Kyle Cutler
Dara Dimoff
Final Project Report

1. Introduction

The purpose of this project was to attempt to simulate rainbow refraction in the ray tracer. After a ray enters a specific medium, it should refract and give an effect similar to what is shown in Figure 1.

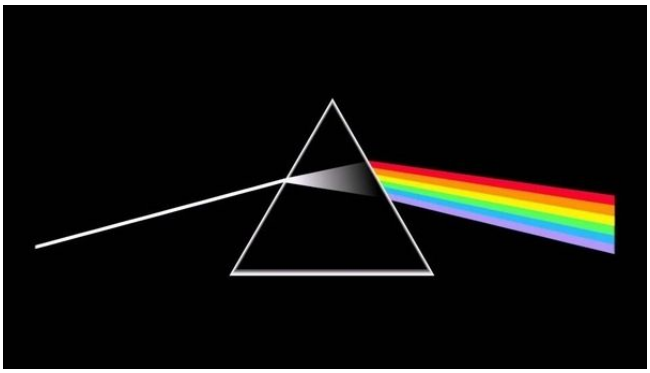


Figure 1: Rainbow refraction through a prism

For the rainbow refraction, we would like the light to diffuse in multiple directions after exiting the medium. This is accomplished using photon mapping. Because we're looking for a rainbow-like effect, treating the photon mapping in terms of RGB doesn't make sense. Instead, we'll be treating rays of light as wavelengths instead to allow simpler construction of the rainbow effect.

2. Design

This project mostly extends on the photon mapping advanced checkpoint. When light passes through a transmissive medium, some of it gets filtered out, and in the ray tracer, each photon has a probability of passing through the medium or reflecting.

The tricky part is that if it does transmit, it should not only refract, but each photon refracts differently based on its wavelength because of a property known as dispersion. This is how the rainbow effect is achieved; photons of longer wavelengths will have a lower index of refraction (and thus bend less), and photons of shorter wavelengths will have a higher index of refraction. This is achieved by adjusting the refractive index based on the wavelength of the photon. To get into some details: the primary index of refraction is based on a principal wavelength of 600nm, and the index of refraction is adjusted linearly around this principal point. The slope of this adjustment determines whether the resulting rainbow spectrum is wide or narrow.

In standard photon mapping, each photon is spawned with the color of the light source. But this is problematic – if every photon has the same initial color, we cannot determine which colors were refracted in different directions. To solve this issue, we decided to use a Monte Carlo method and spawn each photon with a random color (more on this below).

In order to adjust the index of refraction based on wavelength, we had to determine the wavelength of each photon cast. Originally, this was achieved by converting each photon's RGB value into wavelengths and calculating the refractive index from the derived wavelength. To do so, we converted the RGB values into HSV color space and then used the hue value to approximate the wavelength. This gives reasonable results, but we were not completely satisfied because it did not accurately resemble the amount of each color you would expect to see in the visible spectrum. To be more specific, the regions of red were very small, and some of the red even started to "bleed" into the purple region, and the cyan and yellow regions were unreasonably large. For the final product, this was modified so that instead of spawning photons of different RGB values, photons are spawned with a specific wavelength, and in order to determine the color at each point from the photon map, wavelengths are first converted back into RGB values and then averaged. The resulting image depicted the distribution of the visible spectrum much more accurately.

Another parameter we found we the need to tweak was the distribution of color / wavelengths of the spawned photons. When we were using RGB values for the photons, two methods were tried. First, a uniform distribution: each color between (0, 0, 0) and (512, 512, 512) had an equal chance of being spawned (the values went up to 512 in order for the average color to be approximately (255, 255, 255)). This gave some good-looking results that can be seen below. But we weren't satisfied that photons could have near-white color. So our second attempt was to spawn photons having a random hue in HSV space (and 100% saturation and value), in an attempt to simulate a random wavelength. This also gave reasonable results, but colors seemed to be far too saturated, so the image looked a bit unrealistic. After we switched to wavelength-based photons, the distribution was now only one-dimensional. At first, we simply spawned wavelengths uniformly in the visible range. This gave some nice results initially, but the color of the light source appeared rather yellow. To combat this, we tried some other physically-based distributions, including one based on Wien's approximation, which approximately follows the distribution of wavelengths for a blackbody. This gave some very nice, tweakable results that look more like natural sunlight.

3. System

This project was an extension of the ray tracer with the KD-tree and photon mapping advanced checkpoints. The final architecture looked very similar to what was in the ray tracer at this point in the semester, and most modifications were not to the architecture, but to the functions that calculated the results of the photon mapping – the ray tracing code and object materials were the biggest pieces that changed, along with the addition of code to implement the photon mapping itself including the kd-tree for storing photons and its associated algorithms.

The program is designed to be fairly efficient, so that we could simulate many thousands of photons efficiently. For example, the program uses multiple threads for rendering, photons are stored in a kd-tree, and the nearest-neighbor search makes use of a max heap to maximize efficiency.

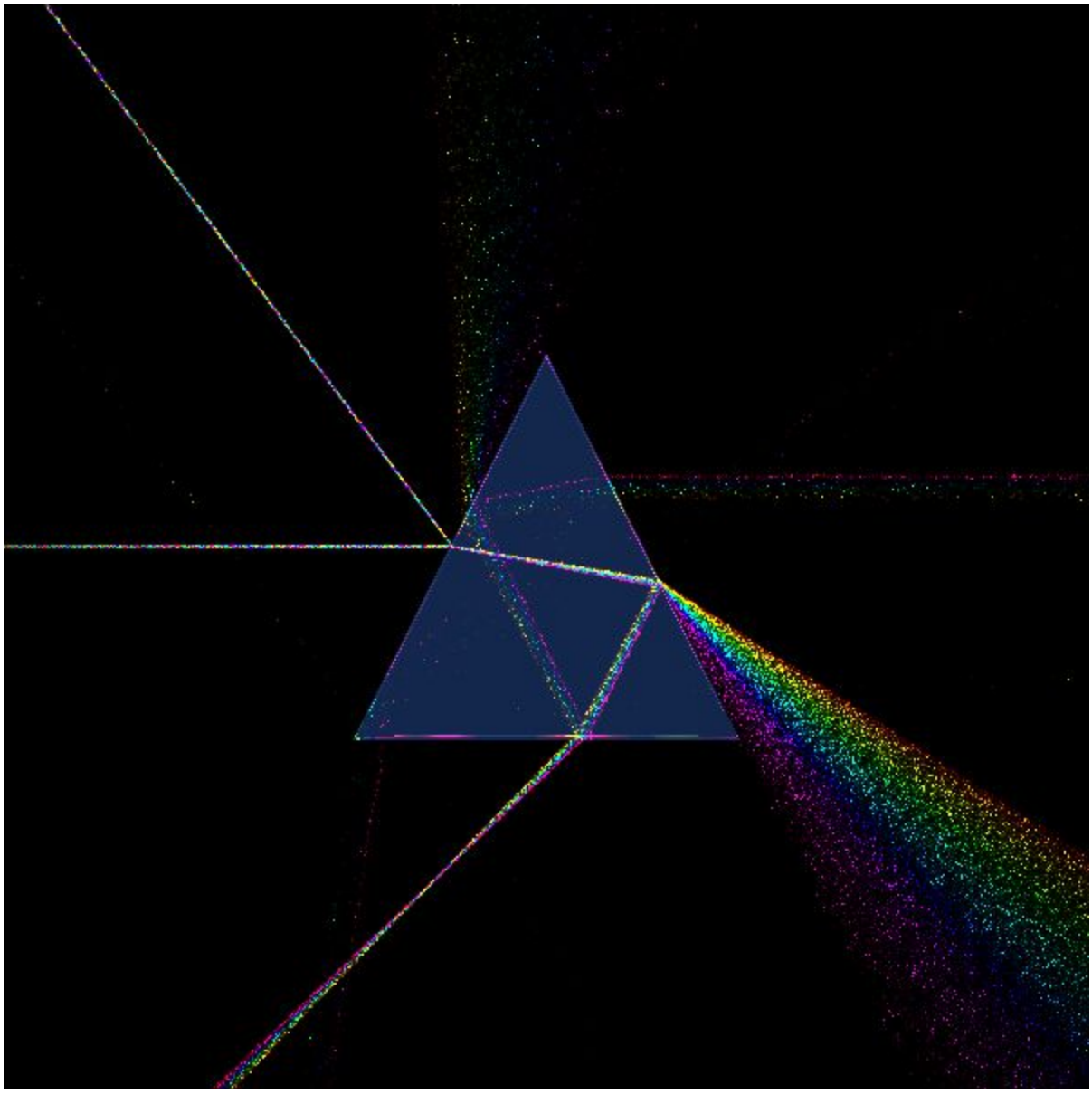
4. Results

There are two sets of results that correspond to our rainbow refraction. The first set is using a more narrow beam of light, so the individual details are more visible here. The second set is using a wider beam of light, which is a bit more visually appealing as the rainbow is more vibrant here.

