Intermediate Graphics & Animation Programming

GPR-300
Daniel S. Buckstein

Bloom & High Dynamic Range Rendering
Week 5

License

HDRR & Bloom

- High Dynamic Range (HDR) photography
- High Dynamic Range Rendering (HDRR)
- HDR Bloom
  - Box blur
  - Bright pass
  - Gaussian blur
  - Compositing
- Optimizations and improvements

HDR Photography

- HDR: ***High Dynamic Range Imaging***
- Uses the principle of "*tone mapping*"
- Combining several ranges of colour into one

Daniel S. Buckstein

https://en.wikipedia.org/wiki/Tone_mapping


5
138
331
62


Eaton Centre
AQUOS LED
Any travel on Points.
First Class
VISA
Now that's First Class.
TD Canada Trust
milestones
WOO
SHOPPERS DRUG MART
EXTREME FITNESS
What did you get?

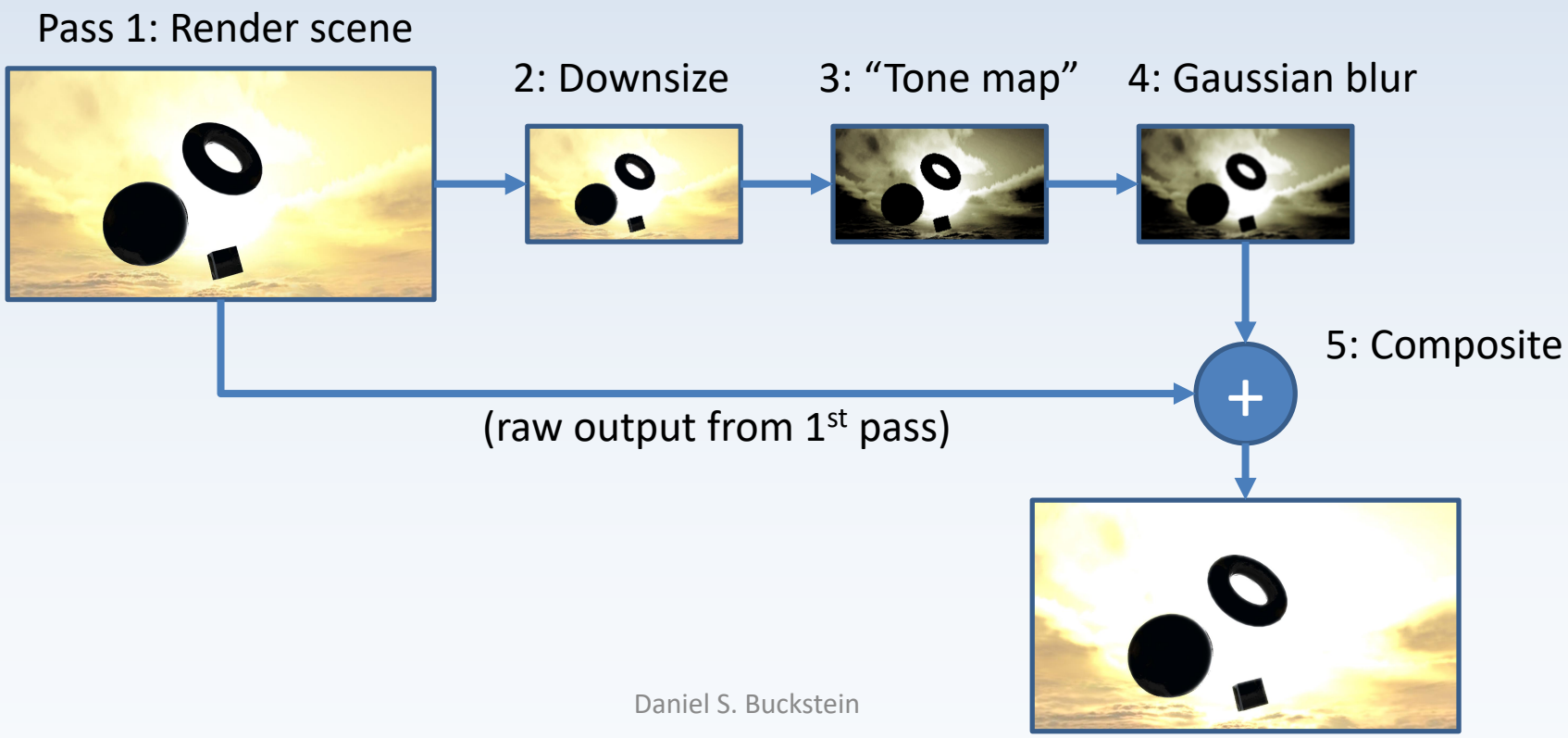
https://en.wikipedia.org/wiki/High-dynamic-range_imaging

HDRR: Bloom

- This process can be *simulated* in real-time rendering: ***High Dynamic Range Rendering***
- Multi-pass post-processing algorithm called "***Bloom***"
- "***The 4 B's***": a mnemonic by yours truly
  - Box
  - Bright
  - Blur
  - Blend

