VGP352 – Week 8

- Agenda:
  - Texture rectangles
  - Post-processing
    - Full-screen post-processing overview
    - Filter kernels
      - Separable filters
    - Special effects
      - Water ripple
      - Depth of field

Texture Rectangle

- Cousin to 2D textures
  - Interface changes:
    - New texture target: `GL_TEXTURE_RECTANGLE_ARB`
    - New sampler type: `sampler2DRect`, `sampler2DRectShadow`
    - New sampler functions: `texture2DRect`, `texture2DRectProj`, etc.
  - Limitations:
    - No mipmaps
    - Minification filter must be `GL_LINEAR` or `GL_NEAREST`
    - Wrap mode must be one of `GL_CLAMP_TO_EDGE`, `GL_CLAMP_TO_BORDER`, or `GL_CLAMP`

Texture Rectangle

- Added features:
  - Dimensions need not be power of two
    - Alas, now only a “feature” on old hardware
  - Accessed by non-normalized coordinates
    - Coordinates are $[0, w] \times [0, h]$

Post-processing Effects

- Apply an *image space* effect to the rendered scene *after* it has been drawn
  - Examples:
    - Blur
    - Enhance contrast
    - Heat “ripple”
    - Color-space conversion (e.g., black & white, sepia, etc.)
    - Many, *many* more

Post-processing Effects

- Overview:
  - Render scene to off-screen target (framebuffer object)
    - Off-screen target should be same size as on-screen window
    - Additional information may need to be generated
  - Render single, full-screen quad to window
    - Use original off-screen target as source texture
    - Configure texture coordinates to cover entire texture
      - Texture rectangles are *really* useful here
    - Configure fragment shader to perform desired effect

Post-processing Effects

- Configure projection matrix to remap [0, 0] × [w, h] to [-1, 1] × [-1, 1] with parallel perspective

$$\begin{bmatrix} \frac{2}{width} & 0 & 0 & -1 \\ 0 & \frac{2}{height} & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  – This is the same as the old `glOrtho` function

Post-processing Effects

- Draw two full-screen triangles
  - Use pixel coordinates for both vertex positions and texture coordinates
  - This assumes texture rectangles are being used

Post-processing Effects

- May need to access many neighbor texels in the fragment shader
  - Can calculate these coordinates in the fragment shader, but this uses valuable instructions
  - Instead use all of the available varying slots and pre-calculate offset coordinates in the vertex shader
    - Query `GL_MAX_VARYING_FLOATS` to determine how many slots are available

Post-processing Effects

- Offset texel locations can also be accessed with `textureOffset` and friends

```
vec4 textureOffset(sampler2D s, vec2 p,
    ivec2 offset);
```

  - Integer offset must be known at *compile* time
  - Requires GLSL 1.30.
  - Available with `EXT_gpu_shader4` as `texture2DOffset`, `texture2DRectOffset`, etc.

Filter Kernels

- Can represent our filter operation as a sum of products over a region of pixels
  - Each pixel is multiplied by a factor
  - Resulting products are accumulated
- Commonly represented as an $n \times m$ matrix
  - This matrix is called the *filter kernel*
  - m is either 1 or is equal to n

