

# Spring Conference on Computer Graphics SCCG 2008

in cooperation with  
Eurographics and ACM SIGGRAPH

## Conference Materials and Posters

Budmerice  
April 21 - 23, 2008



i n v e n t

---

Publisher:  
Editor:  
Cover page & logo design:  
Production:

Comenius University, Bratislava  
Martin Samuelčík  
Matej Novotný, Jozef Martinka  
Martin Samuelčík

Webpage:

<http://www.sccg.sk>

ISSN:

1335 - 5694

# International Programme Committee

IPC chairman  
**Prof. Karol Myszkowski**  
Max-Planck Institute for Informatics,  
Germany

Jeroen van Baar  
Sebastiano Battiato  
Alexander Belyaev  
Bedřich Beneš  
Peter Birkholz  
David Breen  
Jiří Bittner  
Alan Chalmers  
Pavel Chalmovianský  
Ján Cíger  
Silvester Czanner  
Miguel Chover  
Roman Ďurikovič  
Miquel Feixas  
Petr Felkel  
Andrej Ferko  
Franca Giannini  
Markus Grabner  
Helwig Hauser  
Vlastimil Havran  
Bert Jüttler  
Maxim Kazakov  
Ivana Kolingerova

Rafal Mantiuk  
Simone Marini  
Karol Myszkowski  
László Neumann  
Alexander Pasko  
Giuseppe Patanè  
Bernard Péroche  
Tomas Plachetka  
Werner Purgathofer  
Przemyslaw Rokita  
Mateu Sbert  
Benjamin Schmitt  
Peter Shirley  
Pavel Slavík  
Martin Šperka  
Daniel Thalmann  
Petr Vaneček  
Ivan Viola  
Michael Wimmer  
Pavel Zemčík  
Borut Žalik  
Jiří Žára

# List of Reviewers

Jeroen van Baar  
Sebastiano Battiato  
Alexander Belyaev  
Bedřich Beneš  
Peter Birkholz  
David Breen  
Jiří Bittner  
Pavel Chalmovianský  
Ján Cíger  
Silvester Czanner  
Miguel Chover  
Piotr Didyk  
Roman Ďurikovič  
Miquel Feixas  
Petr Felkel  
Andrej Ferko  
Franca Giannini  
Markus Grabner  
Thorsten Grosch  
Helwig Hauser  
Vlastimil Havran  
Matthias Hullin  
Matthias Ihrke  
Bert Jüttler  
Maxim Kazakov

Ivana Kolingerova  
Rafal Mantiuk  
Simone Marini  
Karol Myszkowski  
Alexander Pasko  
Giuseppe Patanè  
Tomas Plachetka  
Werner Purgathofer  
Tobias Ritschel  
Przemyslaw Rokita  
Mateu Sbert  
Benjamin Schmitt  
Peter Shirley  
Elena Šikudová  
Pavel Slavík  
Martin Šperka  
Kateřina Tátraiová  
Daniel Thalmann  
Zsolt Tóth  
Petr Vaneček  
Ivan Viola  
Michael Wimmer  
Pavel Zemčík  
Borut Žalik  
Jiří Žára

## Natural Water Shader

Andrej Mihálik and Roman Ďurikovič  
Comenius University in Bratislava

### Abstract

This work describes an implementation of optical phenomena on water surfaces. Despite high performance of current graphics hardware, shaders need essential simplifications and numerical approximation. Here we propose the implementation of common effects such as reflection, refraction and caustics. In order to have a simple and elegant implementation we have done coarse approximations to achieve real time animation, while still having a realistic appearance, which is important in real time simulation and games.

**Categories:** [Computer graphics]: Three-Dimensional Graphics and Realism

**Keywords:** refraction, reflection, caustics

### 3 Refraction and Reflection texture

The effect of reflection and refraction is achieved by two pass rendering algorithm. In the first pass, we split the whole scene into the afloat part and the underwater part by the horizontal clipping plane.

**Refraction texture.** To obtain refraction texture we keep the camera in its original position and render just the underwater part of the environment. See Figure 1. Finally, we copy frame buffer into our refraction texture resulting in refraction texture consisting of the image of the lake bottom.

