www.vr-ih.com
VRIH
Virtual Reality & Intelligent Hardware
ISSN 2096-5796
CN 10-1561/TP
2020
Vol 2, Issue 2, Apr
Visual Interaction and Its Applications Special Issue
Contributing Editor: Guangzheng FEI
科学出版社
Science Press

•**Article**•

Two-phase real-time rendering method for realistic water refraction

Hongli LIU, Honglei HAN*, Guangzheng FEI

School of Animation and Digital Arts, Communication University of China, Chaoyang District, Beijing 100024*, China*

*** Corresponding author,** hanhonglei@cuc.edu.cn

Received: 18 September 2019 **Accepted:** 3 December 2019

Supported by the Fundamental Research Funds for the Central Universities, the National Key R&D Program of China (2018YFB1403900); the High-quality and Cutting-edge Disciplines Construction Project for Universities in Beijing (Internet Information, Communication University of China).

Citation: Hongli LIU, Honglei HAN, Guangzheng FEI. Two-phase real-time rendering method for realistic water refraction, 2020, 2(2): 132—141

DOI: 10.1016/j.vrih.2019.12.005

Abstract **Background** Realistic rendering has been an important goal of several interactive applications, which requires an efficient virtual simulation of many special effects that are common in the real world. However, refraction is often ignored in these applications. Rendering the refraction effect is extremely complicated and time-consuming. **Methods** In this study, a simple, efficient, and fast rendering technique of water refraction effects is proposed. This technique comprises a broad and narrow phase. In the broad phase, the water surface is considered flat. The vertices of underwater meshes are transformed based on Snell's Law. In the narrow phase, the effects of waves on the water surface are examined. Every pixel on the water surface mesh is collected by a screen-space method with an extra rendering pass. The broad phase redirects most pixels that need to be recalculated in the narrow phase to the pixels in the rendering buffer. **Results** We analyzed the performances of three different conventional methods and ours in rendering refraction effects for the same scenes. The proposed method obtains higher frame rate and physical accuracy comparing with other methods. It is used in several game scenes, and realistic water refraction effects can be generated efficiently. **Conclusions** The two-phase water refraction method produces a tradeoff between efficiency and quality. It is easy to implement in modern game engines, and thus improve the quality of rendering scenes in video games or other real-time applications.

Keywords Real-time rendering; Refraction; Liquid renderings

1 Introduction

In computer graphics, the creation of realistic images is one of the most important research fields. Users are accustomed to modern realistic rendering results in computer games and virtual reality applications. There are increasingly more details added to virtual environments. Among these, water is very common. Modeling and rendering water are involved in getting a convincing water effect[1]. However, how to model water to get plausible animation results[1–4] is out of the scope of this research. After producing an animated water surface, how to render it realistically is vital.

Reflection and refraction effects are essential in realistic water rendering. Although plenty of sophisticated algorithms to produce real-time reflection effects have been proposed, real-time rendering of the refraction effect is still a challenge. Rendering of an object refracted under a water surface requires a test of the intersection between the object and the ray refracted by the water surface, which has a high computational cost. Moreover, such a test is difficult in the deferred rendering pipelines, which are commonly used in modern real-time applications.

In this study, an efficient calculation method of the underwater refraction effect without using ray-tracing is proposed. The structure of the proposed method is shown in Figure 1. In the proposed method, two phases are applied to get the refraction effect on corrugated water surfaces. In the broad phase, the water surface is considered flat, while underwater vertices are transformed into refracted positions in graphics processing units (GPUs) in an efficient way. In the following narrow phase, pixel information on the water surface is recalculated by a screen space method with disturbed pixel normal to synthesize a realistic output on corrugated water.