Filter Kernels

- Uniform blur over 3x3 area:
  - Larger kernel size results in more blurriness

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

24-November-2010

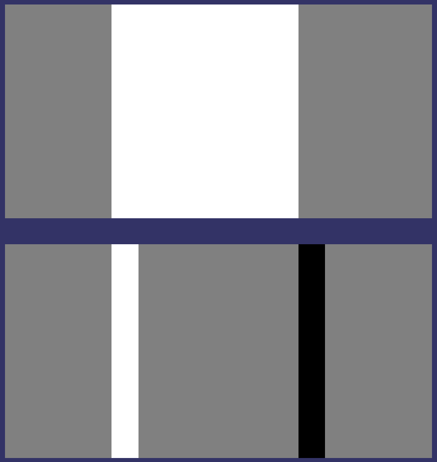
Filter Kernels – Edge Detection

- Edge detection

Filter Kernels – Edge Detection

- Edge detection
  - Take the difference of each pixel and its left neighbor

$$p(x,y) - p(x-1,y)$$



24-November-2010

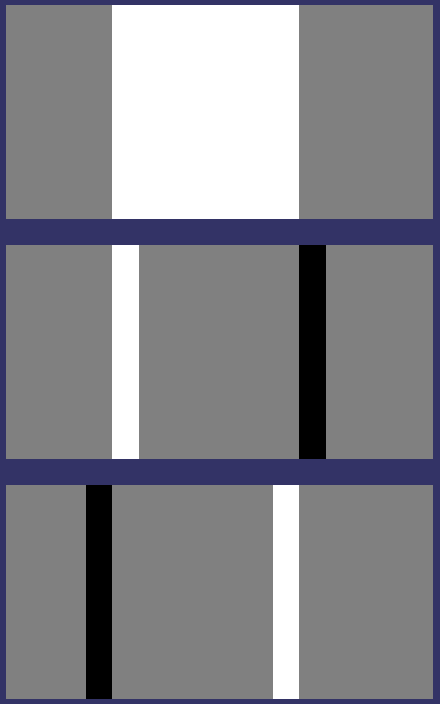
Filter Kernels – Edge Detection

- Edge detection
  - Take the difference of each pixel and its left neighbor

    $$p(x, y) - p(x-1, y)$$

  - Take the difference of each pixel and its right neighbor

    $$p(x, y) - p(x+1, y)$$



Filter Kernels – Edge Detection

- Edge detection
  - Take the difference of each pixel and its left neighbor

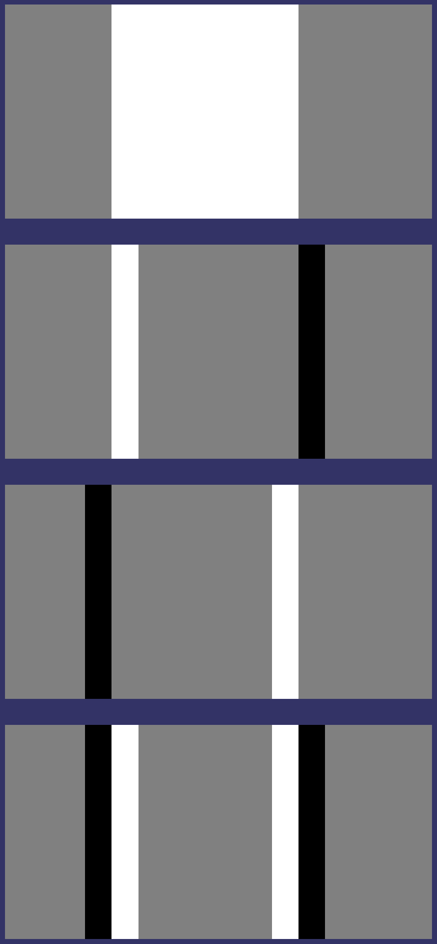
$$p(x,y)-p(x-1,y)$$

  - Take the difference of each pixel and its right neighbor

$$p(x,y)-p(x+1,y)$$

  - Add the two together

$$2p(x,y)-p(x-1,y)-p(x+1,y)$$



Filter Kernels – Edge Detection

- Rewrite as a kernel

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 2 & -1 \\ 0 & 0 & 0 \end{bmatrix}$$

Filter Kernels – Edge Detection

- Rewrite as a kernel

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 2 & -1 \\ 0 & 0 & 0 \end{bmatrix}$$

- Repeat in Y direction

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Filter Kernels – Edge Detection

- Rewrite as a kernel

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 2 & -1 \\ 0 & 0 & 0 \end{bmatrix}$$