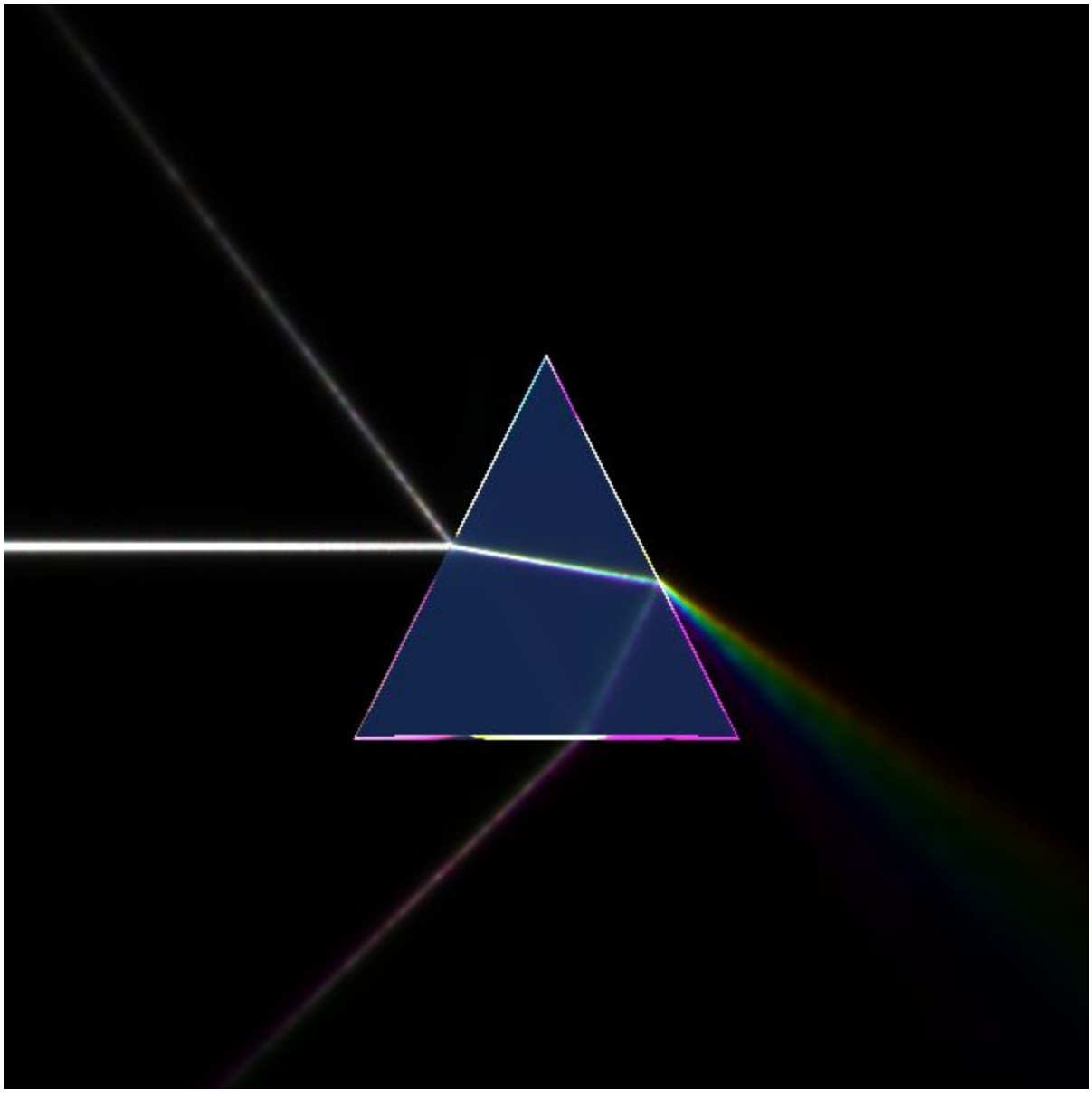
All images labeled “RGB” are images that were developed using our original, RGB-first method – photons are generated with random RGB values, and then their wavelength calculated. Images labeled “Wavelength” are images that were developed using the wavelength-first approach, where photons were spawned with random wavelengths, and RGB values calculated from those wavelengths. Images labeled as “Individual Photons” correspond to what the image looks like when displaying each individual photon without averaging. Images labeled as “Final Image” correspond to the final product. Note that not every individual photon is drawn – only a subset which happen to line up nicely in the center of the pixels. This doesn’t appear to affect the results too much.

