International Journal on Artificial Intelligence Tools
Vol. 22, No. 6 (2013) 1360019 (20 pages)

DOI: 10.1142/S0218213013600191

REAL-TIME DISPERSIVE REFRACTION WITH ADAPTIVE SPECTRAL MAPPING

DAMON BLANCHETTE and EMMANUEL AGU
Computer Science Department, Worcester Polytechnic Institute
Worcester, MA, USA

Received 28 January 2013
Accepted 5 August 2013
Published 20 December 2013

Spectral rendering, or the synthesis of images by taking into account the constituent wavelengths of white light, enables the rendering of iridescent colors caused by phenomena such as dispersion, diffraction, interference and scattering. Caustics, the focusing and defocusing of light through a refractive medium, can be interpreted as a special case of dispersion where all the wavelengths travel along the same paths. In this paper we extend Adaptive Caustic Mapping (ACM), a previously proposed caustics mapping algorithm, to handle physically-based dispersion. Because ACM can display caustics in real-time, it is amenable to extension to handle the more general case of dispersion. We also present a novel algorithm for filling in the gaps that occur due to discrete sampling of the spectrum. Our proposed method runs in screen-space, and is fast enough to display plausible dispersion phenomena at real-time and interactive frame rates.

Keywords: Spectral rendering; real time; dispersion; caustics.

1. Introduction

Spectral rendering is the synthesis of images while taking into account the wave properties of light. Spectral rendering is necessary to render wavelength-dependent optics phenomena such as dispersion, interference, and diffraction, which cause white light to be split into its constituent wavelengths, generating iridescent colors. Spectral rendering can display phenomena such as the rainbows that occur when white light shines through a prism, oil slicks, hummingbird wings and the beautiful colors that appear inside gemstones such as diamonds.

This paper focuses on rendering physically accurate dispersive refraction at real-time frame rates. Dispersion creates the rainbow of colors when white light shines through a prism. Figure 1 shows four examples of dispersive refraction rendered in real-time with our technique. When light travels through a transparent object different wavelengths of light are refracted at different angles, causing white light to be split into its constituent wavelengths (a phenomenon called dispersion). We present an algorithm to render dispersive refraction, a phenomenon that is responsible for rainbows in the sky and the colors generated by a prism, in real-time on programmable graphics hardware.

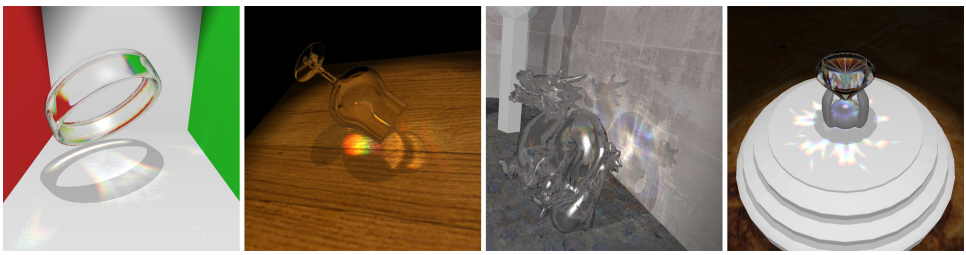

Fig. 1. (Color online) Four images generated with our algorithm. All four scenes are rendered using seven wavelength samples. The ring on the left executed at 15 frames per second, the beer glass at 19 fps, the dragon at 17 fps, and the diamond at 30 fps.

Due to the complexity of spectral rendering, noise-free photorealistic image synthesis of dispersive refraction for example using LuxRender, a physically-based unbiased ray tracer, can take up to one hour. Spectral rendering can be slow because white light has to be sampled at more than three (RGB) wavelengths, and refraction equations are then evaluated at these multiple wavelengths before final conversion to RGB colors for display. By drawing similarities between caustics and dispersive refraction, we have been able to extend a state-of-the-art real-time caustics algorithm to render dispersive refraction in real time. We observe that the key difference between dispersion and caustics is that while the wavelengths of light are refracted along different paths in dispersion, all wavelengths travel along the same paths to generate caustics.

Specifically, we have extended the Adaptive Caustic Mapping (ACM) algorithm[24] to perform real-time spectral dispersion. ACM is a real-time image-space method for generating refractive caustics on programmable graphics hardware. Our method, which we call Adaptive Spectral Mapping (ASM), begins with the ACM algorithm but adds spectral refraction calculations at object surfaces. We extend the concept of caustics maps in order to create spectral maps. To create the spectral map, we simulate external dispersion by refracting seven wavelengths at the surface of refractive objects according to Snell's law. In a separate deferred rendering pass, we also calculate internal dispersion, which occurs when white light is split into component colors inside a refractive object such as the colors seen inside diamonds. Finally, we propose a novel algorithm for filling gaps that occur as a result of sampling at discrete wavelengths of the spectrum. Our algorithm runs at speeds of up to 60 frames per second (FPS) while rendering physically-accurate spectral dispersion. More generally, by pointing out the similarities between caustics mapping and spectral rendering, we hope that our work shall encourage the synthesis of real-time spectral rendering algorithms from caustics algorithms, which have become quite mature in the graphics literature.

The rest of the paper is as follows. Section 2 describes related work. Section 3 gives some background on caustics rendering. Section 4 describes our technique for rendering spectral dispersion using Adaptive Spectral Maps (ASMs), in addition to our method for removing the gaps inherent in the dispersion when taking discrete samples of the spectrum. Section 5 describes our implementation. Section 6 describes our results and Section 7 is our conclusion and future work.

2. Related Work

Due to space constraints, we limit our review of related work to techniques to render dispersive refraction in real time. Initially, spectral rendering was described in the context of ray or path tracing. Cook and Torrance[2] presented a method for rendering materials that takes into account light wavelengths and spectral energy distribution. Thomas,[19] Musgrave,[15] and Wilkie *et al.*[21] all proposed techniques for rendering dispersion using ray tracing methods.

Most recent real-time spectral rendering research uses the Graphics Processing Unit (GPU) to perform wavelength calculations. Guy and Soler[8] describe a technique for real-time rendering of dispersion inside gemstones. Their technique was specific to gemstones and also did not account for the dispersion of light as it exited the object. Kanamori *et al.*[13] published on the physically accurate display of rainbows under different atmospheric conditions. Ďurikovič *et al.*[3] presented an entire spectrally-based framework for interactive image synthesis that could display multilayered thin-film interference. However, they precompute a large portion of the required data and store the resulting data in textures for later access via a shader. We also store scene data in textures (as we are presenting an image-space algorithm), but our approach gathers contributions each frame as opposed to being precomputed. Gathering each frame allows rendering dynamic lights and camera movement, which is not possible when values are precomputed before a scene is actually rendered.

