Rendering a Nebula

Sergey Levine* Edward Luong†

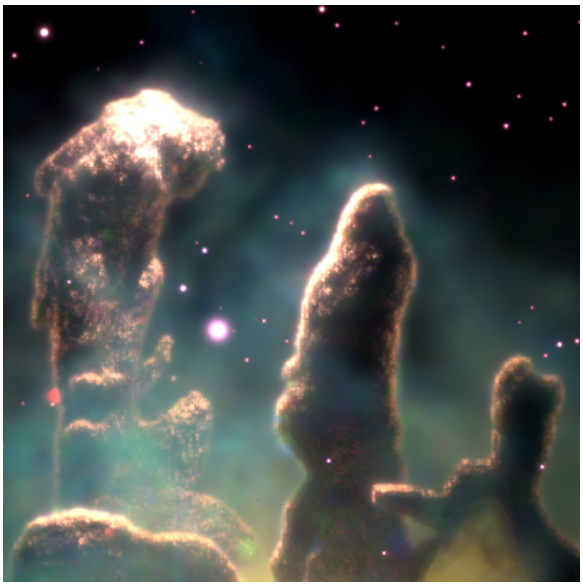


Figure 1: Final rendering of the nebula. This image was rendered at 400×400 resolution with 4 samples per pixel. The grid is a $200 \times 200 \times 200$ voxel grid with 8 samples per voxel for the density. We used 4 million photons and a maximum gathering distance of 15.0. The image took approximately 10 hours on Amazon's EC2 servers.

1 Introduction

An emission nebula is a massive cloud of gas ionized by a bright nearby star. Most of the visible light from such a nebula is emitted by this ionized gas, and creates dramatic lighting effects when viewed in the emission lines of the constituent gases.

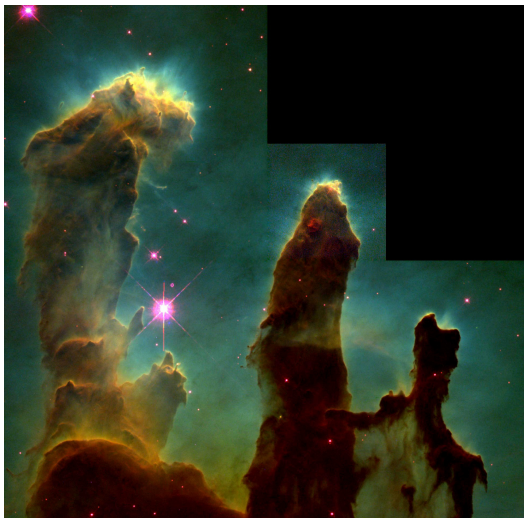


Figure 2: Hubble photograph H95-44a2

*email: svlevine@stanford.edu
†email: edluong@stanford.edu

Such a nebula presents an interesting rendering problem, because the volume for the nebula must be carefully generated to have a plausible variation in density and composition, and the rendering process itself must be able to properly simulate the emission of light and the scattering of emitted light by the dust particles which also make up the nebula.

Our reference image for this project was the famous Hubble photograph H95-44a2, shown in figure 2.

2 Modeling the Pillars

The shape of the pillars was created manually as two isodensity meshes. The inner surfaces corresponds to maximal density, and the outer surface to minimal density. Figure 3(a) shows the wireframe of the inner surface (blue) and outer surface (red) in 3D Studio MAX, 3(b) shows a rendering of the surfaces with the outer surface translucent, and 3(c) shows both surfaces rendered with the reference image texture mapped onto them. This last image provides a good comparison between this relatively primitive method of rendering the nebula and our method.

2.1 Generating the Pillar Volumes

The first step for computing the density grid for the volume is to measure the signed distance to each of the isodensity meshes. The distance is defined as negative if the point is on the inside of the volume defined by the closed surface, and positive otherwise. To this end, we exported the mesh into `pbrt` using a custom-built conversion program, computed vertex normals based on the normals of adjacent triangles, and used this information to determine signed distance.

