Computers & Graphics 98 (2021) 255–267

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Large-scale ray traced water caustics in real-time using cascaded caustic maps ☆

Gerasimos Kougianos, Konstantinos Moustakas*

Electrical and Computer Engineering Department, University of Patras, Greece

ARTICLE INFO

Article history:
Received 19 January 2021
Revised 24 May 2021
Accepted 21 June 2021
Available online 22 June 2021

Keywords:
Three-Dimensional graphics and realism
Caustics
Ray tracing
Real-Time rendering
Photon mapping

ABSTRACT

Achieving interactivity for applications with physically-accurate caustics is proven to be a challenging task. We present a hybrid method utilizing ray tracing and rasterization which enables water caustic coverage to vast view distance in real-time. Inspired by photon mapping, we optimize the generation of photons using cascaded caustic maps to avoid tracing the first bounce of a ray from a light, replacing that step with rasterization. We introduce cascades as a set of caustic maps with varying resolution based on the distance from the viewer. In addition, since we adopt a splatting approach where each photon is rasterized into the image based on the extent of its contribution, we trace photon differentials in order to determine the size, shape and intensity of the splats so as to achieve adaptive anisotropic flux density estimation. Finally, to mitigate undersampled regions where lack of photons leads to noise, we propose the use of a cross-bilateral filter with an adaptive kernel radius. The radius is based on the perceived radiant energy of a photon relative to the scene's luminance, and drastically improves performance. We demonstrate how the use of our method is able to perform interactive rendering of large-scale dynamic water caustics.

1. Introduction

Caustics are an important visual phenomenon of indirect illumination contributing to the perceived realism of scenes involving reflective and refractive surfaces. Caustics occur when light rays from a source get refracted or reflected and converge at a single point on a non-shiny surface. This creates the non-uniform distribution of bright and dark areas. Off-line high quality rendering systems for rendering indirectly visible caustics in a Monte Carlo rendering framework use photon mapping and its variants [1,2,3]. However, real-time physically-accurate caustics remain open to more practical solutions.

More recently, with the advent of efficient frameworks for ray tracing, along with the supporting implementation of acceleration data structures (ADS), researchers developed state-of-the-art hybrid methods for simulating complex light transport phenomena in interactive scenes [4,5,6,7]. The combination of raster and ray tracing techniques is utilised for stabilising a performance-quality balance. Despite various ADS readjustment strategies for animated data, interactivity becomes problematic, in general, when geometry is dynamically computed, tessellated or frequently topologically changed (e.g. water), since the ADS has to be rebuilt from scratch. Deviating from the standard photon mapping paradigm, implementations propose to generate a caustic map, which is effectively a G-buffer of uniform samples of visible surfaces as seen from a light and contains all surface properties needed to generate rays for the initial bounce of the photons [8,9]. Following photon tracing, the expensive photon density estimation step of photon mapping is commonly replaced by a raster-based approach where energy from each photon is distributed to affected points directly visible to the camera, using a pre-calculated radius of influence. When primitive rasterization is used for covering the area of support of the kernel function in image space, the process is referred to as photon splatting. A third rasterization technique commonly integrated into these pipelines as a post-process effect employs image-filtering algorithms, which improve noisy results due to the low sample count.

The motivation behind our work is the extension of these hybrid methods in order to optimize performance without degrading quality. Since the density and accuracy of caustics depend on the resolution of the first-bounce caustic map, photons within large scenes cannot be rendered without the loss of performance or an

☆ This article was recommended for publication by M Doggett.
* Corresponding author.
E-mail address: moustakas@upatras.gr (K. Moustakas).

https://doi.org/10.1016/j.cag.2021.06.008

impact on realism. We enable fast first-bounce photon estimation of photons by using cascaded caustic maps, a set of caustic maps with decreasing resolution based on the distance from the viewer. We purely focus on caustics from water surfaces, which can be large and highly dynamic, providing a challenge to previous work which do not use cascades. Additionally, we exploit RTX graphics hardware to trace photon differentials in order to adjust the splat kernel size, providing more accurate caustics. The filtering process is extended by adapting the radius of a cross-bilateral filter based on the perceived brightness of the human eye. Our work follows the approach of [6], and it's main contributions over the state of the art can be summarized as follows:

- We introduce cascaded caustic maps to enable interactive accurate caustics rendering of vast scene regions with minimum performance drop.
- We incorporate photon differentials to improve photon flux density estimation and reduce noise and blur.
- We extend the denoiser in [6] and [10] with an adaptive radius based on perceived luminance, improving the filtering process drastically.

The remainder of this paper is organized as follows: a survey of related work is presented in Section 2. Our rendering algorithm is then explained in Section 3 with sufficient detail so as to make the paper reproducible, followed by Section 4 discussing implementation details and experiment results. We conclude with a summary of the ideas presented in the paper, limitations, and provide directions for future research in Section 5.

2. Related work

Numerous solutions for rendering high quality caustics in a Monte Carlo rendering framework which extend the path tracing algorithm [11] have been proposed in the past. Metropolis light transport methods uses the Metropolis-Hastings algorithm to sample one caustic path with substantial contribution and can efficiently explore the entire caustic via mutation [12,13,14]. An alternative approach denoted as path guiding utilizes machine learning in order to train a distribution used to choose more samples in important directions in an online process [15,16,17]. However, guiding does not help if the set of techniques is insufficient. Bidirectional path tracing [18] which connects photon paths from the light to paths originating from the camera can be combined with radiance estimates from photon mapping [1] and is the basis of Vertex Connection and Merging (VCM) algorithms [19,20,21]. As a path is traced from the camera, photon map lookups are performed at each intersection point (i.e. camera path vertex). The photon mapping technique is essential for rendering indirectly visible caustics, such as the ones seen at the bottom of a swimming pool. While these Monte Carlo based techniques render accurate caustics, they are unsuitable for interactive applications.

Authors in [22] proposed emitting rays from light sources and accumulating them in illumination maps, but required the need for per-object illumination texture. Others [1] eliminated this need and also render other global illumination effects. Photon mapping uses a kD-tree to accelerate final gathering from stored photons which require an expensive $O(n\log(n))$ build cost. Researchers in [23] and [24] have enabled dynamic kD-tree creation and traversal using a GPU, provided however an insufficient frame rate for dynamic caustics rendering. Researchers In [25], the researcher adapted a uniform grid approach based on photon sorting on the GPU which is found to be efficient for photon gathering but used 1 photon per cell, hence losing information. Various methods for gathering and distributing photon flux by optimising a voxel hashing scheme implemented on a GPU for searching nearest photons have been examined in [26], but the methods for caustics do not scale well to higher resolutions.

In order to avoid the photon density estimation step [27,28], which happens to be the performance bottleneck in an interactive photon mapping context, implementations are based on the inverse operation, scattering, and take advantage of rasterization hardware to efficiently draw the splatting kernels [29,30,5]. While photon splatting is faster than standard photon mapping, the main problem is to find an appropriate splat size (bandwidth) and photon count. The caustic map creation pass typically controls lighting cost and quality. By increasing the photon count, quality improves, but splatting each one into the caustic map quickly becomes the bottleneck. Researchers in [31] and [32] experimented with different photon counts, but because quality does not improve linearly as sampling increases, the millions of photons needed to interactively generate crisp, noise-free results remained infeasible. Adaptive splat sizes based on divergence was proposed by [33], which reduced undersampling noise but slowed rendering. The more recent work of [30] builds view-sample cluster hierarchies and splats photons by traversing the data structure. This allows radiance estimates in interactive settings where photons are few enough to be traced per frame, and large enough to provide a smooth result, however on a small number of large photons, high frequency phenomena, like caustics, are poorly represented. Authors in [5] scale the splatting kernel proportionally to the length of the ray, essentially treating the photon as traveling along a cone instead of a ray and factoring in the growth of the cone base as its height increases. This method is an approximation of the photon differential. In this work we use photon differentials to adapt both the size and the shape of the splats to the structure of the light after it has interacted with specular surfaces. Specifically, we trace photon beams that change shape and size as they travel through a scene according to the concept of ray differentials [34]. The result is a method for rendering caustics that offers improved density estimation while it also retains the speed of the splatting approach.

Screen-space photon mapping has been introduced in [4], where photons are traced in world space and stored as texels in screen space. Instead of scattering photon contribution to neighboring pixels, each photon is compressed into a single pixel, leading to noisy results. The image is then filtered by using the reflection denoiser from NVIDIA GamesWorks Ray Tracing [35], by treating caustics as reflections. However artifacts are still produced, since the footprint of pixels on close geometry is large, and the low number of photons in such pixels causes high variance, photons do not accumulate smoothly when the camera approaches the caustic surfaces. Our adaptive radius kernel is able to effectively blur individual caustics in this case. Other researchers [10] and [6] use a spatial bilateral filter with a fixed radius as a denoising pass. While it effectively blurs undersampled regions of caustics, we extend their approach with an adaptive kernel radius based on the perceived radiant energy of the photon relative to the scene's luminance in order to efficiently filter with less passes.

Recently, researchers [6] proposed a method based on caustic mapping for interactive ray traced water caustics using DXR. This method forms the basis of our approach. A reflected/refracted water mesh is constructed by tracing incident light from the vertices of the water mesh. The intersection points are then projected to screen space, and radiance is accumulated for that location in a buffer. The accumulated radiance is scaled by a compression ratio which describes the convergence of caustics. Since the compression ratio depends on the detail of the water mesh geometry, errors can be introduced when the water surface is not refined enough. We extend this work by tracing photon differentials, instead of pure rays, in order to improve photon flux density estima-

