Computational Photography: HDR Compositing and Basic Post-Processing

Scott Rubey

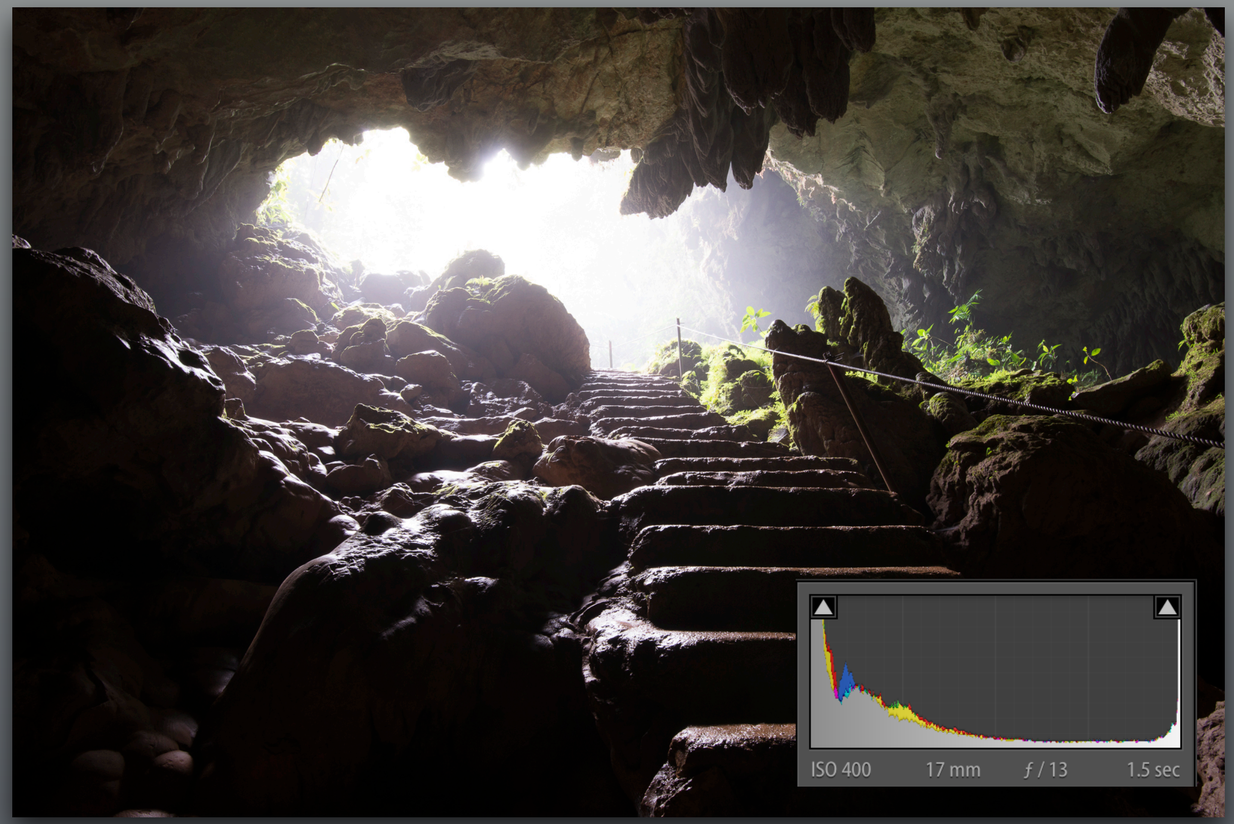
Goals

- To implement an Adobe Lightroom-like interface that could combine *N* images into an HDR composite, allowing the user to easily control tone mapping parameters and make aesthetic adjustments to the resulting composite image
- To explore a broad number of OpenCV features
- To learn more about what certain imaging applications might look like "under the hood"

Why HDR?

The Problem...

- Modern digital sensors have about a 14 f-stop dynamic range
- This can't quite capture the dynamic range present in some scenes
- Attempting to lighten shadows or darken highlights can lead to unnatural looking photos, higher noise, digital artifacts



The Solution...

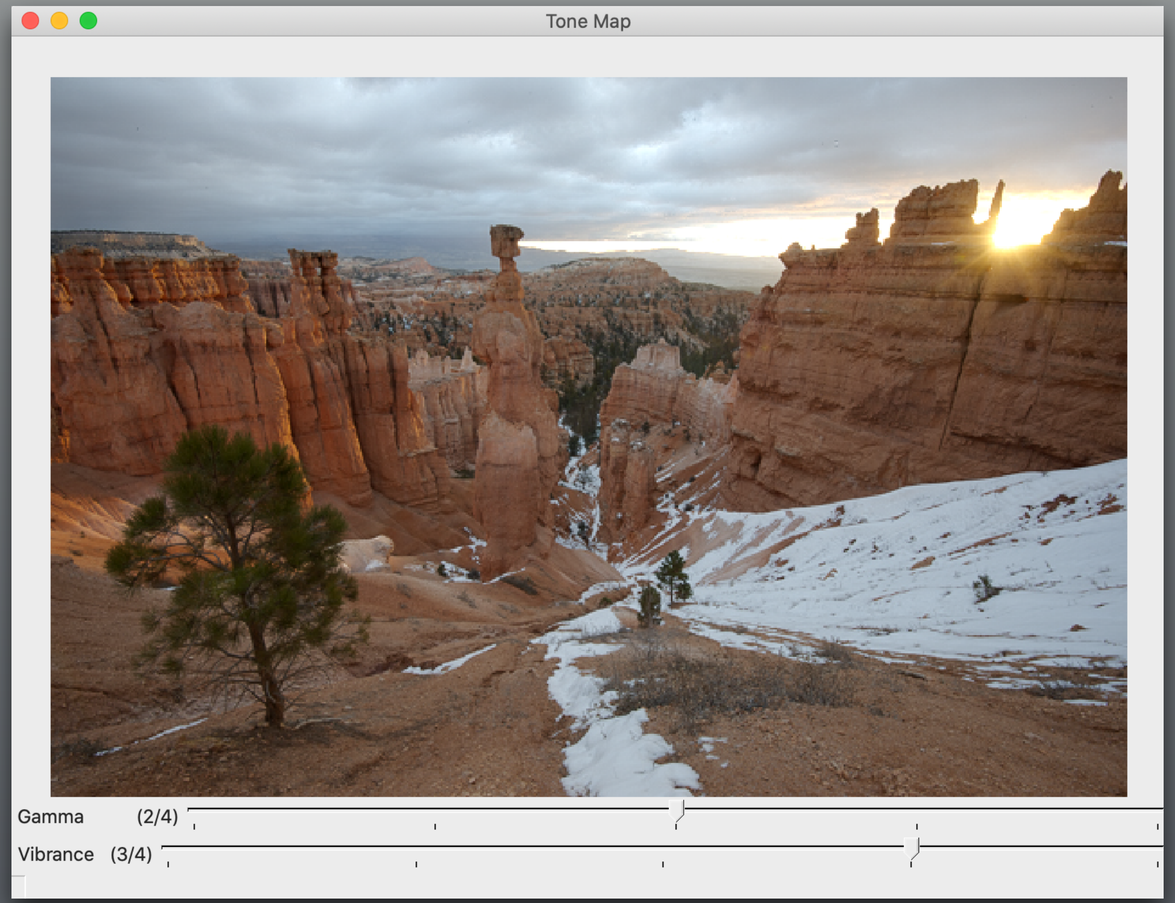
- Take multiple photographs that encompass the full dynamic range (usually 3 or more)
- Generate a response curve:
    - Analyze each photo on a pixel-by-pixel basis
    - Plot values across the entire dynamic range
- Generate a composite LDR image
    - For the set of input images, a pixel is added to the output image (aka radiance map) if it is closest to the middle of the response curve

