EFFICIENT RENDERING OF MULTIPLE REFRACTIONS AND REFLECTIONS IN NATURAL OBJECTS

Stefan Seipel
Department of Information Technology, Uppsala University and Department of Mathematics and Computer Science, University of Gävle
ssl@hig.se

Anders Nivfors
Department of Information Technology, Uppsala University
nivfors@gmail.com

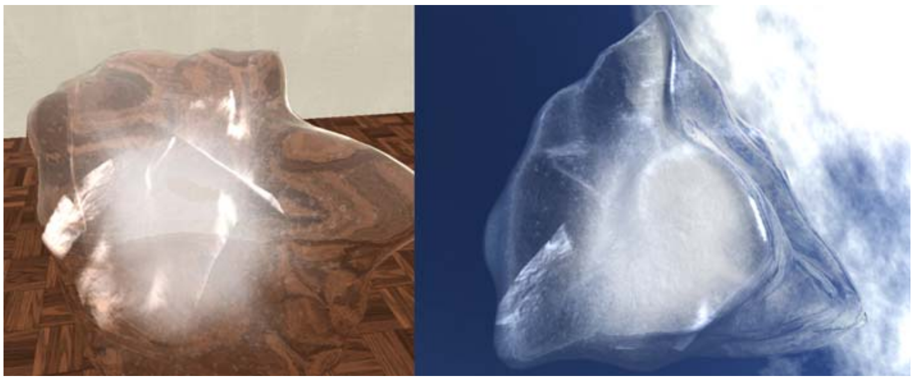
Figure 1: Rendering of ice showing multiple specular reflections and refraction

Abstract

In this paper we present a multi-pass rendering approach for approximating the effects of multiple refractions and specular reflections in transparent or semitransparent materials. These optical effects are typically found in natural materials like ice but also in glass artworks. The rendering technique proposed in this paper is intended to perform at real-time frame rates and aim at achieving naturally looking effects rather than simulating physically correct interaction between light and matter. Part of our method is an improved image space technique for clipping a geometry using the Boolean difference of two geometries in order to create internal surfaces of reflection inside transparent objects. It employs a number of generic cracks surface geometries which are clipped against the geometry to be rendered. Reflection and refraction effects on the ice are implemented by using environment mapping. Two-sided refraction is accomplished by combining the normal vectors of the front and back side of the ice object. Our method renders icy objects with convincing visual appearance in real-time on state-of-the-art graphics hardware.

Keywords: Computer Graphics, Modelling of Natural Phenomena, Illumination Models, Realtime Rendering Algorithms.

1. Introduction

Objects made of glass or naturally grown ice are impressive to look at. Part of the fascination these materials exert on the observer is due to the rich and extreme interaction effects between light and glass or ice objects. Ice shows, in comparison with glass, an even more complex visual appearance because its internal structure is often much more irregular and inhomogeniuos as opposed to glass. The simulation of these complex interactions between light and ice or glass comes almost for free in raytracing approaches. In the field of real-time computer graphics, however, rendering of ice has not been subject of intensive research.
Realistic rendering of natural phenomena in real-time has always been one of the most difficult tasks. In consequence, numerous papers can be found that describe implementation techniques for fire, smoke, clouds, fog, water etc. Yet, rendering of ice appears to be little explored. In this paper we summarize the most prominent visual characteristics of ice and what distinguishes ice from similar materials such as glass. We then present a multi-pass rendering approach to accomplish these characteristics in real-time using the GPU of a modern graphics card.

2. Related Work

To our knowledge, until now hardly any research has been published in relation to real-time rendering of ice objects. Little of the work related to ice in the field of computer graphics is mostly concerned with physically based techniques for offline rendering.
A physically based model for icicle formation was presented as early as 1993 [Kharitonsky and Gonczarowski 1993]. Methods for ice crystal formations have been looked into quite thorough [Kim and Lin 2003; Kim et al. 2004; Witten and Sander 19891]. Other work focusing at animations of melting materials, such as ice, has been presented by Carlson et al. [Carlson et al 2002] and Jones [Jones 2003].

