The Math of Comp

Intro

I'm Ben McEwan, and I'm a Compositing Supervisor at Image Engine here in Vancouver. Some of you might know me from Ben's Comp Newsletter, my Nuke-focused blog, or my popular Python for Nuke 101 course, which has helped over 400 Compositors (and counting) learn Python! In addition to these things, I also enjoy teaching others about Comp, and have done so at Vancouver Film School and at Langara Centre for Entertainment in the past.

Thank you for throwing a few dollars my way to support this course, and everything I'm creating! I'm excited to share some knowledge with you that I hope will deepen your technical understanding of Nuke, and some of the fundamentals behind what Nuke's tools are actually doing to the pixels in your images.

- To start, we'll cover the math of how the Grade node works
- Followed by a demonstration of how different methods of matching blacks affect your image differently
- Next, we'll take a look at the transform node, and specifically how Transforming your image filters the pixels, and why concatenation is important
- Finally, we'll look at how data on a pixel level can help us do pretty interesting things in a flexible way using ST maps.

Lesson 01a

The Grade Node.

One of the most commonly used nodes in Nuke is the Grade node. While Compositors know what the sliders do visually, fewer Compositors understand the relationship between the 7 knobs, and how they manipulate the pixels in your image.

The Grade node's knobs can be divided into two distinct sections:

1. The first 4 knobs are intended for remapping the lowest & highest values of an image to new values
2. The last 3 knobs give Compositors global control on pixel brightness and contrast.

To start, the **Blackpoint** knob is remapping our darkest pixels to the defined value. If we change the blackpoint knob to a value of 0.1, it will crunch our image, as we're using the knob to tell Nuke "A value of 0 is actually now equal to a value of 0.1". Because we don't like negative pixels, any value that was previously between a value of 0 and 0.1 will now be set to 0.

The Blackpoint and **Lift** knobs share a special relationship, as we can use Lift to remap our Blackpoint to a new value. For example, if we set both our Blackpoint and Lift knobs to a value of 0.1, our image goes back to normal! Using both knobs together is common when matching CG black levels to that of a plate -- using Blackpoint to sample the darkest point of a CG object (setting black to 0), then using Lift to remap the darkest value of the CG to a dark value sampled from a plate.

The **Whitepoint** and Gain knobs share a similar relationship, however this time, we're affecting the brightest pixels! As we reduce the value of our whitepoint to 0.7, we are brightening the image, as we're telling Nuke "A value of 1 is now equal to 0.7", and it will remap the pixels in-between in a linear fashion. All the values above a value of 1 will also get brighter as the slope of our line has increased.

When we set both our Whitepoint and **Gain** knobs to 0.7, our image goes back to its default state, as we're previously remapping the whitepoint values of 0.7 up to 1, and then using gain to remap the whitepoint back to 0.7.

Next, let's talk about the difference between **Lift** and **Offset**. Mathematically, offset is simply adding value to our pixels. To demonstrate, I'll sample one of these colours on the side of the image, and watch what happens when we increase the value… The entire curve moves upwards! Our pixels simply have a value of 0.1 added.

Instead, when we use the same value to adjust the **Lift** knob, you can see a difference! The Lift knob lifts the blacks in our image, using the Whitepoint value as it's pivot. To demonstrate more-clearly, I'll set the value of Whitepoint back to 0.7, and watch what happens as we adjust Lift.

Visually, it's hard to see a difference between Lift & Offset, which you can see as I flip back and forth between both examples, but mathematically, which knob you adjust can make a huge difference to how your pixels get manipulated as more nodes get stacked up downstream.

Moving onto the **Multiply** knob. The math behind it is simply multiplying pixel values -- sampling the orange colour in the Kodak logo we get values of 1 in Red, and almost 0.3 in Green. Setting the Multiply knob's value to 2 results in a new Red value of 2, and a Green value of almost 0.6! This math works exactly the same way in the Gain knob, however, Multiply is intended to be used as a global multiplier for your image, vs. the Gain which is intended to be used to remap an image's Whitepoint to a new value. As a practical example, I often find myself using the Gain knob to set my CG render's overall luminance, and the Multiply knob to make any minor colour adjustments.

Lastly, there is **Gamma**. Gamma is a mathematical power function, which adds a curve to our graph, and is generally used by Compositors to add or reduce contrast in an image. Up until now, all other knobs preserved the linearity of our image, as the graphical representations were showing us a straight line with a constant increase. However, as gamma introduces a curve, it breaks our image's linearity.
Curiously, there is a difference between the Gamma knob in the Grade node, and the Gamma node itself...

To prove this, I have both a Gamma and a Grade node, both with the Gamma values set to a value of 2, and am comparing the difference via a Merge node. The result shows us that there is a difference in the highlights, so why is that? The answer lies in the math behind how each node works.