The most similar work to ours is the paper by Sikachev *et al.*,[18] in which they display spectral caustics on planes. Their method of filling in the gaps between color "bands" due to sparse sampling of the spectrum is similar to ours, however they differ in that their algorithm can only project caustics onto planes as opposed to arbitrary surfaces as this paper presents.

3. Background

3.1. *Caustics rendering*

Since our proposed technique extends Adaptive Caustic Mapping, a caustics rendering algorithm, we now review prior work on caustics rendering. The first image synthesis algorithms that could display caustics were offline algorithms such as path tracing. Kajiya[12] and Shirley[17] both described caustics generation using ray and path tracing algorithms. Shirley's paper has a similarity to ours in that sparse "feeler" rays are first fired into the scene to locate any specular objects. A high density of rays is then sent in the direction of specular objects to obtain caustics of sufficient resolution. Jensen's photon mapping algorithm[11] can also generate caustic effects, using a special "caustics photon map" where extra photons are sent in order to create higher resolution data.

In the last decade researchers have been able to generate realistic caustics with interactive algorithms. One of the first was from Wand and Straßer,[20] who utilized the observation that facets of a reflective object lit by a light source create spots on diffuse surfaces that are in essence blurred images of the light source itself. Gunther *et al.*[7]

presented a distributed photon mapping algorithm capable of displaying caustics that ran at interactive rates, but required between 8 and 36 CPUs working in tandem to get sufficient performance. Yu *et al.*[25] presented an algorithm for displaying caustics in real time that renders based on a pair of caustic surfaces instead of using photons gathered on scene geometry as most other methods do.

The idea for "caustic mapping," in which a special texture is created containing caustic data that is projected onto a scene similar to shadow mapping, began with the Shah *et al.* image-space technique.[16] Their method of projecting caustics into the scene is quite similar to ACM, however they refract photons through objects at vertices instead of adaptively sampling the object's surface, as ACM does. Ming *et al.*[14] presented a caustic mapping-like method that showed the realistic effects of light shining through a stained-glass window. However, their caustics are generated based on a stained-glass texture as opposed to dispersive refraction due to an arbitrary object, as ours is.

Wyman *et al*.[22] presented a caustic mapping algorithm similar to Shah's that also operated in image-space. Like photon mapping, it requires two passes: one to emit particles from the light source and interact with a refractive object, and a second to gather their contributions as seen from the eye. Wyman extended his own algorithm with a hierarchical caustics generation method[23] that used mipmaps and a reduced resolution version of the scene to increase algorithm speed. Wyman *et al.* later improved hierarchical caustics mapping to yield Adaptive Caustic Mapping (ACM),[24] which is described in detail in the next section since our work extends it.

3.2. *Adaptive caustic mapping*

We chose to extend Adaptive Caustic Mapping in particular because it solved several issues inherent in other caustic mapping algorithms: notably aliasing due to insufficient sampling and excessive temporal noise due to sampling variations. ACMs use an importance-based adaptive photon sampling algorithm that increases both image quality and rendering speeding when compared to other real-time caustics rendering methods. In addition ACM utilizes a deferred rendering process that displays refractive objects more quickly than other methods.

For a thorough description of ACM, we refer the reader to Wyman's original paper.[24] However, to aid in understanding, we will summarize the important points of the algorithm here and then describe our spectral dispersion extension. Wyman's ACM paper presented separate but related algorithms for two effects: one for caustic generation and a second for displaying refractive objects. We extended both parts, so the next two sections describe each one.

3.2.1. *Caustic generation*

The general structure for using ACMs to create caustic effects is similar to other caustic mapping techniques. Photons are fired into the scene from the light source and a novel method is used to place photons on refractive objects. These photons are refracted

through the object and splatted onto the caustic map, which is then rendered by projecting it into the scene like a shadow map.

Where ACMs differ from other caustic mapping algorithms is in the photon emission and refractive object "locating" phase. Other caustic mapping algorithms normally fix the number of photons prior to emission and send them throughout the entire light's view, often wasting computation time because photons that may not actually hit the refractive object are still processed.

ACMs start with a reduced resolution view of the scene from the light using mipmaps, and only emit a few regularly spaced photons into that image. In a loop, moving up one mipmap level at a time, each photon that actually hits a refractive object is subdivided into four new photons, increasing photon density and thus the resolution of the caustics. Photons that do not hit a refractive object are simply discarded at low mipmap levels, never to be processed. When this photon emission phase is completed, the photon buffer contains a high-resolution set of points that all intersect the surface of the refractive object. Figure 2 illustrates this process on the bunny model. These photons are all then refracted through the object and splatted into the caustic map, as shown in Figure 3 for the case of a sphere.

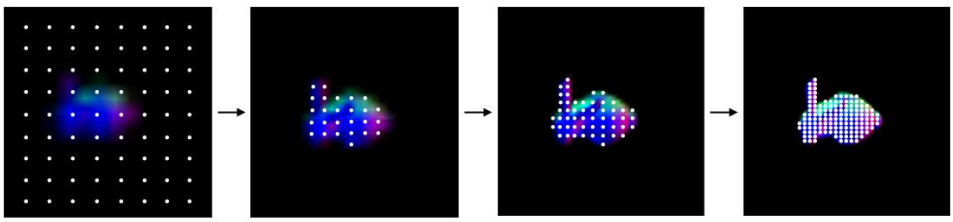

Fig. 2. (Color online) Photon traversal and refinement. In the first stage, very few photons are evenly spread through the scene in order to find the refractive objects. Each subsequent level refines the photons that hit the refractive objects.

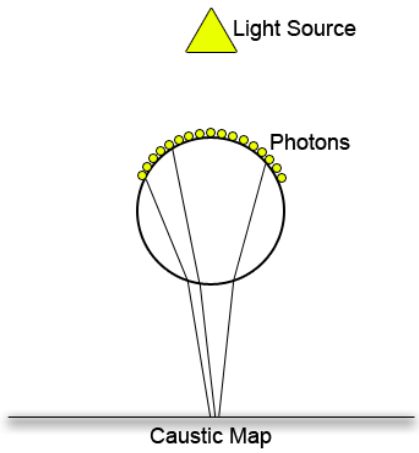

Fig. 3. (Color online) Photon refraction through a sphere, showing three photon paths before splatting.

