# Importance Ordering for Real-Time Depth of Field

Paul Fearing

Department of Computer Science
University of British Columbia
Vancouver, B.C.
Canada
fearing@cs.ubc.ca

## Abstract

Depth of field (DOF) is an important component of real photography. As such, it is a valuable addition to the library of techniques used in photorealistic rendering. Several methods have been proposed for implementing DOF effects. Unfortunately, all existing methods require a great deal of computation. This prohibitive cost has precluded DOF effects from being used with any great regularity.

This paper introduces a new way of computing DOF that is particularly effective for sequences of related frames (animations). It computes the most noticeable DOF effects first, and works on areas of lesser importance only if there is enough time. Areas that do not change between frames are not computed.

All pixels in the image are assigned an importance value. This importance gives priority to pixels that have recently changed in color, depth, or degree of focus. Changes originate from object and light animation, or from variation in the camera's position or focus.

Image pixels are then recomputed in order of importance. At any point, the computation can be interrupted and the results displayed. Varying the interruption point allows a smooth tradeoff between image accuracy and result speed. If enough time is provided, the algorithm generates the exact solution.

Practically, this algorithm avoids the continual recomputing of large numbers of unchanging pixels. This can provide order-of-magnitude speedups in many common animation situations. This increase in speed brings DOF effects into the realm of real-time graphics.

**CR Categories and Subject Descriptors:** I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism.
**General Terms:** Algorithms.
**Additional Key Words and Phrases:** Depth of field, postfiltering, camera model, lens and aperture, importance ordering

## 1 Introduction

DOF control is a useful and important photographic technique. It draws attention to in-focus objects and areas. Changing the focus forces movement of the viewer's center of attention, allowing a cinematographer to express a change in subject importance. DOF can also be used as a general artistic tool for fade outs, fade ins and soft focus sequences. Many tiny details of the world can be omitted or approximated if the cinematographer knows they will be out of focus. This can reduce the time and effort required design scene backgrounds.

DOF provides an important contribution to the general appearance of so-called "photorealistic" images. Photorealistic rendering implies being able to simulate the entire camera and film assembly exactly, including defects and aberrations. Without these effects and defects, it is often hard to fool humans into believing a rendered image was captured by a camera. Computer generated imagery usually looks "too perfect". DOF is expected by humans in both images and our own natural vision processes.

A realistic camera model is especially important in image forgery and computed-augmented reality, where images are a blend of the real and the computer generated. DOF consistency plays an important part in the believability of the image. DOF also forms a good basis for exploring other types of focus-based lens defects, such as spherical [Cox 71] and chromatic [Boult 92] aberrations.

DOF has been used experimentally for isolating areas of 3D MRI data [Wixson 90] and particle systems [vanWijk 92]. The plane of interest is rendered in focus, while other areas are drawn out-of-focus. Current techniques are far too slow to be interactive.

The computer simulation of DOF is also useful in vision, where attempts have been made to derive scene depth from focus [Pentland 87] and to sharpen out-of-focus images [Savakis 91] [Sezan 91]. Synthetic DOF images provide exact ground truth values useful in algorithm error analysis.

Even with all these proven applications, DOF is rarely used in the graphics world. The main drawback appears to be the computational cost. A single frame can take minutes to compute. Even worse, easy DOF control almost always requires a long sequence of related frames. Humans focus real cameras by trial and error: point at a subject and manually change focus until the desired effect is achieved. Slow DOF calculations preclude this sort of focus experimentation in computer generated images. The desired effect must be calculated before starting the computation, using knowledge about object depths, DOF ranges, etc. This is feasible for simple test scenes, but it becomes annoyingly time-consuming for complex scenes containing many moving objects. Often, the rendered result does not contain the exact effect desired. Try after try is required to get the precise amount of DOF. This makes it too bothersome to include DOF except in the most ambitious projects.