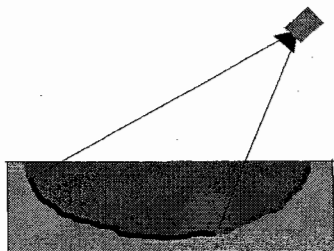


Figure 1. Refraction camera.

Consider the solid objects that are intersected by the plane. After splitting them and removing their afloat part, we may see back faces inside them. See Figure 2.

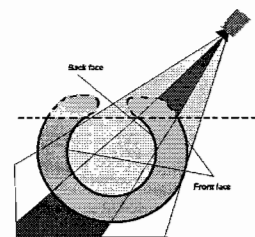


Figure 2. Back face culling

Note that appearance of back faces is not suitable. To avoid this artifact, we should enable back face culling. For future computation (e.g. deep) we may need depth texture of the lake bottom. Therefore, it is time to store the depth buffer to a texture at this step.

Once we have finished generating refraction texture, we can start to generate reflection texture in the second pass.

**Reflection texture.** To obtain this texture we must put the camera upside-down into its mirror position. We render the upper part of the split environment, as shown in Figure 3. After the rendering it, we store the frame buffer to the reflection texture.

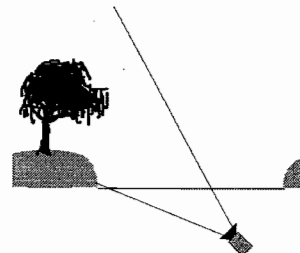


Figure 3. Reflection camera

This approach has the problem of missed texels because rays changed their direction on the water

surface due to refraction. This causes us to see on the surface a larger area than is actually stored in our refraction and reflection textures. See Figure 4. To restore the missing texture parts we extend the field of view before rendering of the reflection and the refraction texture. In our case extending the field of view about 50% was appropriate.

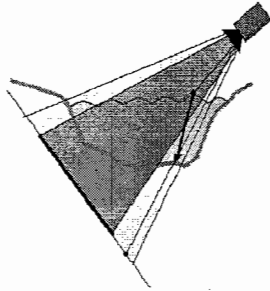


Figure 4. Missing texture parts.

### 5 Pixel shading

In this section we discuss how to obtain the colour of certain pixels on water surfaces. We assume the water surface is represented by the mesh as an input. Let's summarize that we have available surface refraction texture, depth texture of the bottom, surface reflection texture and global colour of the water at this moment. First, we render the whole scene without water using the fixed pipeline. Then we render the surface and compute the colour of pixels in fragment shader. Note that we have all the necessary stuff is in the shader now.

**Refracted colour from texel.** When we are treating a certain pixel, we have got coordinates of its corresponding point on the surface in camera coordinate system. We have also got surface normal  $N$  at this point. We can easily compute normalized eye vector  $E$ , because the camera position is the origin of camera coordinate system. Once we have the eye and the normal vector, we can compute the normalized refraction vector  $R$  using air-water refraction index. See Figure 6. The direction of this refraction vector is essential for determining which part of the bottom is mapped to current fragment of the surface. Note that the functions computing reflection and refraction vectors are defined in the OpenGL Shading Language. Since we do not perform the ray casting, we are unable to find where the ray with direction  $R$

hits the bottom. We propose to estimate the hitting point by following formula:

$$W = U + dv * R \quad (1)$$

where  $U$  is the point on the surface corresponding to current fragment in eye space and  $dv$  is length of the bold line from the surface to the bottom in Figure 6. The length  $dv$  is obtained by comparing  $Z$  coordinate of the surface and our depth texture. Large values of  $dv$  may cause some artifacts. Therefore we should clamp and scale the value of  $dv$ .

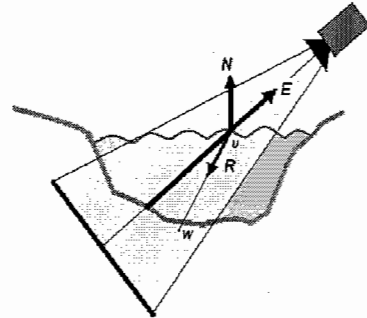


Figure 6. Refraction vector

Then we transform the coordinates of  $W$  to the texture coordinates of our refraction texture. Finally we obtain the refracted colour  $C_{\text{refract}}$  from the texel of the refraction texture at this coordinate. This transformation transforms coordinates of points in camera coordinate system to texture coordinates of corresponding points in the texture rendered from the same camera. It just determines where the specific object with certain camera coordinates lies within the texture.