- Repeat in Y direction

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

- Repeat on diagonals

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Sobel Edge Detection

- Uses two filter kernels
  - One in the Y direction

$$F_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

  - One in the X direction

$$F_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

Sobel Edge Detection

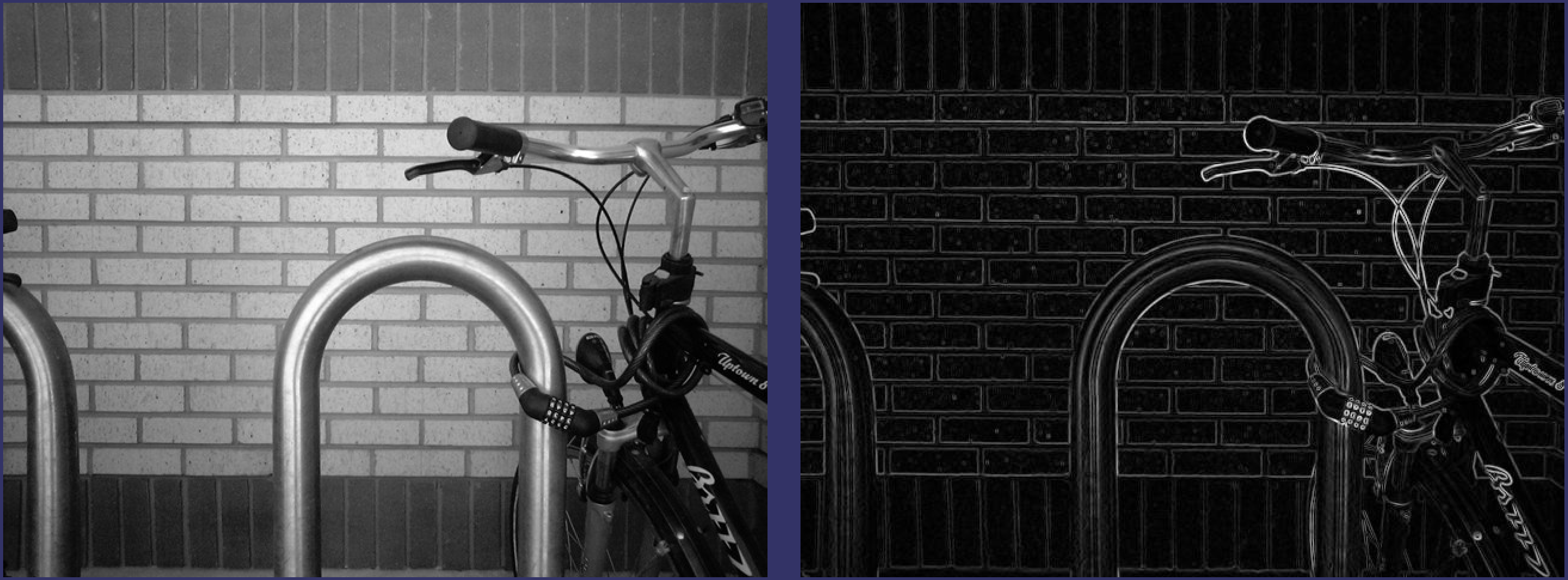
- Apply each filter kernel to the image

$$G_x = F_x * A$$
$$G_y = F_y * A$$

  - G_x and G_y are the gradients in the x and y directions
  - The combined magnitude of these gradients can be used to detect edges

$$G = \sqrt{G_x^2 + G_y^2}$$

Sobel Edge Detection



Images from http://en.wikipedia.org/wiki/Sobel_operator

24-November-2010

Filter Kernels

- Implement this easily on a GPU
  - Supply filter kernel as uniforms
  - Perform n^2 texture reads
  - Apply kernel and write result

Filter Kernels

- Implement this easily on a GPU
  - Supply filter kernel as uniforms
  - Perform n^2 texture reads
  - Apply kernel and write result
- Perform n^2 texture reads?!?

Filter Kernels

- Implement this easily on a GPU
  - Supply filter kernel as uniforms
  - Perform n^2 texture reads
  - Apply kernel and write result
- Perform n^2 texture reads?!?
  - n larger than 4 or 5 won't work on most hardware
  - Since the filter is a sum of products, it could be done in multiple passes

Filter Kernels

- Implement this easily on a GPU
  - Supply filter kernel as uniforms
  - Perform n^2 texture reads
  - Apply kernel and write result
- Perform n^2 texture reads?!?
  - n larger than 4 or 5 won't work on most hardware
  - Since the filter is a sum of products, it could be done in multiple passes
  - Or *maybe* there's a different way altogether...

