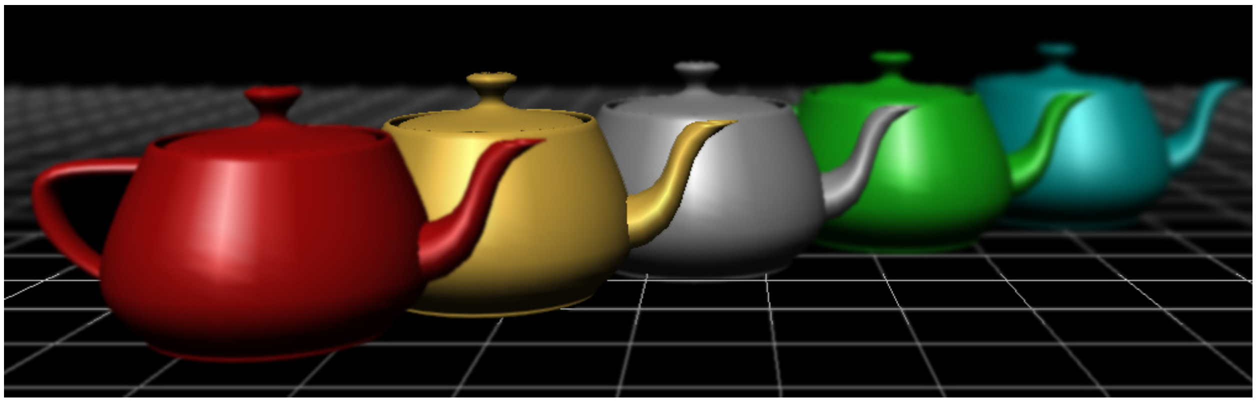

Figure 3: Simulated Layered depth of field. Note the obvious separation of focus on the floor grid as the depth of the scene changes.

Reverse-mapped Z Buffer Depth of field.

Reverse mapped Z buffer Depth of field is called reverse map because Forward-mapped refers to placing arbitrarily blurred objects on the scene which is a common post-processing technique for films and similar media, but has some unique issues with real time graphics. Reverse-Mapped Z Buffer DOF instead arbitrarily blurs part of the scene according to its Z-buffer depth. Many implementations also use graphics hardware to produce the blurred texture to blend with. The first step renders the Z depth of the scene as a texture for texture lookup in a later step. Then it will select which of the blurred textures to use when drawing according to its z-depth.

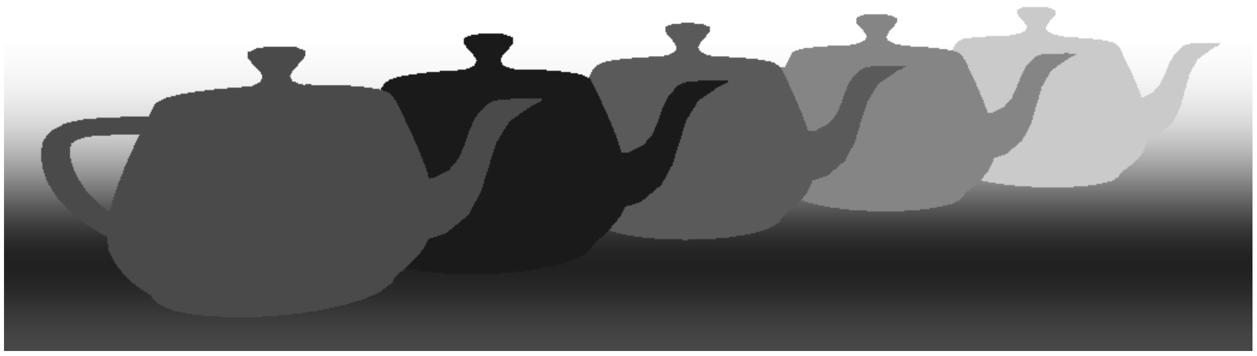

Figure 4: The Z-depth of the scene as rendered to a texture.

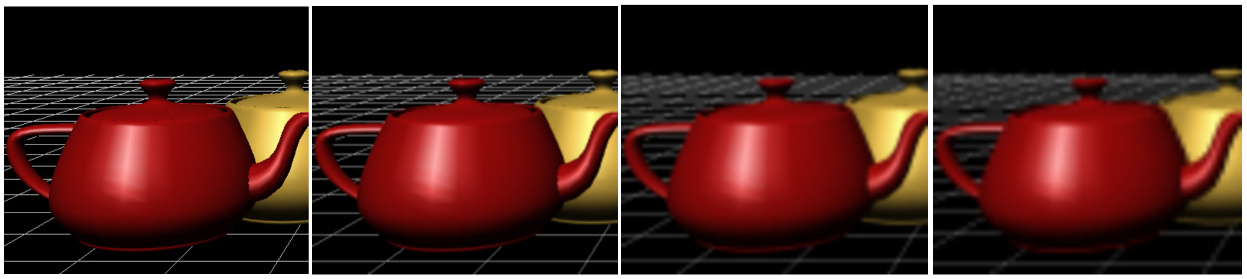

Figure 5: The original scene rendered as a texture, then reduced by 1/2, 1/4 and 1/8 via hardware linear interpolation to affect a blur.