4.1. Narrow Beams

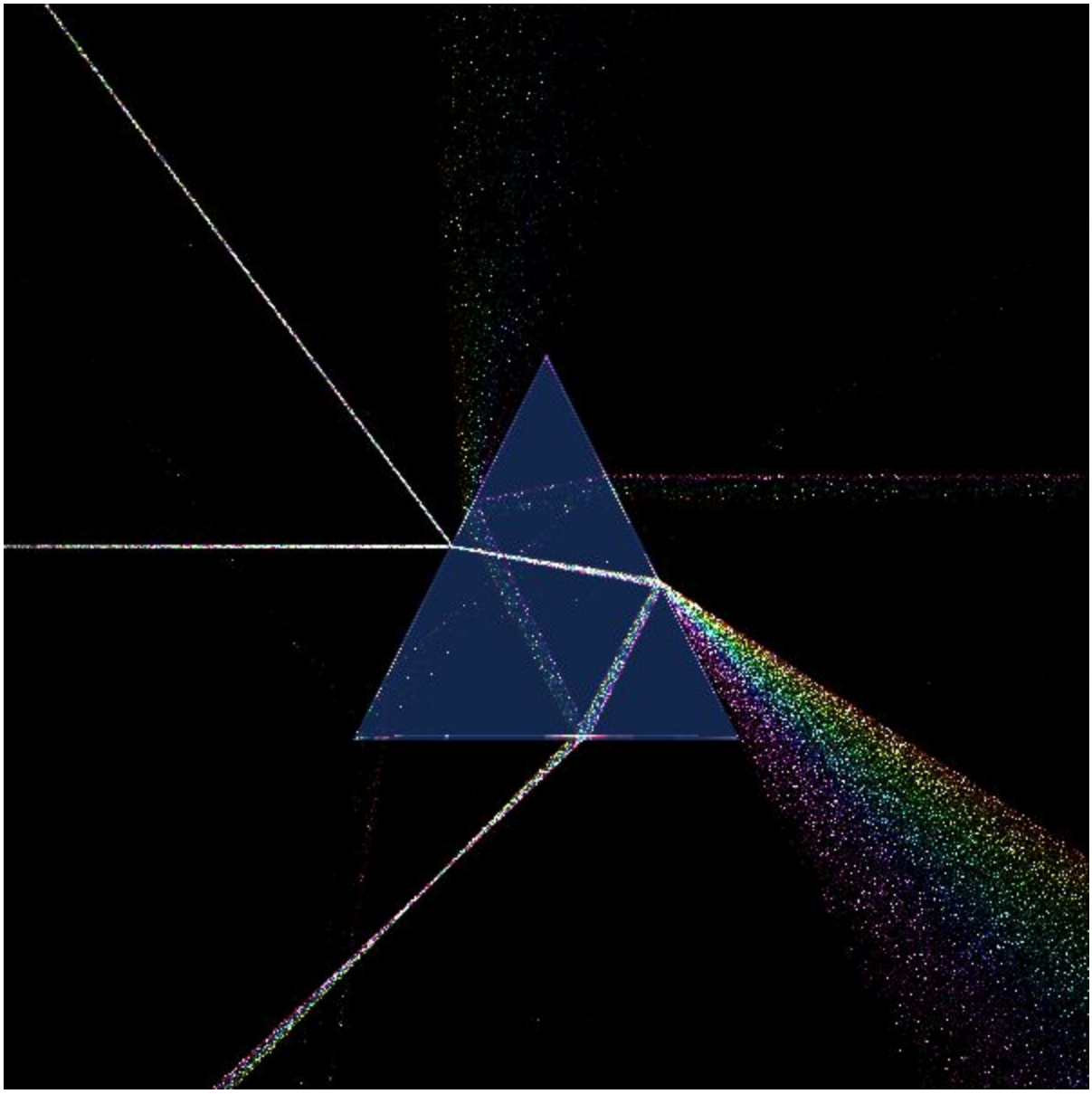
RGB, Random Hue: Individual Photons



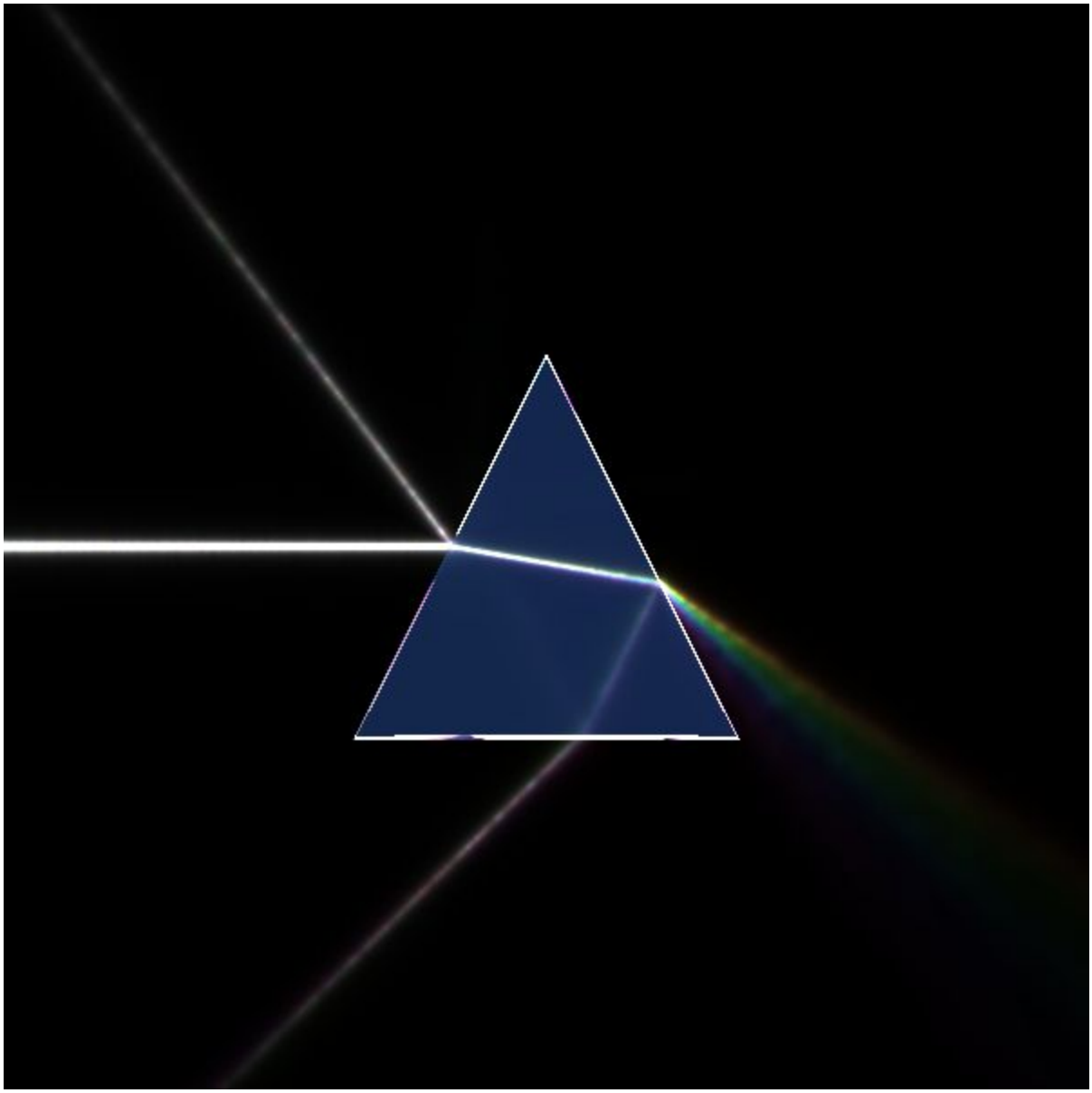
RGB, Random Hue: Final Image



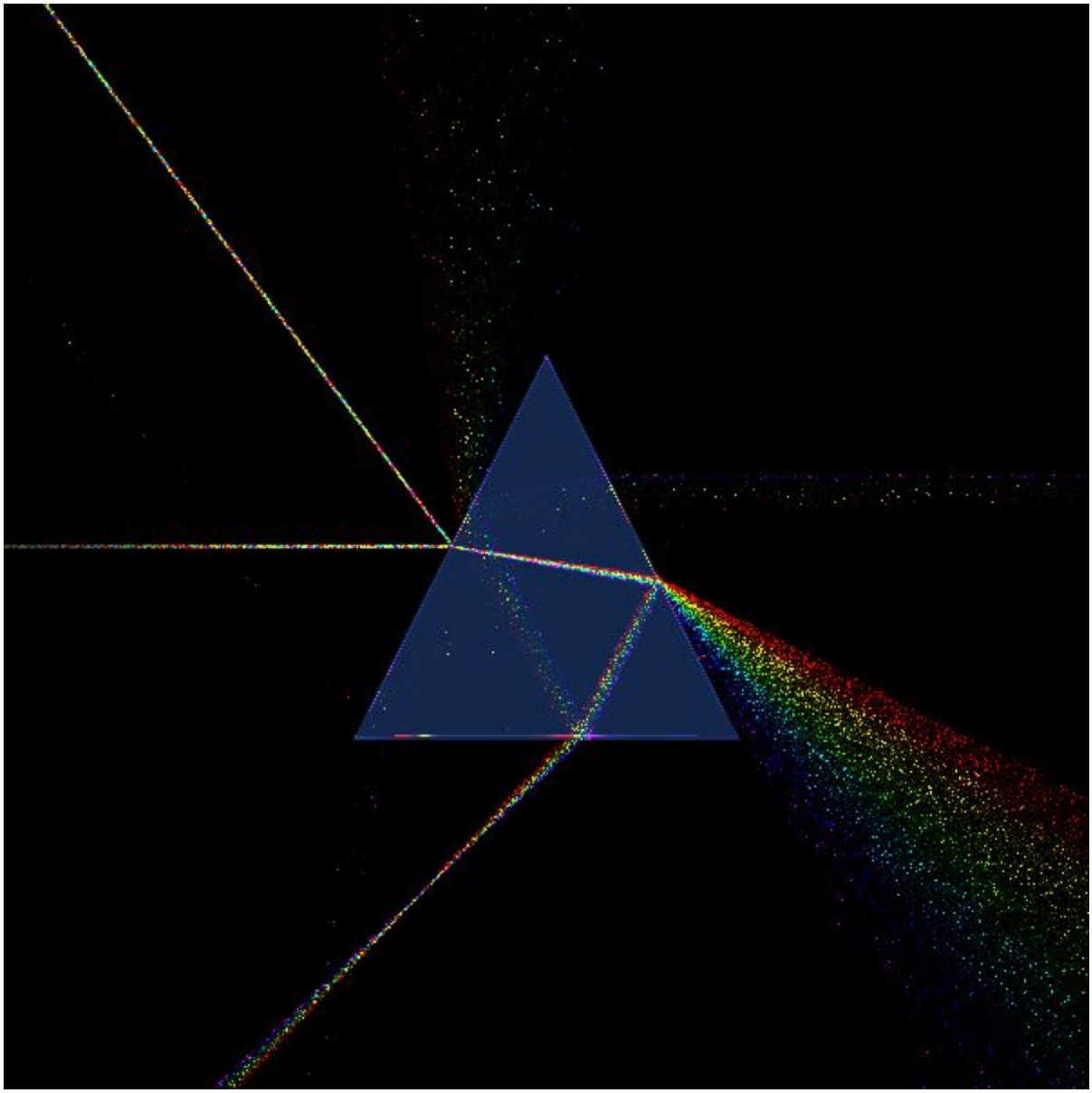
RGB, Uniform Distribution: Individual Photons



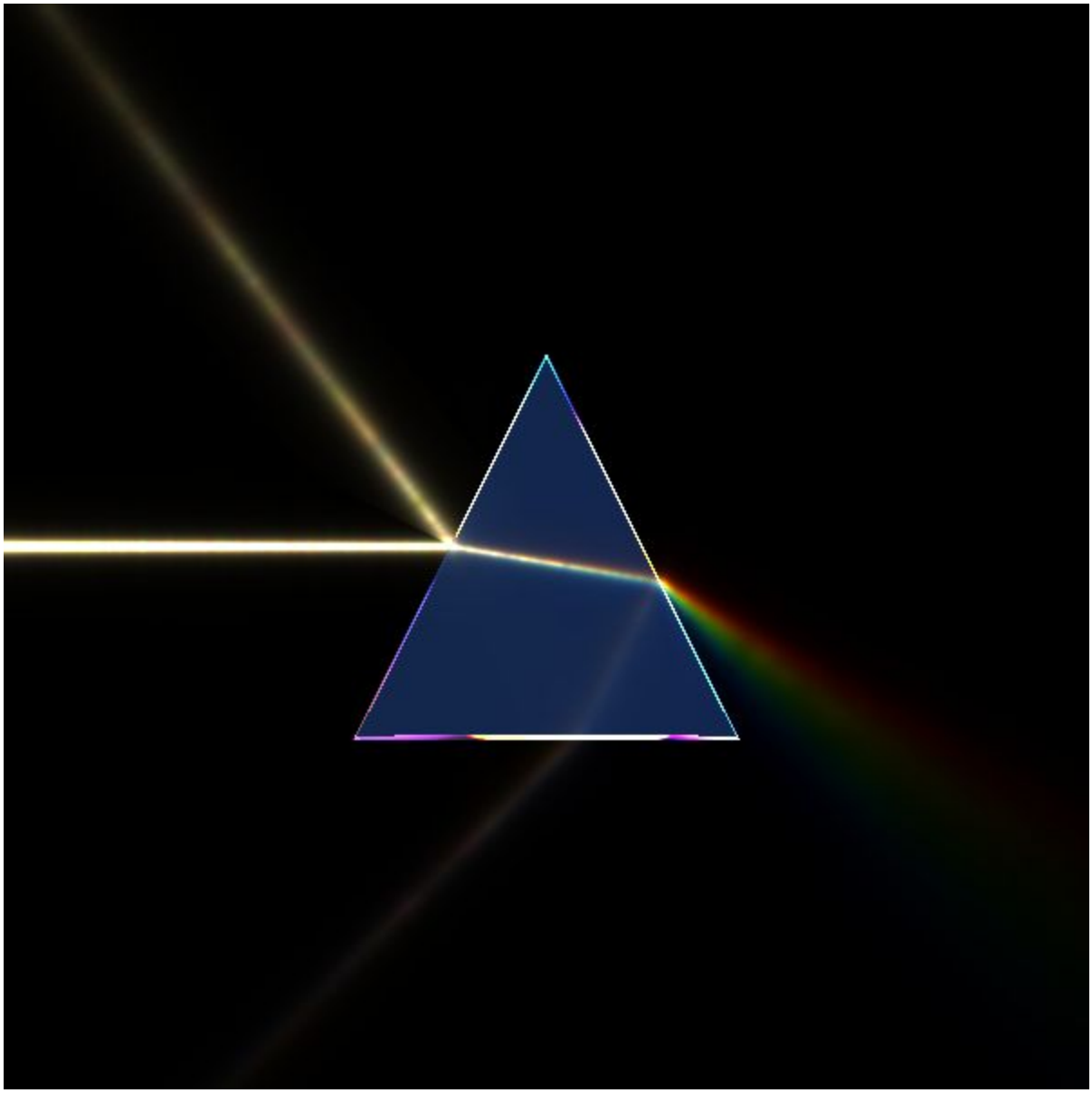
RGB, Uniform Distribution: Final Image



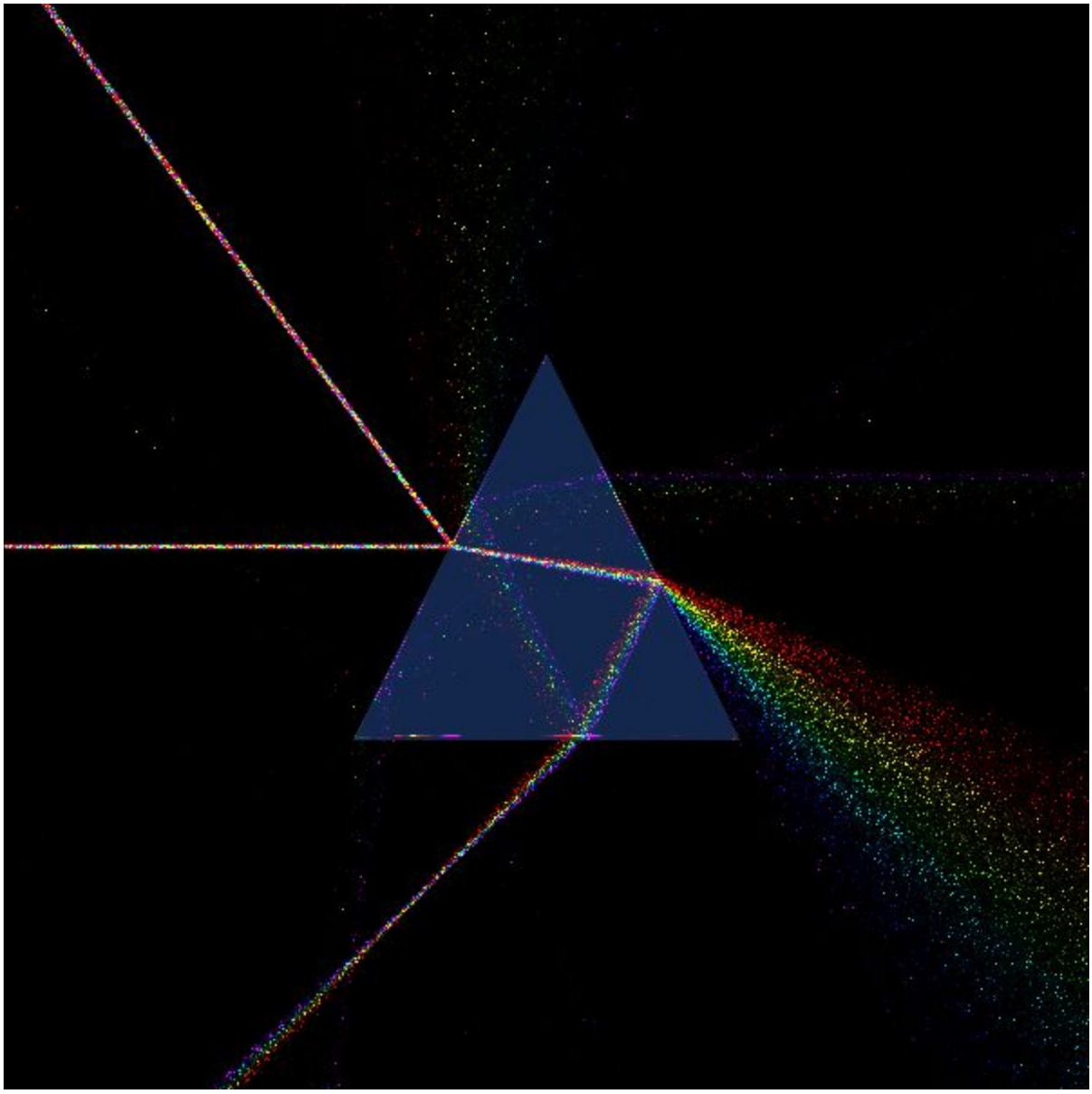
Wavelength, Uniform Distribution: Individual Photons



