Visual Comput (2008) 24: 485–494
DOI 10.1007/s00371-008-0229-9

ORIGINAL ARTICLE

Xinguo Liu
Zhao Dong
Hujun Bao
Qunsheng Peng

Caustic spot light for rendering caustics*

Published online: 29 May 2008

X. Liu (✉) · H. Bao · Q. Peng
State Key Lab of CAD&CG, Zhejiang University, Hangzhou 310027, P.R. China
{xgliu, bao, peng}@cad.zju.edu.cn

Z. Dong
Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany
dong@mpi-sb.mpg.de

Abstract It is difficult to render caustic patterns at interactive frame rates. This paper introduces new rendering techniques that relax current constraints, allowing scenes with moving, non-rigid scene objects, rigid caustic objects, and rotating directional light sources to be rendered in real-time with GPU hardware acceleration. Because our algorithm estimates the intensity and the direction of caustic light, rendering of non-Lambertian surfaces is supported. Previous caustics algorithms have separated the problem into pre-rendering and rendering phases, storing intermediate results in data structures such as photon maps or radiance transfer functions. Our central idea is to use specially parameterized spot lights, called caustic spot lights (CSLs), as the intermediate representation of a two-phase algorithm. CSLs are flexible enough that a small number can approximate the light leaving a caustic object, yet simple enough that they can be efficiently evaluated by a pixel shader program during accelerated rendering. We extend our approach to support changing lighting direction by further dividing the pre-rendering phase into per-scene and per-frame components: the per-frame phase computes frame-specific CSLs by interpolating between CSLs that were pre-computed with differing light directions.

Keywords Image synthesis · Display algorithm · Caustic · Spot light

1 Introduction

Caustic light patterns are formed when incoming light rays are focused or defocused by reflection or refraction. Early works on caustics modeled and rendered the light patterns produced by a rippling surface of water based on beam tracing [4] and experience-based models [14, 15]. Nishita et al. [8] and Iwasaki et al. [5] used prism shaped caustic volumes to calculate the energy flux passed onto the polygon. Brière proposed an adaptive tracing algorithm for the caustic volumes [1], which is however targeted for off-line ray tracing. Based on caustic volumes, Ernest et al. [2] proposed a caustic volume warping to simplify the computation and improve the results. Our interest in this paper is in the light patterns produced by rigid objects, such as those illustrated in Fig. 1.

*This work is supported in part by NSFC (No. 60603078), the 973 Program of China (No. 2006CB303102 and No. 2002CB312100), and the Program for New Century Excellent Talents in University of China (No. NCET-06-0516).

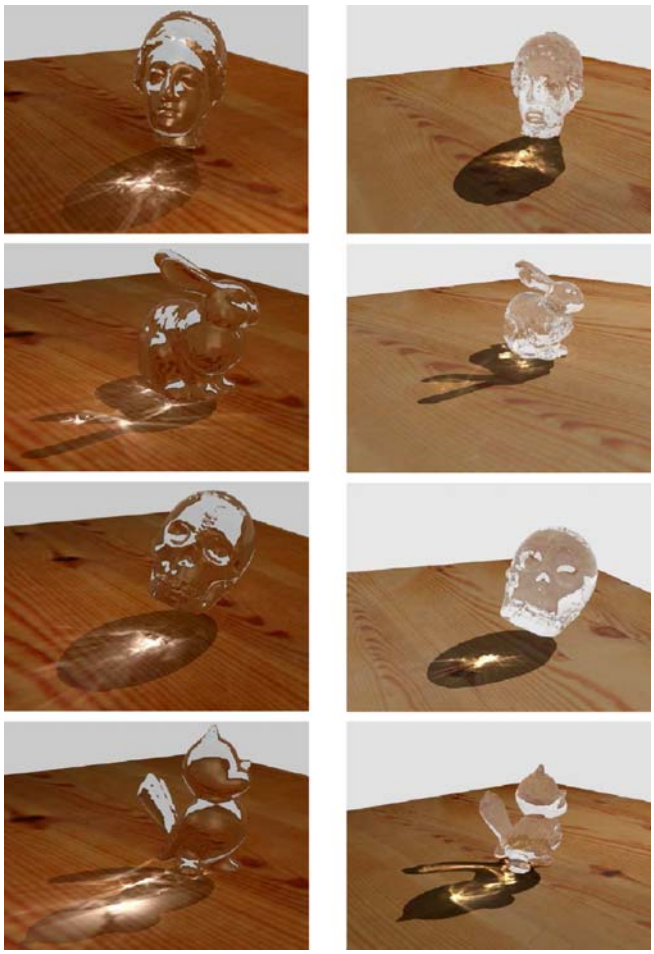

Fig. 1. Caustic light patterns. *Left* column was rendered using CSLs. Frame rates are 19, 19, 20, and 18 frames per second from *top* to *bottom*. *Right* column was rendered using ray tracing software POV-Ray (http://www.povray.org/)

Because caustic patterns are produced by indirect light, a complete solution requires global illumination techniques, such as ray tracing [12] and photon mapping [6, 7]. GPU-accelerated implementation of photon mapping [10] and distributed photon mapping [3] can interactively render caustics. The image-space photon mapping method [18], based on the double frame buffer technique [17], can render caustic images, which are competitive with off-line renderings using a comparable number of photons. Recently, Musawir et al. [11] presented a similar caustic mapping technique to render caustics onto non-shiny surfaces. Yu et al. [20] combines the caustic mapping [11] and volume warping approaches [2] by parameterizing the caustic rays with a pair of parallel planes. Wand et al. placed pinhole cameras at many sampling points on the caustic object, and used the cameras to project distant environmental illumination onto scene objects to render caustics [16]. Rendering time is in linear proportion to output image size and sample numbers. Using roughly 100 samples, small frames can be rendered in real-time.