Whereas most of the above mentioned papers describe offline methods which focus on the formation of ice, Kim and Lin [Kim and Lin 2003] presented techniques for ice formation and rendering that achieves interactive frame rates for low resolution models.
In the work we present below, we focus on visual appearance of icy objects rather than physically correct modeling. Therefore, in order to achieve real-time performance, we aim at techniques that utilize the graphics hardware. In particular we address tricks for rendering refraction and reflection effects found in ice structures.
Since ray tracing still can not be carried out in real-time, interactive graphics applications (i.e. games) generally use environment mapping (EM) [Blinn and Newell 1976], in order to collect light samples from in the nearby environment of an object to be rendered. EM is hardware supported, straight-forward to use and very fast. Environment mapping does, unfortunately, not support recursive reflections nor does it account for backside normal contribution.
In the paper "Interactive Refraction on Complex Static Geometry using Spherical Harmonics" [Génevaux et al 2006] Génevaux et al present a method that achieves realistic two-sided refraction. The method pre-computes some light paths which are used for approximations of the refracted paths during rendering. Since it relies on a pre-computation step, the method is suitable for static objects only. The runtime computations can be calculated on the graphics hardware, which allows the method to perform at interactive frame rates.

Another approach to approximation of two sided reflection was presented by Wyman [Wyman 2005]. By saving the normal vectors for the back facing polygons of the object to a texture along with the thickness of the object, both the back and front normal of the object can be found and accounted for when refracting the incident ray.

Khan [Khan 2004] presents an interesting method to achieve two-sided refraction when using EM by saving a normal cube map for the object. When rendering the surface using a shader, instead of sampling the environment map immediately, the normal cube map is sampled to obtain the normal for the backside of the object. With these two normal vectors the incident ray can be refracted two times hence accounting for both the back and front side of the object. The method suffers from artifacts in the refracted image. Being based on EM, neither Wyman's nor Kahn's method can handle concave objects very well.

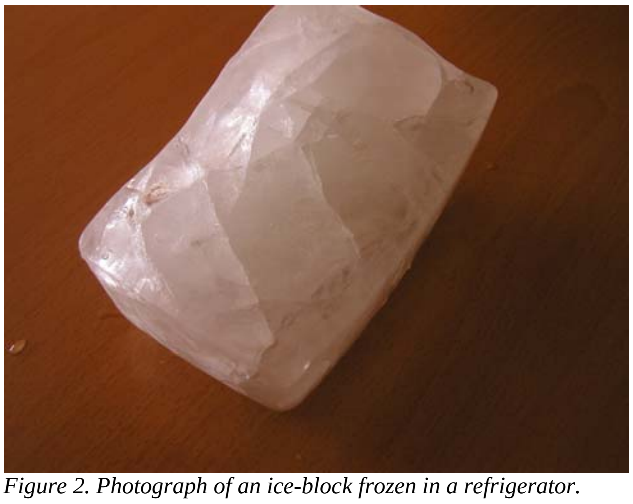
Figure 2. Photograph of an ice-block frozen in a refrigerator.

3. Visualizing features of ice objects

3.1 The look of ice

What renders the visual appearance of icy objects specific is a combination of several different optical effects which are due to the internal morphology and current surface properties of ice. The surface of icy objects can exhibit very different specular reflection properties. For instance, melting ice has a highly specular surface which gives rise to total reflections. In contrast, under cold conditions, the surface of ice is populated with thousands of microscopically small ice crystals yielding to a rather rough diffuse reflection when viewed from a distance. Ice is a semi-transparent material and its light transmission properties do change extremely depending on the internal structure of ice. This is further complicated due to the fact that the morphology of ice changes depending on external conditions like temperature and pressure. Obviously, there is no general model to describe the visual appearance of ice in all its possible different states. It is likewise difficult to recreate the visual appearance of ice by explicitly modeling all the microstructures in ice that give rise to the various visual appearances. Still, a reasonable approach to realistic rendering of ice is to identify a number of typical properties which tell the observer that it is ice she is looking at as well as to apply a number of simplified rendering techniques that approximate the most prominent visual attributes. Figure 2 shows a real ice object and, without any doubt, we can tell that we look at ice rather than e.g. glass. The most prominent features which we in informal interviews with six students identified to contribute most to the visual appearance of ice are:

a) Transmission of light from behind the object involving double refraction at front-face and back-face.

b) Specular reflections on the outer surface combined with specular highlights occurring at interior surfaces (cracks).

c) Irregular shape and bumpiness of the surface.

d) Air enclosed inside the ice causes milky white appearance.

