An efficient and user-friendly tone mapping operator

Mike Day, Insomniac Games
mday@insomniacgames.com

At Insomniac we've been using the well-known Reinhard tone mapping operator for a while now, but have found its parameterization unintuitive, and its curve shape to be lacking the useful 'toe' feature. Presented here is an efficiently computed alternative model which has been designed to qualitatively capture the toe and shoulder features while at the same time using intuitive parameters.

The initial inspiration comes from [Reinhard 02]. Consider remapping a single input HDR value (either luminance or a colour channel) to an output LDR value in the range 0 to 1. As a first step, we'll normalize the input value by dividing it by the image's average luminance.

As per Reinhard, we then select a *white point* – the input level at which (and above which) the output value will be 1. This becomes our model's first parameter, w, expressed as a multiple of average luminance.

The simplest mapping we can apply is a straight line which maps 0 to 0, and w to 1:

$$f_0(x) = \frac{x}{w}$$

with the final value clamped to the interval [0,1] (a convention we'll assume throughout).

Importantly, though, we wish to be able to give a 'shoulder' to the curve shape, causing it to bulge upwards so that it's steep at the beginning and shallow at the end, and forcing the high input luminances to bunch together near the output value 1. Indeed in the absence of a finite white point, we'd like it to be asymptotic to the output value 1. A simple way to achieve this is to use a hyperbola with vertical and horizontal asymptotes – let's say the function f_1, where

$$f_1(x) = \frac{p}{x+q} + r$$

This appears to have 3 degrees of freedom, but we also have two constraints, namely that the curve should pass through the points $(0,0)$ and $(w,1)$. By applying these constraints we can eliminate any two of p, q, r. If we choose to eliminate p and r we we can express f_1 in terms of the single parameter q:

$$f_1(x) = \frac{x(q+w)}{w(q+x)}$$

$$= \frac{x}{\left(\frac{q}{q+w}\right)w + \left(\frac{w}{q+w}\right)x}$$

$$= \frac{x}{(1-s)w + sx}$$

with $s = \frac{w}{q+w}$. The parameter s controls the amount of blending between a straight-line curve and a purely asymptotic curve, with 0 and 1 yielding the two extremes, and intermediate values providing varying amounts of shoulder to the curve. We'll refer to s as the *shoulder strength* parameter. Figure 1 shows a plot of function f_1 for a selection of values of s ranging from 0 (the straight line) to almost 1.

