Advanced Depth of Field

Thorsten Scheuermann
3D Application Research Group
ATI Research, Inc.

Overview

- Background
- Using destination alpha for depth and blur information
- Scene rendering
- Post-processing
- Demo

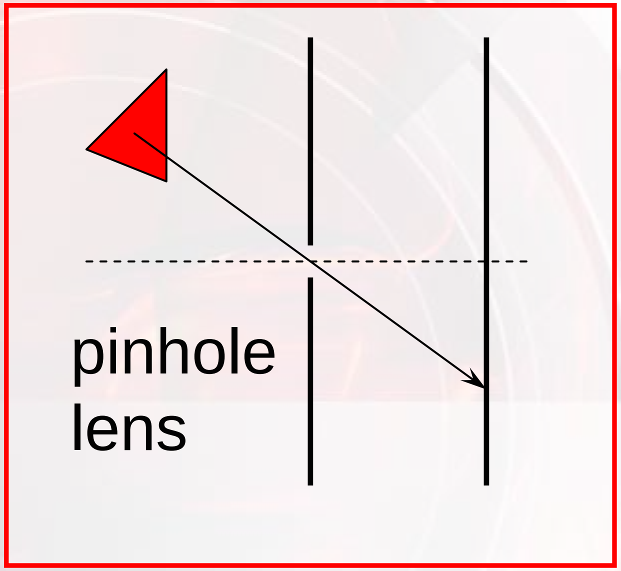
- This depth of field technique is an improvement of a previous technique developed at ATI
  - Doesn't require multiple render targets
  - Better anti-aliasing

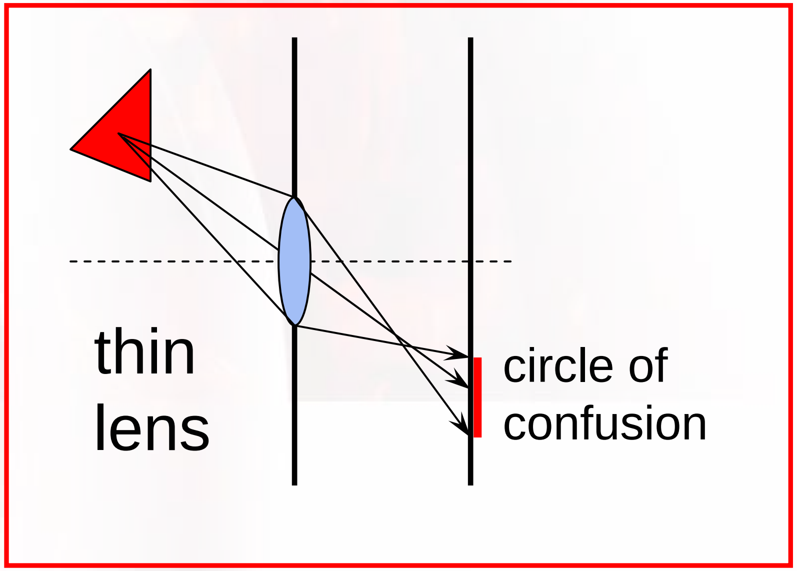
Depth Of Field

- Depth of Field causes out-of-focus objects to appear blurry
- Computer graphics uses pinhole camera model
  - Results in perfectly sharp images
  - See Potmesil and Chakravarty 1981, among others
- Real cameras use lenses with finite dimensions
  - This is what causes depth of field

- Important part of cinematic visual vocabulary
- Fundamental to photo-realistic rendering
- Give control to your artists! Let them control and animate parameters of your camera
  - Probably only reasonable for in-engine cut-scenes