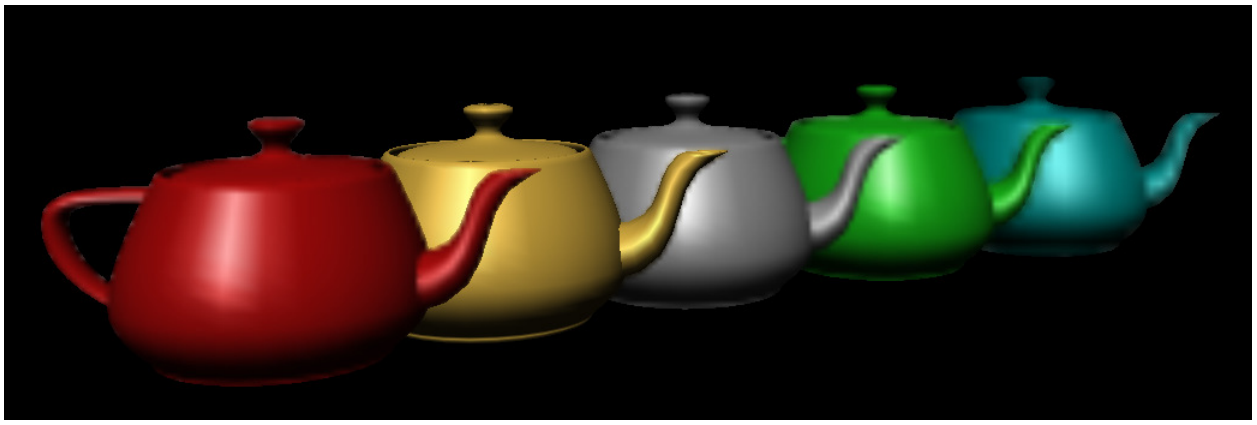

Figure 6: Simulated render of Reverse-Mapped Z buffer Depth of field. Ground lines are removed for clarity.

This method of depth of field is advantageous as it is very quick to perform; it is the fastest method of simulating DOF out of all the methods outlined in this paper. However it has several requirements placed on the graphics hardware in order for it to function. First the hardware must support automatic texture MIP generation, while this is not an issue for any but the oldest of hardware accelerator cards, the card must also support arbitrary selection of the MIP level to use, a feature only available on cards produced in the last five years. There are additionally several other drawbacks to this approach. When using hardware linear interpolation of textures, artifacts can be seen when the mipmapped textures are magnified back to fill size, while this is usually fine for a static image, it can produced a stair stepped “crawling” motion when animated, which is undesirable. This can be alleviated rather easily, by producing multiple blurred images, instead of relying on bilinear interpolation provided by the hardware, but this in turn will increase the performance cost of the algorithm.

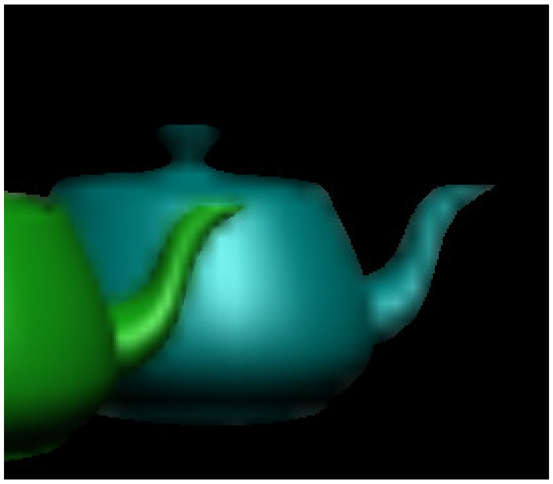

Figure 7: Artifacts from Reverse-mapped DOF. Note the texture artifacts from magnification in the cyan teapot. And that the outline of the teapot remains sharp through it is at the far extreme of the DOF blurring. Also note that the green teapot 'bleeds'.

In addition, because the whole scene is blurred uniformly, areas in focus can bleed into areas out of focus. In practice this is not too noticeable, and can be prevented by splitting overlapping objects into separate layers, as in the Layered Depth of Field method. The most objectionable artifact is discrepancies in depth, because the blur value is determined by a single z depth value, a blurry foreground won’t blur over an in-focus midground object, that is objects, regardless of their blur value will keep their original sharp outlines. There is no completely effective way to prevent this problem, but adding additional full-screen blur when an object hits the extreme foreground can make it less noticeable.