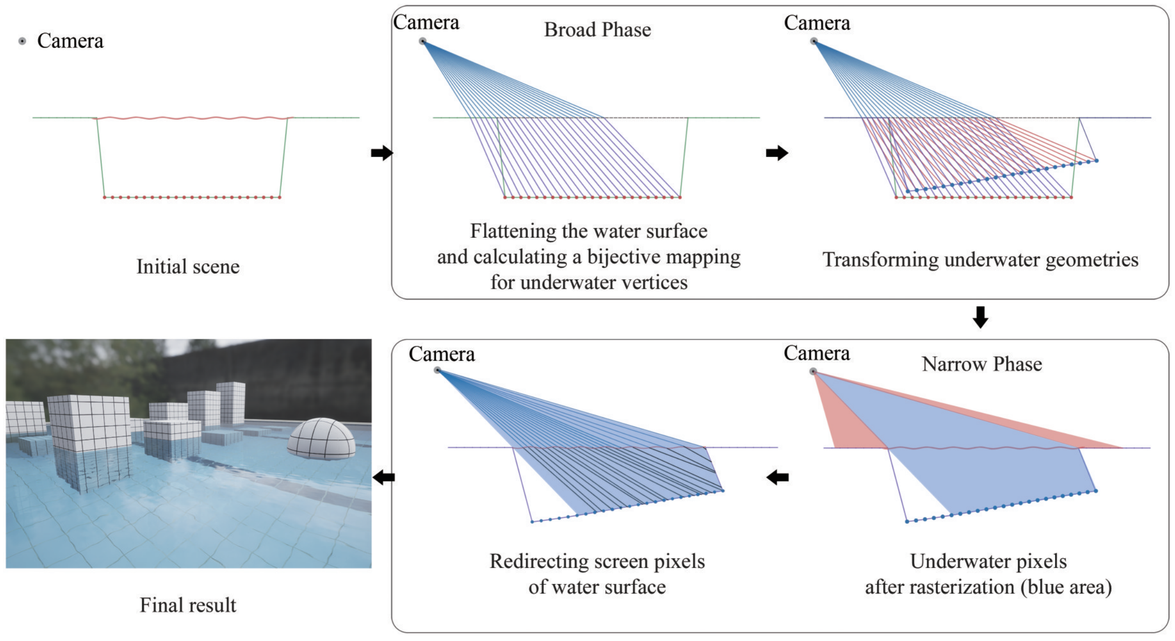
Camera
Initial scene
Camera
Broad Phase
Flattening the water surface
and calculating a bijective mapping
for underwater vertices
Camera
Transforming underwater geometries
Camera
Narrow Phase
Underwater pixels
after rasterization (blue area)
Camera
Redirecting screen pixels
of water surface
Final result

Figure 1 Overview of our two-phase refraction rendering method.

2 Related work

Real-time refraction rendering is a great challenge[5]. The refraction effect requires extra rays, which are hard to achieve in modern deferred rendering pipelines.

Some very recent papers addressed the problem of how to model the water surface in real-time with physical accuracy. Jeschke et al. presented a method to simulate water surface waves as a displacement field on a 2D domain[3]. This approach also presented a direct interface for artistic control. Schreck et al. proposed a fundamental solution sapproach to simulate time-varying water surface waves with moving obstacles[4]. Physically accurate results can be generated when compared to real-world examples. However, how to efficiently render the refraction effects of modeled water surfaces is neglected in these two references.

Ray-tracing or inverse ray-tracing is widely accepted as a simulation approach of opticaleffects[6,7]. However, real-time realistic rendering is still challenging due to the high requirements on the frame rate. Thus, it is almost impossible to do object-based ray-tracing calculations[8].

To obtain the animated water surface, Li et al. utilized 2D mesh modeling water surfaces and optimized the uniform grid of the surfaces using level of detail[9]. Then multi-octave of Perlin noise was generated to construct a random height field of the water surface. They also used a ray-tracing method to analyze and trace lights recursively of the reflection, refraction, and shadow depending on the material of the intersection point. However, the rendering process takes around 20 seconds, which is far from real-time.

Xiao et al. proposed a particle-based fluid rendering method, which provided a trade-off between speed and quality to users[10]. However, it was only applicable to particle-based water, which is uncommon in modern real-time applications.

To render rough refraction over arbitrary mesh surfaces, de Rousiers et al. introduced a method using spherical Gaussians approximated pre-convolution in environment maps[11]. In the environment mapping process, the origins of rays were ignored. As a result, this method was not suitable for rendering large scale water surfaces where the difference of positions cannot be neglected.

Voxel-based methods have been applied to generating refraction effects. Nilsson proposed combining parallel octree construction and voxel-based ray-tracing for the purpose of real-time refraction effects[12]. Rodgman and Chen used a distance field voxel for such effects[13]. These methods use a lot of GPU memory and resources in large-scale scenes. Moreover, it is not easy to implement these methods in modern deferred rendering pipelines.

To improve the efficiency of rendering, McGuire and Mara proposed a screen space technique for glossy reflection and refraction at a low cost[14]. However, this technique produces distinct artifacts in which the underwater area is occluded, which is a common problem in screen spacemethods[12,15].

Iwasaki et al. presented a powerful technique to render refraction by cutting objects into pieces[13]. The ray casting problem is turned into a sampling problem. This method is efficient, but it may cause over-sampling and failures in dynamic scenes.

Different from ray-tracing based methods[6,8,9,12] that involve a time-consuming search of intersections based on refracted rays, the proposed method only calculates the transformation of underwater vertices before the rasterization process in GPUs in the broad phase. Compared with image-based methods[8,11], the proposed method recalculates pixels on the water surface in the narrow phase to fulfill the refraction effect of corrugated surfaces. Moreover, the proposed method needs no preprocess and supports any animated planar water surfaces.

3 Broad phase

Screen space methods for generating refraction effects are very efficient but often introduce apparent artifacts. To reduce artifacts, a broad phase is added to resolve the general refracted information for underwater scenes. In this sub-process, the refraction effect is calculated through an ideally flat water surface to depict the broad appearance of underwater scenes.

