Post Processing Effects

- Applied to a frame post (after) rendering.
Post Processing Effects

- Applied to a frame **post** (after) rendering.
Post Processing Effects

- Applied to a frame **post** (after) rendering.
Post Processing Effects

- Applied to a frame post (after) rendering.
- Some could be applied in the initial pass.
Post Processing Effects

- Lots of different effects!

- Anti-aliasing
- Eye Adaption
- Blendables (lerp)
- Color Grading
- Filmic Tonemapper
- Depth of Field
- Panini Projection

- Chromatic Aberration
- Screen Space Reflections
- Vignette
- Bloom
- ...

https://docs.unrealengine.com/en-us/Engine/Rendering/PostProcessEffects
Post Processing Effects

- Lots of different effects!
  - Anti-aliasing
  - Eye Adaption
  - Fog
  - Color Grading
  - Grain
  - Depth of Field
  - Ambient Occlusion
  - Chromatic Aberration
  - Screen Space Reflections
  - Vignette
  - Bloom
  - Dithering
  - Motion Blur
  - ...

https://docs.unity3d.com/Manual/PostProcessingOverview.html
Blurs and Convolution
Blur

- Every pixel becomes a combination of pixels from a neighbourhood.
Blur

- Every pixel becomes a combination of pixels from a neighbourhood.
- Different possibilities for combinations.

Box blur

Gaussian blur
Convolution

- Operation between 2 signals.
Convolution

- Operation between 2 discrete signals.
Convolution

• Operation between 2 discrete 2D signals.
Convolution

- Operation between 2 matrices.

\[
\begin{pmatrix}
  k_{0,0} & \cdots & k_{p,0} \\
  \vdots & \ddots & \vdots \\
  k_{0,s} & \cdots & k_{p,s}
\end{pmatrix}
\ast
\begin{pmatrix}
  p_{0,1} & p_{0,2} & \cdots & p_{0,n} \\
  p_{1,0} & p_{1,2} & \cdots & p_{1,n} \\
  \vdots & \ddots & \vdots & \vdots \\
  p_{m,0} & p_{m,0} & \cdots & p_{m,n}
\end{pmatrix}
\]
Convolution

- Operation between 2 matrices.
- In digital image processing:

\[
\begin{pmatrix}
k_{0,0} & \cdots & k_{s,0} \\
\vdots & \ddots & \vdots \\
k_{0,s} & \cdots & k_{s,s}
\end{pmatrix}
\ast
\begin{pmatrix}
p_{0,1} & p_{0,2} & \cdots & p_{0,n} \\
p_{1,0} & p_{1,2} & \cdots & p_{1,n} \\
\vdots & \vdots & \ddots & \vdots \\
p_{m,0} & p_{m,0} & \cdots & p_{m,n}
\end{pmatrix}
\]

**Kernel**

{odd}\times{odd}

**Image**

(picture, any dim)
Convolution

• New element is a combination of kernel elements and the current neighbourhood.

\[
(K \ast P)(x, y) = \sum_{\Delta x=r}^{\Delta x=-r} \sum_{\Delta y=r}^{\Delta y=-r} K(x-\Delta x, y-\Delta y) \cdot P(x+\Delta x, y+\Delta y)
\]

\[
K \ast P = Q
\]

\[
K \in \text{Mat}(s \times s)
\]

\[
P \in \text{Mat}(m \times n) \quad r = \left\lfloor \frac{s}{2} \right\rfloor \quad \text{(radius)}
\]

\[
Q \in \text{Mat}((m+r) \times (n+r)) \approx \text{Mat}(m \times n)
\]
Convolution

- New element is a combination of kernel elements and the current neighbourhood.

\[
(K \ast P)(x, y) = \sum_{\Delta x = -r}^{\Delta x = r} \sum_{\Delta y = -r}^{\Delta y = r} K(x - \Delta x, y - \Delta y) \cdot P(x + \Delta x, y + \Delta y)
\]

- For symmetric kernels:

\[
(K \ast P)(x, y) = \sum_{\Delta x = -r}^{\Delta x = r} \sum_{\Delta y = -r}^{\Delta y = r} K(x + \Delta x, y + \Delta y) \cdot P(x + \Delta x, y + \Delta y)
\]
Convolution

\[ (K \ast P)(x, y) = \sum_{\Delta x = -r}^{\Delta x = r} \sum_{\Delta y = -r}^{\Delta y = r} K(x + \Delta x, y + \Delta y) \cdot P(x + \Delta x, y + \Delta y) \]

\[ r = \left\lfloor \frac{3}{2} \right\rfloor = 1 \] (radius)
Convolution

\[(K \ast P)(x, y) = \sum_{\Delta x = -r}^{\Delta x = r} \sum_{\Delta y = -r}^{\Delta y = r} K(x + \Delta x, y + \Delta y) \cdot P(x + \Delta x, y + \Delta y)\]

Symmetric kernel

\[
\begin{array}{ccc}
  a & b & a \\
  b & c & b \\
  a & b & a \\
\end{array}
\]

\[
\begin{array}{cccc}
  & & & \\
  & & & \\
  & & & \\
\end{array}
\]

\[
\begin{array}{cccc}
  & & & \\
  & & & \\
  & & & \\
\end{array}
\]

\[
\begin{array}{cccc}
  & & & \\
  & & & \\
  & & & \\
\end{array}
\]

\[
\begin{array}{cccc}
  & & & \\
  & & & \\
  & & & \\
\end{array}
\]

\[
\begin{array}{cccc}
  & & & \\
  & & & \\
  & & & \\
\end{array}
\]

\[
\begin{array}{cccc}
  & & & \\
  & & & \\
  & & & \\
\end{array}
\]

\[
\begin{array}{cccc}
  & & & \\
  & & & \\
  & & & \\
\end{array}
\]

\[
\begin{array}{cccc}
  & & & \\
  & & & \\
  & & & \\
\end{array}
\]

\[
\begin{array}{cccc}
  & & & \\
  & & & \\
  & & & \\
\end{array}
\]

\[
\begin{array}{cccc}
  & & & \\
  & & & \\
  & & & \\
\end{array}
\]

\[
\begin{array}{cccc}
  & & & \\
  & & & \\
  & & & \\
\end{array}
\]
Convolution

\[(K \ast P)(x, y) = \sum_{\Delta x = r, \Delta y = r} K(x + \Delta x, y + \Delta y) \cdot P(x + \Delta x, y + \Delta y)\]
Convolution

\[(K \ast P)(x, y) = \sum_{\Delta x = -r}^{\Delta x = r} \sum_{\Delta y = -r}^{\Delta y = r} K(x + \Delta x, y + \Delta y) \cdot P(x + \Delta x, y + \Delta y)\]
Convolution

\[(K \ast P)(x, y) = \sum_{\Delta x = -r}^{\Delta x = r} \sum_{\Delta y = -r}^{\Delta y = r} K(x + \Delta x, y + \Delta y) \cdot P(x + \Delta x, y + \Delta y)\]