The Process

Source Images

Tone Mapping

- Tone mapping: maps values to 8-bits to approximate what an HDR scene might look like by way of an LDR medium



Gamma = exposure

Gamma = 1

Gamma = 4

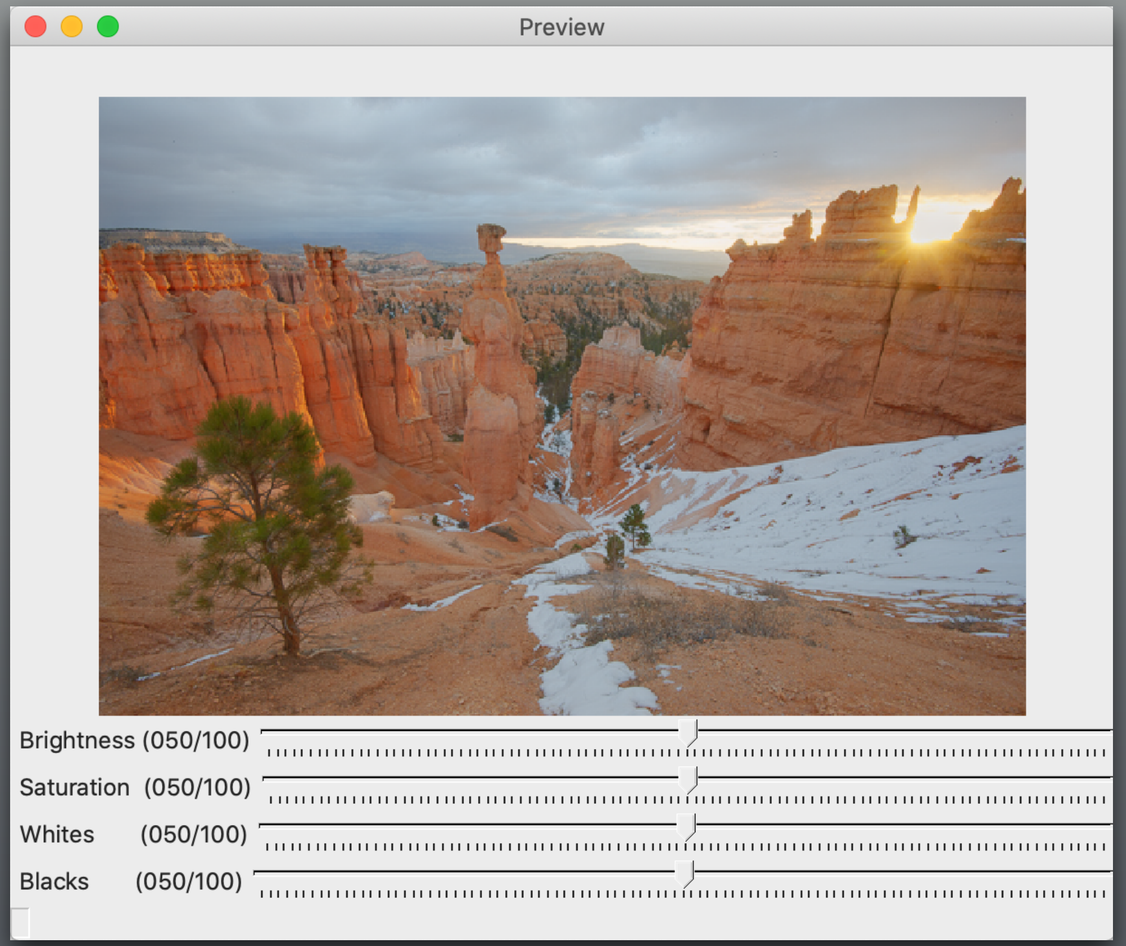
Vibrance

Vibrance = 0

Vibrance = 4

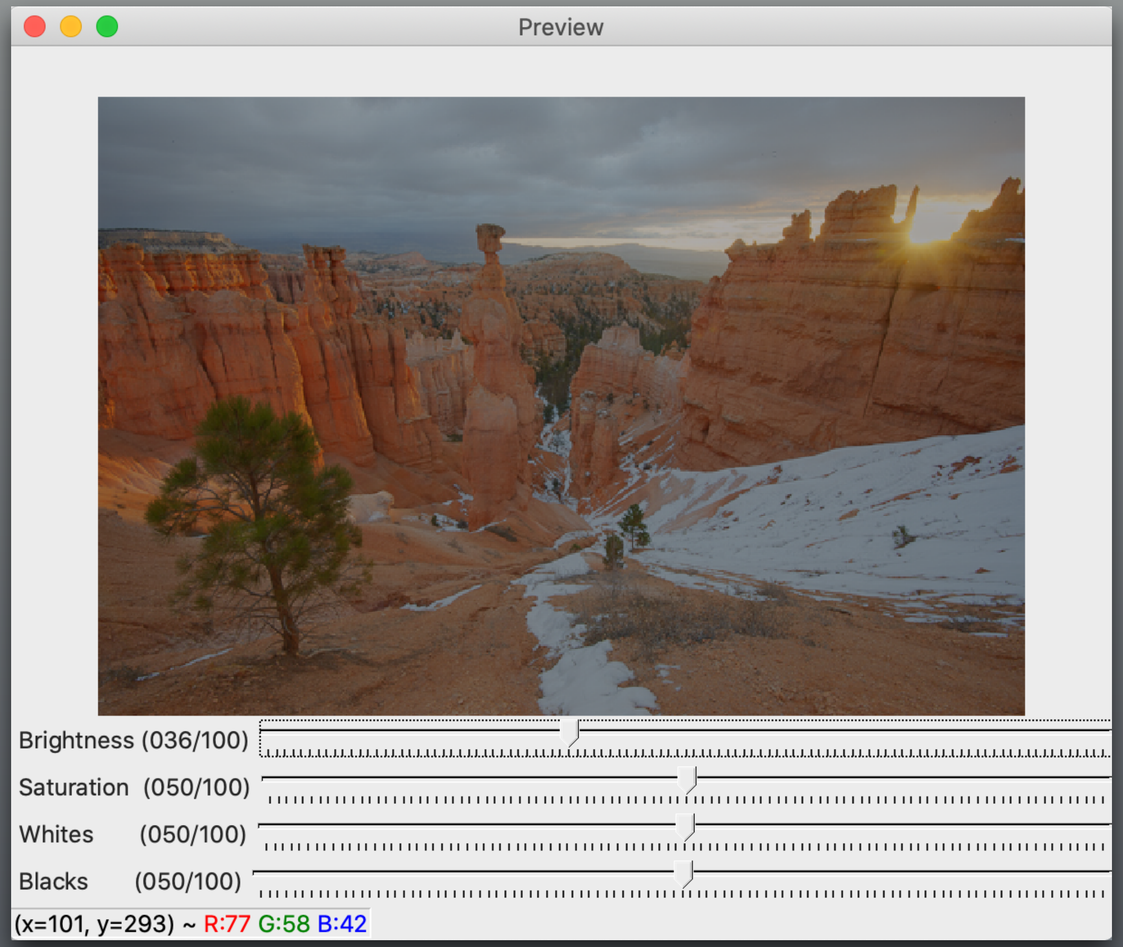
Post-processing

- Implemented brightness, color saturation, and contrast controls; bound them to sliders for easy editing