If DOF computations were fast enough to allow near real-

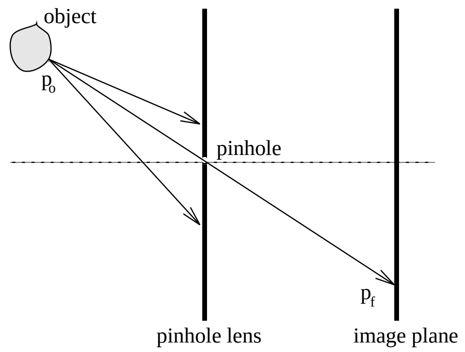


Figure 1: Pinhole Camera Model

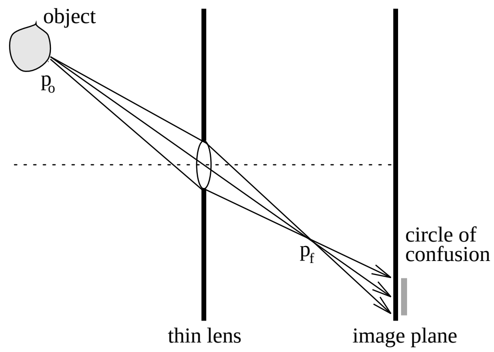


Figure 2: Thin Lens Camera Model

time computation of sequences of related frames, rendering systems could allow trial-and- error adjusting of DOF effects. This would open up DOF to many new types of applications.

## 2 Depth of Field

DOF has been explained in Potmesil and Chakravarty [Potmesil 81], Kingslake [Kingslake 92], and others. We will provide a brief review for clarity.

### 2.1 Pinhole Camera Model

Many graphics systems use the pinhole camera model as the basis for object appearance.

Light rays are scattered off objects in world space. Some of the scattered rays are bounced towards the camera. Rays originating from the same object point $p_o$ (but travelling in different directions) hit an infinitely small pinhole lens. This lens allows only a single ray to pass through the pinhole. Rays going in other directions are ignored. Thus, when the single ray hits the imaging plane, it is always in focus.

Figure 1 shows the pinhole camera model.

### 2.2 Thin Lens Model

Obviously, real lenses have a finite dimension, and let in light coming from several different directions. This can be approximated using a thin lens model, as shown in Figure 2.

As before, rays are scattered from objects in world space. Rays originating from the same object point $p_o$ hit the lens from a number of different directions. The incoming rays are focused by the thin lens into a point $p_f$. If the imaging plane is located at $p_f$, the image of the object will be in focus. If the imaging plane is not at $p_f$, the incoming cone of rays intersect the image plane to form a conic, usually approximated as a circle. This circle is called a point's "circle of confusion" (CoC).

Potmesil [Potmesil 81] calculates CoC diameter for an out-of-focus point $u_o$ as

$$C = |V_u - V_p| \left( \frac{F}{nV_u} \right) \quad (1)$$

where $F$ is the focal length, $n$ is the aperture number, and

$$V_u = \frac{FU}{U-F} \quad U > F \quad (2)$$

$$V_p = \frac{FP}{P-F} \quad P > F \quad (3)$$

where $V_p$ forms a point on the image plane. Figure 3 diagrams the various distances. The CoC is used to compute how points are imaged over an area.

Humans can only resolve angles larger than a certain amount. This implies that (viewed from a certain distance) CoCs smaller than a certain limit are resolved as points, while larger circles are resolved as areas. Viewed from a distance $F$, the common limit of human resolution [Kingslake 92] is a CoC with diameter $\frac{F}{1000}$. Thus, CoC diameters can be thresholded at $\frac{F}{1000}$ when viewed from a distance $F$.

## 3 Previous Work

Existing DOF computation methods are described briefly below.

### 3.1 Linear Postfiltering

Potmesil and Chakravarty [Potmesil 81] were the first to introduce a DOF model to the computer graphics community. They based their approach on a two-pass postfiltering process. The first (image rendering) pass computes an RGB image and a corresponding Z depth map. The second (DOF filtering) pass computes DOF effects. The CoC of each pixel is computed using Eq.1. A pixel $P$'s intensity is then the summation of the weighted intensities of all other pixels with CoCs that overlap $P$. Potmesil and Chakravarty used diffraction properties to come up with a function that modeled intensity distribution within an individual CoC. The authors made heavy use of lookup tables to improve processing speed. The diffraction-based intensity distribution profile was approximated in Chen [Chen 87], with no noticeable effect on image appearance. The vision community [Lee 90] also has a number of intensity distribution models, primarily for depth retrieval and image restoration.

