Andrew Glassner's Notebook

http://www.research.microsoft.com/research/graphics/glassner/

Computer-Generated Solar Halos and Sun Dogs

Andrew Glassner

Microsoft Research

In the January column, I described Greenler's technique for creating dot patterns that approximate solar phenomena such as sun dogs and solar halos. I closed by promising to extend the simulation to include energy considerations, smooth images, and color.

Here's a summary of the basic process for creating the dot patterns (for details and figures refer to the January column). We start by imagining a perfectly hexagonal ice crystal, which may be thin compared with its radius (in which case we call it a plate) or long and skinny (a pencil). To create a solar display, we first find an orientation for the crystal. If there's not much wind, each crystal will tend to fall with one of its biggest faces parallel to the ground. Wind can cause the crystals to tumble as they fall, which we simulate by adding some amount of random rotation.

Once we have rotated the crystal into the chosen orientation, we send a ray from the infinitely distant sun toward the crystal. If the ray hits the crystal, we follow its refractive path through the ice and find its outgoing direction as it leaves the crystal. To see that light, we must be looking in its direction back at the crystal, so we add some light from that direction into our image. I simply quantize the direction to the nearest pixel and color in that pixel. Then we pick a new orientation and trace a new ray, repeating the process over and over. The result is a cloud of dots. By following different paths through the crystals, constraining the possible orientations, and changing the crystal's length, we can simulate a wide variety of solar phenomena.

Energized by old friends

The first thing we'll do to enhance this model is include energy effects, which we will handle very casually. There are lots of places where the light ray can pick up and lose energy, but we'll focus just on the crystal faces. When the ray strikes the first face, it represents a whole beam of light. Thus we need to apply Lambert's Law, diminishing the energy by the cosine of the angle the ray makes with the face normal; this is our old friend from polygonal shading. Similarly, we should apply Fresnel's Law, which accounts for the fact that glancing rays are generally reflected, while rays more perpendicular to the face are better able to penetrate the crystal. Both of these are standard shading models in computer graphics. Now when we draw a dot in our image, we can draw it with a color value indicating the amount of energy it carries.

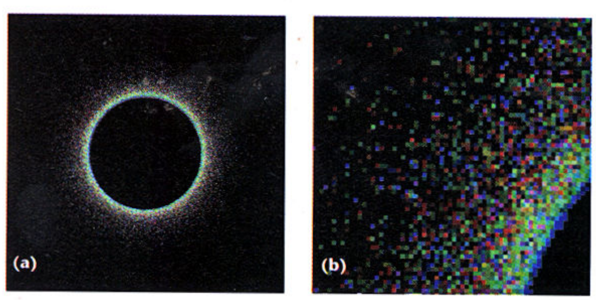
The pursuit of smooth color

My process of generating smooth color images involved some trial and error. In a technical paper, people generally present only the answers that worked and none of the dead ends. I thought it might be interesting to retrace the path I took on my way to creating satisfactory images.

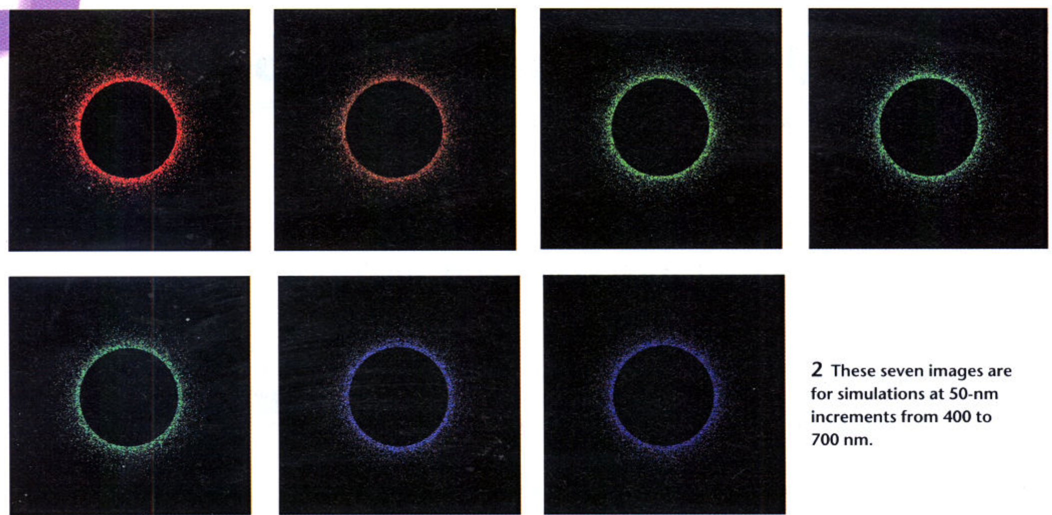
My first extension to the basic algorithm was to include color. My initial approach was to simply pick a different wavelength for each ray of light, follow it through the crystal, and plot it with the right color on the screen. But after throwing lots of rays, I had a very speckled image of all different colors, as shown in Figure 1 for the 22-degree halo. Tracing more rays just changed the speckle pattern, since new rays overwrote the old ones—the image itself didn't get any smoother.