(a) Wireframe (b) Solid (c) Textured

Figure 3: Pillars modeled in 3D Studio MAX

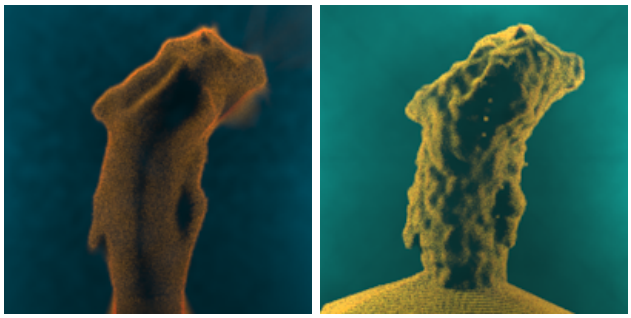


Figure 4: Comparison between non-perturbed and perturbed distance function

A KD tree is used to find the nearest vertex in the mesh, and the point is then tested against all triangles and edges the vertex participates in before being test against the vertex itself. The nearest vertex heuristic does not always produce the correct result, but if the mesh has regular-sized triangles, it is almost always correct. To compute the sign of the distance, the normal of the face, edge, or vertex is used. The normal of a vertex is the mean of the normals of the faces it participates in, and the normal of an edge is the perpendicular projection of the mean of its vertices.

We experimented with various falloff functions to generate the actual density. The initial attempt to use exponential falloff produces very sharp transitions, so we finally settled on a cubic falloff function which has a minimal density tangent at the outer curve and a maximal density tangent at the inner curve.

After initial experimentation, we found that computing distances while rendering the scene is too slow. However, since the distance function is quite smooth, we found that precomputing the distances into a density grid and trilinearly interpolating at render-time worked just as well.

To provide additional visual detail, we decided to perturb the density function for the mesh with procedural noise. The initial attempt to directly vary density by Perlin noise or turbulence did not look particularly appealing, because density would become unnaturally low inside the pillars and produce strange artifacts. Procedurally perturbing the sample location produces better results, and gave the cloud a "fluffy" look, but directly perturbing the distance according to a Perlin noise function appeared to be the most effective, as it gave the cloud well-defined "bumps" - figure 4 shows a side-by-side comparison of a rendering with an without the bumps. In the final image, the size of these bumps was reduced to avoid obscuring the hand-made details too much.

2.2 Generating the Surrounding Haze

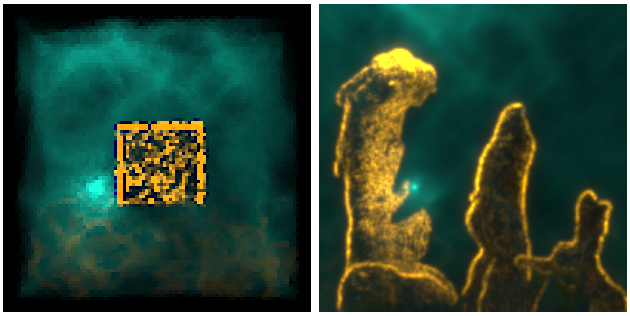


Figure 5: The final turbulence distribution used the turbulence function raised to an exponent

In addition to the density of the space immediately surrounding the pillars, the density of the surrounding haze had to be computed as well. In the reference image, this haze varies appreciably between the top and bottom of the image, and forms wispy clouds around them. To reproduce the irregularity of the haze, we used the Perlin-noise based turbulence function in `pbrt` raised to an exponent, producing the images shown in figure 5.