3.2.2. *Displaying refractive objects with deferred refraction*

After the caustics have been projected onto the background geometry, the display of the refractive objects in the scene is completed in a separate pass. Pixels that lie on a refractive object's surface are treated as photons, just like with caustics calculations. Using the front-facing normal on the refractive object at the current pixel's location, the photon is refracted once, and then a second time at the back-facing surface. The photon is then projected out to the background geometry, where a texture fetch is performed to get the color for that pixel on the refractive object. Figure 7 illustrates this process, with pixel A in that image describing the ACM version.

4. Spectral Dispersion Using Adaptive Spectral Maps

To extend ACMs to render spectral dispersion, we extend the concept of caustics maps in order to create our spectral maps. We made changes to both the caustic generation algorithm and the deferred refraction algorithm. Specifically, the caustic generation algorithm was extended to handle what we will call "external" dispersion, which is produced by light exiting a refractive object and landing on a diffuse surface. This is the type of dispersion is seen when white light passes through a prism. The deferred refraction algorithm was extended to handle "internal" dispersion, which occurs when white light is split into component colors inside a refractive object such as the colors seen inside diamonds.

Our spectral dispersion algorithm begins with a choice of how many wavelengths of the visible light spectrum to utilize. The human eye can see wavelengths between a 400 nm wavelength for violet and around 700 nm for red. We chose to sample seven wavelengths that are evenly distributed through the visible spectrum. The number of samples chosen is arbitrary, though using fewer than seven causes too many missing colors, and with more wavelengths, speed degradation becomes a major issue. Our seven samples each correspond to a different color of the rainbow, and each has a specific wavelength. Figure 4 shows the wavelengths and colors we chose to sample from the spectrum.

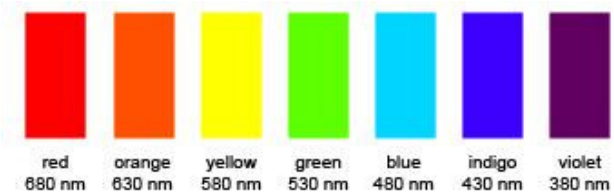

Fig. 4. (Color online) Our seven chosen wavelength samples.

Refraction of light between two mediums with different refractive indices, regardless of wavelength, can be described using Snell's Law.[9] This law is represented by the following equation:

$$n_1 \sin \theta_i = n_2 \sin \theta_t . \quad (1)$$

Light traveling from medium n1 at incident angle ◘i is refracted when entering medium n2 at angle ◘t. The refraction angles in Snell's Law, when considering light dispersion, are wavelength dependent. These angles, taking into account wavelength, can be calculated using Cauchy's equation,[1] which describes an empirical relationship between a wavelength in the visible spectrum and the refractive index of a particular material:

$$n(\lambda) = A + \frac{B}{\lambda^2} + \frac{C}{\lambda^4} + \ldots . \tag{2}$$

Where λ is the wavelength, and *A*, *B*, *C*, etc. are coefficients specific to a particular material. It should be noted that Cauchy's equation only works for the visible spectrum, not the entire spectrum of light. Since we are only interested in the visible spectrum however, it is adequate. The above is the general form of this equation, but for our purposes it is sufficient to use the following two-term form initially used by Musgrave[15]:

$$n(\lambda) = A + \frac{B}{\lambda^2} . \tag{3}$$

The *A* and *B* coefficients are based on physical measurements, and can be found in tables in various sources such as physics textbooks and the Internet. We used data from the Encyclopedic Dictionary of Polymers.[5]

After the photons have all been emitted, positioned in the photon buffer on the refractive object, and are ready to be refracted and splatted, the next stage of our algorithm takes place. Just before each photon refracts at the front surface of the refractive object, it is split into seven separate photons, and each new one is refracted according to the index of refraction generated for it from Cauchy's equation. Each of these seven photons is then refracted a second time on the back-facing surface of the object, after which its final position is calculated for splatting into the spectral map. Figure 5 illustrates this process.

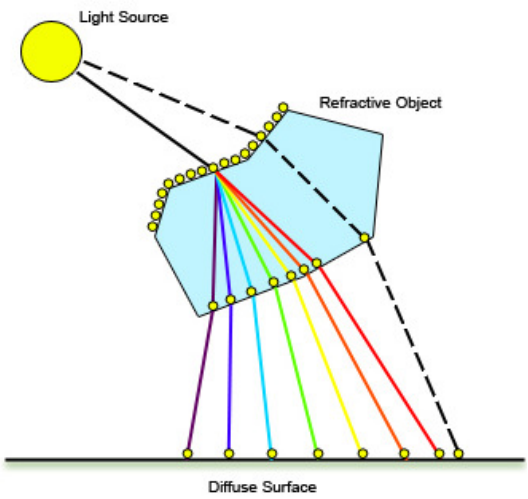

Fig. 5. (Color online) Refraction using seven samples. The dotted line indicates the original ACM algorithm, and the solid lines are our extension.

In Figure 5, the photons are the small yellow circles. Up until they are splatted into the spectral map, they are regarded only as wavelengths. Only after projection from the back of the refractive object, and just before splatting, are they converted from a wavelength to an RGB value.

Again in Figure 5, note how each individual refracted ray's final position on the diffuse surface is in a unique location. Depending on the refractive index of that material, and the shape of the object, all (or most) of the rays may still land in nearly the same location. In this case the colors would all be added back together, producing white. It is for this reason that the RGB values for each wavelength must be carefully calculated so that they sum to white.

At this point the spectral map is complete and ready to be projected into the scene. The spectral map is a texture that contains the final locations of photons that have been refracted through the specular object according to their wavelengths and converted to RGB colors. The difference between the original ACM caustic map and our spectral map is that we have splatted seven times as many photons into the texture, and they are colored and positioned based on wavelength calculations, thus producing spectral dispersion as opposed to simple caustics.

Because we are only using seven samples, the distance between where each specific ray intersects the diffuse surface matters. Issues can sometimes arise in which the caustics have gaps between each color, and Figure 6 illustrates this problem. The next section outlines and describes our novel algorithm for handling these spectrum sampling artifacts on the spectral map.

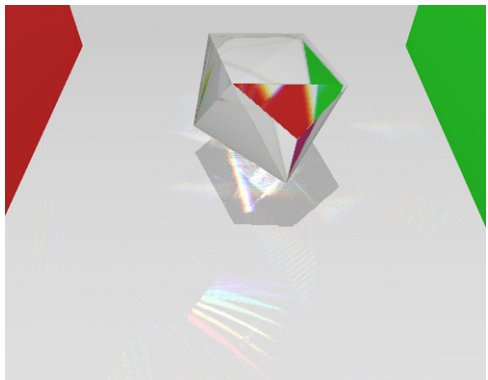

Fig. 6. (Color online) Problems with discontinuous caustics when using seven samples. Each sample color is clearly visible, with gaps between the colors.