Pre-computed radiance transfer functions can render caustics using graphics hardware in real-time when illumination is limited to low spatial frequencies. PRT [13] tabulates outgoing radiance as a function of the source illumination at every vertex in the scene during a preprocessing step. This approach requires dense tessellation of the scene, and the per-frame rendering cost is in linear proportion to the number of the vertices and the number of lights. Recent work [19] pre-computes caustic intensity in the caustic object's surrounding volume for a dense set of directional incident light sources, and looks up the caustic intensity during rendering. Interactive frame rates can be achieved for moving light sources and moving objects.

A common approach to rendering caustic patterns is to separate the problem into pre-rendering and rendering phases, storing intermediate results in data structures such as the photon maps and radiance transfer functions cited above. The pre-rendering phase is executed only once; it can involve complex operations such as casting millions of rays and building complex data structures. The rendering phase is executed repeatedly and is therefore of limited complexity if rendering is to be done at interactive or (especially) real-time rates. Modern GPUs with powerful, programmable pixel shaders have created the opportunity to rebalance the workload between the pre-rendering and rendering phases.

Our central idea is to use specially parameterized spot lights, called caustic spot lights (CSLs), as the intermediate representation of a rebalanced two-phase algorithm. The parameterization of the CSLs was chosen so that a small number can approximate the light leaving a caustic object under the illumination of a single directional light source. During a long pre-rendering phase thousands (or millions) of rays are cast in the lighting direction, reflected or refracted by the caustic object, and sorted into bundles that are coalesced into such a set of CSLs. If the lighting direction is held constant with respect to the caustic objects, a single set of CSLs is computed for each object/light source pair.

We extend our approach to support changing lighting direction by further dividing the pre-rendering phase into per-scene and per-frame phases. During the per-scene phase complete sets of CSLs are generated for a regular distribution of lighting directions, and correspondences are established between CSLs in related directions so that interpolations can be performed. (These correspondences are not obvious, because the CSL sets for different illuminations of a caustic object typically have different numbers of CSLs.) At the start of each frame the CSL sets for the three lighting directions most similar to the current direction are interpolated to form a specific CSL set for that frame.

Table 1. A comparison of the feature strengths of various caustics rendering techniques

	Photon map	Caustic mapping	Caustic volume	Pinhole	PRT	LPI†	CSL
Directional illumination	√	√	√			√	√
Local point illumination	√	√	√			√	
Environment map illumination	√			√	√		
Movable light sources		√	√	√	√	√	√
Movable scene objects		√	√	√		√	√
Movable caustic objects		√	√	√	√	√	√
Occluded light sources	√	√	√				
Occluded caustic light	√	√	√		√		√
Non-diffuse shading	√			√	√		√

† Local pre-computed irradiance

CSLs are robust enough to approximate caustic light, yet simple enough that they can be efficiently sampled by a pixel shader program during accelerated rendering. The render phase is multipass: one pass per CSL in the currently active set. During each pass all the objects in the scene are rendered under the direct illumination of the active CSL, with the results accumulated into the frame buffer, which is initialized to zero at the start of the frame. Rendering by direct illumination of spot lights has several advantages: no dependence on the position or shape of scene objects, ability to perform arbitrary shading calculations, and potential for the computation of shadows due to occluders between the spot lights and other scene objects given an efficient ray-object intersection testing algorithm.

Figure 1 illustrates four frames that were rendered using caustic spot lights. Table 1 summarizes the feature strengths, but not the performances, of various caustics rendering techniques, including the CSL technique that is described in this paper.

The rest of this paper is organized as follows. Section 2 describes the implementation of caustic spot lights, and the algorithm used to evaluate CSLs at specific scene positions. Section 3 describes the generation of a CSL set for static lighting, and the rendering algorithm that is used with these CSLs. Section 4 describes CSL interpolation and a three-phase rendering algorithm that together support rendering with dynamic light sources. Section 5 describes implementation details and results, and Sect. 6 concludes and identifies areas of future work.

2 Caustic spot lights

A caustic spot light (CSL) defines a locus of light rays and the intensities of those rays. Inspired by the refraction pattern of a transparent sphere (Fig. 2), we implemented our CSLs with radially symmetric ray distributions. Ray intensities, however, are specified independently to better accommodate the variations in more complex caustic objects.

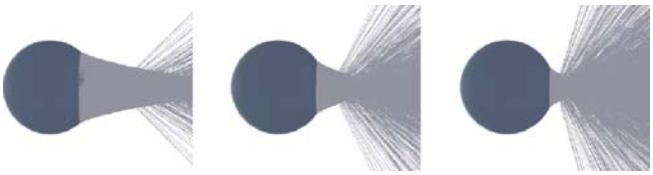

Fig. 2. Refracted rays of a directional light source through a transparent sphere with different refractive indices 1.1, 1.2, and 1.3 from *left* to *right*

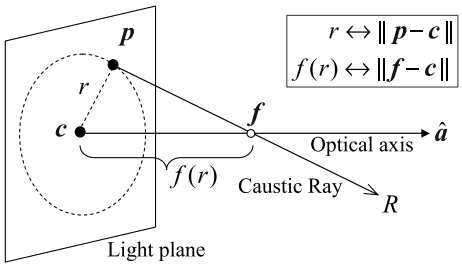

Fig. 3. Geometry of a caustic spot light. The *rectangle* represents the light plane recording the ray's intensity. The term $\boldsymbol{c}$ represents the origin of the light plane, and $\hat{\boldsymbol{a}}$ represents the normal vector of the light plane. The ray extending from $\boldsymbol{c}$ in direction $\hat{\boldsymbol{a}}$ is defined as the optical axis

It is worthy of pointing out that a CSL does not have a single focal point like a normal spot light in OpenGL does. In the following, 3D points are usually denoted by small bold letters, unit vectors by a hat symbol (e.g., $\hat{\boldsymbol{a}}$), and scalars by small letters.