Camera Models

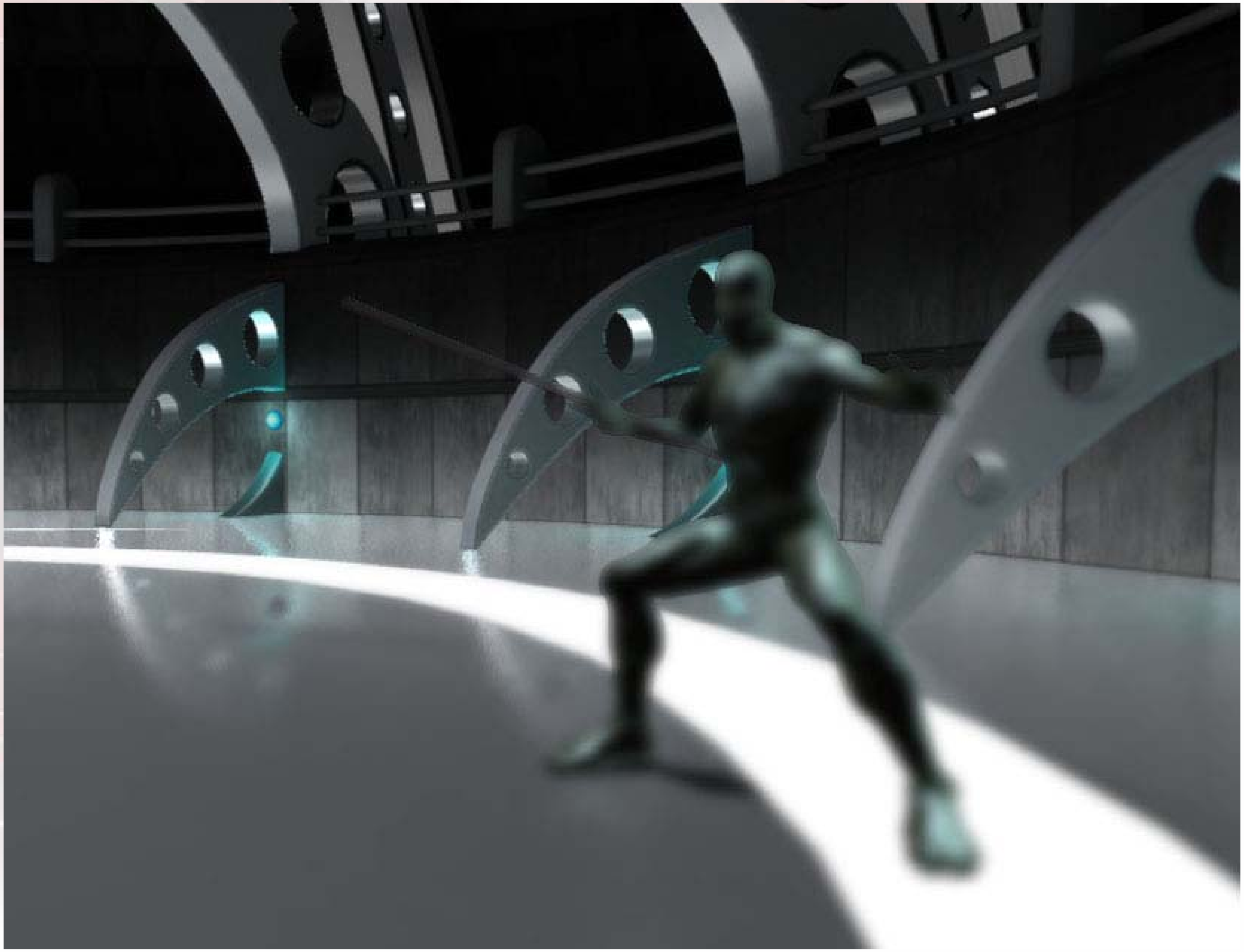




- Pinhole lens lets only a single ray through
- In thin lens model, if point isn't in focal plane, multiple rays contribute to the image
- Intersection of rays with image plane approximated by circle

Depth Of Field

Depth Of Field



Depth of Field Implementation

- Use destination alpha channel to store per-pixel depth and blurriness information
- Pixel shaders for post-processing
  - Downsample and pre-blur the image
  - Use variable size filter kernel to approximate circle of confusion
  - Blend between original and pre-blurred image for better image quality
  - Take measures to prevent “leaking” sharp foreground into blurry background

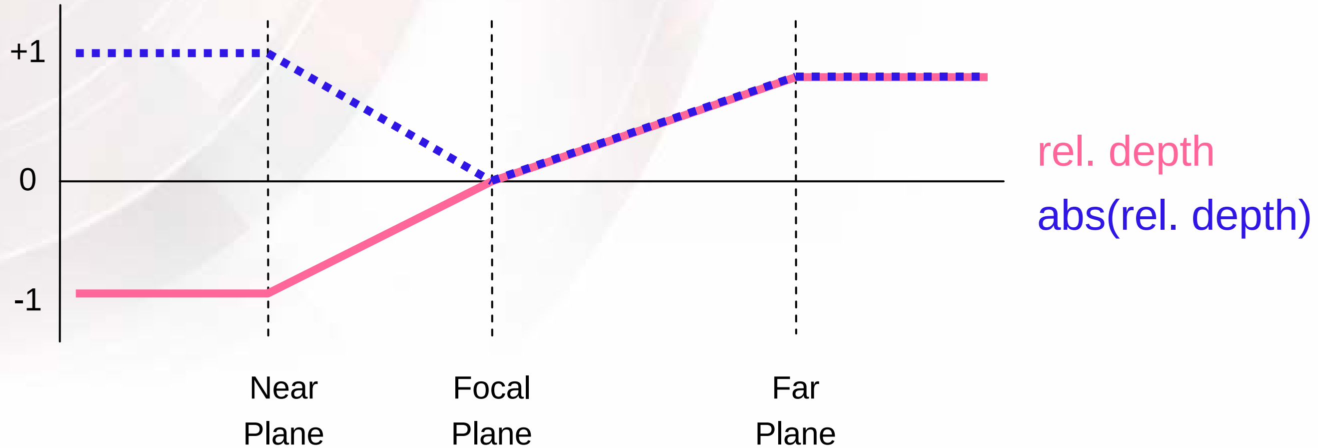
This is new!