This list of visual features is not meant to be all comprehensive, and not all of these characteristics must necessarily be found in ice at the same time. Yet, an integration of these four attributes into a material renderer should deliver a plausible visual result for most natural ice objects.

Methods for rendering of the visual properties mentioned under c) above are readily available under the term bump-mapping in computer graphics text books. Air bubbles and internal light scattering as mentioned under d) above, however, pose some bigger problem if we intend to implement them in a general way. In the course of our work, we have previously been using texture impostors. Hereby, we randomly place a number of transparently textured polygons inside the ice object. The texture images depict air bubbles at certain sizes and spatial frequencies. During a separate rendering pass, these texture images are rasterized and composited using alpha-blending. In this report, we do not describe the technique in more detail; instead we refer to [Nivfors 2006].

3.2 A straightforward approach to multiple refraction

In real-time computer graphics, environment mapping (EM) is used to accomplish effects of reflections and refractions. In our approach we use a cubic environment map that is sampled in the fragment shader of the ice object using the refracted and reflected eye vectors. In order to account for motion of the ice object or any nearby object in the scene, the environment map is updated dynamically if so necessary. The EM texture is updated by rendering the entire scene six times with a field of view set to 90°, rotating the camera to point in all six directions from the centre of the ice object.

When a ray penetrates a transparent object, the ray is refracted at two surfaces: when the ray enters the object and when the ray exits the object. In standard EM only the surface orientation (normal) of the front face is used for refracting the ray. To achieve two-sided refraction we use a simplified method similar to the one presented by Wyman [Wyman 2005]. We render the normal vectors of the back faces of the ice object to a texture using a separate rendering pass. Hereby, the vertex shader calculates the world space normal and the fragment shader normalizes and interpolates the result into a back-face normal buffer. A floating point texture is used for the result to avoid precision artefacts. The normal texture for the back-faces of the object is later sampled by the ice's fragment shader, allowing to simply blend the back and front faces' normal vectors using equation 1.

$$\mathbf{n}_\mathrm{r} = \mathbf{n}_\mathrm{f}(1-a) - \mathbf{n}_\mathrm{b}(a) \qquad (1)$$

where $\mathbf{n}_\mathrm{r}$ is the resulting normal
$\mathbf{n}_\mathrm{f}$ is the front face's normal in world space
$\mathbf{n}_\mathrm{b}$ is the back face's normal in world space, sampled from a texture
a is a scalar value in the range of [0, 1] indicating how much of the back face's normal that should be applied. A value of 0.33 is used for *a* in the screenshots in this report.

Using $\mathbf{n}_\mathrm{r}$ as surface normal, the eye vector (e) is refracted using equation 2. The eye vector, or view vector, is a vector from the viewer to the vertex on the surface and is calculated in the ice object's vertex shader. The interpolated result is used in the fragment shader.

$$\mathbf{r}_\mathrm{refr} = eta \cdot \mathbf{e} - \left(eta(\mathbf{n}_\mathrm{r} \bullet \mathbf{e}) + \sqrt{1 - eta^2\left(1 - (\mathbf{n}_\mathrm{r} \bullet \mathbf{e})^2\right)} \right) \cdot \mathbf{n}_\mathrm{r} \qquad (2)$$

where $\mathbf{r}_\mathrm{refr}$ is the refracted eye vector
eta is the refraction index of the media we are travelling from divided by the media we are travelling to. In this case air/ice = 1.0003/1.3091 ≈ 0.7641 [Nordling and Österman 1999].

$\mathbf{r}_\mathrm{refr}$ is then used to sample the environment map. Figure 3 shows the concept of the method.