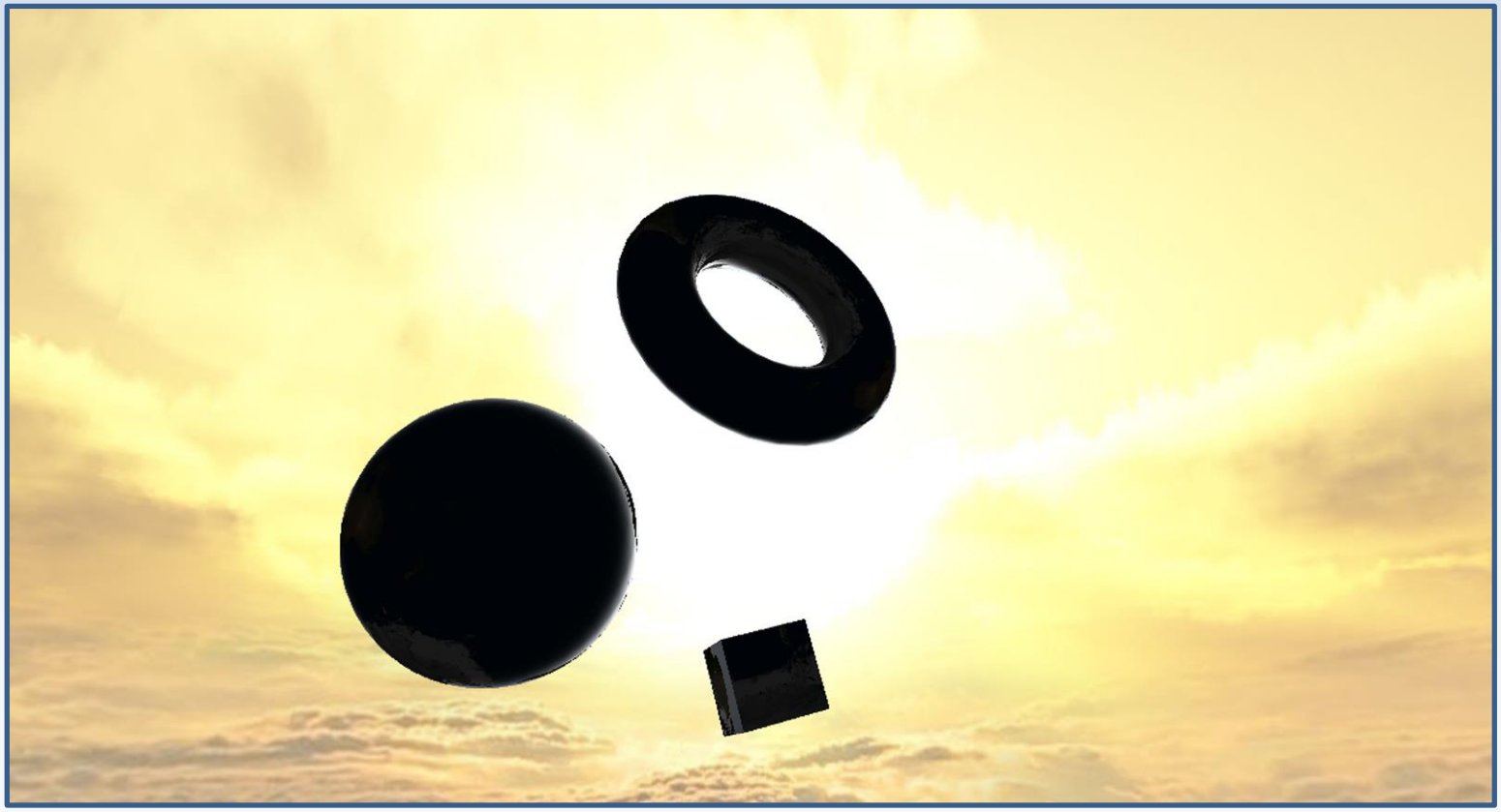
HDRR: Bloom

- ***Bloom*** shader network diagram:
  - Core algorithm



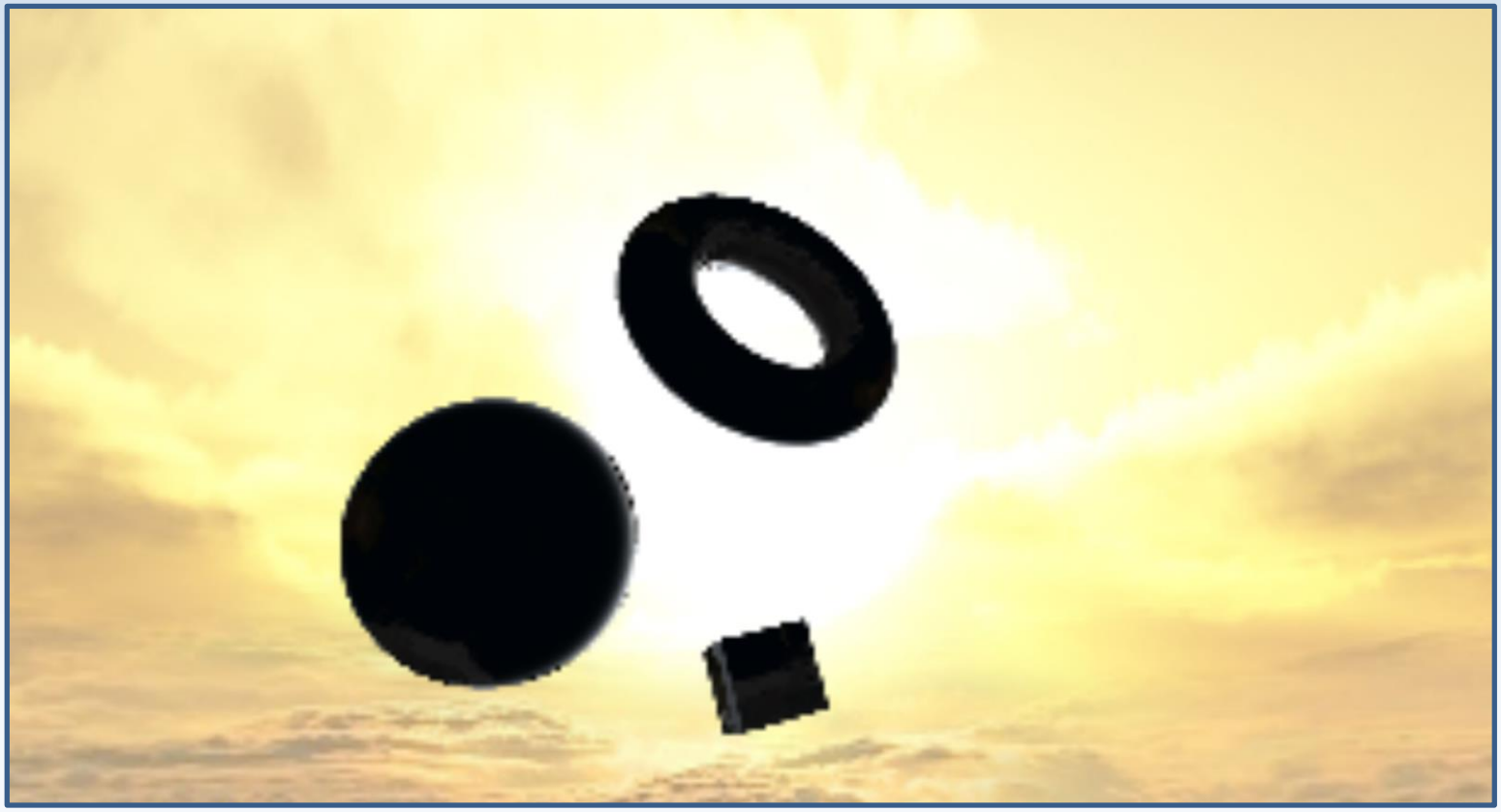
HDRR: Bloom

- 1st pass: Render the scene



HDRR: Bloom

- 2nd pass: Downsize… Free *box blur*, speed



HDRR: Bloom

- 3rd pass: Simulated tone mapping (bright pass)

HDRR: Bloom

- 3rd pass: Simulated tone mapping (bright pass)
- Complex: curve editor like Photoshop
- Simple: use your knowledge of basic algos:
- Good example: Which function do we know that *compresses* a range???