Single output pixel at some \((x, y)\)
Convolution

\[(K \ast P)(x, y) = \sum_{\Delta x = -r}^{\Delta x = r} \sum_{\Delta y = -r}^{\Delta y = r} K(x + \Delta x, y + \Delta y) \cdot P(x + \Delta x, y + \Delta y)\]
Convolution

\[
(K \ast P)(x, y) = \sum_{\Delta x = -r}^{\Delta x = r} \sum_{\Delta y = -r}^{\Delta y = r} K(x + \Delta x, y + \Delta y) \cdot P(x + \Delta x, y + \Delta y)
\]
Convolution

\[(K * P)(x, y) = \sum_{\Delta x = r}^{\Delta x = -r} \sum_{\Delta y = r}^{\Delta y = -r} K(x + \Delta x, y + \Delta y) \cdot P(x + \Delta x, y + \Delta y)\]
Convolution

$$(K * P)(x, y) = \sum_{\Delta x = r}^{\Delta x = -r} \sum_{\Delta y = r}^{\Delta y = -r} K(x + \Delta x, y + \Delta y) \cdot P(x + \Delta x, y + \Delta y)$$
Convolution

\[(K \ast P)(x, y) = \sum_{\Delta x = r}^{\Delta x = -r} \sum_{\Delta y = r}^{\Delta y = -r} K(x + \Delta x, y + \Delta y) \cdot P(x + \Delta x, y + \Delta y)\]

\[\begin{array}{ccc}
a & b & a \\
b & c & b \\
a & b & a \\
\end{array}\]  \ast  \begin{array}{ccc}
\text{red box} & \text{green box} & \text{red box} \\
\text{orange box} & \text{blue box} & \text{orange box} \\
\text{red box} & \text{green box} & \text{red box} \\
\end{array} = \begin{array}{ccc}
\text{red box} & \text{green box} & \text{red box} \\
\text{orange box} & \text{blue box} & \text{orange box} \\
\text{red box} & \text{green box} & \text{red box} \\
\end{array}

\[= a \quad + b \quad + a \quad + \ldots\]
Convolution

\[
(K * P)(x, y) = \sum_{\Delta x = -r}^{\Delta x = r} \sum_{\Delta y = -r}^{\Delta y = r} K(x + \Delta x, y + \Delta y) \cdot P(x + \Delta x, y + \Delta y)
\]

\[
\begin{bmatrix}
  a & b & a \\
  b & c & b \\
  a & b & a \\
\end{bmatrix} \ast
\begin{bmatrix}
  \ast & & \ast \\
  & \ast & \\
  \ast & & \ast \\
\end{bmatrix} = \begin{bmatrix}
  a & +b & +a & +b \\
  +b & +c & +b \\
  +a & +b & +a \\
\end{bmatrix} = \begin{bmatrix}
  \ast \\
  \ast \\
  \ast \\
\end{bmatrix}
\]
Convolution

$$(K\ast P)(x, y) = \sum_{\Delta x = r}^{\Delta x = -r} \sum_{\Delta y = r}^{\Delta y = -r} K(x + \Delta x, y + \Delta y) \cdot P(x + \Delta x, y + \Delta y)$$
Convolution

\[(K \ast P)(x, y) = \sum_{\Delta x = -r}^{\Delta x = r} \sum_{\Delta y = -r}^{\Delta y = r} K(x + \Delta x, y + \Delta y) \cdot P(x + \Delta x, y + \Delta y)\]
Convolution

\[(K \ast P)(x, y) = \sum_{\Delta x = r}^{\Delta x = -r} \sum_{\Delta y = r}^{\Delta y = -r} K(x + \Delta x, y + \Delta y) \cdot P(x + \Delta x, y + \Delta y)\]
Convolution

\[(K \ast P)(x, y) = \sum_{\Delta x = -r}^{\Delta x = r} \sum_{\Delta y = -r}^{\Delta y = r} K(x + \Delta x, y + \Delta y) \cdot P(x + \Delta x, y + \Delta y)\]
Convolution

\[(K \ast P)(x, y) = \sum_{\Delta x = -r}^{\Delta x = r} \sum_{\Delta y = -r}^{\Delta y = r} K(x + \Delta x, y + \Delta y) \cdot P(x + \Delta x, y + \Delta y)\]
Convolution

\[(K \ast P)(x, y) = \sum_{\Delta x = r} \sum_{\Delta y = r} K(x + \Delta x, y + \Delta y) \cdot P(x + \Delta x, y + \Delta y)\]
Convolution

\[(K \ast P)(x, y) = \sum_{\Delta x = -r}^{\Delta x = r} \sum_{\Delta y = -r}^{\Delta y = r} K(x + \Delta x, y + \Delta y) \cdot P(x + \Delta x, y + \Delta y)\]

- All output elements can be calculated like that.
Convolution

- When the neighbourhood is outside the input:
When the neighbourhood is outside the input:

The values could be repeated edge values

The values could be repeated edge values
Convolution

- When the neighbourhood is outside the input:

- The values could be zero values (eg with alpha)
Convolution

- We could also calculate some outside values:
Convolution

- Naive implementation complexity:

\[
(K \ast P) = O(s^2 \cdot m \cdot n) = O(n^4)
\]
Convolution

- Naive implementation complexity:

\[
(K \ast P) = O(s^2 \cdot m \cdot n) = O(n^4)
\]

Kernel size

\[K \ast P = Q\]

\[K \in \text{Mat}(s \times s)\]

\[P \in \text{Mat}(m \times n)\]
Convolution

- Naive implementation complexity:

\[ (K \ast P) = O(s^2 \cdot m \cdot n) = O(n^4) \]

Image size

\[ K \ast P = Q \]

\[ K \in \text{Mat}(s \times s) \]

\[ P \in \text{Mat}(m \times n) \]
Convolontion

- Naive implementation complexity:

\[(K \ast P) = O(s^2 \cdot m \cdot n) = O(n^4)\]

\[K \ast P = Q\]

\[K \in \text{Mat}(s \times s)\]

\[P \in \text{Mat}(m \times n)\]
Convolution

- Naive implementation complexity:

\[
(K \ast P) = O(s^2 \cdot m \cdot n) = O(n^4)
\]

- If our fragment shader runs parallel on every output pixel, we are left with:

\[
(K \ast P) = O(s^2) \quad \text{(for a single pixel)}
\]
Convolution

- Naive implementation complexity:

\[(K \ast P) = O(s^2 \cdot m \cdot n) = O(n^4)\]

- If our fragment shader runs parallel on every output pixel, we are left with:

\[(K \ast P) = O(s^2)\] (for a single pixel)

- That is still not very good if our kernel is larger.
Box Blur

• Finds the average value.

\[ \frac{1}{9} \]

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Kernel

Output

Also called the median filter.
Box Blur

- Finds the average value.

\[
\begin{array}{ccc}
0.1 & 0.1 & 0.1 \\
0.1 & 0.1 & 0.1 \\
0.1 & 0.1 & 0.1 \\
\end{array}
\]

\[
\frac{1}{9}
\]

\[
\begin{array}{cccccc}
0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\
0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\
0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\
\end{array}
\]

\[
\frac{1}{25}
\]
Gaussian Blur

- Kernel is a sampling of a 2D Gaussian function

\[
g(x, y) = \frac{1}{2\pi \sigma^2} e^{-\frac{x^2 + y^2}{2\sigma^2}}
\]

