CHALMERS

Real-Time Rendering of Water Caustics Using Programmable Graphics Hardware

CHRISTOFFER SANDBERG
TOMAS FALEMO

Examensarbete

Civilingenjörsprogrammet för datateknik

CHALMERS TEKNISKA HÖGSKOLA
Institutionen för data- och informationsteknik
Avdelningen för datorteknik
Göteborg 2005

Realtime Rendering of Water Caustics Using Programmable Graphics Hardware

Christoffer Sandberg and Tomas Falemo
Computer Science and Engineering, Chalmers University of Technology, Sweden

April 19, 2005

Abstract

In this paper we present an improved method for rendering refractive water caustics in real-time using modern graphics hardware. Our proposed method improves on earlier methods of volumetric rendering of caustics by allowing for arbitrary caustic receivers and non-planar light volumes. We accomplish this on commodity hardware by moving all caustic intensity calculations to programmable hardware shaders. In contrast to earlier implementations, the frame rate of our method is independent of the number of receivers.

Sammanfattning

I den här rapporten presenterar vi en förbättrad metod för att rendrera refraktiva vattencaustics i realtid med hjälp av modern grafikhårdvara. Vår föreslagna metod bygger vidare på tidigare metoder för volymetrisk rendrering av caustics genom att stödja godtycklig form på causticsmottagare samt icke-plana ljusvolymer. Vi åstadkommer detta på konsument-hårdvara genom att lägga intensitetsberäkningarna för ljusbidraget från caustics i programmerbara hårdvaru-shaders. Till skillnad från tidigare implementationer så är uppdateringsfrekvensen för vår metod oberoende av antalet mottagare.

Contents

1 Introduction **5**

2 Related Work **6**

3 Caustics Rendering **8**

3.1 Overview 8

3.2 Caustic Volumes 9

3.3 Point in Volume 9

3.4 Illumination 10

3.5 Implementation 10

4 Results **12**

5 Conclusion and Future Work **13**

5.1 Adaptability of our method on future hardware 14

5.2 Future features 15

References **16**

List of Figures

1 Illustrating a triangle in a generator mesh creating a caustic volume. 7

2 Non-planar caustic volumes 8

3 Showing caustics on objects in water 13

4 Dynamic setting rendered at 32 fps. 14

5 Comparison of different water resolutions 18

6 Distortion due to refraction mapping of pool-content texture. . . 19

7 Pots on the bottom of a pool 19

List of Tables

1 Performance measures for the scene in Figure 4. 12

1 Introduction

Creation of realistic images is one of the most researched fields within computer graphics today. Water often makes up some part of a scene and even if it only is in the shape of a glass of water it plays huge part in the credibility of the scene.

Realistic water-caustics is a vital component in rendering scenes which contains any water at all. Low-quality or non-existent caustics make even the most perfectly created scene look cheap and artificial. The existing methods for rendering caustics are either too slow for real-time applications (Photon Mapping, Ray Tracing), are not dynamic enough (Pre-generated caustic textures) or does not scale well with scene-complexity (previous work on volume-based caustics).

In this paper, we present a method for rendering caustics using caustics-volumes which is faster than Photon Mapping and Ray Tracing, is more dynamic than Pre-generated caustic textures and scales better than the volume-based method presented by Iwasaki et al. [IDN02][IDN03].

2 Related Work

Rendering Methods for Caustics has been an active topic within computer graphics for almost 20 years. As better hardware has been made available focus has shifted from creating complex patterns to drawing believable caustics at high speeds.

Plenty of methods for rendering caustics have been proposed through the years. In 1986 Arvo used backwards raytracing to accomplish caustics through illumination maps [Arv86]. Heckbert calculated the distribution of the illumination by using adaptive radiosity textures and portrayed caustics due to a lens [Hec90]. At about the same time Watt developed backward beam tracing which he used to render shafts of light and caustics [Wat90]. Further progress was made when Collins presented an improved backwards raytracing technique [Col94]. This method gave sharper and more exact caustics but was limited to planar receivers.

Jensen continued the evolution when he used photon mapping to handle arbitrary geometry [Jen96] and then added support for volume caustics and participating media [JC98].