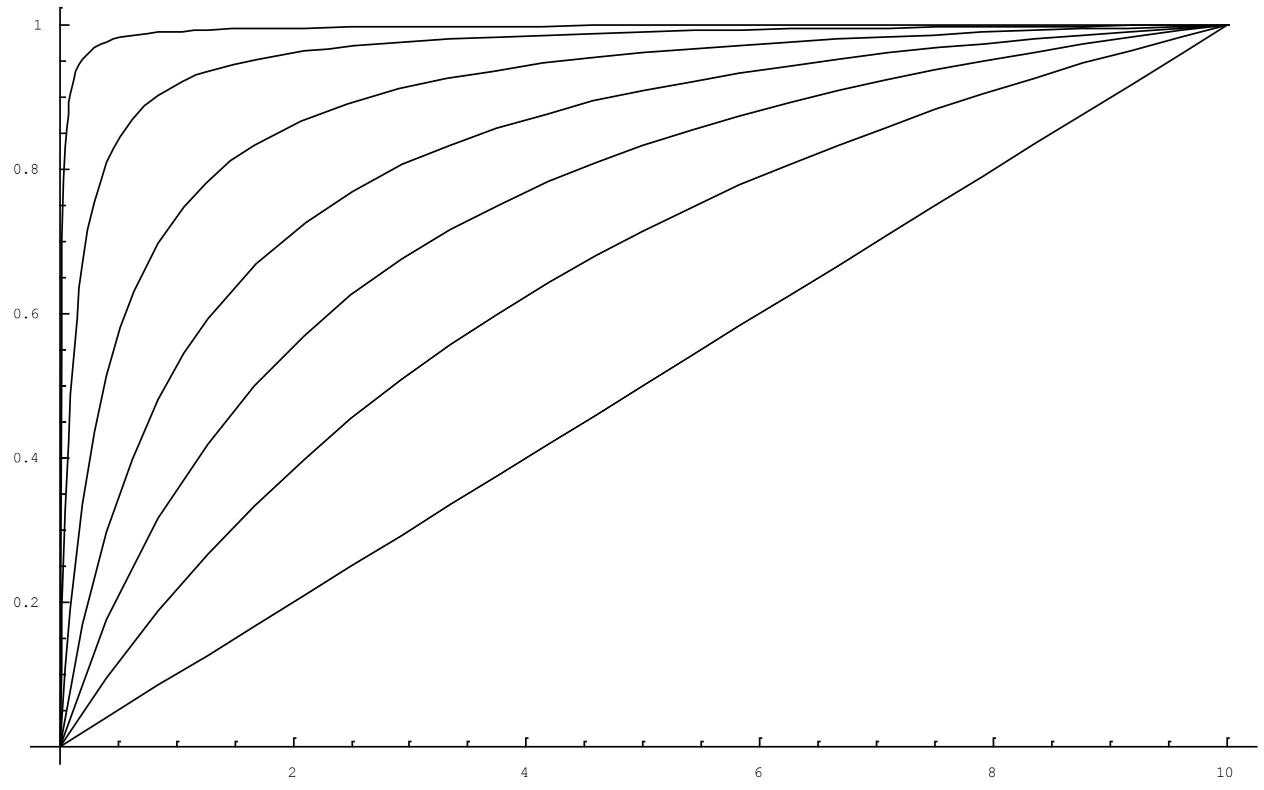

Figure 1: the effect of varying the shoulder strength

Note that the form of the function is just one linear polynomial divided by another, and remains so even with a change of origin. This is not intended to be an empirically correct function, merely one which gives the desired aesthetic qualities at little computational expense.

To achieve a filmic look, as per [Hable 10], we need our curve to have not only a shoulder but also a 'toe' – a small region of opposite curvature at the bottom end. One simple way we can achieve this at little expense is by composing our curve piecewise out of two such hyperbolic sections. In this case, the key to the toe's parameterization is to observe what happens to the shoulder curve when the shoulder strength is allowed to become negative.

Figure 2 shows the function f_1 plotted for a range of shoulder strengths from zero downwards. The one with highest curvature has a shoulder strength of -300, and it becomes clear that the full range of curves is covered by the interval $s \in (-\infty, 0]$. If we wish our toe strength parameter – call it t – to operate in the same manner as the shoulder strength s, we'll need to remap the one interval to the other.