- …so what function would *expand* a range???

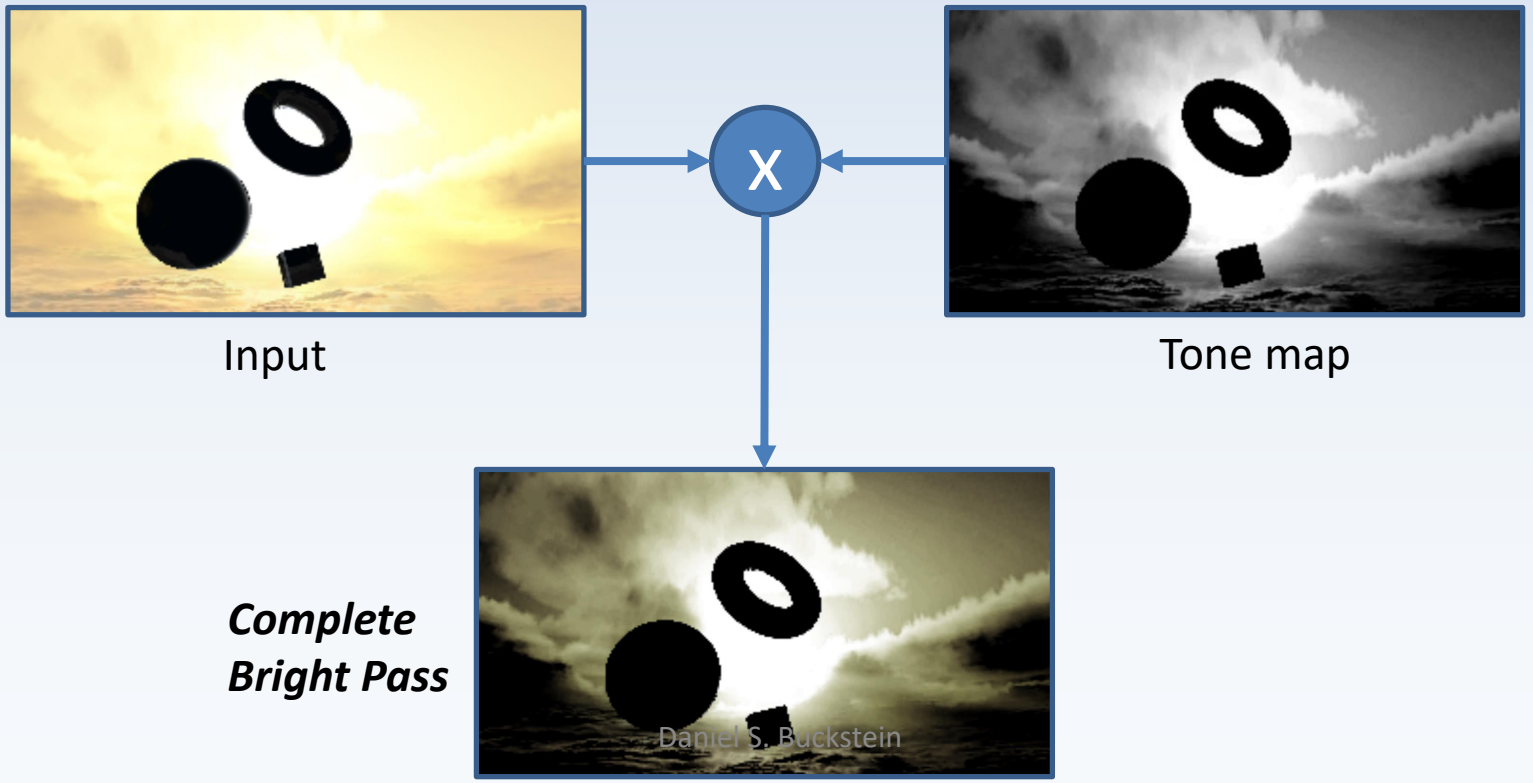
HDRR: Bloom

- 3rd pass: Simulated tone mapping (bright pass)
- Bright values are kept, dark values are tossed
- Calculate the *luminance* of each pixel and
- Run your bright pass function (e.g. deserialize)
    - The result of this step is called the ***tone map***

Input image | Luminance (L) | Bright = Deserialize(Deserialize(L))

HDRR: Bloom

- 3rd pass: Simulated tone mapping (bright pass)
- Multiply your ***tone map*** by the ***input image***



