WATERGLASS

a demo by Peter Houska, 9907459, e881

What's it all about?:
This demo was implemented for the "Realtime Graphics" course at the Technical University of Vienna. Everything was coded in about three weeks time [not fulltime, though ;-)]. Conversely to the name of my demo, no waterglasses are rendered in the demo - quite confusing, isn't it? ;-) Sometimes names are just names...

System requirements:
The demo was developed and tested on a
P4@2.4 GHz, 512 MB RAM, GeForce FX 5950 Ultra, 256MB VRAM,
running WinXP

Development-environment and libraries:
C/C++ (with MSVC++6 compiler), GLSL shaders, OpenGL+GLU+GLEW, SDL+SDL_image and fmod

Tech Info:
First of all, there are not many effects in the traditional sense in this demo, even though many (long/more complex) shaders are driving the demo. The shaderprograms are mainly there to "make things work", not to "make things look better"...

The watermesh is rendered with a smooth level-of-detail technique which is a mixture of Persistent Grid Mapping**[1]** and the Projected Grid Concept**[2]**. The basic idea is as follows:
Draw a viewplane-aligned quad and use the texture-coordinates, together with the camera position and -orientation to perform a ray-plane intersection. The intersection point is interpreted as a texture coordinate to access a heightmap. The looked up heightvalue is then used to displace the intersection-point along the y-coordinate. One major advantage is that view frustum culling comes at no cost and is near optimal!

In **[1]** this technique is used somehow differently because the authors rely on the GPUs ability to perform a vertex texture lookup (VTF in the following). In **[2]** some sort of CPU intervention is needed to perform this update. Since I don't have a GPU that can do a VTF but didn't want to resort to CPU-intervention I implemented the OpenGL-way of "Render to Vertex Buffer", which is, in it's pure form, a DX9 (and later?) feature, available on ATI cards. The coordinates of the above mentioned intersection point are determined in the fragment shader (where a texture access is no problem) and written to a float - texture FBO. This data is read into a PBO which is used as a VBO subsequently. The driver should be able to perform these operations on the server-side (data should remain on the GPU, little or no CPU intervention necessary). A very good tutorial for R2VB with OpenGL is available from **[3]**.

The watermesh-animation comes from a 3D textures that wraps around seamlessly in x, y and z. The xy-slices are the actual water-heightmap images and the z-coordinate is incremented based on the current time to select a slice from the 3D texture in each frame. The geometry is rendered with a single lookup into the 3D texture, for the final lighting calculations an additional bump-map (stored in the same 3D texture), generated in a preprocessing step from the height values, comes into play and is accessed at several scales to improve shading of the watersurface.

Screenspace reflection- and refraction maps (512x512 textures) are blended according to a Fresnel term to model local reflections and refractions. Before accessing the texture, some perturbation, proportional to the approximated surface normal at each watersurface-point is applied along the xy projected texture coordinate. While this may not be 100% physically correct, it is a convincing fake ;-) Where no "geometry" is hit in the reflection map, a simple phong lighting model is evaluated.

The reflection and refraction maps are not rendered with clip-planes to separate geometry above and below the waterlevel. A shaderprogram clips the geometry at the fragment-level based on the waterheight at this point, instead. This way, "seam-artifacts" are considerably reduced **[4]**.

In order to simulate "deep water", the colors in the refraction-map are slightly modified to look more "underwater". Where the wave above the ground is high (above some threshold), caustics are simulated by rendering white color. These effects are again not a physically correct simulation/model, more a good-looking fake. I spent some time on tweaking all the parameters (blending intermediate results, evaluating light models, ...) to make the water look convincingly natural - I hope I succeeded ;-)

In the scene, there is a sphere that mimics a buoy. This buoy casts a shadow on the terrain which is generated using "traditional shadow mapping" **[5]**, **[6]**. A special depth-attachment FBO (1024x1024 texture) is used to store the depth values from the light's point of view. The well-known shadowmapping artifacts are somewhat alleviated in this case, due to the fact that the shadowed geometry is rendered after the above mentioned water-perturbation. Another reason for this is, that the "shadow-camera"-parameters (position, near, far plane, etc.) are tweaked for the small area around the buoy. Things would be much worse if the whole island was to be covered by the shadowmap.

Cardinal splines are calculated for the camera animation along a path. The "tension" parameter was set to 0.0 (this class of curves is referred to Catmull-Rom splines) **[7]**. Since the path is evaluated according to "real-time", the demo should take the same time to finish, independent of the framerate.

Performance:
The demo is quite a performance-killer... the main reason for this is, that due to my hardware-constraints, I couldn't use handy features such as MRTs. This forces me to do multipass-rendering even though a single pass would suffice in many situations (e.g. when rendering the screen-space reflection and refraction mask). Another reason is that this project is my first project that makes excessive use of Vertex- and Fragmentshaders, so I most probably lack experience. :-(
On my machine, the framerate never drops below 13fps when rendering to a 800x600 window which is the default window-size of the demo. On the other hand, the water-surface covers the whole area, starting at the camera location, towards the horizon - that's not bad!

Keys:

ESC	quit demo
l	toggle wireframe mode
i	show reflection-, refraction- and shadow map
n	toggle watermesh display

References:

[1] Yotam Livny, Neta Sokolovsky, Tal Grinshpoun, Jihad El-Sana **"Persistent Grid Mapping: A GPU-based Framework for Interactive Terrain Rendering"**, Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

[2] Claes Johanson, **"Real-time water rendering, Introducing the projected grid concept"**, Master of Science thesis March 2004, Lund University

[3] http://developer.apple.com/samplecode/PBORenderToVertexArray/index.html

[4] Yung-Feng Chi, **"True-to-life Real-Time Animation of Shallow Water on Today's GPUs"**, Shader X4, Advanced Rendering Techniques

[5] Randi Rost, **"OpenGL Shading Language, 2nd Edition"**, better know as the "Orange book", p.341

[6]
http://www.cs.cmu.edu/afs/cs/academic/class/15462/web.06s/asst/project3/shadowmap/index.html

[7] Hearn, Baker, **"Computer Graphics with OpenGL, 3rd Edition"**, p.429