Typically, rendering of the refraction effect involves casting rays due to the complex distribution of light incident on the surface. However, it is efficient and straight forward to calculate refraction effects when the water surface is ideally flat. It can be seen from the broad phase in Figure 1 that only one translation calculation is required to calculate a new position that satisfies the refraction effect for each underwater vertex.

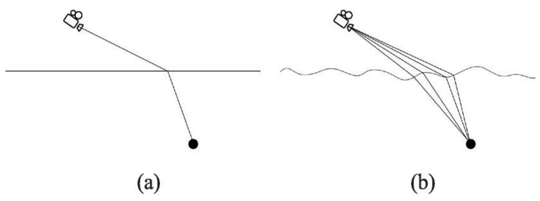

Figure 2 Comparison of bijective and non-bijective cases. (a) Bijective mapping; (b) Non-bijective mapping.

The refraction path from the camera to an underwater vertex is bijective for flat water surfaces. Thus, it is definite that every underwater vertex has a unique refraction path to the camera (Figure 2a). Otherwise, there may be multiple refraction paths for a particular underwater vertex (Figure 2b) on the corrugated water surface. In this case, a unique position that satisfies the refraction effect for the vertex cannot be determined.

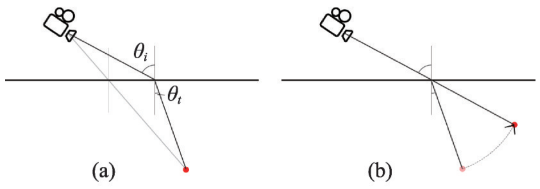

Figure 3 The processes of transforming the underwater vertex into the refracted position. (a) Calculate the ray path; (b) Transform the vertex.

After flattening the water surface into a plane, the refraction path can be determined by Snell's law (Figure 3a). Any position along the section of an identified path above the refracting plane can be considered as the refracted position of the underwater vertex. However, the position with a preserved length is selected as the refracted position of the underwater vertex (Figure 3b). It maintains the depth order of underwater geometries and ensures the accuracy of the calculated results of the further depth-dependent special effects, like subsurface scattering and foam effects, and the narrow phase. The transformed vertex is regarded as the refracted vertex and sent into the next stage of the pipeline to synthesize the broad refraction information for the later narrow phase. Algorithm 1 shows the implementation details of this process (Table 1).

Table 1 Broad phase algorithm

1:	procedure Calculate new vertex positions
2:	$a \leftarrow abs(CameraPosition.z - WaterPlane.z)$
3:	$b \leftarrow abs(WaterPlane.z - VertexPosition.z)$
4:	$d \leftarrow$ length of $CameraPosition.xy - VertexPosition.xy$
5:	$i \leftarrow$ Index of reflection
6:	If the camera is underwater then $i \leftarrow 1/i$
7:	solve $x/sqrt(a^2 + x^2) = i * (d - x)/sqrt((d - x)^2 + b^2)$ for x
8:	$delta \leftarrow x * normalize(vec3(CameraForwardVector.xy, 0))$
9:	$P \leftarrow vec3(CameraPosition.xy + delta, WaterPlane.z)$
10:	$pathlen \leftarrow sqrt(x^2 + a^2) + sqrt((d - x)^2 + b^2)$
11:	return $normalize(P - CameraPosition) * pathlen$

Traditional methods except for ray-tracing cannot cope with the problem when the camera is under the water surface. This is because the geometries above the water surface are complex, and the deflected angle is large. Since the above water scene is hard to manage for voxel-based methods, it is almost impossible to calculate the refraction effect, especially for complex or animated scenes. For screen space methods, the deflected degree of the paths is too large, such that many redirected pixels fall out of the rendering buffer. Consequently, apparent artifacts,where many pixels fail to find refraction information, occur.

The comparison can be seen in Figure 4. Since the broad phase calculation is physically accurate in our method, the refraction result is correct (Figure 4a). However, the screen space refraction[14] introduces apparent artifacts (Figure 4b), where the refracted pixels do not exist in the rendering buffer.

4 Narrow phase

In the narrow phase, we focus on more frequent cases where the surfaces involve waves. The normal vary with an amplitude change in water surfaces. Therefore, the underwater scene looks distorted. This can be achieved by refining the details of the refracted information in the broad phase by adding an extra GPU pass for water surface meshes. Every pixel on the water surface is redirected to a position according to the normal change caused by water waves.

