Appendix A: Volumetric Shader

The volumetric triangle shader is primarily implemented in two stages. First, a geometry shader constructs a screen-space bounding box containing all view rays intersecting each triangle. Second, a fragment shader computes the absorbance along the view ray intersecting each pixel and emits a color which will darken the pixel by the correct proportion. The vertex shader simply applies the model-view transformation and stores the position, $\boldsymbol{p}_i$, density, d_i, blurring radius, σ_{bi}, and thickness, σ_{ti} for each vertex.

Geometry Shader We first place a minimum bound of $\frac{1}{3}$ of a pixel radius on the blurring radius, σ_{bi}, and thickness, σ_{ti}, at each vertex. This avoids precision issues in the fragment shader while having little effect on the final image.

To construct the bounding region for each triangle, we surround each vertex with a bounding sphere of radius $\Lambda \max_i(\sigma_{bi}, \sigma_{ti})$ (where Λ is the cutoff described in section 5.3.2) and compute the square region in screen-space containing these bounding spheres. For each sphere, we construct a bounding cube such that the ray from the camera to the center of the sphere is perpendicular to the front face of the cube, guaranteeing that the entire volume of the cube is behind this face from the camera's perspective, and compute the screen-space coordinates of each corner of this front face.

We emit four vertices defining the screen-space bounding box containing all corners of the front face of each vertex's bounding cube.

As uninterpolated ("flat") outputs, we provide σ_{bi}, σ_{ti}, and the density, d_i, at each vertex as well as the position of the first vertex, $\boldsymbol{p}_1$, and the maximum blurring radius and thickness across the entire triangle, $\sigma_{b\max}$ and $\sigma_{t\max}$. To simplify the fragment shader, we compute a 2D *triangle space* with its origin located at $\boldsymbol{p}_1$, first axis containing $\boldsymbol{p}_2$, and second axis perpendicular to the first in the plane of the triangle. We provide $\hat{\boldsymbol{t}}$ and $\hat{\boldsymbol{u}}$ vectors to the fragment shader representing these axes and provide vertex coordinates, $\boldsymbol{p}'_i$ in the triangle space. (We only construct variables for the three non-zero coordinates.)

Fragment Shader We compute absorbance for each pixel by numerically integrating across the triangle surface using a sampling pattern like that shown in figure 9. In order to avoid visible patterns due to the method by which we compute the sampling region, we sample twice, with two independent random offsets, and average the results.

We first construct the normalized view ray $\hat{\boldsymbol{v}}$ based on the pixel coordinates, and begin by computing the region of the view ray, from $\hat{\boldsymbol{v}}t_+$ to $\hat{\boldsymbol{v}}t_-$, which lies within $\sigma_{t\max}$ distance of the triangle plane:

$$t_\pm = \frac{\boldsymbol{p}_0 \cdot (\hat{\boldsymbol{t}} \times \hat{\boldsymbol{u}}) \pm \sigma_{t\max}}{\hat{\boldsymbol{v}} \cdot (\hat{\boldsymbol{t}} \times \hat{\boldsymbol{u}})}. \tag{17}$$

We also compute $t_0 = \frac{\boldsymbol{p}_0 \cdot (\hat{\boldsymbol{t}} \times \hat{\boldsymbol{u}})}{\hat{\boldsymbol{v}} \cdot (\hat{\boldsymbol{t}} \times \hat{\boldsymbol{u}})}$, defining the intersection point between the view ray and the triangle.

From this point, we work entirely in the triangle space. The view ray in the triangle space is defined by its origin, $\boldsymbol{o}' = (-\boldsymbol{p}_1 \cdot \hat{\boldsymbol{t}}, -\boldsymbol{p}_1 \cdot \hat{\boldsymbol{u}})$ and direction, $\boldsymbol{v}' = (\hat{\boldsymbol{v}} \cdot \hat{\boldsymbol{t}}, \hat{\boldsymbol{v}} \cdot \hat{\boldsymbol{u}})$. Corresponding to the normalized $\hat{\boldsymbol{v}}'$ vector, we define $\hat{t}_X = \frac{t_X}{||\boldsymbol{v}'||}$. For convenience, we let $\boldsymbol{v}'(t) = \boldsymbol{o}' + \boldsymbol{v}'t$ define the triangle space view ray.

Our goal is to construct the smallest possible sampling region aligned to $\hat{\boldsymbol{v}}'$ containing the entire region of the triangle that falls within Λ standard deviations of the view ray in the smoothing distribution. We define $\hat{t}_{\min}$ and $\hat{t}_{\max}$ respectively to be the minimum and maximum of $\hat{t}_\pm$ and $\hat{t}_0 \pm \Lambda\sigma_{bmax}$, as a first guess of the extent of this region in the direction of $\hat{\boldsymbol{v}}'$. The second term here ensures that we always sample a sufficient range to capture smoothing due to σ_b. We further bound $\hat{t}_{\min}$ and $\hat{t}_{\max}$ by the maximum and minimum of $(\boldsymbol{p}'_i - \boldsymbol{o}') \cdot \hat{\boldsymbol{v}}'$ for all vertices i, clipping the region to the $\hat{\boldsymbol{v}}'$ aligned triangle bounding box.