Brière and Poulin [BP01] presented an approach to handle the blockiness that often appears when using uniform intensity over the caustic-polygons. The method tracks the wavefront of each ray of the volume and uses barycentric interpolation over each caustic triangle. This algorithm gives less blocky caustics, but it is targeted for ray tracing, and rendering times of minutes up to many hours are reported. Iwasaki et al [IDN02][IDN03] described how to render caustics by dividing the objects into slices. Unfortunately their method does not scale well with increasing scene-complexity. This is due to the fact that each receiver in the scene needs to be divided into planes which requires extra rendering passes and thus yielding lower framerates.

Wand et al. [WS03] projects caustics from sample points of caustics generating objects. Each causics receiving surface is rendered with a pixel shader that for a sample point on a caustics generating object, computes the reflected light from that sample point, using an environment map to approximate the incoming fully reflected light at the point. The contributions from all render passes corresponding to all sample points are blended together. A filter is then used to lower sampling artifacts. This method has real-time performanced for up to a few hundred sample points, and would thus probably not be suitable for large water surfaces. Furthermore, by using the environment maps, the light is assumed to be infinitely far away.

Larsen et al. [LC04] simulates photon mapping, using GPU accelerated final gathering. Caustic photons are traced on the CPU and then drawn using points

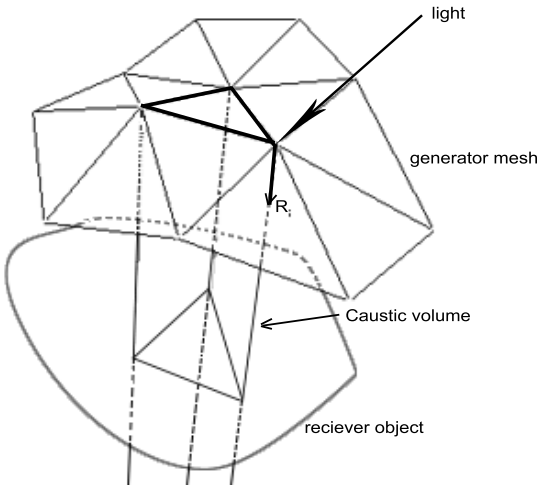


Figure 1: Illustrating a triangle in a generator mesh creating a caustic volume.

to a texture that fits the screen. This screen-space caustics map is then used by a pixel shader to compute the illumination for a point, by approximating the photon density from the closest texels corresponding to the point's pixel position and its neighboring pixels. Photon mapping is also the method for Guenther et al. [GWS04], they achieve framerates that are close to real-time using a cluster of 18 dual-CPU PCs.

Recently, Iwasaki et al [IYND04] presented a method which makes it possible to render caustics from light passing through transparent objects at interactive rates using a precomputed look-up table of pairing entry and exit positions for the lightrays.

In order to provide better scalability we modify the method described by Iwasaki et al[IDN03]. Our method use hardware vertex and fragment programs to render the caustic volumes directly into the framebuffer without the need to use multipass object slices.

3 Caustics Rendering

In this section we propose a method of drawing caustics due to refracted light using vertex and fragment shader based volume rendering.

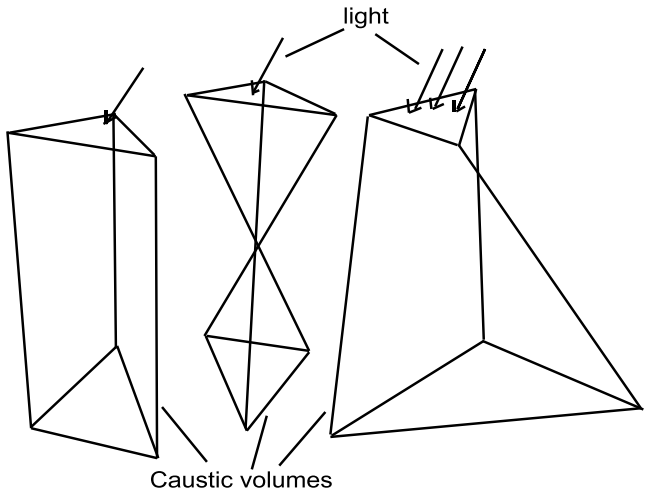


Figure 2: Example of 3 caustic volumes showing how non planar sides of the caustic volumes appear. In the example of a converging volume (middle figure) and a more general warped volume (right figure) the sides are clearly not planar.