\( \sigma \) – standard deviation
Kernels normalized to have 1 in the middle

What is the best combination and why?
Gaussian Blur

dim=72

std=1  std=3  std=6  std=9  std=12

std=12, dim=28  std=12, dim=22  std=12, dim=18  std=12, dim=12

Aliasing
Separable Kernels

- If the columns of $K$ are linearly dependent*, then:

$$K = v_0 \cdot v_1$$

* Matrix $K$ is of rank 1 – the vector space spanned by the columns is 1D. All the columns are linear multiples of one column.
Separable Kernels

- If the columns of $K$ are linearly dependent, then:

$$K = v_0 \cdot v_1$$

$$\begin{bmatrix}
\begin{array}{ccc}
1/9 & 1/9 & 1/9 \\
1/9 & 1/9 & 1/9 \\
1/9 & 1/9 & 1/9 \\
\end{array}
\end{bmatrix} \quad \begin{bmatrix}
1/9 & 1/9 & 1/9 \\
1/9 & 1/9 & 1/9 \\
1/9 & 1/9 & 1/9 \\
\end{bmatrix} = \begin{bmatrix}
1/3 \\
1/3 \\
1/3 \\
\end{bmatrix} \cdot \begin{bmatrix}
1/3 & 1/3 & 1/3 \\
\end{bmatrix}$$
Separable Kernels

- If the columns of $K$ are linearly dependent, then:

$$K = v_0 \cdot v_1$$

$$\begin{bmatrix}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1
\end{bmatrix} = \begin{bmatrix} 1 \\
2 \\
1
\end{bmatrix} \cdot \begin{bmatrix}
-1 & 0 & 1
\end{bmatrix}$$
Separable Kernels

- If the columns of $K$ are linearly dependent, then:

$$K = v_0 \cdot v_1$$

$$\begin{align*}
\text{Mat}(s \times s) & \quad \text{Mat}(s \times 1) & \quad \text{Mat}(1 \times s) \\
1 & 2 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1
\end{align*}$$

\[
\begin{array}{c}
1 \\
2 \\
1
\end{array} \quad \cdot \quad 
\begin{array}{c}
1 & 2 & 1
\end{array}
\]
Separable Kernels

\[
\begin{array}{|c|c|c|}
\hline
\text{a} & \text{d} & \text{ad} \\
\hline
\text{b} & \text{e} \\
\hline
\text{c} & \text{f} \\
\hline
\end{array}
\]
Separable Kernels

\[
\begin{bmatrix}
a & b & c \\
b & & \\
c & & \\
\end{bmatrix} \cdot \begin{bmatrix}
d & e & f \\
e & & \\
f & & \\
\end{bmatrix} = \begin{bmatrix}
ad & ae \\
b & & \\
c & & \\
\end{bmatrix}
\]
Separable Kernels

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\text{d} & \text{e} & \text{f} \\
\end{array}
\cdot
\begin{array}{ccc}
\text{ad} & \text{ae} & \text{af} \\
\end{array}
= \begin{array}{ccc}
\end{array}
\]
Separable Kernels

\[
\begin{array}{ccc}
  a & b & c \\
  d & e & f \\
  ad & ae & af \\
  bd & be
\end{array}
\]
Separable Kernels

\[
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
 a \\
b \\
c
\end{array}
\end{array}
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
 d \\
e \\
f
\end{array}
\end{array}
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
 ad \\
 ae \\
 af \\
 bd \\
 be \\
 bf
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
 a \\
b \\
c
\end{array}
\end{array}
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
 d \\
e \\
f
\end{array}
\end{array}
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
 ad \\
 ae \\
 af \\
 bd \\
 be \\
 bf
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\]
\end{array}
\]
Separable Kernels

\[
\begin{array}{ccc}
a & b & c \\
\end{array}
\begin{array}{ccc}
d & e & f \\
\end{array}
\begin{array}{ccc}
ad & ae & af \\
bd & be & bf \\
bd & \\
\end{array}
\]
Separable Kernels

\[
\begin{array}{ccc}
 a & b & c \\
 \cdot & d & e & f \\
\end{array}
\quad =
\begin{array}{ccc}
 ad & ae & af \\
 bd & be & bf \\
 cd & ce \\
\end{array}
\]
Separable Kernels

\[
\begin{array}{ccc}
  a & d & f \\
  b & e & f \\
  c & & \\
\end{array}
\quad \cdot \quad
\begin{array}{ccc}
  ad & ae & af \\
  bd & be & bf \\
  cd & ce & cf \\
\end{array}
= 
\begin{array}{ccc}
  & & \\
  & & \\
  & & \\
\end{array}
\]
Separable Kernels

\[
\begin{array}{ccc}
  a & & \\
  b & & \\
  c & & \\
\end{array}
\begin{array}{ccc}
  \cdot & & \\
  d & e & f \\
\end{array}
= 
\begin{array}{ccc}
  \cdot & & \\
  a & ae & af \\
  bd & be & bf \\
  cd & ce & cf \\
\end{array}
\]
Separable Kernels

\[
\begin{array}{c}
a \\
b \\
c
\end{array}
\ast
\begin{array}{ccc}
d & e & f
\end{array}
=
\begin{array}{cccc}
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
\end{array}
\]
Separable Kernels

\[
\begin{array}{ccc}
  a & b & c \\
  \times & \begin{array}{ccc}
      0 & 0 & 0 \\
      d & e & f \\
      0 & 0 & 0 \\
    \end{array} & = \\
  & \begin{array}{ccc}
    & & \\
    & & \\
    & & \\
  \end{array}
\end{array}
\]
Separable Kernels

Kernel is not symmetric!
So we must use values opposite of the current pixel.
Separable Kernels

Kernel is not symmetric!
So we must use values opposite of the current pixel.
Separable Kernels

\[ ae + 0 b + 0 c = ae \]
Separable Kernels

\[ af + 0 \cdot b + 0 \cdot c = af \]
Separable Kernels

\[ 0a + bd + 0c = bd \]
Separable Kernels

\[ 0a + be + 0c = be \]
Separable Kernels

\[
\begin{array}{c}
\begin{array}{c}
a \\
b \\
c
\end{array}
\end{array} \ast
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & 0 \\
d & e & f \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{array}
= \\
\begin{array}{ccc}
ad & ae & af \\
bd & be & bf \\
\end{array}
\]

\[0a + bf + 0c = bf\]
Separable Kernels

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & 0 \\
d & e & f \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{ccc}
ad & ae & af \\
bd & be & bf \\
\cd \\
\end{array}
\]

\[0 \cdot a + 0 \cdot b + 0 \cdot \cd = \cd\]
Separable Kernels

\[
\begin{array}{c}
a \\
b \\
c
\end{array} \ast 
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & 0 \\
d & e & f \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{array} = 
\begin{array}{ccc}
ad & ae & af \\
bd & be & bf \\
0 & 0 & 0 \\
0 & 0 & 0 \\
cd & ce
\end{array}
\]

\[0 \ a + 0 \ b + 0 \ ce = ce\]
Separable Kernels

\[ 0a + 0b + 0 cf = cf \]
Separable Kernels

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & 0 \\
d & e & f \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{ccc}
ad & ae & af \\
bd & be & bf \\
cd & ce & cf \\
\end{array}
\]

Convolution

Multiplication

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & 0 \\
d & e & f \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{ccc}
ad & ae & af \\
bd & be & bf \\
cd & ce & cf \\
\end{array}
\]

Convolution

Multiplication
Separable Kernels

- Turns out that in our current case:

\[ K = v_0 \cdot v_1 = v_0 \ast v_1 \]