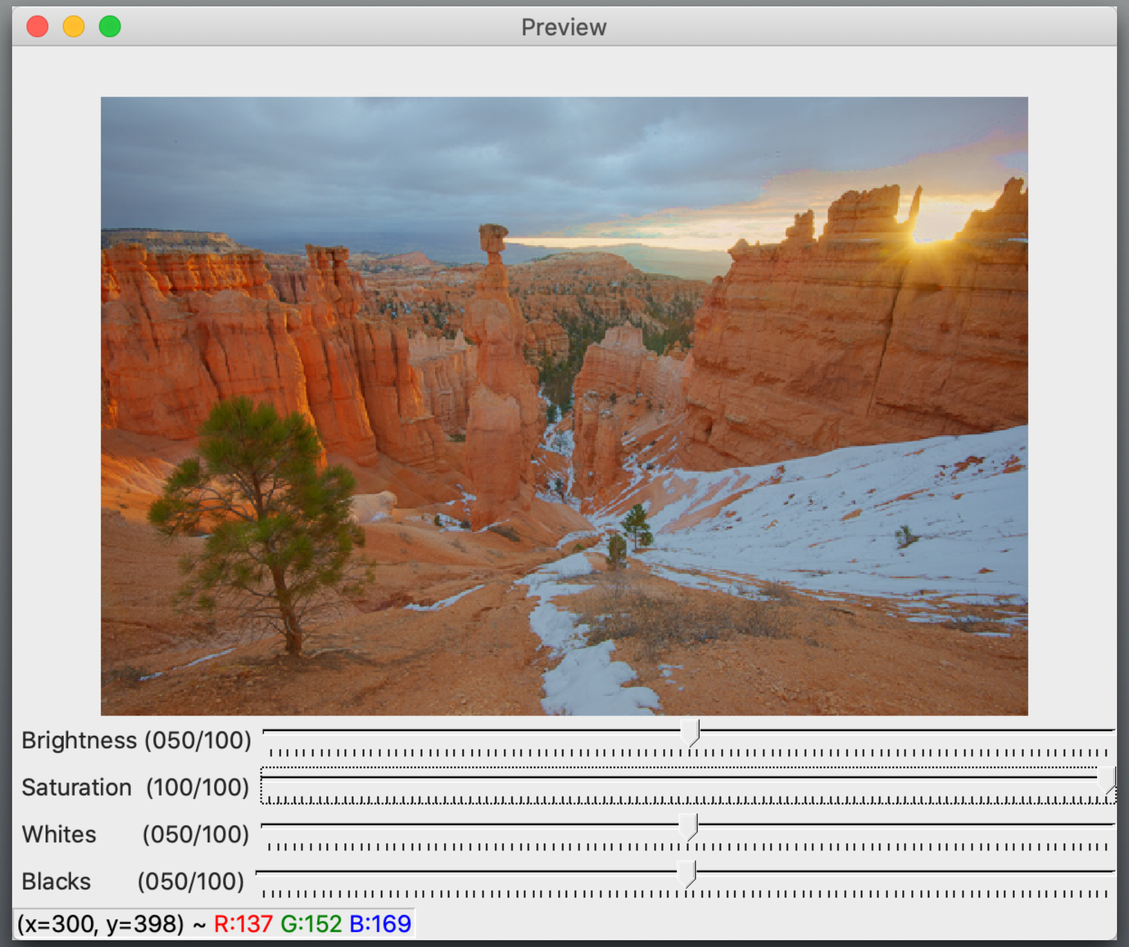
Brightness

- cv2.convertScaleAbs(src, dest, alpha, beta) calculates the absolute value of each element in an array/matrix.
- By capturing the slider value and adjusting 'alpha', one can alter the brightness of an image
- This adjusts all pixels equally, which isn't necessarily the most aesthetically pleasing tool...



