CHAPTER 4

A Planetarium Dome Master Camera

John E. Stone
Beckman Institute for Advanced Science and Technology,
University of Illinois at Urbana-Champaign

ABSTRACT

This chapter presents a camera implementation for high-quality interactive ray tracing of planetarium dome master images using an azimuthal equidistant projection. Ray tracing is aptly suited for implementing a wide variety of special panoramic and stereoscopic projections without sacrificing image quality. This camera implementation supports antialiasing, depth of field focal blur, and circular stereoscopic projections, all effects that are difficult to produce with high quality using conventional rasterization and image warping.

4.1 INTRODUCTION

Planetarium dome master images encode a 180° hemispherical field of view within a black square, with an inscribed circular image containing the entire field of view for projection onto a planetarium dome. Dome master images are produced using a so-called azimuthal equidistant projection and closely match the output of a real-world 180° equidistant fisheye lens, but without a real lens' imperfections and optical aberrations. There are many ways of creating dome master projections using rasterization and image warping techniques, but direct ray tracing has particular advantages over other alternatives: uniform sample density in the final dome master image (no samples are wasted in oversampled areas as when warping cubic projections or many planar perspective projections [3]), support for stereoscopic rendering, and support for depth of field on an intrinsically curved focal surface. By integrating interactive progressive ray tracing of dome master images within scientific visualization software, a much broader range of scientific visualization material can be made available in public fulldome projection venues [1, 5, 7].

E. Haines, T. Akenine-Möller (eds.), *Ray Tracing Gems*, https://doi.org/10.1007/978-1-4842-4427-2_4

4.2 METHODS

Dome master images are formed using an *azimuthal equidistant projection*, as illustrated in Figure 4-1. The dome master image is normally computed within a square viewport, rendered with a 180° field of view filling a circle inscribed in the square viewport. The inscribed circle just touches the edges of the viewport, with a black background everywhere else. The dome master projection may appear roughly similar to an orthographic projection of the dome hemisphere as seen from above or below, but with the critical difference that the spacing between rings of latitude in the dome master image are uniform. This uniform spacing conveniently allows a ray tracer camera to employ uniform sampling in the image plane. Figure 4-2 shows the relationship between locations in the image plane and their resulting ray directions on the dome hemisphere. Figure 4-3 shows an example sequence of ray traced dome master images produced using the camera model described here.

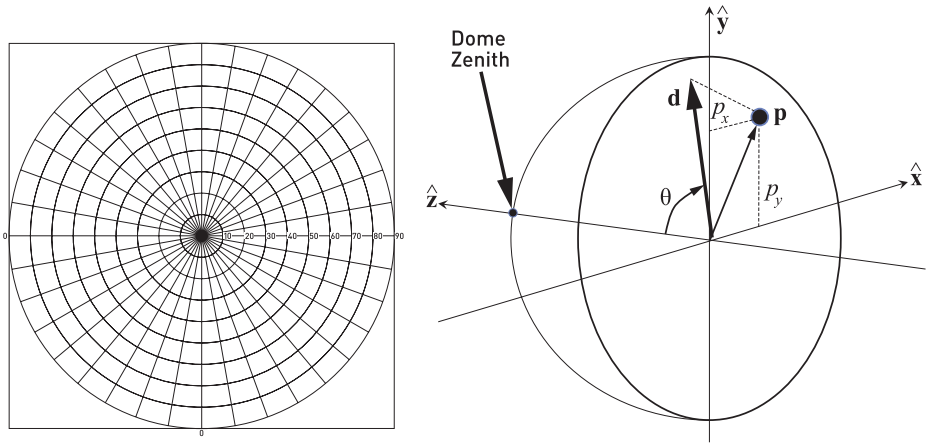

Figure 4-1. *Dome master images use an azimuthal equidistant projection and appear similar to a photograph from a 180° fisheye lens. Left: the dome master image has visibly uniform spacing of latitude (circles) and longitude (lines) drawn at 10 intervals for the projected 180° field of view. A pixel's distance to the viewport center is proportional to the true angle in the center of the projection. Right: the vector p in the dome master image plane, the azimuth direction components p_x and p_y, the ray direction $\hat{\boldsymbol{d}}$, the angle θ between the ray direction $\hat{\boldsymbol{d}}$ and the dome zenith, and the camera orthogonal basis vectors $\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{y}}$, and $\hat{\boldsymbol{z}}$.*