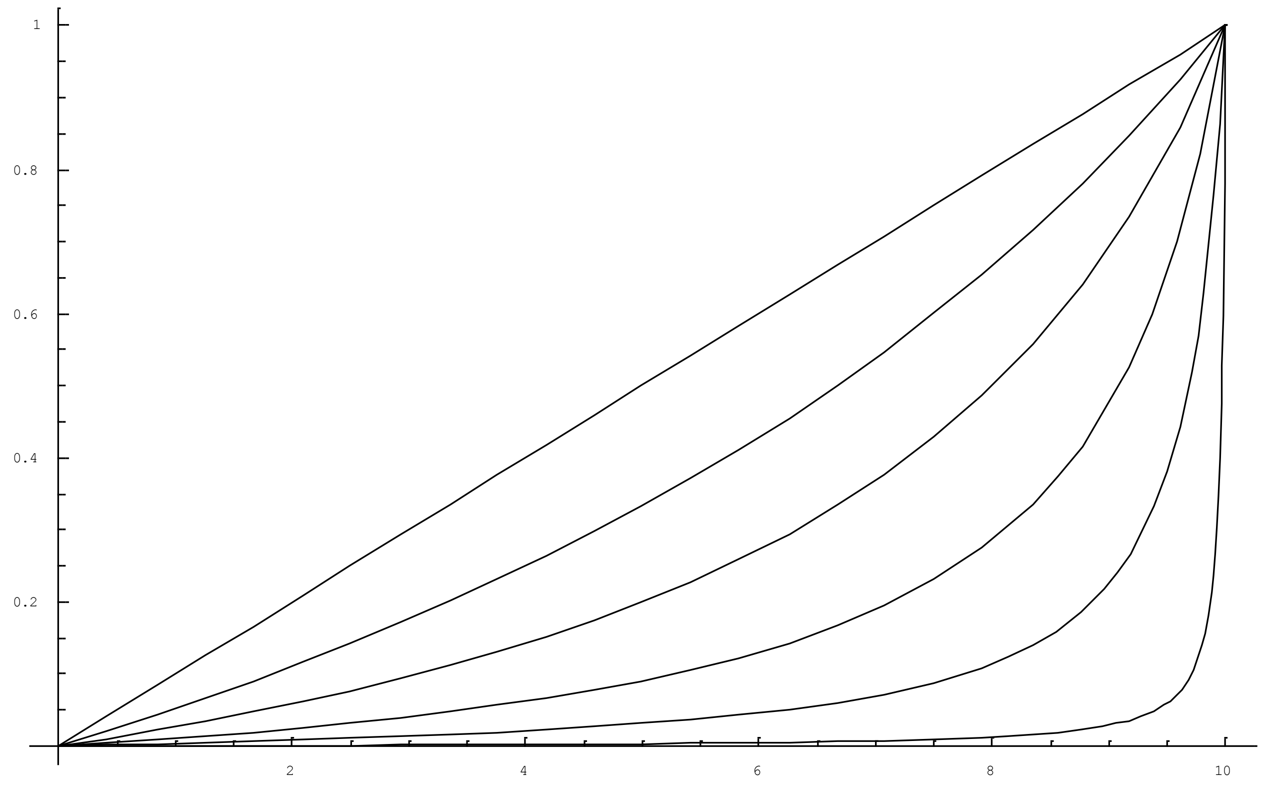

Figure 2: negative shoulder strength values can be remapped to toe strength values

A simple remapping which works is

$$t = \frac{s}{s-1}$$

The inverse of this mapping is

$$s = \frac{t}{t-1}$$

and if we insert this into the expression for f_1 we obtain

$$f_2(x) = \frac{x}{\left(\frac{1}{1-t}\right)w - \left(\frac{t}{1-t}\right)x}$$

$$= \frac{(1-t)x}{w-tx}$$

Note that f_2 is really the same function (or family of functions) as f_1, just using the toe strength parameter instead of the shoulder strength one.

Now we wish to piece the toe and shoulder together into a single curve. We'd like to have control over where along the horizontal axis the toe gives way to the shoulder, so we'll introduce a new parameter c, the *cross-over point*. If $x < c$ we'll choose the toe, and if $x > c$ we'll choose the shoulder. At $x = c$ we can pick either one, assuming we've joined the curves correctly.

Since some work will be required to formulate new toe and shoulder curves to satisfy the new requirements, this is a good time to introduce our final parameter, the *black point*, which we'll denote by b. This plays the complementary role to the white point: input values at or below b are mapped to zero. The black and white points are closely analogous to the near and far depth values used in mapping camera-space depth to [0,1] depth buffer values. The introduction of b comes at no additional run-time cost, since it doesn't change the functional form of the curves – merely the coefficients.

Let's therefore assume that our final toe and shoulder functions, which we'll name $T(x)$ and $S(x)$ respectively, will each take the form of one linear polynomial divided by another, with coefficients to be determined in terms of the parameters b, c, w, s, t. The functions must satisfy the following constraints:

- the toe must pass through the black point;
- the shoulder must pass through the white point;
- the toe and shoulder must coincide at the cross-over point.

A simple procedure for satisfying these constraints is to appropriately scale and translate the functions f_1 and f_2 from earlier – for example f_1 requires its input range to be remapped from [0,w] to [b,c], and its output range to be remapped from [0,1] to [0,$T(c)$]. There is one remaining unknown to be found, namely the value of both functions where they join, say $T(c) = S(c) = k$. In order to find the value of k we impose one additional constraint:

- the toe and shoulder must join with tangent continuity.

The algebra is left for the reader, but the resulting value is

$$k = \frac{(1-t)(c-b)}{(1-s)(w-c)+(1-t)(c-b)}$$

and the resulting expressions for the toe and shoulder functions are

$$T(x) = \frac{k(1-t)(x-b)}{c-(1-t)b-tx}$$

$$S(x) = \frac{(1-k)(x-c)}{sx+(1-s)w-c} + k$$

Figure 3 shows a sample curve, with $b = 0.5,\ c = 2,\ w = 10,\ t = 0.7,\ s = 0.8$. In practice the black point and cross-over point would typically be set considerably lower; the values were chosen here to show the salient features of the curve.