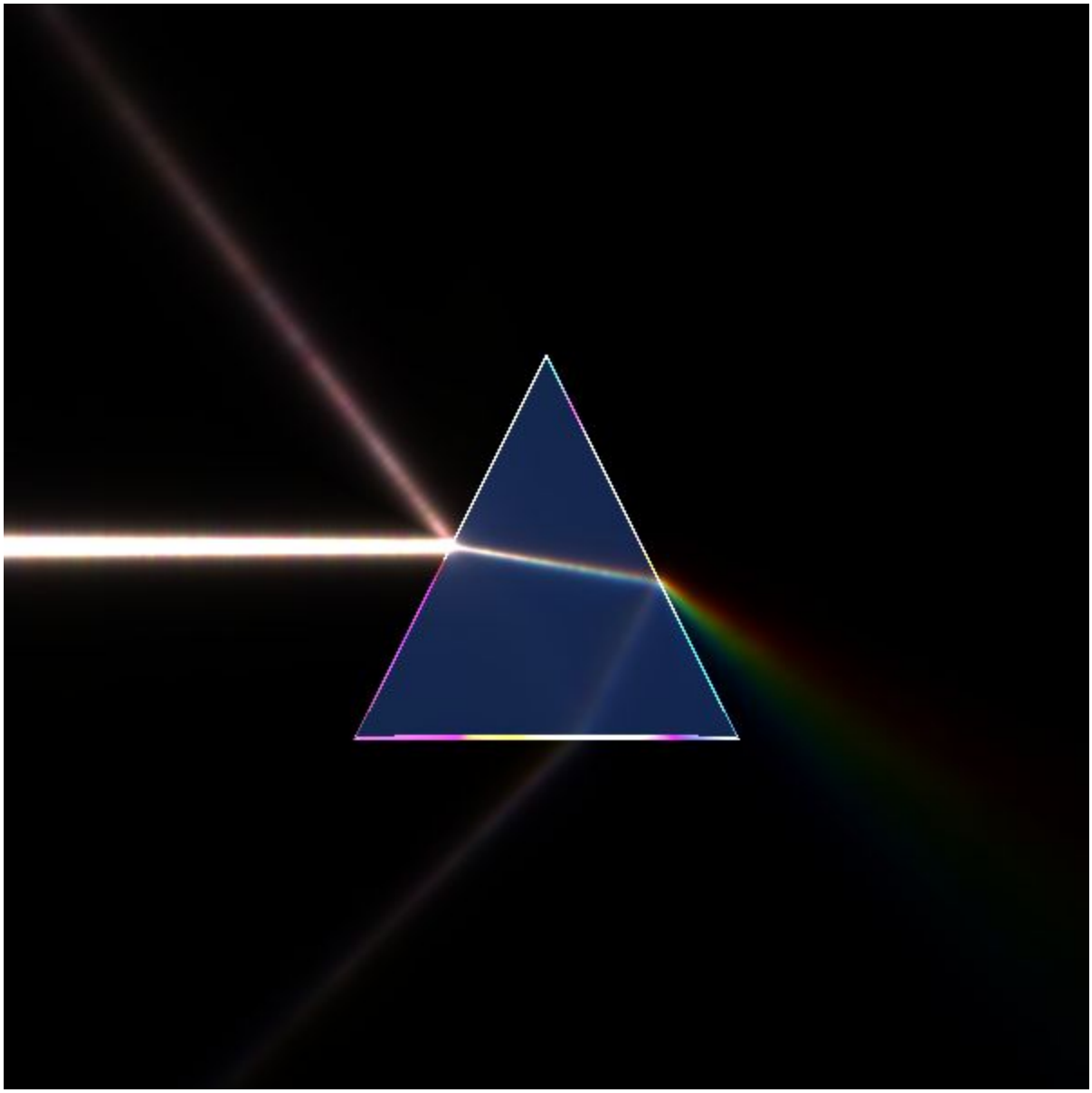
Wavelength, Uniform Distribution: Final Image