- Matrix multiplication
- Convolution
Separable Kernels

- Turns out that in our current case:

\[ K = v_0 \cdot v_1 = v_0 \ast v_1 \]

\[ K \ast P = (v_0 \cdot v_1) \ast P = (v_0 \ast v_1) \ast P \]
Separable Kernels

• Turns out that in our current case:

\[ K = v_0 \cdot v_1 = v_0 \ast v_1 \]

\[ K \ast P = (v_0 \cdot v_1) \ast P = (v_0 \ast v_1) \ast P \]

• Convolution is commutative*:

\[ K \ast P = (v_0 \cdot v_1) \ast P = v_0 \ast (v_1 \ast P) \]

*Proof is out of the scope here.
Separable Kernels

• If our kernel is separable:

\[ K = v_0 \cdot v_1 \]
Separable Kernels

• If our kernel is **separable**:

\[ K = v_0 \cdot v_1 \]

• Then a 1×2D convolution is the same as 2×1D:

\[ K * P = v_0 * (v_1 * P) \]
Separable Kernels

- If our kernel is **separable**:

\[ K = v_0 \cdot v_1 \]

- Then a 1×2D convolution is the same as 2×1D:

\[
K \ast P = v_0 \ast (v_1 \ast P) \\
= O(s^2) + O(s) + O(s) = O(s)
\]

Ignoring the complexity of parsing all the image pixels.
Separable Kernels

- The Gaussian blur kernel is **separable**:

\[ g(x, y) = \frac{1}{2\pi \sigma^2} \cdot e^{-\frac{x^2 + y^2}{2\sigma^2}} = \]

\[ = \frac{1}{\sqrt{2\pi \sigma^2}} \cdot e^{-\frac{x^2}{2\sigma^2}} \cdot \frac{1}{\sqrt{2\pi \sigma^2}} \cdot e^{-\frac{y^2}{2\sigma^2}} \]

1D Gaussian functions
Singular Value Decomposition

- Decomposes matrix $K$ into 3 other matrices.

\[ K = U \Sigma V \]
Singular Value Decomposition

- Decomposes matrix $K$ into 3 other matrices.
- Singular values are ordered.
Singular Value Decomposition

- The number of singular values = rank of $K$.
- If $K$ is separable, rank of $K = 1$. 

\[
\begin{array}{c}
\begin{array}{ccc}
& & \\
& & \\
& & \\
& & \\
& & \\
& & \\
& & \\
& & \\
\end{array} \\
= \\
\begin{array}{ccc}
& & \\
& & \\
& & \\
& & \\
& & \\
& & \\
& & \\
& & \\
\end{array} \\
\cdot \\
\begin{array}{ccc}
0 & 0 & \\
0 & 0 & \\
0 & 0 & \\
0 & 0 & \\
0 & 0 & \\
0 & 0 & \\
0 & 0 & \\
0 & 0 & \\
\end{array} \\
\cdot \\
\begin{array}{ccc}
& & \\
& & \\
& & \\
& & \\
& & \\
& & \\
& & \\
& & \\
\end{array}
\end{array}
\]
Singular Value Decomposition

\[
\begin{bmatrix}
    a \\
    b \\
    \vdots \\
    d
\end{bmatrix} \approx
\begin{bmatrix}
    s & 0 & 0 \\
    0 & 0 & 0 \\
    0 & 0 & 0
\end{bmatrix} \cdot
\begin{bmatrix}
    \text{as} & 0 & 0 \\
    \text{bs} & 0 & 0 \\
    \ldots & \ldots & \ldots
\end{bmatrix} \cdot
\begin{bmatrix}
    a \\
    b \\
    \ldots \\
    d
\end{bmatrix}
\]
Singular Value Decomposition

\[ \begin{pmatrix} s & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} a & b & \ldots & d \\ \end{pmatrix} = \begin{pmatrix} as & bs & \ldots & ds \\ bs & 0 & 0 \\ cs & 0 & 0 \end{pmatrix} = \begin{pmatrix} s \\ \end{pmatrix} \cdot \begin{pmatrix} a & b & \ldots & d \end{pmatrix} \]
Singular Value Decomposition

\[ \begin{pmatrix} a & b & \ldots & d \\ \end{pmatrix} = \begin{pmatrix} s & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \end{pmatrix} \begin{pmatrix} a & b & \ldots & d \\ \end{pmatrix} \]
Singular Value Decomposition

\[ \begin{bmatrix} a \\ b \\ \vdots \\ d \end{bmatrix} = \sqrt{s} \cdot \begin{bmatrix} a & b & \ldots & d \end{bmatrix} = v_0 \cdot v_1 \]

where

\[ K = v_0 \cdot v_1 \]
### Singular Value Decomposition

When the kernel is circularly symmetric:

\[
K = v_0 \cdot v_1 = v_0 \cdot v_0^T
\]
Singular Value Decomposition

\[ a \quad b \quad \ldots \quad d = \sqrt{s} \cdot a \quad b \quad \ldots \quad d \cdot 0 \quad 0 \quad 0. \]

When the kernel is circularly symmetric:

\[ v_0^T (c + \Delta x) = v_0^T (c - \Delta x) \]

\[ K = v_0 \cdot v_0^T \]
Singular Value Decomposition

\[ a \begin{bmatrix} b \\ \vdots \\ d \end{bmatrix} = \begin{bmatrix} \sqrt{s} \\ \sqrt{s} \end{bmatrix} \begin{bmatrix} a & b & \ldots & a \end{bmatrix} \begin{bmatrix} s & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a & b & \ldots & d \end{bmatrix} \]

When the kernel is circularly symmetric:

\[ v_0(c + \Delta x) = v_0(c - \Delta x) \]

\[ K = K^T \]

\[ K = v_0 \cdot v_0^T \]
Depth of Field
Depth of Field

- Standard rendering models a *pinhole camera*.

- Every surface point has 1 light ray in the projection.
Depth of Field

- Problems with a pinhole camera:
  - Smaller hole means less light passes through
  - Less light means a darker image
Depth of Field

• Problems with a pinhole camera:
  • Smaller hole means less light passes through
  • Less light means a darker image
  • Longer exposure could compensate
  • Longer exposure creates motion blur
Depth of Field

- Problems with a pinhole camera:
  - Smaller hole means less light passes through
  - Less light means a darker image
  - Longer exposure could compensate
  - Longer exposure creates motion blur
  - More sensitive sensor could compensate
  - More sensitive sensor means more noise
Depth of Field

- Cameras with lenses*

* There are actually differently shaped lenses that each refract light differently for specific goals.
Depth of Field

- Cameras with lenses

- Close objects are *out of focus* on the sensor
Depth of Field

• Cameras with lenses

• Close objects are out of focus on the sensor

• Far away objects are out of focus on the sensor

Depth of Field

Acceptably sharp
Depth of Field

- Area on the sensor where the out of focus surface point is spread out in.
Depth of Field

Implementation

Object Space

Image Space
Depth of Field

Implementation

Object Space

Image Space

eg
Depth of Field

Implementation

Object Space

Image Space

eg

Scatter

Gather
Depth of Field

Implementation

Object Space

Image Space

eg

Scatter

Gather

(convolution)
Depth of Field (1)