I decided instead to run several different simulations, each at its own wavelength, and then combine them. I used seven wavelengths from 400 to 700 nanometers, evenly spaced 50 nm apart. The results are shown in Figure 2 on the next page. Figure 3 shows a close-up view of just the 400- and 700-nm simulations, where you can see the inner ring of red generated by the change in the index of refraction with wavelength.

To compute the index of refraction η as a function of wavelength λ, I went to the literature. I found a table for the index of refraction of ice at three different wavelengths for temperatures from 0 to 100 degrees Celsius in 10-degree increments. A plot of this reference data is



1 (a) A dot pattern generated using uniformly distributed random wavelengths in a single simulation. (b) Close-up view.



2 **These seven images are for simulations at 50-nm increments from 400 to 700 nm.**

shown in Figure 4 (left). My favorite formula for approximating η as a function of wavelength λ is Sellmeier's formula, which in two-term form is $\eta(\lambda) = A + B/\lambda^2$ (you can add more terms for more accuracy, but I found that the two-term form fit the data with excellent precision). Since the table gave values at different temperatures, I made the coefficients in Sellmeier's formula temperature-dependent:

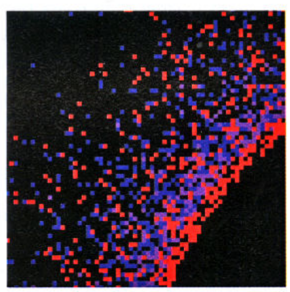
$$\eta(\lambda, T) = A(T) + B(T)/\lambda^2$$

I used a symbolic algebra program to compute A and B at each of the 11 temperatures, so now I had 11 samples of the functions $A(T)$ and $B(T)$. I found that quadratic polynomials matched both functions very well; these polynomials are

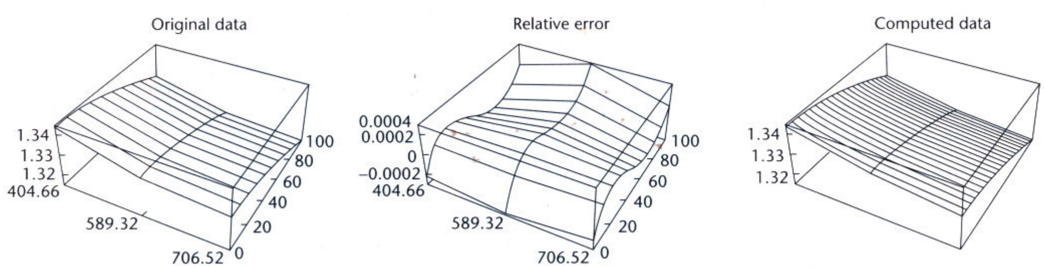
$$A(T) = 1.32491 - 0.0000399278T - 0.000001206787T^2$$
$$B(T) = 3105.31 + 1.25203T - 0.0353608T^2$$

where T is the temperature in degrees Celsius.