HDRR: Bloom

- 4th pass: Gaussian blurring

HDRR: Bloom

- 4th pass: Gaussian blurring
- The goal: bright areas ***flood over*** dark areas in the scene
- This is achieved using ***blurring***
- Blurring uses a ***convolution kernel***

HDRR: Bloom

- 4^{th} pass: Gaussian blurring
- 5x5 Gaussian blur kernel (normalized):

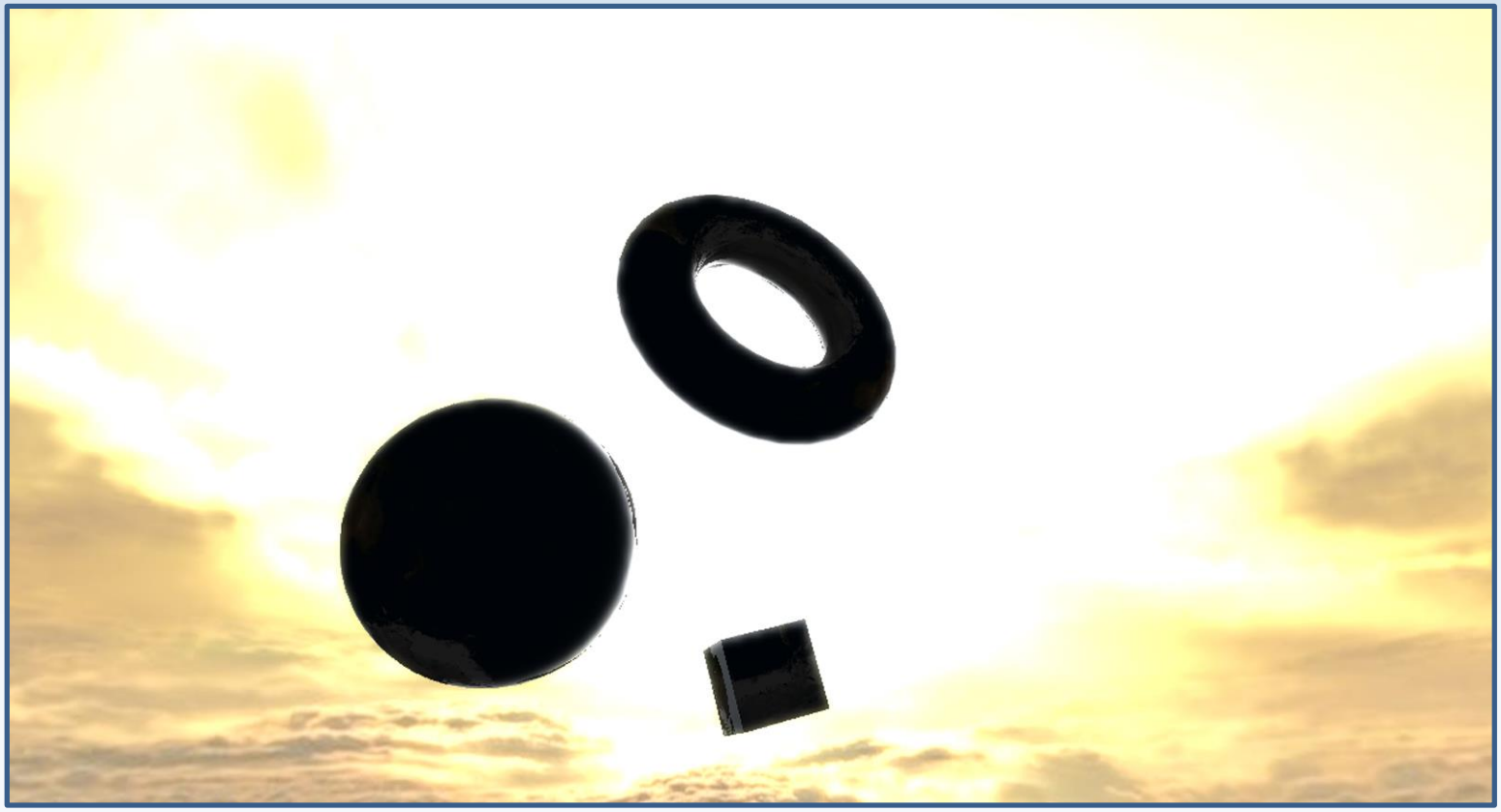
$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

HDRR: Bloom

- 4th pass: Gaussian blurring
- Major problems with this???
- Nested for loops >__<
- …or 25 manual texture samples per pixel
- 7x7 kernel: 49 samples
- 9x9 kernel: 81 samples
- …terrible