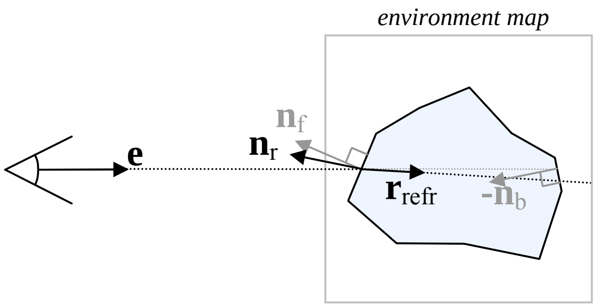

Figure 3. The idea behind two sided refraction using blended normal vectors.

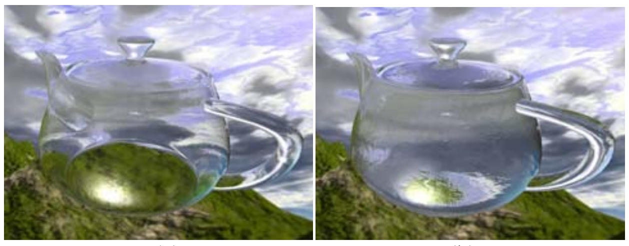

Figure 4. Model of the teapot with the ice shader demonstrating (a) two-sided refraction using blended normals compared with (b) one-sided refraction. Air and cracks are turned off.

So instead of following the incident ray and refracting it twice, a new normal is simply created by combining the back and front face's normal vectors. This is not a physically correct method but since ice is a highly distorted material this is a reasonable trade off for increased performance. Also, highly concave objects will not refract correctly since only one front face and one back face are accounted for.
For the purpose of reflection calculations, only the normal vector of the front face needs to be accounted for. So e is simply reflected using equation 3.

$$\mathbf{r}_\mathrm{refl} = \mathbf{e} - 2(\mathbf{n}_\mathrm{f} \bullet \mathbf{e}) \cdot \mathbf{n}_\mathrm{f} \qquad (3)$$

where $\mathbf{r}_\mathrm{refl}$ is the reflected eye vector.

Reflected and refracted light is sampled by using the reflection and refraction vectors as indexes into the environmental map. The resulting two color samples are then mixed using an approximation of the Fresnel equation similar to the one proposed in [Jensen and Golias 2001]. The Fresnel equation determines the ratio between the reflected and refracted ray's color contribution using a rather heavy equation. Our approximation is created with the intention of being fast and yet to produce a convincing visual result rather than being physically correct (equation 4).

$$f = (1 - |\mathbf{e} \bullet \mathbf{n}|)^3 \qquad (4)$$

where *f* is the Fresnel term
e is the eye vector to the vertex
n is the vertex's normal.

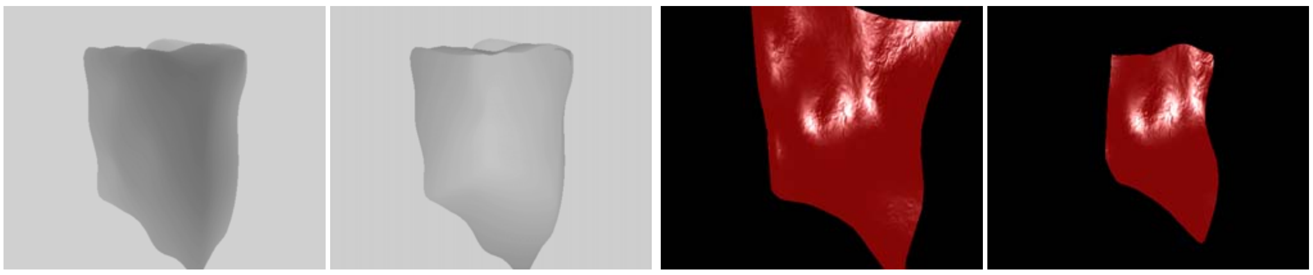

Figure 5. The depth buffer saved to a texture for the (a) front faces and (b) back faces of the ice object. (c) The crack model without clipping. (d) The final clipped crack. The cracks are colored red on these pictures for better visibility.

Equation 4 is calculated in the ice object's vertex shader and f is interpolated and sent to the fragment shader where it is used for mixing the reflection and refraction colors sampled from the environment map (equation 6). The result is shown in Figure 4.