Once the spectral map has been created and projected into the scene like a shadow map, internal dispersion is calculated in a completely separate pass at the end. No spectral map is required because we are only coloring the pixels on the surface of the refractive object. In ACMs the color of each pixel on the surface of the refractive object is calculated using a single background color texture fetch, however in ASMs we perform seven texture fetches — one for each wavelength sample. Figure 7 shows how this works.

Pixel A is a representation of original ACMs, and pixel B shows how our extension works. Note that the angles of the lines are exaggerated to illustrate the process better.

Each of these texture fetches may be in a slightly different location, akin to the photons for each wavelength being splatted into a different location in the spectral map. The color of the texel chosen from the background texture is altered by the color of the wavelength that hits it, so if all wavelengths arrive at the same location, or the same color as in Figure 7 (because they add up to white), then the color is exactly the same as the background. The wavelength is converted to an RGB value here, when it is calculated based on the color of the background texture. The largest effect is seen where the background texture has a transition between light and dark colors, because the separate photons of different wavelengths may hit both the light side and the dark side. This can be seen in Figure 6, where the red and gray walls intersect as viewed through the gem an orange color is present.

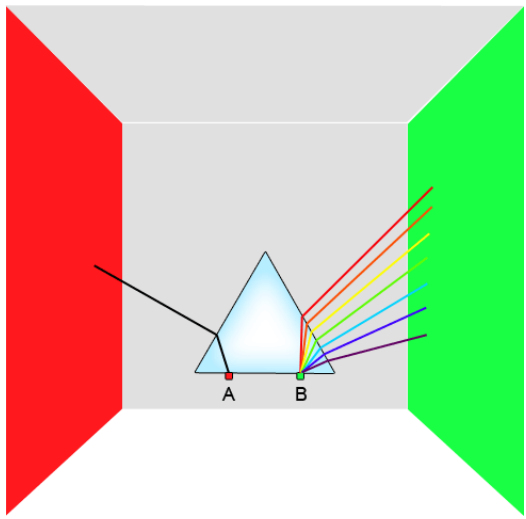

Fig. 7. (Color online) Deferred refraction. View is from above, looking straight down (A and B are on the front of the prism). The pixel on the surface of the prism at A shows original ACM, and the pixel at B illustrates our extension.

4.1. *Filling the gaps*

One of the major issues with spectral rendering using discrete sampling of the spectrum is that gaps or empty portions occur in the resulting spectral map as shown in Figure 6. Note the absence of gaps in the rainbows directly under the refractive object. A general solution to this problem, which works under all situations, must be found: detecting whether gaps occur in the caustics or not in order to fix it.

The simplest solution is to increase the number of wavelength samples taken along the visible spectrum. This approach indeed results in fewer gaps and holes in addition to greater physical accuracy. Unfortunately, it also greatly reduces rendering speed. In

addition, Sikachev *et al.*[18] reported that for their algorithm, gaps still existed even when using 20 wavelength samples. To confirm their claims, we tested our algorithm with 21 samples, and Figure 8 shows the result.

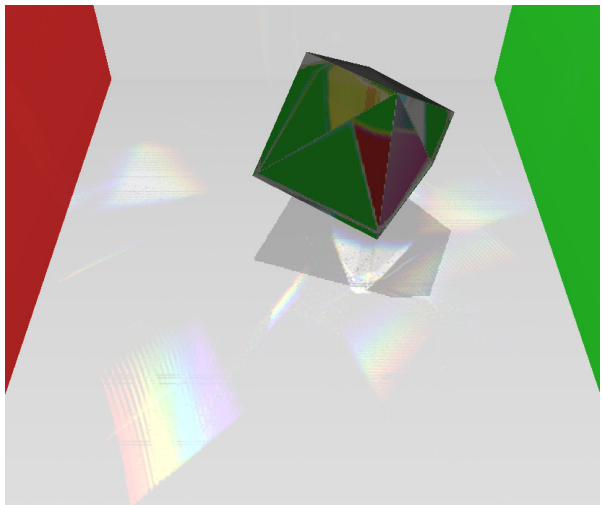

Fig. 8. (Color online) Adaptive Spectral Mapping using 21 evenly-distributed wavelength samples.

As can be seen in Figure 8, the worst large gaps have actually been reduced to a great degree. Where there is a large dispersion amount however, there are still some visible gaps between each color band (notably on the bottom left of the image), and in addition the frame rate drops by 75 percent when 21 samples were used. The gem object Figure 6, with seven samples, was performing at 40 frames per second, and Figure 8, with 21 samples, was performing at 10 frames per second.

Besides increasing the number of wavelength samples, various methods have been proposed. In Ref. 18 again Sikachev *et al.* proposed interpolating colors between the caustics that do exist in order to solve the color gap problem. They integrate the interpolation results for each point by performing additive blending, and use a given step size which is taken in the view space coordinates.

We propose a different method utilizing random sampling. For each texel in the spectral map, a number of randomly-located samples are first taken from a preset radius around the current texel. If the sampled pixel contains no color, it is simply disregarded. If the sampled pixel contains color, it is added to an accumulating color variable. In addition to this color, a counter is incremented so the number of samples that "hit" a color are added up. If the number of hits is above a certain threshold when the sampling is complete, the current pixel's color (if any) is added to the accumulated color from the hit samples. If this threshold is not reached, the current texel's color (if any) is pushed through without adjustment. This method has three results: the first is that gaps are filled effectively using a combination of the pixel colors surrounding the current texel, which is

in essence an interpolation process. The second effect is that if the rainbow has no gaps, but does have banding due to only using seven samples, the bands are removed. The third and final effect is that due to the counting of "hits" around the current pixel, the edges of the dispersion effects are able in many cases to stay relatively sharp compared to just a simple blurring or smearing process. Figure 11 shows gap filling, and Figure 12 illustrates the last two effects.

Gap filling is accomplished completely in image space, with the only input being the spectral map generated in the previous pass, while the output is the spectral map texture with gaps filled in. Pseudocode for our algorithm is shown in Figure 9.

A diagram illustrating the idea behind our gap-filling algorithm is shown in Figure 10. The algorithm begins with line 1 in Figure 9 by fetching the color of the current pixel in

```
1.  fetch spectral map texel at current pixel location, set old_color;
2.  new_color = 0.0;
3.  for (num_samples)
4.        fetch random texel within radius_size from current pixel;
5.        if (random texel has color)
6.              add random texel color to new_color;
7.              increment hit_counter;
8.  if (hit_counter > threshold)
9.        set current_pixel to new_color;
    else
10.       set current_pixel to original color;
```

Fig. 9. Pseudocode for our gap-filling algorithm.