Populating Destination Alpha

- The post-processing shader needs blurriness and relative depth of each pixel
- We pass the camera distance of three planes to scene shaders:
  - Focal plane: Points on this plane are in focus
  - Near plane: Everything closer than this is fully blurred
  - Far plane: Everything beyond the far plane is fully blurred
- Each object's pixel shader renders depth and blurriness information into destination alpha

Mapping Depth to Blurriness

- Map a point's camera depth to [-1, 1] range as shown in pink graph
  - This gives us relative depth
- To get blurriness, just take the absolute value
- Scale and bias relative depth into [0, 1] range before writing to destination alpha
  - Saves us from writing blurriness and depth into two separate channels



HLSL Code for Alpha Output

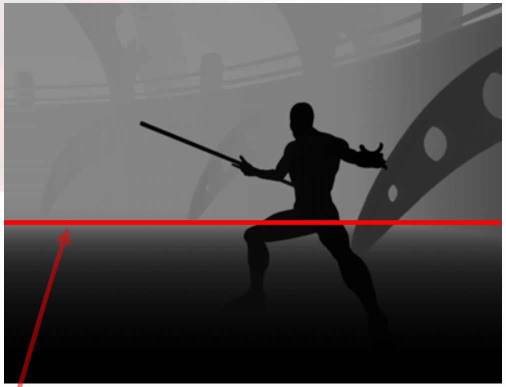
```
// vDofParams coefficients:
// x = near blur depth; y = focal plane depth; z = far blur depth
// w = blurriness cutoff constant for objects behind the focal plane
float4 vDofParams;

float ComputeDepthBlur (float depth /* in view space */)
{
   float f;

   if (depth < vDofParams.y)
   {
      // scale depth value between near blur distance and focal distance to
      // [-1, 0] range
      f = (depth - vDofParams.y)/(vDofParams.y - vDofParams.x);
   }
   else
   {
      // scale depth value between focal distance and far blur distance to
      // [0, 1] range
      f = (depth - vDofParams.y)/(vDofParams.z - vDofParams.y);
      // clamp the far blur to a maximum blurriness
      f = clamp (f, 0, vDofParams.w);
   }
   // scale and bias into [0, 1] range
   return f * 0.5f + 0.5f;
}
```

All pixel shaders write the result of **`ComputeDepthBlur()`** to destination alpha.