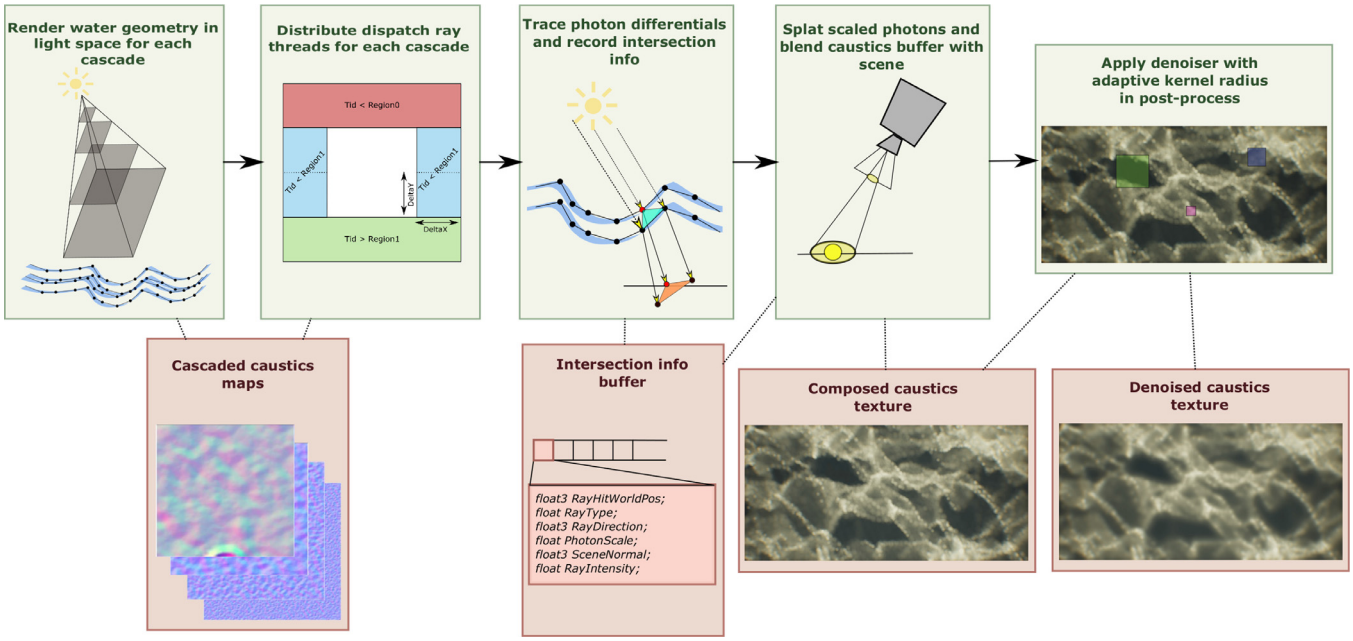


Fig. 1. The caustics algorithm pipeline. Cascaded caustic maps generated in the first step contain multi-resolution water position and normals G-buffers and are used during photon differential tracing in order to avoid tracing the costly first hit from the light source to the water surface. Before emission, dispatch ray threads are distributed for each cascade level. Upon tracing, a hit buffer records intersection information for each photon. This information is used to scale photon sprites during splatting. The composited caustics texture is ultimately filtered by our adaptive radius denoiser. The colored boxes in the top-right image represent kernels with varying sizes based on the perceived brightness of the current pixel.

tion. We also introduce cascaded caustic maps to enable rendering caustics at large scale scenes, which is othrewise prohibitively expensive. Lastly, we improve the denoiser by using an adaptive kernel radius based on the perceived radiant energy of the photon relative to the scene's luminance.

3. Water caustics algorithm

In this section, we initially describe the algorithm from an architectural perspective, since it is based on [6], and continue by analysing our contributions.

The method for rendering water caustics involves mainly two sets of buffers: caustic maps that store rasterized water geometry in light-view, and caustics buffers that accumulate photon footprints in screen space. The workflow is depicted in Fig. 1 and consists of five steps:

- Render water surface into cascaded caustic maps from the light's point of view, recording the positions and normals of the water surface.
- Distribute ray threads for dispatch for each cascade by exploiting overlapping geometry regions due to scaled projection mappings.
- Emit photon differentials from positions recorded in the caustic maps and trace them along the reflected or refracted directions calculated from the surface normals. Once the photon differentials intersect the scene, record the information of the hit points.
- Render caustics into the caustics buffer, which is placed in screen space, using the data of the hit points from the previous step.
- Perform denoising using an adaptive radius kernel on the caustics buffer and composite the result with the scene.

3.1. Cascaded caustic maps

We introduce cascaded caustic maps (CCMs), an analog to cascaded shadow maps [36], to enable interactive rendering of caustics caused by large water bodies. CCMs are a set of caustic maps with varying resolution. Different areas of the camera frustum require caustic maps with different resolutions. Objects nearest the eye require higher resolution than do more distant objects. The basic idea of CCMs is to partition the frustum into multiple frusta. A caustic map is rendered for each subfrustum; the ray gen shader then samples from the map that matches the required resolution.

We assume directional lights, but CCMs can also be applied to other light types without loss of generality. Each directional light stores properties for it's CCMs including:

- the number of CCM levels N,
- the CCM level scale factor s,
- the CCM precision p,
- the CCM view size of the inner-most cascade ccm_0

During the first step of the workflow, we define the view and projection matrices of the directional light for each cascade. Since we want all the visible objects to fit into the light view projection matrix, we need to fit the view frustum into the light frustum. To achieve this we need to set the position of the light. That position and its direction will be used to construct the light view matrix. In order to calculate the position of the light $\boldsymbol{l_{pos}}$, we start from the origin of the camera o. We then go back to the direction of light $\boldsymbol{l_{dir}}$ an amount equal to the distance between the near and far z planes of the view frustum:

$$\boldsymbol{l_{pos}} = o - \boldsymbol{l_{dir}} * (z_{near} + z_{far}) \quad (1)$$

The view matrix of the light is independent of the current cascade level. However, the projection matrix of the light depends on the range vector of the light r which is scaled by the current cascade

- 根据主摄像机的各种视口参数，求出视口八个顶点在世界空间的位置。
- 将八个顶点转换到光源摄像机的视口空间中，求出能包含这八个顶点的包围盒。这个包围盒即为光源相机的正交平截头体（平行光的光源相机是个正交相机，视口其实是个方的六面体）。
- 将计算出的光源相机的正交平截头体转回世界空间，光源相机的位置为正交平截头体近平面的中心，朝向和光源相同，视口长宽远近等参数根据视口空间的数据也很容易求出。有了这些参数即可唯一确定光源相机。

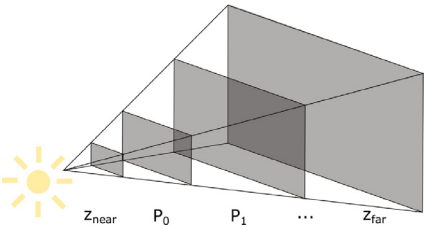


Fig. 2. As the cascade level increases, the extents of the projection matrix increases as well, forming different projection mappings for each level.

level:

$$\boldsymbol{P_i} = \begin{bmatrix} \frac{2}{r.width \cdot s^i} & 0 & 0 & 0 \\ 0 & \frac{2}{r.height \cdot s^i} & 0 & 0 \\ 0 & 0 & \frac{1}{r.far - r.near} & 0 \\ 0 & 0 & \frac{-r.near}{r.far - r.near} & 1 \end{bmatrix}, \quad \forall i \in [0, N-1] \quad (2)$$

The width and height extents of the projection matrix grows as the cascade level goes deeper, ultimately expanding the resolution, indicating frustum partitioning. Fig. 2 depicts this process for different cascade levels.

Rasterization of the water surface from the perspective of the light produces two CCMs. The first stores the positions in the scene and the second stores the geometry normals. Essentially, these are G-Buffers [37], albeit multi-layered, which will determine the first bounce of photons during ray-tracing. The CCMs are implemented as texture arrays with each array slice holding a separate cascade.

Prior to the ray tracing pass, the algorithm initially accumulates ray threads to be dispatched for each cascade level. The inner-most cascade size ccm_0 defines the cardinality of ray threads dispatched for the first level and is specified by the following ratio:

$$ccm_0 = \begin{cases} (256, 256), & \text{if } \frac{r_{max}}{p} < 256 \\ (512, 512), & \text{if } \frac{r_{max}}{p} < 512 \\ (1024, 1024), & \text{if } \frac{r_{max}}{p} < 1024 \\ (2048, 2048), & \text{otherwise} \end{cases} \quad (3)$$

where $r_{\max} = \max(r.width, r.height)$, r is the range component of the directional light, and the resulting sizes are experimentally defined based on the screen resolution. The number of threads dispatched for the remaining cascades are calculated as:

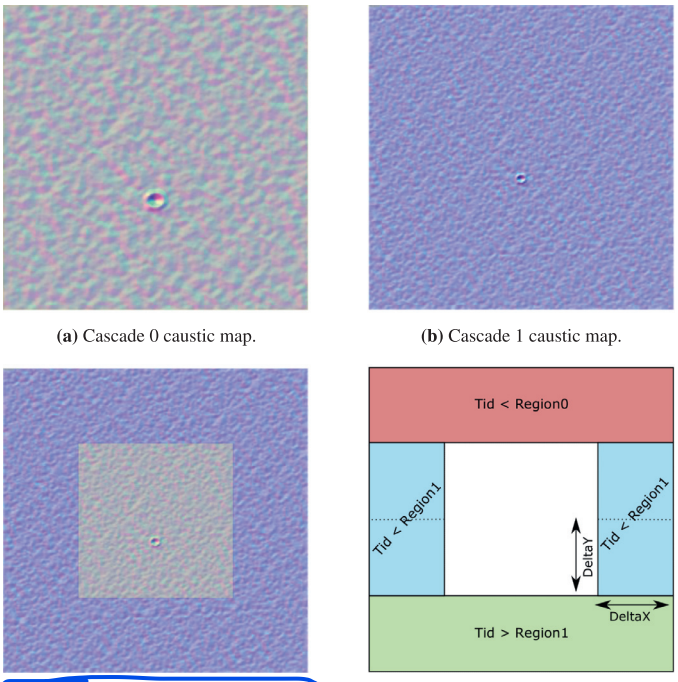
$$ccm_i = ccm_0 \cdot (1 - (\frac{1}{s^2})), \quad \forall i \in [1, N-1] \quad (4)$$

and the total accumulated ray threads for dispatch is thus:

$$ccm_{acc} = \sum_{i=0}^{N} ccm_i \quad (5)$$

The number of ray threads for the remaining cascades are less than ccm_0. The reason for this is simple: the projection matrices which are utilized to generate the CCMs depend on the CCM scale factor s which can be seen as the amount of "zoom-out" as the CCM level goes deeper. If we appropriately scale, align and stack the cascades as shown in Fig. 3c, we observe that a central region of deeper cascades is overlapped by the previous one, and the distant, unseen geometry is basically around the borders of the caustic map. Therefore, when distributing ray threads for maps $i \neq 0$, we leave an empty space in the center, where threads are not dispatched, since the space is already covered by previous cascades due to the "zoom-out" effect from the scaled projection matrices. The thickness of the borders is adjusted by *DeltaX* and *DeltaY* which depend on the same scale factor s as the projection matrices:

$$DeltaX = \frac{ccm_0.x \cdot (1 - \frac{1}{s})}{2}, \quad DeltaY = \frac{ccm_0.y \cdot (1 - \frac{1}{s})}{2} \quad (6)$$



(a) Cascade 0 caustic map.

(b) Cascade 1 caustic map.