3.3 Multiple specular reflections

The occurrence of cracking surfaces inside solid ice object is very typical for natural ice objects. It is at these internal, often curved and irregularly shaped crack surfaces where incident light is reflected towards the observer. An example of these internal specular reflections can be seen in the left part of a real ice block in figure 2. To accomplish this light contribution due to internal specular reflections, we use a rendering pass that only calculates a specular highlight for internal crack surfaces and which uses bump mapping for crack surfaces contained inside an ice object. Hence, if no specular highlighting occurs, the crack is completely transparent. Crack surfaces are rendered without back face culling to make them visible from both sides as an ice object is rotated. So, how do cracks appear inside ice objects?

Unless they are explicitly defined by the model creator, we need to define a means of adding a certain number of cracking surfaces into the object. Generally, it would be preferable to add an arbitrary number of cracks dynamically to any ice model without explicitly defining them as subcomponents of the object geometry. We therefore propose to use a number of generic crack models of sufficient size, to be subsequently used for rendering of any ice model. These crack models can be positioned and randomly oriented inside an ice model. Knowing the bounding box of an ice object, the scale of the generic crack geometries can easily be adjusted to cover the volume of the object. The difficulty lies in performing a volumetric clipping operation, that prevents crack surfaces from exceeding the ice objects actual boundaries.

Constructive Solid Geometry (CSG) [Goldfeather 1986] is a way of constructing solids by combining existing solids using the Boolean operations such as union, difference and intersection. Various techniques of performing interactive CSG using the default depth buffer of the fixed point rendering pipeline exist already (e.g. [McReynolds and Blythe 1996]). With the possibilities provided by programmable graphics cards, we make use of a more straightforward screen space method. It uses a render-to-texture approach for both frontal and rear faces and employs a specific fragment shader which evaluates depth values of the crack geometries in order to perform interactive CSG.

Consequently, cracks are created at rendertime by first randomizing the rotation and position, preferably near the centre of the ice. Then for each frame the depth values of the back and front faces of the ice model are rendered to a texture (Figure 5 (a, b)). In order to optimize the precision of the depth values, the near plane of the viewing frustum is aligned with the front most distance of the object's bounding volume and the far plane is aligned with the farthest distance of the object's bounding volume. Then the cracks can be rendered as usual with their shader that performs bump mapping and specular highlighting. In the fragment shader some instructions are added to sample the two depth textures and to compare the current pixel's depth value with the corresponding depth interval. If the value is not contained within the interval of the two sampled values nothing is drawn since then the pixel is outside the ice object. Figure 5 (c, d) shows a generic crack geometry rendered without and with clipping against the objects boundary. Regardless of the number of crack surfaces, only three render-to-texture passes need to be performed: the ice object's front and back faces' depth values and the result with the clipped cracks.
This method is based in the assumption that the object that should be clipped (the crack) is a surface model and not a solid. Furthermore, only Boolean intersection is performed. So the method can not be considered a full alternative method for interactive CSG, although it can probably be altered to be more general. The method only works correct for convex objects, otherwise artefacts will show. The method fulfils the criteria of an appropriate clipping method for the cracks and can handle an arbitrary amount of cracks.

4. Performance

We implemented an application to demonstrate and evaluate the proposed method using C++/OpenGL. For all passes that render to a texture the framebuffer object (FBO) extension was used [Juliano and Sandmel 2006]. When using FBOs the color or depth values can be written directly to a texture instead of being copied from the frame or depth buffer. Shader programs were written in Cg (version 1.4.1).
The resolution of all the textures attached to the FBOs (for the depth buffers, crack bumps, and air bubbles) was set to 512x512 for the measured benchmarks as well as in the screenshots in this report. The only exception was the texture keeping the backface normal vectors which was set to the same resolution as the application window, 1024x768, to avoid filtering artefacts of the normal vectors. In all tests the ice covered approximately 1/5 of the screen. Unless stated otherwise, in all tests all effects were enabled, with 19 air planes and 2 cracks rendered and clipped in real-time using our method.