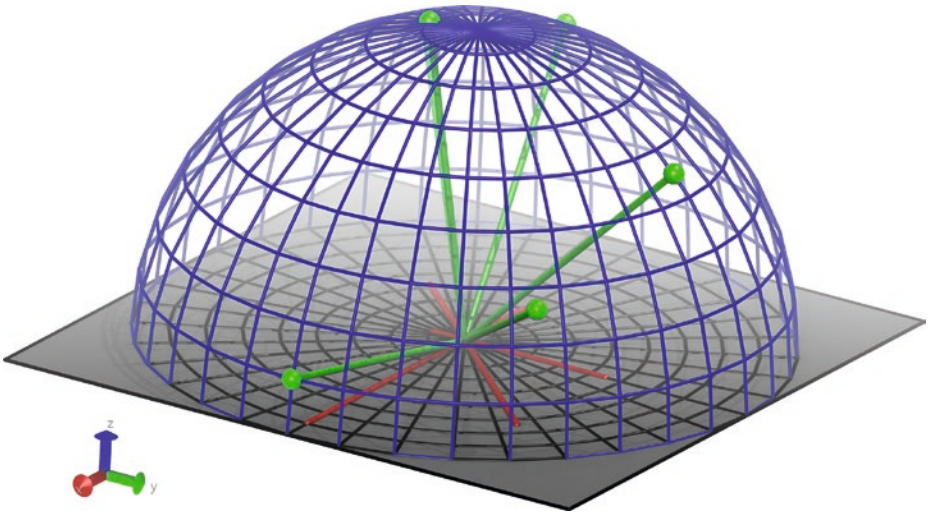

Figure 4-2. *A visual depiction relating the image plane (gray square with inscribed latitude/longitude lines), dome hemisphere (blue), example* ***p*** *vectors (red) in the image plane, and corresponding ray directions on the dome surface (green).*

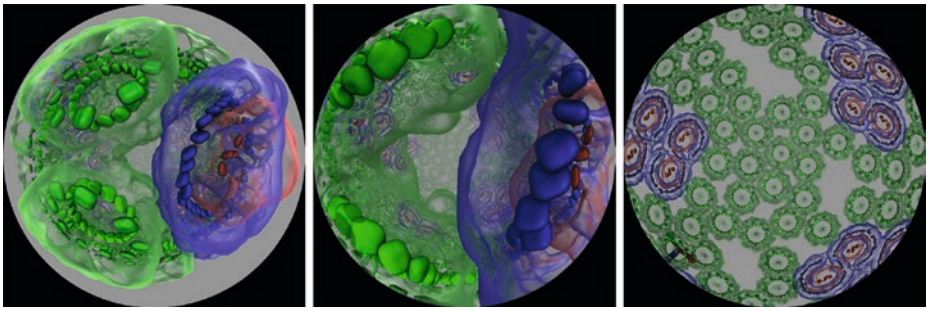

Figure 4-3. *A sequence of dome master images interactively rendered in Visual Molecular Dynamics (VMD) [4, 7] with OptiX. The sequence shows the camera flying into a photosynthetic vesicle found in a purple bacterium. Since the structure is predominantly spherical, when the camera reaches the vesicle center, the dome projection appears flat in the rightmost image.*