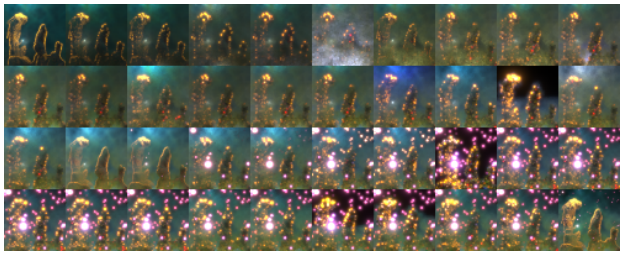


Figure 6: We ran numerous trial renders to find the haze distribution which most resembled the reference image

To vary the density of the haze in the vertical direction, we experimented with various functions before finally settling on cubic falloff with height. However, in addition to depending on height, the density of the haze towards the top of the image is higher closer to the pillars, so we also used the distance to the outer surface to affect the density. The final density function is a linear falloff with distance from the outer surface, with the rate of falloff depend on height, until about $\frac{2}{3}$ of the way down, where the haze smoothly becomes uniform in the horizontal slice and slowly increases until the bottom of the volume. Figure 6 shows the many experiments we ran to determine the proper density function - each square is a 50x50 test image.

2.3 Ionized Gas Model

The reference image was made with narrow-band filters that closely conform to the emission lines of a few gasses. The red color channel corresponds to the emission line from singly-ionized sulfur gas, the green to ionized hydrogen, and the blue to doubly-ionized oxygen. We used the same semantic for the color channels in our image.

To simulate emission we added an additional *emission* term to each voxel, and an $ionizing_{absorption}$ term. The ionizing absorption term indicates what percentage of light ab-

sorbed by the voxel in each color channel counts towards emission, and the emission term indicates how much of the total absorbed energy is re-emitted in each color channel. To simulate the fact that the majority of the ionizing radiation arrives in the UV frequency, we added a 4^{th} "UV" color channel. We decided that full simulation of the light spectrum was unnecessary to produce the phenomena we saw, because although gasses technically absorb and emit in the same frequencies, since the vast majority of the incident radiation was UV, we did not need to account for non-UV ionizing radiation. Strictly speaking, different materials absorb different frequencies of UV radiation, but we found that our image did not suffer significantly from treating all UV radiation as homogeneous, and the 4-channel model was sufficient for modeling the emission phenomenon. We assumed that emitted light is emitted evenly in all directions, so the emitted radiance for incoming light L is equal to $L \cdot \sigma_a \cdot ionizing_{absorption} \cdot emission/4\pi$.

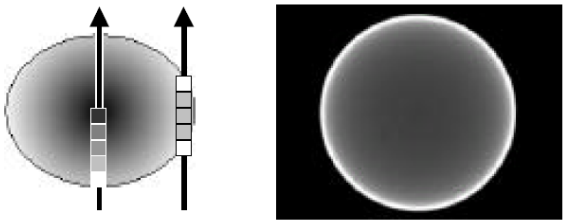


Figure 7: Halos form when highly emissive, thin gas surrounds denser but more opaque, cooler gas

One of the most interesting phenomena in the reference image is the bright "halo" around the edges of the clouds. This halo is caused by the ionization front of the gas that constitutes the pillars. The pillars themselves consist of cooler gas and dust which does not emit light. Because the ionization front is so thin, it is not readily visible from the front, because it is quite transparent. However, from the side, rays traveling to the eye pass through much more of the ionized gas, and therefore appear brighter. The diagram in figure 7 illustrates this phenomenon [Nadeau et al. 2001].

Besides the ionization front of the pillars, the surrounding haze is also slightly emissive, and contributes to the greenish-blue hue of the image.