Wavelength, Wien’s Approximation-Based: Individual Photons

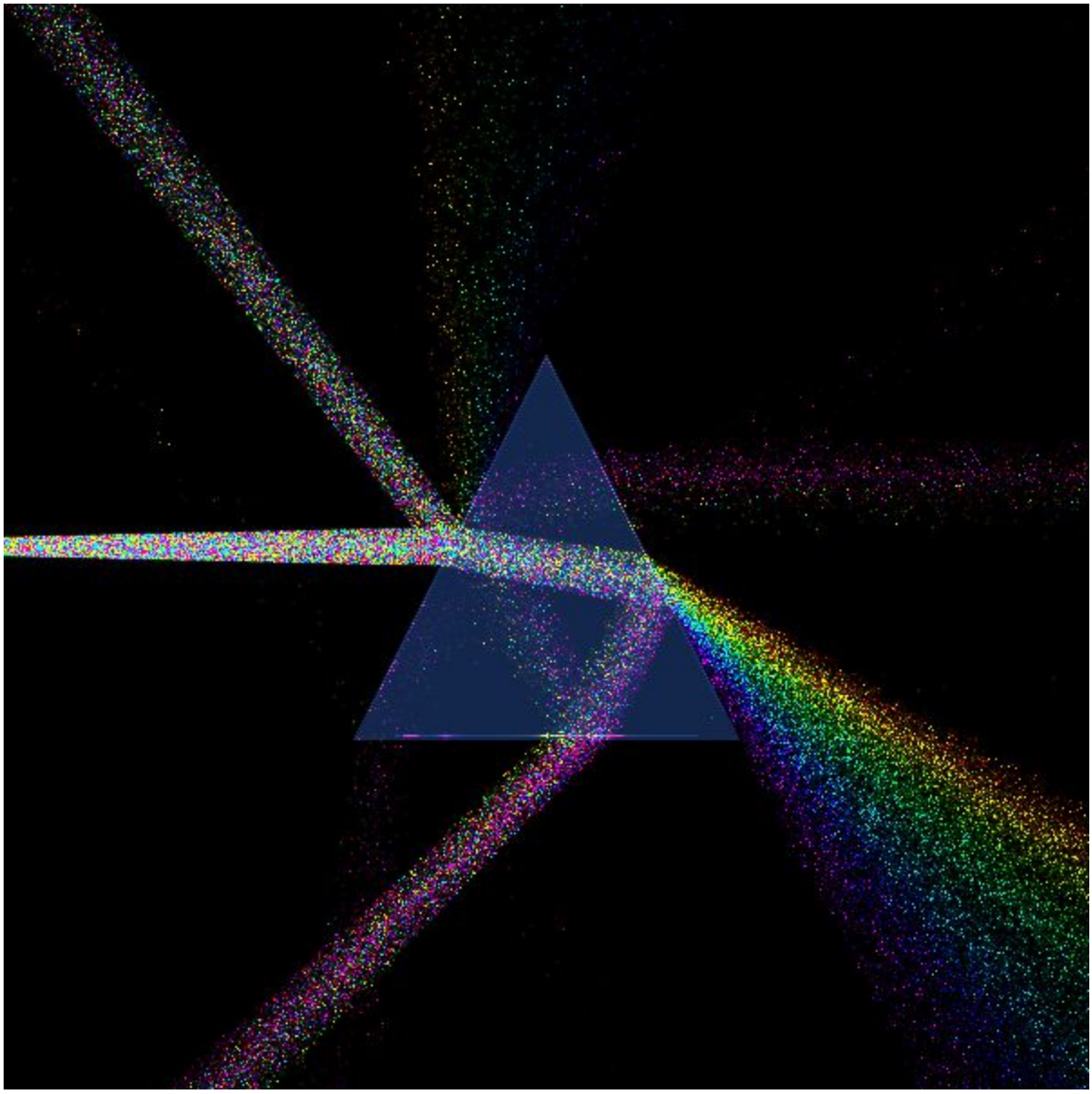


Wavelength, Wien’s Approximation-Based: Final Image

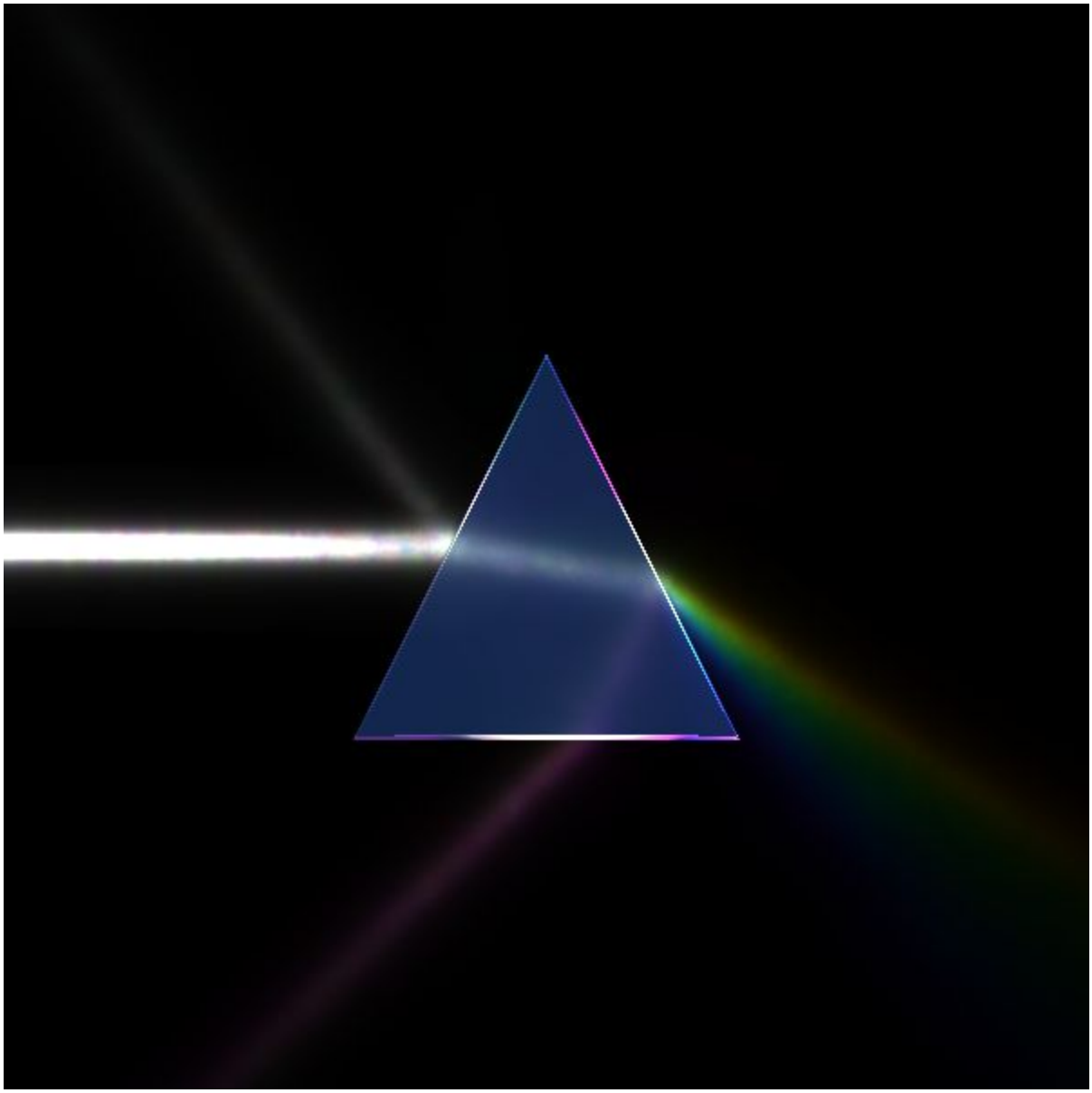


4.2. Wide Beams

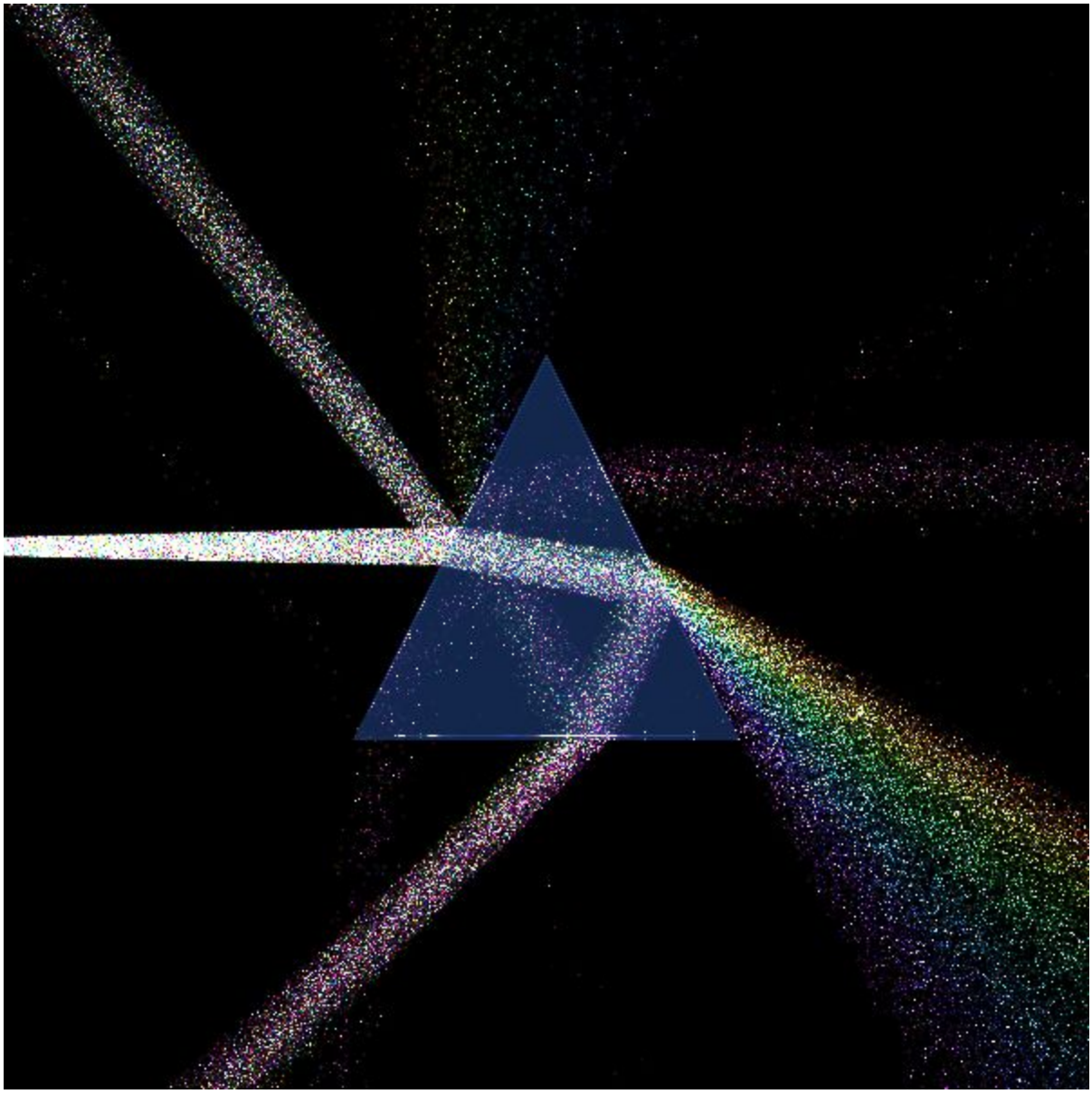
Hue: Individual Photons



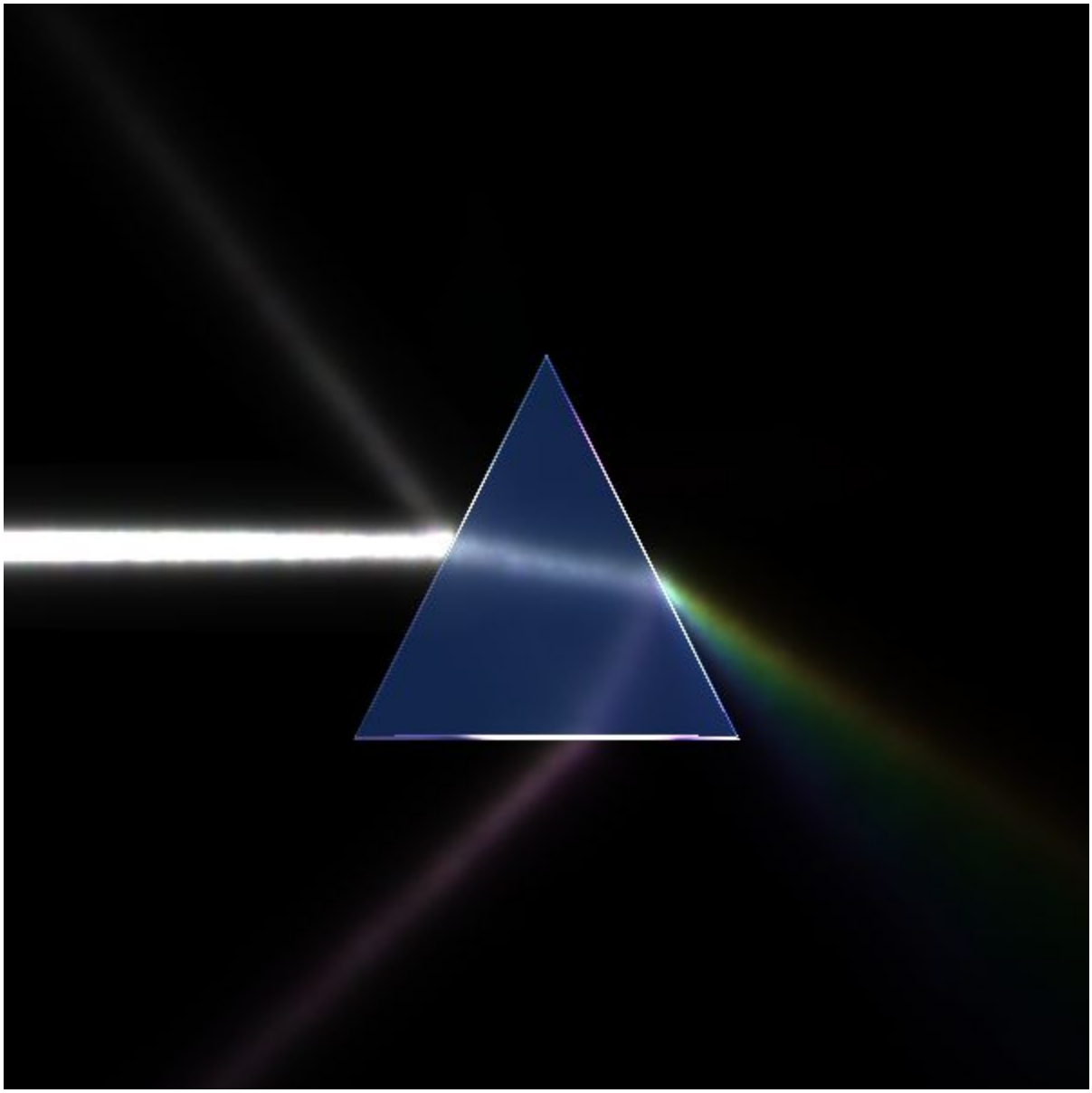
