

# Under the River

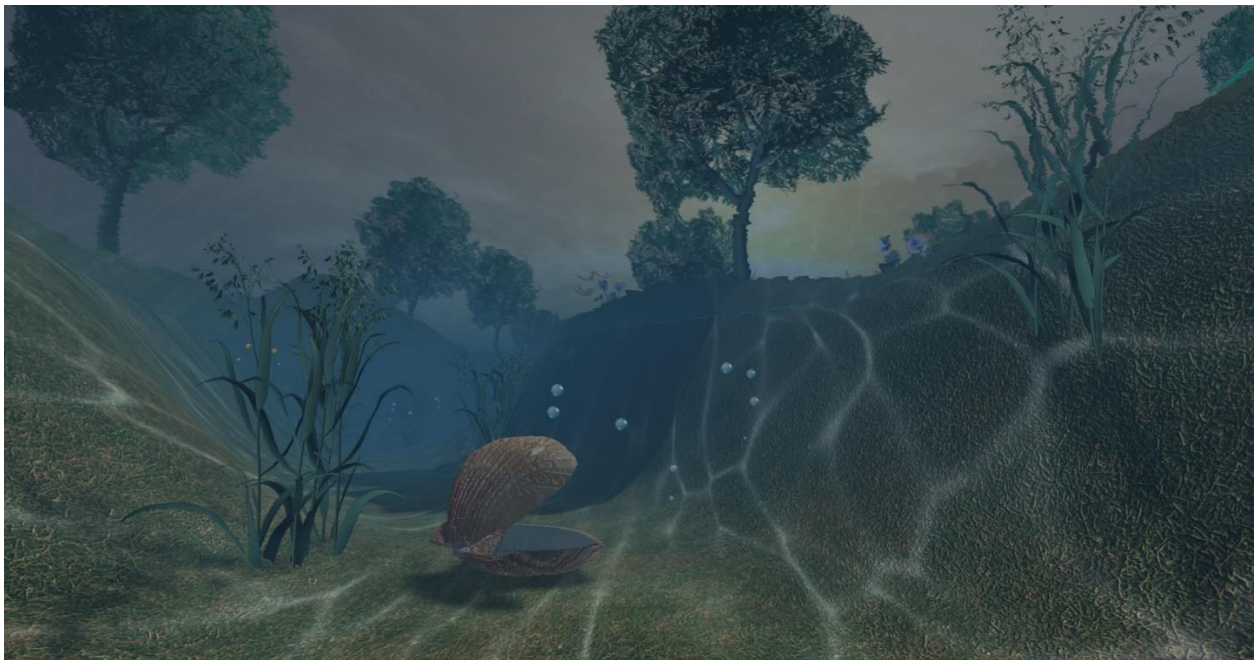
---

186.140 Echtzeitgraphik 2017W

Lisa Kellner (1428183) & Maria Schimkowitsch (0525297)

'Under the River' extends the game 'Samy the Salmon' from the CGUE lab course with some further effects. For the graphics demo all relevant game elements, like obstacles, collectables, health or gathered points, were removed. Collision detection was also disabled.

We tested our implementation on **NVIDIA** graphics cards.



## Implementation details

---

### Illumination, materials & textures

---

The scene is lit using the Phong illumination model and a static directional light, simulating the sun.

The terrain, water, plants, shell and particles have all different materials and textures applied, some using normal mapping additionally, like the terrain or the shells.

### Camera

---

The application starts with a predefined automatic moving camera. The required locations and orientations are loaded from file, which were previously sampled and stored using a debug camera, which can be moved freely using controls, see the table below. For the automatic movement, the positions are interpolated using a Catmull-Rom spline and the orientations are interpolated using a spherical interpolation of Quaternions (squad).

