Algorithmic Art

Oliver Kirkpatrick

March 20, 2021

1 Introduction

2 Some Important Concepts

2.1 Alpha Channels

2.2 Perlin Noise

2.3 Vector Fields and Massless Particles

3 A Gentle Blend

3.1 Premise

While the following exercise won't exactly shake computational art lovers to their core, it demonstrates the importance of the aforementioned alpha channel. This alpha channel will allow blending of images, giving smooth transitions (and low computational cost filler frames) in the output images and clips we produce.

Suppose there are two images (for now, we'll just label them matrices), $\mathcal{I}_{1,2}$. Both are three channel BGR images:

(a) $\mathcal{I}_1$ (b) $\mathcal{I}_2$

Figure 1: Images to be blended.

The most obvious method to transition from one image to another is just do exactly that—just stop showing one image and show the other. But this is a very discrete transition, furthermore, it isn't very

friendly on the eye. In spaghetti speak: we ah want to see ah the transition! Hence, we must find a way to slowly change from one image to another, such that if the transition were to take long enough, you would not even notice the change unless you looked at a start and finish frame—yes, we are indeed looking for *that* level of continuity and smoothness.

3.2 The Mechanism

Recall a problem of a similar (yet four dimensional) vein: massless particle integration over a discrete wind vector field for HAB path prediction.

But how is this even remotely a similar problem?!

In the Ballet path prediction algorithms, the massless particle (HAB) exists in a *very* (> 15 km)coarse vector field. Deep inside the bowels of the Ballet library is a spatiotemporal interpolation engine, which takes the concept of interpolation into four dimensions! It is the single dimension version of this that is applicable to our problem. Take some "in between" point, α, perhaps 0.45 (45% along the "line"), between two values, $v_{1,2}$, 27 and 33, respectively. To determine what the value of α should be based on its position relative to $v_{1,2}$ and their values, the following equation can be used:

$$v_\alpha = v_1 + \alpha\,(v_2 - v_1) \tag{1}$$

Alternatively, we could use the mathematical equivalent which uses only addition on $v_{1,2}$:

$$v_\alpha = v_1(1 - \alpha) + \alpha v_2 \tag{2}$$

With either equation, we achieve the following result:

$$\begin{aligned} v_\alpha = & \\ & 27 + 0.45\,(33 - 27) \\ & 27 + 2.7 \\ & = 29.7 \end{aligned} \tag{3}$$

$$\begin{aligned} v_\alpha = & \\ & 27\,(1 - 0.45) + 0.45 \cdot 33 \\ & 14.85 + 14.85 \\ & = 29.7 \end{aligned} \tag{4}$$

It should not be too great a leap of logic to subsitute our images $\mathcal{I}_{1,2}$ into either 1 or 2, and utilize scalar multiplication on matrices to accomplish what the equations do:

$$\mathcal{I}_\alpha = (1 - \alpha)\,\mathcal{I}_1 + \alpha\mathcal{I}_2 \tag{5}$$

Programmatically, when this is implemented in OpenCV [1]

References

[1] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.