4.2.1 COMPUTING RAY DIRECTIONS FROM VIEWPORT COORDINATES

The dome master camera computes the primary ray directions in a few key steps. The maximum field of view angle from the center θ_{max} is computed as half of the overall field of view, e.g., for the typical 180° field of view, θ_{max} is 90° or $\frac{\pi}{2}$ radians. For the azimuthal equidistant projection, the distance from each pixel to the center of the viewport is proportional to the true angle from the center of the projection in radians. Dome master images are normally square, so for a 4096 × 4096 dome image with a 180° field of view, we would have a radian/pixel scaling factor of $\frac{\pi}{4096}$ in both dimensions. For each pixel in the image plane, a distance is computed

between the pixel I and the midpoint M of the viewport and then multiplied by a field of view radian/pixel scaling factor, yielding a two-dimensional vector in units of radians, $\mathbf{p} = (p_x, p_y)$. The length $\|\mathbf{p}\|$ is then computed from p_x and p_y, the two distance components from the viewport center, yielding θ, the true angle from the dome zenith, in radians. The key steps for calculating θ are then

$$\mathbf{p} = (I - M)\frac{\pi}{4096} \tag{1}$$

and

$$\theta = \|\mathbf{p}\|. \tag{2}$$

For a dome master with a 180° field of view, the angle θ is complementary to the elevation angle of the ray computed from $\mathbf{p}$.

It is important to note that θ is used both as a distance (from the center of the viewport, scaled by radian/pixel) and as an angle (from the dome zenith). To calculate the azimuthal direction components of the ray, we compute $\hat{\mathbf{p}}$ from $\mathbf{p}$ by dividing by θ, used here as a length. For $\theta = 0$, the primary ray points at the zenith of the dome, and the azimuth angle is undefined, so we protect against division by zero in that case. If θ is greater than θ_{max}, then the pixel is outside of the field of view of the dome and is colored black. For θ values between zero and θ_{max}, the normalized ray direction in dome coordinates is

$$\hat{\mathbf{n}} = \left(\frac{p_x \sin\theta}{\theta}, \frac{p_y \sin\theta}{\theta}, \cos\theta\right). \tag{3}$$

If orthogonal up ($\hat{\mathbf{u}}$) and right ($\hat{\mathbf{r}}$) directions are required for each ray, e.g., for depth of field, they can be determined inexpensively using existing intermediate values. The up direction can be computed by negating the ray direction's derivative as a function of θ, yielding a unit vector aligned with the vertical lines of longitude pointing toward the dome zenith,

$$\hat{\mathbf{u}} = \left(\frac{-p_x \cos\theta}{\theta}, \frac{-p_y \cos\theta}{\theta}, \sin\theta\right). \tag{4}$$

The right direction can be determined purely from the azimuth direction components p_x and p_y, yielding a unit vector aligned with the horizontal latitude lines,

$$\hat{\mathbf{r}} = \left(\frac{-p_y}{\theta}, \frac{p_x}{\theta}, 0\right). \tag{5}$$

See Listing 4-1 for a minimalistic example computing the ray, up, and right directions in the dome coordinate system. Finally, to convert the ray direction from dome coordinates to world coordinates, we project its components onto the camera's orthogonal orientation basis vectors $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$ by

$$\hat{\mathbf{d}} = \left(n_x\hat{\mathbf{x}} + n_y\hat{\mathbf{y}} + n_z\hat{\mathbf{z}}\right). \tag{6}$$

The same coordinate system conversion operations must also be performed on the up and right vectors if they are required.