To achieve more foggy appearance of water we must mix in the global colour of water. First we define the attenuation coefficient:

$$a := e^{(-d * k)} \quad (5)$$

where  $k$  is a suitable constant which determines transparency and  $d$  is the length of the ray from bottom to surface which is computed from our depth texture by comparing distance of surface from camera in  $Z$  direction and distance from bottom to camera. Now the new refraction colour is updated by:

$$C_{\text{newrefract}} := C_{\text{water}} + a * (C_{\text{refract}} - C_{\text{water}}) \quad (6)$$

where  $C_{\text{water}}$  is global colour of water. Figure 5 shows variance of the colour in the dependence of the deep.

**Reflected colour from texel.** Let us consider reflection. In this case we have to compute the reflection vector. See Figure 7. Letter  $R$  denotes normalized reflection vector. We determine reflection colour  $C_{\text{reflect}}$  by obtaining a texel from reflection texture. This texel lies at the texture coordinates obtained by transformation from coordinates of the following point:

$$W = U + c * R \quad (2)$$

where  $U$  is a point on the surface that corresponds to the current fragment in the camera coordinate system and  $c$  is a suitable scaling factor.

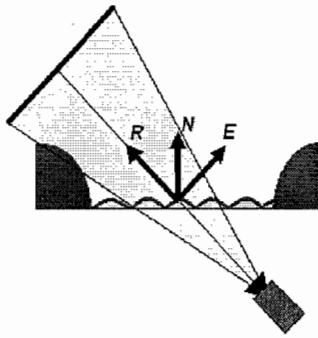


Figure 7. Reflection vector

This approach is a very coarse approximation. However, a visual result is plausible. See Figure 8.

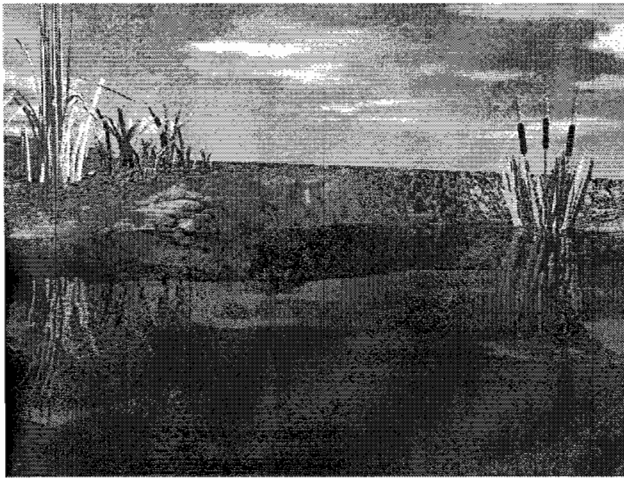


Figure 8. Reflection.

Final colour of the fragment is a combination of the new refraction colour  $C_{\text{newrefract}}$  computed above

and the reflection colour  $C_{\text{reflect}}$  obtained from texel in the reflection texture. To add Fresnel phenomenon we evaluate coefficient  $F$ .

$$F := (1 - (E.N))^q \quad (3)$$

Where  $q$  is a suitable positive constant and  $E.N$  is a dot product of normal and eye vector. Final colour is calculated by the following formula :

$$C := C_{\text{newrefract}} + F * (C_{\text{reflect}} - C_{\text{newrefract}}) \quad (4)$$

where  $C$  is the final colour,  $C_{\text{newrefract}}$  is new the refraction colour and  $C_{\text{reflect}}$  is the reflection colour.

```
for x = 0 to frustum_width
  for y = 0 to frustum_height
    u_coord = array[x][y].r;
    v_coord = array[x][y].g;
    illum = array[x][y].b;
    caustics_texture(u_coord,v_coord) += illum;
```

Thus the caustics texture consists of the values of light that reach a bottom from light a position. We map the caustics texture to the lake bottom.