(c) Cascade 1 scaled by $\frac{1}{s}$ and stacked onto cascade 0 demonstrating the "zoom-out" effect achieved by the scaled projection matrices.

(d) Distribution of dispatched ray threads for cascades $1 \ldots N$.

Fig. 3. Relation between the cascaded caustic map generated from the scaled projection matrices and the distribution of dispatched ray threads for the outermost CCM levels $1 \ldots N$ (here $N = 2$). The distribution exploits the "zoom-out" effect from the scaled projection matrices, by leaving an empty space in the center, where threads are not dispatched since it is already covered by previous cascades.

The innermost cascade level 0 does not leave an empty space and ray threads are dispatched uniformly since no previous cascade covers its region. Fig. 3 depicts the main idea and the ray thread distribution procedure is laid out in Algorithm 1. Refer to Appendix A for implementation details.

Depending on the depth of the CCM, the shader scales the intensity of the light by the CCM level scale factor as so:

$$L = s^{2 \cdot c} \quad (7)$$

where c is the CCM level of the current dispatch thread and L the light's intensity. The above formula is inspired by the literature of photon mapping density estimation, adjusted to our case of cascades. Photon mapping computes the density by dividing by the area. Since the cascades repeatedly shrink the area by s^2, we obtain a density estimation of $s^{2 \cdot c}$ overall. Intuitively, this means that the deeper the CCM level is, the brighter the intensity. This process is used to mitigate faded caustics at a large depth.

In order to query the positions and normals from the CCMs, the implementation samples them by mapping the new aligned dispatch thread coordinates outputted from Algorithm 1 to UV pixel coordinates and uses the output current cascade level c to specify the array slice of the texture array.

Fig. 4 depicts three cascades for a directional light. The range is set to $r.width = r.height = 500$, the CCM scale factor is $s = 4$ and the screen resolution is 1920×1080. We deliberately set the range of the light very narrow in order to illustrate the varying resolution at each cascade border.

3.2. Emitting, tracing and rendering photon differentials

As opposed to [6], we emit photon differentials [38] for each pixel in the CCM that represents a valid point on the water sur-

Algorithm 1 Distribution of CCM dispatch ray threads.

Require: $Tid, ccm[N], s$
Ensure: $NewTid, c$ {Level 0 cascades leaves no empty space}
if $Tid < ccm[0].x \times ccm[0].y$ **then**
  $c \leftarrow 0$
  $NewTid.x \leftarrow Tid \mod ccm[0].x$
  $NewTid.y \leftarrow \frac{Tid}{ccm[0].x}$
  return $NewTid, c$
end if {Delta factors scaled by CCM scale factor s}
$DeltaX \leftarrow 0.5 \times ccm[0].x \times (1.0 - \frac{1}{s})$
$DeltaY \leftarrow 0.5 \times ccm[0].y \times (1.0 - \frac{1}{s})$ {Figure 3*d* depicts regions}
$Region0 \leftarrow ccm[0].x \times DeltaY$
$Region1 \leftarrow Region0 + 2 \times DeltaX \times (ccm[0].y - 2 \times DeltaY)$
$bound[0] \leftarrow ccm[0].x \times ccm[0].y$
for $i = 1$ **to** $N - 1$ **do**
  $bound[i] \leftarrow bound[i-1] + ccm[i].x \times ccm[i].y$
end for
$LocalTid \leftarrow 0$
for $i = 0$ **to** $N - 1$ **do**
  if $Tid < bound[i+1]$ **then**
    $c \leftarrow i + 1$
    $LocalTid = Tid - bound[i]$
    if $LocalTid < Region0$ **then**
      $NewTid.x \leftarrow LocalTid \mod ccm[0].x$
      $NewTid.y \leftarrow \frac{LocalTid}{ccm[0].x}$
    else if $LocalTid < Region1$ **then**
      $LocalTid \leftarrow LocalTid - Region1$
      $NewTid.x \leftarrow LocalTid \mod (2 \times DeltaX)$
      if $NewTid.x > DeltaX$ **then**
        $NewTid.x \leftarrow NewTid.x + (ccm[0].x - 2 \times DeltaX)$
      else
        $NewTid.x \leftarrow NewTid.x$
      end if
      $NewTid.y \leftarrow DeltaY + \frac{LocalTid}{2 \times DeltaX}$
    else
      $LocalTid \leftarrow LocalTid - Region0$
      $NewTid.x \leftarrow LocalTid \mod ccm[0].x$
      $NewTid.y \leftarrow (ccm[0].y - DeltaY) + \frac{LocalTid}{ccm[0].x}$
    end if
    return $NewTid, c$
  end if
end for

face. Since we treat ray hit points as photon footprints and render them as decal sprites against scene depth, a fixed size leaves gaps or overlaps in between footprints. Photon differentials help determine the size and shape of the splats such that we achieve adaptive anisotropic flux density estimation in photon splatting.

Let us model a photon ray by the parameterization of a straight line:

$$r(t) = x + t \cdot \boldsymbol{\omega} \tag{8}$$

with $t \in [0, \infty)$, x being the photon origin and $\boldsymbol{\omega}$ the normalized photon direction. The differential of r is the partial derivative of its position and direction with respect to some initial offset and consists of four vectors:

$$\left\{ \frac{\partial x}{\partial u}, \frac{\partial x}{\partial v}, \frac{\partial \boldsymbol{\omega}}{\partial u}, \frac{\partial \boldsymbol{\omega}}{\partial v} \right\} \tag{9}$$

These derivatives describe the spread of the photon beam as it is traced through a scene. The positional derivatives describe its relative size at a given position, while the directional derivatives give the rate and direction of change of the photon beams spread.

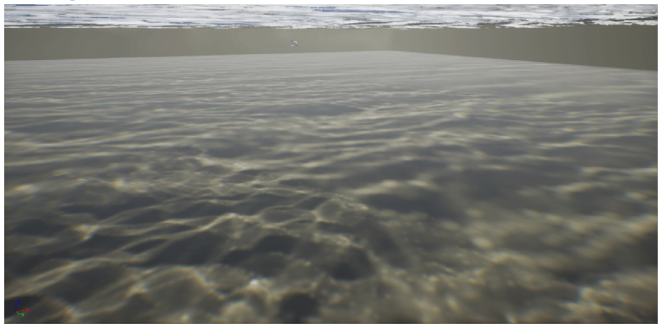


Fig. 4. Caustics created from a CCM with three cascades depicting the decreasing resolution, as the distance from the camera increases. The selected range is purely for demonstration purposes - in a normal scenario, the next cascade wouldn't be this close to the camera.

Since in our case we have G-buffers available at our disposal due to the generated CCMs, we exploit this fact and describe how the ray differential is defined when emitting the first reflection/refraction ray from the position in the G-buffer. The idea of this method is simply to access the adjacent pixels to the right and above the current pixel within the G-buffer and create a ray differential from these values. The positions x and the normals $\boldsymbol{n}$, for the current pixel (u, v) and for the neighbors $(u+1, v)$ and $(u, v+1)$, are sampled from the G-buffer during the ray gen pass. If we denote the sampled position values as $x_{0:0}$ for the current pixel, $x_{+1:0}$ for the pixel to the right, and $x_{0:+1}$ for the pixel above, then the triplet $(x_{0:0}, x_{+1:0}, x_{0:+1})$ forms a triangle with area $A_{x\boldsymbol{\omega}}$, which in the context of water caustics we will note as a surface triangle.

We can compute the positional ray differentials of the ray origin at the first hit as:

$$\frac{\partial x}{\partial u} = x_{+1:0} - x_{0:0} \quad \text{and} \quad \frac{\partial x}{\partial v} = x_{0:+1} - x_{0:0} \tag{10}$$

When a photon ray intersects an object its positional differential vectors are projected down onto the tangential surface of the object at the intersection point. Thus if we let t' denote the distance to the first intersection along the reflected or refracted photon ray on the scene geometry, then the same value t' is used to calculate the distance along the rays with origins $x_{+1:0}$ and $x_{0:+1}$. The projected positional differential vectors $\frac{\partial x}{\partial u}'$ and $\frac{\partial x}{\partial v}'$ form a caustics triangle $A'_{x\boldsymbol{\omega}}$ with a possibly different area than $A_{x\boldsymbol{\omega}}$ which depends on convergence or divergence of the light beam. Fig. 5 depicts the process of tracing a photon differential for refraction.

The scale correction factor of the photon footprint is calculated using the information from the traced photon differentials. Specifically it is calculated through

$$\sigma = \sqrt{\frac{A'_{x\boldsymbol{\omega}}}{A_{x\boldsymbol{\omega}}}} \cdot s^c \tag{11}$$

where ratio $\frac{A'_{x\boldsymbol{\omega}}}{A_{x\boldsymbol{\omega}}}$ is called the compression ratio as defined in [39], s is the CCM level scale factor and c is the CCM level of the currently dispatched photon. The CCM scale factor is used to mitigate faded caustics at a large depth, as previously described for Equation (7). To the author's knowledge, the compression ratio hasn't been combined with a cascade scale factor before. The procedure is laid out in Algorithm 2. Refer to Appendix B for implementation details.

The kernel size correction factor σ is stored to an off-screen buffer, along with other information about the hit, such as hit position, hit normal, ray direction, ray type (reflection/refraction) and ray intensity.

The ray intensity R is defined as:

$$R = L \cdot I_{att} \cdot F \tag{12}$$

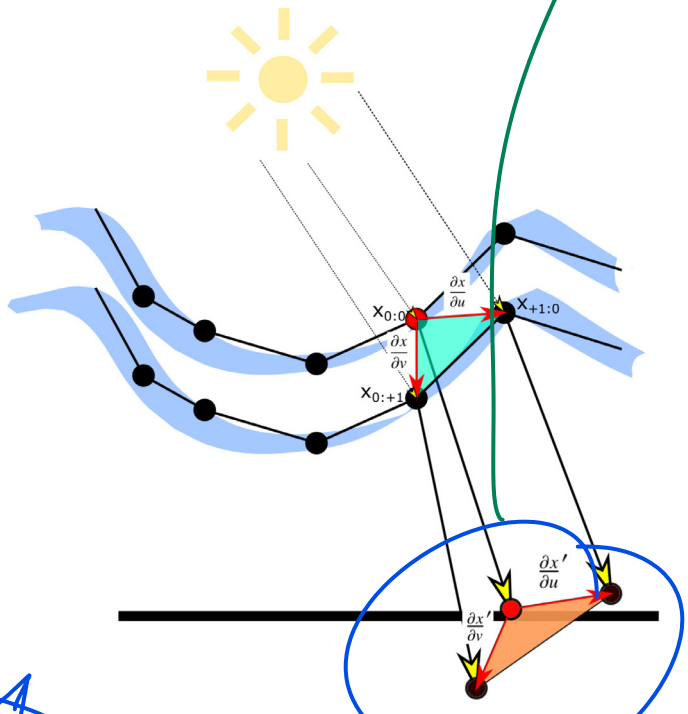


Fig. 5. A traced photon differential for a refracted ray. The positions of the water surface (black nodes) and the respective normals are sampled from the CCMs. The red node denotes the current sampled position, while the two adjacent red vectors represent it's positional differential vectors. The cyan triangle forms a surface triangle $A_{x\omega}$. By tracing the photon differential via refraction and using the current ray's hit distance for all, a caustic triangle $A'_{x\omega}$ is formed (orange triangle). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Algorithm 2 Kernel size calculation via photon differentials.

