Project in EDAN35, High Performance Computer Graphics — Bubbles —

Johnny Dang[*] Rikard Olajos[†]

Lund University
Sweden

Abstract

We generated realistically looking bubbles using OpenGL and the RenderChimp framework. The bubbles are using translucency, noise functions, and dynamic cube maps.

1 Introduction

The goal of the project was to generate photorealistic bubbles. Our motivation was that we wanted to do something that had not been done before in previous years of the course and was on an appropriate difficulty level.

2 Algorithms

We identified three different effects that had to be implemented for generating the bubbles. Those were the translucency, the rainbow colour pattern often seen in soap bubbles, and the reflection of the environment on the outside as well as the inside of the surface of the bubbles. One could think that refraction is also needed when simulating bubbles, but by looking at various pictures of soap bubbles, we concluded that the refraction in the bubbles was unnecessary to implement, as it had almost no impact on the bubbles. Physically, this is due to the infinitesimally thin bubble surface. See Figure 1 for real life bubble.

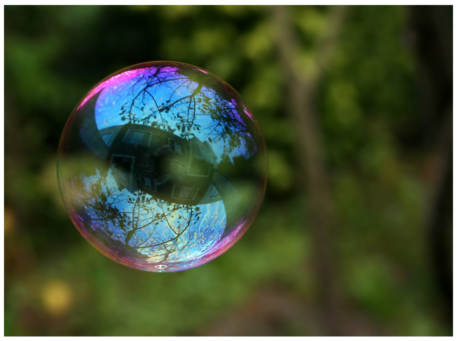

Figure 1: A real life soap bubble. By using many pictures like this one, we decided on how and which algorithms that should be implemented. Source: [Alvesgaspar 2007].

2.1 Translucency

We wanted the bubbles to be more transparent in the middle while still having a clear and coloured edge, like the bubble in Figure 1. This was implemented in the fragment shader by letting the alpha value depend like this

$$\alpha = const. - f(N \cdot V),$$

where $f(N \cdot V)$ is a function that depends on the scalar product between N, the normal vector, and V, the view vector.

[*]e-mail: tfy11jda@student.lu.se
[†]e-mail: lat11rol@student.lu.se

Looking at Figure 1 we can see that the bright reflections are dominant and that the sky is not only reflected on top of the bubble but on the inside as well. This meant that we needed to draw the inside of out bubbles also; wherefore culling was disabled.

To get the transparency right, the following equation was used for the alpha blending

$$RGB_d = A_s \times RGB_s + (1.0 - A_s) \times RGB_d, \qquad (1)$$

where RGB_s is the source RGB value outputted from our shader, RGB_d is the RGB value of the destination, i.e. the value on the framebuffer, and A_s is the alpha value of the source. The equation is a simple interpolation between the destination value and the source value, using the alpha of the source as the interpolation variable [Graham 2013].

Equation 1 can be rewritten in the following manner

$$RGB_d = RGB_d + A_s \times (RGB_s - RGB_d). \qquad (2)$$

Here we can see a problem with the transparency. Lets say that we have render a bubble to the framebuffer and the would like to render another one behind it. Because the last part in equation 2 ($RGB_s - RGB_d$) does not commute, the second bubble that we are trying to render behind will be rendered on top instead [Graham 2013]. This result is nasty.

The solution is to sort the bubbles in a order so that they are rendered from the furthest to the nearest and therefore using the blend equation in the right way.

2.2 Noise

Ian McEwan's simplex 3D noise function was used as noise function [Ian McEwan 2011]. The simplex noise was used at two points in the generation of the bubbles. To create a bit of a wobbliness, that occur in large soap bubbles, we used simplex noise passed through a sine function to alter the positions of the vertices in the vertex shader.

We coloured the bubbles in the fragment shader using a premade colour gradient texture, that we constructed as close as possible to the colours of real life bubbles. See Figure 2.

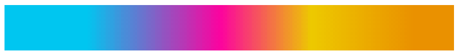

Figure 2: The colour gradient used for colouring the bubbles.

The colours of the bubbles are made up by two parts. The first part is a radial component that creates a radial colour gradient by mapping the same value that the alpha channels uses to the colour gradient in Figure 2. The second part of the colour is based on the simplex noise which gives a bit of variation to the colour of the bubbles. These two parts were then mixed into a colour value. This procedure gives the bubble a colour similar to the real bubble in Figure 1, i.e. a colour that gradually changes closer to the edge but with some variation.

2.3 Dynamic Cube Map

In the previous course, we simulated a reflective surface by applying a cube map texture using six static images taken in beforehand from the same location in six different directions. This can give a quite realistic reflection, but because the images are static, the illusion breaks down as soon as the the scene around the object changes or when the object moves around.

To handle the changing reflections, we choose to implement dynamic cube mapping, where instead of having six static images, the images are taken continuously. Simply put, during the computations for every frame, six different cameras, which are looking in different directions, are moved to the centre of a bubble. The scene is then rendered once for every camera and these frames are then stored as textures for a cube map. The cube map is then stuck onto the bubble in the same way as with the static cube mapping.

The implementation of the dynamic cube mapping was of course not this straight forward. RenderChimp has an implementation for storing the rendered scene as simple textures, but unfortunately there is no equivalent function for creating cube map textures. Therefore we had to write all the code to create cube map texture ourselves [OpenGL.org 2014].

We started by generating a framebuffer, an appropriate texture to hold the cube map texture, and the six cameras. The cameras were moved to the position of the bubble and the scene would be rendered for each camera. Because the final scene used deferred shading, in which different parts of the scene are rendered separately into different framebuffers and then rendered once again together, we could not just bind our framebuffer and render the scene, as it would just get overridden. Instead, the different parts had to be rendered first, and just before rendering the whole scene together, we would bind our framebuffer instead of letting the program render onto the screenbuffer. Afterwards the cube map texture is assembled and loaded into the shader for the bubble. This process is done for every bubble and then the whole scene including the bubbles are rendered one final time for the view camera.