Z-Buffer Destination Alpha DOF

This particular effect does not have a unique name on the published paper it was outlined in, so I’m attributing this one to it for the purpose of this paper. This technique relies on a post process shader to compute the depth value for *each pixel in the scene*, downsample and pre-blur the image, and finally blend the original and blurred versions together. (Scheuermann, 2004) (Křivánek, Žára, & Bouatouch, 2003) Like in the Z-mapped example, the shader writes the distance from the focal point out for each pixel, and stores it as the alpha value for that picture in the destination image.

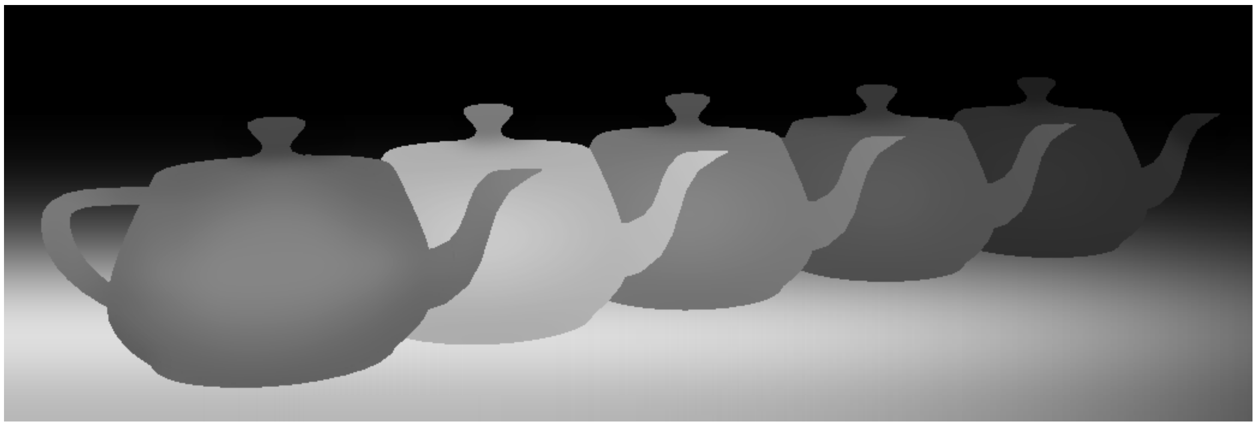

Figure 8: Alpha channel for destination image. Each pixel represents its distance from the focal point and therefore the amount of blur it will receive.

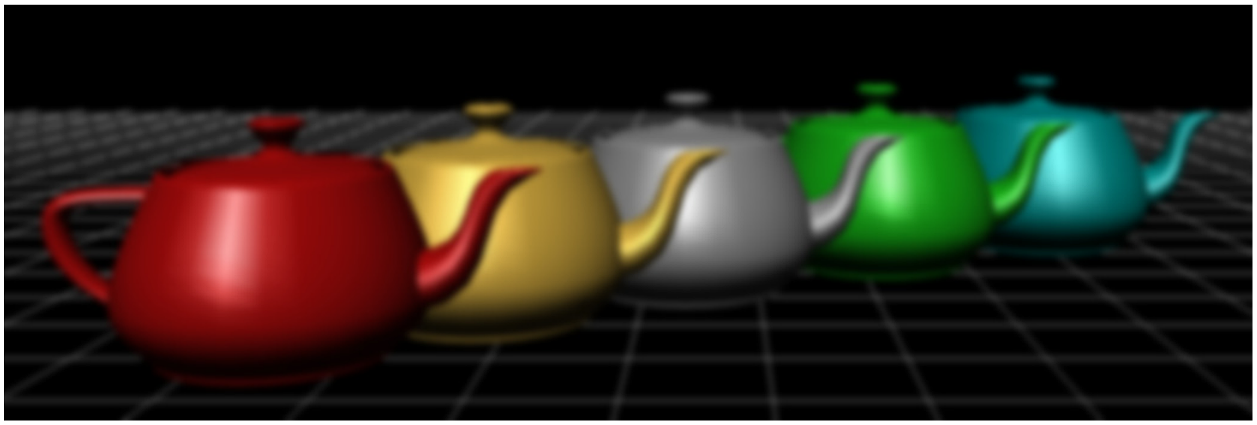

Figure 9: A copy of the image is pre-blurred. This one has been blurred with a 3x3 Gaussian filter.