Require: $Ray, LDir, x00, x10, x01, n10, n01, Payload, s, c$
Ensure: $KernelSize$
$RayDirRight \leftarrow normalize(reflect(LDir, n10))$
$RayDirUp \leftarrow normalize(reflect(LDir, n01))$
$Hit0 \leftarrow x00 + Payload.HitT \times Ray.Dir$
$Hit1 \leftarrow x10 + Payload.HitT \times RayDirRight$
$Hit2 \leftarrow x01 + Payload.HitT \times RayDirUp$
$KernelSize \leftarrow \sqrt{\frac{TriArea(Hit0,Hit1,Hit2)}{TriArea(x00,x10,x01)}} \times s^c$ {Eq. 11}
return $KernelSize$

where L is defined in Equation (7), I_{att} is the intensity attenuation and F is the Fresnel term. I_{att} is proportional to the distance along the ray the hit point is from the origin:

$$I_{att} = \begin{cases} 1 - ((1 - \frac{t'}{tMax}) - \gamma)/(1 - \gamma), & \text{if } (1 - \frac{t'}{tMax}) > \gamma \\ 1 - (\gamma - (1 - \frac{t'}{tMax}))/\gamma, & \text{otherwise} \end{cases} \quad (13)$$

where t' is the distance along the ray where the intersection occurred, $tMax$ is the maximum valid ray distance, and γ is an experimentally defined constant set to 0.95 which represents the focus bias. This proposed formula attempts to simulate underwater intensity attenuation, and exploits the maximum ray distance parameter which is provided by the DXR API. The Fresnel term for reflected rays is defined as:

$$F_{refl} = F_{frac} + (1 - F_{frac}) \cdot (1 - (-\boldsymbol{l_{dir}} \cdot \boldsymbol{n_{0:0}}))^{F_{exp}} \quad (14)$$

where F_{frac} is the Fresnel base fraction which specifies the fraction of specular reflection when the surface is viewed from straight on, $\boldsymbol{l_{dir}}$ is the direction of the light, $\boldsymbol{n_{0:0}}$ is the normal of the water surface sampled from the CCM, and $F_{\exp}$ is the Fresnel exponent which controls the falloff of the Fresnel effect. The Fresnel term for refracted rays is the complement of F_{refl}:

$$F_{refr} = 1 - F_{refl} \quad (15)$$

The ray gen shader which emits and traces photon differentials is laid out in Algorithm 3. Refer to Appendix C for implementation details.

Algorithm 3 Photon differential emission and tracing.

Require: $Tid, ccm[N], CCMPosTex, CCMNormalTex, s, L, LDir$
Ensure: $PhotonBuffer[ccm[0].x \times ccm[0].y \times 2]$
$c \leftarrow 0$ {current cascade level}
if $N > 1$ **then** {if cascaded caustic maps}
  $Tid, c = GetCCMTid(Tid, ccm[N], s)$ {Alg. 1}
  for $i = 0$ **to** c **do**
    $L \leftarrow L \times s^2$ {Eq. 7}
  end for
end if{Map dispatch thread id to CCM pixel coord}
$uv \leftarrow \frac{Tid+0.5}{ccm[0]}$ {CCM sampling}
$x00 \leftarrow CCMPosTex.sample(uv, c).xyz$
$n00 \leftarrow CCMNormalTex.sample(uv, c)$ {Neighbor CCM sampling}
$uvRight \leftarrow uv + \frac{1}{(ccm[0].x, 0)}$
$x10 \leftarrow CCMPosTex.sample(uvRight, c).xyz$
$n10 \leftarrow CCMNormalTex.sample(uvRight, c)$
$uvUp \leftarrow uv + \frac{1}{(0, ccm[0].y)}$
$x01 \leftarrow CCMPosTex.sample(uvUp, c).xyz$
$n01 \leftarrow CCMNormalTex.sample(uvUp, c)$
if $n00.w$ **then** {if valid water point}
  $Ray.O \leftarrow x00$ {ray origin}{Shadow rays}
  $Ray.Dir \leftarrow -LDir$
  $Payload \leftarrow TraceRay(Ray)$
  if $Payload.IsHit()$ **then**
    return {early exit, point not visible from light}
  end if{Reflected rays}
  $Ray.Dir \leftarrow normalize(reflect(LDir, n00))$
  $Payload \leftarrow TraceRay(Ray)$
  if Payload.IsHit() **then**
    $HitWPos \leftarrow Ray.O + Ray.Dir \times Payload.HitT$
    $Iatt \leftarrow GetIntensityAttenuation()$ {Eq. 13}
    $Frefl \leftarrow GetReflFresnel()$ {Eq. 14}
    $R \leftarrow L \times Iatt \times Frefl$ {Eq. 12}
    $GBufferData \leftarrow GetGBufferData(HitWPos)$
    $nScene \leftarrow GBufferData.WorldNormal$
    if $-Ray.Dir \cdot nScene > 0$ **then** {front-face}
      $PhotonScale \leftarrow CalcPhotonScale()$ {Alg. 2}{Update intersection buffer}
      $InterlockedAdd(InstanceCount, 1, cnt)$
      $PhotonBuffer[cnt].RayHitPos \leftarrow HitWPos$
      $PhotonBuffer[cnt].RayType \leftarrow 0$ {refl/refr}
      $PhotonBuffer[cnt].RayDirection \leftarrow Ray.Dir$
      $PhotonBuffer[cnt].PhotonScale \leftarrow PhotonScale$
      $PhotonBuffer[cnt].SceneNormal \leftarrow nScene$
      $PhotonBuffer[cnt].RayIntensity \leftarrow R$
    end if
  end if
end if

• The next step of the workflow consists of a rasterization pipeline, where caustics are rendered into the caustics buffer placed in screen space, using the photon information from the previous ray tracing step. Each photon has a quad sprite associated with it that represents the extent of the scene (and thus, the image) to which it possibly contributes. The task is to accumulate contribution of each photon at each pixel.

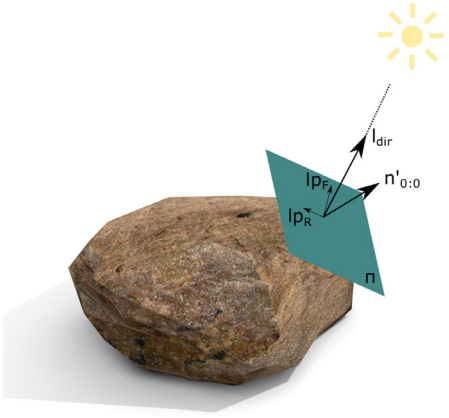


Fig. 6. Base vectors for patch space Π: the hit point's normal $\boldsymbol{n'_{0:0}}$ which defines the patch plane, $\boldsymbol{lp_F}$ is the projection of the photon direction $\boldsymbol{l_{dir}}$ on to Π and $\boldsymbol{lp_R}$ is orthogonal to $\boldsymbol{lp_F}$.

The vertex shader used in this step defines the final quad sprite size for the photon:

$$q = \sqrt{\frac{r.width}{ccm_0.x - 1} \cdot \frac{r.height}{ccm_0.y - 1}} \cdot q_0 \cdot \sigma \tag{16}$$

where the square root denotes the caustic map grid scale, q_0 is a user-defined parameter for the initial quad size, and σ is the correction factor from Equation (11) calculated via photon differentials. Since the range of the light and the caustic map resolution are variable values, we propose the adjustment of the quad sprite size by a factor which depends on the relation of these values. Initially, the sprite size is adjusted based on the proportionality of the range of the light and the dimensions of the caustic map - the radicand of the square root denotes the caustic map grid unit area. Then, the correction factor is applied.

After computing the photon quad size, the vertex shader computes the vertex positioning by centering a local square patch on the hit point, and scaling each local vertex by q:

$$lp = \{(-1, 1), (1, 1), (-1, -1), (1, -1)\} \tag{17}$$

$$x'_{0:0} = x'_{0:0} + \frac{q \cdot lp[v].x \cdot \boldsymbol{lp_F} \cdot a + q \cdot lp[v].y \cdot \boldsymbol{lp_R}}{2} \tag{18}$$

lp stores the coordinates of the local unit square patch, v is the vertex shader's current vertex id, i.e. $v \in [1, 4]$ since each patch primitive consists of four vertices, $x'_{0:0}$ is the position of the photon, and $\boldsymbol{lp_F}$ and $\boldsymbol{lp_R}$ are vectors placed on the tangent plane defined by the hit point's normal $\boldsymbol{n'_{0:0}}$, which we denote as the patch plane Π. The first of the two, $\boldsymbol{lp_F}$, has the direction of the photon $\boldsymbol{l_{dir}}$ projected onto Π, while the second, $\boldsymbol{lp_R}$, is orthogonal to it and in the same plane. This vector basis is illustrated in Fig. 6. In order to model the projected area that the direction of the light covers on the surface, we scale the patch in its direction. The patch can be stretched into a rectangular form by an foreshortening scale factor a which is defined as a function of the cosine of the angle between the hit normal and the light direction:

$$a = \frac{1}{-\boldsymbol{l_{dir}} \boldsymbol{cdot} \boldsymbol{n'_{0:0}} + \epsilon} \tag{19}$$

where $\boldsymbol{l_{dir}}$ is the direction of the ray, $\boldsymbol{n'_{0:0}}$ is the normal in the hit point and ϵ is a small offset value to avoid the case that a tangential ray hits the plane. Therefore, as the incoming direction of the photon becomes orthogonal to the normal direction of the surface, the photon quad will be more stretched in the projected light's direction.