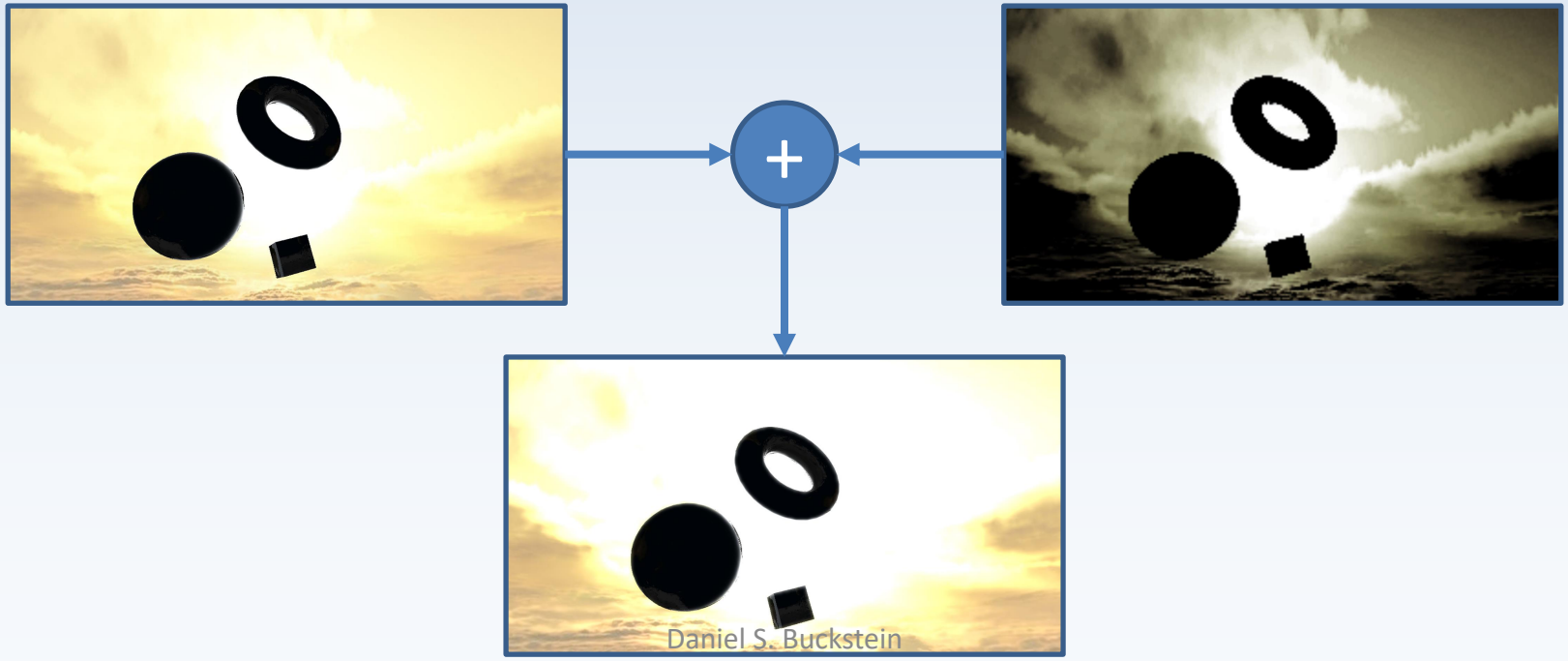
HDRR: Bloom

- 5th pass: Composite → “Bloom”



HDRR: Bloom

- 5th pass: Composite
- Fastest way to composite: ***additive***
- Original scene render + Gaussian pass



HDRR: Bloom

- For all passes:

```
// Demo.cpp: single pass draw routine
    ...
    fboList[currentPass].Activate();
    fboList[currentPass - 1].BindColour();
    programList[currentPass].Activate();
    DrawFSQ();
    ...
```

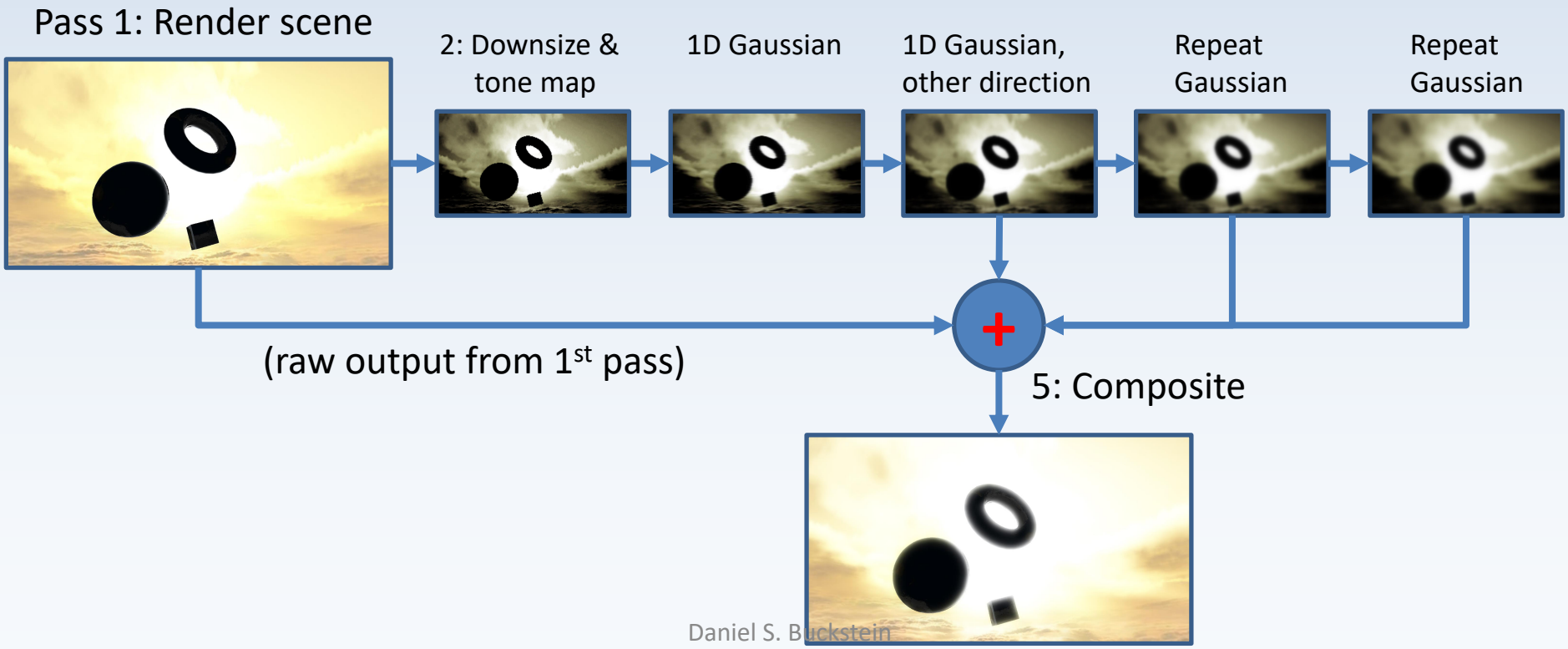
HDRR: Bloom

- For all passes:

```
// overall fragment shader structure:
    ...
varying vec2 texcoord;
uniform sampler2D inputTex;
    ...
    vec4 pix = texture2D(inputTex, texcoord);
    gl_FragColour = processPixel(pix);
```

Bloom Optimizations

- Optimizations and improvements… lots ☺
- Improved bloom shader network diagram:



Bloom Optimizations

- First optimization (dead simple):
- Bright pass and downscale happen at the same time
- The result is identical to doing them separately
- Immediately cuts an entire pass.
- The end.