3.1 Overview

To render caustics effects, the reflection and refraction of incident light has to be taken into account. An object causing caustics has to be either refractive, reflective or both. We call these objects caustic generators. Objects to receive caustic patterns are called receivers. In our method, arbitrary objects that can be tesselated into triangle meshes can be generators and receivers, in particular a generator can also be a receiver. The level of tesselation of receivers does not effect the outcome of the quality of the caustics as this is a per pixel operation on already rasterized geometry. In the following section we will go through, in detail, the rendering of caustics due to a refractive water surface.

3.2 Caustic Volumes

The water surface is subdivided into a mesh of triangles. For each vertex in the mesh, a refraction-vector, R_i, for the incident light is computed. Each triangle in the generator mesh along with the three refracted rays, R_i from the triangle vertices constitutes a caustic volume. The refracted rays are extended as far as needed for the scene. In particular for underwater scenes this would be either below the ocean/container bottom or as far down in the homogenous body of water that the illumination approximately is zero, in case of the later a bottom triangle should be calculated forming an enclosed caustic volume making sure to capture any receiver that would possibly be in its path. The illumination calculation is further discussed in detail in Figure 3.4. The caustic volume formed by its generator triangle and the three refracted rays are used for receiver-point-in-volume tests described in the following section. However this volume is ill suited for rendering as it does not necessarily have flat sides as shown by 2. To tesselate the surface would require extensive CPU power as for a normal application the number of caustic volumes would be quite high, as well as dynamic on a per frame basis. Drawing the caustic volumes with a screen sized quad on the other hand would waste valuable fill rate. Instead, a tight body needs to be constructed. If the caustic volume is fully enclosed and convex, the backface culling in hardware can be used to drastically reduce the amount of pixels being rasterized.

3.3 Point in Volume

For each pixel rasterized by a volume, a point in volume test is needed to see if the receiving point P at the pixel position is within the volume or not. This point, tested in world space, is defined by the z-coordinate from the z-buffer, the x,y screen coordinate of the pixel, and the screen space to world space transformation matrix. Note that a pixel may be in question for many point in volume tests, and in particular may also be found to be in several volumes due to warped caustic volumes. By forming a plane Π with the normal vector $N = (0, 1, 0)$ containing P and finding the intersection-points P_i^* between R_i and Π, the problem can be reduced into a point in triangle test instead. Since P_i^* and P are in a plane with $N_y = 1$ only the xz components of P_i^* and P needs to be used to perform the point in triangle test which can lead to more efficient code in the fragment shader.

3.4 Illumination

We use a similar method as Iwasaki [IDN03]. If P is within the volume, the intensity L_P at P as seen from the eye can be expressed as

$$L_P(\lambda) = I_0 e^{-c(\lambda)d} F_P + L_A(\lambda)$$

where λ is the wavelength sampled in RGB. I_0 is the irradiance just below the water surface, $-c(\lambda)$ the attenuation coefficient, d is the distance the light has traveled through the body of water, F_P is the Flux-ratio between the generator at the surface and the receiver and $L_A(\lambda)$ is the ambient light at P. Using hardware additive blending, each caustic volume can be processed separately and thus each generator triangle add only its own contribution to L_P. $L_A(\lambda)$ is the intensity of the receiver without caustic so it can be rendered separately from the caustics. Since the rendering of caustic volumes are made with additive blending, rendering L_A before the caustic volumes would conserve framebuffer bandwith as no readback from the framebuffer would be needed for ambient rendering. The only variables affected by the caustic volume and P are d and F_P which must be calculated at the pixel, the remaining parts of the expression are scene specific and constant. F_P is calculated by taking the ratio between the generator triangle area and the triangle formed at P.

3.5 Implementation

As the water surface is treated as a heightfield with evenly distributed xz coordinates in worldspace, many already developed water simulating methods can be applied to dynamicly update the watersurface movements such as Fast Fourier Transform and Navier Strokes Equations. The process of creating caustic volumes is done as soon as the water surface heightfield has been updated. Each caustic volume contains 6 vertices, the 3 vertices from the generator triangle, V_{Gi} and 3 vertices, V_i, calculated by adding R_i refracted ray vectors to V_{Gi} and extending R_i enough for the vertices to be placed just below the pool bottom, see Figure 1.