Aside from the shape and size of the photon quad, the vertex shader also calculates an intensity factor:

$$I = \frac{R}{a \cdot q^2} \tag{20}$$

The intensity factor I is inversely proportional to the foreshortening scale factor a, thus a weaker intensity if the reflected/refracted ray does not hit the diffuse surface head-on. It is also inversely proportional to the squared sprite size q^2, thus a (much) weaker intensity if the range of the light is much larger than the cascade-adaptive caustic map view size, and if the compression ratio of the caustics triangle area to the water surface triangle area σ is large. Intuitively, the compression ratio is large when the water surface triangle diverges (spreads) light, and is small when it converges (focuses) light onto the diffuse surface. The intensity I is proportional to the ray intensity R defined in Equation (12).

The pixel shader initially runs a scene depth test for discarding occluded fragments of photons. The depth of the hit point (minus a small bias to prevent z-fighting) is tested with the depth buffer's sampled value. If the fragment does not pass the test, it is discarded. We believe that the behaviour of the quad splatting on high frequency geometry using this simple z-test method would suffice but was not tested and is left for future work. The hit point is then shaded based on material properties, light-normal angle and the view vector. I is used to scale the shaded hit point's luminance value.

3.3. Adaptive kernel radius denoiser

Even after emitting millions of photons, the caustics buffer contains undersampled regions where lack of photons leads to noise. In order to compensate for the low sample count, it is necessary to apply image filtering algorithms. Denoising of the surface caustics buffer in [6] is done in post-processing through a set of iterated cross-bilateral blurring steps [40,41] that use edge-stopping functions [42] that account for differences in view-space depth, and positions, using a fixed-size kernel radius. We extend this filter by using a kernel radius which adapts it's size based on the perceived brightness of the caustic's pixel color on the display by the human eye.

The weight function $w(p, q)$ between pixels p, q uses depth and world-space position edge-stopping functions:

$$w(p, q) = w_z(p, q) \cdot w_{pos}(p, q) \tag{21}$$

The aim is to prevent filtering across geometric boundaries by generating weights based on surface attributes of two different pixels p and q. The depth edge-stopping function w_z is defined as:

$$w_z(p, q) = \frac{1}{|z(p) - z(q)| + \epsilon} \tag{22}$$

where $z(p)$ denotes the screen-space depth value for pixel p and is sampled from the depth buffer, and ϵ is a small value to avoid division by zero. The world-space position edge-stopping function w_{pos} is defined as:

$$w_{pos}(p, q) = \frac{1}{1 + 10 \cdot \frac{\boldsymbol{d} \cdot \boldsymbol{d}}{\delta^2}} \tag{23}$$

where $\boldsymbol{d} = pos(p) - pos(q)$ is the distance vector between the world-space positions of pixels p, q calculated using the depth value, and δ is a distance scale parameter which adapts its value based on the perceived brightness of the color of the caustic in a pixel, as described below.

The filtered caustics intensity value for a current pixel p is thus:

$$C(p) = \frac{\sum_{q \in \Omega} w(p, q) \cdot C(q)}{\sum_{q \in \Omega} w(p, q)} \tag{24}$$

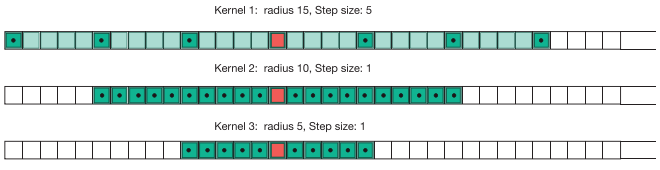


Fig. 7. The three kernel levels for the column-wise pass based on the pixel's luminance value. Light green denotes the filter span, dark green are the contributing samples within the filter, and red is the current pixel. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

where Ω is the gathered filter footprint and $C(q)$ is the sampled caustics intensity value of the contributing sample pixel q within the filter.

We extend the fixed filter footprint Ω by adjusting the radius of the kernel based on the perceived brightness of the color of the caustic in a pixel. More specifically, we define a threshold luminance value C_{scene}:

$$C_{scene} = 0.114 \cdot lum(L_{min}) \tag{25}$$

where $lum(L_{\min})$ denotes the minimum light luminance among all scene lights, and 0.114 is an experimentally defined constant. The luminance of the caustics intensity value $lum(C(p))$ for the current pixel p is tested against various fractions of C_{scene} in order to determine the perceived brightness of the pixel of the caustic relative to the luminance of the scene, and ultimately adapt the radius ρ, step size λ and distance scale parameter δ of the filter:

$$\rho = \begin{cases} 15, & \text{if } lum(C(p)) < 0.1 \cdot C_{scene} \\ 10, & \text{if } lum(C(p)) < 0.5 \cdot C_{scene} \\ 5, & \text{otherwise} \end{cases} \tag{26}$$

$$\lambda = \begin{cases} 5, & \text{if } lum(C(p)) < 0.1 \cdot C_{scene} \\ 1, & \text{if } lum(C(p)) < 0.5 \cdot C_{scene} \\ 1, & \text{otherwise} \end{cases} \tag{27}$$

$$\delta = \begin{cases} 15, & \text{if } lum(C(p)) < 0.1 \cdot C_{scene} \\ 2, & \text{if } lum(C(p)) < 0.5 \cdot C_{scene} \\ 1, & \text{otherwise} \end{cases} \tag{28}$$

Fig. 7 depicts three possible filter kernels, of our implementation. The motivation behind the adaptivity for the above parameters is performance - quality balance. If the luminance of the pixel of the caustic is very low (relative to the minimum luminance of the scene), then for the first level the shader uses a coarse-stepped, large kernel, expanding the filter scope while being efficient ($\frac{2\cdot\rho}{\lambda} = \frac{30}{5} = 6$ iterations of contributing pixels). The second level of values uses a smaller, fine-stepped kernel ($\frac{2\cdot\rho}{\lambda} = \frac{20}{1} = 20$ iterations of contributing pixels). The third layer falls back to the fixed radius case using an even smaller, fine-stepped kernel ($\frac{2\cdot\rho}{\lambda} = \frac{10}{1} = 10$ iterations of contributing pixels). The key idea is that our reconstruction should avoid changing samples in regions with little or no noise (e.g., fully shadowed regions) while filtering more in sparsely sampled, noisy regions.

One denoise pass consists of two sequential sub-passes: a column-wise and a row-wise. A parameter is exposed which determines the total number of denoise passes. Since filtering occurs in post-process, the vertex shader simply lays out the vertices in screen-space.

The column-wise sub-pass pixel shader is laid out in Algorithm 4. Refer to Appendix D for implementation details.

Algorithm 4 Column-wise denoise sub-pass of caustic pixel using adaptive kernel radius.

Require: $SC, DepthBuffer, IntensityBuffer, LMin$
Ensure: Cp
$p \leftarrow (SC.xy, 0)$
$zp \leftarrow DepthBuffer.Load(p)$
$Cp \leftarrow IntensityBuffer.Load(p)$
$VSp \leftarrow ViewSpaceFromZ(SC, zp)$
$w \leftarrow 0.0$
$sum_w \leftarrow 1.0$
$LumCScene \leftarrow 0.144 \times lum(LMin)$ {Eq. 25}
$LumCp \leftarrow lum(Cp)$
if $LumCp < .1 \times LumCScene$ **then**
  $rho \leftarrow 15 \quad \lambda \leftarrow 5 \quad \delta \leftarrow 15$
else if $LumCp < .5 \times LumCScene$ **then**
  $rho \leftarrow 10 \quad \lambda \leftarrow 1 \quad \delta \leftarrow 2$
else
  $rho \leftarrow 5 \quad \lambda \leftarrow 1 \quad \delta \leftarrow 1$
end if
for $i = -\rho$ until:$i \leq \rho$ step:$i \leftarrow i + \lambda$ **do**
  if $i \neq 0$ **then** {skip reference pixel}
    $q \leftarrow (p.x + i, p.y, 0)$ {column neighbor}
    $zq \leftarrow DepthBuffer.Load(q)$
    $Cq \leftarrow IntensityBuffer.Load(q)$
    $VSq \leftarrow ViewSpaceFromZ(SC + (i, 0), zq)$
    $wz \leftarrow \frac{1}{|zp-zq|+1.0}$ {Eq. 22}
    $d \leftarrow VSp - VSq$
    $wpos \leftarrow \frac{1}{1+10\times\frac{d\cdot d}{\delta^2}}$ {Eq. 23}
    $w \leftarrow wz \times wpos$ {Eq. 21}
    $Cp \leftarrow Cp + w \times Cq$ {contribution}
    $sum_w \leftarrow sum_w + w$ {accumulate weight}
  end if
end for
$Cp \leftarrow \frac{Cp}{sum_w}$ {normalization}
return Cp

4. Implementation details and results

Our prototype was implemented within Unreal Engine 4, essentially by modifying the engine itself, and uses DirectX Raytracing (DXR), a feature included in Microsoft's DirectX 12 application programming interface that implements hardware-accelerated ray tracing. The water surface was modelled using a high resolution plane with normal maps and simulates ocean waves using the Gerstner wave equation [43] in order to form caustics of controlled complexity.

We evaluate our three extensions to the caustics algorithm on performance and image quality impact. All reported results use an NVIDIA RTX 2080 Ti GPU running at a resolution of 1920×1080.

Our first experiment demonstrates how the use of cascaded caustic maps enables caustic coverage to a vast view distance. We create a custom scene with a large body of water refracting a directional light, thus creating caustics which span a large amount of the scene visible by the camera. Columns are additionally modelled in the scene to simulate shadows within surface caustics. The range of the directional light is $r.width = r.height \geq 3000, r.far = 6000$, the CCM precision is set to $p = 2$, and therefore $ccm_0 = (2048, 2048)$. We use $N = 2$ CCM layers for cases where multiple cascades are enabled and a CCM scale factor of $s = 2$. We compare our CCM-enhanced prototype with the implementation of [6] with volumetric scattering disabled since it is irrelevant to our surface caustics case but considered as future work (refer to Section 5).

