SDK White Paper

Depth of Field
Controlling Camera Parameters

WP-01396-001-v01
July 2004

Depth of Field Example

This document describes an example which allows control over several camera parameters such as camera position, camera orientation, focal length, focal distance, and f-stop, and renders the scene according to those parameters. In particular, it approximates depth of field, which means that objects which are in focus are rendered sharply while objects which are out of focus are blurred, while running in real-time.

The example is inspired by an mpeg trailer for a car-racing game. The trailer uses in-game video footage that is post-processed to add depth-of-field to make the visuals more dramatic. The technique shown here runs in real-time and applies to all racing games that have a replay option once the race is over. The many cameras used in replay mode would become more dramatic if depth of field effects were applied to them.

The menu options allow rendering in wire-frame, to view the per-pixel distance-to-camera, or to visualize the amount of blurriness. Left-click-hold and moving the mouse controls the camera's view-direction, and the arrow-keys move the camera forward, backward, left, and right. The keys **I** and **O** zoom in and out (i.e., change focal-length); **U** and **P** change the focus-distance, and **K** and **L** change the f-stop.

This example is written for DirectX 9.1 and requires PS/VS1.1 support.

Bryan Dudash
bdudash@nvidia.com

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050

Discussion

The effect renders the scene only once. During this rendering, vertex-shaders compute the per-vertex distance to the camera. This distance gets interpolated on a per-pixel basis; an associated pixel-shader transforms this distance into a "blurriness-interpolator" that is stored in the alpha component of the frame buffer. The frame-buffer is then filter-blitted multiple times to generate blurred versions. Finally, based on the "blurriness-interpolator" stored in the alpha-component you can blend between the original in-focus frame-buffer and its various blurred versions to derive the final result. This technique is very similar to the algorithm described by M. Potmesil and I. Chakravarty in "A lens and aperture camera model for synthetic image generation," Computer Graphics (Proceedings of SIGGRAPH 81), *15 (3)*, pp. 297-305 (August 1981, Dallas, Texas).

A look-up is performed for the "blurriness-interpolator" in a texture. This texture is currently parameterized by distance to camera, and focus distance. Hence, when changing the f-stop, this texture is regenerated. If f-stops change frequently you could re-parameterize the texture accordingly. Alternatively, you could operate on a 1D texture, parameterized by distance to camera only, and regenerate it every time any of the other parameters change – since the size of this 1D texture would be 1Kb approximately in real-time.

Depth of Field Algorithm

1. Render Scene to texture
   a) In Vertex Shader, calculate the vertex distance from camera
   b) In Pixel Shader, convert distance factor to blurriness factor and store in alpha channel. Also lookup and store "circle of confusion" interpolant based on camera distance.
2. Generate Blurred versions (3 filter targets)
   Looping for a certain number of filter steps (5 this sample), blur the filter target using the previous filter target as the initial source texture.

   Note: This uses a couple temporary textures, since you can't source and sink the same texture in a pixel shader.

3. Source filter targets, and render a full screen quad, interpolating the color between the three filter targets using the circle of confusion value stored in alpha.

The Circle of Confusion

The circle of confusion is actually two circles representing the interpolation endpoints of three filter targets. You can compute the circle of confusion on the CPU and enter it into a look-up table in a texture. If volume textures are available, you can enter values in multiple look-up tables based on focus distance into the volume texture. Otherwise, you have to recalculate the look-up table whenever the focal length changes.

The **x**-dimension represents distance to the camera.

The **y**-dimension represents focus distance. Min and Max are constant

The **z**-dimension represents focal length. Min and Max are constant, and if not using volumes, the focal length value used in the circle calculation is set from user input.

$$value = abs\left(\frac{\left(FocusDist / CameraDist - 1 \right) * FocalLength^2}{FStop * \left(FocusDist - FocalLength \right)} \right)$$

Equation 1. Circle of Confusion Table Equation

Essentially, you have to figure out what the **min-** and **max-distance** is for the depth of field for a circle of confusion of ratio**2*c0**. Anything *blurrier* than that cannot be represented anyway, so there is no need to waste interpolator-range on it. Then you can iterate from **min-** to **max-distance** and compute the circle of confusion for that particular distance.

The circle of confusion diameter computed above is then mapped into an interpolator ranging from **0** to **1**, with **0** corresponding to circles of diameter **c0** or less, **0.5** corresponding to diameters ratio**1*c0,** and **1** corresponding to diameters ratio**2*c0** or more.
All formulas are from *"Photographic Lenses Tutorial"* by David M. Jacobson (www.graflex.org/lenses/photographic-lenses-tutorial.html).

References

M. Potmesil and I. Chakravarty, "*A lens and aperture camera model for synthetic image generation*," Computer Graphics (Proceedings of SIGGRAPH 81), *15 (3)*, pp. 297-305 (August 1981, Dallas, Texas).

David M. Jacobson, "*Photographic Lenses Tutorial*" (www.graflex.org/lenses/photographic-lenses-tutorial.html).

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com