The rendering of a final image is split into three passes, where the caustics are drawn in the second pass. The first pass is used to generate world space coordinates for all receiver geometry. This is done using a vertex- and fragmentshader that writes the receivers x,y and z worldspace coordinates to a floatingpoint buffer. The buffer is then bound to a texture unit and used as a lookup table in the next rendering pass.

The second pass is split into two stages, firstly all receiver geometry is once again rendered, but using the geometry's ordinary shading vertex and fragment programs. In essence this is the step of rendering $L_A(\lambda)$ in 3.4. To render the

caustics, additive blending and backface culling is turned on and depth writing is switched off. The additive blending mode is needed because each volume only calculate the light intensity from its generator, and since volumes can be warped, several caustic volumes may affect light intensity of a receiver pixel. Depth writing to the Z buffer must be disabled because volumes are overlapping each other and thus requires the ability to draw the volumes in arbitrary order. Backface culling is not needed to achieve the correct caustic pattern, but it is a significant speed optimization as all pixels which can be inside the volume is covered by the front faces of the volume. The output of this pass is bound to a texture used in the third and final pass.

The purpose of the third pass is to render the water surface, and at the same time create a simple refraction effect. First all objects which are partially or fully above the water surface are rendered, then the surface mesh is rendered mapping the texture created in the second pass onto it. We calculate these texture coordinates on the fly within the vertex shader and fragment shader using a virtual plane approximating the average water level in the pool. This plane is transformed using the same model-view-projection matrix and clip space transformation as the real water surface vertices to act as texture coordinates for the water surface mesh. This creates the illusion of refraction from the eye.

4 Results

System	framerate
AMD 1.8Ghz, Radeon 9800Pro	10.5
AMD 1.2Ghz, Geforce 6800GT	16.7
Intel P4 3.2Ghz, Radeon X800Pro	21.4
AMD64 2.2GHz, Geforce 6800Ultra	32.3

Table 1: Performance measures for the scene in Figure 4.

The tested scene, as shown in Figure 4, is rendered with a water gridsize of 64x64 and pixel resolution of 512x512. All features of our method are turned on during this rendering, which means that in between each frame the directional light source is moving, the receiver objects worldspace positions are being re-evaluated, and the water surface is being evaluated using a dynamic fluid solver implementation [JJ95]. The results on a few testsystems can be seen in table 1.

Figure 7 shows our method applied for a pool with some pots added to the bottom, representing caustics rendered on simple objects. Figure 5 shows a comparison of different generator gridsizes for the same scene.

The described refraction effect becomes clearly visible in Figure 6, as the wave front evolves. Finally, caustics rendering on complex receivers, which is the strength of our method, is displayed in Figure 3.