Fig. 8 shows the qualitative results of the comparison. Fig. 8 a depicts the result of our implementation with CCMs with $N = 2$ levels and light range: $r.width = r.height = 3000$. We observe that by using CCMs, caustics at a distance are visible, i.e. caustics on the wall. Caustics near the viewer are of high resolution, while resolution decreases as they are rendered further away, thus main-

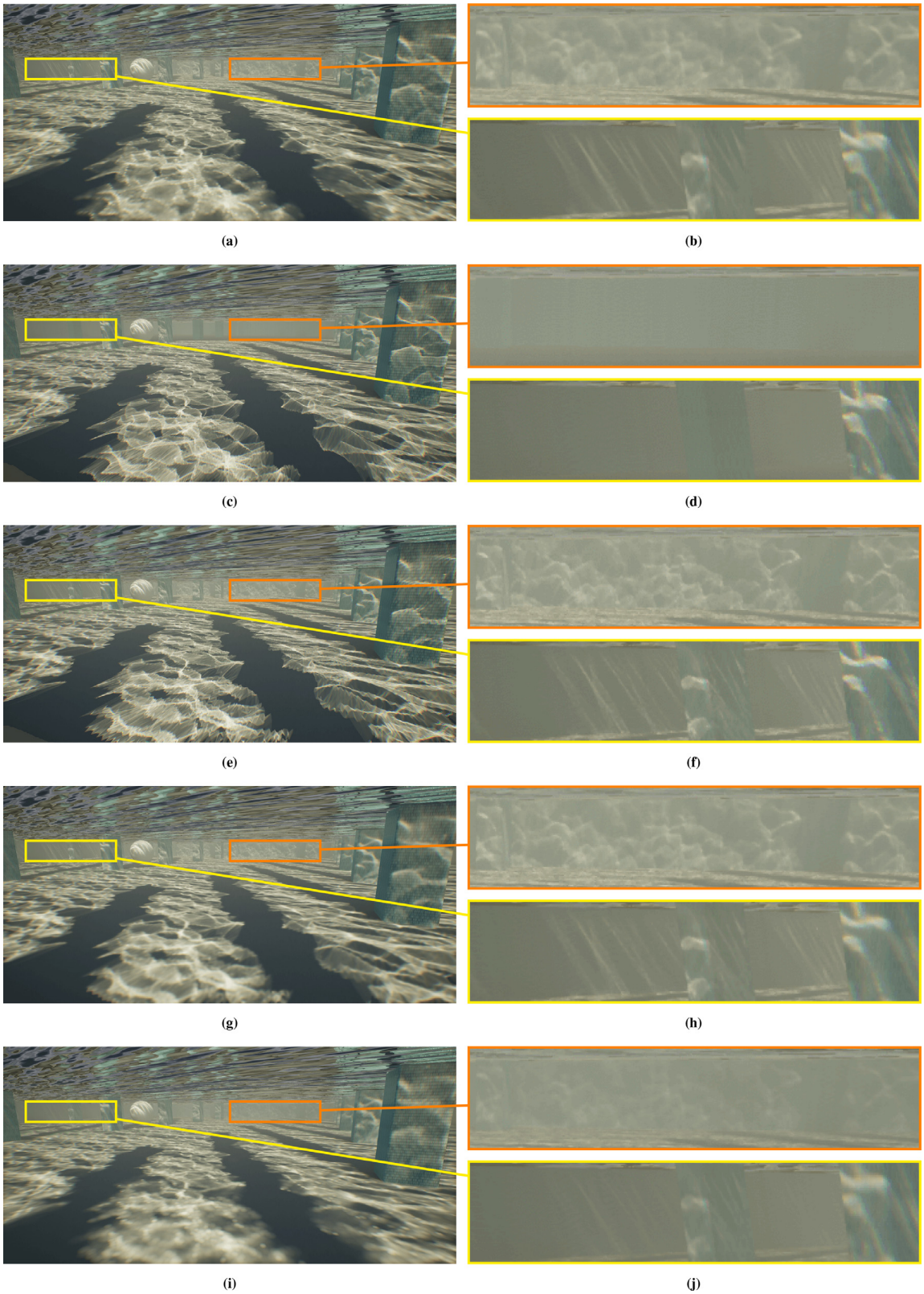


Fig. 8. Rendering of a scene with large-distance caustics coverage. Fig. 8 a uses our implementation of CCMs with $N = 2$ levels and light range: $r.w = r.h = 3000$. Fig. 8 c displays an implementation of [6], and uses a light range of: $r.w = r.h = 3000$. Fig. 8 e displays Gruen's implementation with an increased light range $r.w = r.h = 6000$. Fig. 8 g maintains the same parameters as 8 e, but is additionally denoised to mitigate distortion. Fig. 8 i uses our implementation, albeit without CCMs, but we increase the light range to $r.w = r.h = 6000$.

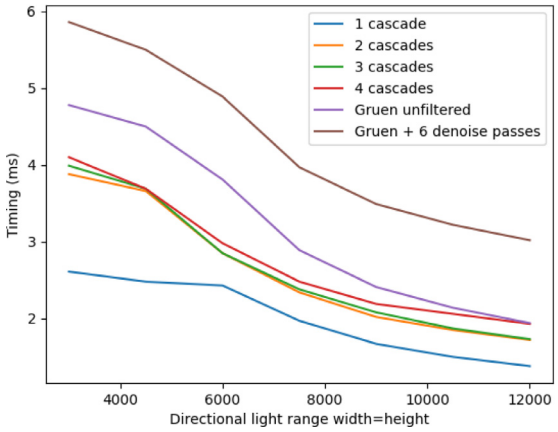


Fig. 9. Caustics workload timings for increasing light range for the frame in Fig. 8. Cases tested are our implementation for each valid number of cascades, the implementation of [6] unfiltered and denoised.

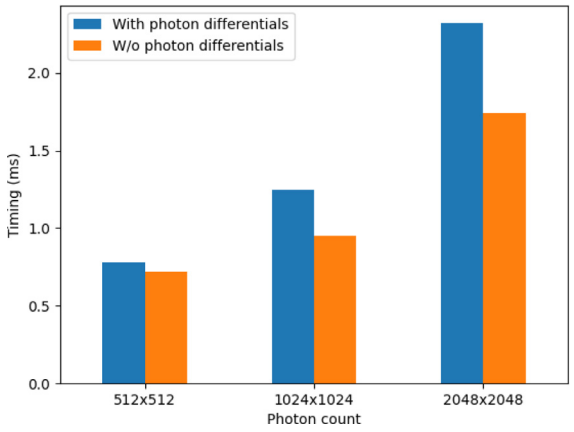
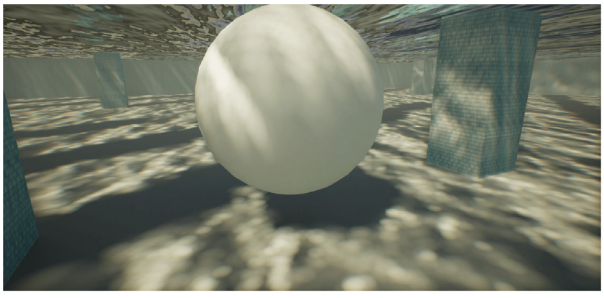


Fig. 11. Caustics workload timings for various caustic maps resolution for the frame in Fig. 10. We compare two cases for each resolution: with and without photon differentials.

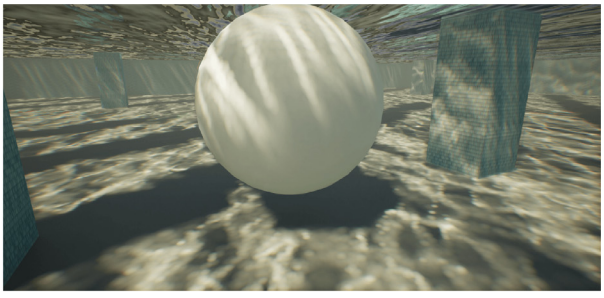
taining high performance at about 3.89ms per frame. Fig. 8 c displays an implementation of [6], and uses the same light range of: $r.width = r.height = 3000$. Fig. 8 d shows that faraway caustics cannot be viewed with a small light range without cascades. We can see that the caustics on the wall are missing. Comparing to [6] which does not utilize CCMs, the caustics on the wall could not be rendered without increasing the range of the light. This causes a low resolution caustic map and ultimately a low-quality compression ratio, overall producing distorted caustics even close to the camera (Fig. 8 e). This issue could be resolved by applying image filtering as in Fig. 8 g, but a number of denoise passes are necessary, ultimately impacting performance. Fig. 8 i uses our implementation, albeit without CCMs, but we increase the light range to $r.width = r.height = 6000$ in order to make caustics at a distance visible. This however decreases the caustic map resolution, and thus the overall final quality of the caustics is blurry, an analogous side-effect to the distortion in Fig. 8 e.

Fig. 9 compares caustics workload timings for increasing light range values for each valid number of cascades and for the unfiltered and denoised implementation of [6]. The difference in using CCMs compared to not using any, i.e. 1 cascade, is minimal, especially if the light range width and height is ≥ 6000. Also the difference in performance between using $N = 2$, $N = 3$ and $N = 4$ cascades is extremely small. Additionally, our implementation is faster than [6] even when using $N = 4$ cascades. We safely conclude that for scenes with distant surface caustics, CCMs should be used to increase realism, caustics quality and performance.

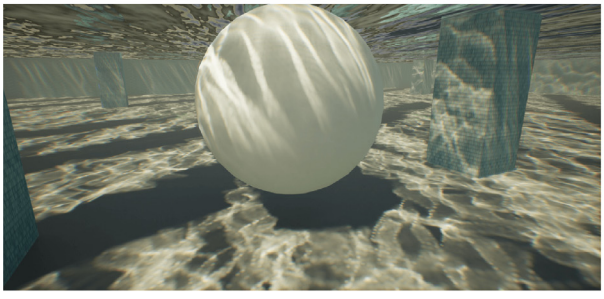
Moreover, we demonstrate the quality improvement and efficiency of tracing photon differentials with varying photon footprint size and intensity, in comparison to tracing regular rays with photon footprints of equal size and intensity. Fig. 10 displays a quali-



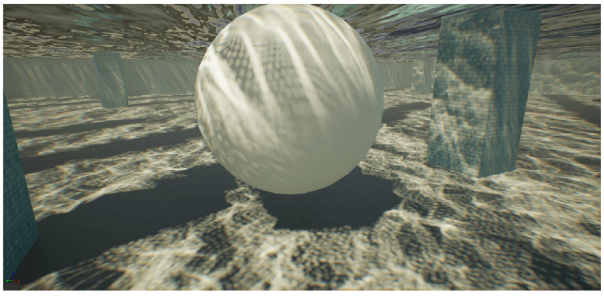
(a) Caustics traced with photon differentials and caustic map resolution of 512 × 512.



(b) Caustics traced with photon differentials and caustic map resolution of 1024 × 1024.

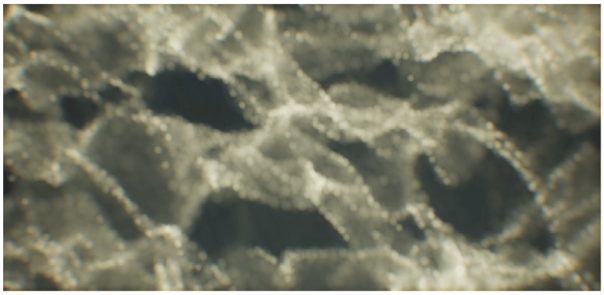


(c) Caustics traced with photon differentials and caustic map resolution of 2048 × 2048.