2.4 Three Materials

Initially, we had hoped that simply varying the density of volume close to the pillars would be sufficient to create an ionization layer that is absorbent enough to emit a lot of light but transparent enough to allow "halos" to be visible at grazing angles. However, such halos appear very blurred because non-emitting gas just under the ionizing layer was transparent enough to allow halos to "bleed" from the back of the structure. The cooler hydrogen gas in such structures is actually molecular H_2, which is very efficient at absorbing radiation. The ionizing radiation on the surface breaks up these molecules, so the ionization front actually has different optical properties. We modeled this by providing the interior of the pillars with different σ_t, σ_a, $ionizing_absorption$, and $emission$ values. To determine which value to use, we test against the density of the voxel: the density is raised to a power, and this is used as the concentration of the H_2. The difference between the density and this value is the ionized gas. While this is a rough approximation, it is sufficient for estimating the upper bound on the size of the ionization layer, since it is not visible when not illuminated by UV light, and therefore does not need to be computed exactly.

The haze surrounding the pillars also has a different composition, as shown by its bluish color. To model this, we implemented a third material and determined its percentage of the density by checking how close the density was to the mean haze density, with a smooth transition into ionization front material as the density increased.

In addition, the haze contains scattering interstellar dust, which was modeled according to [Magnor et al. 2005] as having a Henyey-Greenstein phase function with an anisotropy factor of 0.6 and an albedo of 0.6. This scattering component is part of what makes the bright beams from the emitting gas on the pillars visible in the reference image.

3 Volumetric Photon Mapping

Simulating light transport is necessary to achieve realistic images of volumes. While single scattering does fairly well for thin, homogeneous volumes, it is insufficient in thick, highly scattering participating media. Our scene is made entirely of scattering, non-homogeneous volumes. Moreover, much of the light in the scene is a result of emitted light that is scattered which produces the halo around the pillars. The halo is necessary to create a convincing image of a gaseous volume. This combination presents a difficult challenge for the standard volumetric rendering packages in `pbrt`.

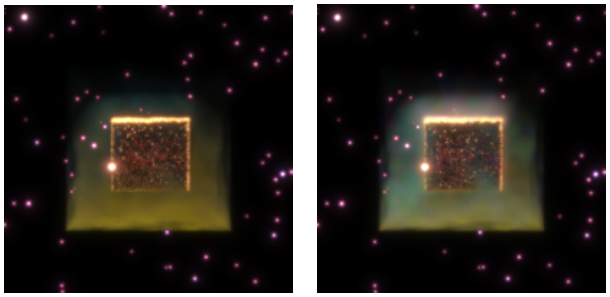
Such an effect cannot be properly captured using an area light either. The main issue is that we cannot properly define the surface that is visible to UV rays. Moreover, that approach is much more difficult and would require a new area light be generated for different volumes.

The natural choice for us was to extend volumetric photon mapping techniques explain in [Jensen and Christensen 1998]. Volumetric photon mapping is a two-stage algorithm that solves the light transport equation by first tracing paths of photons from the light and storing where interactions occur. In the second stage, the radiance across a ray is computed by marching across and gathering nearby photons. The gathered photons can be used to estimate the radiance along the ray. Figure 8 compares images rendered with and without photon mapping.

3.1 Photon Tracing in Non-homogeneous Medium

During the photon tracing stage, we fire photons out of the lights and trace their progress through the medium. Photons may either interact with the medium or be transmitted through. In the case of interaction, we store the photon and determine whether or not it becomes scattered or absorbed. At first, we implemented photon tracing using ray marching; however, this approach resulted in too much variance. Instead, we importance sampled interaction time, as alluded to in both [Jensen and Christensen 1998] and [Magnor et al. 2005].

Neither paper explains explicitly how to do this so we had



(a) Single scattering (b) Multiple scattering

Figure 8: Comparison of a box nebula volume rendered with different volume integrators. Notice how with single scattering, the box looks very sharp, whereas multiple scattering adds volume around it to soften it. Also, multiple scattering makes the image brighter as invisible UV light is being scattered into the visible spectrum.

to fill in the details on our own. Jensen gives the cumulative probability density function, $F(x)$ for the probability of a photon interacting with the media.