Separable Filter Kernels

- Some 2D kernels can be re-written as the product of 2 1D kernels
  - These kernels are called *separable*
  - Applying each 1D kernel requires n texture reads per pixel, doing both requires $2n$
    - $2n \ll n^2$

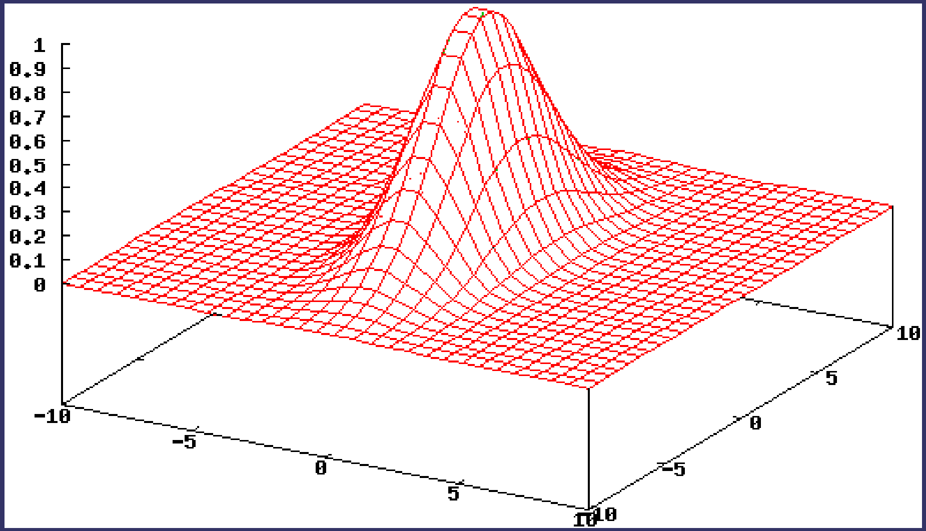
Separable Filter Kernels

- 2D kernel is calculated as the outer-product of the individual 1D kernels

$$\mathbf{A} = \mathbf{a}^T \mathbf{b} = \begin{bmatrix} \mathbf{a}_0 \mathbf{b}_0 & \cdots & \mathbf{a}_0 b_n \\ \vdots & & \vdots \\ \mathbf{a}_n \mathbf{b}_0 & \cdots & \mathbf{a}_n \mathbf{b}_n \end{bmatrix}$$