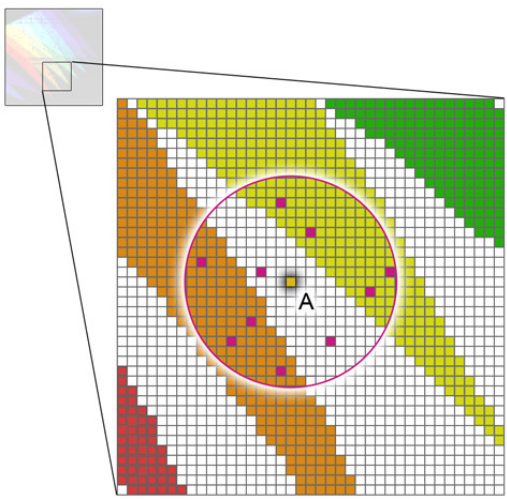

Fig. 10. (Color online) A diagram showing how our filling algorithm works. Each grid square is one texel in the spectral map. Here the purple squares are the random samples. Four are in the yellow band and four are in the orange band, with two not hitting any color. Thus, pixel A is output as a 50–50 combination of orange and yellow.

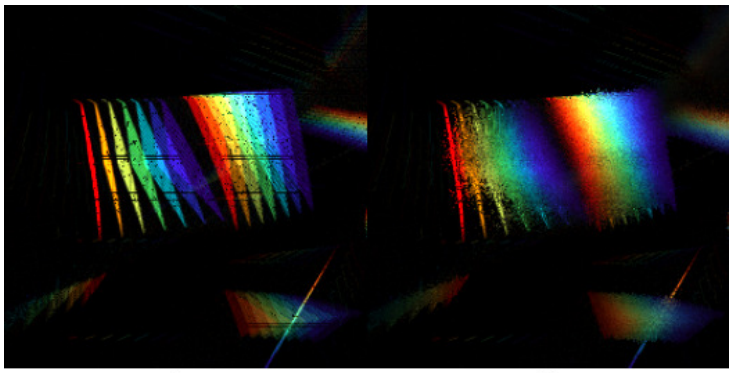

Fig. 11. (Color online) The spectral map texture before (A) and after (B) filling in the gaps. Note small gaps are filled effectively, but larger ones are not — this is due to the sample radius not reaching all the way between the gaps. The edges of the dispersion effect are still sharp, however. A larger radius, while filling all gaps, would create a more blurry dispersion effect.

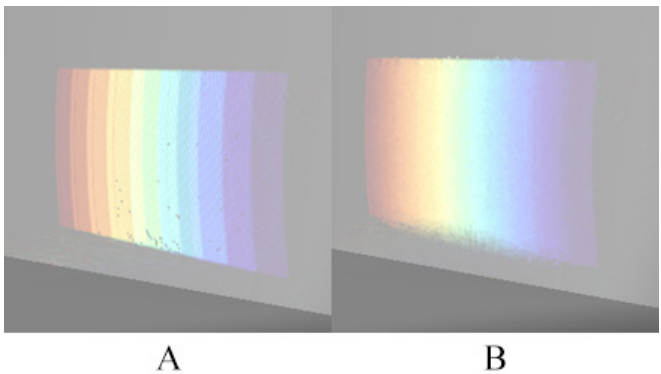

Fig. 12. (Color online) (A) shows the bands of spectral dispersion inherent when using seven samples. (B) shows the result of performing our gap-filling algorithm. Note in (B) the still-sharp border across the top of the dispersion.

the pipeline. A new black color is set in line 2. After this, we iterate for a pre-determined number of samples. Each sample in line 4 is a texel chosen from a random location within a set radius from our current pixel. In line 5 we check whether the sample texel contains any color or not. If color exists, it is added to the black color set back in line 2, and a counter is incremented. In line 8 we check whether this counter is greater than a certain threshold of pixels with color, and if it is, our new color is added to the current pixel's color (if it has any) and is ouput to the new spectral map. Figure 11 shows images of the spectral map before and after our gap-filling procedure.

Of note are that the number of samples, the radius to choose samples from, and the threshold for outputting an adjusted color or a simple pass-through color all have effects on the performance and quality of the final image. Number of samples obviously has a

significant effect on the calculation time of the algorithm. The other two choices are more subtle. The radius for choosing samples matters because if it is too large we may end up picking colors that should not be mixed with the current pixel, or miss adjacent colors altogether. If the radius is too small, then even small gaps will not be filled in. The threshold of hit pixels vs. missed pixels has an effect on both filling gaps and on what the edges of the dispersion will look like. Too low of a threshold will cause a noisy or blurry effect, and too high of a threshold might cause few or no gaps to be filled in. The specific values we chose are described in section 5 with the rest of the implementation information.

5. Implementation

We implemented using C/C++ and OpenGL 4.2, with vertex, geometry, and fragment shaders written in GLSL. The video card we used was a NVIDIA GeForce GTX480 in a Windows 7 environment.

We began by implementing Wyman's Adaptive Caustic Mapping algorithm.[24] His shaders were altered and extended to handle spectral dispersion — specifically to handle multiple wavelength samples instead of just single photon calculations. The photon splatting shaders were extensively modified by inserting a new geometry shader to perform the photon splitting into seven samples, and also to handle the refraction for each wavelength. The fragment shader was altered to handle the extra photons and convert the wavelength values to RGB.

The ACM code was also altered to make it faster, separate from the specific dispersion extensions; instead of traversing through all six mipmap levels of the refractive object texture, we only traverse through three. This had two effects: almost an order of magnitude speed increase (in one scene going from 2 fps to 12 fps), and a reduction in caustic quality. The quality decrease specifically meant dimmer caustics and more "holes", or missing pixels, in them. We took care of some of this with the gap-filling shader.

After creating the spectral map and before projection onto the scene's background geometry, we inserted our novel gap-filling shader. It works directly on the spectral map itself in image-space. Figure 13 shows the entire pipeline for this project from beginning to final image.

Each box in Figure 13 describes a separate render pass. The yellow, blue, and green boxes in the background show how those passes are being rendered — whether it is from the light's view, from the camera's view, or in image space (on a full-screen quad). This diagram also compares our Adaptive Spectral Mapping algorithm with the original ACMs: each white box is unaltered from the original ACM algorithm, light red boxes are altered from the original ACMs, and pass 5, the dark red box, is a completely new pass. As can be seen in the figure, all of the actual spectral dispersion calculations described in this paper are performed in the image space passes. Each and every pass utilizes a completely different shader, except for passes 2 and 3, which both use the same one to output the pixel normal data.