Listing 4-1. *This short example function illustrates the key arithmetic required to compute a ray direction from the floor of the dome hemisphere from a point in the image plane, given a user-specified angular field of view (normally 180°) and viewport size. The dome angle from the center of the projection is proportional to the distance from the center of the viewport to the specified point in the image plane. This function is written for a dome hemisphere with the zenith in the positive* z*-direction. The ray direction returned by this function must be projected onto camera basis vectors by the code calling this function*

```
static __device__ __inline__
int dome_ray(float fov,             // FoV in radians
             float2 vp_sz,          // viewport size
             float2 i,              // pixel/point in image plane
             float3 &raydir,        // returned ray direction
             float3 &updir,         // up, aligned w/ longitude line
             float3 & rightdir) {   // right, aligned w/ latitude line
  float thetamax = 0.5f * fov;      // half-FoV in radians
  float2 radperpix = fov / vp_sz;   // calc radians/pixel in X/Y
  float2 m = vp_sz * 0.5f;          // calc viewport center/midpoint
  float2 p = (i - m) * radperpix;   // calc azimuth, theta components
  float theta = hypotf(p.x, p.y);   // hypotf() ensures best accuracy
  if (theta < thetamax) {
    if (theta == 0) {
      // At the dome center, azimuth is undefined and we must avoid
      // division by zero, so we set the ray direction to the zenith
      raydir = make_float3(0, 0, 1);
      updir = make_float3(0, 1, 0);
      rightdir = make_float3(1, 0, 0);
    } else {
      // Normal case: calc+combine azimuth and elevation components
      float sintheta, costheta;
      sincosf(theta, &sintheta, &costheta);
      raydir     = make_float3(sintheta * p.x / theta,
                               sintheta * p.y / theta,
                               costheta);
      updir      = make_float3(-costheta * p.x / theta,
                               -costheta * p.y / theta,
                               sintheta);
      rightdir   = make_float3(p.y / theta, -p.x / theta, 0);
    }

```

```
33      return 1; // Point in image plane is within FoV
34    }
35
36    raydir = make_float3(0, 0, 0); // outside of FoV
37    updir = rightdir = raydir;
38    return 0; // Point in image plane is outside FoV
39  }
```

4.2.2 CIRCULAR STEREOSCOPIC PROJECTION

The nonplanar panoramic nature of the dome projection focal surface presents a special challenge for stereoscopic rendering. While non-stereoscopic dome master images can be synthesized through multistage rendering, warping, and filtering of many conventional perspective projections, high-quality stereoscopic output essentially requires a separate stereoscopic camera calculation for every sample in the image (and thus per ray, when ray tracing). This incurs significant performance overheads and image quality trade-offs using existing rasterization APIs, but it is ideally suited for interactive ray tracing. The mathematics naturally extend the ray computations outlined in the previous section and introduce insignificant performance cost relative to rendering a pair of monoscopic images.

To use stereoscopic circular projection [2, 6, 8] with a dome master camera, each ray's origin is shifted left or right by half of the interocular distance. The shift occurs along the stereoscopic interocular axis, which lies perpendicular to both the ray direction ($\hat{\mathbf{d}}$) and the audience's local zenith or "up" direction ($\hat{\mathbf{q}}$). This accounts for various tilted dome configurations, including those shown in Figure 4-4. The shifted ray origin is computed by $O = O + e\left(\hat{\mathbf{d}} \times \hat{\mathbf{q}}\right)$, where $\mathbf{e}()$ is an eye-shift function that applies the shift direction and scaling factors to correctly move the world-space eye location, as shown in Figure 4-5. By computing the stereoscopic eye shift independently for each ray, we obtain a circular stereoscopic projection.

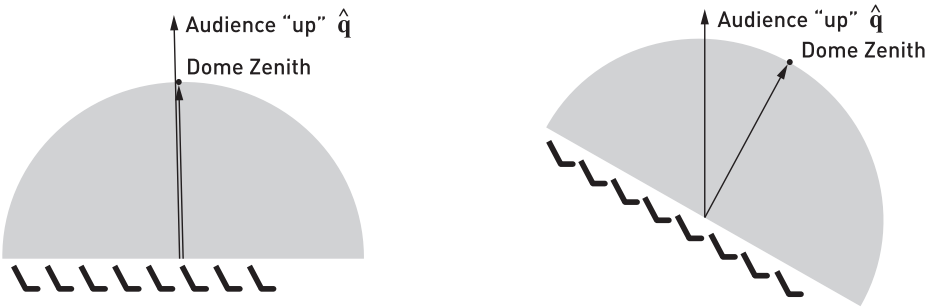

Figure 4-4. *Relation between the dome zenith and audience "up" direction $\hat{\mathbf{q}}$ in both a traditional flat planetarium dome (left) and a more modern dome theater with 30 tilt and stadium style seating (right).*