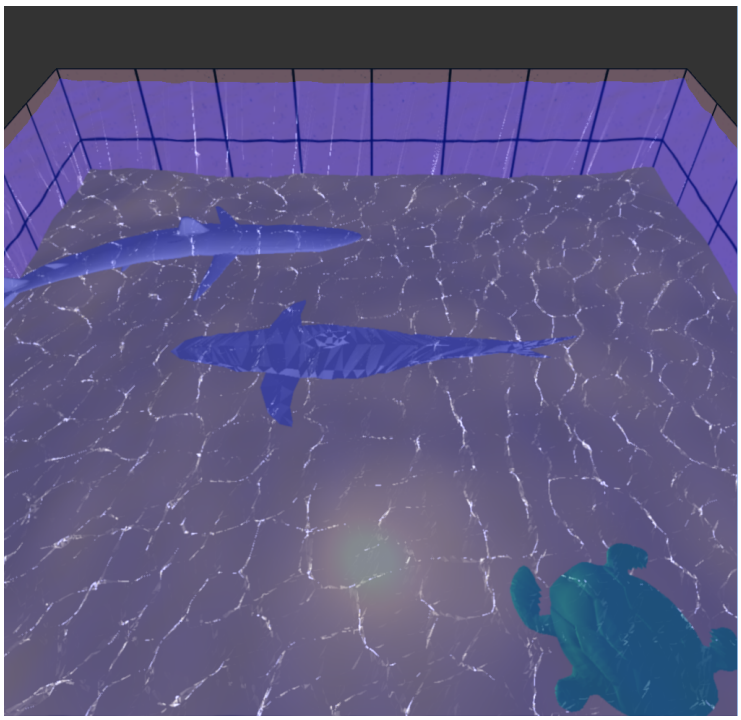


Figure 3: Volumetric caustic on complex objects under a 128x128 water surface. This image was rendered at 3fps in 1024x1024 pixel resolution. The same scene rendered at 8.6 fps using 512x512 pixel resolution.

5 Conclusion and Future Work

In this paper, we have presented a method for rendering caustics using programmable graphics hardware. The algorith is designed to be run by standard programmable graphics hardware on standard PCs with a bare minimum of functions performed on the host system's CPU. We have shown that the algorithm performs well in realtime with as much as 32 fps running on our testscene using high end hardware available on home consumer market.

The main characteristic of our method is that we allow arbitrary receivers without any cost to the caustic rendering except for a rendering of underwater scene objects to a special worldspace buffer bound as a texture. Also the design is to run the algorithm on a GPU with little interaction with the host CPU, leaving the CPU to perform other tasks, such as simulation of the environment in which caustics are to be displayed in.

The method we have described allows for creation of refractive caustic due to

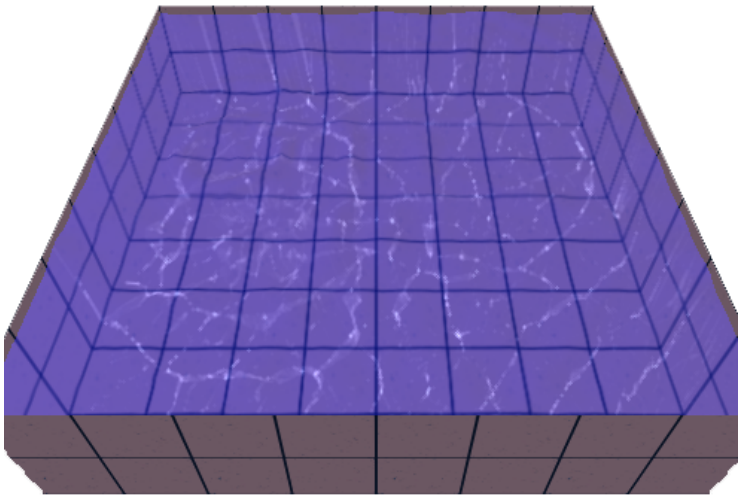


Figure 4: Displaying an empty pool which is rendered at 32 fps with full dynamic waterwave simulation, moveable lightsource, re-evaluating the geometry in the scene between every frame, as well as calculating the reduced intensity of the lightbeams due to the participating homogenous watermedia.

light interaction with water surfaces. However, the concept of caustic volumes is far more generic and can be used to render reflective caustic as well. To achieve this, nothing in the concept of the actual caustic volumes need to be changed. For reflective caustic the reflection of the light needs to be computed instead of the refraction as presented in this paper. The caustic volume is then created from these new rays. For a generic reflective object, there is a need to compute which triangles are visible from the light source and add these to the generator mesh. Other refractive objects, such as glass, can use caustic volumes for rendering as well. The generator mesh for these can be constructed from a light distribution formula describing how the light is reflected and refracted within the body. Iwasaki et al. [IYND04] have presented a method for calculating such light distribution functions within transparent bodies. However, such methods might prove hard to perform in realtime.