- When rendering:
  - Separate pixels in the fore- or background
Depth of Field (1)

• When rendering:
  • Separate pixels in the fore- or background
  • Blur those pixels
Depth of Field (1)

- When rendering:
  - Separate pixels in the fore- or background
  - Blur those pixels
  - Render the midground
Depth of Field (1)

- When rendering:
  - Separate pixels in the fore- or background
  - Blur those pixels
  - Render the midground and blend the results
Depth of Field (1)

Transparent pixels on the screen
Depth of Field (2)

Extend the midground to cover them.
Depth of Field (2)

- Current layers:
Background blur should not cover objects in the front.
Depth of Field (3)

Render background first, then blend midground, then foreground.
Depth of Field (3)

Render background first, then blend midground, then foreground.
Depth of Field (3)

- Current layers:
Depth of Field (3)

Too sharp border between the midground and background
Depth of Field (4)

Extend the background instead of the midground.
Create a linear gradient between the midground and background
Depth of Field (4)

- Current layers:
Depth of Field (4)

Sharp border between the foreground and midground.
Depth of Field (5)

Use a varying size blur.
Depth of Field (5)

- Current layers:
When CoC is read only from the current pixel, there will be no outside blur (halo).
When CoC is read only from the current pixel, there will be no outside blur (halo).
Depth of Field (5)

CoC = 1
Full blur applied

CoC = 0
No blur applied

Transparent pixels in front of midground.

Difference between scatter and gather techniques.
Depth of Field (6)

Take CoC to be MAX CoC from the neighbourhood.
Depth of Field (6)

Sharp large edge around objects with different CoC values.
Depth of Field (7)

Take CoC to be a mix between the MAX CoC and AVG CoC from the neighbourhood. Mix by the percentage of 0 transparency pixels.
No visible inside blur on the foreground. There is no data to blend inside pixels with.
Depth of Field (8)

Render the midground with a clipping plane at the midground-foreground border.

Now the blur is dependent also on the scene's complexity a bit.
Depth of Field (8)

Varyingly blurred background

Clipped midground

Varyingly blurred foreground
Circular Depth of Field

Instead of modelling the circle of confusion with this Gaussian blur kernel ...
Circular Depth of Field

Instead of modelling the circle of confusion with this Gaussian blur kernel ...

... it should rather be like this.
Circular Depth of Field

This is not of rank 1.
Thus not separable.

When the cross-section increases, there are new affected dimensions.
Circular Depth of Field

- Different shapes can be approximated via a combination of sinusoids (sines and cosines).

http://www.falstad.com/fourier/
Circular Depth of Field

- Different shapes can be approximated via a combination of sinusoids (sines and cosines).
- More waves (components) means a better approximation.
Circular Depth of Field

- Complex numbers

\[ z = x_z + iy_z \]

\[ z = a \cdot (\cos(b) + i \sin(b)) \]

\[ z = a \cdot e^{ib} \quad \text{(via Euler's formula)} \]
Circular Depth of Field

• Complex numbers

$$z = a \cdot (\cos(b) + i \sin(b))$$
Circular Depth of Field

- Complex numbers

\[ z = a \cdot (\cos(b) + i \sin(b)) \]

\[ z(x) = a \cdot (\cos(bx) + i \sin(bx)) \]

This function outputs complex numbers.

- Amplitude
- Frequency
Circular Depth of Field

• We are looking for conditions for a circularly symmetric separable kernel...

... to see if we could create one out of complex numbers (because they include sinusoids)
Circular Depth of Field

- We are looking for conditions for a circularly symmetric separable kernel.

\[
K = v_0 \cdot v_1 = v_0 \cdot v_0^T
\]

When the kernel is circularly symmetric:

\[
v_0(c + \Delta x) = v_0(c - \Delta x)
\]

\[
K = K^T
\]
Circular Depth of Field

• We are looking for conditions for a circularly symmetric separable kernel.

\[ a \cdot b \cdot \ldots \cdot a = \begin{array}{ccc}
  a & b & \ldots \\
  a & a & \ldots \\
  \ldots & \ldots & \ldots \\
\end{array} \cdot \begin{array}{c}
  a \\
  b \\
  \ldots \\
  a \\
\end{array} \]
Circular Depth of Field

- We are looking for conditions for a \textit{circularly symmetric separable kernel}.
Circular Depth of Field

• We are looking for conditions for a **circularly symmetric separable kernel**.

\[ v(x) \cdot v(y) \cdot v(0) = v(x) \cdot v(y) \]

All values at the same radius need to be the same.
Circular Depth of Field

- We are looking for conditions for a circularly symmetric separable kernel.

\[ v(x) = v(y) = \cdots \]
Circular Depth of Field

- We are looking for conditions for a circularly symmetric separable kernel.

$$v(x) = v(y) \cdot v(0) \cdot v(\sqrt{x^2 + y^2})$$
Circular Depth of Field

- We are looking for conditions for a circularly symmetric separable kernel.

\[ v(x) = v(y) \cdot v(0) \cdot v(\sqrt{x^2 + y^2}) \]

If we have enough precision, this value is somewhere on the 1D kernel.
Circular Depth of Field

- Condition for a circularly symmetric separable kernel:

\[ v(x) \cdot v(y) = v(0) \cdot v(\sqrt{x^2 + y^2}) \]

(constant)  (value at the radius)
Circular Depth of Field

- Condition for a **circularly symmetric separable kernel**:

\[ v(x) \cdot v(y) = v(0) \cdot v(\sqrt{x^2 + y^2}) \]

\[ g(x) = \frac{1}{\sqrt{2\pi \sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \]

Checking the Gaussian blur kernel.
Circular Depth of Field

- Condition for a circularly symmetric separable kernel:

\[
\mathbf{v}(x) \cdot \mathbf{v}(y) = \mathbf{v}(0) \cdot \mathbf{v} \left( \sqrt{x^2 + y^2} \right)
\]

\[
g(x) = \alpha \cdot e^{-a x^2}
\]

(simplification)
Circular Depth of Field

- Condition for a \textit{circularly symmetric separable kernel}:

\[
v(x) \cdot v(y) = v(0) \cdot v(\sqrt{x^2 + y^2})
\]

\[
\alpha \cdot e^{-ax^2} \cdot \alpha \cdot e^{-ay^2} = \alpha \cdot e^0 \cdot \alpha \cdot e^{-a(\sqrt{x^2 + y^2})^2}
\]
Circular Depth of Field

- Condition for a circularly symmetric separable kernel:

\[ v(x) \cdot v(y) = v(0) \cdot v(\sqrt{x^2 + y^2}) \]

\[ \alpha \cdot e^{-ax^2} \cdot \alpha \cdot e^{-ay^2} = \alpha \cdot e^0 \cdot \alpha \cdot e^{-a(\sqrt{x^2 + y^2})^2} \]

\[ \alpha \cdot e^{-ax^2} \cdot \alpha \cdot e^{-ay^2} = \alpha \cdot \alpha \cdot e^{-a(x^2 + y^2)} \]
Circular Depth of Field