Let’s take a look at another example: here I have a Ramp with values between 0 and 3. I have a Grade node, boosting the gamma to 2 in the Red channel, and a Gamma node boosting the gamma to 2 in the green channel. Now we are able to use the **Sampler** node to sample a live colour curve!

Blue is a straight line from 0 to 3, just as our Ramp demonstrates, however you can clearly see that there is a difference between the Red and Green channels. If we zoom in closer, the curve only deviates as values extend beyond a value of 1! What is happening here, is the Grade node is interpolating values above one to continue as a linear slope -- similarly to how you would interpolate an animation curve to make it continue beyond the last keyframe. The Gamma node, on the other hand, is continuing the power function up to the image’s brightest value, or more simply, continuing to introduce a bend in the curve.

At this point you’re probably wondering which Gamma is most correct? The answer is, it depends on the application! Colour science is a vast topic, and one we don’t have time to unravel in-depth today… But technically speaking, the Gamma node produces the mathematically-correct response, and operates the same way as LUTs and CDLs would. As an artist working in a pre-determined colour pipeline, when you’re using gamma to grade your image and make creative decisions, the difference is so small that it might not matter. However technically, it’s actually doing different things to your pixels!

I thought it would be interesting to show you the formula that the Grade node uses, with the correct Gamma function in place! *(walk through colour coded demo)*

If you were to use this formula in an Expression node, it would look like this! The only difference is we’re specifying each colour channel that we’re operating on.

Lesson 01b

Matching Black Levels.

So with this knowledge in mind, I'd like to ask you: what is the most technically-accurate method of matching blacks? I'll give you a few seconds to think about it…

This is an interesting question, as there really isn't a clear, correct answer. For Compositors, our plate is always the best reference, and provides all the answers we need. Depending on the camera used, the camera's ISO settings, how much light is in the scene, etc. the shadowy areas of our plates can appear slightly different.

When matching blacks, the two things you want to look out for are:

1. How do the darkest areas of objects in your image change in relation to the distance from the camera?
2. How much detail is there in the darkest areas of the image? That's what we'll be discussing here!

Let's bring back our friend Marcie, and our graph, to look into different methods we can use.

In a scenario where you're integrating CG into a plate that was shot in direct sunlight, you'll likely have a lot of detail in the shadows, which we'll need to match. In these cases, I like to use Nuke's Viewer exposure slider to check out the highlights of my images first. Does the plate have crazy-hot values, or are the highlights a little over 1? What about our CG renders? Do they have similar characteristics?

In a shot where the plate has very hot highlights, and my CG has values just over 1, I'll likely start by using the **Offset** knob to match blacks. As we saw before, offset simply adds value to an image, and won't compress or expand the range of our CG, which means our renders will retain their detail.

In a plate where there is a lot of detail in the blacks, and highlight values barely over 1, I would choose **Lift**. This is because Lift doesn't compress our black details, but it also doesn't change our whitepoint. However, as you remember, there is a relationship between the blackpoint, lift, whitepoint, and gain knobs, which allows for even finer control of matching your CG to your plate where needed. When matching blacks, you should only change what

you need to, in order to save yourself time, keep your Nuke script clear, and only process your pixels as much as you need to…

In some plates, such as those shot in low-light conditions, a camera's sensor may not be able to pick up a lot of detail in the darkest areas of a scene. In this situation, when you expose-up the shot, you will notice that images start to have flat dark colours, or clamped values. To match this, the obvious solution might be to use a **Clamp** node, which is telling Nuke to throw away pixels below a certain value, and replace them with that same value.

While this is an extreme example, it happens to a small extent in almost all low-light plates. However using a Clamp node makes our image look flat and horrible, with a sudden transition between our clamped pixels, and every other pixel. That's represented by this sharp elbow in our graph. We sometimes need to match this clamp in our CG, but we don't want this type of result, and that's where the **Toe** node comes in.

The **Toe** node is doing a similar thing to our pixels as a Clamp, but also includes a mathematical power function to smooth out that sharp elbow, which gives our image a smoother transition between pixels.

So to loop back to our original question, what is the most technically-correct way to match blacks? It depends on your plate's reaction to light, and you can use Offset, Lift, or Toe to mimic any situation!

Lesson 02a

Pixel Filtering.

Whenever we manipulate an image in Nuke, we're filtering the pixels in some way. The most obvious example of pixel filtering happens when we transform an image. The **Transform** node allows us to use sub-pixel values, but how does Nuke know what those are? What about when we Scale, or Rotate an image? How does Nuke figure out which pixels go where?

When doing any transformation of pixels, Nuke uses a filtering algorithm to interpolate pixels & come up with new values that make sense. We have 11 different filters to choose from in Nuke, and I'd like to demonstrate what they all do. With this knowledge, you'll have greater control over the pixels in your comps!