5.1 Adaptability of our method on future hardware

Today's rendering pipelines are capable of creating new triangles, and in fact must do so when point sprites and line segments are to be rendered as the actual

hardware rasterizer normally only implements triangle rasterization. However within rendering pipelines of today this stage is fixed and all triangle generating functionality must lay on the host processor, as in the case of caustics volumes, Microsoft intends to change this stage to become programmable in their release of Windows Longhorn, which will include Windows Graphics Foundation (WGF) [Bey03], the replacement of DirectX. When this stage becomes programmable in hardware, which will be a requirement for hardware manufacturers to call their products WGF compliant, the entire process of rendering caustics can be moved from the host processor to a GPU. The process of creating and transmitting caustic volumes to the GPU is an expensive operation, with this new programmable stage in the pipeline called Topology processing only the surface triangles have to be sent down the pipeline along with batch-constant values such as light direction. The Topology Processor then can calculate the caustics volume associated with each surface triangle and feed this directly further down the pipeline to the Vertex Processor.

5.2 Future features

In future work, we would like to add true refraction as seen from the eye through the water surface. Also, as discussed in section 5.1, the creation of caustic volumes should be moved to the GPU when it allows for non-abusive implementations, preferably through a generic topology processor.

References

[Arv86] Arvo J.: Backwards ray tracing. In Developments in Ray Tracing, SIGGHRAPH Course, 1986.

[Bey03] Beyond3d: Directx next early preview. http://www.beyond3d.com/articles/directxnext/, 2003.

[BP01] Brière N., Poulin P.: Adaptive representation of specular light flux. Computer Graphics Forum, Vol.20, No.2, 2001, pages 149-159, 2001.

[Col94] Collins S.: Adaptive splattering for specular to diffuse light transport. Fifth Eurographics Workshop on Rendering, pages 119-135, June, 1994.

[GWS04] Guenther J., Wald I., Slusallek P.: Realtime caustics using distributed photon mapping. Eurographic Symposium on Rendering, pages 111-121, 2004.

[Hec90] Heckbert P.: Adaptive radiosity textures for bidirectional ray tracing. Proc. SIGGRAPH 90, pages 145-154, 1990.

[IDN02] Iwasaki K., Dobashi Y., Nishita T.: An efficient method for rendering underfwater optical effects using graphics hardware. Computer Graphics Forum, 21(4):701-712, 2002.

[IDN03] Iwasaki K., Dobashi Y., Nishita T.: A fast rendering method for refractive and reflective caustics due to water surfaces. EUROGRAPHICS, pages 283-291, 2003.

[IYND04] Iwasaki K., Yoshimoto F., Nishita T., Dobashi Y.: A rapid rendering method for caustics arising from refraction by transparent objects. Cyberworlds, 2004 International Conference, 2004.

[JC98] Jensen H. W., Christensen P. H.: Efficient simulation of light transport in scenes with participating media using photon maps. In Computer Graphics, SIGGRAPH, pages 311-320, 1998.

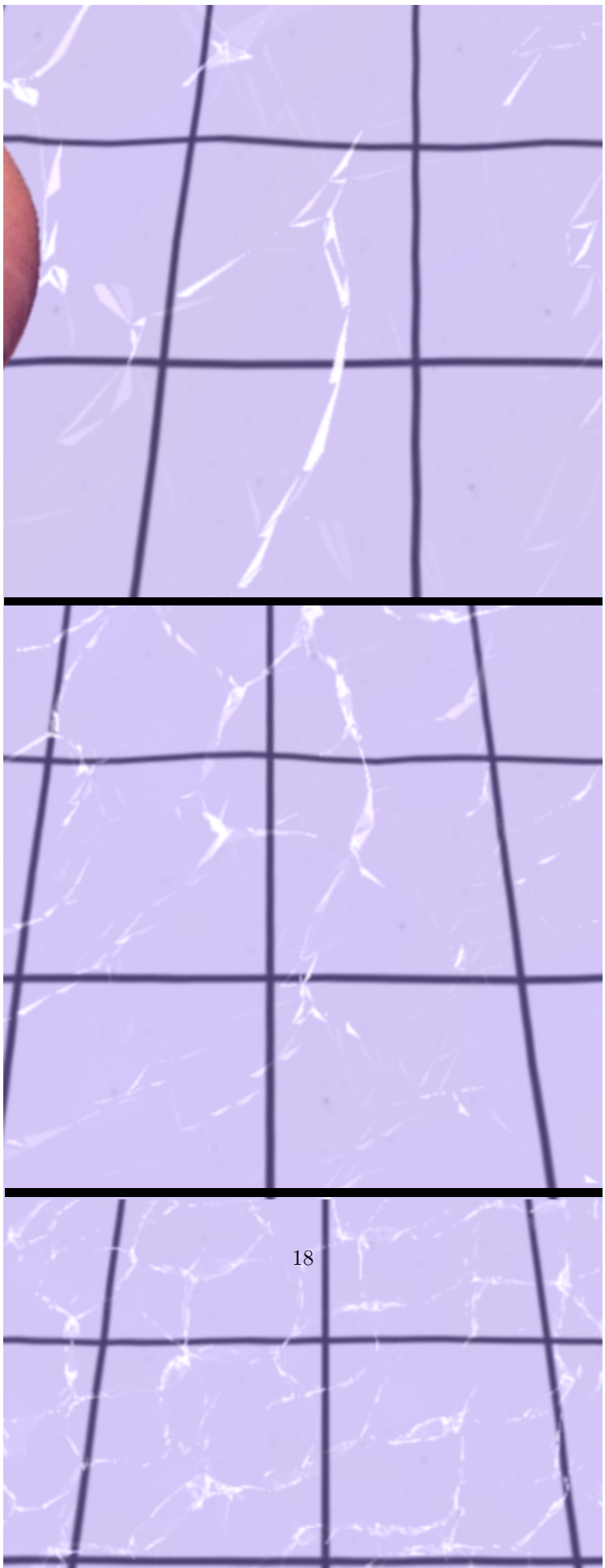
[Jen96] Jensen H. W.: Global illumination using photon maps. In Proceedings of the 7th EUROGRAPHICS Workshop on Rendering, pages 21-30, 1996.