Destination Alpha Example

3m focal plane

6m focal plane



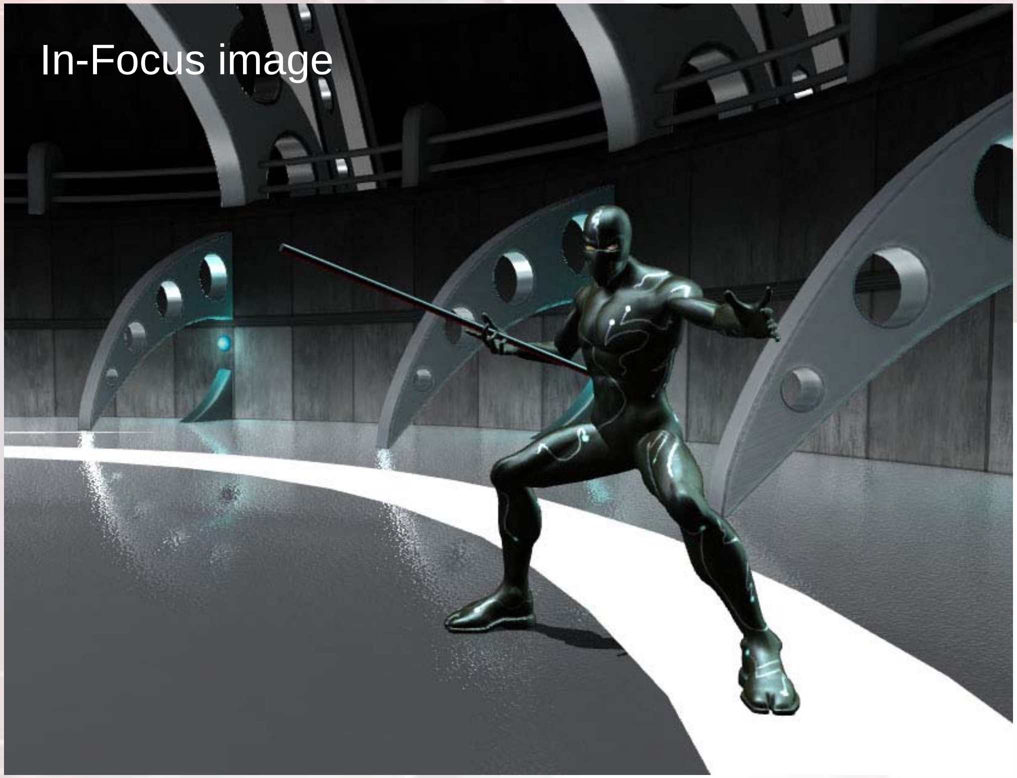
12m focal plane

This is where the focal plane intersects with the floor

Dealing with Alpha Blending

- Even though we use destination alpha for blur information, we can still do alpha-blending
- 1st pass:
  - Render only to RGB with blending enabled
- 2nd pass:
  - Render output of **`ComputeDepthBlur()`** only to destination alpha

Post-Processing: Pre-blurring the Image



1/16th Size

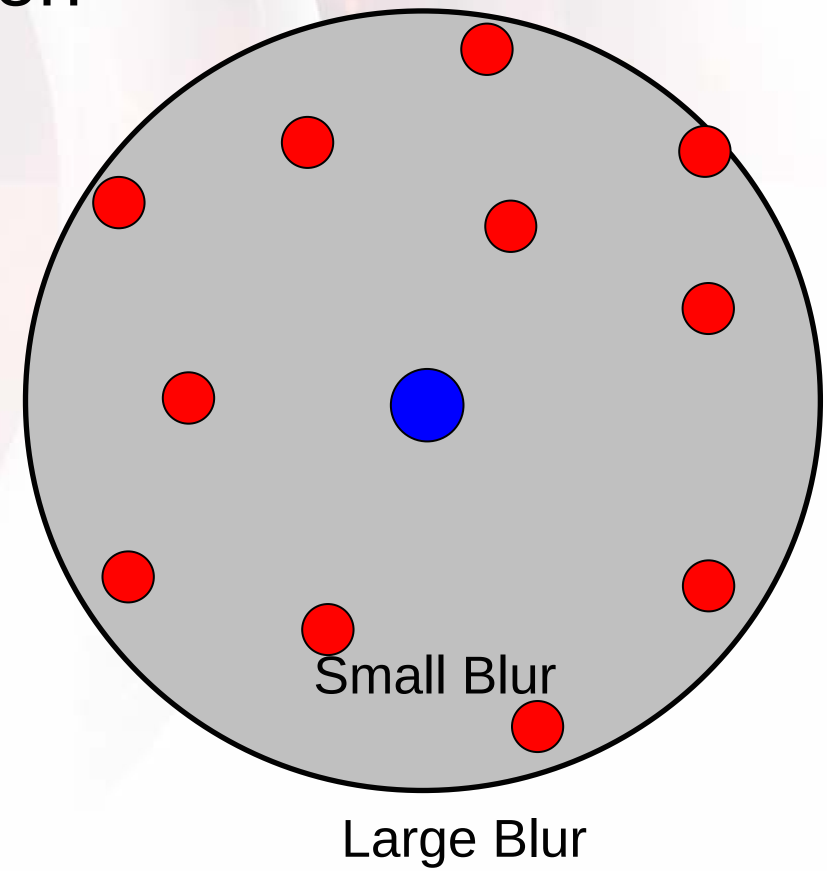
3×3 Gaussian Blur

MSAA image from back buffer

(Destination alpha contains blurriness)