Potmesil and Chakravarty's method has the advantages of postfiltering, including simplicity and speed proportional to image size. The main disadvantage is that the CoC is computed from a single object point. This means that filtering does not recognize objects partially blocking the CoC's effect. This partial occlusion can cause blurry backgrounds to "leak" onto sharp foreground objects, as shown in Figure 5. The foreground black band appears partially transparent, especially in front of the bright filing cabinet.

### 3.2 Distributed Ray Tracing

Cook, Porter, and Carpenter [Cook 84] implemented DOF using distributed ray tracing. Each image point $P_o$ is traced through the center of the lens to an in-focus point $p_f$ on the

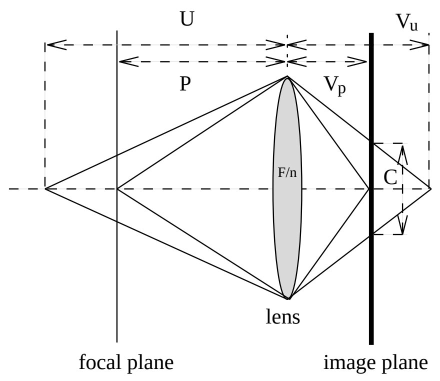


Figure 3: Calculation of the Circle of Confusion

focal plane. Subsequent rays from $P_o$ are sent to positions on the lens, and then aimed at point $p_f$. If no object is located at $p_f$, the rays diverge into the environment and are averaged to compute a result. With enough samples, and sufficient variation over the lens area, distributed ray tracing can compute DOF.

Because rays are actually traced into the environment, distributed raytracing solves the partial CoC occlusion problems of linear postfiltering. It also computes a better estimation of object color, as different parts of the lens see the object from different angles (which implies potentially different colors). Distributed ray tracing is also attractive because the DOF problem is solved at the same time as several other problems, including shadow penumbras, motion blur, and translucency.

The main disadvantage of distributed raytracing is that object space renderers can be potentially very slow - certainly much slower than real time. As well, combining DOF with object rendering makes it very expensive to change the DOF without changing the objects.

### 3.3 Accumulation Buffer

Haeberli and Akeley's [Haeberli 90] accumulation buffer was introduced to provide hardware support for antialiasing. It can also be used for DOF effects. The accumulation buffer integrates the results of multiple rendering passes of the same image. On each pass, the image is drawn from a slightly different eyepoint. Eyepoints are sampled across the lens aperture.

The accumulation buffer avoids partial CoC occlusion problems, allows a smooth improvement in image quality with time, and can be used to simultaneously implement antialiasing, soft shadows and motion blur. The accumulation buffer resembles a frame-buffer implementation of distributed ray-tracing. As such, cost is proportional to the complexity of the world scene, and the number of viewpoint samples.

### 3.4 Ray Distribution Buffer

Shinya [Shinya 94] adapted the distributed ray tracing idea and converted it into a postfiltering process. It requires an initial RGB image and Z-buffer. Each pixel in the image is assigned a small personal Z-buffer, called a ray distribution buffer (RDB). Each element in a pixel's RDB is associated with an incoming ray direction.

For each pixel $U$ within pixel $P$'s CoC, the algorithm computes an associated object point and incoming ray direction. This direction is used to assign $U$'s object point color and Z value to some of $P$'s RDB elements. Incoming RDB elements are Z-buffered, leading to an averaged result that takes into account the partial occlusion of CoC areas.

Algorithm complexity depends upon image size and RDB size, offering an improvement over distributed ray tracing and the accumulation buffer. However, it does not account for color changes due to variations in ray direction.