2.1 CSL implementation

Figure 3 illustrates our implementation of a caustic spot light. A 2D image plane, called the light plane, is the origin of all rays. The term $\boldsymbol{c}$ is the origin of the light plane. The ray extending orthogonally from $\boldsymbol{c}$ is called the optical axis. From each point $\boldsymbol{p}$ on the light plane, a single ray R extends with intensity specified as the light plane value

at $\boldsymbol{p}$, and intersects the optical axis at a point $\boldsymbol{f}$. The term r is the distance of point $\boldsymbol{p}$ to the origin of the light plane, i.e., $r = \|\boldsymbol{p} - \boldsymbol{c}\|$. Because we require a radially symmetric distribution, the distance of point $\boldsymbol{f}$ to the light plane origin can be expressed as a function of r: $\|\boldsymbol{f} - \boldsymbol{c}\| = f(r)$. We parameterize this function as

$$f(r) = \frac{\alpha}{1 + \beta r}, \quad \text{where } \alpha > 0 \text{ and } \beta \geq 0. \tag{1}$$

This parameterization expresses a gamut of patterns, from typical spot lights ($\alpha = 0$) through collimated light sources (as α approaches ∞), which includes the diffraction patterns of spheres with varying refraction indices. Note that a CSL cannot be generalized to an area light source because exactly one ray leaves each light plane position.

When ray R is colinear with the optical axis, the intersection point $\boldsymbol{f}$ is not defined. For this special ray, we define $\boldsymbol{f} = \boldsymbol{c} + f(0)\hat{\boldsymbol{a}}$ to maintain the continuity.

2.2 CSL evaluation

Because a CSL defines intensities for only a finite locus of rays, determining its effect on a point $\boldsymbol{x}$ involves first determining which, if any, CSL rays intersect $\boldsymbol{x}$. This determination is complicated by the movement of the ray focus point along the optical axis. Let $\boldsymbol{z}$ be the projection of $\boldsymbol{x}$ on the optical axis. In the general situation, in which $\boldsymbol{x}$ is not on the optical axis, solutions are possible for two cases: (i) $\boldsymbol{z}$ is nearer to the light plane than the focus point $\boldsymbol{f}$ of the intersecting ray, and (ii) $\boldsymbol{z}$ is farther from the light plane than the focus point $\boldsymbol{f}$. Figure 4 illustrates these two cases.

In both cases (i) and (ii) the line of possible ray origins in the light plane is easily computed; the difficulty is in determining the values of r, the distance of the ray origin to the light plane origin $\boldsymbol{c}$. By similar triangles, we have $\frac{\|\boldsymbol{f}-\boldsymbol{z}\|}{\|\boldsymbol{f}-\boldsymbol{c}\|} = \frac{\|\boldsymbol{x}-\boldsymbol{z}\|}{\|\boldsymbol{p}-\boldsymbol{c}\|}$. Let z and ρ be the distance of $\boldsymbol{x}$ to the light plane and the optical axis, respectively. Then,

$$\text{Case (i):} \quad \frac{f(r) - z}{f(r)} = \frac{\rho}{r}, \quad \text{or} \quad \text{case (ii):} \quad \frac{z - f(r)}{f(r)} = \frac{\rho}{r}.$$

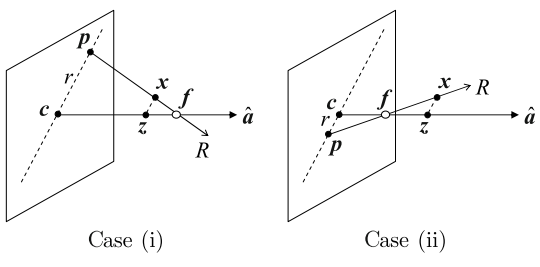

Fig. 4. Evaluation of a CSL. As many as three CSL rays may intersect point $\boldsymbol{x}$

In case (i), substituting Eq. 1 leaves

$$r_{1,2} = \frac{(\alpha - z) \pm \sqrt{\delta_i}}{2\beta z}, \tag{2}$$

where $\delta_i = (z - \alpha)^2 - 4\alpha\beta\rho z > 0$. Since negative r does not make sense, there are two valid solutions for r when $\delta_i > 0$ and $z < \alpha$, and there are no valid solutions otherwise. In case (ii) the same substitution leaves

$$r_3 = \frac{(\alpha - z) + \sqrt{\delta_{ii}}}{2\beta z}, \tag{3}$$

where $\delta_{ii} = (z - \alpha)^2 + 4\alpha\beta\rho z > 0$. There is exactly one valid solution r_3, because δ_{ii} is always greater than $(z - \alpha)^2$.

The solution set is therefore either one radius (r_3) or three radii (r_1, r_2, and r_3). Because the light plane is implemented as a finite image array, rays whose radii extend beyond the light plane are eliminated. Before the intensities corresponding to the remaining rays are summed, however, intensity correction corresponding to focusing effects must be performed.

2.3 CSL intensity correction

The light plane intensity at point $\boldsymbol{x}$ actually specifies the total intensity emitted from a unit region centered at $\boldsymbol{p}$. Because these rays are focused on the optical axis at (approximately) distance $f(r)$, the aggregate beam may be focused or defocused when it reaches $\boldsymbol{x}$, as shown in Fig. 5.

To determine the beam density at $\boldsymbol{x}$, we define a small patch around $\boldsymbol{p}$ on the light plane, extending $\mathrm{d}r$ in the radial direction and $\mathrm{d}\theta$ in the angular direction. The area of the patch is: $((r + \mathrm{d}r)^2 - (r - \mathrm{d}r)^2)\,\mathrm{d}\theta$. Consider a cross section parallel to the light plane at point $\boldsymbol{x}$, as shown in Fig. 5. By the symmetric property of CSL, CSL rays extending from the circles $r + \mathrm{d}r$ and $r - \mathrm{d}r$ intersect the cross section at two circles.