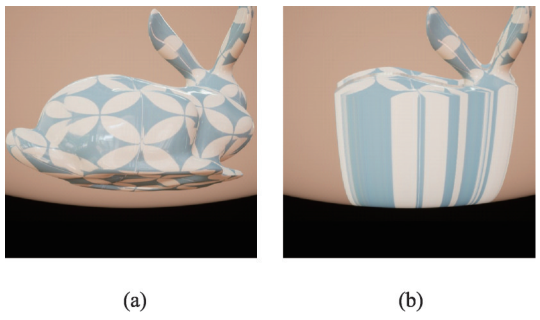

Figure 4 The comparison of different refraction methods when the camera is underwater. (a) The proposed method; (b) Screen space method[14].

This study targets real-time refraction of the water surface with shallow waves, which is the most common, in virtual scenes. The normal of the water surface varies slightly. As a result, redirected pixels tend to fall into the rendered frame buffer after rasterization.

As shown in Figure 5, the narrow phase is divided into three steps for calculating a new screen-space coordinate to redirect each pixel on the water surface. A matrix that rotates the flattened normal to a new one affected by waves is initially found. Then, this matrix is applied to the camera ray, and an approximated intersection with scene geometries can be found. Lastly, the intersection is projected onto the camera plane to gain a new screen space coordinate, and the color of this pixel in the frame buffer is assigned to the original pixel. Algorithm 2 shows the process of the narrow phase (Table 2).

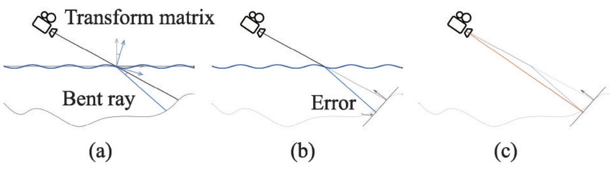

Figure 5 The three steps of the narrow phase. (a) Transform the ray; (b) Approximate an intersection; (c) Project the intersection.

Table 2 Narrow phase algorithm

```
1:  procedure Redirect rendered pixels
2:      $N \leftarrow$ get world normal from render buffers
3:      $P \leftarrow$ get world position from render buffers
4:      $D \leftarrow$ calculate deflected ray direction
5:      $P0 \leftarrow$ current position on the water plane
6:      $P1 \leftarrow$ intersection of line(P0, D) and plane(P, N)
7:      $UV \leftarrow$ project P1 to the screen
8:      return SampleScreenPixel(Clamp(UV))
```

The calculation of the transform matrix in the first step is shown in Figure 5a. In the second step, the expression of scene geometries in searching for analytic solutions to the intersection is simplified to avoid ray-tracing. We regard the implicit function describing the scene geometries as $F(x,y,z) = 0$. According to the Taylor series, we have:

$$\begin{aligned} F(x,y,z) &= F(x_0 + \delta x, y_0 + \delta y, z_0 + \delta z) \\ &= F(x_0,y_0,z_0) + \delta x \cdot F_x(x_0,y_0,z_0) + \delta y \cdot F_y(x_0,y_0,z_0) + \delta z \cdot F_z(x_0,y_0,z_0) + O(\delta x^2 + \delta y^2 + \delta z^2) \\ &\approx \delta x \cdot F_x(x_0,y_0,z_0) + \delta y \cdot F_y(x_0,y_0,z_0) + \delta z \cdot F_z(x_0,y_0,z_0) \end{aligned} \tag{1}$$

where (x_0,y_0,z_0) is a known position on the surface of the geometries and thus $F(x_0,y_0,z_0) = 0$. When the high-order infinitesimal of the total differential of the function is assumed to be negligible, the implicit function can be rewritten as follows in a neighborhood:

$$(p - p_0) \cdot \nabla F(p_0) = 0 \tag{2}$$

or as:

$$(x - x_0) \cdot F_x(x_0,y_0,z_0) + (y - y_0) \cdot F_y(x_0,y_0,z_0) + (z - z_0) \cdot F_z(x_0,y_0,z_0) = 0 \tag{3}$$

where $p_0 = (x_0,y_0,z_0)$, $p = (x,y,z)$ and p is the position to beapproximated. This approximation is illustrated in Figure 6.

This process approximates the neighborhood of a known position as a plane, which makes the intersection test faster and easier without ray-tracing. After the intersection with the approximated plane is found and projected back to the screen, new screen space coordinates are gained (Figure 5b). We resample the rendered pixels using these coordinates to produce the desired output. As shown in Figure 5b, the approximation of scene geometries mainly contributes to the error in this step. The error is negligible in most cases for water

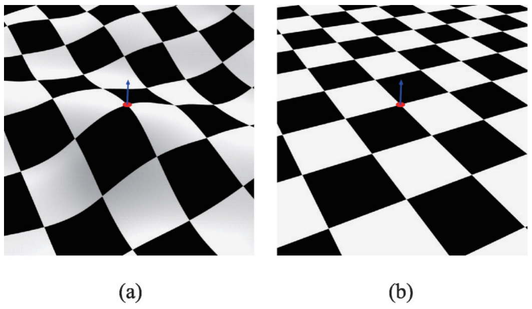

Figure 6 Approximation of the surrounding geometry at the red spot. (a) The original geometry; (b) The approximated plane.

surfaces with shallow waves.