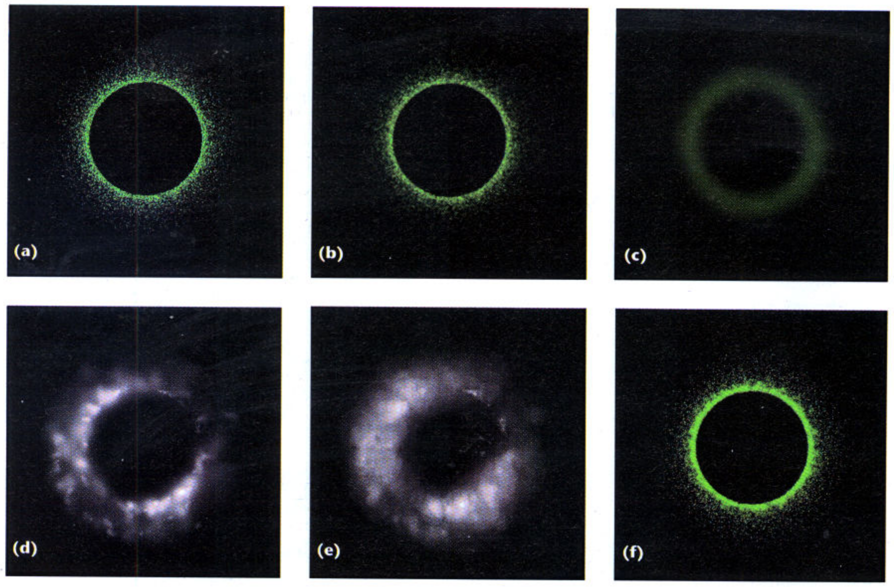
So now to find η at any wavelength and temperature, I compute $A(T)$ and $B(T)$, and plug that into Sellmeier's formula along with the wavelength in nanometers. The maximum absolute and relative errors in this approximation are both less than 6 parts in 10,000. The relative error is shown in Figure 4 (center), and the values generated by the approximation are plotted in Figure 4 (right). It's winter now in Seattle, so I decided to send my simulations on vacation. All of the figures in this month's column were made at a balmy 30 degrees Celsius.



3 **A close-up of the overlaid upper-left corner of the 400- and 700-nm simulations in Figure 2. Note that the red dots appear inside the blue ones, and the blue dots predominate at the outside.**



4 **Reference data for the index of refraction of ice (left), relative error in the approximation used here (center), and sampling of the approximation function (right).**



5 First attempts at smoothing. (a) Original dot pattern at 550 nm. (b) Small blurring of (a) still looks speckled; (c) large blurring looks too fuzzy; (d) using Gaussian blobs produces a splotchy image; (e) using disks instead isn't much better; (f) compositing (b) with (a), using the intensity of (b) as a matte, makes the inner edge too blurry and the outer parts too speckled.

Scanned image, scant results

I first tried to combine the color runs by simply adding the images together. I hoped that the dense regions on all plots would sum to white and that smooth regions would emerge where the different colors faded out. Unfortunately, the result looked just like Figure 1—speckles. I realized that the density plots had to be smoothed before they could be combined.

My impulse was to abandon the dot patterns and generate a smooth picture directly by inverting the image-making process. Rather than orienting the crystal and then tracing rays, I would scan the screen and find the probability of a crystal being in the right orientation to send light through each pixel in the image. This sounded like a simple enough problem: the crystal is rigid, the geometry of refraction is well known, and everything should be straightforward. Indeed, it is straightforward to a point, but things get very messy very fast.

Just setting up the equations is complicated because of all the trig functions involved. Then I realized that in fact there was no single solution. Lots of different crystal orientations can send rays back in any given direction. Just as a quick test I selected one pixel and ran the simulator. Dozens of different sets of angles sent light back in almost the same direction, passing through the same pixel. I decided to stick with the dot patterns and try to smooth them out.

Blurs and blobs

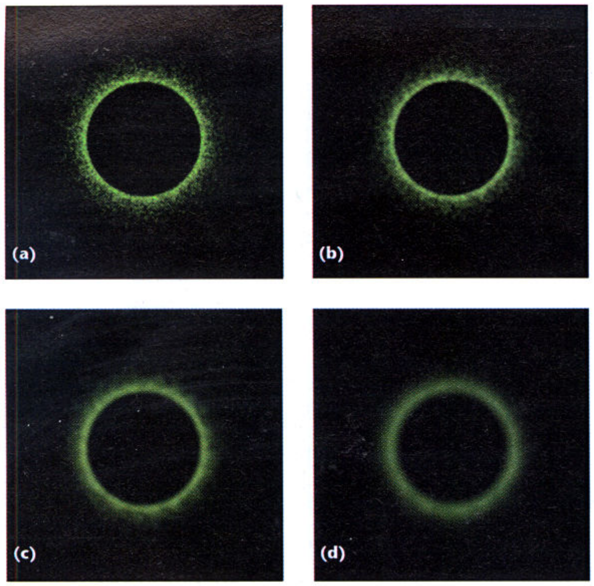
Looking at the dot pattern of Figure 5a, my first thought was to blur the density plot. But no value of blur worked well. The small blur of Figure 5b didn't get the dots to join up and form a smooth field, and the large blur of Figure 5c made the whole picture go fuzzy. Next, I drew a Gaussian blob over each spot, like the splatting technique used in volume visualization. I used the distance from each ray to its nearest neighbor to determine the blob's radius, and scaled the blob's height by the amount of energy carried by the ray. The radius was set so the blob was at half-height at its nearest neighbor.