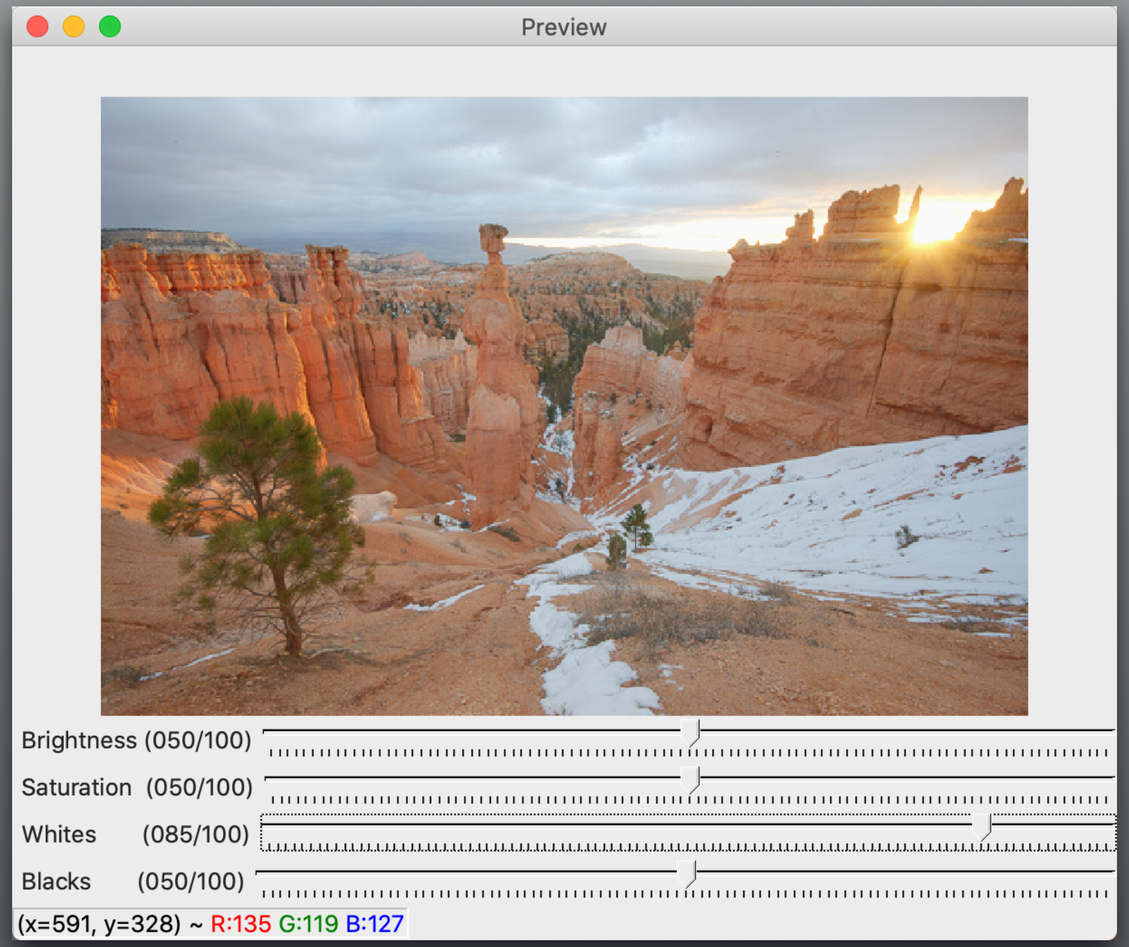
Saturation

- Convert color space to HSV using OpenCV
- Capture the value of the Saturation slider and apply it to the S-channel
- Convert color space back to BGR
- Also uniformly adjusts each pixel value, which can lead to artifacts when colors go out of gamut