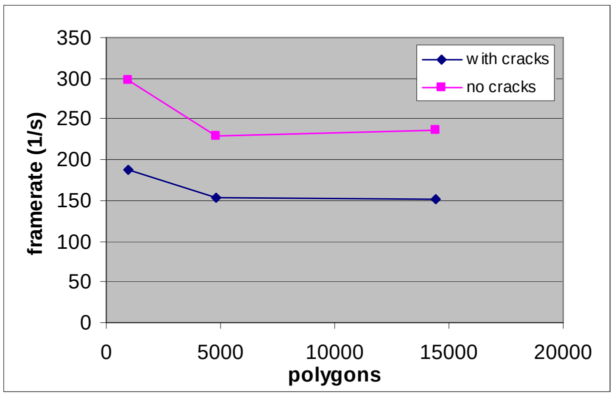

Figure 6. Frame rates for increasingly complex objects rendered without cracks and air-textures (upper curve) or with cracks and air-textures (lower curve), respectively.

For the performance tests we used an AMD Athlon 64 3700+ 2.2 GHz with 1 GB of RAM; the tested graphics card was nVidia's GeForce 6600GT with 128 MB of video memory. We measured framerates for three different objects: A sphere with 960 polygons, an irregular ice-cube with 4800 polygons, and the teapot counting 14400 polygons. Figure 6 illustrates obtained framerates for a static scene.

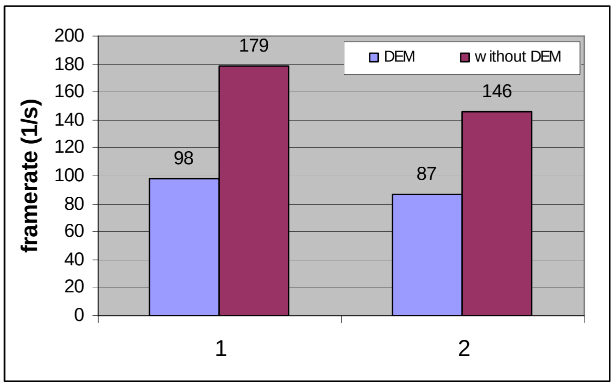

Figure 7. Frame rates without two-sided refraction (category=1) and with the approximation for two-sided enabled (category=2).The blue bars indicate rendering performance with dynamic update of the environment cube map. Purple bars are for a static environment map.

We tested how rendering performance is affected when our approximation of two-sided refraction based on normal vector blending is used. To that end, the ice cube object (4800 polygons) seen in figure 1 was rendered in a simple landscape environment containing 2340 polygons. It was rendered in a dynamic scene which requires a continuous update of the environment cube map. For comparison, we rendered the scene also without dynamic update of the environment cube map i.e. a static map. The results of these measurements are shown in figure 7.

5. Discussion and conclusions

In the final implementation of our ice renderer, the proposed rendering method uses many rendering passes and the ice shader accesses no less than six texture maps including the environment cube map. Yet, this did not affect the performance too much. This is partly due to the use of frame buffer objects but also thanks to the very powerful GPU that performs all calculations. Figure 6 shows that high frame rates are achieved even with fairly complex ice objects. The teapot has three times as many faces as the ice cube but the frame rate is barely affected, which suggests that the rendering method is fill bound. Apart from air bubbles, one of the most important features in our renderer is cracks in the ice. When using the methods proposed in this paper the frame rate drops about 26% compared with rendering without air and cracks. The performance mainly depends on two factors: how many pixels the ice object occupies on screen and the resolution of the textures attached to the FBOs. If the ice covers approximately 1/5 of the screen 148 fps is achieved, compared to 67 fps when the ice fills the entire screen. This is because the fragment program for ice is complex and requires many texture accesses. While this might be seen as a problem for large icy objects, it makes on the other hand distant or small pieces of ice very cheap to render.

Our clipping method fulfils the goal of a method for adding an arbitrary amount of cracks to any convex object. However, a possible future improvement would be to implement some pre-computation step which clips the cracks once and saves them as new models. In that way, cracks would not have to be clipped for each rendered frame yielding increased performance. The method could even be constructed to handle concave objects. When testing our ice rendering programs, we experimented with letting the cracks affect the refracted image. We did this by modifying the method for two-sided refraction in such way, that the crack normal vectors would also distort the final surface normal. To that end cracks were rendered on top of the ice object's back faces when the backside normal texture was rendered. The cracks seemed to affect the final result a bit too much, especially when many cracks were used. Still, in nature the cracks do affect the refracted image, so if pre-computed CSG is to be implemented a modified version of this technique might prove useful. One small drawback of pre-computing the cracks is that if the ice is to be animated, e.g. due to melting, the computation of new vertices for the cracks in real-time would be too expensive.