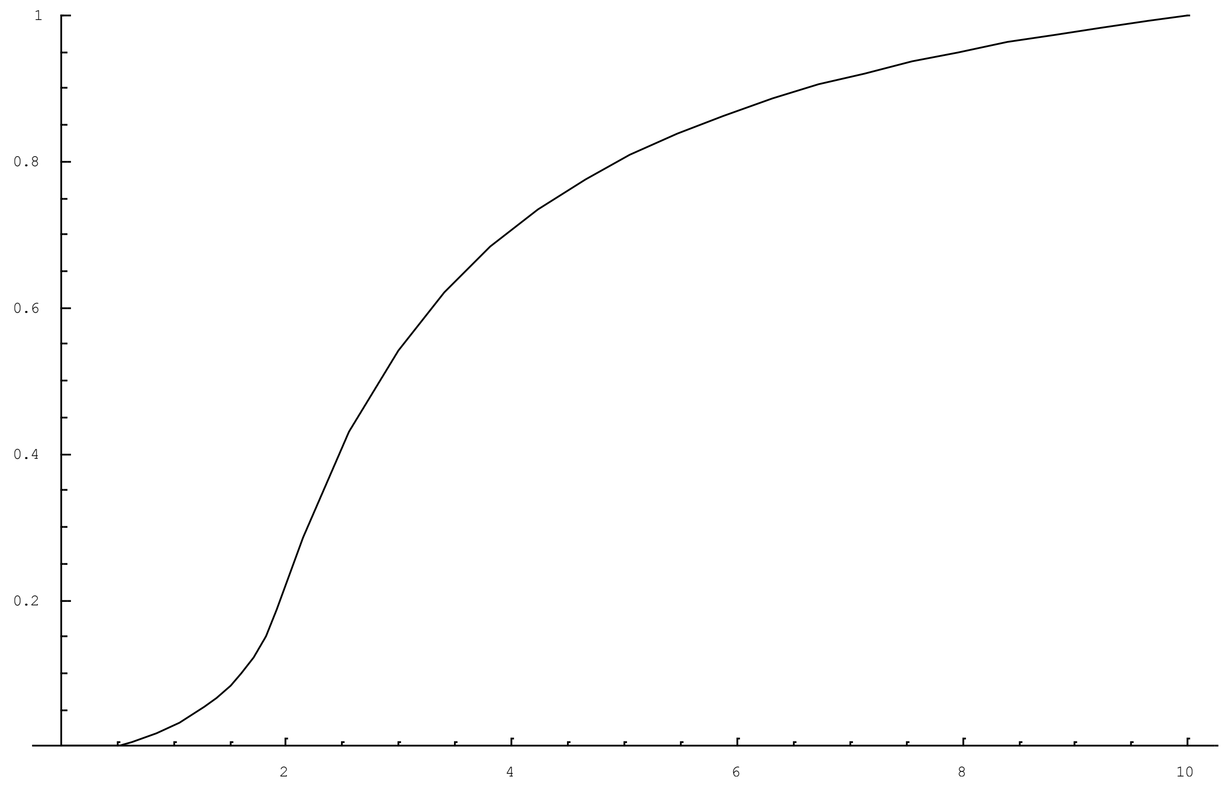

Figure 3: final tone mapping curve with sample values.

Expressing this curve in a pixel shader is a simple matter of using a remapping function such as the following:

```
float Remap(float x)
{
  float4 coeffs = (x < cross_over_point) ? toe_coeffs : shoulder_coeffs;
  float2 fraction = coeffs.xy * x + coeffs.zw;
  return fraction.x / fraction.y;
}
```

The division by average luminance, needed as a first step to normalize all pixel values, can be made cost-free by baking it into the constants in the `Remap()` function shown above.

Notes:

1. Depending on taste, it might be preferable to apply a non-linear mapping upfront to the s and t intervals, to counteract their rather sluggish onset.

2. If better filmic accuracy is desired, a linear section of the curve can be spliced between the toe and shoulder, requiring the use of a second cross-over point. The calculations are left to the reader.

Bibliography

[Hable 10] “Uncharted 2: HDR Lighting”, John Hable, GDC 2010 presentation.

[Reinhard 02] “Photographic Tone Reproduction for Digital Images”, Erik Reinhard et al, Proc. SIGGRAPH 2002.