Finally, pixels are redirected to the projection of the identified intersection to synthesize the desired effect. This may bring some artifacts where the projection is out of screen boundaries. However, they are not so critical to the rendering quality since users mainly focus on the central part of the screen[17] (Figure 7). In such cases, a clamp method is applied to assign coherence colors to those pixels.

Using the narrow phase solely can generate a realistic refraction effect in some simple situations. However, there are apparent artifacts when the redirected pixel is occluded by other objects (Figure 8a). The artifact can be avoided by the broad phase (Figure 8b). Please note in Figure 8a, a wrong pixel (green) is collected due to the complex occlusion of the scene. In Figure 8b, the issue is avoided by the broad phase. Figure 9 reveals the artifacts in a rendering scene by solely using the narrow phase.

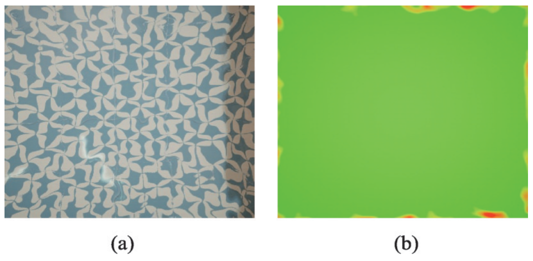

Figure 7 The artifacts in the narrow phase are hard to notice. (a) The render result; (b) The artifacts, showing in red color.

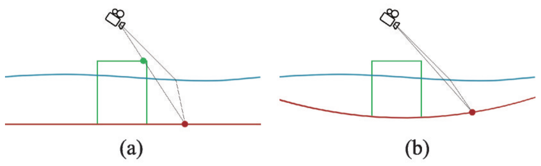

Figure 8 Failure cases may happen when using the narrow phase solely. (a) Using the narrow phase solely; (b) Combining the broad phase and the narrow phase.

5 Experiment results and comparisons

We analyzed the performances of three different conventional methods and ours in rendering refraction effects for the same scenes. The BVH accelerated ray-tracing method[6] is used as the ground truth. The proposed method, screen-space method[14], and the distance field voxel accelerated ray-tracing method[13] are compared with the ground truth. A perceptual metric[18] is applied to measure the difference between the rendering results of different methods and ground truth. As it can be seen in Figure 10 (flat water surfaces) and Figure 11 (wavy water surfaces), the proposed method achieves the highest similar results with reference images under both conditions.

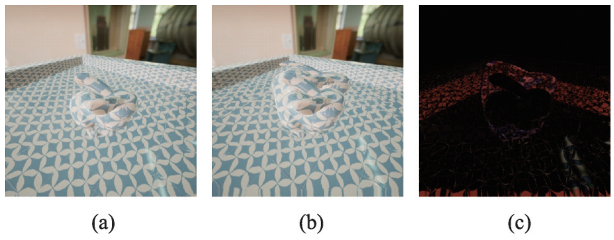

Figure 9 The rendering result comparisons. (a) Combining the broad phase and the narrow phase; (b) Using the narrow phase solely which results in the artifacts; (c) The artifacts of (b).

The degree of artifacts at the screen boundary is related to the viewing angle of the surfaces in the screen space method. This is attributed to a lack of information in the screen space. The voxel-based approach introduces aliasing artifacts at the mesh boundary, and further leads to massive memory consumption. The proposed method does not need precomputation and compensation for the information loss in the broad phase; therefore, it does not exhibit the problems of the other methods.

As shown in Table 3, the proposed method does not result in a noticeable FPS decrease, while there is a significant FPS reduction in other methods. Moreover, our method is not sensitive to the complexity of the underwater scene because it does not involve ray-tracing.

In the ray-tracing methods[6,13], the average time consumption shows a logarithmic decrease in the number of triangles in the scene. Therefore, it is hard to use them directly in real-time applications such as

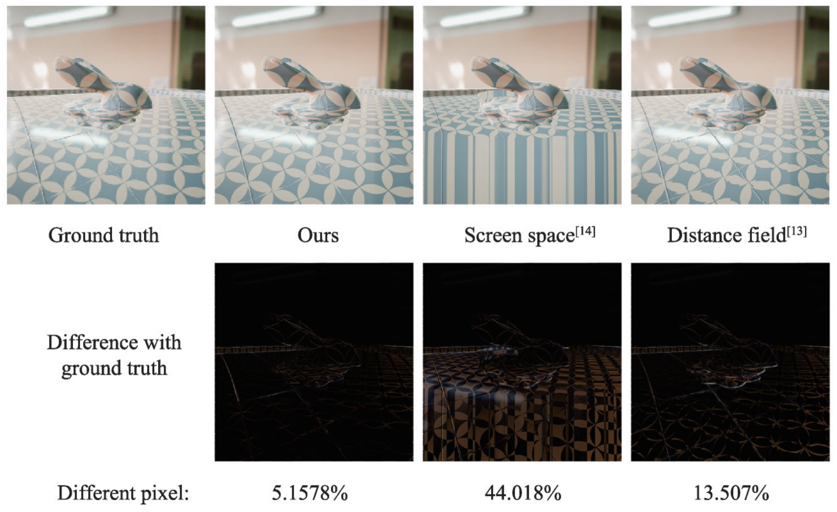

Figure 10 The comparison of different refraction methods in a scene with a flat water surface.