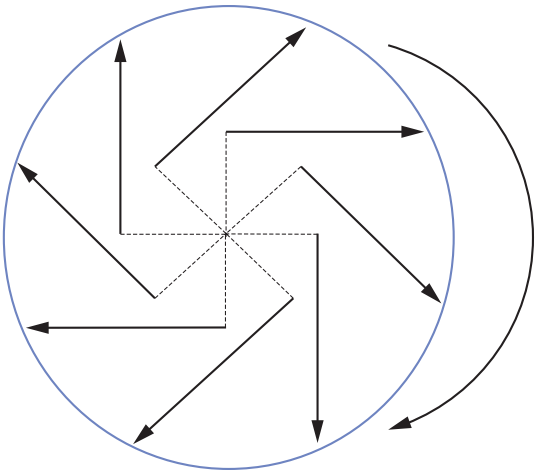

Figure 4-5. *Illustration of the circular stereoscopic projection technique and the effect of applying an eye offset of half of the interocular distance to each ray's origin, according to the ray direction. The drawing shows the eye-shift offsets (dotted lines) for the left eye projection.*

While circular stereoscopic projections are not entirely distortion-free, they are "always correct where you are looking" [6]. Circular stereoscopic projections are most correct when viewers look toward the horizon of the stereoscopic projection, but not when looking near the audience zenith ($\hat{\mathbf{q}}$). Viewers could see *backward-stereo* images when the region behind the stereoscopic polar axis is visible. To help mitigate this problem, the stereoscopic eye separation can be modulated as a function of the angle of elevation of $\hat{\mathbf{d}}$ relative to the audience's stereoscopic equator or horizon line. By modulating the eye separation distance to zero at the audience's zenith (thereby degrading to a monoscopic projection), the propensity for backward-stereo viewing can be largely eliminated. See Listing 4-2 for a simple but representative example implementation.

Listing 4-2. *A minimal eyeshift function implementation that handles both stereoscopic and monoscopic projections.*

```
1 static __host__ __device__ __inline__
2 float3 eyeshift(float3 ray_origin,   // original non-stereo eye origin
3                  float eyesep,       // interocular dist, world coords
4                  int whicheye,       // left/right eye flag
5                  float3 DcrossQ) {   // ray dir x audience "up" dir
```

```
 6   float shift = 0.0;
 7   switch (whicheye) {
 8     case LEFTEYE :
 9       shift = -0.5f * eyesep; // shift ray origin left
10       break;
11
12     case RIGHTEYE:
13       shift = 0.5f * eyesep; // shift ray origin right
14       break;
15
16     case NOSTEREO:
17          default:
18       shift = 0.0; // monoscopic projection
19       break;
20   }
21
22   return ray_origin + shift * DcrossQ;
23 }
```

Stereoscopic dome master images are computed in a single pass, by rendering both stereoscopic sub-images into the same output buffer in an over/under layout with the left eye sub-image in the top half of a double-height framebuffer and the right eye sub-image in the lower half. Figure 4-6 shows the over/under vertically stacked stereoscopic framebuffer layout. This approach aggregates the maximal amount of data-parallel ray tracing work in each frame, thereby reducing API overheads and increasing hardware scheduling efficiency. Existing hardware-accelerated ray tracing frameworks lack efficient mechanisms to perform progressive ray tracing on lists of cameras and output buffers, so the packed stereo camera implementation makes it possible to much more easily employ progressive rendering for interactive stereoscopic dome visualizations. This is particularly beneficial when using video streaming techniques to view live results from remotely located, cloud-hosted rendering engines. A key benefit of the vertically stacked stereoscopic sub-image layout is that any image post-processing or display software can trivially access the two stereoscopic sub-images independently of each other with simple pointer offset arithmetic because they are contiguous in memory. Dome master images and movies produced with circular stereoscopic projections can often be imported directly into conventional image and video editing software. Most basic editing and post-processing can be performed using the same tools that one would use for conventional perspective projections.