$$F(x) = 1 - \tau(x_s, x) = 1 - e^{-\int_{x_s}^{x} \kappa(\xi) d\xi}$$

where x_s is the current location position of the photon and x is the location of the next interaction. Given a uniformly random $u \in [0, 1)$, we essentially solve for x using the inversion method though inversion here is not necessarily simple because the medium is non-homogeneous. That is, with $u = F(x)$:

$$\begin{aligned} u &= 1 - e^{-\int_{x_s}^{x} \kappa(\xi) d\xi} \\ e^{-\int_{x_s}^{x} \kappa(\xi) d\xi} &= 1 - u \\ \int_{x_s}^{x} \kappa(\xi) d\xi &= -\ln(1 - u) \end{aligned}$$

We solve this integral using simple numerical methods. Note that the probability density function, $f(x) = F'(x)$ is simply the transmittance, $\tau(x_s, x)$ multiplied by the extinction term, $\kappa(x)$.

Another difficulty in photon tracing was getting all the weights correct such that energy was conserved. The majority of the time spent on implementing multiple scattering (aside from tweaking settings in the scene file) was devoted to understanding the literature and balancing the equations. Our problem had the added difficulty that we wanted to simulate the scattering of emitted light as well whereas conventional approaches only calculate emission directly (using single scattering).

3.2 Emittance and Scattering

When a photon interacts with the medium, it is either scattered or absorbed. The probability of scattering is given by the albedo. We flip a coin to determine which event occurs. If scattered, we pick a new direction uniformly at random

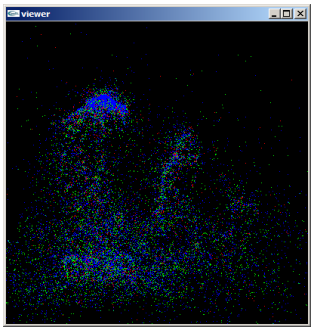


Figure 9: OpenGL viewer to help visualize the photon tracing stage.

and attenuate it by the phase function. If emitted, we effectively create a new photon with the emitted light and pick a direction uniformly.

3.3 Per Color Channel Tracing

The type of interaction with the medium depends largely on the density and the wavelength of light. Rather than simulating the whole spectrum, we augmented the existing spectrum to include UV light and made scattering and absorbency dependent on color channel. This extension required that photons that contained more than one color component to be split up into the individual channels, and then each traced separately.

3.4 Photon Viewer

In order to aid in debugging the photon tracing stage, we wrote a small OpenGL application that allowed us to visualize the locations of photons. It allowed us to examine where photons of certain color channels were and helped explain artifacts we had when trying to tweak the final image.

3.5 Photon Gathering

The photon gathering stage was relatively straight-forward once weights on the photon tracing stage were correct. One problem we ran into was bright photons appearing as sharp spheres. `pbrt`'s extended photon map dealt with this problem by using a blur kernel during gathering which we adapted for our gathering stage as well.

3.6 Managing High Variance

Since there is so much participating media, our photon map has a lot of variance. We spent a considerable amount of time into reducing the variance. In the end, we still had to use 4 million photons to get a relatively smooth image.

3.6.1 Importance Sampling Lights

Importance sampling the lights was the first variance reduction technique we employed. By choosing lights according to their power, we avoid emitting photons with low energy

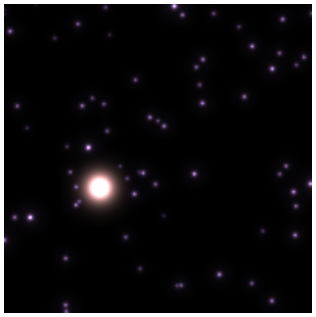


Figure 10: Stars are rendered as points and form Gaussians in the final image due to the "bloom" effect used in EXR conversion

that are unlikely to contribute during the gathering stage. `exphotonmap` provided with `pbrt` uses this as well, so porting that code into our photon tracer was straight-forward.