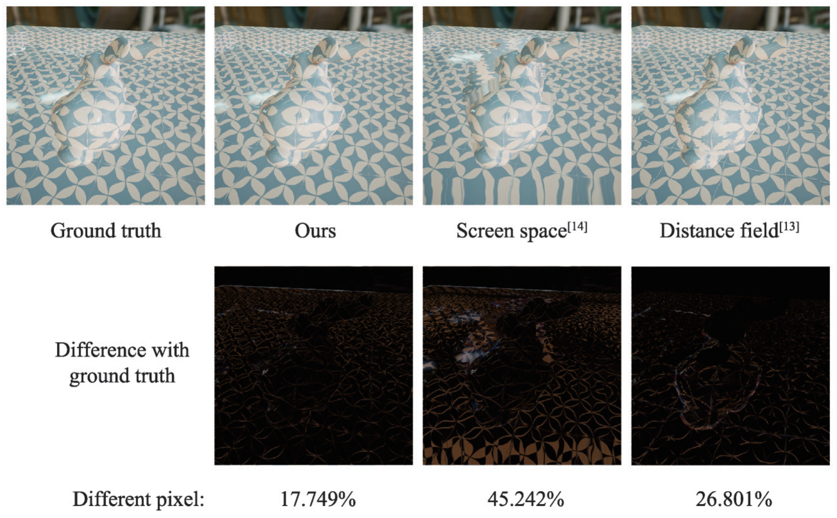

Figure 11 The comparison of different refraction methods in a scene with a corrugated water surface.

Table 3 Comparison of efficiency (using FPS) of different refraction methods

	Ground truth[6]	Ours	Screen space[14]	Distance field[13]	No refraction
16k Tris	55.90	362.53	294.77	166.47	373.92
128k Tris	18.33	366.82	258.89	36.50	369.63

video games. Although the voxel-based method[13] shows a higher time efficiency than the conventional ray-tracing method[6], it is slow and impractical when the path length and voxelized volume are increased. It not only exhibits severe aliasing as a result of a lack of voxel resolution, but is also challenging to operate. On the contrary, the FPS of the conventional screen space method[14] remained stable. Nevertheless, it produces too many noticeable artifacts.

In the broad phase, the rendering quality relies on the vertex density of refracted meshes. If the meshes do not have sufficient vertex density, the non-linear transform may be inadequate and thereby cause artifacts. In Figure 12, the bunny looks like it has sunk into the floor because of insufficient vertices. The tessellation process is recommended for such meshes.

See Figures 13 to 15 for more graphic references for the rendering of water. In Figure 13, the stable rendering results are generated using our method for any wave amplitude or viewing angle. In Figures 14 and 15, our method produces a more realistic water refraction effect in a video game than the commonly used approach. Please note that the missing refraction effect in Figures 14b, and 15b have been resolved in Figures 14a and 15a.

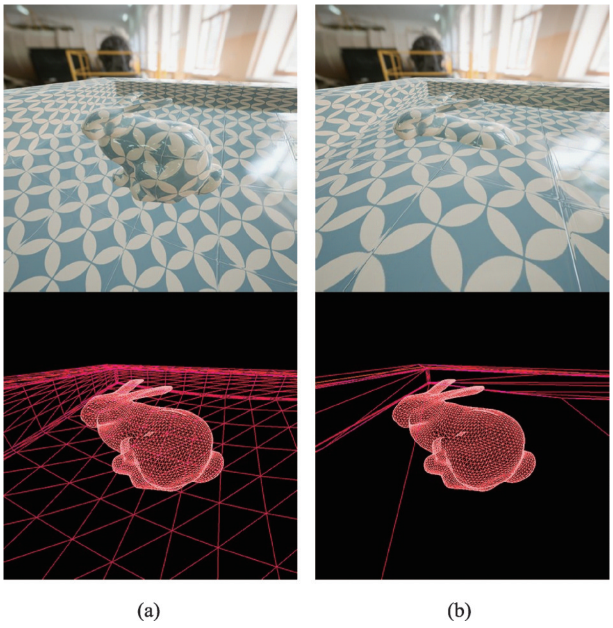
Figure 12 The comparison of results with different vertex density. (a) The pool has 1093 vertices and is rendered correctly; (b) The pool has only 98 vertices and produces artifacts.