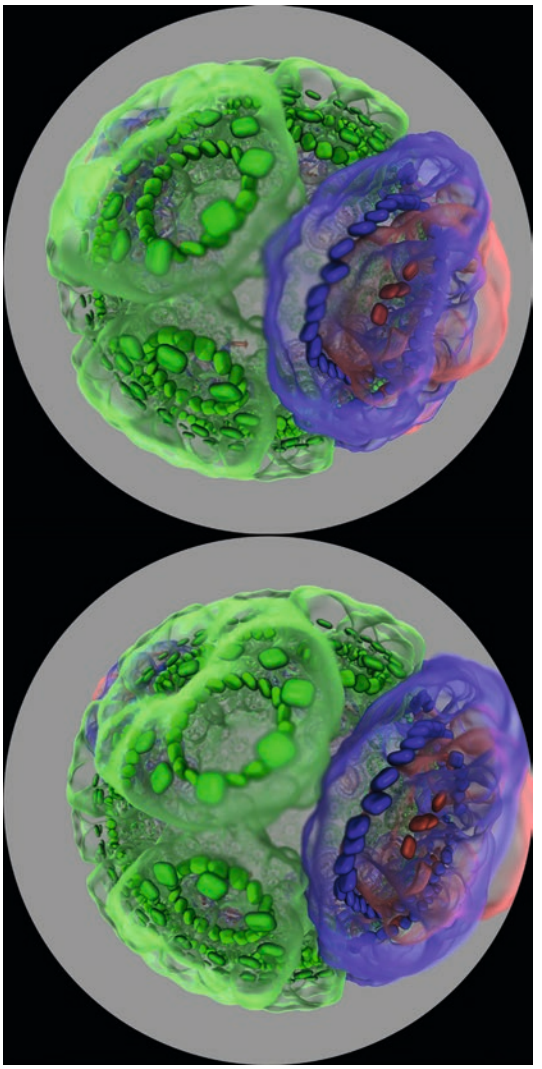

Figure 4-6. *A vertically stacked stereoscopic pair of dome master images rendered in a single pass, with depth of field applied on the spherical focal plane.*

4.2.3 DEPTH OF FIELD

Depth of field focal blur can be implemented for the dome master projection by computing basis vectors for a depth of field circle of confusion disk, and subsequently using the basis vectors to compute jittered ray origin offsets and, finally, updated ray directions. The circle of confusion basis vectors $\hat{\mathbf{u}}$ and $\hat{\mathbf{r}}$ are best computed along with the ray direction $\hat{\mathbf{d}}$ as they all depend on the same intermediate values. Equations 4 and 5 describe the calculation of $\hat{\mathbf{u}}$ and $\hat{\mathbf{r}}$,

respectively. Once the jittered depth of field ray origin is computed using $\hat{\mathbf{u}}$ and $\hat{\mathbf{r}}$, the ray direction must be updated. The updated ray direction is calculated by subtracting the new ray origin from the point where the ray intersects the focal surface (a sphere in this case) and normalizing the result. See Listing 4-3 for a simple example implementation.

Listing 4-3. *This short example function illustrates the key arithmetic required to compute the new ray origin and direction when depth of field is used.*

```
// CUDA device function for computing a new ray origin and
// ray direction, given the radius of the circle of confusion disk,
// orthogonal "up" and "right" basis vectors for each ray,
// focal plane/sphere distance, and a RNG/QRNG seed/state vector.
static __device__ __inline__
void dof_ray(const float3 &ray_org_orig, float3 &ray_org,
             const float3 &ray_dir_orig, float3 &ray_dir,
             const float3 &up, const float3 &right,
             unsigned int &randseed) {
  float3 focuspoint = ray_org_orig +
                      (ray_dir_orig * cam_dof_focal_dist);
  float2 dofjxy;
  jitter_disc2f(randseed, dofjxy, cam_dof_aperture_rad);
  ray_org = ray_org_orig + dofjxy.x*right + dofjxy.y*up;
  ray_dir = normalize(focuspoint - ray_org);
}
```

4.2.4 ANTIALIASING

Antialiasing of the dome master image is easily accomplished without any unusual considerations, by jittering the viewport coordinates for successive samples. For interactive ray tracing, a simple box-filtered average over samples is inexpensive and easy to implement. Since samples outside of the field of view are colored black, antialiasing samples also serve to provide a smooth edge on the circular image produced in the dome master image.