Hue: Final Image



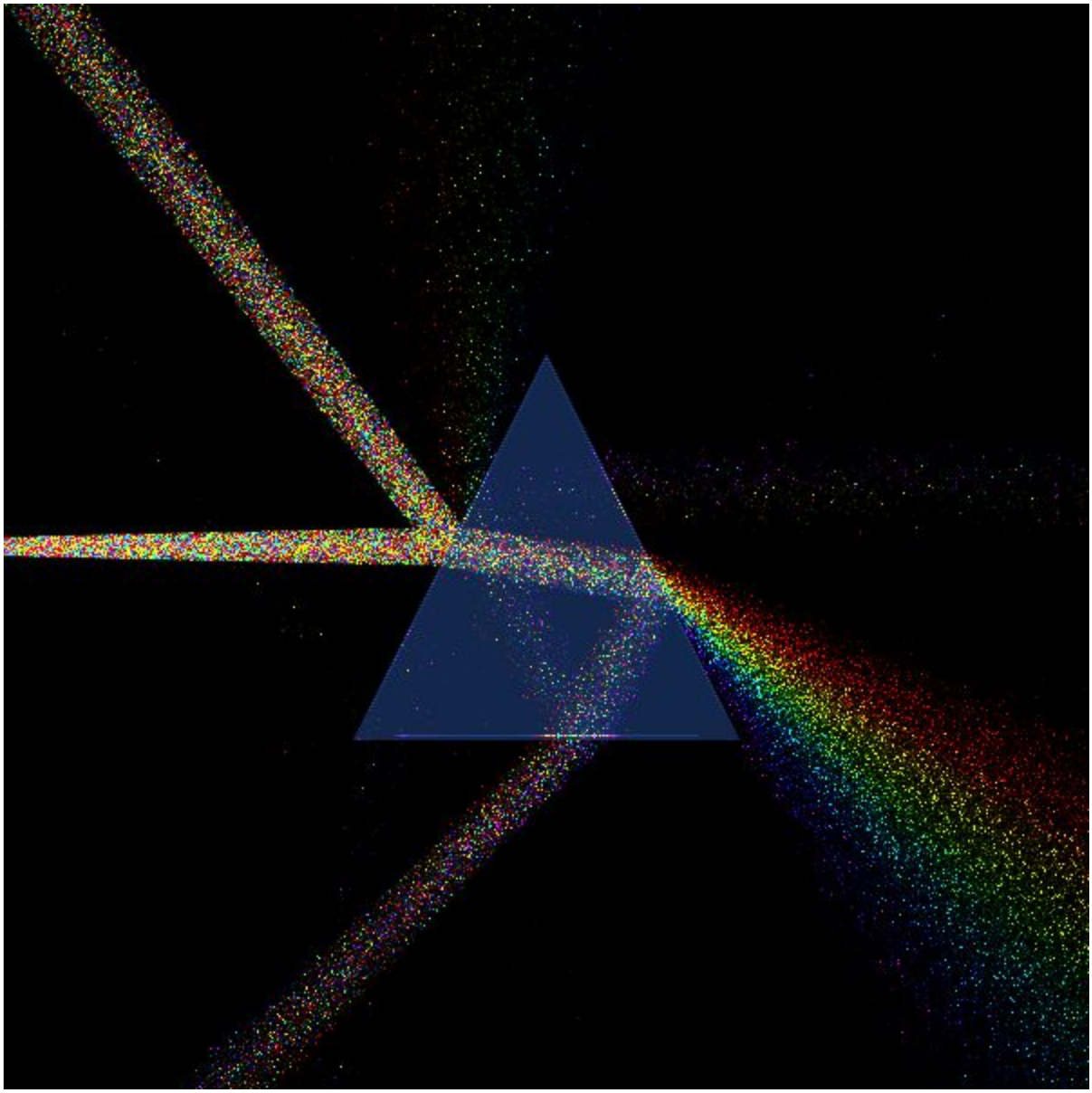
RGB, Uniform Distribution: Individual Photons



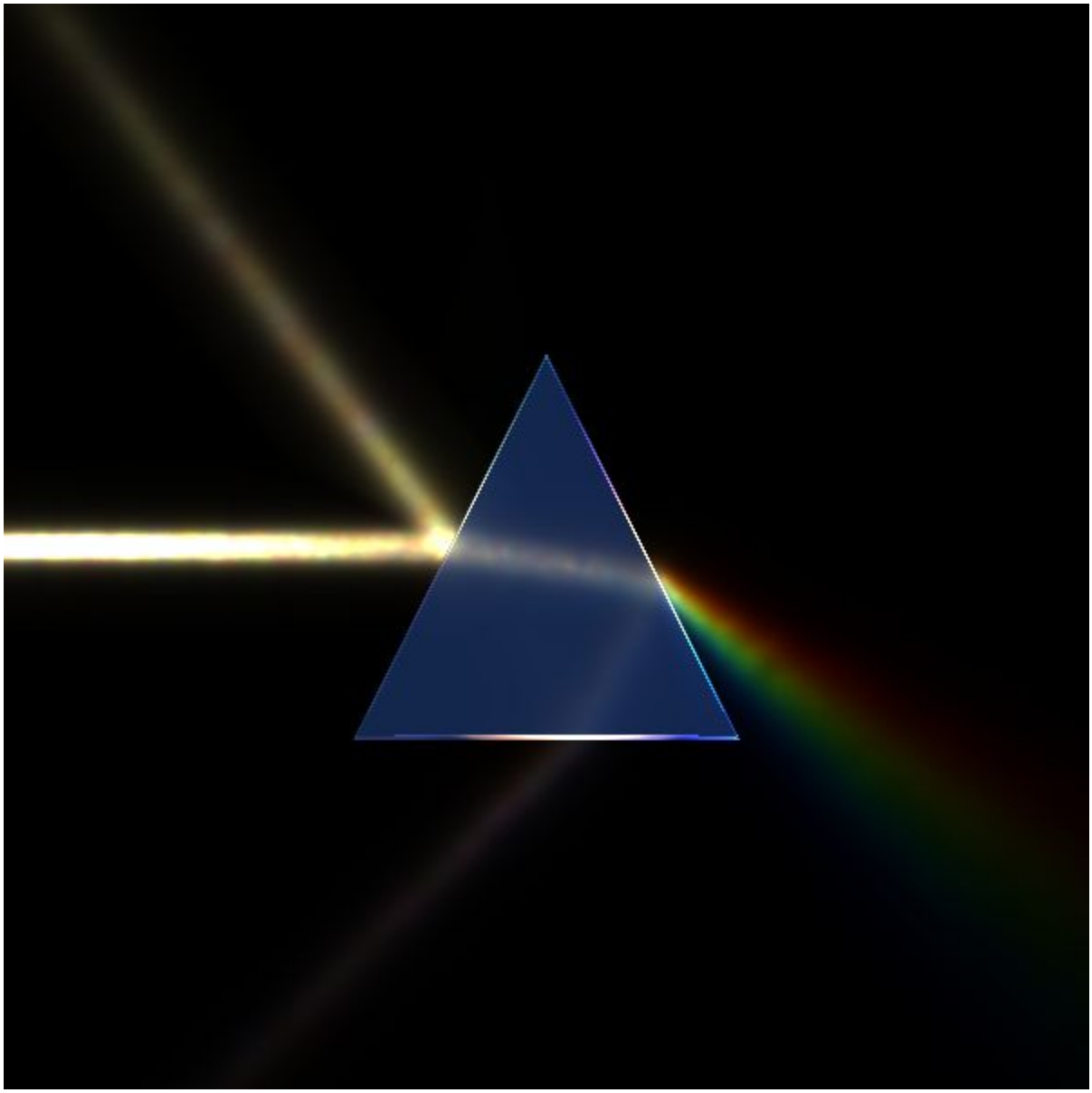
RGB, Uniform Distribution: Final Image



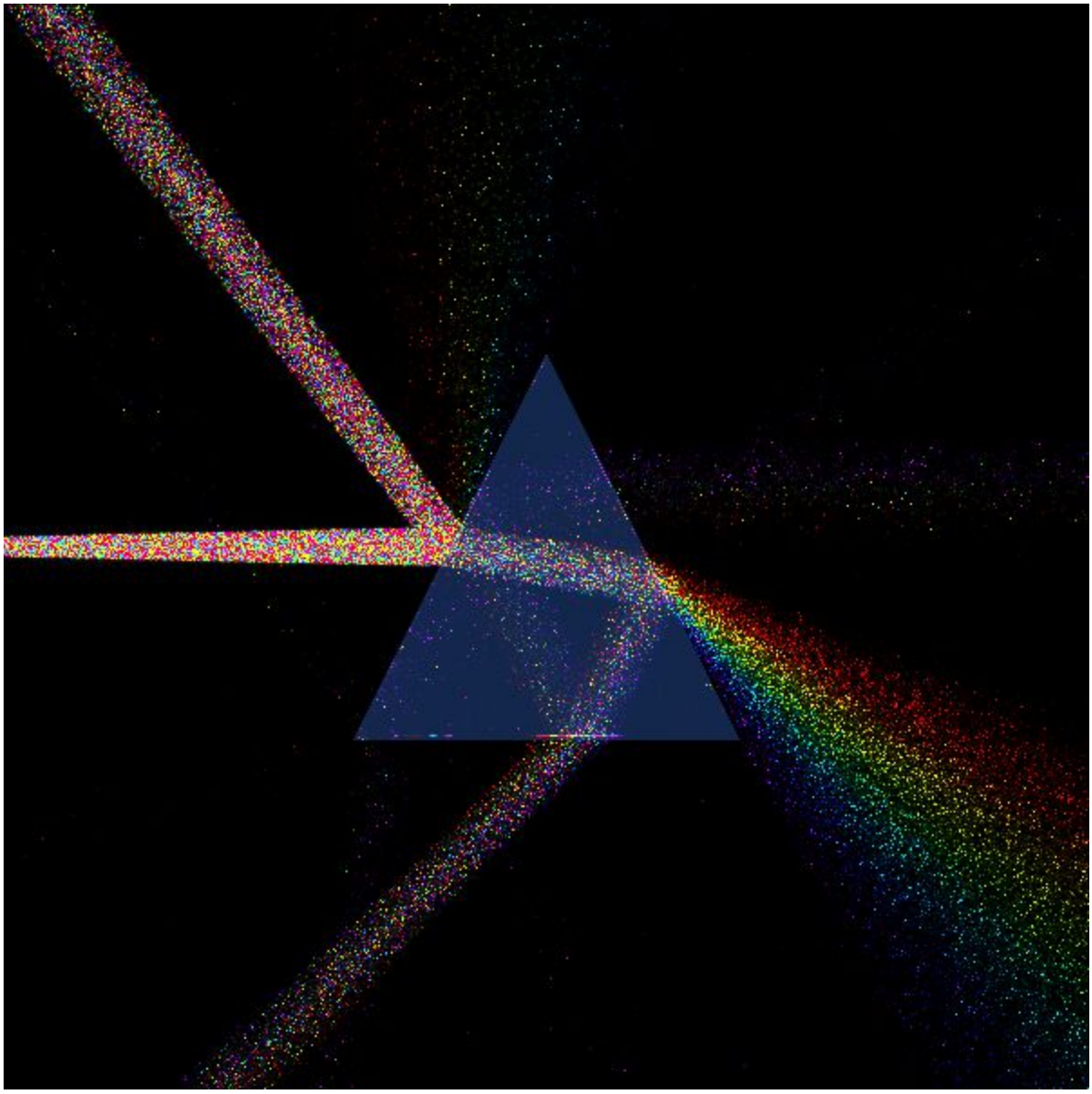
Wavelength, Uniform Distribution: Individual Photons