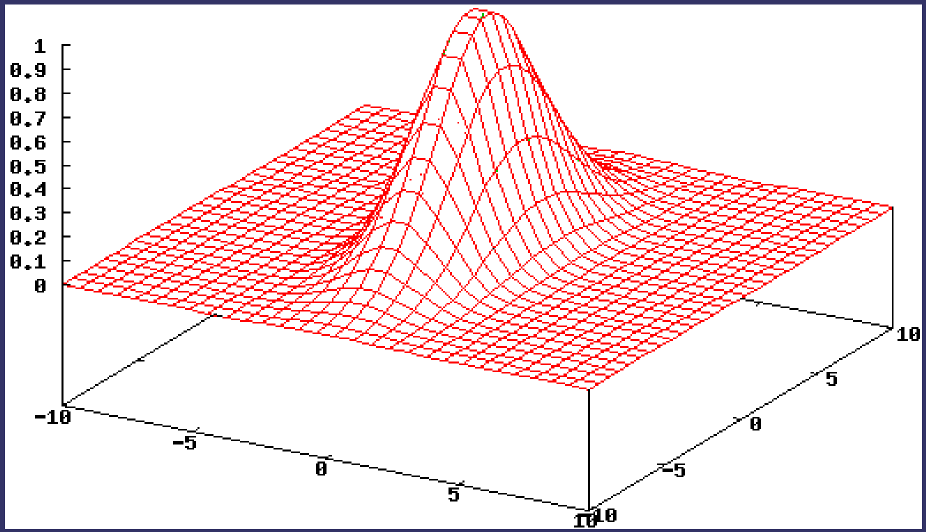
Separable Filter Kernels

- The 2D Gaussian filter is *the classic* separable filter

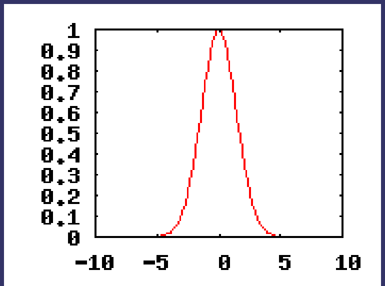


Separable Filter Kernels

- The 2D Gaussian filter is *the classic* separable filter

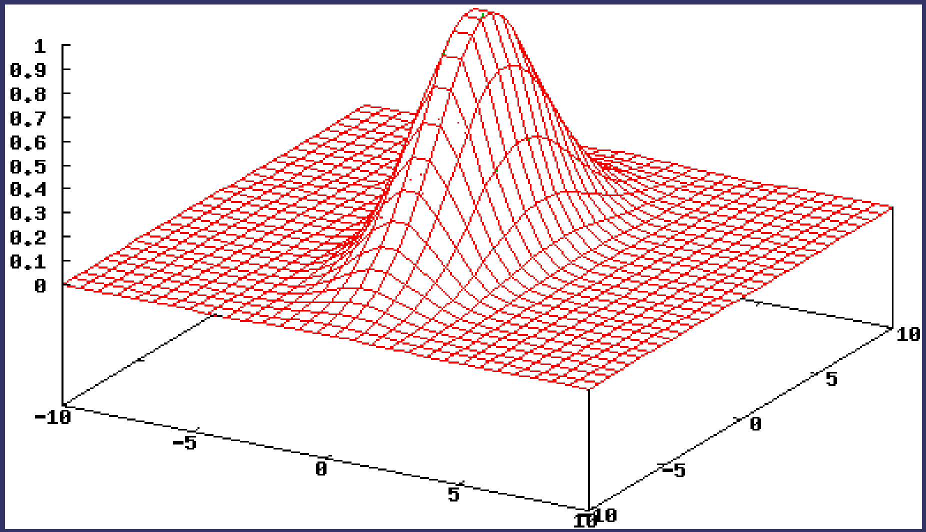


  - Product of a Gaussian along the X-axis

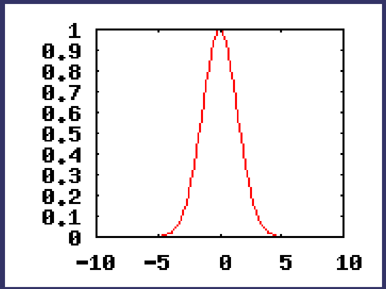


Separable Filter Kernels

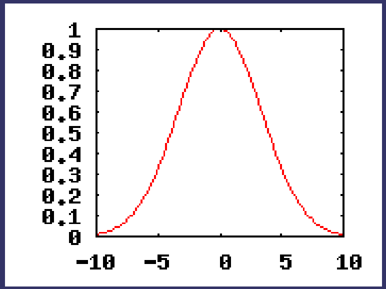
- The 2D Gaussian filter is *the classic* separable filter



  - Product of a Gaussian along the X-axis



  - ...and a Gaussian along the Y-axis



Separable Filter Kernels

- Implementing on a GPU:
  - Use first 1D filter on source image to *temporary image*
  - Use second 1D filter on *temporary image to window*

Separable Filter Kernels

- Implementing on a GPU:
  - Use first 1D filter on source image to *temporary image*
  - Use second 1D filter on *temporary image to window*
- Caveats:
  - Precision can be a problem in intermediate steps
  - May have to use floating-point output
  - Can also use 10-bit or 16-bit per component outputs as well
    - Choice ultimately depends on what the hardware supports

References

http://www.archive.org/details/Lectures_on_Image_Processing

Ripple Effect

Note the frame-to-frame difference

Image from Enemy Territory: Quake Wars, © Copyright 2007 id Software, Inc.

Ripple Effect

- Render multiple passes:
  1) Render scene normally to one texture
  2) Render water surface to a separate texture
     - Instead of color, render a perturbation vector
     - Clear color is a perturbation vector of {0, 0}
  3) Render final scene by using water texture to select texels from scene texture
  4) Render water over final scene

Ripple Effect

Note the bleeding of out-of-water elements into the ripples

Image from Enemy Territory: Quake Wars, © Copyright 2007 id Software, Inc.

Optimization

- Multiple texture look-ups for every pixel can be expensive

Optimization

- Multiple texture look-ups for every pixel can be expensive
  - Can render “effect area” to stencil buffer
  - Perform combine step in two passes:
    - First pass just copies areas where stencil is not set
    - Second pass performs effect in areas where stencil is set
  - Can be extended to select multiple screen-space effects using different stencil values

References

Tutorials for several post-processing effects:

http://www.geeks3d.com/20091116/shader-library-2d-shockwave-post-processing-filter-glsl/

Depth-of-field

- What is depth of field?
  - "...the depth of field (DOF) is the portion of a scene that appears acceptably sharp in the image.[1]"

[1] http://en.wikipedia.org/wiki/Depth_of_field
Images also from http://en.wikipedia.org/wiki/Depth_of_field
24-November-2010

Depth-of-field

- Why is DOF important?