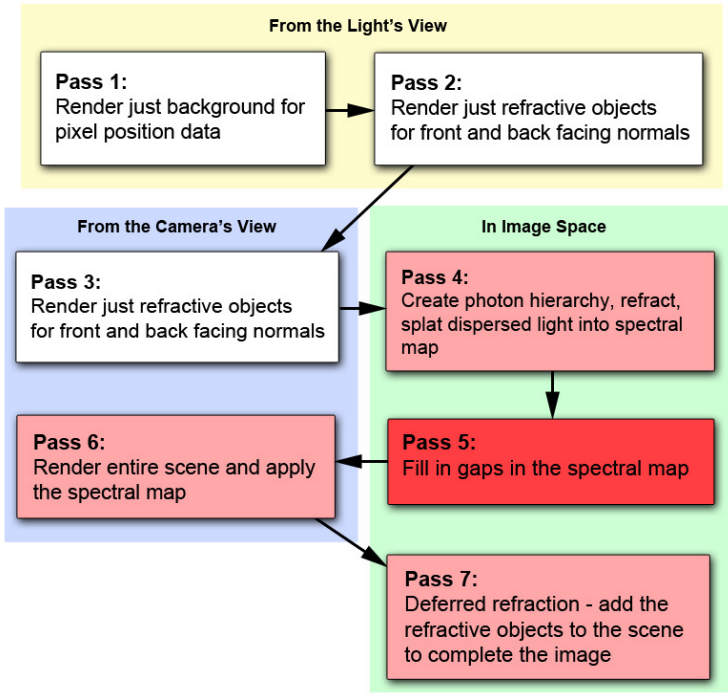

Fig. 13. (Color online) Our rendering pipeline from beginning to end. This whole pipeline is completed each frame. This diagram shows both our Adaptive Spectral Mapping algorithm and the original ACM algorithm: the red boxes are passes altered from original ACMs, and pass 5, with the dark red background, is a completely new pass.

```
1. for each photon wavelength
2.     refract at front surface;
3.     refract at back surface;
4.     emit photon;
5.     compute gaussian intensity;
6.     convert wavelength to RGB;
7.     splat into spectral map;
```

Fig. 14. Photon splat pseudocode.

Pass 4's alterations are shown in the pseudocode in Figure 14. Line 1 was added in order for the algorithm to work with all seven wavelengths instead of just one photon, as with the original ACM code. Line 2 was altered in order to use the GLSL built-in refract() function instead of the custom ACM refraction function for speed reasons (it is not as physically accurate, but the visual difference is quite minimal). Specifically, the difference is that the ACM refraction function takes care of the case where an incident ray reflects off the object's surface, whereas the built-in GLSL function does not. Line 3 still

uses the ACM implementation's more physically accurate refraction function, but both lines 2 and 3 were edited to use our Cauchy equation-calculated refractive indices per wavelength. Line 4 is a geometry shader requirement, just there to emit the photon/vertex so the fragment shader will see each and every photon. Line 5 is identical to the ACM author's original code. In our newly created line 6, we convert the wavelength for that photon into an RGB value. Line 7 is a simple call to gl_FragData, required for all fragment shaders.

Our conversion from a wavelength to an RGB value is simple. Because we know exactly which wavelengths are being sent to the splat shader, we can set a constant red, green, and blue value for each one. As mentioned previously, these values must be carefully chosen to make sure they sum to white.[15] Table 1 gives the RGB values we used for each wavelength. These are approximations based on using the CIE color matching functions to get the relative contributions of light from wavelength, and converting them to XYZ color space coordinates.[4] The color matching functions can be described by the integral:

$$X = \int_0^\infty I(\lambda)\bar{x}(\lambda)d\lambda\,. \tag{4}$$

Where $I(\lambda)$ is the spectral power distribution, $\bar{x}$ is the color-matching function, and λ is the wavelength in nanometers. The Y and Z components are calculated in the same way. From the XYZ coordinates, it is possible to get RGB values using the CIE color space.

Table 1. The RGB values we chose for each wavelength. Colors are on a 0–255 scale. Before splatting, the totals are scaled so all dispersed values are not bright white.

Wavelength (nm)	Red	Green	Blue
380	77	0	204
430	51	51	255
480	0	229.5	255
530	76.5	255	102
580	204	229.5	77
630	229.5	128	0
680	255	0	0
Total	893	893	893

In this way, a particular pixel's color in the scene is summed for each wavelength-specific photon that hits it. If all wavelengths end up on the same pixel, it will be white. If only one photon wavelength hits a particular pixel, the pixel will only be that color.

After the spectral map is created, the next pass performs our gap filling shader to take care of gaps and any noise or missing pixels in the spectral map as described in Section 4.1. In our testing and for our scenes, we found that a good maximum sample radius was seven pixels out from the current pixel. A smaller radius did not catch enough

gaps or missing pixels, and a larger one caused too much blurring. Within that radius, 15 samples appeared to give sufficient color quality while still performing well. Our threshold for detecting whether we are inside a gap or not was set at 10 — that is, if at least 10 out of the 15 total samples actually detect color in the spectral map, the pixel's color is changed. These are the values we used for our performance tests and in the images in Figures 1, 11, 12, 15, 16, and 17. Note that these values worked well for our particular scenes, and different values may work better for scenes where the refractive object takes up most of the light's view. For example, a diamond that has a light very close to it would most garner better performance with fewer than 15 samples within the sample radius. Gap-filling is not performed on the surface of the refractive object as it is with the spectral caustic map because gaps and missing pixels do not occur there. This is due to the fact that photons are not being splatted into a separate map — colors are being pulled from background geometry, which always exists (or is black if nothing is there). The one issue that may come up is non-smooth transitions between some colors, which was also observed and outlined in Ref. 21. It is normally only apparent when using an unnatural and extremely dispersive refractive index for the object.

6. Results

Table 2 lists performance results of our algorithm compared to Adaptive Caustic Mapping, and Figure 15 shows all our test scenes. We performed tests on our Adaptive Spectral Mapping algorithm using both seven and 21 wavelength samples. As can be seen in the table, the extra photons needed for dispersion and gap-filling reduce performance in some scenes, and increasing wavelength samples severely reduced speed in all scenes. The sphere, at least with seven samples, still performs at the same speed as with ACMs, most likely due to its simplicity and the small size of its footprint from the view of the light. The gem, being composed of far fewer faces than all the other objects, still performs slower than the sphere since it is rendered onto more fragments due to its size, which is an image-space algorithm issue discussed in the following paragraphs. The greatest difference in performance is the glass on the table, which we believe is also due to its physical size in the light's view. Also, because the original ACM algorithm has no smoothing or blurring shader, it can display the scenes with better performance.