Consider case (i). By similar triangles, the radius of the two circles on the cross section are, respectively, $h(r - \mathrm{d}r)$ and $h(r + \mathrm{d}r)$, where $h(r) = \frac{f(r)-z}{f(r)} r$. Therefore,

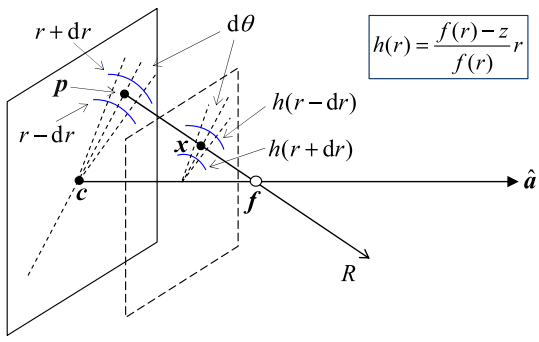

Fig. 5. Light intensity changes with the area of the cross section

Fig. 6. Comparison of caustic renderings between with (*left*) and without (*right*) CSL intensity correction

the mapped area in the cross section is $|h^2(r+\mathrm{d}r) - h^2(r-\mathrm{d}r)|\,\mathrm{d}\theta$. Then the limit of the area ratio is

$$\sigma = \lim_{\mathrm{d}r\to 0} \frac{|h^2(r+\mathrm{d}r) - h^2(r-\mathrm{d}r)|\,\mathrm{d}\theta}{((r+\mathrm{d}r)^2-(r-\mathrm{d}r)^2)\,\mathrm{d}\theta} = \left|\frac{h(r)h'(r)}{r}\right|. \quad (4)$$

For case (ii), a similar calculation yields exactly the same area ratio.

The corrected intensity of a CSL ray at point $\boldsymbol{x}$ is therefore the light plane value at $\boldsymbol{p}$ divided by the area correction factor defined in Eq. 4; and the overall evaluation of the CSL is the sum of the corrected intensities of the $1 \sim 3$ valid rays. The result comparison in Fig. 6 shows that our intensity method can greatly improve the rendering quality.

2.4 Points on the optical axis

Evaluating the illumination of a point $\boldsymbol{x}$ that lies on the optical axis is a special case that is currently not handled well by our algorithm. On-axis points are intersected by the central ray and all the rays exiting a circle of radius r in the light plane. While this is correct in theory, it is difficult in practice to implement the required summations using a programmable GPU. Also, the intensity correction described in the previous section scales intensity to infinity at such points. We currently sidestep the problem of on-axis points by perturbing such points by a small epsilon value.

3 Static lighting

A scene is statically lighted if the directions of the (directional) light sources are fixed with respect to the orientations of the caustic objects. The positions of the caustic objects need not remain constant, and their orientations may also change if the light directions change with them. Viewpoint and scene objects are completely unconstrained. Such a scene is rendered by first generating a set of CSLs for each caustic object in the scene, then rendering frames during which each CSL is evaluated for every pixel of every scene object.

3.1 Static CSL generation

We present our approach to CSL generation for statically lighted scenes in the spirit of the MPEG specification, which tightly specifies the decompression algorithm (as we do CSL evaluation), but allows ongoing innovation in compression techniques. To generate a set of CSLs for a single caustic object, we first cast a large number of rays in the light source direction, with a regular distribution over the silhouette region of the object. Each ray from the light source carries a unit of energy, which is reduced to model absorption and division as the ray's path is traced. If the object is refracting, we recursively trace reflected and refracted rays, terminating when a ray exits the object or drops in energy below a minimum threshold. The origin, direction, and energy of each reflected and exiting ray are saved. These caustic object rays provide a statistical definition of the light field that produces caustic patterns in the scene.

We take a two-step approach to organizing the caustic object rays into a set of CSLs: first clustering the rays into bundles; then parameterizing a CSL to represent each bundle.

A set of rays is a bundle if there exists an axis ray such that the distance between each ray and the axis ray is within a given threshold, and the direction of each ray is within 90° of the direction of the axis ray. We define the distance between two rays as the minimum distance between the lines that are colinear with the rays. It is obvious that there are many possible ray bundles, because each ray can be regarded as a ray bundle. To minimize the number of CSLs required to represent the object's caustic light field, we seek to identify large ray bundles. Given the distance threshold, the key to finding a large ray bundle is to identify a promising axis. Once the axis of a ray bundle is known, we can judge if a ray belongs to the bundle by testing its distance and angle against the bundle axis.

3.1.1 Bundle extraction

We used the following method to search for promising bundle axes and to construct the corresponding ray bundles:

1. *Rasterization*. Create an extended bounding volume around the object and uniformly subdivide the bounding volume into cells. Then, for each caustic object ray, determine the cells its colinear line passes through using a 3D DDA algorithm, and add the ray's energy to each cell of this type (Fig. 7).
2. *Bundle initialization*. Select the cell with the maximum energy, and determine all the rays passing through it. Then define a bundle axis whose origin is the mean of the rays' origins and whose direction is the mean of the rays' directions.
3. *Ray assignment*. For each caustic ray, add it to the bundle if its origin and direction with respect to those of the bundle axis are within the threshold amounts.
4. *Bundle optimization*. Optimize the location and direction of the bundle axis by minimizing their summed squared differences with respect to the rays in the bundle.

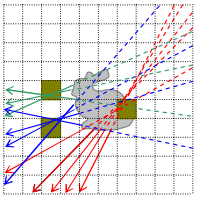

Fig. 7. Caustic ray rasterization. Caustic rays first rasterized into cells of an extended bounding box. The *filled cells* are those with maximum energy, based on intersections with (extended) caustic object rays. The *dashed lines* represent the extended part of the caustic rays

5. *Iteration*. Go back to step 3 to update the rays in the bundle, until the bundle axis location and direction converge or a maximum iteration number is reached.