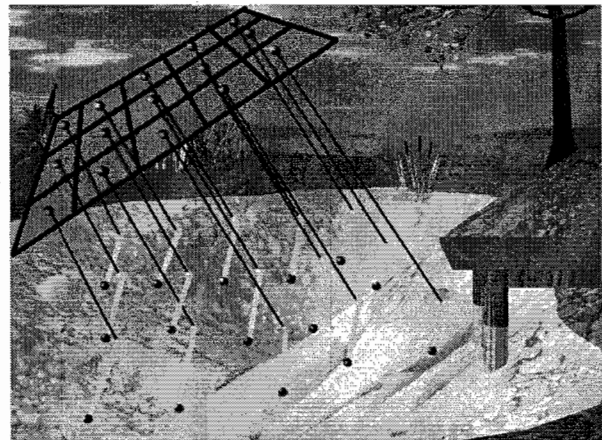


Figure 9. Mapping the photons from the orthogonal camera.

## 8 Results

We tested the proposed algorithm on multiple machines. Our lake scene (see Figure 10) consist of 131437 vertices and 85409 faces. The water surface was represented by the mesh with 2401 vertices and 4608 faces. The animation of the surface was performed using sine function. As the caustics and the surface mesh are performed on CPU, the framerates are not very high. See Table 2.

Referenced machine	caustics	
	on	off
Intel P4 3.0GHz, 1GB ram, nvidia 6600 (256MB)	10fps	19fps
Intel P4 3.0GHz, 1GB ram, nvidia 7600GT (256MB)	10fps	18fps
AMD Sempron 2600+ (1.6GHz), 1GB ram, nvidia 7950GT (512MB)	6fps	10fps
Intel Core2Duo 2.6Ghz, 2GB ram, nvidia 7950GT (512MB)	17fps	25fps
AMD Athlon 64 dual core 4400+ geforce 800GT (512MB)	9fps	14.12fps

Table 2. Framerates.

## 9 Conclusion

This work insists on the simplicity of implementation. Although the visual aspect of generated images are plausible, there are still some performance issues. The results are obtained from demonstrations implementing the above procedures in OpenGL. See Figure 10. For the reflection and the refraction textures the resolution is set to 512x512. By extending the field of view we may lose the resolution, which starts to be notable in case of 512x512. For caustics texture the resolution of 256x256 with enabled filtering was used. Some artifacts may appear in the water near the shore that can be fixed by putting the clipping plane a little bit above and adjusting the position of the bottom before the rendering of refraction texture. Rendering of the reflection texture is done by moving the clipping plane a little bit under the water surface.



Figure 10. Demonstration application.

## Acknowledgments

The author wishes to thank his advisor Roman Ďurikovič for the help during the course of this work. Frame rate measurements were done by Michal Červeňanský and Filip Zigo on their hardware. This research was supported by a Marie Curie International Reintegration Grant within the 6th European Community Framework Programme EU-FP6-MC-040681-APCOCOS.

## References

- [1] Belyaev V., Real-time simulation of water surface. GraphiCon-2003, Conference Proceedinks, pp. 131-138.
- [2] Sousa T., Generic Refraction Simulation. GPU Gems 2, NVIDIA Corporation 2005, pp. 295-305. ISBN-10:0321335597. ISBN-13:978-0321335593.
- [3] Galin, E., Chiba, N., Realistic Water Volume in Real-Time. Eurographics Workshop on Natural Phenomena (2006), pp. 1-8.
- [4] Tessendorf, J., Simulating Ocean Water, SIGGRAPH 2002 Course Notes #9 (Simulating Nature: Realistic and Interactive Techniques), ACM Press.
- [5] Musawir, S., Konttinen J., Pattaniak, S., Caustics Mapping: An Image-space Technique for Real-time Caustics IEEE Transactions On Visualization And Computer graphics.
- [6] Premože, S., Ashikhmin M., Rendering Natural Waters. Computer Graphics forum. Volume 20, number 4 (2001), pp189-199.
- [7] Blinn, J., Texture and Reflection In Computer Generated Images, CACM, 19(10), October 1976, pp 542-547.