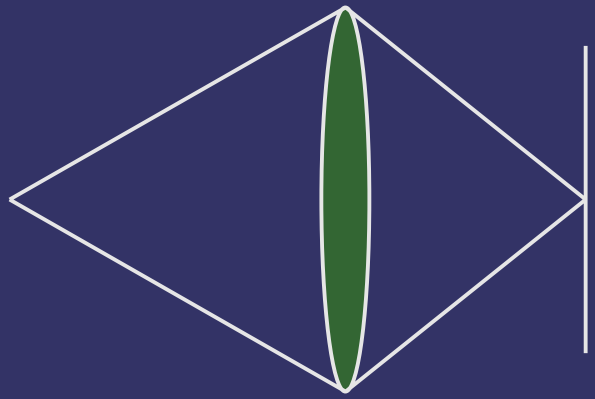
Images from http://en.wikipedia.org/wiki/Depth_of_field

Depth-of-field

- Why is DOF important?
  - Draws viewer's attention
  - Gives added information about spatial relationships
  - etc.

Images from http://en.wikipedia.org/wiki/Depth_of_field

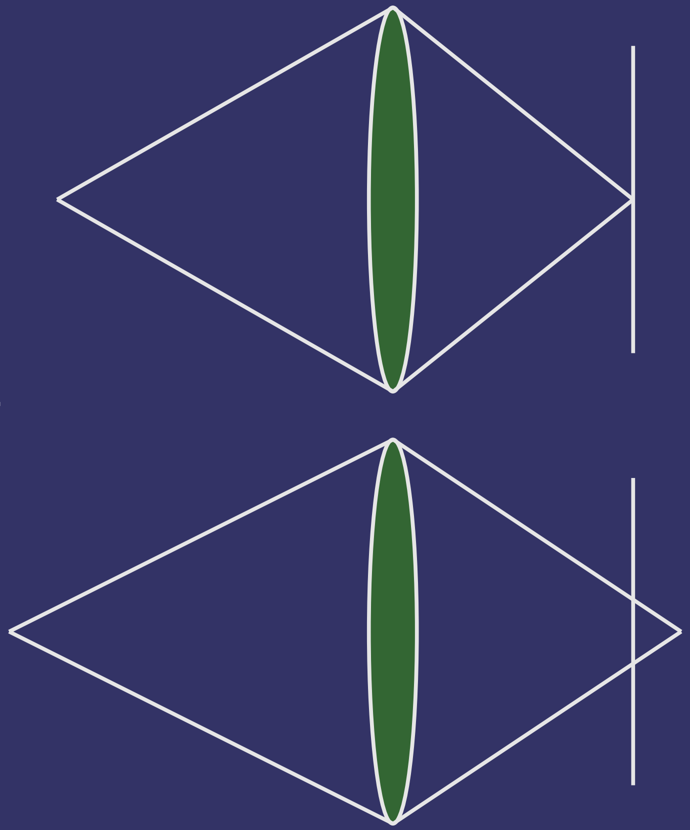
Depth-of-field



- Basic optics:
  - A point of light focused through a lens becomes a point on the object plane