6 Conclusion

Dividing a complex problem into two phases is an ingenious strategy in many areas like math and computer science. The idea behind this strategy is to get a basic feasible solution in the first phase, and finally get the result in the second phase based on the solution in the first phase.

The two-phase method proposed in this study produces a tradeoff between efficiency and quality to render realistic refraction effects for water scenes. Since our proposed method divides the refraction calculation problem, which normally involves a time-consuming ray-tracing process, into two separate phases. Each phase has a very efficient way to get the result with the help of modern GPU hardware. It is superior to other methods in terms of speed and accuracy. Moreover, the proposed method is easy to implement in modern game engines, like Unity or Unreal Engine.

The proposed method aims to produce refraction effects for animated waters with small wave amplitudes, which are very common in interactive applications like computer games. The results of large overturning waves or splashes like the ocean, may not be usually used in interactive applications,however have apparent artifacts. This is because the surface is flattened in the broad phase, which gives rise to errors. It can be resolved by precisely modelling the waves to reduce the probability of occlusions between waves. This will be further discussed in the future.

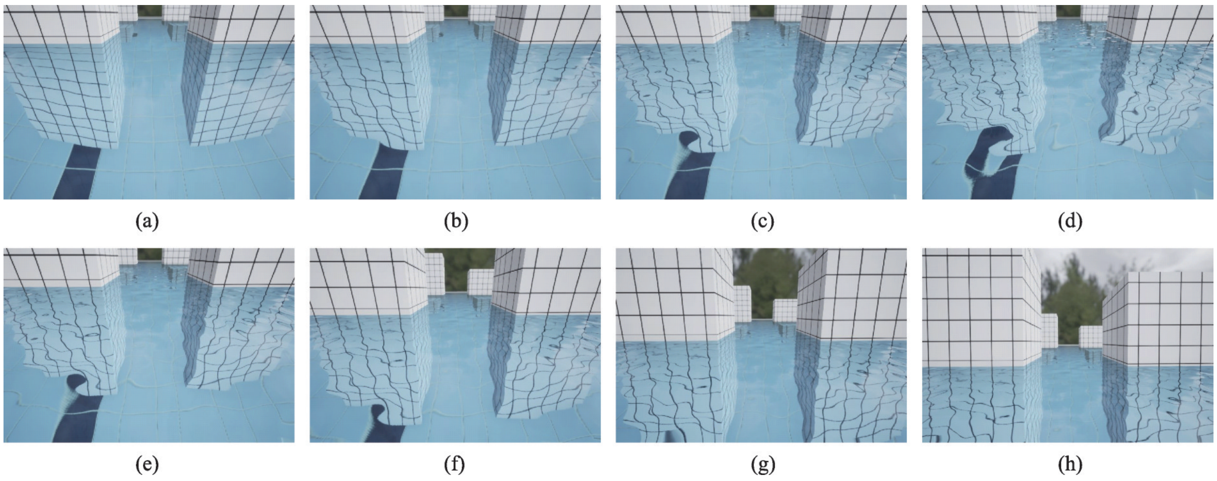
Figure 13 The rendering results of a swimming pool scene with different wave amplitudes and viewing angles using our method. The wave amplitude is increasing in (a)(b)(c)(d); (e)(f)(g)(h) are rendering results in different viewing angles of (c).

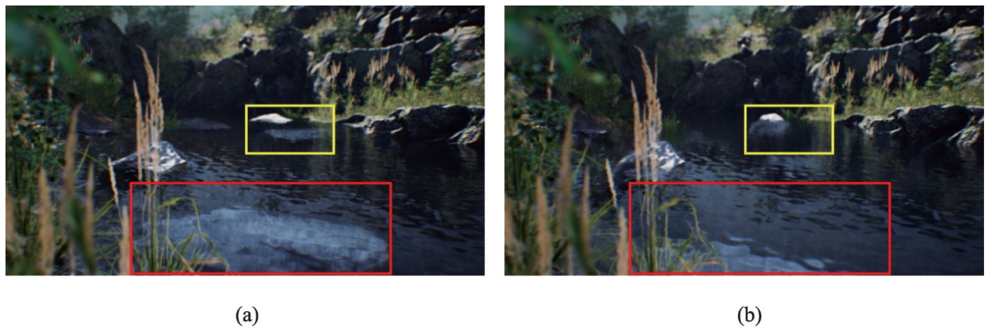

Figure 14 Comparisons of the rendering results of water in an outdoor screen of a video game. (a) The underwater geometries are refracted using our method; (b) The refracted parts are incorrect in the commonly used screen-space approach in most video games.