Table 2. Frame rates for our test objects, which can be seen in Figure 15. This table compares the relative speeds of the different rendering methods and shows the number of faces in each scene. Note that the renderer had could run at a maximum frame rate of 60 Frames Per Second (FPS).

Object	Number of Faces	ACMs (FPS)	ASMs 7 Samples (FPS)	ASMs 21 Samples (FPS)
Sphere	5120	60	60	19
Ring	65536	27	20	9
Gem	24	60	40	10
Bunny	138902	12	10	6
Glass on Table	12137	60	16	5

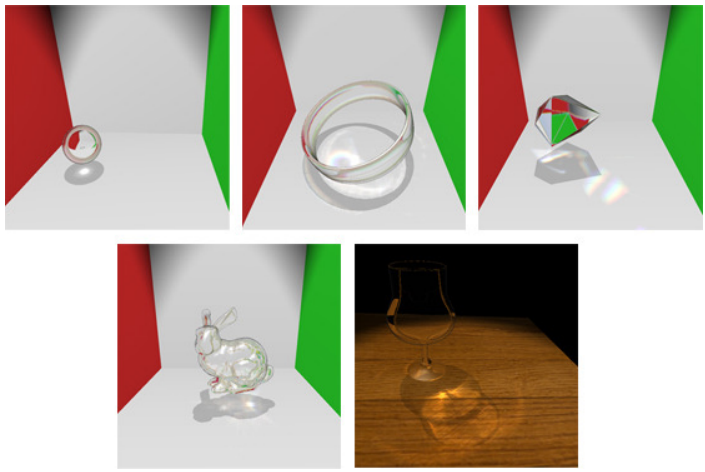

Fig. 15. (Color online) Our five testing scenes. Dispersion is intentionally made brighter than it normally would be to show detail.

Since this algorithm runs in image space, the number of pixels covered by the refractive object from the light's view has an impact on frame rate. The closer the object is to the light, the more pixels involved in caustic calculations, and the slower the performance. In fact, the relationship between this number of pixels involved and frame rate is closely tied. Table 3 shows what happens as the sphere is moved closer to the light source.

Table 3. Table showing relative number of pixels taken up by a refractive sphere as seen from the light source and a frame rate comparison.

Frame Rate (frames per second)	Percentage of Total Pixels Covered by Refractive Object
60	2.5% (sphere on "floor" of Cornell box)
40	4%
30	7%
20	13%
10	33%

The percentage of fragments covered in the light's view by the refractive object directly affects the algorithm's performance. Of course, this is closely related to the common graphics problem of quality vs. speed as well. If the refractive object is close to the light, then more photons will be refracted through the object, increasing the quality of the caustics and dispersion. However, as Table 3 illustrates, the quality boost also results in lower frame rates.

Figure 16 shows a comparison between a screenshot of our software and a ground truth image rendered with the unbiased offline engine LuxRender. Part A in the figure took 1.5 hours to render, and part B was performing at 44 frames per second. Our

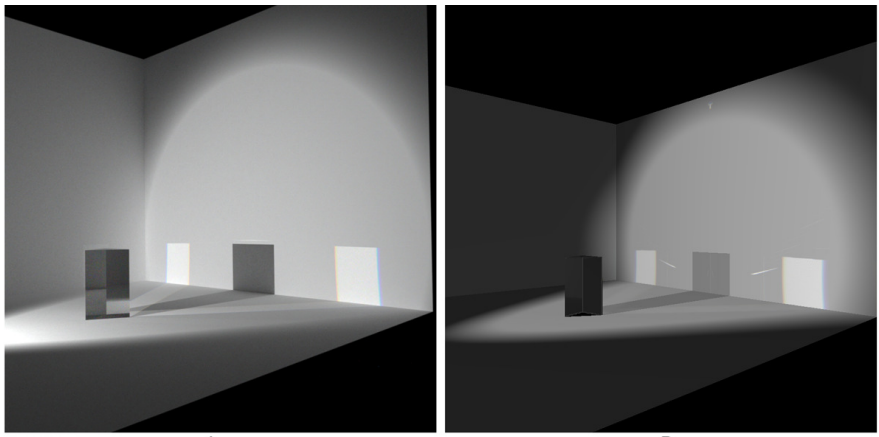

Fig. 16. A comparison between a ground-truth render of the scene. A, and a scene created with our algorithm, B. Render time for A was 1.5 hours, and B was performing at 44 frames per second.

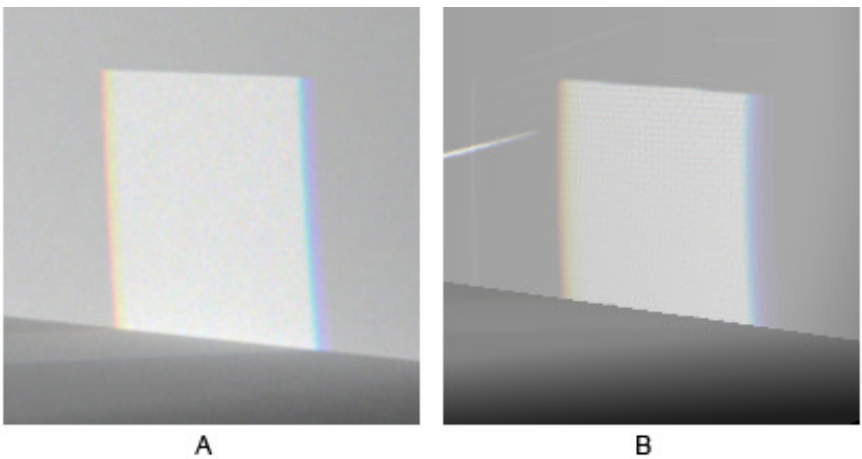

Fig. 17. (Color online) A close-up of some of the dispersion from Figure 16.

algorithm has produced a physically plausible representation of dispersion through a prism, which runs at real-time frame rates. There are a few things to note: the size and position of the dispersion and shadow on the wall are quite similar, though in the scene using our algorithm they are slightly shorter. This is most likely due to the positioning of the light being slightly different in each scene. Our scene contains a couple artifacts: a straight line between the shadow and the dispersion. These are possibly a result of imperfect sampling of the spectral map, or issues with light refraction calculations through certain triangles of the prism itself. The prism itself is also slightly different, probably due to the image-space refraction method used in our algorithm, as opposed to the more physically accurate method used in LuxRender.