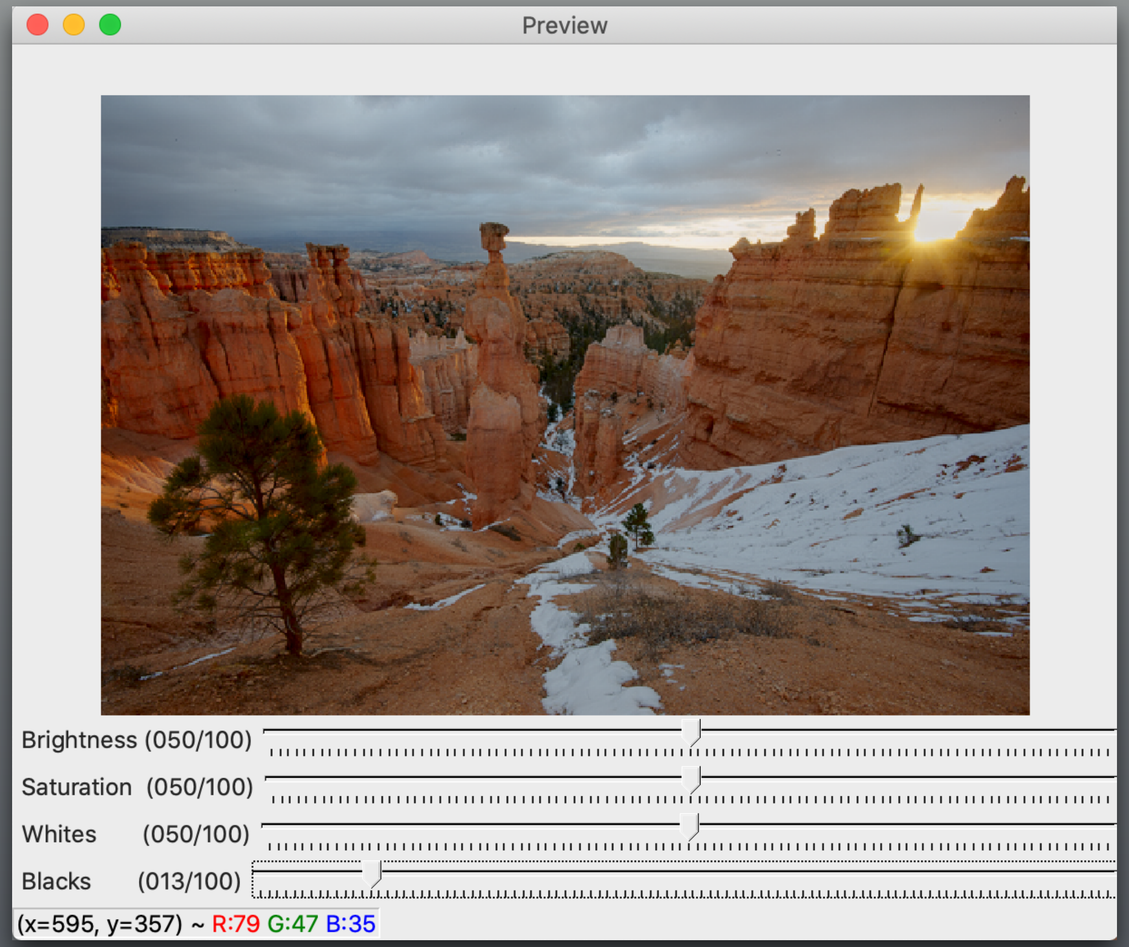
Contrast - Whites

- Convert to L*a*b color space and split the channels
- Capture the highest L pixel value in a temp variable (Lmax = L.max())
- Add/subtract slider value to/from Lmax
- Normalize histogram between L.min() and Lmax
- Convert back to BGR
- This gives the user more control over brighter pixels while preserving the darker ones



Contrast - Blacks

- Same process as Whites, but flipped
- This gives the user control over darker pixel values while preserving the lighter ones
- Whites/Blacks sliders are a way of fine-tuning contrast; 'scalpel' approach rather than 'sledgehammer'



The Result

Digital artifacts, but not unexpected

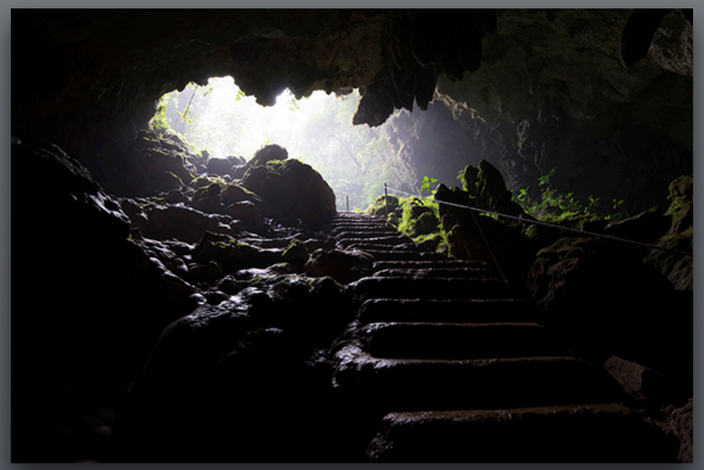
Would be nice to have some local control

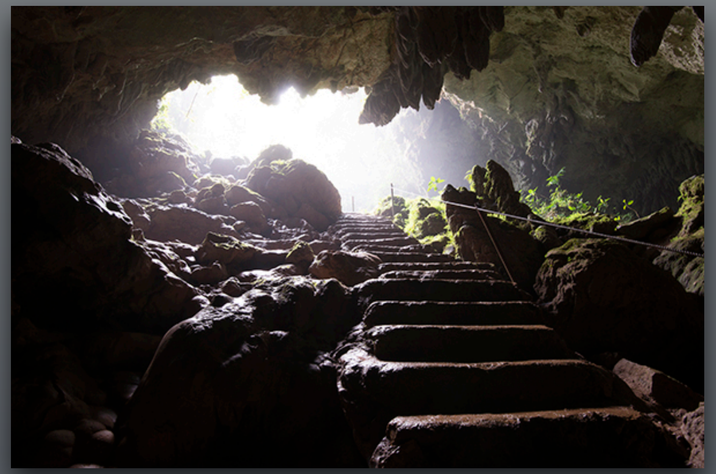
Let's see it in action...