3.6.2 Tracing UV Rays

We noticed early on that the main effect we wanted from multiple scattering was the scattering of emitted light. Single scattering capture many effects in the sparse gas quite well and we wanted to leave as much of that intact as possible. Consequently, we only emit UV photons from light sources. After these photons are absorbed and emitted into the visible spectrum, we continue to trace these photons. This allowed us to ignore the scattering of RGB light from the stars which was unnoticeable when scattered multiple times.

3.6.3 Ray Marching for Direct Lighting

As suggested by Jensen, we continued to use ray marching to compute the direct lighting term rather than store those photons in the photon map. The photon map could store direct lighting interaction; however, doing so would result in a lot of photons being deposited on the first interaction. This decreases the number of photons being used for multiple scattering which is what we primarily needed it for. Moreover, since we were only tracing UV rays, we were not even considering direct interaction in RGB channels emitted from the lights.

3.6.4 Forced Interaction

Another strategy to reducing variance in multiple scattering was artificially increasing the coefficients to force interactions. This allowed us to have more photons in the areas we cared about. While this does introduce bias into the image, it does so in a way that produces much more desirable results in the long run. Moreover, this provides speed-ups in photon tracing as less photons need to be traced from the lights.

4 Rendering Foreground and Background Stars

Considering the scales involved in the image, stars are point light sources. As such, they are not visible on their own. In reality, even the tiniest point light sources are visible because their radiation strikes the photo-sensor and illuminates adjacent cells. To properly render stars, we added an additional "star-rendering" pass to the `pbrt` sampler model, where each star location is passed as a special sample to the camera, which computes the position on the film that the light from the star strikes. This ray is then processed as usual, except that it is terminated at the position of the star, where the star's power, attenuated by the square of the distance and the participating medium, is added to the ray.

The accumulated energy is deposited at a single point on the film, which creates a very bright spot. When converting from EXR to low dynamic range formats, a bloom filter is used to expand this bright spot into a Gaussian.

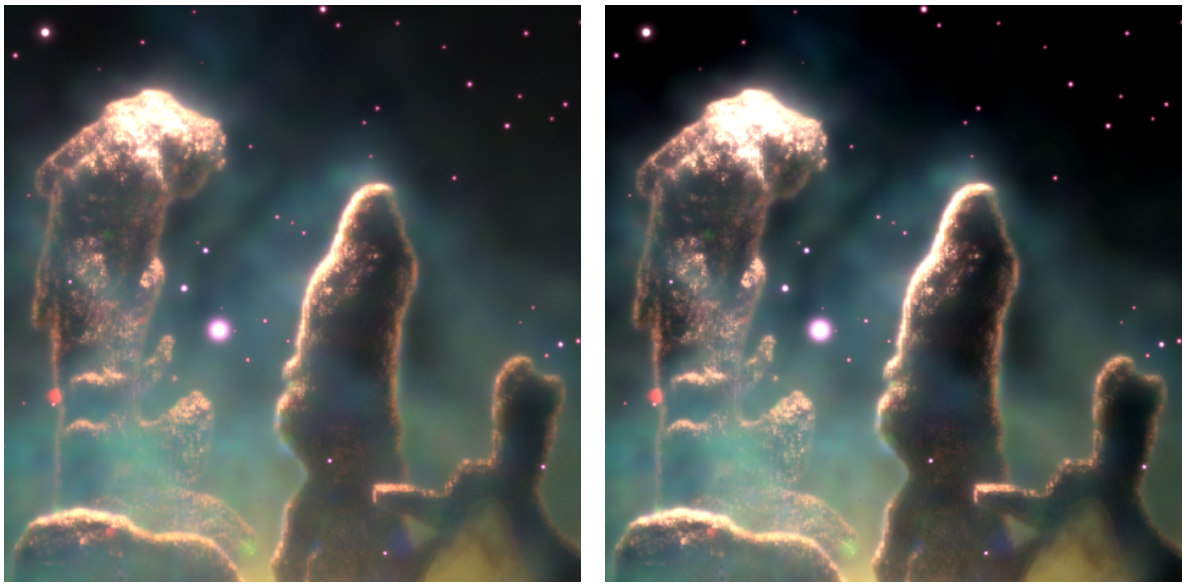
Except for a handful of manually placed stars, most stars in the image are placed randomly to provide the starry background. Light sources are not attached to most of the stars to reduce variance, since most are not bright enough to appreciably alter the appearance of the image.

