UNIVERSITÄT DES SAARLANDES
Prof Dr. Philipp Slusallek
Lehrstuhl für ComputerGraphik
Sven Woop (woop@graphics.cs.uni-sb.de)
Stefan Popov (popov@graphics.cs.uni-sb.de)

22th January 2007

Computer Graphics I
Assignment 10

Submission deadline for the exercises: Thursday, 1st February 2007

10.1 Clipping (40 Points)

Given an axis aligned clipping window spanned by the two points $(0, 0)$ and $(12, 6)$

$$C_1 = (0, 0), C_2 = (0, 6), C_3 = (12, 6), C_4 = (12, 0)$$

and a polygon with the lines $P_1 - P_2 - P_3 - P_4 - P_1$ with

$$P_1 = (9, 8), P_2 = (-6, 3), P_3 = (14, 20), P_4 = (15, -4).$$

Use the Sutherland-Hodgeman algorithm to perform the first step of the algorithm by clipping the polygon against the top edge of the clipping window only. Describe each step of the algorithm. You do not have to compute the intersection point of the lines with the clipping window, give them names in a figure and use that names instead.

10.2 Rasterization (10 Points)

Why does mipmapping speed up texturing in rasterization hardware? Look at the memory access patterns and especially the cache.

10.3 Triangle Order for Rasterization (5 + 5 Points)

Assume you are rasterizing some triangles in different orders.

a) Is the depth buffer the same for each order of the triangles after the computation?

b) Is the color buffer the same for each order of the triangles after the computation (with and without Z-Buffer)?

10.4 Vertex and Pixel shading (10 + 10 + 20 + 20 Bonuspoints for d)

In this exercise you will write an OpenGL application using CG vertex- and pixel shaders. To start, download the new framework from `http://graphics.cs.uni-sb.de/Courses/ws0607/cg/skeleton.tgz`. Unzip it and read through the code to understand what is roughly happening. The result of this exercise should be a waving water surface (animated with a vertex program) with a Fresnel reflection of an environment map.

a) In this exercise we will implement a simple water wave simulation using vertex-programs. Vertex-programs transform the vertices of the primitives prior to rasterization. All you have yo do here is to change the world z-cooridinate of the incoming vertex position. To do so edit `water_vert.fx` and add the appropriate parameter bindings to `water_sim.cpp`. A simple way to simulate water behaviour is to combine *sin* funtions.

b) Once your waves are moving, you have to calculate the proper normals. To calculate tangents to a surface you can use the numerical derivative approximation given by $f'(x) \approx \frac{f(x+d)-f(x)}{|d|}, x$ and $d \in R^2$. This formula gives you the derivative of f in direction d. The length of d determines the approximation error, so smaller lengths give better approximations, but be aware of numerical errors.
Hint: You can visualize the calculated normals by setting the fragment color to the normal value in the fragment program.

c) Now, you have to implement the water-surface reflection with the envirnoment map. To do so, edit the fragement program `water_frag.fx` and add the proper bindings to `water_sim.cpp`. Calculate the reflection vector from the eyepoint to the shaded point. Use the reflected vector to look up the environment map.

In order to calculate the world eyepoint coordinates you need to multiply the point $(0, 0, 0, 1)$ with the inverse model-view transformation.*(see OpenGL documentation for glGet())*. You can pass the calculated point to the fragment shader as a uniform parameter. You also have to bind the cube-environment map texture (*look in the method drawSky()*).
Hint: Disable the wave generation (both height and normal) and get the reflections working for the flat surface first. The normal should be $(0, 1, 0)$ *here.*

d) (Bonus exercise (20 Points)) Once everything is working you can apply the Fresnel term to simulate more realistic water surface reflections.

To make things work you will need a graphics-board that supports CG-shaders like everything above the nVidia 5200 FX. The CIP pool (105) has been tested with our solution and everything works fine.