Bloom Optimizations

- Second optimization: separate Gaussian blur into two 1D passes
- 5x5, 2D Gaussian blur convolution kernel:
- ..actually the product of ***two 1D kernels***:

$$G_{5\times5} = \frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

$$G_{5\times5} = G_{5\times1} * G_{1\times5}$$

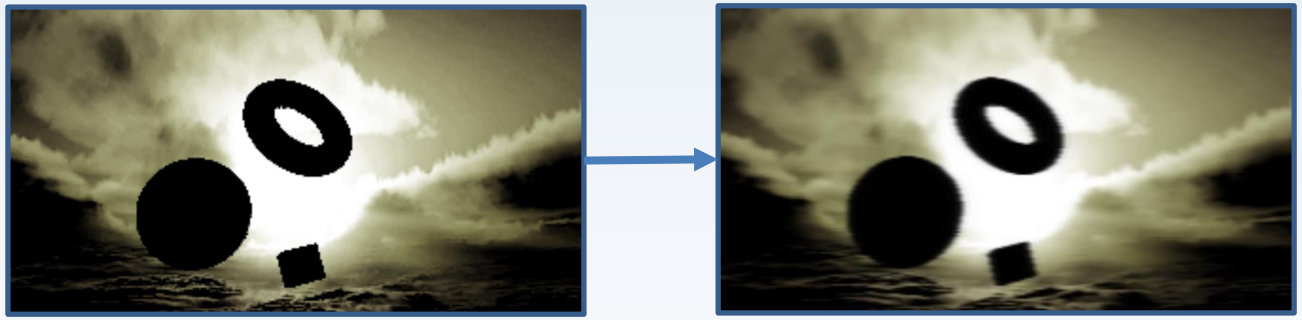
1D CONVOLUTION KERNELS!

$$= \frac{1}{16}\begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} * \frac{1}{16}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Bloom Optimizations

- Second optimization: separate Gaussian blur into two 1D passes
- Part 1: Horizontal Gaussian blur:
- ***Input*** is our tone mapped (bright-only) image
- ***Output*** is a horizontally-blurred image:

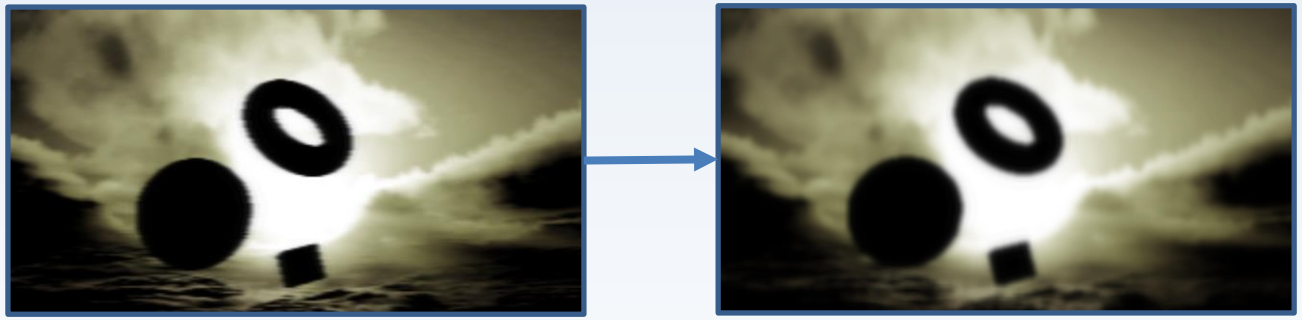
$$\frac{1}{16}[1 \quad 4 \quad 6 \quad 4 \quad 1] \ *$$



Bloom Optimizations

- Second optimization: separate Gaussian blur into two 1D passes
- Part 2: Vertical Gaussian blur:
- ***Input*** is our ***horizontal blur output!!!***
- ***Output*** is a *fully-blurred* image:

$$\frac{1}{16}\begin{bmatrix}1\\4\\6\\4\\1\end{bmatrix} * \quad$$



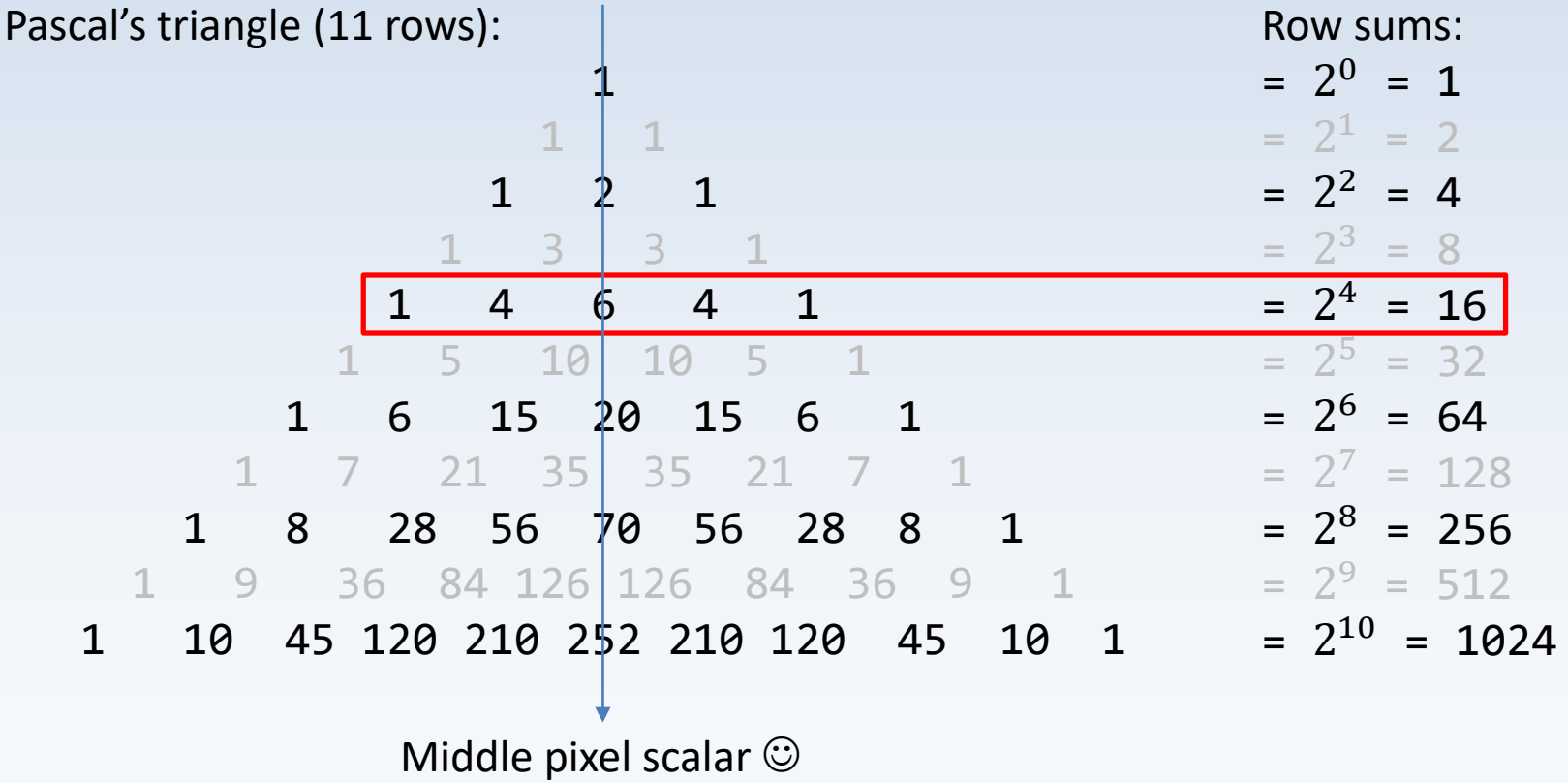
Bloom Optimizations

- Super pro tip: Where did these kernel values come from???

$$G_{5\times5} = \frac{1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & 36 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix} = \frac{1}{16}\begin{bmatrix}1\\4\\6\\4\\1\end{bmatrix} * \frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix}$$