Circle Of Confusion Filter Kernel

- Stochastic sampling
- Poisson distribution

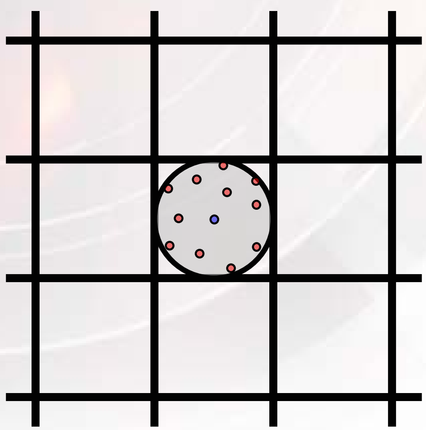


Center Sample

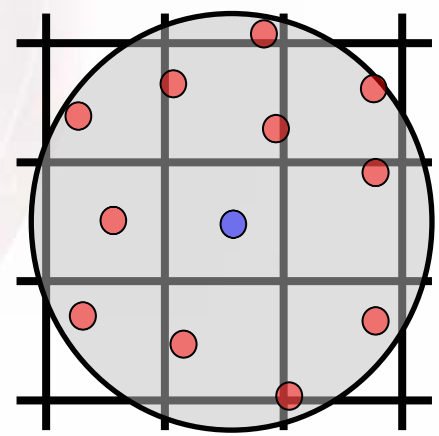
Outer Samples

Filter Kernel For Circle Of Confusion

- Vary kernel size based on the “blurriness” factor
- Sample all taps from original and pre-blurred image
  - Blend between them based on tap blurriness



Point in focus



Point is blurred

Reduction Of "Leaking"

- Conventional post-processing blur techniques cause "leaking" of sharp foreground objects onto blurry backgrounds
- Depth compare the samples and discard ones that can contribute to background "leaking"

Depth Of Field Shader

- Variables used in the HLSL function:

```
#define NUM_TAPS  8   // number of taps the shader will use

sampler tSource;      // full resolution image
sampler tSourceLow;   // downsampled and filtered image

float2 poisson[NUM_TAPS];  // contains poisson-distributed positions on the
                           // unit circle

float2 pixelSizeHigh; // pixel size (1/image resolution) of full resolution image
float2 pixelSizeLow;  // pixel size of low resolution image

float2 vMaxCoC = float2(5.0, 10.0); // maximum circle of confusion (CoC) radius
                                    // and diameter in pixels

float radiusScale = 0.4; // scale factor for maximum CoC size on low res. image
```

```
float4 PoissonDOFFilter (float2 texCoord /* screen-space quad texture coords*/)
{
  float4 cOut;
  float discRadius, discRadiusLow, centerDepth;

  cOut = tex2D (tSource, texCoord);  // fetch center tap
  centerDepth = cOut.a;              // save its depth

  // convert depth into blur radius in pixels
  discRadius = abs (cOut.a * vMaxCoC.y - vMaxCoC.x);
  discRadiusLow = discRadius * radiusScale; // compute radius on low-res image
  cOut = 0;                                 // reusing cOut to accumulate samples

  for (int t = 0; t < NUM_TAPS; t++)
  {
    // compute tap texture coordinates
    float2 coordLow = texCoord + (pixelSizeLow * poisson[t] * discRadiusLow);
    float2 coordHigh = texCoord + (pixelSizeHigh * poisson[t] * discRadius);

    // fetch high-res tap
    float4 tapLow = tex2D (tSource, coordLow);
    float4 tapHigh = tex2D (tSource, coordHigh);

    // mix low- and hi-res taps based on tap blurriness
    float tapBlur = abs (tapHigh.a * 2.0 - 1.0);  // put blurriness into [0, 1]
    float4 tap = lerp (tapHigh, tapLow, tapBlur);

    // "smart" blur ignores taps that are closer than the center tap and in focus
    tap.a = (tap.a >= centerDepth) ? 1.0 : abs (tap.a * 2.0 - 1.0);

    cOut.rgb += tap.rgb * tap.a;       // accumulate
    cOut.a += tap.a;
  }
  return (cOut / cOut.a);
}
```

Demo

Conclusion

- Depth of field technique produces a convincing photorealistic visual cue
- Use destination alpha for depth and blur information
- Post-processing does the heavy lifting

References

- M. Potmesil, I. Chakravarty, *"A lens and aperture camera model for synthetic image generation"*. Computer Graphics (Proceedings of SIGGRAPH 81). 15 (3), pp. 297-305, 1981.
- G. Riguer, N. Tatarchuk, J. Isidoro, *"Real-Time Depth of Field Rendering"*. ShaderX2