Rain

Jayke Samson Nguyen

August 9, 2024

1 Explanation

1.1 Problem Statement

Given an image, apply a distortion map to the pixels in the image whilst preserving the original flux of the image.

1.2 Derivation

Ultimately, this algorithm takes an input kernel, divides it into multiple discrete pieces with normalized volumes, then sums all the values of the the pixels that fall into a given pixel (like rain).

We mathematically define the image to be a matrix:

$$I = \begin{bmatrix} A_{1,1} & \dots & A_{1,s_x} \\ \vdots & \ddots & \vdots \\ A_{s_y,1} & \dots & A_{s_y,s_x} \end{bmatrix}$$

A single pixel with indicies i, j has the value $A_{i,j}$.

We distinguish that indicies i, j correspond to the row and column of the pixels in the matrix and x, y correspond to the pixel center coordinates.

$$x = \frac{d}{2} + j$$

$$y = \frac{d}{2} + i$$

Confusingly, x is related to j, the column index and y is related to i, the row index, a consequence of the fact that the "origin" of the matrix in math is typically the top left corner. We define d, the pixel size (this is arbitrary and usually set to 1).

We define an arbitrary coordinate transformation which yields the following transformation:

$$(x, y) \to (x', y')$$

Where:

$$f_x(x, y) = x'$$

$$f_y(x, y) = y'$$

are some functions that perform the transformation.

Futhermore, we define a normalized kernel $\Omega(x, y)$ that distributes the pixel value across multiple pixels in the image. This kernel is always normalized such that:

$$\frac{1}{C} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \Omega(x, y) dx dy = 1$$

Where C is the normalization factor. We also define the unnormalized kernel $\omega(x, y)$. The normalized kernel is thus:

$$\omega(x, y) = \frac{\Omega(x, y)}{C}$$

When the kernel is applied to the pixel, the original value of the pixel is preserved. That is:

$$A_{i,j} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \omega(x, y) dx dy = A_{i,j}$$

Smoothing all the pixels over the original image I and creating new image I':

$$\sum_{i=1}^{s_x} \sum_{j=1}^{s_y} \left[A_{i,j} \int_{j}^{j+d'} \int_{i}^{i+d'} \omega_{i,j}(x', y') dx' dy' \right] = I'_{i',j'}$$

Where $\frac{d}{n} = d'$, the new pixel size and $n \in \mathbb{Z}^+$.
In practice, when applying the kernel to an image, we must use discrete sums:

$$\sum_{i=1}^{s_x} \sum_{j=1}^{s_y} \left[A_{i,j} \sum_{u=1}^{n} \sum_{v=1}^{n} \omega_{i,j}(x'_v, y'_u) \right] \approx I'_{i',j'}$$

We define n to correspond to the number of rectangles we are using to approximate the integral. We also define $x'_v = \frac{d'}{2} + v + j$ and $y'_u = \frac{d'}{2} + u + i$ are the coordinates of the mapped pixel centers. Since we are using pixel centers, this is a midpoint Riemann sum and error goes as $O(n^2)$.

However we must note that the product $A_{i,j}\omega_{i,j}(x', y')$ almost always affects more than one pixel since after translation and kernel smoothing, the pixel becomes spread out from its original value. The above sums represent the contribution of all pixels into one value. For all i', j' there are a completely different set of sums that must be computed individually but in practice this is easy to speed up with $\sim O(s_x s_y n^2)$ complexity.

2 Kernels

2.1 Gaussian

The Gaussian kernel is defined over $-\infty < x < \infty$ and $-\infty < y < \infty$ is given by:

$$\Omega_G(a, \sigma_x, \sigma_y, x, y) = a \exp\left[-\left(\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2} \right) \right]$$

$$C_G = \pi \sigma_x \sigma_y$$

Where a is the amplitude of the distribution, σ_x and σ_y are width parameters.

This is the simple case of a single pixel represented by a Gaussian probability distribution, with the flux corresponding to the area under the Gaussian. Typically, for high accuracy (~1%), n must be greater than 8.

2.2 Lanczos

The 2D Lanczos kernel is only defined over $-a < x < a$ and $-a < y < a$ is given by:

$$\Omega_L(a, x, y) = \text{sinc}(x)\text{sinc}\left(\frac{x}{a}\right) \text{sinc}(y)\text{sinc}\left(\frac{y}{a}\right)$$

Where a is an integer parameter that determines the size of the kernel, typically equal to 2 or 3, corresponding to 2 or 3 pixels in width.

For $a = 3$:

$$C_L = \left(\frac{2}{\pi}(2\mathrm{Si}(4\pi) - \mathrm{Si}(2\pi))\right)^2 \approx 0.994$$

The Lanczos kernel is special in the sense that Lanczos resampling works by applying the Lanczos kernel to a given data point, and evaluating the linear combination of all the overlapping Lanczos functions at a given sample point. In the case of Lanczos 3, the kernel extends in 3 pixels from the center, thus multiple pixel overlapping occurs for almost all pixels. See Figure 1. In short, if this kernel is used, $n = 1$ must be used. A nice byproduct of this is that this reduces the complexity to $\sim O(N^2)$.

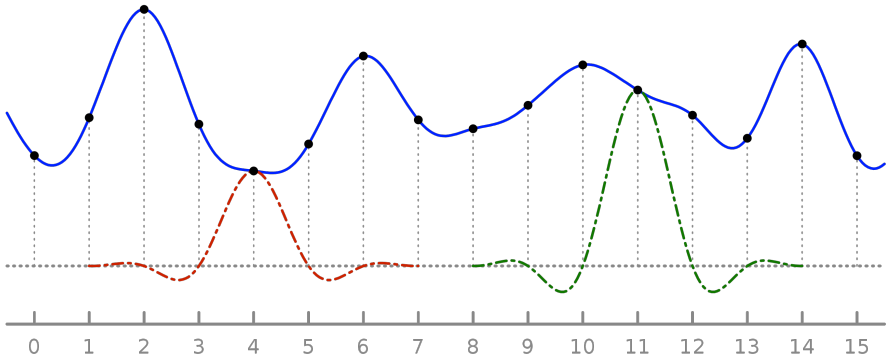

Figure 1: Lanczos 3 functions applied to two representative points in red and green. The blue curve is the sum of all smoothed points. The sampling is done on any arbitrary point on the blue curve.