4.3 PLANETARIUM DOME MASTER PROJECTION SAMPLE CODE

The example source code provided for this chapter is written for the NVIDIA OptiX API, which uses the CUDA GPU programming language. Although the sample source code is left abridged for simplicity, the key global-scope camera and scene parameters are shown using small helper functions, e.g., for computing depth of field, generating uniform random samples on a disk, and similar tasks. These are provided so that the reader can more easily interpret and adapt the sample implementation for their own needs.

The dome master camera is implemented as a templated camera function, to be instantiated in several primary ray generation "programs" for the OptiX ray tracing framework. The function accepts STEREO_ON and DOF_ON template parameters that either enable or disable generation of a stereoscopic dome master image and depth of field focal blur, respectively. By creating separate instantiations of the camera function, arithmetic operations associated with disabled features are eliminated, which is particularly beneficial for high-resolution interactive ray tracing of complex scenes.

ACKNOWLEDGMENTS

This work was supported in part by the National Institutes of Health, under grant P41-GM104601; the NCSA Advanced Visualization Laboratory; and the CADENS project supported in part by NSF award ACI-1445176.

REFERENCES

[1] Borkiewicz, K., Christensen, A. J., and Stone, J. E. Communicating Science Through Visualization in an Age of Alternative Facts. In *ACM SIGGRAPH Courses* (2017), pp. 8:1–8:204.

[2] Bourke, P. Synthetic Stereoscopic Panoramic Images. In *Interactive Technologies and Sociotechnical Systems*, H. Zha, Z. Pan, H. Thwaites, A. Addison, and M. Forte, Eds., vol. 4270 of *Lecture Notes in Computer Science*. Springer, 2006, pp. 147–155.

[3] Greene, N., and Heckbert, P. S. Creating Raster Omnimax Images from Multiple Perspective Views Using the Elliptical Weighted Average Filter. *IEEE Computer Graphics and Applications 6*, 6 (June 1986), 21–27.

[4] Humphrey, W., Dalke, A., and Schulten, K. VMD—Visual Molecular Dynamics. *Journal of Molecular Graphics 14*, 1 (1996), 33–38.

[5] Sener, M., Stone, J. E., Barragan, A., Singharoy, A., Teo, I., Vandivort, K. L., Isralewitz, B., Liu, B., Goh, B. C., Phillips, J. C., Kourkoutis, L. F., Hunter, C. N., and Schulten, K. Visualization of Energy Conversion Processes in a Light Harvesting Organelle at Atomic Detail. In *International Conference on High Performance Computing, Networking, Storage and Analysis* (2014).

[6] Simon, A., Smith, R. C., and Pawlicki, R. R. Omnistereo for Panoramic Virtual Environment Display Systems. In *IEEE Virtual Reality* (March 2004), pp. 67–73.

[7] Stone, J. E., Sener, M., Vandivort, K. L., Barragan, A., Singharoy, A., Teo, I., Ribeiro, J. V., Isralewitz, B., Liu, B., Goh, B. C., Phillips, J. C., MacGregor-Chatwin, C., Johnson, M. P., Kourkoutis, L. F., Hunter, C. N., and Schulten, K. Atomic Detail Visualization of Photosynthetic Membranes with GPU-Accelerated Ray Tracing. *Parallel Computing 55* (2016), 17–27.

[8] Stone, J. E., Sherman, W. R., and Schulten, K. Immersive Molecular Visualization with Omnidirectional Stereoscopic Ray Tracing and Remote Rendering. In *IEEE International Parallel and Distributed Processing Symposium Workshop* (2016), pp. 1048–1057.