Finding the nearest neighbor involved saving all the ray locations (at subpixel accuracy) and then searching for neighbors for each ray. To speed the search, I built a data structure of overlapping rectangles in the image; then for each ray I searched only the other rays in its rectangle. Because the rectangles overlapped, a ray could belong to more than one such rectangle; so I avoided the problem where two nearby rays straddle a boundary between rectangles and don't see each other. When the image was built up, I normalized the pixel values to use the whole display range.

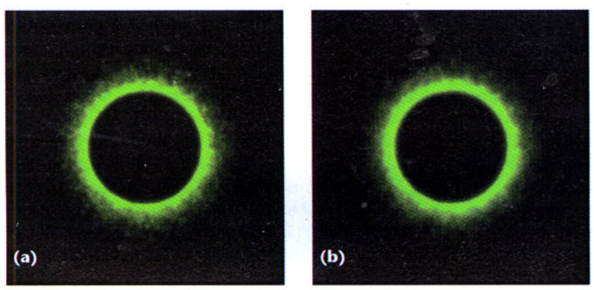
Figure 5d shows the result of the blobs. The bright spots come from places where lots of rays just happened to land on top of one another (this artifact isn't visible in Figure 5a because all of these rays land on the same pixel). This result was disappointing not only because it looked so bad, but because it required a whole lot of additional computation and storage. I thought maybe if I replaced the blobs by flat disks it would look a bit better; but as you can see from Figure 5e, this wasn't a success either.

Fuzzy thinking

The bridge to a better answer came from looking at the blurry picture and the sharp picture together. I thought that if I could get the blurry picture to show up where the sharp picture was black, I'd get a smooth field. So I turned the blurry picture into its own matte and composited it with the sharp picture. Where the blurry picture was bright, it dominated the result. As you can see in Figure 5f, this method smoothed out the inner



6 (a) Figure 5a filtered from 400 to 300 pixels on a side, then filtered back up to 400. (b) Figure 6a filtered down to 200 pixels on a side, then filtered back up to 400. (c) Figure 6b filtered down to 100 pixels on a side, then back up to 400. (d) Figure 6c filtered down to 50 pixels, then back up to 400.



7 (a) Equally weighted average of Figure 6b, 6c, and 6d. (b) A better average using one part of Figure 6b, two parts of 6c, and three parts of 6d.

ring, but retained the unwelcome bright dots around the outside.

While looking at this picture, I realized that I could probably get what I wanted by combining several images, each blurred by different amounts. So I created the images shown in Figure 6. To get the blur, I scaled the original image down by increasing amounts, then blew the results back up. My original dot pattern was 400 pixels on a side. I applied a Gaussian filter kernel with a radius of about 1.3 pixels to make a 300-pixel image, then enlarged it back to 400. Then I applied a filter of radius 1.5 to that to make a 200-pixel image, which I filtered back up to 400. I filtered that image down to 100 pixels, and filtered it back up to 400 pixels. Finally, I took that result, reduced it to 50 pixels on a side, and brought it back up to 400. Note that the blurriest image in Figure 6d is the result of eight filters applied in sequence (four reducing and four enlarging), not just a single blur to the original. When all the images were back up to 400 pixels on a side, I renormalized them to use the full range of image intensities.

At first I simply added the three blurriest pictures together equally, as in Figure 7a. That still showed a little too much of the speckling for my taste. So I tried again, this time changing the recipe to use three parts of the 50-pixel image, two parts of the 100-pixel image, and one part of the 200-pixel image. Figure 7b shows the result. The inner edge is still relatively sharp, and the outer edges fade out nicely.