Depth-of-field

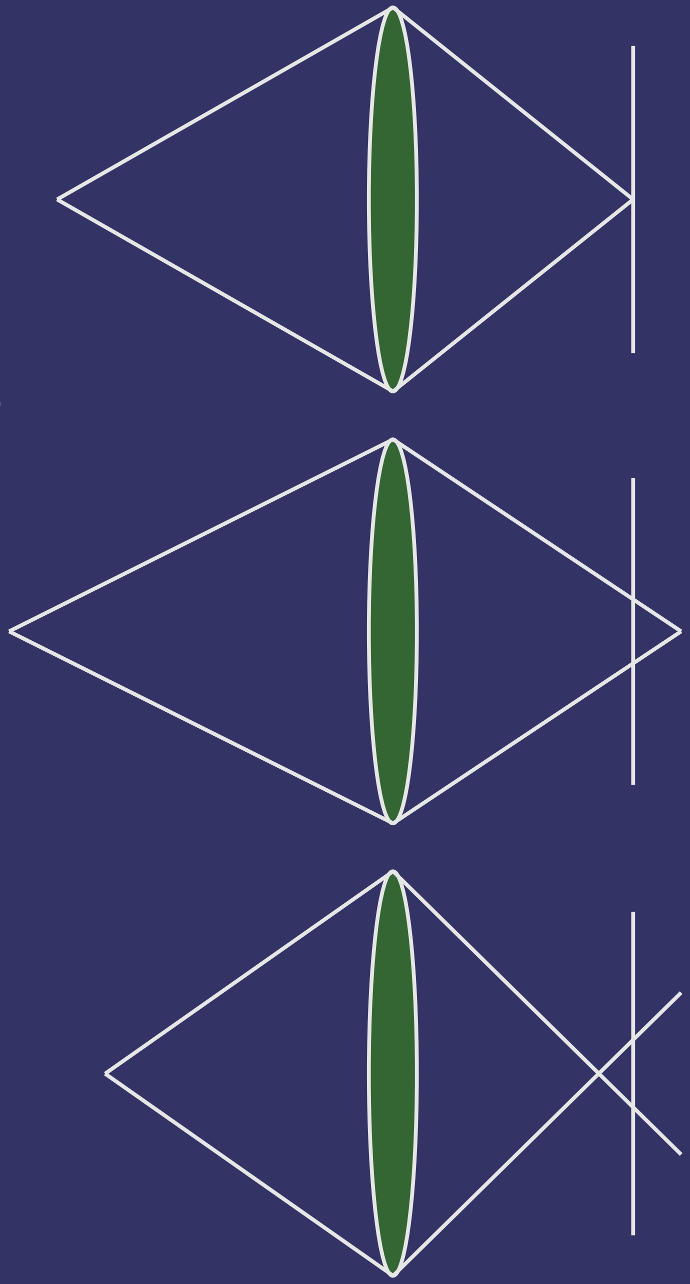
- Basic optics:
  - A point of light focused through a lens becomes a point on the object plane
  - A point farther than the focal distance becomes a blurry spot on the object plane



24-November-2010

Depth-of-field

- Basic optics:
  - A point of light focused through a lens becomes a point on the object plane
  - A point farther than the focal distance becomes a blurry spot on the object plane
  - A point closer than the focal distance becomes a blurry spot on the object plane
- These blurry spots are called *circles of confusion* (CoC hereafter)



Depth-of-field

- In most real-time graphics, there is no depth-of-field
  - Everything is perfectly in focus all the time

Depth-of-field

- In most real-time graphics, there is no depth-of-field
  - Everything is perfectly in focus all the time
  - Most of the time this is okay
    - The player may want to focus on foreground and background objects in rapid succession. Without eye tracking, the only way this works is to have everything in focus.

Depth-of-field

- In most real-time graphics, there is no depth-of-field
  - Everything is perfectly in focus all the time
  - Most of the time this is okay
    - The player may want to focus on foreground and background objects in rapid succession. Without eye tracking, the only way this works is to have everything in focus.
  - Under some circumstances, DOF can be a *very* powerful tool
    - Non-interactive sequences
    - Special effects
      - Very effective use in the game Borderlands

Depth-of-field

- Straight-forward GPU implementation:
  - Render scene color *and* depth information to off-screen targets
  - Post-process:
    - At each pixel determine CoC size based on depth value
    - Blur pixels within circle of confusion
      - To prevent in-focus data from bleeding into out-of-focus data, do *not* use in-focus pixels that are closer than the center pixel

Depth-of-field

- Problem with this approach?

Depth-of-field

- Problem with this approach?
  - Fixed number of samples within CoC
    - Oversample for small CoC
    - Undersample for large CoC
  - Could improve quality with multiple passes, but performance would suffer

Depth-of-field

- Simplified GPU implementation:
  - Render scene color *and* depth information to off-screen targets
  - Post-process:
    - Down-sample image and Gaussian blur down-sampled image
      - Reduced size and filter kernel size are selected to produce maximum desired CoC size
    - Linearly blend between original image and blurred image based on per-pixel CoC size