In this setup, I have 4 transforms rotating the image 45 degrees, and I am breaking concatenation by putting a Grade node in-between each Transform operation. Node concatenation is an important topic, which we'll cover in the next section of this talk!

To start, this is what our RAW image looks like, with zero transforms and no filtering needed. Now, let's switch on our transform setup. The first filter to talk about is the **Impulse** filter. This filter is actually not filtering our image at all -- it's simply remapping pixels to a new location and making no attempt at preserving a nice, smooth relationship between each pixel. The results are horrible, and are a perfect example of why pixel filtering exists! Perhaps the only time I would use an Impulse filter is when Clone-painting, to avoid any soft-edged paint strokes. This is what the "round" checkbox also does, as the tooltip explains.

The next, and most commonly used filter in Nuke is the **Cubic** filter, as it's the default in Nuke! As we toggle between Impulse and Cubic, you can see the graphical representation of our pixel filter gets wider. When we filter an individual pixel, we are also interpolating values from surrounding pixels to make a new pixel value. As this graph gets wider, we are simply using more of the surrounding pixel values to interpolate our individual pixel. As a result, this smooths our image, and gives a nicer result!

As we switch to the **Keys** filter, you'll notice the graph dips below the baseline. This represents that we're introducing negative pixel interpolation, which produces a sharpening effect as we're filtering our pixels.

As we switch to **Simon**, and **Rifman**, our pixel filtering remains moderately smooth like the Cubic filter, but introduces an increased amount of sharpening. Using a sharper filter is helpful for retaining detail in an image, however it will also introduce sharpening artefacts like dark haloing around bright edges, as we can see on both our checkerboard, and on Marcie.

The effect of this artefacting in pixel filtering can be reduced by converting your image to a logarithmic colourspace before doing your transform, and converting back to linear afterward. A log colourspace changes the relationship between pixel values, as it compresses your image down to values between 0 and 1, and introduces a logarithmic curve. Looking at Marcie & our graph in log space, notice how most of our pixel values live in a mid-tone range. This means there are less-extreme differences between the brightest & darkest pixels in our image, and we get a smoother result when filtering our image.

While the Keys, Simon and Rifman filters have a sharpening effect on our image, the Mitchell filter has an overall softening effect. This is demonstrated in the pixel interpolation graph getting wider.

The **Parzen** filter removes all sharpening entirely, and interpolates over a wider range of pixels, while the **Notch** filter takes this to the extreme. As the center-point of this graph gets lower, our resulting filtered pixel is using less of the original, pre-filtered pixel's value. A smoother pixel filter is ideal in situations where you might need to prevent buzzing edges, or moiré patterns.

The next three filters are more-modern pixel filters. **Lanczos 4** finds a nice balance between the smoothness of our default Cubic filter, while maintaining some sharpness in our image. **Lanczos 6** is the same filter, however interpolated over 6 surrounding pixels, instead of 4.

Finally, the **Sinc4** filter. It is easily the most heavy-handed sharpening filter of the lot, and the only use I've ever found for it is to creatively munge up an image to make it look like low-quality CCTV footage.

Lesson 02b

Node Concatenation.

Like in a Grade node, Nuke's Transform node calculates what's happening to each pixel in a single mathematical operation, by using an underlying matrix. That means that if you're Translating, Rotating and Scaling your image, it's only filtering a pixel once!

When we stack multiple nodes doing transform operations in a row, including Reformat nodes, Tracker nodes, CornerPin nodes, etc. Nuke is smart enough to combine all these node's underlying transformation matrices into a single operation, and again, only filters our image once!

In our Nuke setup, this tree with the Grade nodes isn't concatenating, whereas the one to the left with only Transform nodes is. If I switch to the concatenating branch, our image quality becomes significantly clearer! As we cycle through the image filters again, we can see there is still a difference, but it's much less noticeable as we're only filtering the image once, instead of four times.

As you can see, paying close attention to node concatenation, and limiting the amount of times you filter the pixels in your image, will result in a far cleaner & sharper end result!

Lesson 03

Manipulating Images with Pixel Data, via ST Maps.

In the last section of this presentation, we're going to explore how you can effectively use an ST map to warp an image with maximum control, and minimal image filtering.

Understanding what an ST map is, and how you can use one to your advantage, quickly pays dividends when working in Nuke. An ST map is an image where every pixel has a unique Red and Green colour value that corresponds to an X and Y coordinate in screen-space. You might also see ST Maps looking more cyan & purple vs. the standard red & green, however the only difference here is the blue channel contains a value of 1 instead of 0 in every pixel. This has no effect on our ST map, as we're only working with two coordinates, X and Y, which again, correspond to Red and Green channel data.

