IceMan - Filtering and Convolution

ice.Image Blur(*x, y, blurType*)

Fast, general-purpose blurring operation. Suitable for most purposes, this operation is essentially constant-time with kernel width. The different types of kernels yield different visual results. Note that each of the examples below produces a result image of a different size: as mentioned before, the rectangle occupied by a result is guaranteed to contain all non-zero contributions from the original image.

Box filtering employs an incremental moving window algorithm which makes it extremely fast. Note that IceMan allows the filter width to be non-integral, and that such widths yield the expected "weighted-sum" result.

Parameters

x

Width of blur in x. In pixels (int).

y

Width of blur in y. In pixels (int).

blurType

Type of blurring kernel (int). Default Box.

Available blur types from *IceBlurType*:

- ice.constants.BLUR_BOX
- ice.constants.BLUR_GAUSSIAN
- ice.constants.BLUR_BICUBIC
- ice.constants.BLUR_CIRCULAR

Example

```
blurType = ice.constants.BLUR_BOX
result = m1_1.Blur(10, 1, blurType)
```

```
blurType = ice.constants.BLUR_GAUSSIAN
result = m1_1.Blur(10, 1, blurType)
```

```
blurType = ice.constants.BLUR_BICUBIC
result = m1_1.Blur(10, 1, blurType)
```

ice.Image Convolve(*x, y, kernel*)

General convolution with arbitrary kernel. The kernel must have x * y entries.

Parameters

x

Size in x of kernel (int).

y

Size in y of kernel (int).

kernel

Convolution kernel (list).

Example

```
// DiscKernel is a function that generates a circular convolution
// kernel 30 pixels in diameter.
disc = DiscKernel(30)
result = fruit.Convolve(30, 30, disc)
```

⚠ This is full-fledged, no-magic-tricks convolution. For large kernels, this can be **very** time-consuming.

ice.Image ConvolveSelective(*kernelImage, pointList*)

This is not a true convolution. Effectively, the operation is a multiplication of the kernel by the channel value at the center of the kernel, and subsequent addition with all the pixels under the kernel. This operation is performed for all points in the specified point list. Note that the points may be non-integral.

Useful for generating "sparkle" or "starburst" effects.

Parameters

kernelImage

Image to be used as convolution kernel (ice.Image)

pointList

List of points at which convolution is to be performed (list of lists of 2 ints)

ice.Image ConvolveTrig(*kernelImage, triggerImage*)

This is not a true convolution. Effectively, the operation is a multiplication of the kernel by the channel value at the center of the kernel, and subsequent addition with all the pixels under the kernel. This operation is performed for all points in the trigger image that are not zero.

Useful for generating "sparkle" or "starburst" effects.

Parameters

kernelImage

Image to be used as convolution kernel (ice.Image)

triggerImage

"Trigger" image (ice.Image)

Example

```
# Create a starburst "kernel"
sparkleType = ice.constants.SPARKLE_FRAUNHOFERSLITS
size = [128, 128]
center = [64, 64]
kernel = ice.Sparkle(sparkleType, size, center, ice.constants.FRACTIONAL, 23)

# Translate the kernel so that it's centered at the origin
kernel = kernel.Translate((-64, -64))

# Perform selective convolution ..
result = image.ConvolveTrig(kernel, image.Shuffle([0]).Gt( ice.Card(ice.constants.FLOAT,[7.9])))

# and add a fraction of the result back into the original
# image
result = result.Add(result.Multiply( ice.Card(ice.constants.FLOAT,[0.05]) ))
```

⚠ The trigger image is typically generated by thresholding an input image.

ice.Image Dilate(*pixels*)

Morphological dilation/erosion operation. Positive *pixel* values result in dilation and negative ones in erosion. Values may be non-integral, and results vary as expected.

⚠ Dilation is done by filling all values in a neighborhood with the maximum value, and erosion by filling with the minimum value. It is particularly useful for tweaking matte edges.

Parameters

pixels: Amount in pixels to dilate or erode (int).

ice.Image DirectionalBlur(*width, angle, blurType*)

Directional blur operation.

Parameters

width

Blur width in pixels (int).

angle

Blur direction in degrees (float).

blurType

Type of blur (int).

Example

```
blurType = ice.constants.BLUR_GAUSSIAN
result = cdev.DirectionalBlur(20, 20, blurType)
```

ice.Image GaussianBlur(*blurWidth*)

Simple Gaussian blur operation. The operation is separable, and is thus performed in one dimension at a time, but there's no other attempt at optimization. Intended for use only in special cases where *Blur* is not suitable.

Parameters

blurWidth: Width of blur in pixels (list).

ice.Image Gradient()

Compute the gradient of an image using center-differencing. Each channel in the original image results in two channels in the result, corresponding respectively to the horizontal and vertical components of the gradient vector field.

ice.Image Lowpass(*freq*)

This operation performs convolution with a bicubic (which approximates a sinc function) such that the image is band-limited to the specified spatial frequency. The frequency is expressed as a fraction of the maximum spatial frequency representable in the image.

Parameters

freq

Spatial threshold frequency in x and y (list).

ice.Image VariableBlur(*width, widthRange, blurType*)

Variable-width blur operation. The extent of blur at each pixel is specified by an image. The *widthRange* argument specifies the blur widths corresponding to 0 and 1 in the blur-width image.

Parameters

width

Blur width image (ice.Image).

widthRange

Blur width scale (list)

blurType

Blur type (int).