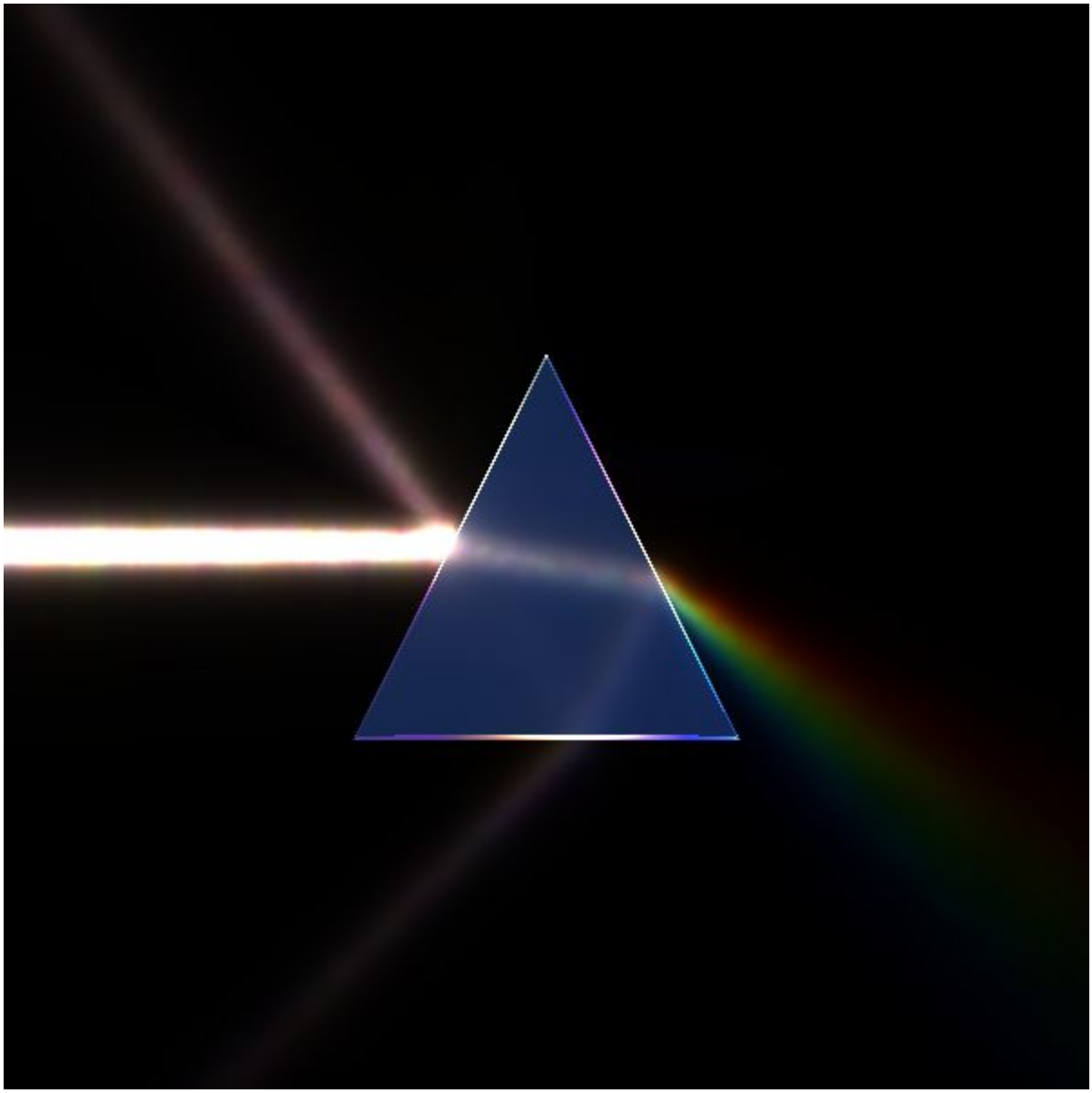
Wavelength, Uniform Distribution: Final Image



Wavelength, Wien’s Approximation-Based: Individual Photons

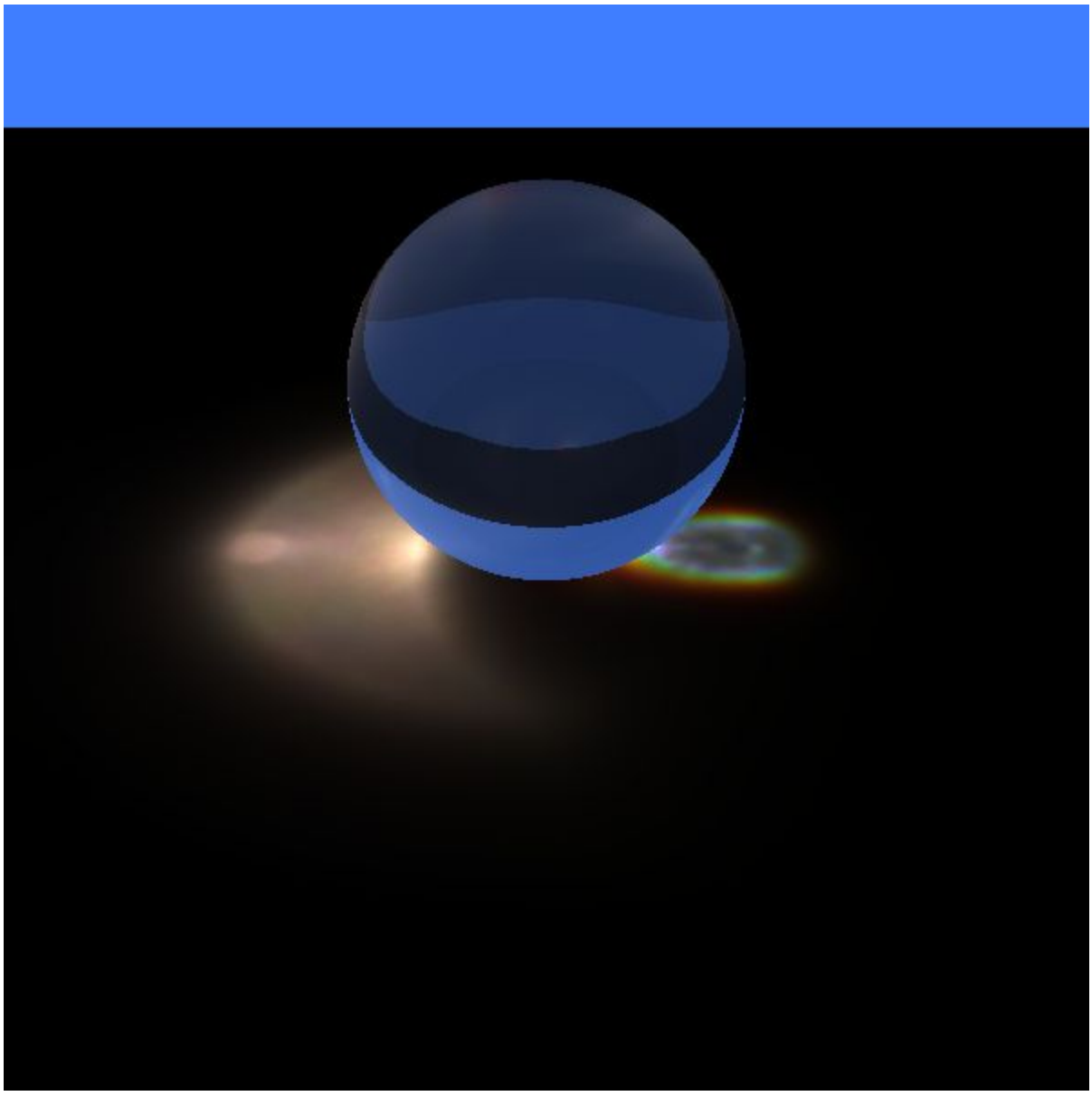


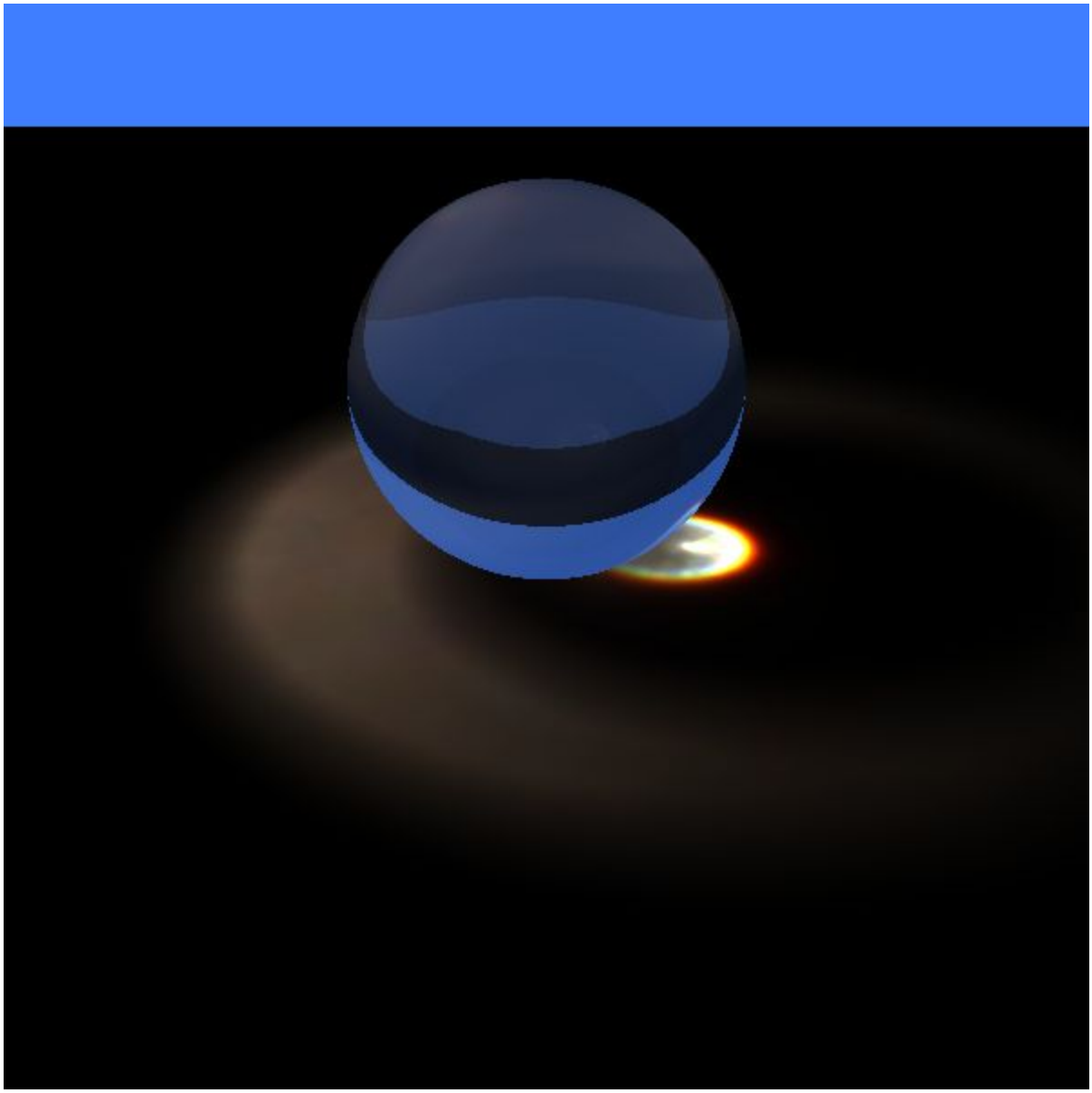
Wavelength, Wien’s Approximation-Based: Final Image