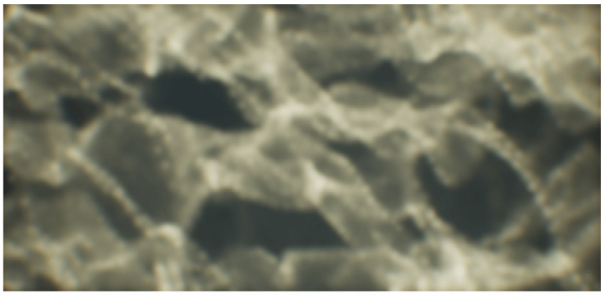


(d) Caustics traced without photon differentials and caustic map resolution 2048 × 2048.

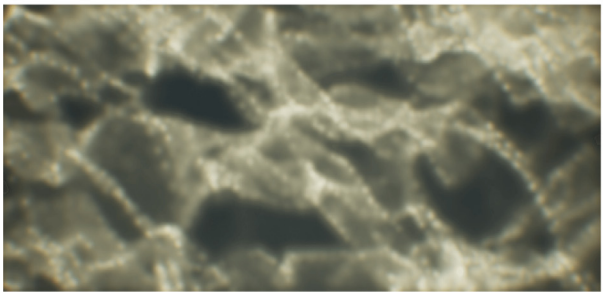
Fig. 10. Comparing caustics traced with different resolutions with and without using photon differentials, we observe that we can obtain the same or even better quality using a smaller map resolution by using photon differentials.



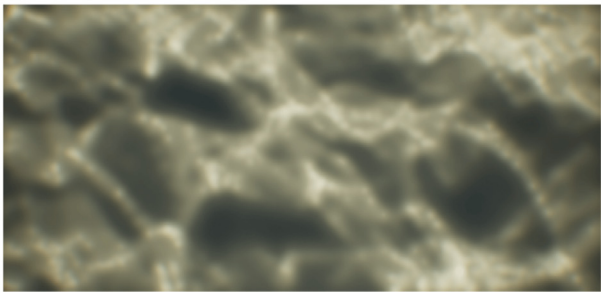
(a) Unfiltered caustics



(b) Caustics filtered with a spatial filter of fixed kernel radius [6], using 10 sequential passes.



(c) Caustics filtered with our adaptive radius spatial filter, using 1 sequential pass.



(d) Caustics filtered with our adaptive radius spatial filter, using 4 sequential pass.

Fig. 12. Qualitative comparison of caustics unfiltered, filtered with a fixed kernel radius [6], and filtered with an adaptive kernel radius. Individual photons are obvious in the unfiltered case. In order to sufficiently blur out undersampled regions using a fixed radius, at least 10 sequential passes are necessary. Figs. 12c and 12 d display caustics filtered using our adaptive radius approach. We observe that the same quality result can be achieved even by using 1 pass. Using 4 sequential passes effectively blurs out undersampled regions.

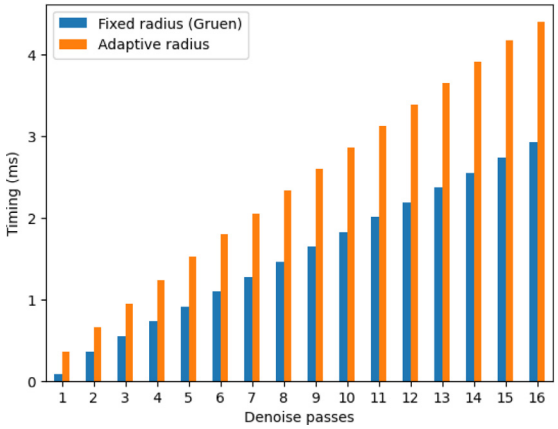


Fig. 13. Performance evaluation of each denoiser type for the frame in Fig. 12.

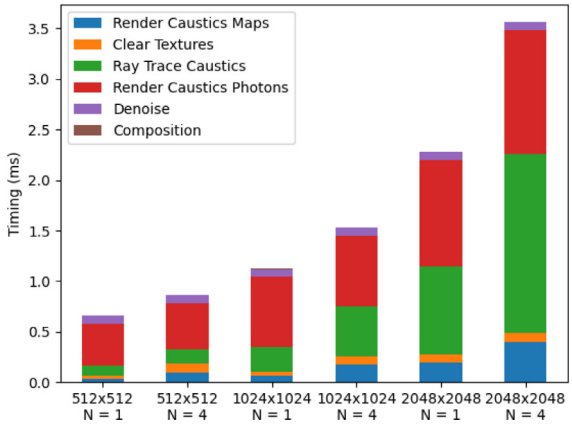


Fig. 14. Caustics workload timings for the frame in Fig. 12c for each separate subtask within the water caustics pipeline. We test three caustic map resolutions for $N = 1$ and $N = 4$ cascade levels. We trace with photon differentials and use one adaptive radius filter pass in each case.

tative comparison of caustics traced with photon differentials for various photon counts, compared to original photon tracing for the maximum photon count. The implementation without differentials forms correct caustic envelopes, but either leaves gaps or overlaps in between footprints due to photon footprints being rendered at fixed size, ultimately leading to visible individual photons. However, our implementation fills the scene surface with compact quads by estimating the size by utilizing photon differentials. In addition, the correct area calculation for density estimation using the foreshortening term ensures correct intensity distribution from all incident angles. As can be seen, the footprints have formed into continuous patterns, and the brightness distribution respects the incident angle of the light. Note that even though no denoise pass was used to produce the images of Fig. 10, enlarging the size of the splats results in larger areas for density estimation, which can be seen as pre-filtering or shifting the trade-off between variance and bias towards bias.

We also observed that by using photon differentials, we can obtain the same quality of the original photon tracing method with fewer rays, thus greatly improving the ray tracing performance. Fig. 11 compares the workload timings for each case of Fig. 10. The overhead from integrating photon differentials is minimal, compared to the quality improvement it provides.

In order to evaluate our extension to the spatial filter, we compared the quality of refracted caustics without the application of a denoise pass, applying a denoise pass with a fixed kernel radius [6,10], and applying the proposed pass with an adaptive kernel radius. Fig. 12 depicts the results of the three cases. Undersampled regions are obvious without applying the spatial filter. We observe that in order to satisfactorily blur the individual photons of undersampled regions when using a fixed radius, 10 sequential denoise

passes were necessary, while when filtering with the adaptive radius approach, only 1 denoise pass was needed to produce similar results. Using 4 sequential denoise passes effectively blurs out undersampled regions.

This naturally led us to evaluate the performance for each case. Fig. 13 indicates the average timing in milliseconds required for each denoiser type for each case depicted in Fig. 12. The adaptive radius denoiser executes on average in twice the amount of time for the same number of passes compared to the fixed radius type. However, since we can achieve the quality of 10 sequential fixed radius passes by using 1 adaptive radius pass, the performance gain is approximately five-fold.

We also present caustics workload timings taken for the frame in Fig. 12 for various caustic map resolutions and cascade levels, in order to evaluate the overall timing of our implementation in a common parameter setup. For each case, we trace with photon differentials and use one adaptive radius denoise pass. Fig. 14 indicates that the algorithm operates within a time span that is acceptable for integration in modern interactive real-time applications such as video games.

5. Conclusions, limitations and future work

In the proposed caustics rendering scheme, inspired by photon mapping, we optimize the generation of photons using cascaded caustic maps. Thus, we avoid tracing the first bounce of a ray from a light, replacing that step with rasterization. To achieve this we define cascades as a set of caustic maps with varying resolution based on the distance from the viewer. While our extended caustics algorithm is able to perform interactive ray tracing of large-scale water caustics accurately, we identify several aspects for improving and extending our work.

When tracing photon differentials, it is important to know that the caustics quality is sensitive to the total photons emitted in relation to the light's covering range - applying a low-resolution caustic map, i.e. using a low photon count, to a large scene area which requires an expanded light range, may result in very blurry caustic patterns. Fig. 10a displays this issue, where $512 \times 512 = 262144$ photons, a relatively low number, are emitted to a large scene area.

The algorithm is capable of simulating one reflection/refraction bounce. We implemented it in this way due to water surfaces typically requiring one-bounce reflection or refraction. However, multiple bounces could enhance quality in various special cases, i.e. simulating reflection of a refracted ray onto the submerged bottom of a boat, which could need two or more bounces.

Our idea of cascades was applied only to surface caustics. It would be interesting to investigate the application of cascades to volumetric caustics as well, enabling the rendering of physically-accurate, vast-scale underwater scenes which contain simulated god rays in an efficient approach.

A limitation of our adaptive radius approach for the denoiser is the loss of high frequency details even by using a few passes. The manual parameter setup for the number of iterations can be a tedious task which can vary from scene to scene.

Finally, our technique cannot efficiently deal with high frequency geometry since the created caustic maps need to have a resolution that is high enough to capture the scene in enough detail. An adaptive approach to cast rays, such as [44], could overcome this issue, potentially improving overall performance.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Gerasimos Kougianos: Conceptualization, Methodology, Software, Validation, Formal analysis, Writing - original draft, Visualization. **Konstantinos Moustakas:** Conceptualization, Writing - review & editing, Supervision, Funding acquisition.

Acknowledgments

This research has been co-financed by the European Regional Development Fund of the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call Special Actions in aquaticfarming-industrial materials-open innovation in culture (project code: Τ6ΥΒΠ-00120 -CURVE).

Appendix A. Distribution of CCM dispatch ray threads

The ray thread distribution procedure is laid out in Algorithm 1. The ray gen shader accepts the accumulated ray threads as an input and is responsible for executing Algorithm 1 in order to map `Tid`, which denotes the id of the current ray thread with range $1 \dots ccm_{acc}$, to the new dispatch thread ID for the respective CCM, which corresponds to the 2D coordinates within the caustic map of dimensions ccm_0. Additionally it stores the CCM level c of the current dispatch thread. The `ccm[N]` input array stores the dimensions of each cascade precalculated via Equations (3) and (4). If the input ray thread IDs belong to the zero-level cascade, then they are positioned on the 2D caustic map uniformly without leaving an empty space, and the algorithm exits early. If they belong to the remaining cascades, then depending on their location in the thread array, the threads are positioned on one of the colored regions displayed in Fig. 3d.

Appendix B. Kernel size calculation via photon differentials

The process for calculating the kernel size using photon differentials is laid out in Algorithm 2. It accepts the current ray traced `Ray`, the light's direction `LDir`, the current and adjacent positions and normals sampled from their respective CCMs `x00`, `x10`, `x01`, `n10` and `n01`, the intersection info `Payload`, the CCM scale factor `s` and the current cascade level `c`.