Source:

Akenine-Möller, Haines & Hoffman. 2008. *Real-Time Rendering*. A K Peters, LTd (Natick MA) p.77 and pp.589

It is possible to switch during runtime between the debug and automatic camera by pressing the key "C". You can move the debug camera, and sample the required control points for the automatic camera, with the following controls:

Key	Effect
<b>W,A,S,D</b>	Forward, left, backward, right
<b>Mouse</b>	Pan camera
<b>ESC</b>	Quit
<b>P</b>	Sample current position and orientation
<b>I</b>	Store the sampled points to file
<b>C</b>	Switch between automatic and debug camera
<b>T</b>	Tessellation on/off

### Underwater caustics

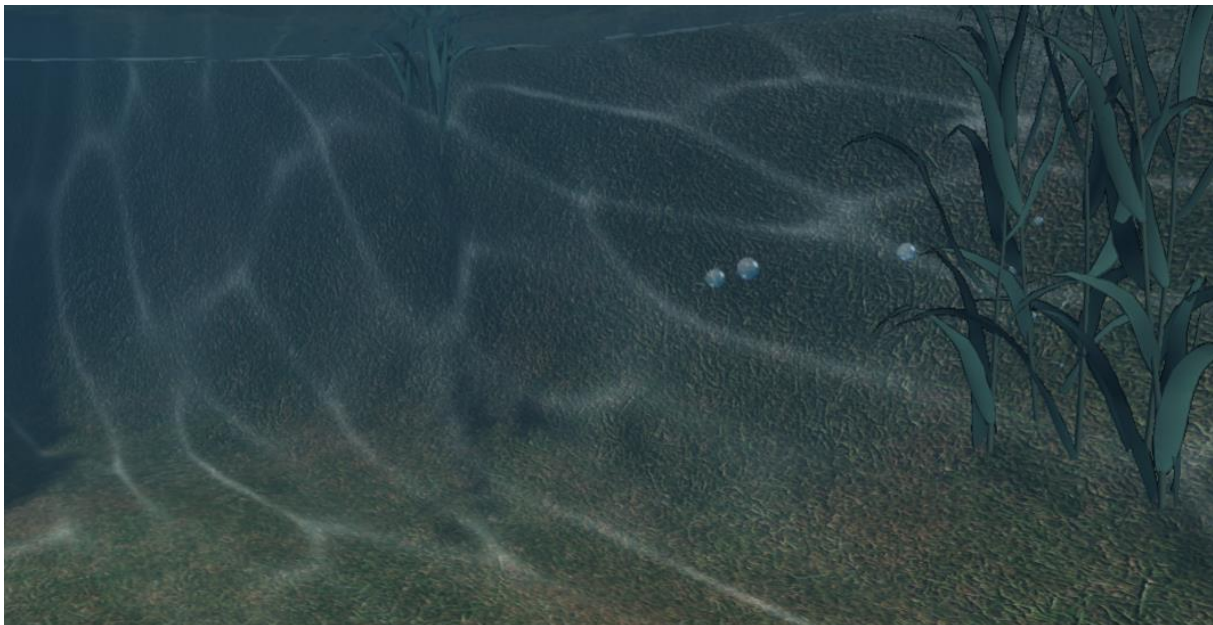
To compute underwater caustics, a ray (note: it's not a ray, but we can think of it as one for better understanding) is shot from the surface of the terrain through the water surface. The water surface bends the ray and then the intersection point of the bend ray with the light map is computed. This position is sampled from a caustics map and gives the impression of moving caustics.

To bend the ray, the derivatives of  $x$  and  $y$  of a wave function are computed. These are parts of the normal vector of a wave and can be used to get the new direction of the ray after hitting the water surface.

Source:

[http://developer.download.nvidia.com/books/HTML/gpugems/gpugems\\_ch02.html](http://developer.download.nvidia.com/books/HTML/gpugems/gpugems_ch02.html)

Underwater caustics:



### Tessellation Displacement Mapping

One object's appearance beneath the water surface is altered using tessellation displacement mapping.