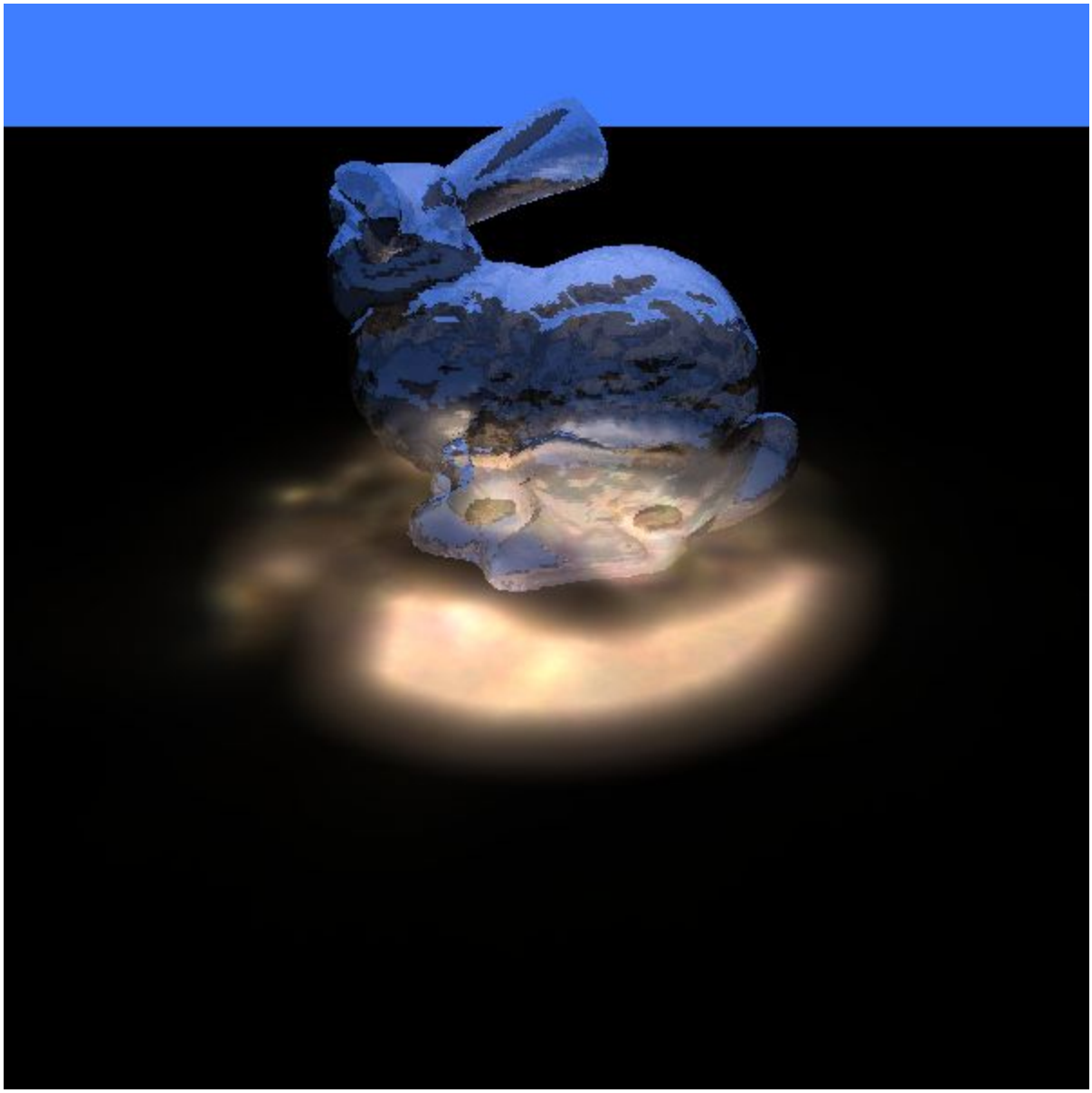


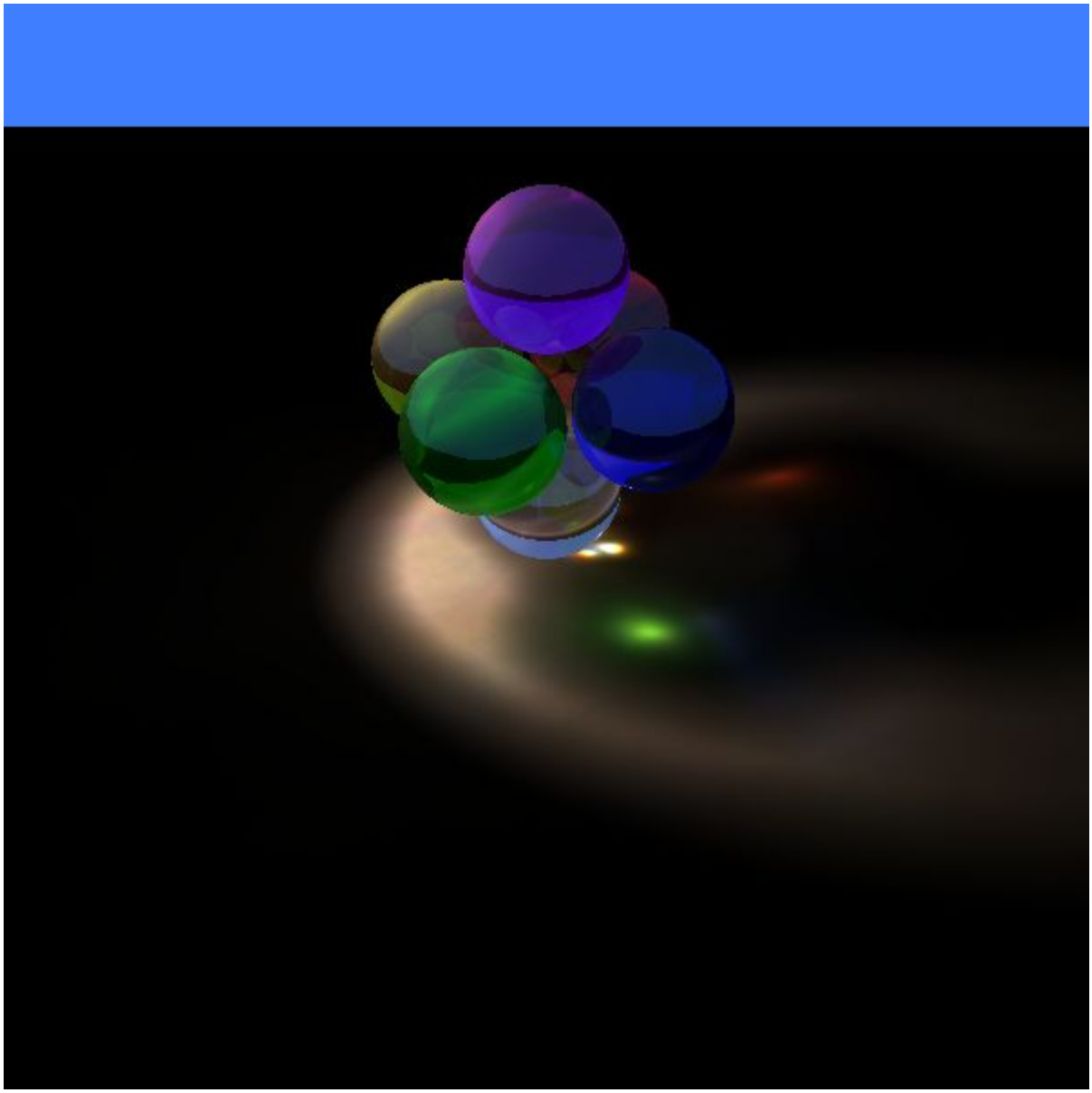
4.3. Other Results

We also attempted to use this method on a sphere and other transparent objects to see how well this would replicate a caustics effect. Here are some of the resulting images:









In addition to these, we developed several animated versions of the results, links to which follow:

1. Prism: https://youtu.be/mZRwXQ-hArE
2. Sphere: https://youtu.be/J4a_0Gij8jY
3. Several GIFs available on Kyle's website: http://www.kylecutler.com/courses/cs711/project/

5. Issues / Challenges

The result is not perfect. Some of the current issues include:

- The parameters for photon mapping must be adjusted by hand in order to attain a good result for different setups. For example, the sphere caustics image required a different number of photons spawned and a different number of closest neighbors than the prism.
- Material colors are still represented using RGB, so the means by which photons are reflected from surfaces are imperfect – a better solution would use reflectance values measured at several wavelengths to represent the colors.
- The reverse effect does not work – images viewed through a transparent surface will not have any color aberration – in order to do this, the rays spawned from the camera would need to have a wavelength attached to them.
- Because the photons are spawned with random wavelengths, there is color noise visible on diffuse surfaces, even when the photons have not been diffracted.

Problems we ran into while working on the project include:

- In order for the results from the prism to look good, we had to disable diffuse reflections from refracted photons. Otherwise the inside of the prism would appear far too bright. This means that several of our results are not exactly physically accurate.

6. Future Work

There are multiple ways that we could continue to work on this project. One of our original desires was to create a “light playground” where we would be able to interact with the lights and objects in real-time. The major challenge behind this is that it would require huge amounts of optimization in terms of the code. Currently, it takes a few seconds to develop even a single simple image, and if we want to be able to create these effects in real time with a reasonable amount of loading time, we would need to further optimize our algorithms and structures, and most likely offload some of the work to the GPU.

Another idea for possible improvement is when the photons are initially created, have them be “white”, but then if they are refracted, have them split off probabilistically into various wavelengths. This way, diffuse surfaces will have less color noise, but we can still get the diffractive effects we are looking for.

We would also like to add interference as one of the effects our ray tracer is able to simulate. Our original goals were refractive effects, caustics, dispersion, and interference; the refractive effects are complete, the caustics came out relatively well, and the dispersion goes hand-in-hand with the refraction and has been simulated relatively accurately. Since the current state of the project is already dealing with photons in terms of wavelengths, interference would be a great effect to pursue.

7. Resources

RGB to HSV conversion:
http://coecsl.ece.illinois.edu/ge423/spring05/group8/finalproject/hsv_writeup.pdf

Hue to wavelength conversion foundation:
https://stackoverflow.com/questions/11850105/hue-to-wavelength-mapping

Wavelength to RGB conversion:
http://www.fourmilab.ch/documents/specrend/

Sampling a random variable according to a Planck / Wien Distribution:
https://www.osti.gov/servlets/purl/420378