### 3.5 Clustering Methods

DOF can be simulated without using a model of the synthetic camera. Scofield [Scofield 92] groups scene objects into foreground and background planes. Each plane is rendered separately, blurred using a low-pass filter, and then composited together. The filter sizes are unrelated to the camera model, and are chosen ad hoc. The artificial grouping of objects, segregated scene rendering, and trial-and-error filter sizing make this method hard to use.

## 4 Importance Ordering Depth of Field

All DOF methods were originally discussed as they applied to single images. This implies that all pixels must be recomputed on each and every frame, even if there is little or no change between frames.

This can be especially expensive for scenes where sharp objects are moving in front of large and blurry (hence costly to compute) backgrounds. The DOF changes in only a small portion of the total number of pixels, yet the expensive background must be recomputed for each frame. There are many other examples of minimal (yet noticeable) scene changes requiring a complete image recompute.

Recomputing the entire image also proves tiresome for the human viewer. Results are not available until all work has been completed. This prevents a human from previewing (and potentially interrupting) the image in-progress.

In this section we describe a fast postfiltering DOF method using *importance ordering*. The main contribution of this algorithm is to recognize that there is a great deal of consistency between sequential frames of an animation. We can use this continuity to avoid recomputing areas of the scene that do not change between frames. Pixels that do require recomputation are processed in the approximate order of their noticeable visual effect.

Importance ordering DOF rendering allows substantial speedups when generating multiple frame animations. It also allows the most obvious DOF effects to be previewed at an early stage in the computation.

We have chosen a postfiltering approach for several reasons. Postfiltering requires no knowledge of world space, thus allowing a complexity dependent on image size. Postfiltering techniques can also be meshed with existing frame and Z-buffer hardware. This provides hope that fast DOF can be added to polygonal based graphics rendering engines. Adding DOF effects in after rendering allows a user to modify camera parameters without having to re-render the original image.

We use Potmesil and Chakravarty's postfiltering method, for reasons explained Section 5.

### 4.1 Algorithm

Each frame begins with the current picture, consisting of an RGB image $I_c(x,y)$, a Z-buffer $Z_c(x,y)$, and CoC values, $C_c(x,y)$. Each pixel also contains $I_l(x,y)$ and $C_l(x,y)$ elements that contain the pixel's status at the time of its last update. Individual pixels have R, G and B numerators and a single denominator to keep track of the current summation of all overlapping CoC effects, including its own.

The first pass computes $C_c(x,y)$ for every pixel $P$. After $C_c(P)$ is computed, a pixel $P$'s update importance is calculated using an importance function (detailed below). This function returns an importance measure $h$ within the range $[0..h_{max}]$. Depending on $h$, pixel $P$ is placed into one of $h_{max}$ hash buckets. Hash bucket 0 represents pixels that do not need to be recomputed this frame.

The second pass computes DOF effects for the image. A pixel $P$ is taken from the highest non-empty hash bucket. The previous CoC effect of pixel $P$ on its neighbors is computed using $I_l(P)$, and $C_l(P)$. The current CoC effect on $P$'s neighbors is determined with $I_c(P)$, and $C_c(P)$. The difference of the two effects is added to all affected neighboring pixels. Once pixel $P$ has been processed, $I_l(P)$ and $C_l(P)$ values are updated.

The algorithm continues to take pixels from hash buckets until all but hash bucket 0 are empty, the user interrupts, or some preset number of pixels have been processed.

After interruption, the average intensity value of all pixels is computed. Pixels that were not updated on this pass must be included, in order to count changes due to updated neighbors. If a new frame is required, it is loaded into $I_c(x,y)$ and $Z_c(x,y)$.

### 4.2 Intensity Distribution Function

CoC effects take the form of an intensity distribution $D$ centered around pixel $P$. A neighboring pixel $U$ is affected by the area under the intensity distribution curve $D$, multiplied by pixel $P$'s color values. Intensity curves can be flat, gaussian, or as complicated as Potmesil and Chakravarty's diffraction model.

The intensity distribution function can be computed a priori, and stored as a number of tables of various sizes. Tables can be computed and interpolated between with varying degrees of precision.