[JJ95] James O., Jessica H.: Dynamic simulation of splashing fluids. proceedings of Computer Animation, pages 198-205, 1995.

[LC04] Larsen B. D., Christensen N.: Simulating photon mapping for real-time applications. Eurographics Symposium on Rendering, 2004.

[Wat90] Watt M.: Light-water interaction using backward beam tracing. SIGGRAPH 90, pages 377-385, 1990.

[WS03] Wand M., Strasser W.: Real-time caustics. Computer Graphics Forum, 22(3):611-611, 2003.



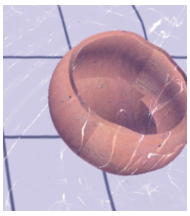


Figure 6: Distortion due to refraction mapping of pool-content texture.

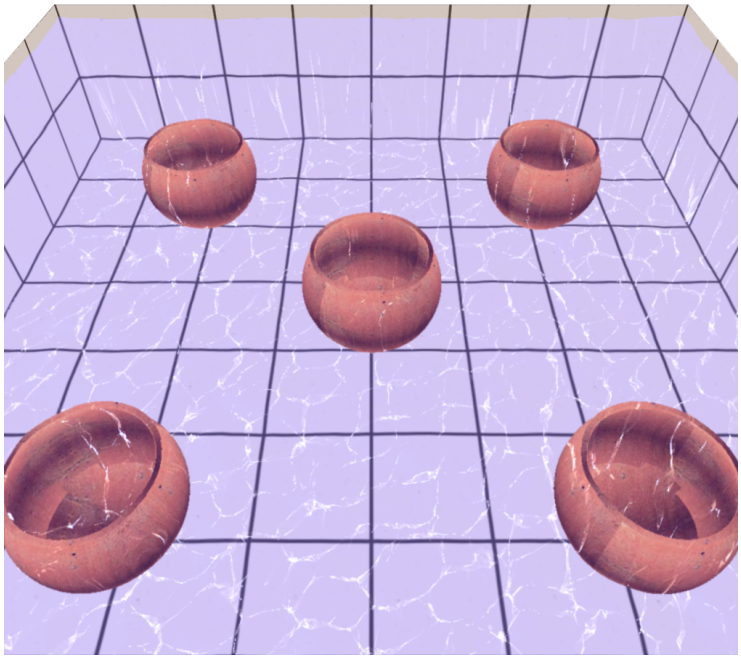


Figure 7: Displaying some pots on the bottom of a pool, the water surface mesh size is 128x128 and the image was rendered at 512x512 pixels. Note that the caustics correctly, on a per pixel resolution, are placed on the geometry of the pots.