The above procedure usually takes about 15 iterations, but we set 100 as the maximum iteration number. When a ray bundle is obtained, a CSL is used to approximate it. During the creation of the optimized CSL rays that cannot be approximated well are dismissed from the ray bundle. After the CSL is finalized the rays remaining in the ray bundle are removed from the set of caustic object rays. Then the above procedure is then repeated to identify another bundle axis, until the total energy of the remaining rays falls below a small threshold.

3.1.2 CSL parameterization

For each ray bundle, a CSL is initialized with its optical axis aligned with the ray bundle's axis. The origin of the CLS's light plane is positioned on the axis at point $\boldsymbol{c} = \boldsymbol{b} + \ell\hat{\boldsymbol{a}}$ using a scalar parameter ℓ, where $\boldsymbol{b}$ is the origin of the bundle axis as shown in Fig. 8. Then three parameters must be determined: the α and β parameters in $f(r)$, and ℓ.

Ray correspondence. As shown in Fig. 8, for each ray R_b in the bundle, a unique CSL ray (R) that corresponds to R_b can be found by the intersection point $\boldsymbol{p}$ of the (extended) bundle ray with the light plane. The following shows the procedure.

Let $\boldsymbol{q}$ and $\hat{\boldsymbol{d}}$, respectively, be the origin and direction of the bundle ray R_b, then the intersection point of the (extended) R_b and the CSL's light plane is: $\boldsymbol{p} = \boldsymbol{q} + \frac{(\boldsymbol{c}-\boldsymbol{q},\hat{\boldsymbol{a}})}{(\hat{\boldsymbol{d}},\hat{\boldsymbol{a}})}\hat{\boldsymbol{d}} = \boldsymbol{q} + \frac{(\boldsymbol{b}-\boldsymbol{q},\hat{\boldsymbol{a}})+\ell}{(\hat{\boldsymbol{d}},\hat{\boldsymbol{a}})}\hat{\boldsymbol{d}}$. And the radius (distance to the light plane origin) of point $\boldsymbol{p}$ is $r = \|\boldsymbol{p} - \boldsymbol{c}\|$.

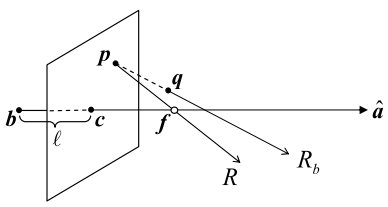

Fig. 8. Corresponding bundle ray (R_b) and CSL ray (R)

Point $\boldsymbol{p}$ gives R_b's corresponding CSL ray R, which extends from point $\boldsymbol{p}$ in the CSL. By CSL definition, ray R's intersection on the CSL axis is $\boldsymbol{f} = \boldsymbol{c} + f(r)\hat{\boldsymbol{a}}$. Then the direction of ray R is $\hat{\boldsymbol{e}} = \frac{\boldsymbol{f}-\boldsymbol{p}}{\|\boldsymbol{f}-\boldsymbol{p}\|}$.

An ideal CSL for a ray bundle is one for which the directions of all corresponding bundle rays and CSL rays are identical. This ideal correspondence is never achieved, however, so we formulate the above approximation as an optimization problem to minimize the difference between the corresponding ray subject to the CSL's parameters: α, β and ℓ.

Approximation error. There are several possible ways to measure how close two rays are. Since ray R and ray R_b share point $\boldsymbol{p}$ on the light plane, a simple measurement would be the angular/Euclidean difference of their direction vectors. In our experiments, we found that the following method measuring the distances of two pairs of points on ray R and ray R_b works best. For convenience, we denote $t_1 = \|\boldsymbol{p} - \boldsymbol{q}\|$ and $t_2 = t_1 + w$, where w is four times the bounding box's size of the caustic object. The point pair we take are: $\boldsymbol{p} + t_i\hat{\boldsymbol{d}}$ (on ray R_b) and $\boldsymbol{p} + t_i\hat{\boldsymbol{e}}$ (on ray R), for $i = 1, 2$.

Then, the approximation error regarding ray R_b is defined as:

$$E(R_b) = \sum_{i=1,2} \|(\boldsymbol{p} + t_i\hat{\boldsymbol{d}}) - (\boldsymbol{p} + t_i\hat{\boldsymbol{e}})\|^2 = \left(t_1^2 + t_2^2\right)\|\hat{\boldsymbol{d}} - \hat{\boldsymbol{e}}\|^2.$$

Note that t_1, t_2, and $\hat{\boldsymbol{e}}$ are all functions of the unknown CSL parameters (α, β and ℓ).

Therefore we minimize the following objective function to optimize the CSL:

$$\min_{\alpha,\beta,\ell} \sum_{all\ ray\ R_b} E(R_b). \tag{5}$$

This is a non-linear energy minimization problem, which we use Brent's method to solve (see Chap. 10.3 in [9]).

Intensity image generation. After optimizing the CSL parameters, the intensity of each bundle ray is recorded in the light plane intensity image. Each ray is treated as a cylinder of a certain radial size (denoted by δ), and its energy is summed into the image using EWA splatting [21]. The splatting ellipse is centered at $\boldsymbol{p}$, and aligned with the direction of $\boldsymbol{p} - \boldsymbol{c}$ in the light plane (see Fig. 8). The lengths of the ellipse axes are $\frac{\delta}{(\hat{\boldsymbol{d}},\hat{\boldsymbol{a}})}$ and δ. The splatting parameter δ takes three times the averaged distance value of the pre-computed caustic rays on the caustic objects, which is large enough to avoid the gaps in the intensity images.