3 Results

In Figure 3 the radial colour components is presented. We can clearly see how the colour changes towards the edges. The noise generated colour component can be seen in Figure 4. This shows that the colour gradient is continuous over the whole surface of the bubble. The reflections generated by the dynamic cube mapping is shown in Figure 5. The final result is presented in Figure 6. The different components are blended with different weights.

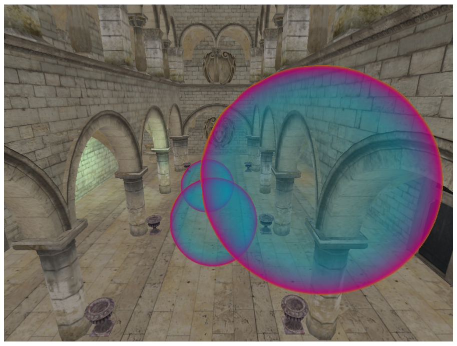
Figure 3: Bubbles with only the colour generated by the radial component as texture.

We found that when using just the dynamic cube map texture together with the alpha channel the resulting bubble looks similar underwater bubbles. See Figure 7.

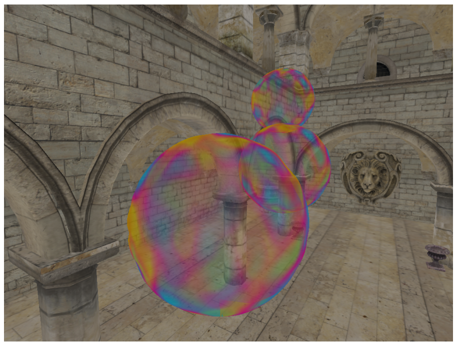
Figure 4: Bubbles with only the colour generated by noise as texture.

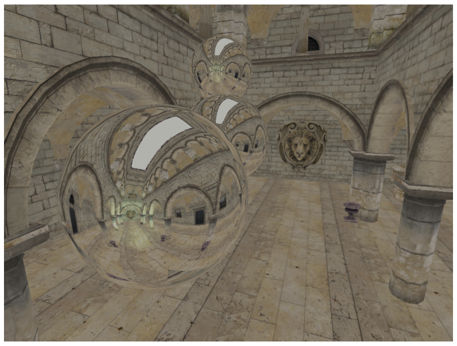
Figure 5: Bubbles with only the dynamic cube mapping as texture.

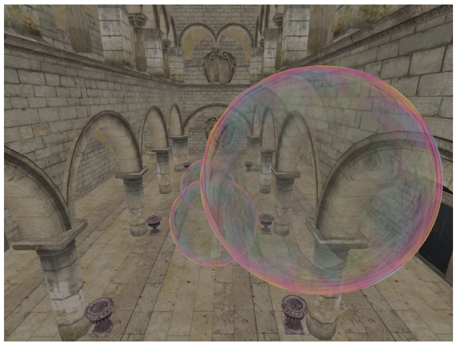
Figure 6: The final result with all the different components together on the bubble.

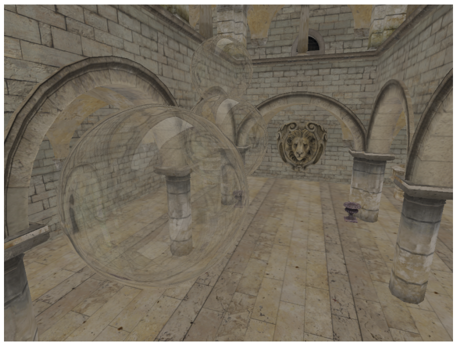
Figure 7: Bubbles with only the dynamic cube mapping, but with a varying alpha as described earlier, as texture.

4 Discussion

The translucency worked well after the sorting. However, because we disabled culling on the bubbles and we only sorted the bubbles but not the fragments within the bubbles, the far side of a bubble might be rendered on top of the near side when viewed from certain directions. This can be solved by sorting the triangles within the bubble or since the bubbles are spherical, the bubbles could just simply be rotated so that the right side is always facing the camera. A rotation like this would not interfere with the noise functions as they depend on the world space coordinates.

The noise function works quite well for simulating the colouring and the wobbling. The use of the noise function and the premade colour gradient was more of an esthetical approximation to get a realistic look, though it is not based on any scientific observations.

The dynamic cube map works well for the reflections but is in its very nature computationally very expensive. However, this can be alleviated a bit by using various approximations and cheats. For instance, not every cube map has to be computed for every frame. You could for example update every cube map every nth frame or you could just update a few cube maps every frame. Another way to optimise it could be to compute the cube map for one bubble and then apply the same cube map to bubbles in the near vicinity.

References

ALVESGASPAR. 2007. Wikipedia.org. `http://en.wikipedia.org/wiki/Soap_bubble#mediaviewer/File:Reflection_in_a_soap_bubble_edit.jpg` [18 Dec 2014].

GRAHAM. 2013. Order Independent Transparency, OpenGL Super Bible. `http://www.openglsuperbible.com/2013/08/20/is-order-independent-transparency-really-necessary/` [8 Dec 2014].

IAN MCEWAN. 2011. Simplex 3D Noise. `https://github.com/ashima/webgl-noise/blob/master/src/noise3D.glsl` [8 Dec 2014].

OPENGL.ORG. 2014. Framebuffer Object Examples. `https://www.opengl.org/wiki/Framebuffer_Object_Examples` [8 Dec 2014].