5 Acceleration and Video Rendering

(a) Aliasing Detail (b) Color grid (c) Original

Figure 11: Comparison of images rendered using the color grid and from the original density field

Rendering the entire scene using single scattering integration and photon mapping takes a very long time. In order to construct an animation of the scene, we needed some way to significantly decrease rendering times. To this end, we constructed a color grid out of the image by computing the total radiance in six basis directions at each voxel on a 200x200x200 grid, as well as the transmittance. This allowed the entire scene to be re-rendered quickly using a simple emission integrator and a new volumetric primitive which returns the emissive color at a particular point in a particular direction by trilinearly interpolating the color grid to obtain values for the three adjacent directions, and interpolating between them. Although our program is able to construct the color grid with an arbitrary number of samples per voxel, we had to point-sample the volume when constructing the final color grid since the computational times were becoming very high. Because of this, mild aliasing is visible when rendering using the color grid at higher resolutions. However, we were able to render one frame per minute at 400x400 resolution. Figure 11(a) below shows detail from a high-resolution image rendered using the color grid which shows aliasing, and a side-by-side comparison of low-resolution images rendered with the color grid 11(b) and from the original density map 11(c).



(a) Rendered Image (b) Adjusted Contrast

Figure 12: Final renders of the nebula.

6 Results

In order to reduce the rendering times, we tiled our image and rendered each tile on a separate `pbrt` process. The final render was done in parallel across 16 different cores via Amazon EC2 and took less than 10 hours. In the end, we increased the contrast just slightly to provide a more drastic effect. The change is subtle (see Figures 12(a) and 12(b)) but adds a lot to the final image.

7 Conclusion

7.1 Challenges

Most of the challenges encountered are discussed in the appropriate sections. One problem that most volume-centric scenes suffer from is huge rendering times from single-stepping for direct lighting and photon gathering. We found that the single-stepping was our biggest bottleneck and we had no feasible way to avoid this. Because of this, all of our test images were typically done with a small number of samples and/or at a low-resolution. While rendering the final image at higher resolution and samples per pixel, we saw severe aliasing artifacts in the volume that were not present in our test images. In order to meet the deadline, we had to quickly fix our code and resort to using Amazon's EC2 to render the final image. Originally, our final image render was scheduled to take over 48 hours using 4 available machines. With Amazon's EC2, the final render took just under 10 hours.

7.2 Division of Work

Sergey Levine worked on modeling the pillars, rendering the stars, and the acceleration structure for the animation. Edward Luong worked on the volumetric photon mapping integrator and setting up Amazon EC2. Both of us spent a considerable amount of time tweaking settings in the scene file to get the image just right.

References

Bajaj, C., Ihm, I., and Kang, B. 2005. Extending the photon mapping method for realistic rendering of hot gaseous fluids. *Computer Animation and Virtual Worlds 16*, 3–4.

Jensen, H. W., and Christensen, P. H. 1998. Efficient simulation of light transport in scenes with participating media using photon maps. In *Proceedings of SIGGRAPH 1998*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 311–320.

Magnor, M., Hildebrand, K., Lintu, A., and Hanson, A. 2005. Reflection nebula visualization. *Proc. IEEE Visualization 2005,* Minneapolis, USA (Oct.), 255–262.

Nadeau, D. R., Genetti, J. D., Napear, S., Pailthorpe, B., Emmart, C., Wesselak, E., and Davidson, D. 2001. Visualizing stars and emission nebulas. *Computer Graphics Forum 20*, 1, 27–33.