3.2 Static CSL rendering

Once the caustic rays are approximated by a static set of CSLs, frames are rendered using the following algorithm:

```
1 Clear the frame buffer
2 For each caustic spot light
3     Initialize the pixel shader for the CSL
4     For each pixel's visible sample point x
5         Transform x's coordinates into the CSL's
6             local coordinate system
7         Evaluate the CSL (Sect. 2)
8         Compute the shading values
9         Update the frame buffer
```

The algorithm is multipass, with one complete rendering pass over all scene objects for each CSL in the scene (line 2). At the start of each pass the CPU initializes the pixel shader algorithm and texture memory to correspond to the needs of the current CSL (line 3). Then GPU rasterization (line 4) generates pixel sample points, which are shaded by the pixel shader algorithm (lines 5–8) and used to update the frame buffer (line 9). Refer to Sect. 2 for details on the evaluation of a CSL, which typically includes solving two quadratic equations to identify up to three intersecting rays, correcting the intensities of these rays for focusing distance with Eq. 4, and summing the results. Because the ray directions are known, the pixel shader can model light interactions with non-Lambertian surfaces. The complete pixel shader algorithm is small enough that it can be implemented with HLSL (High Level Shading Language) of DirectX 9.0 in GPUs. Because many potentially small lighting contributions are accumulated in the frame buffer, 16-bit floating-point color representation is used in our implementation.

4 Dynamic lighting

We extend our approach to support changing lighting direction by further dividing the pre-rendering phase into per-scene and per-frame phases. During the per-scene phase a complete set of static CSLs is generated for each of many regularly distributed lighting directions, corresponding to each vertex of a regularly subdivided icosahedron. Also during the per-scene phase, correspondences are established between CSLs in the sets that correspond to the three vertices of each icosahedron facet. Each CSL correspondence is a triple, involving one CSL from each of the three sets. These correspondences are not obvious, because the CSL sets for different illuminations of a caustic object typically have different numbers of CSLs.

At the start of each frame we first find the triangle t that the current light direction intersects on the subdivided icosahedron mesh, and compute the intersection point's barycentric coordinates in t. Then we create a set of CSLs for the current light direction by interpolation of t's CSL triples, using the barycentric coordinates as weights. The resulting set of CSLs is then used to render the frame, just as if it were a statically lighted scene.

4.1 CSL interpolation

Let c_0, c_1, and c_2 be the CSL triple from which CSL $\bar{c}$ is to be formed. The interpolation uses the barycentric coordinates of the triple, and is performed as follows:

1 Spherically interpolate the CSL axis directions.
2 Linearly interpolate the CSL light plane origins.
3 Linearly interpolate CSL parameters α and β in $f(r)$.
4 Linearly blend the intensity images.

The intensity image of each c_i is aligned with the axis of $\bar{c}$ before it is interpolated. Let $\hat{\boldsymbol{a}}$ be the (interpolated) axis of $\bar{c}$, and $\hat{\boldsymbol{a}}_i$ be the axis of c_i. A rotation matrix $\boldsymbol{R}$ that rotates $\hat{\boldsymbol{a}}_i$ to $\hat{\boldsymbol{a}}$ around the vector of $\hat{\boldsymbol{a}} \times \hat{\boldsymbol{a}}_i$ is uniquely determined, and is used to rotate c_i's light plane into $\bar{c}$'s light plane.

4.2 Matching CSLs

Forming CSL triples that can be successfully interpolated is a complicated task. We divide it into two subtasks:

- First, match compatible pairs of CSLs for each two neighboring directions that share an edge in the subdivided icosahedron triangle mesh.
- Then, use the pairwise correspondences to make CSL triples for each three neighboring directions that share a triangle in the subdivided icosahedron.

This section describes the matching of CSL pairs. For convenience, $\hat{\boldsymbol{s}}_0$ and $\hat{\boldsymbol{s}}_1$ denote two neighboring precomputed light directions, $G(\hat{\boldsymbol{s}})$ denotes the set of CSLs of light direction $\hat{\boldsymbol{s}}$, and $A(c)$ denotes the axis position and direction of a CSL c.

Our approach is to track the movement of CSL axes when the incident light direction $\hat{\boldsymbol{s}}_t$ $(0 \le t \le 1)$ gradually moves from $\hat{\boldsymbol{s}}_0$ to $\hat{\boldsymbol{s}}_1$, and then to evaluate the CSL correspondence by inspecting the axis distance between the moved CSLs of $\hat{\boldsymbol{s}}_0$ and the CSLs of $\hat{\boldsymbol{s}}_1$.

Starting with $t = 0$, the tracking process is as follows:

(1) $\hat{\boldsymbol{s}}_t$ is moved to $\hat{\boldsymbol{s}}_{t+\Delta t}$.
(2) A set of rays is traced from the light source in direction $\hat{\boldsymbol{s}}_{t+\Delta t}$.
(3) For each CSL $c \in G(\hat{\boldsymbol{s}}_0)$, the optical axis $A(c, t)$ is optimized to a new location and direction $A(c, t + \Delta t)$ using the same approach as steps 3–5 in Sect. 3.1.1.
(4) Steps (1) through (3) are repeated until $\hat{\boldsymbol{s}}_t$ arrives at $\hat{\boldsymbol{s}}_1$.

In our implementation $\hat{\boldsymbol{s}}_0$ takes 20 small steps of Δt to arrive at $\hat{\boldsymbol{s}}_1$. When the tracking stage is complete, the axis distance between each moved CSL of $\hat{\boldsymbol{s}}_0$ and each CSL of $\hat{\boldsymbol{s}}_1$ is computed. Let $A_i = (\boldsymbol{b}_i, \hat{\boldsymbol{a}}_i)$, $i = 0, 1$, be two axes, where $\boldsymbol{b}_i$ and $\hat{\boldsymbol{a}}_i$ are the position and direction of axis A_i.

We define the distance between A_0 and A_1 as follows:

$$\text{dist}(A_0, A_1) = \sqrt{\|\boldsymbol{b}_0 - \boldsymbol{b}_1\|^2 + B\|\hat{\boldsymbol{a}}_0 - \hat{\boldsymbol{a}}_1\|^2},$$

where B is the bounding sphere size of the object.