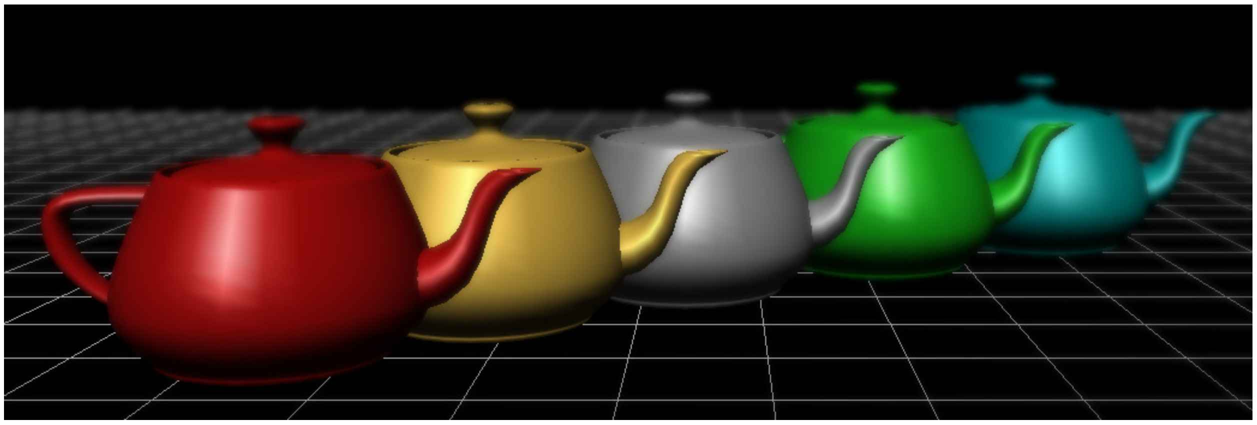

Figure 10: The destination Image and the blurred texture are blended together as a function of the depth value written to the image as alpha.

This method improved over Reverse Z-mapped depth of field as it does not have the problem when objects are at the extreme of the focal range, it is not necessary to further blur objects when they are in the extreme foreground of the scene. However, while the above example does not show it clearly, this approach can still cause sharp foreground objects to leak into the background. However the convolution shader can take this into account and not blur together samples that are likely to contribute to pixel leaking. This approach is also more hardware intensive that the others, requiring hardware that supports at least shader model 2.0. It can be improved by using hardware texture interpolation to blur the texture rather than a pixel shader, but it then offers only negligible improvement in quality over previous methods.

Conclusion

Several of the popular depths of field algorithms have been outlined in this document, but these are not all known methods, nor are any of the listed methods necessarily the best for any given situation. While frustum jittering is still the only accurate method of producing the effect, being the only method that produces correct shading and lighting, however with 3d scenes becoming more and more complex, it is unlikely that this method will be used for any but the simplest of scenes or perhaps as a reference for testing other methods. Layered Depth will be the fastest, but least accurate method; Destination Alpha DOF produces some of the best appearing images in the most number of cases, it is usually for many general purposes. However, it requires the most processing time and hardware support of the methods listed. Reverse-mapped Z buffer DOF is a very good algorithm and if some of the specific limitations of the method, it can give results similar to the Destination Alpha method, but much faster.

Bibliography

Barsky, T. J. (2007). *An Algorithm for Rendering Generalized Depth of Field Effects Based on Simulated Heat Diffusion.* EECS Department, University of California, Berkeley.

Cant, R., Chia, N., & Al-Dabas, D. (2001). *New Anti-Aliasing and Depth of Field Techniques for Games.* The Nottingham Trent University, Department of Computing and Mathematics.

Demers, J. (2003). *Depth of Field in the 'Toys' Demo.*

Demers, J. (2004). Depth of Field: A Survey of Techniques. In *GPU Gems* (pp. 375-389). Addison-Wesley.

Henriksson, O. (2002). *A Depth of Field Algorithm for Realtime 3D Graphics in OpenGL.* Degree Project, Institutionen för teknik och naturvetenskap (ITN), Department of Science and Technology.

Křivánek, J., Žára, J., & Bouatouch, K. (2003). *Fast Depth of Field Rendering with Surface Splatting.* Czech Technical University in Prague, Department of Computer Science and Engineering, Prague.

Mulder, J. D., & Liere, R. (2000). *Fast Perception-Based Depth of Field Rendering.* Center for Mathematics and Computer Science. Amsterdam: ACM Press.

Scheuermann, T. (2004). *Advanced Depth of Field*. Retrieved 11 14, 2008, from ati.amd.com: http://ati.amd.com/developer/gdc/Scheuermann_DepthOfField.pdf

Sheiner, D., Woo, M., Neider, J., & Davis, T. (2007). *OpenGL Programming Guide.* Addison-Wesley Professional.

Yu, T.-T. (2004). DEPTH OF FIELD IMPLEMENTATION WITH OPENGL. *Journal of Computing Sciences in Colleges* , 136-146.