Looking at only the Red channel shows up a ramp from left to right, and looking at the Green channel shows us a ramp from bottom to top. We can easily create this by plussing two ramp nodes together, but I prefer using an expression node as it's faster to set up. Let's create a new expression node and break down what's actually happening here...

Typing x, and sampling pixels shows that as we move left to right in the image, the value is increasing by 1. We want our entire image to be limited to a range of 0 to 1, so we can divide these pixels by the overall width of the image, like so. Sampling the left-most pixel is still 0, whereas sampling the right-most pixel is just beneath 1. If we could sample one more pixel to the right, that would equal 1!

We can use this same logic on the Green channel. However instead of operating on the X axis, we're operating on the Y. Instead of dividing by the image's width, we'll be dividing by the height!

Now, let's create a checkerboard node, and an ST Map node to test our newly-created ST Map. A blank ST map should not change the image at all, but in this example you can see that our pixels are being shifted up and to the right by half a pixel! This is because UV coordinates are calculated at the bottom-left corner of a pixel in Nuke, as opposed to the centre. To offset this, we can simply add a value of 0.5 to each pixel in both X and Y. Now, as

we toggle the ST Map node on and off, there is no change, and we have a blank ST map that is ready to manipulate!

Each pixel in an ST Map represents a unique coordinate. As we modify our ST Map, like we're doing with this GridWarp and Splinewarp, we're relocating these coordinates to a new position. We can apply these pixel repositions to an image via our ST Map node. It is preferable to warp images via an ST Map for two reasons:

1. You gain more control over your warps, as you are able to dissolve back to the original, un-modified ST Map to essentially "mix" your warps, like I am demonstrating here! You can even Keymix, or dissolve between different ST Maps to gain more control, or blur an ST map to get a smoother result.
2. When applying a warp to your image through an ST Map node, you're only filtering your image once. In this example, the GridWarp is warping our image left-to-right, and the Splinewarp is warping our image bottom-to-top. If we copy/paste our two warps to a checkerboard, you can see that directly warping the image is causing some softening, as it's processing our pixels once for each node, and therefore filtering our pixels twice.

Perhaps the most practical example of this that I use in production is modifying a SmartVector-driven VectorDistort to stick an object onto an actor, or correct an actors performance. SmartVectors can be amazingly accurate, but on a shot with fast-moving detail in an actor's performance, for example, sometimes a VectorDistort can only get you 90% of the way to your desired result. Thankfully, the **VectorDistort** node lets us output results as an ST Map! Now, we can add a Gridwarp or Splinewarp to the result, to make some tweaks, before applying the warp to our footage via an ST Map node. This makes both me and Marcie happy!

We can also use ST Maps to speed up camera projections! In this example we have a simple, static environment with a camera move. We're all familiar with how to use the Project3D node to project an image onto geometry, although when working with dense geometry, it can take an age for the ScanlineRender node to crunch the data.

To speed up this process, we can instead feed an STMap into said projection, and precomp the result! This way, any time we need to project an image into our scene, we can simply use our speedy precomp with an STMap node, and get the same result much faster. Even

comparing this relatively small 3D Scene vs. the precomp, you can see the difference in speed!

Lastly, we're able to use ST Maps to help us generate some natural, organic motion in otherwise-static elements. Perhaps you have an object in your scene that is on fire, or smoking? Simply tracking a 2D smoke element to the object will look a bit janky, as 100% of the motion is being applied to 100% of the element. In the real-world, this is not how smoke behaves... The emitter would be tracked to the object, and the longer the smoke has been lingering, the less of an effect that transform would have.

In this setup, I have two gizmos that are generating random animation curves, and I am using them to randomly translate my ST map in X and Y. Then, I am dividing my ST map into 5 parts using these rectangles as mattes in a series of Keymixes. In each Keymix, I am time-offsetting the Transformed-ST Map back in time, so that as the smoke rises to the top of frame, the animation incrementally lags behind the Transform's animation. Additionally, I'm incrementally dissolving the Transformed ST map back to an un-modified ST map so the further the smoke rises, the less it will be affected by the transform overall.

We can obviously see some matte lines where each of these keymixes are intersecting though. The beauty of using ST Maps means we can blur the end result to average the pixels in our ST Map, which not only smoothes out the matte lines, but blend the differences in animation together for a much smoother transition!

It's a total cheat, but can be a very effective technique to add to your compositing toolkit!

Conclusion

That's it from me! I hope I have helped you gain a deeper technical understanding of the nodes you use every day in Nuke, and how they actually manipulate the pixels in an image. Additionally, I hope you have gained an appreciation for how understanding these concepts will improve the quality of your Compositing, and ultimately the final output of your shots!

If you would like to learn more about improving your workflow in Nuke, you can find years worth of content on my Nuke-focused blog, as well as looking through the archives of Ben's Comp Newsletter.

Please find both by visiting my website: www.benmcewan.com.

I hope you enjoyed the course!