• Condition for a **circularly symmetric separable kernel**:

\[
\mathbf{v}(x) \cdot \mathbf{v}(y) = \mathbf{v}(0) \cdot \mathbf{v}(\sqrt{x^2 + y^2})
\]

\[
\alpha \cdot e^{-ax^2} \cdot \alpha \cdot e^{-ay^2} = \alpha \cdot e^0 \cdot \alpha \cdot e^{-a(\sqrt{x^2 + y^2})^2}
\]

\[
\alpha \cdot e^{-ax^2} \cdot \alpha \cdot e^{-ay^2} = \alpha \cdot \alpha \cdot e^{-a(x^2 + y^2)}
\]

\[
\alpha \cdot e^{-ax^2} \cdot \alpha \cdot e^{-ay^2} = \alpha^2 \cdot e^{-a(x^2 + y^2)}
\]

Satisfies the condition
Circular Depth of Field

- What about our function with complex numbers?

\[ z(x) = a \cdot (\cos(bx) + i \sin(bx)) \]
Circular Depth of Field

- What about our function with complex numbers?

\[ z(x) = a \cdot (\cos(bx) + i \sin(bx)) \]

\[ z(x) = a e^{ibx} \]
Circular Depth of Field

• What about our function with complex numbers?

\[ z(x) = a \cdot (\cos(bx) + i \sin(bx)) \]
\[ z(x) = a e^{ibx} \]

\[ a e^{ibx} \cdot a e^{iby} = a e^0 \cdot a e^{ib(x+y)} \]

\[ v(x) \cdot v(y) = v(0) \cdot v(\sqrt{x^2 + y^2}) \]
Circular Depth of Field

• What about our function with complex numbers?

\[ z(x) = a \cdot (\cos(bx) + i \sin(bx)) \]
\[ z(x) = a e^{ibx} \]

\[ a e^{ibx} \cdot a e^{iby} = a e^0 \cdot a e^{ib(x+y)} \]
\[ \neq a e^0 \cdot a e^{ib(\sqrt{x^2+y^2})} \]

Does not satisfy the condition

\[ v(x) \cdot v(y) = v(0) \cdot v(\sqrt{x^2+y^2}) \]
Circular Depth of Field

• What about our function with complex numbers?

\[ z(x) = a \cdot (\cos(bx^2) + i \sin(bx^2)) \]

\[ z(x) = a e^{ibx^2} \]

\[ a e^{ibx^2} \cdot a e^{iby^2} = a e^0 \cdot a e^{ib(x^2+y^2)} \]

\[ = a e^0 \cdot a e^{ib(\sqrt{x^2+y^2})^2} \]

Satisfies the condition

\[ v(x) \cdot v(y) = v(0) \cdot v(\sqrt{x^2+y^2}) \]
Circular Depth of Field

\[ z(x) = a \cdot (\cos(bx^2) + i \sin(bx^2)) \]

1D

\[ z(x) = a \cdot e^{ibx^2} \]

2D

\[ K(x, y) = a^2 \cdot e^{ib(x^2 + y^2)} \]

Real part.

Values scaled. Blue is negative.
Circular Depth of Field

\[ z(x) = a \cdot (\cos(bx^2) + i \sin(bx^2)) \]

\[ K(x, y) = a^2 \cdot e^{ib(x^2 + y^2)} \]

\[ B = K \ast P \]

Blurred picture (real)

Kernel (cmplx)

Input picture (real)

Real part.

When we do the 2D convolution, only the real part matters.
Circular Depth of Field

\[ z(x) = a \cdot (\cos(bx^2) + i \sin(bx^2)) \]

We are currently only using the cosine wave (real part).

\[ K(x, y) = a^2 \cdot e^{ib(x^2 + y^2)} \]
Circular Depth of Field

\[ z(x) = a \cdot (\cos(bx^2) + i \sin(bx^2)) \]

\[ a = (c - id) \]

\[ K(x, y) = a^2 \cdot e^{ib(x^2 + y^2)} \]

Multiplication of two complex numbers is still a complex number.
Circular Depth of Field

\[ z(x) = (c-id) \cdot (\cos(bx^2) + i \sin(bx^2)) = \]

\[ = c \cos(bx^2) + d \sin(bx^2) + i(c \sin(bx^2) - d \cos(bx^2)) \]

Now we have both a cosine and a sine in the real part.

This is also important for a separated kernel.

\[ K(x, y) = (c-id)^2 \cdot e^{ib(x^2+y^2)} \]
Circular Depth of Field

\[ z(x) = (c - id) \cdot (\cos(bx^2) + i \sin(bx^2)) \]

\[ K(x, y) = (c - id)^2 \cdot e^{ib(x^2 + y^2)} \]

We want a bound circle instead of a wavy pattern.
Circular Depth of Field

\[ z(x) = (c - id) \cdot e^{ax^2} \cdot (\cos(bx^2) + i\sin(bx^2)) \]

Multiply the kernel with a Gaussian envelope.

Element-wise multiplication of two circularly symmetric separable kernels is still a circularly symmetric separable kernel.

\[ K(x, y) = (c - id)^2 \cdot e^{a(x^2 + y^2)} \cdot e^{ib(x^2 + y^2)} \]
Circular Depth of Field

\[ z(x) = (c - id) \cdot e^{ax^2} \cdot (\cos(bx^2) + i \sin(bx^2)) \]

\[ K(x, y) = (c - id)^2 \cdot e^{a(x^2 + y^2)} \cdot e^{ib(x^2 + y^2)} \]
Circular Depth of Field

\[ z(x) = (c - id) \cdot e^{ax^2} \cdot (\cos(bx^2) + i \sin(bx^2)) \]

- Parameters:
  - \( a \sim \text{std of the Gaussian} \)

\[ K(x, y) = (c - id)^2 \cdot e^{-a(x^2 + y^2)} \cdot e^{ib(x^2 + y^2)} \]
Circular Depth of Field

\[ z(x) = (c - id) \cdot e^{ax^2} \cdot (\cos(bx^2) + i \sin(bx^2)) \]

- Parameters:
  - \( a \sim \text{std of the Gaussian} \)
  - \( b = \text{frequency of the waves} \)

\[ K(x, y) = (c - id)^2 \cdot e^{a(x^2 + y^2)} \cdot e^{-ib(x^2 + y^2)} \]
Circular Depth of Field

\[ z(x) = (c - id) \cdot e^{ax^2} \cdot (\cos(bx^2) + i \sin(bx^2)) \]

- Parameters:
  - \( a \) \sim \text{std of the Gaussian}
  - \( b \) = frequency of the waves
  - \( c \) = contribution of \( \cos() \) in 1D

\[ K(x, y) = (c - id)^2 \cdot e^{a(x^2 + y^2)} \cdot e^{ib(x^2 + y^2)} \]
Circular Depth of Field

\[ z(x) = (c - id) \cdot e^{ax^2} \cdot (\cos(bx^2) + i \sin(bx^2)) \]

- Parameters:
  - \( a \) \sim \text{std of the Gaussian}
  - \( b \) = frequency of the waves
  - \( c \) = contribution of \( \cos() \) in 1D
  - \( d \) = contribution of \( \sin() \) in 1D

\[ K(x, y) = (c - id)^2 \cdot e^{a(x^2 + y^2)} \cdot e^{ib(x^2 + y^2)} \]
Circular Depth of Field

- Too broad falloff, almost like Gaussian blur.

\[
\begin{align*}
    a &= -1 \\
    b &= 0.9 \\
    c &= 0.4 \\
    d &= 0.2
\end{align*}
\]
Circular Depth of Field

- Sharper falloff, but negative values.
- Negative values create a sharpening effect.

\[ a = -0.86 \]
\[ b = 1.62 \]
\[ c = 1.18 \]
\[ d = 0.79 \]
Circular Depth of Field

- No negative values, but broader falloff again
- Creates a more doughnut shaped bokeh.

\[a = -2.01\]
\[b = 1.05\]
\[c = 1.78\]
\[d = 1.57\]
Circular Depth of Field

- Normalization:
  - Differently from the Gaussian kernel, this kernel does not have its values sum up to 1.