In order for this method to work, the intensity distribution function must be individually retrievable. That is, a pixel $P$ must be able to determine its last contribution to its neighbors based solely on its own current and past information.

### 4.3 Importance Function

Every pixel is ranked in update priority by an importance function. This function attempts to link a pixel's update order to the amount of change in its appearance.

Noticeable differences in a pixel occur in three cases: CoC diameter changes, Z value changes, and color changes. CoC diameters change with the camera model parameters and Z values of the target. Differing CoC diameters cause changes in the $D$ filter size, resulting in new intensity distributions among a pixel's neighbors. Differences also occur when a pixel changes color. All of these effects will be usually be present in an animation sequence.

The importance of a change in CoC for a point $P$ is measured by $h_{coc} = |C_c(P) - C_l(P)|$. The color change effect $h_{rgb}$ is based on a simple Euclidean color distance. The CIE LUV uniform color space can also be used to calculate color distances that are perceptually equal.

P's total importance $h$ is a linear weighting of the CoC and color effects:

$$h = W_{coc}h_{coc} + W_{rgb}h_{rgb} \tag{4}$$

The relative weighting between the effects can be chosen to suit a specific part of an animation. Pure color changes are most likely to occur when scene lighting moves around. Pure CoC changes are most likely to occur when changing camera parameters. Without a priori knowledge of the animation, equal weights can be used. Of course, the weighting functions need to be scaled to spread values out across $h_{max}$ hash buckets.

Note that pixels that do not change in color or CoC are given an importance of 0, and not updated on this frame. We can also threshold very small $h_{coc}$ or $h_{rgb}$ to 0, if color or CoC changes are small enough not to be detectable or important. For example, CoC changes can be thresholded at $\frac{F}{1000}$, at the limit of human resolution.

Skipping a pixel $P$ in the importance ordering algorithm means the last change to $P$'s effect has not yet been reflected in the image. Skipping a pixel in the linear algorithm implies that pixel $P$'s entire effect has been neglected.

### 4.4 DOF Interruptions

Importance ordering allows the user to interrupt the algorithm to see the most important results completed so far. The program can also interrupt the algorithm after a set percentage $M < 100\%$ of hashed pixels. This allows a gradual increase in speed with a gradual decrease in accuracy (with respect to a linear DOF method). Pixels not updated on one pass grow in importance the next pass.

If $M$ is too small, pixels can be starved out. This can cause objectionable artifacts if starved points are part of a moving object. Reducing $M$ is most useful for gradual zooms on static scenery, where it is harder to detect focus changes for points that lag on the update.

Unchanged pixels are not hashed, allowing large speedups even when $M = 100\%$ of the hashed pixels. In this case, each frame produces the same result as the linear DOF method.

## 5 Partial CoC Occlusion

We use Potmesil and Chakravarty's postfiltering method because the DOF of pixel $P$ on neighbor $U$ can be computed solely by looking at pixel $P$. This allows past DOF to be computed and removed from neighboring pixels. This precludes using the RDB postfiltering method, because RDB buffers do not allow DOF reversal. Consider a pixel $P$ that is updated so that $C_c(P) < C_l(P)$. This information is propagated to neighbor $U$'s RDB. Pixel $P$'s previously larger effect cannot be removed from $U$'s RDB Z-buffer because there is no record of what $P$ previously occluded.

Using Potmesil and Chakravarty's method introduces some problems, mainly the intensity leakage due to partial CoC occlusion. In some cases, the effect is not noticeable and can be ignored. However, we would like to minimize the effect while still maintaining reversibility.

One possible solution is to Z-buffer pixel $P$'s contribution to its neighbors. Pixel $P$'s effect is added to neighbor $U$ only if $Z_c(P) < Z_c(U)$. This prevents a blurry background $P$ from contributing to a sharp foreground $U$. This helps reduce intensity leakage, but destroys reversibility. In order to recover past DOF effects, pixel $P$ must keep track of which $U$s were occluded.