In regard to our approximation of two-sided refraction using normal vector blending, the observed frame rates shown in figure 7 demonstrate that the extra performance penalty is, with approximately only 10%, relatively low.

In our rendering model, cracks and air in the ice are coarsely approximated rather than being based on physical correct models. Yet, the visual result is convincing, in particular for animated scenes. The pictures obtained (see e.g. figure 1) show typical characteristics of ice at high frame rates using standard hardware. This makes the method suitable for interactive applications such as computer games.

We have presented a comprehensive method for real-time rendering of ice incorporating multiple refraction and reflection on internal surfaces. Given a more or less convex geometry, it creates an ice object filled with air particles and bubbles. The number of cracks added to the ice can be specified in real-time. The environment and light sources are dynamically reflected by the bumpy surface, and a refracted environment can be seen through the ice. Our objective was to recreate visual effects of ice in real-time. The demonstrated results show the relevance of these visual characteristics, was well as the obtained frame rates that confirm real-time performance on standard graphics hardware.

References

BLINN, J. F., and NEWELL, M. E. 1976. Texture and Reflection in Computer Generated Images. Communications of the ACM v. 19 i. 10, ACM Press, New York, 542-547.

CARLSON, M., MUCHA, P. J., VAN HORN III, R. B., and TURK, G. 2002. Melting and Flowing. In Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation, ACM Press, New York, 167-174.

GÉNEVAUX, O., LARUE, F., and DISCHLER, J. 2006. Interactive Refraction on Complex Static Geometry using Spherical Harmonics. In Proceedings of the 2006 symposium on Interactive 3D graphics and games, ACM Press, New York, 145-152.

GOLDFEATHER, J. 1986. Fast Constructive Solid Geometry Display in the Pixel-Powers Graphics System. In Proceedings of the 13th annual conference on Computer graphics and interactive techniques, ACM Press, New York, 107-116.

JENSEN, L. S., and GOLIAS, R. 2001. Deep-Water Animation and Rendering. Gamasutra article. http://www.gamasutra.com, April 2006.

JONES, M. W. 2003. Melting Objects. In Journal of WSCG 2003, 247–254.

JULIANO, J., and SANDMEL, J. (contact persons). 2006. Framebuffer object extension specification. http://oss.sgi.com/projects/ogl-sample/registry/EXT/framebuffer_object.txt, May 2006.

KHAN, A. 2004. Dual-sided Refraction Simulation. Shadertech Contest, Summer 2004. http://www.shadertech.com/contest, April 2006.

KHARITONSKY, D., and GONCZAROWSKI, J. 1993. Physically based model for icicle growth. The Visual Computer: International Journal of Computer Graphics, Springer-Verlag New York Inc., Secaucus, 88-100.

KIM, T., and LIN, M. C. 2003. Visual Simulation of Ice Crystal Growth. In Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation, Eurographics Association, Aire-la-Ville, 86-97.

KIM, T., HENSON, M., and LIN, M. C. 2004. A hybrid Algorithm for Modeling Ice Formation. In Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation, ACM Press, New York, 305-314.

MCREYNOLDS, T., and BLYTHE, D. 1996. Programming with OpenGL: Advanced Rendering. Course notes from ACM SIGGRAPH 1996 Course 23, ACM Press, New York, 31-42.

NIVFORS, A. 2006. Real-time rendering of ice. Masters Project Thesis in Computer Science. Department of information Technology, Uppsala University.

NORDLING, C., and ÖSTERMAN, J. 1999. Physics handbook for Science and Engineering. Studentlitteratur.

WITTEN, T. A., and SANDER, L. M. 1981. Diffusion-limited aggregation, a kinetic critical phenomenon. Physical Review Letters 47, 1400–1403.

WYMAN, C. 2005. An Approximate Image-Space Approach for Interactive Refraction. In Proceedings of ACM SIGGRAPH 2005, ACM Press, New York, 1050-1053.