Appendix C. Photon differential emission and tracing

The ray gen shader which emits and traces photon differentials is laid out in Algorithm 3. The input `Tid` is the current dispatch thread id, `ccm[N]` stores the dimensions of each cascade, `CCMPosTex` and `CCMNormalTex` are the CCMs of the water geometry's positions and normals, `s` is the CCM level scale factor, `L` and `LDir` are the current light's intensity and direction. The output array `PhotonBuffer` stores information for each intersection. Its dimension is the inner-most area of the cascade times two, for reflected and refracted rays respectively. Since we use an indirect draw method, we atomically increment the photon instance count parameter by using `InterlockedAdd()`. Since the refracted rays case is implemented in a similar fashion, it is omitted for brevity.

Appendix D. Column-wise denoise sub-pass of caustic pixel using adaptive kernel radius

The column-wise sub-pass pixel shader is laid out in Algorithm 4. The input to the pixel shader `SC` are the 2D screen coordinates of the current pixel. The two buffers `DepthBuffer` and

`IntensityBuffer` contain information for depth and the intensity of caustics displayed on the screen. Lastly, the input `LMin` contains the intensity of the light within the scene with the minimum value.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.cag.2021.06.008

References

[1] Jensen HW. Global illumination using photon maps. In: Pueyo X, Schröder P, editors. Rendering Techniques '96. Vienna: Springer Vienna; 1996. p. 21–30. ISBN 978-3-7091-7484-5.

[2] Hachisuka T, Ogaki S, Jensen HW. Progressive photon mapping. In: ACM SIGGRAPH Asia 2008 Papers, SIGGRAPH Asia '08. New York, NY, USA: Association for Computing Machinery; 2008. ISBN 9781450318310.

[3] Hachisuka T, Jensen HW. Stochastic progressive photon mapping. In: ACM SIGGRAPH Asia 2009 Papers, SIGGRAPH Asia '09. New York, NY, USA: Association for Computing Machinery; 2009. ISBN 9781605588582.

[4] Kim H. Caustics using screen-Space photon mapping. Berkeley, CA: Apress; 2019. p. 543–55. ISBN 978-1-4842-4427-2. doi:10.1007/978-1-4842-4427-2_30.

[5] Smal N, Aizenshtein M. Real-Time global illumination with photon mapping. Berkeley, CA: Apress; 2019. p. 409–36. ISBN 978-1-4842-4427-2. doi:10.1007/978-1-4842-4427-2_24.

[6] Gruen H. Ray-Guided volumetric water caustics in single scattering media with DXR: high-Quality and real-Time rendering with DXR and other APIs; 2019. p. 183–201. ISBN 978-1-4842-4426-5. doi:10.1007/978-1-4842-4427-2_14.

[7] Gamito MN, Musgrave FK. An accurate model of wave refraction over shallow water. Computers & Graphics 2002;26(2):291–307. doi:10.1016/S0097-8493(01)00181-9.

[8] McGuire M, Luebke D. Hardware-accelerated global illumination by image space photon mapping. In: Proceedings of the Conference on High Performance Graphics 2009, HPG '09. New York, NY, USA: Association for Computing Machinery; 2009. p. 77–89. ISBN 9781605586038. doi:10.1145/1572769.1572783.

[9] Shah MA, Konttinen J, Pattanaik S. Caustics mapping: an image-space technique for real-time caustics. IEEE Trans Vis Comput Graph 2007;13(2):272–80. doi:10.1109/TVCG.2007.32.

[10] Ouyang Y, Yang X. Generating ray-traced caustic effects in unreal engine 4, part 2. Unreal Tutorial; 2020.

[11] Kajiya JT. The rendering equation. In: Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '86. New York, NY, USA: Association for Computing Machinery; 1986. p. 143–50. ISBN 0897911962. doi:10.1145/15922.15902.

[12] Veach E, Guibas LJ. Metropolis light transport. In: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '97. USA: ACM Press/Addison-Wesley Publishing Co.; 1997. p. 65–76. ISBN 0897918967. doi:10.1145/258734.258775.

[13] Šik M, Otsu H, Hachisuka T, Křivánek J. Robust light transport simulation via metropolised bidirectional estimators. ACM Trans Graph 2016;35(6) 245:1–245:12. doi:10.1145/2980179.2982411.

[14] Šik M, Krivanek J. Survey of markov chain monte carlo methods in light transport simulation. IEEE Trans Vis Comput Graph 2018;PP. doi:10.1109/TVCG.2018.2880455. 1–1

[15] Müller T, Gross M, Novák J. Practical path guiding for efficient light-transport simulation. Comput Graphics Forum 2017;36:91–100. doi:10.1111/cgf.13227.

[16] Vorba J, Karlík O, Šik M, Ritschel T, Křivánek J. On-line learning of parametric mixture models for light transport simulation. ACM Trans Graph 2014;33(4). doi:10.1145/2601097.2601203.

[17] Müller T, McWilliams B, Rousselle F, Gross M, Novák J. Neural importance sampling. CoRR 2018. arXiv:1808.03856.

[18] Lafortune E, Willems Y. Bi-directional path tracing. Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics' 1998;93.

[19] Georgiev I, Křivánek J, Davidovič T, Slusallek P. Light transport simulation with vertex connection and merging. ACM Trans Graph 2012;31(6). doi:10.1145/2366145.2366211.

[20] Grittmann P, Pérard-Gayot A, Slusallek P, Krivanek J. Efficient caustic rendering with lightweight photon mapping. Comput Graphics Forum 2018;37:133–42. doi:10.1111/cgf.13481.

[21] Šik M, Křivánek J. Implementing one-click caustics in corona renderer. In: Boubekeur T, Pradeep S, editors. Eurographics Symposium on Rendering - DL-only and Industry Track. The Eurographics Association; 2019. p. 61–7. ISBN 978-3-03868-095-6. doi:10.2312/sr.20191221.

[22] Arvo J. Backward ray tracing. In: In ACM SIGGRAPH '86 Course Notes - Developments in Ray Tracing; 1986. p. 259–63.

[23] Zhou K, Hou Q, Wang R, Guo B. Real-time KD-tree construction on graphics hardware. In: ACM SIGGRAPH Asia 2008 Papers, SIGGRAPH Asia '08. New York, NY, USA: Association for Computing Machinery; 2008. ISBN 9781450318310.

[24] Foley T, Sugerman J. Kd-tree acceleration structures for a gpu raytracer. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware. HWWS '05. New York, NY, USA: Association for Computing Machinery; 2005. p. 15–22. ISBN 1595930868. doi:10.1145/1071866.1071869.

[25] Hachisuka T, Jensen HW. Parallel progressive photon mapping on GPUs. In: ACM SIGGRAPH ASIA 2010 Sketches, SA '10. New York, NY, USA: Association for Computing Machinery; 2010. ISBN 9781450305235.

[26] Mara M, Luebke D, McGuire M. Toward practical real-time photon mapping: Efficient GPU density estimation. In: Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, I3D '13. New York, NY, USA: Association for Computing Machinery; 2013. p. 71–8. ISBN 9781450319560. doi:10.1145/2448196.2448207.

[27] Xu X, Wang B, Wang L, Xu Y, Yang C, Meng X. A task and data balanced distributed photon mapping method. Computers & Graphics 2019;82:214–21. doi:10.1016/j.cag.2019.05.009. URL http://www.sciencedirect.com/science/article/pii/S0097849319300640

[28] Liu X-D, Zheng C-W. Adaptive importance photon shooting technique. Computers & Graphics 2014;38:158–66. doi:10.1016/j.cag.2013.10.027. URL http://www.sciencedirect.com/science/article/pii/S0097849313001684

[29] Stuerzlinger W, Bastos R. Interactive rendering of globally illuminated glossy scenes; 1997. p. 93–102. ISBN 978-3-211-83001-7. doi:10.1007/978-3-7091-6858-5_9.

[30] Moreau P, Sintorn E, Kämpe V, Assarsson U, Doggett M. Photon splatting using a view-sample cluster hierarchy. In: Proceedings of High Performance Graphics, HPG '16. Goslar, DEU: Eurographics Association; 2016. p. 75–85. ISBN 9783038680086.

[31] Szirmay-Kalos L, Aszódi B, Lazányi I, Premecz M. Approximate ray-tracing on the GPU with distance impostors. Comput Graph Forum 2005;24:695–704. doi:10.1111/j.1467-8659.2005.0m894.x.

[32] Wyman C, Davis S. Interactive image-space techniques for approximating caustics, vol. 2006; 2006. p. 153–60. doi:10.1145/1111411.1111439.

[33] Wyman C, Dachsbacher C. Improving image-space caustics via variable-sized splatting; 2006.

[34] Igehy H. Tracing ray differentials. In: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '99. USA: ACM Press/Addison-Wesley Publishing Co.; 1999. p. 179–86. ISBN 0201485605. doi:10.1145/311535.311555.

[35] NVIDIA. Gameworks ray tracing overview. 2018. https://developer.nvidia.com/gameworks-ray-tracing.

[36] Dimitrov R. Cascaded shadow maps. Developer Documentation, NVIDIA Corp 2007.

[37] Saito T, Takahashi T. Comprehensible rendering of 3-d shapes. In: Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '90. New York, NY, USA: Association for Computing Machinery; 1990. p. 197–206. ISBN 0897913442. doi:10.1145/97879.97901.

[38] Schjøth L, Frisvad J, Erleben K, Sporring J. Photon differentials; 2007. p. 179–86. doi:10.1145/1321261.1321293.

[39] Watt A, Watt M. Advanced animation and rendering techniques - theory and practice; 1992.

[40] Eisemann E, Durand F. Flash photography enhancement via intrinsic relighting. ACM Trans Graph 2004;23(3):673–8. doi:10.1145/1015706.1015778.

[41] Petschnigg G, Szeliski R, Agrawala M, Cohen M, Hoppe H, Toyama K. Digital photography with flash and no-flash image pairs. ACM Trans Graph 2004;23(3):664–72. doi:10.1145/1015706.1015777.

[42] Dammertz H, Sewtz D, Hanika J, Lensch HPA. Edge-avoiding À-trous wavelet transform for fast global illumination filtering. In: Proceedings of the Conference on High Performance Graphics. HPG '10. Goslar, DEU: Eurographics Association; 2010. p. 67–75.

[43] Tessendorf J. Simulating ocean water. SIG-GRAPH'99 Course Note 2001.

[44] Wyman C, Nichols G. Adaptive caustic maps using deferred shading. Comput Graph Forum 2009;28:309–18. doi:10.1111/j.1467-8659.2009.01370.x.