Other Test Images

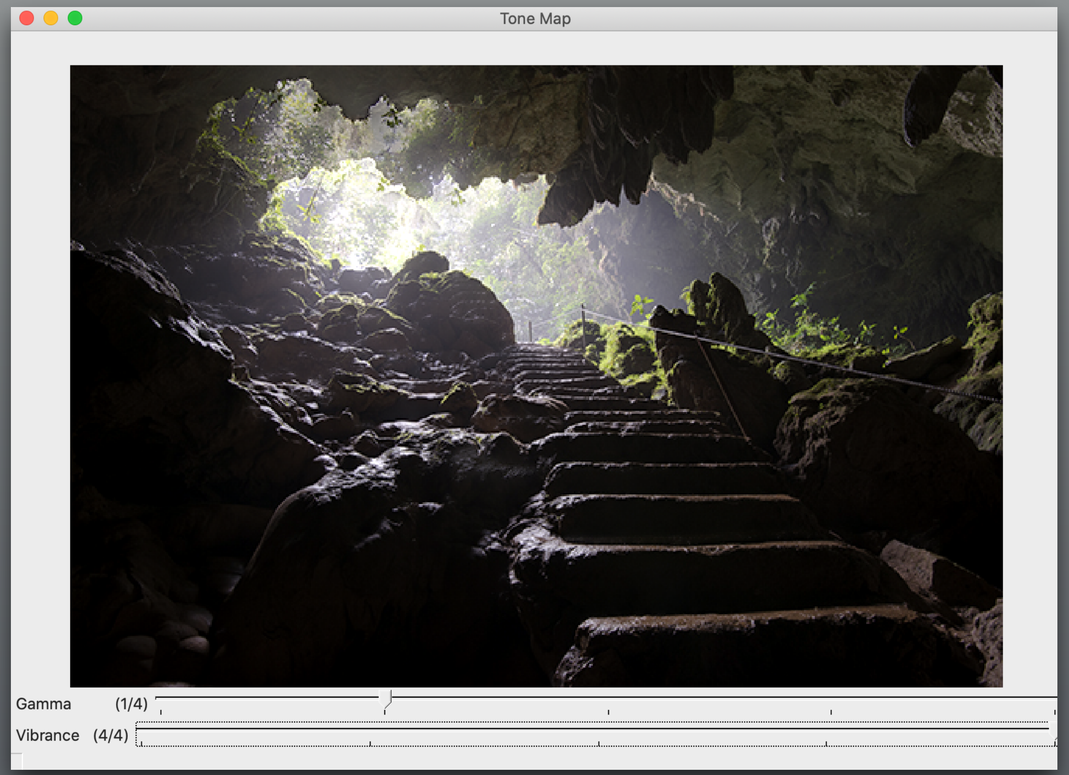
Source Images



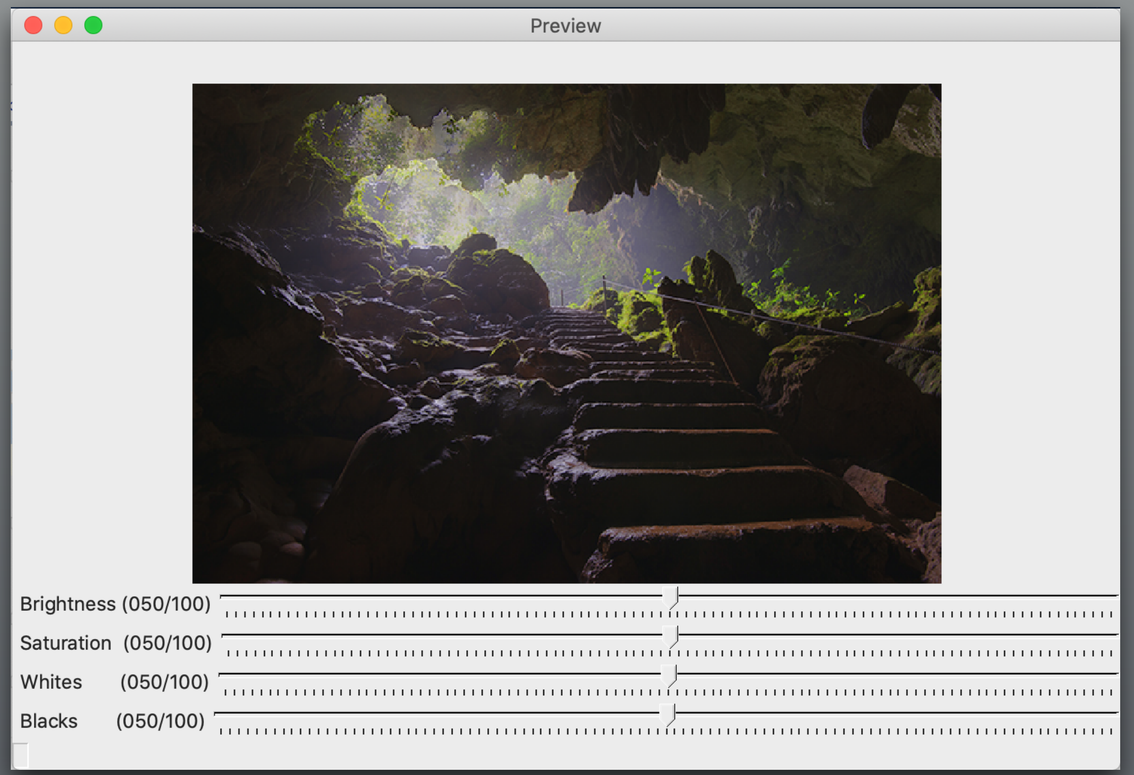





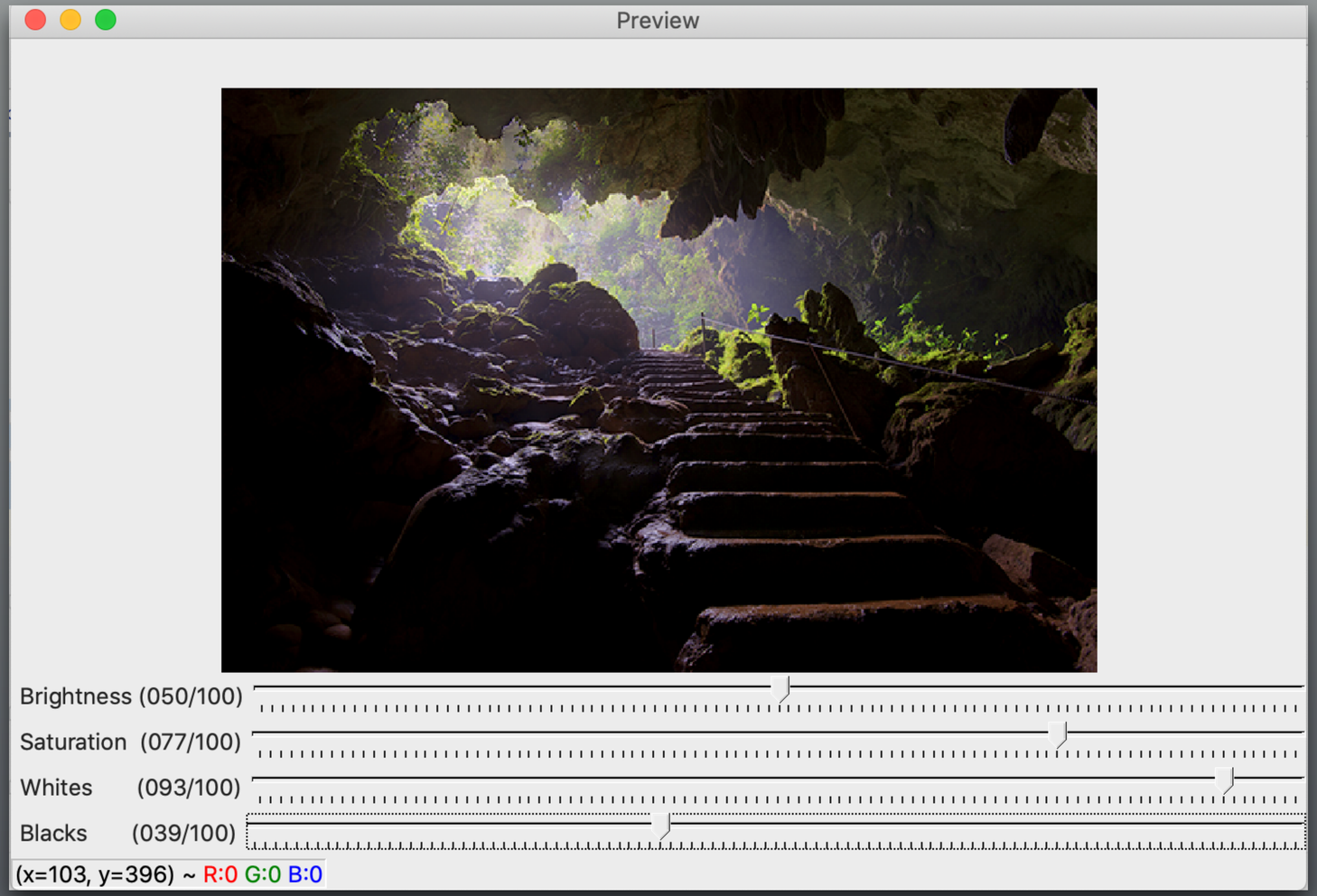
Merged Image



Tone mapped LDR

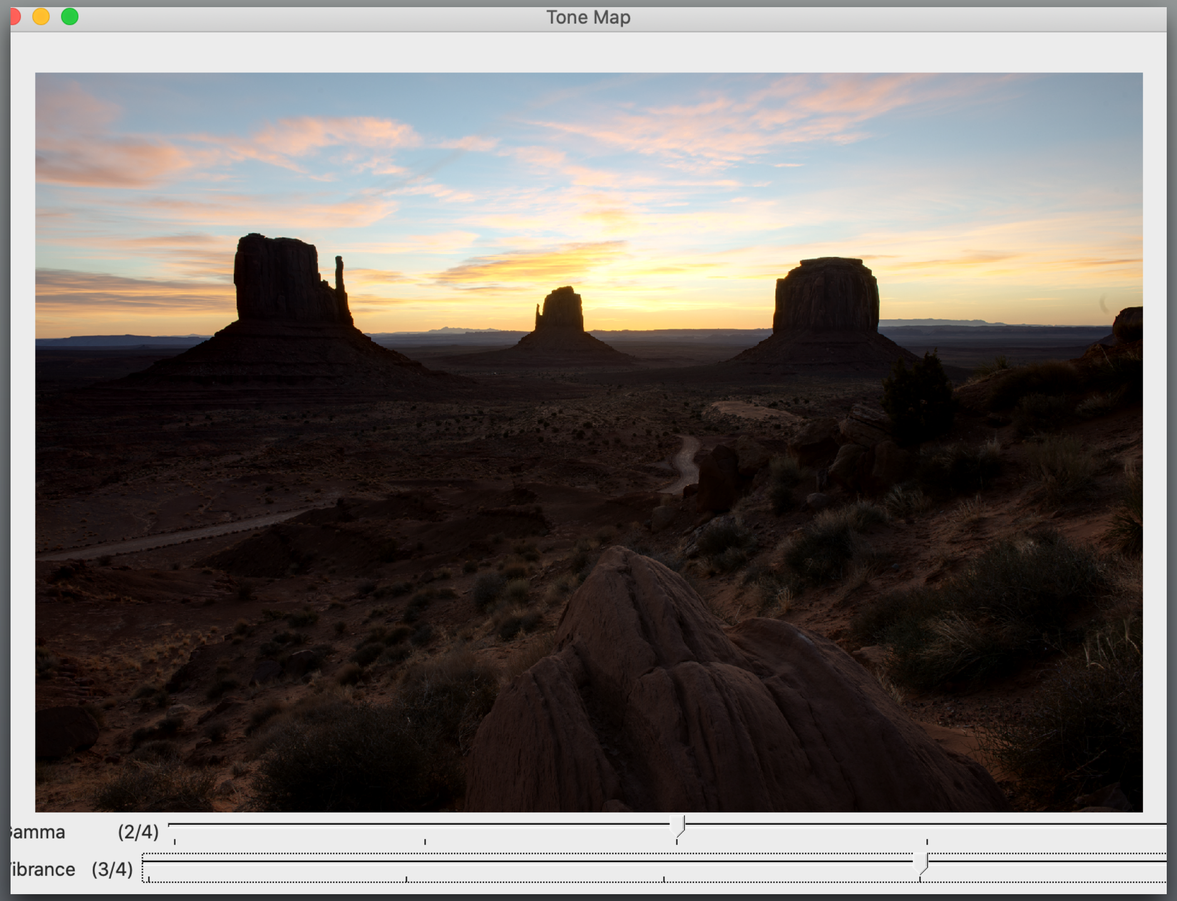


Final Edit

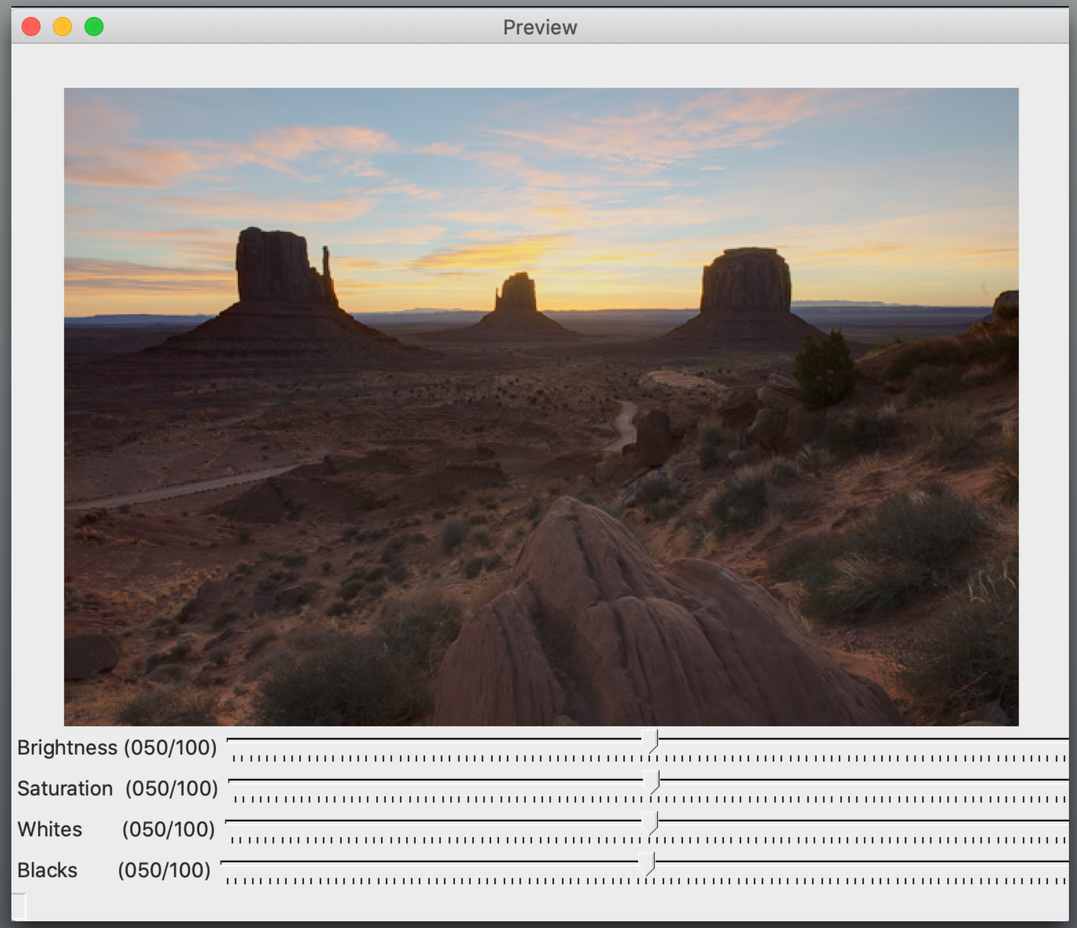

Preview
Brightness (050/100)