Figure 17 gives a close-up comparison of the dispersion on the wall. As shown, the dispersion is very similar between the ground truth and our algorithm's image, especially

in color. Take particular note of the slight red shift on the left of each band of light, and the blue shift on the right of each band. These color shifts are a result of utilizing spectral dispersion calculations, and would not be present using an algorithm that does not account for the wave nature of light.

7. Conclusion and Future Work

We have presented Adaptive Spectral Mapping, a spectral dispersion extension to the proposed algorithm Adaptive Caustic Mapping. We described our changes to both ACM itself and to the related algorithm for deferred refraction. Our algorithm displays a plausible approximation of the dispersion phenomenon of light, and does so at interactive and real-time frame rates. Our ASM algorithm is one of the first of its kind, bringing spectral rendering one step closer to being fully displayed in real-time contexts such as games.

There are some limitations to our algorithm, however. The gap-filling procedure creates horizontal and vertical lines in some situations due to our sampling process. This could be ameliorated with a more random sampling method, perhaps inside a certain radius around the current pixel. This would introduce a temporal cohesion issue (depending on the randomness of the sampling), but at the same time there would be fewer vertical and horizontal noise lines.

Many opportunities exist for future directions of research. The first of which is to extend ASMs to simulate reflective caustics, which at least one other caustics mapping algorithm[16] has succeeded in accomplishing. Others include extending ASMs to display other spectral phenomena that require wavelength calculations, such as diffraction and thin-film interference. Integrating a fast volumetric caustics algorithm with ours would produce beautiful images, along the lines of recent research such as Ref. 10. Dispersion colors, like shadows, become more diffuse the farther they are from the object that creates them. Any gaps between the colors also become larger the farther they are from the refractive object. A distance-aware blurring algorithm such as Screen-Space Soft Shadows[6] could be modified to work with our spectral maps to make them more physically accurate, and also help with very distant gaps as well.

References

1. Germain Chartier, *Introduction to Optics* (Springer, 2005).
2. Robert Cook and Kenneth Torrance, A reflectance model for computer graphics, *ACM Transactions on Graphics* **1**(1) (1982) 7–24.
3. Roman Ďurikovič and Ryou Kimura, Spectrum-based rendering using programmable graphics hardware, SCCG ’05: *Proc. 21st Spring Conference on Computer Graphics* (2005), pp. 233–236,
4. Glenn F. Evans and Michael D. McCool, Stratified wavelength clusters for efficient spectral Monte Carlo rendering, *Graphics Interface* (1999).
5. Jan W. Gooch, *Encyclopedic Dictionary of Polymers*, Vol. 1 (Springer, 2010).
6. Jesus Gumbau, Miguel Chover and Mateu Sbert, Screen space soft shadows, *GPU Pro* (AK Peters, 2010), pp. 477–490.

7. Johannes Gunther, Ingo Wald and Philipp Slusallek, Realtime caustics using distributed photon mapping, *Eurographics Symposium on Rendering* (2004), pp. 111–122.
8. Stephane Guy and Cyril Soler, Graphics gems revisited: Fast and physically-based rendering of gemstones, *ACM Transactions on Graphics* **23**(3) (2004) 231–238.
9. Eugene Hecht, *Optics* (Addison Wesley, 2001), 4th edition.
10. Wei Hu, Zhao Dong, Ivo Ihrke, Thorsten Grosch, Guodong Yuan and Hans-Peter Seidel, Interactive volume caustics in single-scattering media, I3D 2010 in *Proc. 2010 ACM SIGGRAPH Symp. Interactive 3D Graphics and Games* (2010), pp. 109–117.
11. Henrik Wann Jensen, Global illumination using photon maps. *Rendering Techniques '96* (1996), pp. 21–30.
12. James Kajiya, The rendering equation, *SIGGRAPH '86 Proc.*, Vol. 20, No. 4, pp. 143–150 (1986).
13. S. Kanamori, K. Fujiwara, T. Yoshinobu, B. Raytchev, T. Tamaki and K. Kaneda, Physically-based rendering of rainbows under various atmospheric conditions, *Computer Graphics and Applications* (*PG*) (2010), pp. 39–45.
14. Shihua Ming, Jung-A Kim, Kyung-kyu Kang, Xianji Li, Sung-yul Yim and Dongho Kim, Realistic illumination model and caustics generation method for real-time stained glass rendering, Lecture Notes in Computer Science, Vol. 4563/2007, pp. 80–87 (2007).
15. F. Kenton Musgrave, Prisms and rainbows: A dispersion model for computer graphics, in *Proc. of Graphics Interface '89* (1989), pp. 39–45.
16. Musawir Shah, Jaakko Konttinen and Sumanta Pattanaik, Caustics mapping: An image-space technique for real-time caustics, *IEEE Transactions on Visualization and Computer Graphics* (2005).
17. Peter Shirley, A ray tracing method for illumination calculation in diffuse-specular scenes, *Proc. on Graphics Interface '90* (1990), pp. 205–212.
18. Peter Sikachev, Ilya Tisevich and Alexey Ignatenko, Rendering smooth spectrum caustics on plane for refractive polyhedrons, *18th Int. Conf. on Computer Graphics* (*Graphicon '08*) (2008), pp. 172–176.
19. Spencer Thomas, Dispersive refraction in ray tracing, *The Visual Computer* **2**(1) (1986) 3–8.
20. M. Wand and W. Straßer, Real-time caustics, *Eurographics 2003* **22**(3) (2003) 611–620.
21. Alexander Wilkie, Robert Tobler and Werner Purgathofer, Raytracing of dispersion effects in transparent materials, *WSCG 2000 Conference Proc.* (2000), pp. 200–207.
22. Chris Wyman and Scott Davis, Interactive image-space techniques for approximating caustics, *ACM Symp. on Interactive 3D Graphics and Games* (2006), pp. 153–160.
23. Chris Wyman, Hierarchical caustic maps, *ACM Symp. on Interactive 3D Graphics and Games* (2008), pp. 163–171.
24. Chris Wyman and Greg Nichols, Adaptive caustic maps using deferred shading, *Computer Graphics Forum* **28**(2) (2009) 309–318.
25. Xuan Yu, Feng Li and Jingyi Yu, Image-space caustics and curvatures, *Computer Graphics and Applications* (2007), pp. 181–188.