Solutions

10.1 Clipping

Sutherland-Hodgeman's algorithm is a polygon clipping algorithm. It works by clipping the polygon in four steps against each edge of the clipping window thus after the last step the remaining polygon is the fully clipped one. This exercise only considers the first clipping step, of clipping against the top edge. The initial polygon is given by a implicitly closed sequence of points:

$$P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow$$

We compute a new list of implicitly closed vertices by applying the algorithm:

a) i=1: P_1 is outside the clipping region defined by the top vertex. Thus we store no new item to the new vertex list.

b) i=2: P_2 is inside the clipping region defined by the top vertex and the last point P_1 was outside, thus we store the vertices Q_1 and P_2 to the new vertex list.

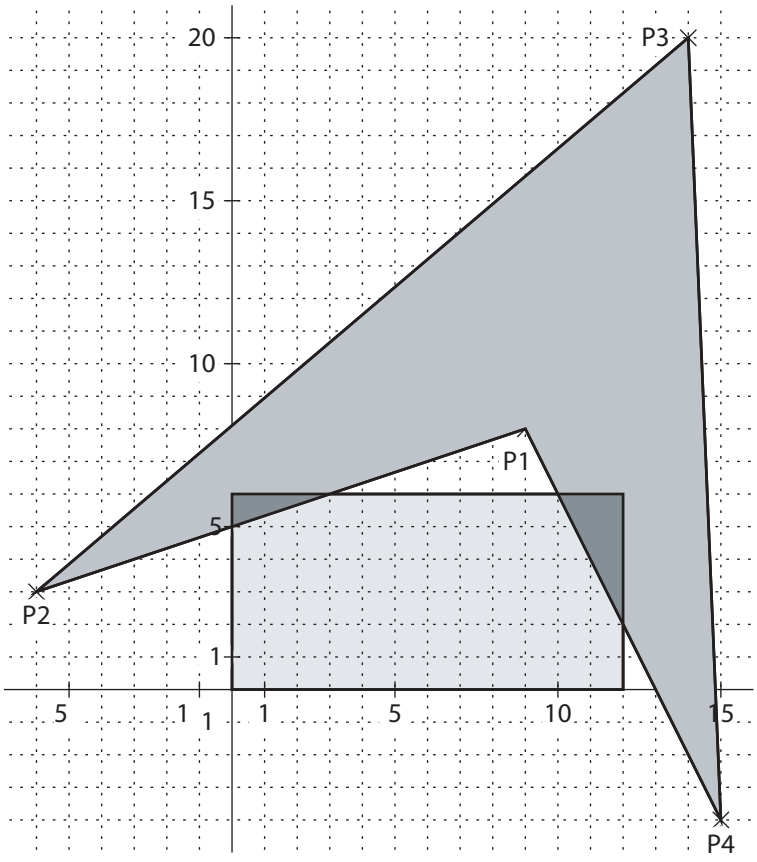
c) i=3: P_3 is outside the clipping region defined by the top vertex and the last point P_2 has been inside, thus we store only Q_2 to the new vertex list.

d) i=4: P_4 is inside the clipping region defined by the top vertex and the last point P_3 was outside, thus we store Q_3 and P_4 to the new vertex list.

e) i=5: $P_5 = P_1$ is outside the clipping region defined by the top vertex and the last point P_4 was inside, thus we store Q_4 to the new vertex list. We do not store P_1 as we are in the last special case of testing the closure line of the curve for intersection with the clipping region.

The implicitly closed polygon clipped to the top vertex of the clipping region looks like:

$$Q_1 \rightarrow P_2 \rightarrow Q_2 \rightarrow Q_3 \rightarrow P_4 \rightarrow Q_4 \rightarrow$$



10.2 Rasterization

Mipmapping selects a texture level whose texels fit in size approximately the pixels on the screen. If mipmapping is disabled, a large texture being far away from the camera causes random samples from this texture to be accessed. Despite the number of memory accesses is the same in both cases, in the first one they are much more locally, making the texture cache more efficient as it works on blocks of the texture. Thus less memory requests need to be performed.

10.3 Triangle Order for Rasterization

a) The depth buffer is exactly the same after the computation, as the algorithm computes the minima of all depth values.

b) The color buffer is not the same in general. Consider two triangles lying at the same position but having different color.