Finally the matched correspondence pairs are identified by inspecting the CSL pairs $\{(c_i, c_j) \mid c_i \in G(\hat{\boldsymbol{s}}_0), c_j \in G(\hat{\boldsymbol{s}}_1)\}$ in order of increasing axis distances $\text{dist}(A(c_i, 1), A(c_j))$. If either c_i or c_j has been previously matched with a CSL, or the distance is beyond a pre-specified threshold, the pair is skipped. Otherwise c_i is matched with c_j. Because $\hat{\boldsymbol{s}}_1$ may have fewer CSLs than $\hat{\boldsymbol{s}}_0$, and some correspondence pairs with large distances may have been skipped, it is possible that some CSLs of $\hat{\boldsymbol{s}}_0$ will be unmatched. For each unmatched CSL in $\hat{\boldsymbol{s}}_0$, a new CSL match is created in $\hat{\boldsymbol{s}}_1$, using the $\hat{\boldsymbol{s}}_0$ CSL's moved axis and a completely black intensity image.

4.3 Building CSL triples

The rules that guide the process of assembling CSL triples from matched pairs are: (1) that the CSLs in a triple are all matched with each other, (2) that triple interpolation results maintain continuity when the incident light direction crosses from one icosahedron triangle to another triangle, and (3) that interpolated results exactly match those of the original CSL set when the incident light direction is aligned with a vertex location.

Our approach is to propagate the pairwise correspondence on the edge of neighboring direction to the interior of the triangles. The propagation begins with a matched CSL pair (c_0, c_1) in $\hat{\boldsymbol{s}}_0$ and $\hat{\boldsymbol{s}}_1$. First the neighboring triangle t incident to edge $\langle \hat{\boldsymbol{s}}_0, \hat{\boldsymbol{s}}_1 \rangle$ is visited, and its third vertex $\hat{\boldsymbol{s}}_2$ is found. Then the CSLs in direction $\hat{\boldsymbol{s}}_2$ are found and matched with c_0 or c_1. In most cases there is only one such CSL c_2. If so, the triple (c_0, c_1, c_2) is formed in triangle t. If there is more than one CSL, one is arbitrarily picked to form the CSL triple in triangle t. To preserve energy c_0, c_1, and c_2 are marked so that they are not used in other triples of triangle t.

The propagation continues recursively with CSL pairs (c_0, c_2) and (c_1, c_2). During the recursive propagation it will infrequently be the case that all CSLs in direction $\hat{\boldsymbol{s}}_2$ that are matched with c_0 or c_1 are already included in CSL triples of triangle t. In such cases using c_2 to make a new triple for triangle t is inconsistent with the rule of energy conservation. Our solution is to use c_2 to construct a CSL triple for triangle t, but to label c_2 so that its intensity image is not added in later interpolations.

5 Results

We have implemented a CSL-based caustic rendering system in C++ with DirectX as the graphics API. The pixel shader program is about 50 lines of HLSL code, and vertex shader program just performs some trivial transformation tasks. In our current experiments we have typically subdivided the icosahedron twice to sample the light directions, resulting in 162 complete sets of static CSLs. Using a less subdivided icosahedron will lead to noticeable jumping artifacts, because the CSL interpolation is not reliable for faraway light directions. Increasing the sampling number significantly increases the memory cost for storing the CSL intensity images. Table 2 lists statistics and performance numbers for four caustic objects. The performance numbers are for 800×600-pixel images that were rendered on a PC using a GeForce 6800 graphics card. Almost all of the per-frame computation is done by the graphics card, so the performance of the CPU is inconsequential. The performance bottleneck is limited by the number of rendering passes required for the CSLs of the current light direction.

The per-scene setup cost consists mainly of ray tracing, CSL generation, and CSL tracking. The time required by these operations is linearly proportional to the number of direction samples and emitted rays, and was $4 \sim 8$ hours with different models in our experiments. For each light direction, 1280×1280 parallel rays were traced. The CSL intensity images were $N \times N$ pixels, with N chosen as 128, 256, or 512 depending on the actual size of the rectangle covered by the intensity image in the light plane. The images were quantized into 16-bit integer values. For each light direction, CSLs were generated until at least 95% of the ray traced energy was accounted for.

Table 2. Statistics for renderings with various caustic objects

	Head	Bunny	Skull	Bird
Vertices in the model	33*k*	35*k*	20*k*	48*k*
Average CSLs per light	7.96	8.50	7.56	8.52
Average triples per facet	10.48	10.55	9.70	11.96
Total image data (MB)	103	107	87	112
Performance (fps)	19	19	20	18

Fig. 9. Caustics rendered on a non-Lambertian surface. *Left* side shows image without view-dependent shading and *right* image with view-dependent shading

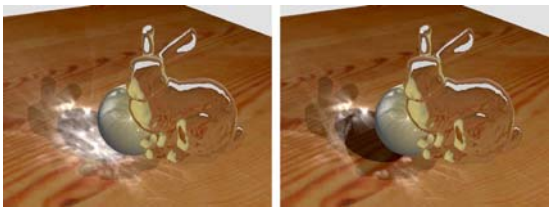

Fig. 10. Shadow effects in caustics. *Left* shows image without shadow, and *right* image with shadow

Examples of our caustic rendering results are presented in Figs. 1, 9, and 10. In these figures scene objects that are visible through caustic objects were rendered into an environment map centered at the caustic object. The map was then accessed using refraction directions as texture coordinates during the rendering of the caustic object itself. The double buffer technique in [17] could be used to improve the refraction effects of the caustic objects.

Figure 1 illustrates that our method can produce convincing caustic effects in real-time. Figure 9 illustrates the improvement in image quality that results from non-Lambertian lighting. Figure 10 demonstrates a rudimentary form of shadowing implemented with a simplified (spherical) occluder, for which the ray occlusion is analytically checked in the pixel shader program.

6 Conclusion and future work

We have described an algorithm that renders caustic light patterns by converting indirect light into caustic spot lights (CSLs) that directly illuminate objects in the scene. Our experimental results demonstrate that this approach generates convincing caustic patterns in real-time when accelerated by a modern programmable graphics card. The features and performance of the CSL algorithm compare favorably with those of other caustics rendering approaches.