We can add an $L \times L$ bitmap centered around each pixel $P$. Bit position $U$ is 1 if $P$ did not contribute to neighbor $U$. When $P$ is next updated, this bitmap allows the recovery of previous contributions, even if neighboring Z values have changed.

Attaching a bitmap to each pixel can be expensive, especially with large CoC sizes. Space can be reduced by setting $L$ to be less than the maximal cutoff CoC filter size. Neighbors within $\frac{L}{2}$ may be omitted or included. Neighbors outside $\frac{L}{2}$ are always added. This means that pixels can only get intensity leakage effects from pixels $\frac{L}{2}$ positions away. With a gaussian or diffusion intensity distribution, $L$ does not need to be very large before bleeding effects are small.

In practice, a straight Z comparison works well for different foreground/background objects. It can cause artifacts on intersections and single surfaces, where adjacent points are only slightly in front or behind each other. We can improve results by occluding only if $P$ is behind $U$, and $|C_c(P) - C_c(U)|$ is greater than some user-variable cutoff $K$. This has the effect of not occluding surfaces that are at about the same level of blurriness, within some tolerance.

Figure 6 compares the effect of intensity leakage prevention with $L = 32$, and $K$ equal to the equivalent of 3 pixels. The darkness of the foreground band has been improved.

## 6 Experiments and Discussion

Both linear and importance ordering DOF methods were implemented as a set of library functions. These libraries read and write values directly to the framebuffer and z-buffer, allowing the addition of DOF to any framebuffer based program. Adding DOF to an existing program requires approximately 3-4 lines of new code.

For simplicity, both methods used a gaussian intensity distribution function, without intensity z-scaling, or table interpolation. The distribution tables were computed to half-pixel boundaries. Filter vignetting was ignored. Because both methods use the same tables, the intensity distribution function does not affect the relative comparisons.

Several experiments were carried out to test the speed of the importance ordering DOF method.

### 6.1 Experimental Results

Five different animations were generated, each consisting of 100 frames of size 256h by 256w. The background was raytraced offline, and was loaded into the framebuffer/z-buffer on each frame. The raytraced background allowed extremely complex scenery without a corresponding increase in rendering time. The star was rendered directly into the framebuffer/z-buffer, and thus can occlude and be occluded by the background. Figure 7 shows the scene without DOF.

In the first four experiments, the importance ordering DOF method was directly compared with the linear Potmesil DOF. The importance ordering method used $M = 100$, $w_{coc} = 0.5$, and $w_{rgb} = 0.5$. Both methods used the same code where possible, including the intensity distribution function. No particular code optimizations were implemented. Experiments were performed on a 100 mHz SGI Indy.

Results are shown in Table 1. Times are in elapsed seconds, and not CPU seconds. Note that to compute the first frame, the importance ordering method must spend as least as much time as the linear method. Table 2 shows the update speed with the cost of the first frame omitted. The longer the animation sequence, the faster the first-frame cost is amortized across all frames.

The first experiment consisted of the focused star moving left-to- right in between an unfocussed foreground and background (Figure 8). The camera was focused at 40 mm, with $F = 8$ mm, and $n = 3.5$. The importance ordering DOF method was able to perform over 22.1× faster than the linear method, mainly because the majority of the out-of-focus area was not recomputed. The greater the cost of the unchanging area, the larger the speedup.

The second experiment involved the star zooming away from the camera. The object started in focus, and moved out-of-focus as it approached the background. The camera was focused at 24 mm, with $F = 8$ mm and $n = 17$. The importance ordering DOF gained a 13.4× speedup. Speedups were less than experiment one because the target object covered more of the total picture area over the duration of the animation.

Table 1: Importance Ordering DOF vs. Linear DOF

| **Exp . #** | **1** | **2** | **3** | **4** |
|---|---|---|---|---|
| Linear | 2572 s | 1165 s | 500 s | 522 s |
| Importance | 116 s | 87 s | 209 s | 213 s |
| No DOF | 3 s | 3 s | 3 s | 3 s |
| Speed-up | 22.1 x | 13.4 x | 2.4 x | 2.4 x |

Times are for 100 frames.