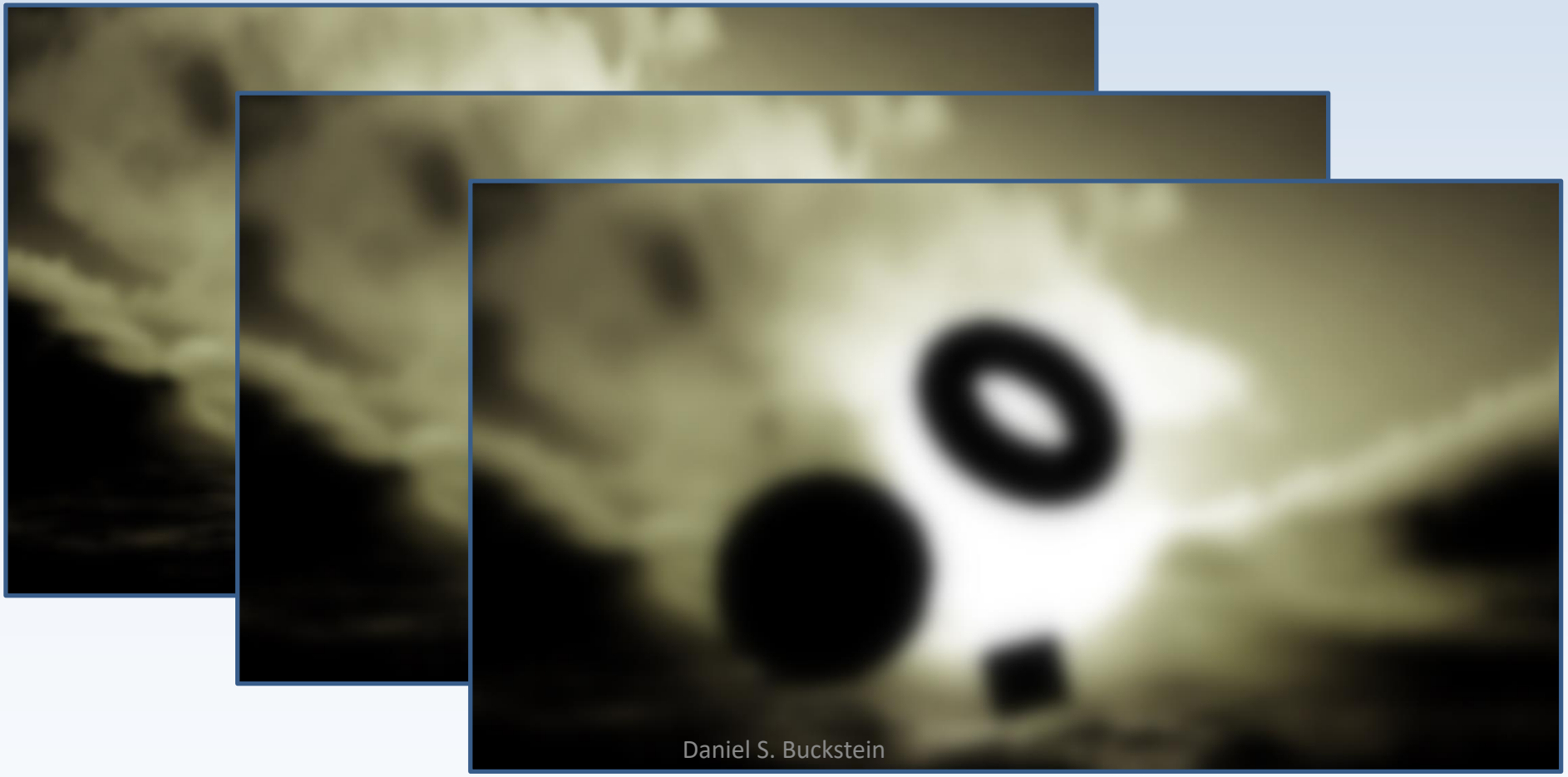
Bloom Optimizations

- …the *even-power* rows in ***Pascal's Triangle!***



Bloom Optimizations

- Third optimization: Repeat Gaussian blurring!



Bloom Optimizations

- Fourth optimization: Use a better compositing function ☺
- Additive is okay because it's easy…
- …but it can get too bright
- What other options does Photoshop offer?
- *Multiply*: product of images… problem?

Bloom Optimizations

- ***Screen*** function for compositing:
- We want to do what multiply does, but for brightness instead of darkness
- Solution???
- Invert inputs, multiply, invert result!

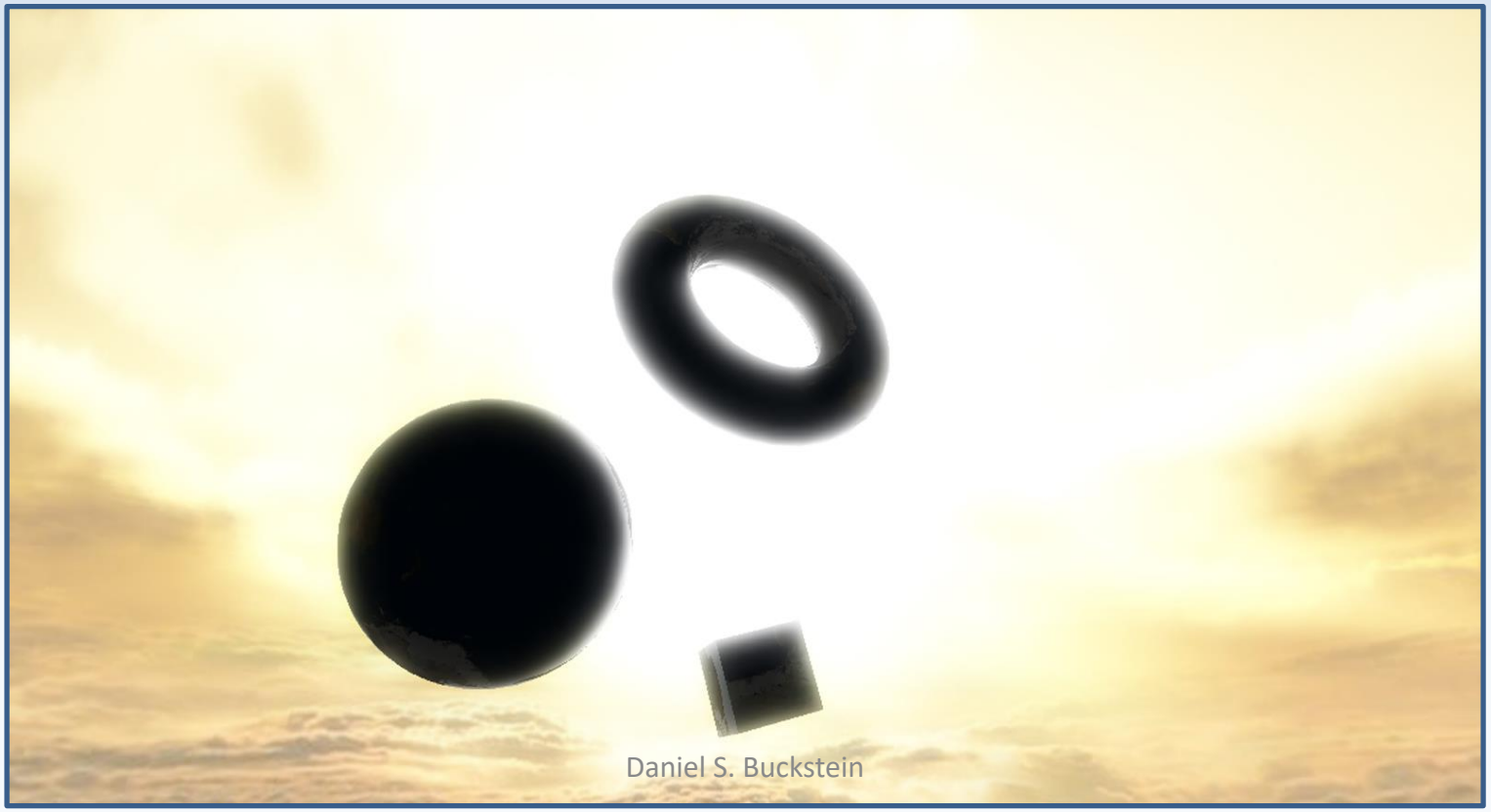
$$\text{screen}(A, B) = 1 - (1 - A)(1 - B)$$

- Same principle for more than 2 inputs:

$$\text{screen}(A, B, C, D) = 1 - (1 - A)(1 - B)(1 - C)(1 - D)$$

Bloom Optimizations

- Fifth optimization: For better compositing, blend original scene with ***all*** blur pass results!



Bloom Optimizations

- Final optimization: True HDR requires a *dynamic* range of color, as opposed to a *fixed* range
- Problem with standard FBOs???

  0-1 color range
- Solution: *float* buffers can store values beyond 0-1 range!
- Use this information however you please ☺

The end.

- Questions? Comments? Concerns?