In future work we hope to extend CSL shadowing to generalized occluders evaluation of CSLs, for points on the optical axis is currently not robust, and will be improved. Finally, though the CSL parameterization described in this paper works well for many caustic objects, it is not applicable for all object shapes. By the CSL's definition, it is suitable for parameterizing the caustic rays that focused on a moving point. As a result, the caustics refracted by sphere-like objects can be well rendered. But, the caustic rays refracted by a transparent cylinder cannot be approximated, because they are focused on a moving line rather than on a moving point. We will experiment with other parameterizations to address this problem.

Acknowledgement We thank Kurt Akeley for insights and suggestions and for help in revising the paper.

References

1. Brière, N., Poulin, P.: Adaptive representation of specular light flux. In: Graphics Interface, pp. 127–136. Canadian Human-Computer Communications Society, Montréal, Québec (2000)
2. Ernst, M., Akenine-Möller, T., Jensen, H.W.: Interactive rendering of caustics using interpolated warped volumes. In: Graphics Interface, pp. 87–96. Canadian Human-Computer Communications Society, Victoria, British Columbia (2005)
3. Guenther, J., Wald, I., Slusallek, P.: Realtime caustics using distributed photon mapping. In: Eurographics Symposium on Rendering, pp. 111–121. Eurographics Association, Norrköping (2004)
4. Heckbert, P.S., Hanrahan, P.: Beam tracing polygonal objects. In: SIGGRAPH, vol. 18, pp. 119–127. ACM Press, New York, NY (1984)
5. Iwasaki, K., Dobashi, Y., Nishita, T.: A fast rendering method for refractive and reflective caustics due to water surfaces. Comput. Graph. Forum **22**(3), 601–609 (2003)
6. Jensen, H.W.: Rendering caustics on non-Lambertian surfaces. Comput. Graph. Forum **16**(1), 57–64 (1997)
7. Jensen, H.W.: Realistic Image Synthesis Using Photon Mapping. Peters, Wellesley, MA (2001)
8. Nishita, T., Nakamae, E.: Method of displaying optical effects within water using accumulation buffer. In: SIGGRAPH '94, pp. 373–379. ACM Press, Orlando, FL (1994)
9. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes in C: The Art of Scientific Computing, 2nd edn. Cambridge University Press, Cambridge (1993)
10. Purcell, T.J., Donner, C., Cammarano, M., Jensen, H.W., Hanrahan, P.: Photon mapping on programmable graphics hardware. In: Graphics Hardware 2003, pp. 41–50. Eurographics Association, San Diego, CA (2003)
11. Shah, M.A., Konttinen, J.: Caustics mapping: An image-space technique for real-time caustics. IEEE Trans. Vis. Comput. Graph. **13**(2), 272–280 (2007)
12. Shirley, P.: Realistic Ray Tracing. Peters, Wellesley, MA (2000)
13. Sloan, P., Kautz, J., Snyder, J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In: SIGGRAPH '02, pp. 527–536. ACM Press, San Antonio, TX (2002)
14. Stam, J.: Random caustics: Natural textures and wave theory revisited. In: ACM SIGGRAPH Visual Proceedings, pp. 150–150. ACM Press, New Orleans, LA (1996)
15. Stam, J.: Aperiodic texture mapping. Tech. Rep. R046, ERCIM Research (1997)
16. Wand, M., Straßer, W.: Real-time caustics. Comput. Graph. Forum **22**(3), 611–620 (2003)
17. Wyman, C.: An approximate image-space approach for interactive refraction. In: ACM SIGGRAPH, pp. 1050–1053. ACM Press, Los Angeles, CA (2005)
18. Wyman, C., Davis, S.: Interactive image-space techniques for approximating caustics. In: I3D, pp. 153–160. ACM Press, Redwood City, CA (2006)
19. Wyman, C., Hansen, C.D., Shirley, P.: Interactive caustics using local precomputed irradiance. In: Pacific Conference on Computer Graphics and Applications, pp. 143–151. IEEE Computer Society, Seoul (2004)
20. Yu, X., Li, F., Yu, J.: Image-space caustics and curvatures. In: Pacific Graphics, pp. 181–188. IEEE Computer Society, Maui, Hawaii (2007)
21. Zwicker, M., Pfister, H., van Baar, J., Gross, M.: EWA splatting. IEEE Trans. Vis. Comput. Graph. **8**(3), 223–238 (2002)




XINGUO LIU is a professor of the Computer Science School in Zhejiang University. His research interests include geometry processing, appearance modeling, real-time rendering, and deformable objects. He received a B.Sc. in 1995 and a Ph.D. in 2001 from Zhejiang University. He was a researcher at the Internet Graphics Group of Microsoft Research Asia from 2001 to 2006.

ZHAO DONG is a Ph.D. candidate in the Computer Graphics Group of Max-Planck-Institut fuer Informatik. His research interests include global illumination, real-time rendering, and volume graphics. He received a B.S. degree in 2001 and a M.S. degree in 2005 from Zhejiang University.

HUJUN BAO is a professor of the Computer Science School in Zhejiang University, and is the director of State Key Laboratory of CAD&CG in Zhejiang University. His research interests include modeling and rendering techniques for large scale virtual environments and their applications. He is also the principal investigator of the virtual reality project sponsored by Ministry of Science and Technology of China. He received his Bachelor degree and Ph.D. in applied mathematics from Zhejiang University in 1987 and 1993.

QUNSHENG PENG is a Professor of Computer Graphics at Zhejiang University. His research interests include realistic image synthesis, computer animation, scientific data visualization, virtual reality, and bio-molecule modeling. Prof. Peng graduated from Beijing Mechanical College in 1970 and received a Ph.D. from the Department of Computing Studies, University of East Anglia in 1983. He currently serves as a member of the editorial boards of several international and Chinese journals.