Next, we bound in the direction of $\hat{\boldsymbol{v}}'^{\perp} = (-\hat{\boldsymbol{v}}'_y, \hat{\boldsymbol{v}}'_x)$. We take $\hat{t}^{\perp}_{\min} = -\Lambda\sigma_{b\max}$ and $\hat{t}^{\perp}_{\max} = \Lambda\sigma_{b\max}$ as an initial range containing all points within Λ standard deviations of the view ray along the triangle plane and further bound by the maximum and minimum of $(\boldsymbol{p}'_i - \boldsymbol{o}') \cdot \hat{\boldsymbol{v}}'^{\perp}$ for all vertices i, clipping the width of the sampling region to the triangle bounds.

Though it is possible to further bound the sampling region to the triangle edges rather than just the $(\hat{\boldsymbol{v}}', \hat{\boldsymbol{v}}'^{\perp})$ aligned triangle bounding box, the added complexity does not result in a substantial visual improvement when $\sigma_{b\max}$ is larger than the maximum triangle height, as it usually is in our system.

We discretize the sampling region according to a regular grid with a user-defined number of divisions in the direction of $\hat{\boldsymbol{v}}'$ and $\hat{\boldsymbol{v}}'^{\perp}$. (In practice, we found 8 and 5 divisions respectively to produce sufficiently smooth results.) We sample twice, with independent per-fragment random offsets in oder to improve quality.

For each sample point $\boldsymbol{p}'$, we first compute barycentric coordinates, ignoring the sample if it lies outside the triangle bounds, and then compute the values σ_b, σ_t, and d using those coordinates. We obtain an overall sample density by scaling the density at the sample point, d, by the area of each cell on the sample grid, a.

We now define $\boldsymbol{v}'_z = \boldsymbol{v} \cdot (\boldsymbol{t} \times \boldsymbol{u})$ and $\boldsymbol{o}'_z = -\boldsymbol{p}_1 \cdot (\boldsymbol{t} \times \boldsymbol{u})$, allowing us to integrate over the full 3D view ray in triangle space:

$$\begin{aligned} \alpha_{\boldsymbol{p}'} &= \int_{-\infty}^{\infty} \frac{d}{a} \frac{\exp\left[-\frac{1}{2}\left(\frac{||\boldsymbol{v}'(t) - \boldsymbol{p}'||}{\sigma_b^2} + \frac{\boldsymbol{v}'_z(t)}{\sigma_t^2}\right)\right]}{2\sigma_b^2\sigma_t\sqrt{2\pi^3}} \,\mathrm{d}t \\ &= \frac{d}{a} \frac{\exp\left[-\frac{1}{2}\left((\dot{\boldsymbol{o}} - \dot{\boldsymbol{p}}) \cdot (\dot{\boldsymbol{o}} - \dot{\boldsymbol{p}}) - \frac{(\dot{\boldsymbol{v}} \cdot (\dot{\boldsymbol{o}} - \dot{\boldsymbol{p}}))^2}{\dot{\boldsymbol{v}} \cdot \dot{\boldsymbol{v}}}\right)\right]}{2\pi\sigma_b^2\sigma_t||\dot{\boldsymbol{v}}||} \end{aligned} \tag{18}$$

where $\dot{\boldsymbol{v}} = (\frac{\boldsymbol{v}'}{\sigma_b}, \frac{\boldsymbol{v}'_z}{\sigma_t})$, $\dot{\boldsymbol{o}} = (\frac{\boldsymbol{o}'}{\sigma_b}, \frac{\boldsymbol{o}'_z}{\sigma_t})$, and $\dot{\boldsymbol{p}} = (\frac{\boldsymbol{p}'}{\sigma_b}, 0)$.

Though we leave the ray unbounded in both directions for simplicity, it is straightforward to compute the bounded integral using any standard computer algebraic system. By bounding the integral at the origin and the current pixel depth, it would be possible to correctly draw smoke intersecting opaque objects.

Note that in this formulation, the numerator of the exponent is simply $-\frac{1}{2}$ the squared distance between the scaled view ray $\dot{\boldsymbol{o}} + \dot{\boldsymbol{v}}t$ and sample point $\dot{\boldsymbol{p}}$. By scaling the triangle space by the blurring radius and thickness, we are able to integrate a spherical Gaussian of variance 1 and renormalize according to the original variances.

As with the thin-sheet renderer, the final output of the shader is the proportion of light absorbed along the view ray due to the triangle. In this case, we must sum over all samples, and divide by two since we sample the region twice:

$$\exp\left(-\frac{\sum \alpha_{\boldsymbol{p}'}}{2}\right). \tag{19}$$