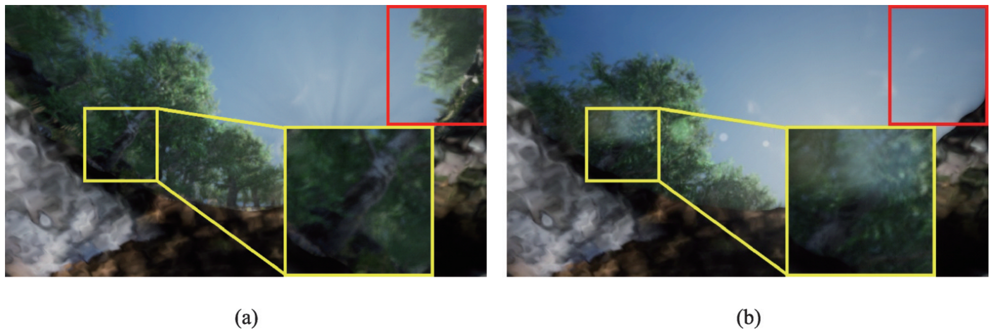

Figure 15 Comparisons of the rendering results from an underwater viewpoint. (a) The geometries above water are correctly rendered using our method; (b) The geometries are missing in the screen-space approach.

References

1 Enright D, Marschner S R, Fedkiw R. Animation and rendering of complex water surfaces. International Conference on Computer Graphics and Interactive Techniques, 2002, 21(3): 736‒744
DOI:10.1145/566570.566645

2 Klingner B M, Feldman B E, Chentanez N, O'Brien J F. Fluid animation with dynamic meshes. In: ACM SIGGRAPH 2006 Papers. Boston, Massachusetts, USA, ACM Press, 2006
DOI:10.1145/1179352.1141961

3 Jeschke S, Wojtan C. Water wave packets. ACM Transactions on Graphics, 2017, 36(4): 103
DOI:10.1145/3072959.3073678

4 Schreck C, Hafner C, Wojtan C. Fundamental solutions for water wave animation. ACM Transactions on Graphics, 2019, 38(4): 130
DOI:10.1145/3306346.3323002

5 Magnus J G, Bruckner S. Interactive dynamic volume illumination with refraction and caustics. IEEE Transactions on Visualization and Computer Graphics, 2018, 24(1): 984‒993
DOI:10.1109/tvcg.2017.2744438

6 Gunther J, Popov S, Seidel H P, Slusallek P. Realtime ray tracing on GPU with BVH-based packet traversal. In: 2007 IEEE Symposium on Interactive Ray Tracing. Ulm, Germany, IEEE, 2007
DOI:10.1109/rt.2007.4342598

7 Watt A H, Watt M. Advanced Animation and Rendering Techniques, Theory and Practice. New York: ACM Press, 1992

8 Cabeleira J. Combining rasterization and ray tracing techniques to approximate global illumination in real-time. Portugal: Universidade Técnica de Lisboa, 2010

9 Li H, Yang H M, Zhao J P. Water reflection, refraction simulation based on perlin noise and ray-tracing. International Journal of Signal Processing, Image Processing and Pattern Recognition, 2017, 10(3): 63‒74

DOI:10.14257/ijsip.2017.10.3.07

10 Xiao X Y, Zhang S, Yang X B. Real-time high-quality surface rendering for large scale particle-based fluids. In: Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. San Francisco, California, USA, ACM Press, 2017
DOI:10.1145/3023368.3023377

11 de Rousiers C, Bousseau A, Subr K, Holzschuch N, Ramamoorthi R. Real-time rendering of rough refraction. IEEE Transactions on Visualization and Computer Graphics, 2012, 18(10): 1591–1602
DOI:10.1109/tvcg.2011.282

12 Nilsson F. Real-time screen space reflections and refractions using sparse voxel octrees. Sweden: Lund University, 2015

13 Rodgman D, Chen M. Refraction in discrete ray tracing. Eurographics. Vienna, Springer Vienna, 2001: 3–17
DOI:10.1007/978-3-7091-6756-4_1

14 McGuire M, Mara M. Efficient GPU screen-space ray tracing. Journal of Computer Graphics Techniques (JCGT), 2014, 3/4: 73–85

15 Imai T, Kanamori Y, Mitani J. Real-time screen-space liquid rendering with complex refractions. Computer Animation and Virtual Worlds, 2016, 27(3/4): 425–434
DOI:10.1002/cav.1707

16 Iwasaki K, Dobashi Y, Nishita T. A fast rendering method for refractive and reflective caustics due to water surfaces. Computer Graphics Forum, 2003, 22(3): 601–609
DOI:10.1111/1467-8659.t01-2-00708

17 Renninger L W, Verghese P, Coughlan J. Where to look next? Eye movements reduce local uncertainty. Journal of Vision, 2007, 7(3): 6
DOI:10.1167/7.3.6

18 Yee H. Perceptual metric for production testing. Journal of Graphics Tools, 2004, 9(4): 33–40
DOI:10.1080/10867651.2004.10504900