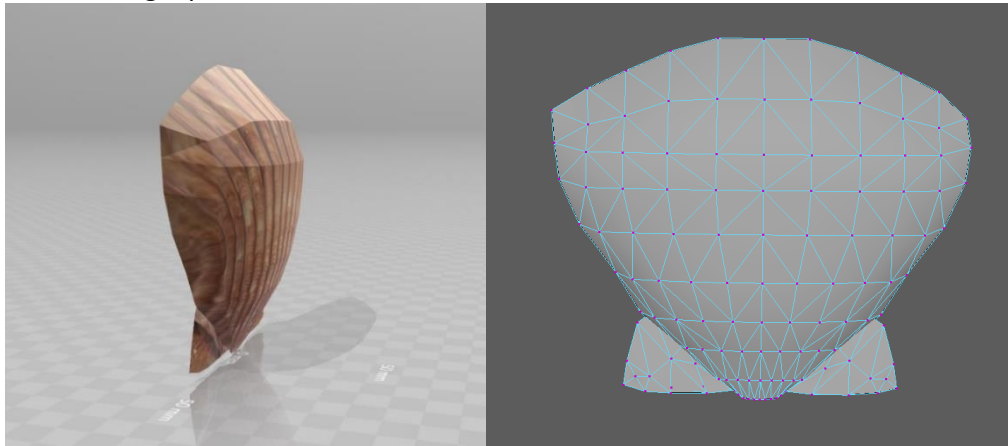
The original model, that was generated using Maya 2018, shows a flat scallop half. That scallop is further tessellated, using a constant inner and outer tessellation level of 12, and displaced along the new vertices' interpolated normals using a height map and a predefined height scaling factor. The height map was generated in Photoshop. The scallop's surface is further changed using normal mapping and specular highlighting, in addition to texturing.

In the application, two halves are combined to create the scallop. The lower half stays fixed, while the upper half is rotated between two predefined angles. The angle is updated using a timer.

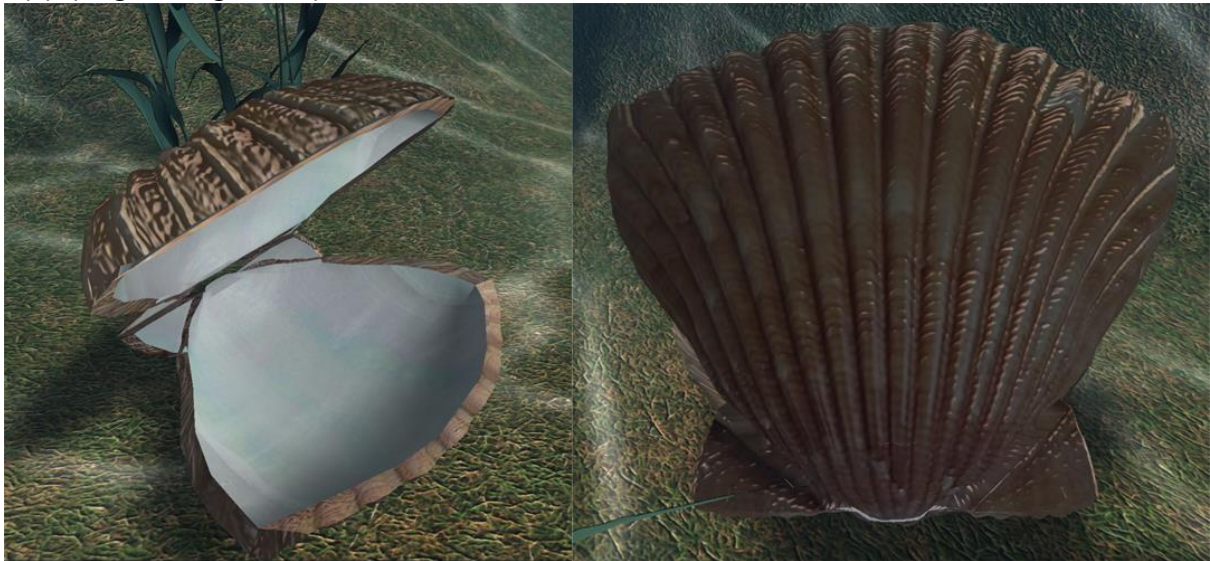
Source:

<http://prideout.net/blog/?p=48> (tutorial shows how tessellation shader are used generally, displacement mapping was developed from that point onwards)

Original scallop Model (textured on the left and with highlighted polygon outline and vertices on the right):



Tessellated Scallops: Scallop ripples generated by tessellating the model and applying a height map



## GPU Particles

The GPU particles are a quite straightforward implementation of a standard particle system, using acceleration, velocity and position buffers. All buffers are updated in a compute shader and the geometry for the billboards is created in a geometry shader.

We implemented two particle systems:

- **Underwater bubbles**

These particles also have a time-to-live buffer, which is counting down and



when it's at zero the particle will die and a new one will spawn instead. This is approximately timed when the bubbles hit the water surface.

Here we used billboards, which are always looking into the camera, to simulate the bubbles. To achieve this, the billboard corner positions are computed in the camera space.