Table 2: Update Rate Without First Frame

| **Exp . #** | **1** | **2** | **3** | **4** |
|---|---|---|---|---|
| Linear | 25.7 s | 11.6 s | 4.8 s | 5.0 s |
| Importance | 0.9 s | 0.8 s | 1.9 s | 1.9 s |
| Speed-up | 28.5 x | 14.5 x | 2.5 x | 2.6 x |

Times are seconds/frame averaged over frames [2..100].

The third experiment involved a foreground to background focus change on a stationary scene. The camera focussed from the knot to the computer over 100 frames, with $F = 8mm$, and $n = 12$. The importance ordering DOF ran 2.4× faster, mainly because $h_{coc}$ values less than the minimum table division were hashed to $h = 0$. There was no noticeable visual difference between the two methods. Figure 9 shows the start of the animation.

The fourth experiment consisted of a combination of experiments one, two, and three. The star moved from left-to-right, zooming away from the camera. The camera focussed on the star as it moved away from the camera, using $F = 8$ mm and $n = 12$. The importance ordering algorithm ran 2.4× faster than the linear method.

The final experiment approximated experiment three, except that $M$ was varied to truncate the computation. The zoom direction was reversed to maximize blur inconsistencies. The first frame always used $M = 100\%$. Corresponding times are shown in Figure 4. Faster times mean less accuracy. In this particular experiment, a few pixels require most of the effort. Thus, large numbers of pixels can be removed without much affect on speed or visual accuracy. Figure 10 shows importance ordering with $M = 15$, on the final frame of the animation. Figure 9 shows $M = 100$ for comparison. Note the similarity of the images, even though only 15% of all changing pixels were updated each frame.

### 6.2 Discussion

The importance ordering DOF algorithm works extremely well for scenes with a constant focus and moving objects. This type of scene setup is very common in films and video. In experiment one, we were able to achieve a sub-second average time, even with a very blurry foreground and background. This is within our definition of achievable real-time

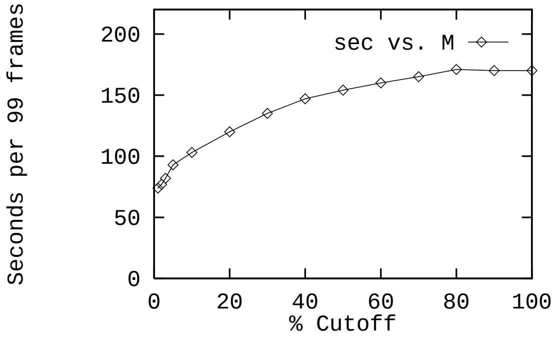


Figure 4: Experimental Results - Running Time vs. M Cutoff

DOF response. The main speedup comes from not having to update a large section of the image on each frame. The more expensive this unchanging area is to compute, the greater the improvement over the linear method. When objects are moving, $M$ should be set to 100 to ensure all moving pixels are updated.

This algorithm uses frame coherence to gain considerable speedups over previous methods. Obviously, we cannot gain improvements in situations where there is little or no frame coherence, such as a scene with a moving viewpoint. If a scene changes totally, all pixels must be updated on each pass. The importance ordering method must both remove and add DOF effects, making it slower than the linear method. This is not as much of constraint as it might first seem. First, static camera shots account for a very large number of cinematographic situations. The virtual reality-like "roaming viewpoint" is not as common as the static "talking-heads" shot. In the vast majority of 30 frame/second animations with a static camera, both object motion and focus motion are slow enough to leave substantial portions of the image unchanged between frames. Secondly, we can still use importance ordering to help us set up the correct focus parameters for a moving-camera shoot, even if the algorithm does not help us during actual camera motion. Finally, we can impose frame coherence on a scene by "morphing" a few sequential frames of a moving camera sequence to a common reference frame. By imposing frame coherence, even only for limited sequences of two or three frames, we can still gain substantial speedups at the cost of a small loss in accuracy.