The sum here is 214
Circular Depth of Field

• Normalization:
  • Differently from the Gaussian kernel, this kernel does not have its values sum up to 1.
  • That will cause the image to become too bright.

The sum here is 214
Circular Depth of Field

- Normalization:
  - Differently from the Gaussian kernel, this kernel does not have its values sum up to 1.
  - That will cause the image to become too bright.
  - Values must be normalized.

\[
\bar{K}(x, y) = \frac{K(x, y)}{\sum_{x', y'} K(x', y')}
\]

The sum here is 214
Circular Depth of Field
Circular Depth of Field
Circular Depth of Field
Circular Depth of Field

• Components:
  • We can create a better circle if we use multiple complex-numbered kernels (components).
Circular Depth of Field

- We can create a better circle if we use multiple complex-numbered kernels (components).

\[ K_0(x, y) = (c_0 - id_0)^2 \cdot e^{a_0(x^2 + y^2)} \cdot e^{ib_0(x^2 + y^2)} \]

\[ K_1(x, y) = (c_1 - id_1)^2 \cdot e^{a_1(x^2 + y^2)} \cdot e^{ib_1(x^2 + y^2)} \]
Circular Depth of Field

- We can create a better circle if we use multiple complex-numbered kernels (components).
- Sum of separable kernels is not separable.

\[
K_0(x, y) = \begin{array}{c}
\text{Separable} \\
K_1(x, y) = \end{array}
\]

\[
K_0(x, y) + K_1(x, y) = \text{Not separable}
\]
Circular Depth of Field

• We can create a better circle if we use multiple complex-numbered kernels (components).
• Sum of separable kernels is not separable.
• Convolution is distributive:

\[ K_0 * P + K_1 * P = (K_0 + K_1) * P \]
Circular Depth of Field

- We can create a better circle if we use multiple complex-numbered kernels (components).
- Sum of separable kernels is not separable.
- Convolution is distributive:

\[ K_0 \ast P + K_1 \ast P = (K_0 + K_1) \ast P \]

- But if we separate the kernels:

\[ (K_0 + K_1) \ast P = (v_0 \ast v_0^T + v_1 \ast v_1^T) \ast P \]
Circular Depth of Field

- We can create a better circle if we use multiple complex-numbered kernels (components).
- Sum of separable kernels is not separable.
- Convolution is distributive.
- But if we separate the kernels:

\[
(K_0 + K_1) \ast P = (v_0 \ast v_0^T + v_1 \ast v_1^T) \ast P
\]

- We see that we need to perform 2 \times 1D chains of (2×) convolution and add the results:

\[
(v_0 \ast v_0^T + v_1 \ast v_1^T) \ast P = v_0 \ast v_0^T \ast P + v_1 \ast v_1^T \ast P
\]

\[\text{2×1D conv} \quad \text{2×1D conv}\]
Circular Depth of Field

• Scaling
  • Instead of generating multiple kernels, mixing between them and normalizing in the shader...
Circular Depth of Field

• Scaling
  • Instead of generating multiple kernels, mixing between them and normalizing in the shader...
  • ... we can change the sampling offset scale instead.

\[
(K \ast P)(x, y) = \sum_{\Delta x = -r}^{\Delta x = r} \sum_{\Delta y = -r}^{\Delta y = r} K(x + s\Delta x, y + s\Delta y) \cdot P(x + \Delta x, y + \Delta y)
\]
Circular Depth of Field

- Scaling
  - Instead of generating multiple kernels, mixing between them and normalizing in the shader...
  - ... we can change the sampling offset scale instead.

\[
(K \ast P)(x, y) = \sum_{\Delta x = -r}^{\Delta x = r} \sum_{\Delta y = -r}^{\Delta y = r} K(x + s\Delta x, y + s\Delta y) \cdot P(x + \Delta x, y + \Delta y)
\]

Changing it for the kernel would require renormalization of the result...
Circular Depth of Field

• Scaling
  • Instead of generating multiple kernels, mixing between them and normalizing in the shader...
  • ... we can change the sampling offset scale instead.

\[
(K \ast P)(x, y) = \\
\sum_{\Delta x = -r}^{\Delta x = r} \sum_{\Delta y = -r}^{\Delta y = r} K(x + \Delta x, y + \Delta y) \cdot P(x + s\Delta x, y + s\Delta y)
\]

So, instead change it for the sampled picture.
Circular Depth of Field

• Scaling
  • Instead of generating multiple kernels, mixing between them and normalizing in the shader...
  • ... we can change the sampling offset scale instead.

\[
(K * P)(x, y) = \sum_{\Delta x = r}^{\Delta x = -r} \sum_{\Delta y = r}^{\Delta y = -r} K(x + \Delta x, y + \Delta y) \cdot P(x + s \Delta x, y + s \Delta y)
\]

Note that we can not change the kernel radius, because loops in shaders need to have fixed conditions.
Circular Depth of Field

$s=0$

$s=1$

$s=2$

$s=3$

$s=4$

$s=5$

Kernel: 32×32
Output: 500×500
Components: 2
Circular Depth of Field

- Separated kernel

\[ K(x, y) = (c-id)^2 \cdot e^{a(x^2+y^2)} \cdot e^{ib(x^2+y^2)} \]
Circular Depth of Field

- Separated kernel

\[ K(x, y) = \mathbb{R} \left[ (c - id)^2 \cdot e^{a(x^2 + y^2)} \cdot e^{ib(x^2 + y^2)} \right] \]

With a 2D kernel we only used the real part of the values, because both our input and output were reals.
Circular Depth of Field

- Separated kernel

\[ K(x, y) = \Re \left[ (c - id)^2 \cdot e^{a(x^2 + y^2)} \cdot e^{ib(x^2 + y^2)} \right] = \]

\[ = \Re \left[ (c - id) \cdot e^{a(x^2)} \cdot e^{ib(x^2)} \cdot (c - id) \cdot e^{a(y^2)} \cdot e^{ib(y^2)} \right] \]

1D kernel 1D kernel
Circular Depth of Field

- Separated kernel

\[ K(x, y) = \mathbb{R} \left[ (c-id)^2 \cdot e^{a(x^2+y^2)} \cdot e^{ib(x^2+y^2)} \right] = \]

\[ = \mathbb{R} \left[ (c-id) \cdot e^{a(x^2)} \cdot e^{ib(x^2)} \cdot (c-id) \cdot e^{a(y^2)} \cdot e^{ib(y^2)} \right] \]

1D kernel

Complex number multiplication

1D kernel
Circular Depth of Field

- Separated kernel

\[
K(x, y) = \Re[(c - id)^2 \cdot e^{a(x^2 + y^2)} \cdot e^{ib(x^2 + y^2)}] =
\]

\[
= \Re[(c - id) \cdot e^{a(x^2)} \cdot e^{ib(x^2)} \cdot (c - id) \cdot e^{a(y^2)} \cdot e^{ib(y^2)}]
\]

\[
e^{ax^2} \cdot (c - id) \cdot (\cos(bx^2) + i \sin(bx^2))
\]
Circular Depth of Field