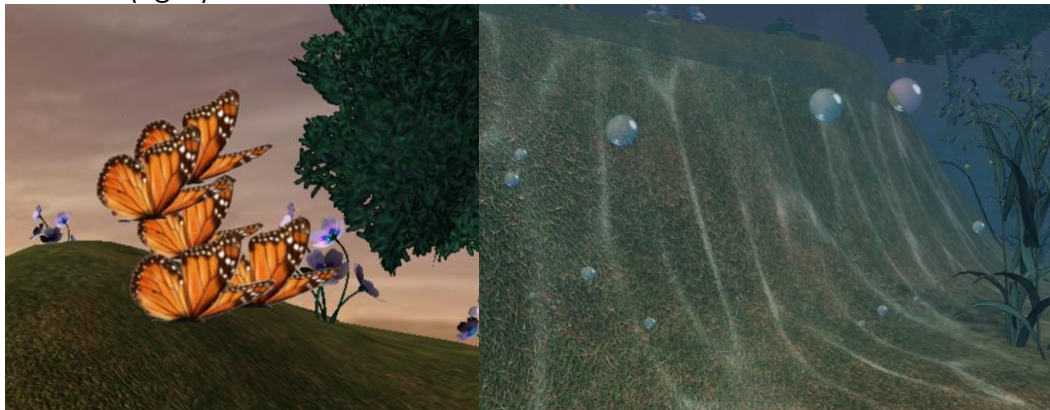
Source: <http://www.geeks3d.com/20140815/particle-billboarding-with-the-geometry-shader-glsl/>

- **Butterflies**

The butterfly billboards were computed in the world space, so they are not aligning with the camera. Here 6 vertices for the butterfly geometry are created. 2 for the left wing corners, 2 for the middle of the butterfly and another 2 for the right wing corners. To simulate the stroke of the wings, the particle center is translated to the origin, then the position of the wing corners are computed and rotated around the y-axis (wing movement). Then the new rotated points are transformed back.

Source: Jacobo Rodríguez (2013). GLSL Essentials, Enrich your 3D scenes with the power of GLSL!

GPU particles: Butterflies above the water surface (left) and underwater bubbles (right).



*Note:* The following effects were implemented in the prior CGUE lab course.

### **Shadow Maps**

The scene is rendered from the view of a single direction light source (the sun). The objects and plants shadow the scene according to the resulting shadow texture.

### **Water**

For the refraction and reflection effect two textures are generated that are sampled when rendering the water surface itself. The refraction texture is generated without the need of an additional pass, by simply rendering to a second color attachment when rendering the scene. The reflection texture is obtained in a second pass, rendering the scene mirrored by the water plane clipping geometry below it (also the backface culling must be inverted). When rendering the surface both textures are sampled, using a dUdV map to get some distortion. Both values are then

combined using Schlick's approximation of the Fresnel factor. Finally, some specular highlights are added based on the water's normal map.

### Normal Mapping

Normal Mapping is implemented, by using a normal map texture to enhance the details of the scene. This effect can especially be seen at the terrain. The grass seems to be more detailed and appears more realistic.

### Source of models/textures

---

- Terrain: Model generated with Blender
- Underwater plants: <http://xfrog.com/product/AG01.html>
- Flowers: <https://www.turbosquid.com/3d-models/pink-primrose-flowering-3d-obj/516226>
- Trees: <https://www.turbosquid.com/FullPreview/Index.cfm/ID/1206038>  
Royalty Free License - All Extended Uses
- Scallops: Image source: <http://www.foodofy.com/scallops.html>, 15.12.2017  
Normal, Displacement and Specular maps were generated with ShaderMap and Photoshop  
Model generated in Maya 2018 (Educational license)
- Bubble texture: Texture generated with Photoshop
- Butterfly texture: [https://www.freepik.com/free-vector/butterfly-realistic-isolated\\_1530375.htm#term=butterfly&page=1&position=7](https://www.freepik.com/free-vector/butterfly-realistic-isolated_1530375.htm#term=butterfly&page=1&position=7)  
Created by Macrovector - Freepik.com

### Additional libraries

---

The Aardvark platform (<https://github.com/aardvark-platform>) from the VRVis Research Center is used to create this demo. It is configured to use Vulkan underneath. Assimp (<http://www.assimp.org>) is used for loading models from files.