## 7 Summary

This paper describes a new method for calculating DOF based upon the notion of importance of change. Our importance ordering DOF algorithm is based upon Potmesil and Chakravarty's postfiltering process. The importance ordering DOF method avoids recomputation of large areas of unchanged pixels, concentrating only on areas of importance. In multi- frame animations, this results in large speedups over linear DOF methods without loss in accuracy. We were able to animate an in-focus object moving between an extremely blurry foreground and background with subsecond frame update rates. In addition, DOF computation can be truncated early by skipping over less important areas. This gives even faster results, and allows the user to preview progress.

Importance ordering DOF brings DOF into the feasible range of real-time applications. Real-time DOF will allow trial-and-error camera focus adjustment, as well as more realistic rendering.

## References


[Boult 92] Boult, T. and Wolberg, G. "Correcting Chromatic Aberrations Using Image Warping". In *Image Understanding Workshop*, pages 363–377. Defence Advanced Research Projects Agency, 1992.

[Chen 87] Chen, Y. "Lens Effect on Synthetic Image Generation Based on Light Particle Theory". In *CG International 87*, pages 347–366. Computer Graphics, 1987.

[Cook 84] Cook, R., Porter, T., and Carpenter, L. "Distributed Ray Tracing". *Computer Graphics (Proc. SIGGRAPH)*, 18(3):137–145, July 1984.

[Cox 71] Cox, A. *Photographic Optics*. Focal Press, New York, New York, 1971.

[Haeberli 90] Haeberli, P. and Kurt, A. "The Accumulation Buffer: Hardware Support for High-Quality Rendering". *Computer Graphics (Proc. SIGGRAPH)*, 24(4):309–317, August 1990.

[Kingslake 92] Kingslake, R. *Optics in Photography*. SPIE Optical Engineering Press, Bellingham, Wash., 1992.

[Lee 90] Lee, H.-C. "Review of Image-Blur Models in a Photographic System Using the Principles of Optics". *Optical Engineering*, 5(29):405–421, May 1990.

[Pentland 87] Pentland, A. "A New Sense for Depth of Field". *IEEE Trans. Pattern Analysis and Machine Intelligence*, 9(4):523–531, July 1987.

[Potmesil 81] Potmesil, M. and Chakravarty, I. "A Lens and Aperture Camera Model for Synthetic Image Generation". *Computer Graphics (Proc. SIGGRAPH)*, 15(3):297–305, August 1981.

[Savakis 91] Savakis, A. and Trussell, H. "Restorations of Real Defocused Images Using Blur Models Based on Geometrical and Diffraction Optics". In *SOUTHEASTCON 1991*, volume 2, pages 919–922. IEEE, April 1991.

[Scofield 92] Scofield, C. $2\frac{1}{2}$ D Depth-of-Field Simulation for Computer Animation. In Kirk, D., editor, *Graphic Gems III*, pages 36–38. Academic Press Ltd, 1992.

[Sezan 91] Sezan, I., Pavlovic, G., Tekalp, M., and Erdem, T. "On Modelling the Focus Blur in Image Restoration". In *ICASSP '91: Acoustics, Speech and Signal Processing Conference*, volume 4, pages 2485–2488. IEEE, April 1991.

[Shinya 94] Shinya, M. "Post-filtering for Depth of Field Simulation with Ray Distribution Buffer". In *GI*, pages 59–66. Canadian Information Processing Society, 1994.

[vanWijk 92] vanWijk, J. "Rendering Surface Particles". In *Visualization 1992*, pages 54–61. IEEE, October 1992.

[Wixson 90] Wixson, S. "The Display of 3D MRI Data with Non-Linear Focal Depth Cues". In *Computers in Cardiology*, pages 379–380. IEEE, September 1990.

Figure 5: Intensity Leakage from Linear DOF Filtering

Figure 6: Reduced Intensity Leakage Through Z-Buffering

Figure 7: Target Scene Without DOF

Figure 8: Experiment 1 - Object Moves Laterally

Figure 9: Experiment 3 - Near to Far Focus Change

Figure 10: Importance Ordering DOF, M = 15 of Changing Pixels Recomputed