• Separated kernel

\[ K(x, y) = \Re \left[ (c - id)^2 e^{a(x^2 + y^2)} e^{ib(x^2 + y^2)} \right] = \]
\[ = \Re \left[ (c - id) e^{a(x^2)} e^{ib(x^2)} \cdot (c - id) e^{a(y^2)} e^{ib(y^2)} \right] \]
\[ e^{ax^2} (c - id) \left( \cos(bx^2) + i \sin(bx^2) \right) = \]
\[ = e^{ax^2} \left( c \cos(bx^2) + d \sin(bx^2) + i \left( c \sin(bx^2) - d \cos(bx^2) \right) \right) \]
Circular Depth of Field

- Separated kernel

\[
K(x, y) = \Re \left[ (c - id)^2 \cdot e^{a(x^2 + y^2)} \cdot e^{ib(x^2 + y^2)} \right] = \\
= \Re \left[ (c - id) \cdot e^{a(x^2)} \cdot e^{ib(x^2)} \cdot (c - id) \cdot e^{a(y^2)} \cdot e^{ib(y^2)} \right]
\]

\[
e^{ax^2} \cdot (c - id) \cdot (\cos(bx^2) + i \sin(bx^2)) = \\
= e^{ax^2} \cdot (c \cos(bx^2) + d \sin(bx^2) + i (c \sin(bx^2) - d \cos(bx^2))) = \\
= e^{ax^2} \cdot (c \cos(bx^2) + d \sin(bx^2)) + ie^{ax^2} \cdot (c \sin(bx^2) - d \cos(bx^2))
\]

1D kernel \textbf{real} part

1D kernel \textbf{imaginary} part
Circular Depth of Field

- Separated kernel

\[
K(x, y) = \mathbb{R} \left[ (c - id)^2 \cdot e^{a(x^2 + y^2)} \cdot e^{ib(x^2 + y^2)} \right] = \\
= \mathbb{R} \left[ (c - id) \cdot e^{a(x^2)} \cdot e^{ib(x^2)} \cdot (c - id) \cdot e^{a(y^2)} \cdot e^{ib(y^2)} \right]
\]

\[
\begin{align*}
\text{1D kernel (complex)} \\
V \\
e^{ax^2} \cdot (c - id) \cdot (\cos(bx^2) + i \sin(bx^2)) = \\
= e^{ax^2} \cdot (c \cos(bx^2) + d \sin(bx^2)) + i (c \sin(bx^2) - d \cos(bx^2)) = \\
= e^{ax^2} \cdot (c \cos(bx^2) + d \sin(bx^2)) + i e^{ax^2} \cdot (c \sin(bx^2) - d \cos(bx^2))
\end{align*}
\]

\[
\begin{align*}
\text{1D kernel real part} \\
R \\
\text{1D kernel imaginary part} \\
I
\end{align*}
\]
Circular Depth of Field

• Separated kernel

\[
K \ast P = V \ast V^T \ast P
\]

\[
K \ast P = (R + iI) \ast (R + iI)^T \ast P
\]

\[
K \ast P = (R + iI) \ast (R + iI) \ast P
\]
Circular Depth of Field

- Separated kernel

\[ K \ast P = (R + iI) \ast (R + iI) \ast P \]

Horizontal pass(es)
Circular Depth of Field

- Separated kernel

\[ K \ast P = (R + iI) \ast (R + iI) \ast P \]

Horizontal pass(es)

\[ R \ast P \rightarrow P'_R \]

\[ I \ast P \rightarrow P'_I \]

Vertical pass

\[ V \ast P' \rightarrow P'' \]

(complex multiplication of elements)
Circular Depth of Field

- Separated kernel

\[ K \ast P = (R + iI) \ast (R + iI) \ast P \]

\begin{align*}
R \ast P & \rightarrow P'_R \\
I \ast P & \rightarrow P'_I \\
V \ast P' & \rightarrow P'' \\
\mathbb{R}[P''] & \rightarrow B
\end{align*}

Horizontal pass(es) \hspace{2cm} Vertical pass \hspace{2cm} Result
Circular Depth of Field

- Separated kernel

\[ K \ast P = (R + iI) \ast (R + iI) \ast P \]

\( R \ast P \rightarrow P'_R \)

\( I \ast P \rightarrow P'_I \)

(Real part)

(Imaginary part)

Horizontal pass(es)  Vertical pass  Result

\( \mathbb{R}[P'''] \)

\( B \)
Circular Depth of Field

• Normalization
  • We want to scale the separated kernels such that the sum of the real parts of the final pass will be 1.
  • We need to calculate the real values of the kernel $K$.

$$\overline{V}(x) = \frac{V(x)}{\left| \sum_{x',y'} \Re[K(x',y')] \right|}$$

$$\Re[K(x,y)] = e^{a(x^2+y^2)} \cdot (c^2-d^2) \cos(b(x^2+y^2)) + 2cd \sin(b(x^2+y^2))$$
Examples

The MAX-AVG border mix.
Examples

2D

2×1D
Examples

2D

2×1D
Examples

Gaussian blur
Examples

Circular DOF
Examples

Gaussian blur
Examples

Circular DOF
Examples

- Very slow (unoptimized) demos.
  - Gaussian DOF
  - Circular DOF
- ~2000 LOC
References

- Circular Separable Convolution Depth of Field (Kleber Garcia – EA, SIGGRAPH Talks 2017)
  - https://www.youtube.com/watch?v=QKhydJSbcno
  - https://dl.acm.org/citation.cfm?id=3085022
References

• Circularly symmetric convolution and lens blur (Olli Niemitalo)

• http://yehar.com/blog/?p=1495
References

- **Video Game & Complex Bokeh Blurs**  
  (Dr Mike Pound, Computerphile)

- [https://www.youtube.com/watch?v=vNG3ZAd8wCc](https://www.youtube.com/watch?v=vNG3ZAd8wCc)
References

● Shadertoy implementations

Bart Wronski
https://www.shadertoy.com/view/lsBBWy

Blogpost:
https://bartwronski.com/2017/08/06/separable-bokeh/

Kleber Garcia
https://www.shadertoy.com/view/Xd2BWc

Simplified by Egon Elbre
https://www.shadertoy.com/view/wdXXRS

Filter generator
• Terms
  • https://en.wikipedia.org/wiki/Kernel_(image_processing)

• GPU Gems
  • Depth of Field: A Survey of Techniques
  • Practical Post-Process Depth of Field

• Other Stuff
  • Algorithms for Rendering Depth of Field Effects in Computer Graphics (Survey)
  • MIT Computer Vision: Linear Filters lecture
  • Deep Learning Book Series · 2.8 Singular Value Decomposition, Hadrien J.
  • Another approach to DoF in Unity
    https://catlikecoding.com/unity/tutorials/advanced-rendering/depth-of-field/
  • Thread about DoF
    https://forum.libcinder.org/topic/depth-of-field-blur-weighted-sampling#232860000002682009
  • Single-pass Bokeh DoF
This + (instead of -) cost 10h of debugging.