Nothing new under the sun

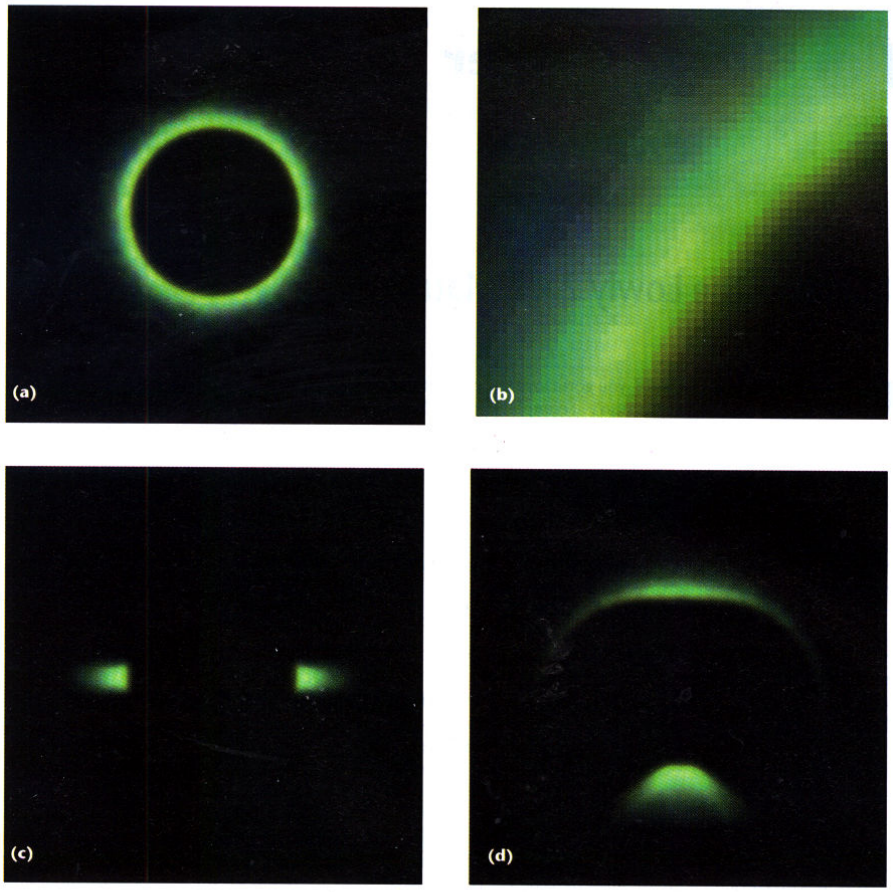
As soon as I made these images, I recognized this process as multiresolution compositing, which is a standard technique in image processing and computer graphics production. I wished I had thought of it before trying all the other approaches, but I did enjoy playing around with this problem looking for a good answer. This approach also has a big efficiency advantage over the Gaussian blob-type solutions, because it's simple image processing and doesn't require lots of additional data structures and processing. Just draw the dot pictures, blur them out, and add them up.

I applied the multiresolution blur and compositing to each of the dot patterns in Figure 2, and generated the composite 22-degree halo shown in Figure 8a. It's a pretty good match to the photographs shown in last month's column. Figure 8b shows a close-up of the inner edge; it has a satisfyingly red tint. I ran through the same process for sun dogs, as shown in Figure 8c. Note that the sun dogs aren't just slices of the halo; their color fringes have a subtly different shape. The upper and lower tangent arcs for a sun elevation of 30 degrees are shown in Figure 8d.

The movie

I made a movie of the upper and lower tangent arcs for the rising sun from 0 to 90 degrees. You can find it on my Web page at http://www.research.microsoft.com/research/graphics/glassner/ or on the *CG&A* Web page at http://www.computer.org/. I will also maintain a listing of notes and errata for these columns on those pages.

There's a lot more going on up in the sky than I've talked about in these two columns. I hope to return to the topic again sometime. One thing we haven't addressed is what happens to light that reflects off the crystals, rather than passing through them. These reflections give rise to a phenomenon called sun pillars. I encourage you to write a little simulation program and investigate them yourself. ■



8 (a) Multiresolution reconstruction of the 22-degree halo. (b) A close-up of (a), showing the red inner and blue outer bands. (c) Multiresolution reconstruction of sun dogs. (d) Multiresolution reconstruction of the upper and lower tangent arcs for a sun elevation of 30 degrees.

Further reading

The basic technique for creating the dot patterns in this pair of columns was developed by Robert Greenler and his colleagues and is described in his book, *Rainbows, Halos, and Glories* (Cambridge University Press, 1980).

If you want to learn more about the colors in the skies, you can look at Minnaert's classic book, *Light and Color in the Outdoors* (Springer-Verlag, 1937, revised in 1985). A more recent volume with lots of good information is *Sunsets, Twilights, and Evening Skies* by Aden and Marjorie Meinel (Cambridge University Press, 1983).

My data for the index of refraction of ice came from the *CRC Handbook of Chemistry and Physics* (CRC Press, New York, 1996). You can read up on the index of refraction and different formulas for computing it in my book, *Principles of Digital Image Synthesis* (Morgan-Kaufmann, San Francisco, 1995); I also talk about Lambert's and Fresnel's Laws in there.

You can learn more about multiresolution compositing in a graphics setting from the famous "apples and oranges" paper by Peter J. Burt and Edward H. Adelson, "A Multiresolution Spline With Application to Image Mosaics" (*ACM Transactions on Graphics*, Vol. 2, No. 4, Oct. 1983, pp. 217-236).