Saturation (077/100)
Whites (093/100)
Blacks (039/100)
(x=103, y=396) ~ R:0 G:0 B:0

Source Images

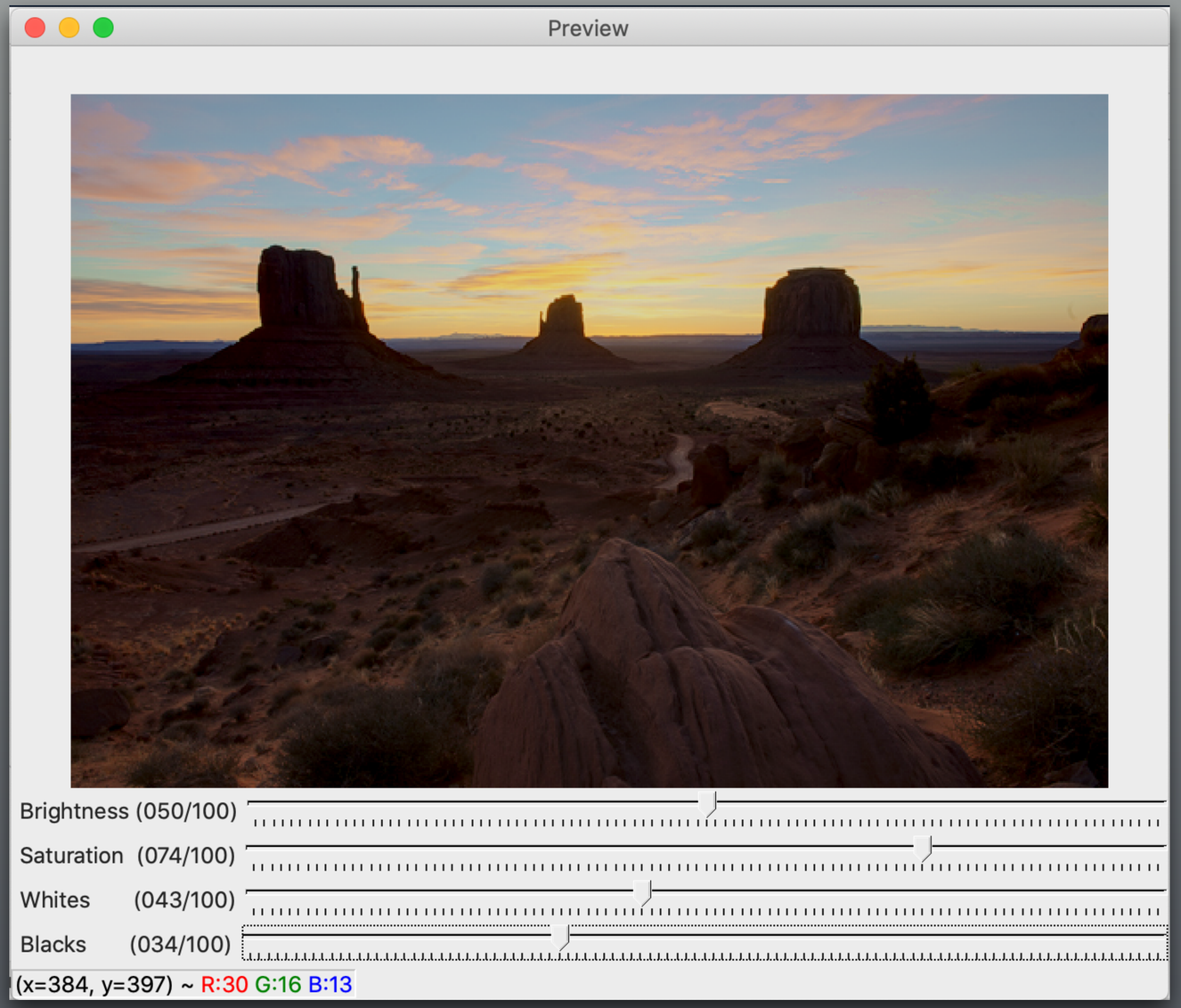
Merged Image



Tone mapped LDR



Final Edit


Preview
Brightness (050/100)
Saturation (074/100)
Whites (043/100)
Blacks (034/100)
(x=384, y=397) ~ R:30 G:16 B:13

Conclusions

- More source images = more control over the final product
- With some images, this works great; with others, maybe not so much
- Global editing controls can go a long way, but fine-tuning with targeted adjustments can prevent artifacts and lead to a more aesthetically pleasing image
- Program for performance, look for optimizations