Depth-of-field

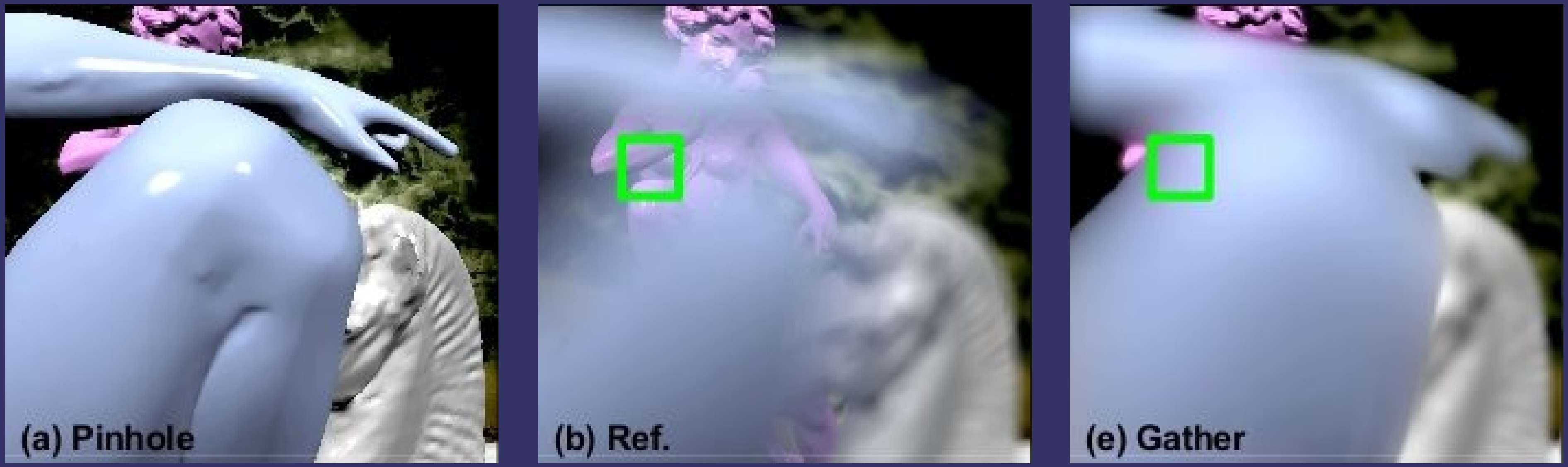
- Simplified GPU implementation:
  - Render scene color *and* depth information to off-screen targets
  - Post-process:
    - Down-sample image and Gaussian blur down-sampled image
      - Reduced size and filter kernel size are selected to produce maximum desired CoC size
    - Linearly blend between original image and blurred image based on per-pixel CoC size
- Problems with this approach?

Depth-of-field

- Simplified GPU implementation:
  - Render scene color *and* depth information to off-screen targets
  - Post-process:
    - Down-sample image and Gaussian blur down-sampled image
      - Reduced size and filter kernel size are selected to produce maximum desired CoC size
    - Linearly blend between original image and blurred image based on per-pixel CoC size
- Problems with this approach?
  - No way to prevent in-focus data from bleeding into out-of-focus data

Depth-of-Field

- “Gather” methods can't make objects obscured in the single image be visible in the blurred image



Images from [Lee 2009]

24-November-2010

References

J. D. Mulder, R. van Liere. *Fast Perception-Based Depth of Field Rendering*, In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology* (Seoul, Korea, October 22 - 25, 2000). VRST '00. ACM, New York, NY, 129-133. http://homepages.cwi.nl/~mullie/Work/Pubs/publications.html

Guennadi Riguer, Natalya Tatarchuk, John Isidoro. *Real-time Depth of Field Simulation*, In *ShaderX2*, Wordware Publishing, Inc., October 25, 2003. http://developer.amd.com/documentation/reading/pages/ShaderX.aspx

M. Kass, A. Lefohn, J. Owens. 2006. *Interactive Depth of Field Using Simulated Diffusion on a GPU.* Technical Memo #06-01, Pixar Animation Studios. http://graphics.pixar.com/library/DepthOfField/

Sungkil Lee, Elmar Eisemann, and Hans-Peter Seidel. 2009. Depth-of-field rendering with multiview synthesis. ACM Transactions on Graphics. 28, 5, Article 134 (December 2009). http://www.mpi-inf.mpg.de/~slee/pub/

Next week...

- Quiz #4
- Beyond bumpmaps:
  - Relief textures
  - Parallax textures
  - Interior mapping

Legal Statement