


Worksheet 9: Reflections

Reading	<p>Angel: 7.1, 7.10-7.10.3</p> <p>McReynolds: 17.1-17.1.2 (“Planar Reflectors”)</p> <p>Kilgard: 0-3 (“Improving Reflections via the Stencil Buffer”)</p> <p>Lengyel: 5.6 (“Reflections and Oblique Clipping”)</p>
Purpose 	<p>The purpose of this set of exercises is to understand and implement planar reflections using WebGL. Like shadows, reflections are almost trivial to compute in ray tracing and more difficult when we use rasterization (WebGL). However, if we simplify the situation and consider only planar reflectors, we can simply draw the reflection. Imagine that we render an object in front of a reflecting planar surface. To draw the reflection, we mirror all the vertices of the object in the plane, and render the mirrored object.</p>
Part 1 Reflected object	<p>Start from the shadow mapping result of Worksheet 8. Set the camera to a 65 degrees field of view and place it in $(0, 0, 1)$ looking at $(0, 0, -3)$. Comment out the rendering of the ground plane. Draw the teapot twice, but use a reflection transformation matrix R with one of them. Construct the matrix R such that the reflected teapot is mirrored in the ground quad. Ensure that the lighting of the teapot is also reflected. [McReynolds 17.1.2]</p>
Part 2 Reflector	<p>Let the teapot move from $y = -1$ to $y = 0$ when jumping. Reinsert the ground and observe that the teapot appears below the ground quad when jumping high enough. Blend the ground quad with the mirrored teapot by setting the ground quad to be partly transparent. [Worksheet 7, Angel 7.10-7.10.3]</p>
Part 3 Stencil buffer for clipping	<p>The appearance of the teapot below the ground quad ruins the illusion that the teapot is reflected in the ground. We only want to draw the reflected object in the part of the canvas covered by the ground quad. Unfortunately, we cannot use the depth buffer for clipping (as with shadow polygons), because the depth buffer is needed for proper rendering of the reflected objects. We therefore use the stencil buffer instead. [Angel 7.1] [McReynolds 17.1.2] [Kilgard 0-3]</p> <p>To have a stencil buffer in WebGL, you need to ask for it:</p> <pre>var gl = WebGLUtils.setupWebGL(canvas, { alpha: false, stencil: true });</pre> <p>Once the stencil buffer is available, follow the first approach described in AGP to clip the reflected objects (not the technique using stencil-based clearing, as this seems not to work in WebGL).</p> <p>The references use OpenGL. Mozilla’s documentation of the WebGL rendering context interface is useful for translating the operations into WebGL: https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext</p>
AGP = Advanced Graphics Programming [McReynolds]	

Worksheet 9: Reflections

<p>Part 4 Clipping submerged objects using the view frustum</p>	<p>The reflection has one problem: The reflection plane always reflects the whole teapot – even parts that are behind the reflector surface. Let the teapot move from $y = -1.8$ to $y = 0$ when jumping to illustrate this problem.</p> <p>Fix the problem by aligning the near plane of the view frustum with the reflection plane when drawing reflected objects. This technique is referred to as oblique near-plane clipping, and it is described in the text on “Reflections and Oblique Clipping” by Eric Lengyel (available on File Sharing).</p> <p>This method relies on the following function that modifies the near clip plane (clipplane) of a view frustum, which is defined by a perspective projection matrix (projection):</p> <pre>function modifyProjectionMatrix(clipplane, projection) { // MV.js has no copy constructor for matrices var oblique = mult(mat4(), projection); var q = vec4((Math.sign(clipplane[0]) + projection[0][2])/projection[0][0], (Math.sign(clipplane[1]) + projection[1][2])/projection[1][1], -1.0, (1.0 + projection[2][2])/projection[2][3]); var s = 2.0/dot(clipplane, q); oblique[2] = vec4(clipplane[0]*s, clipplane[1]*s, clipplane[2]*s + 1.0, clipplane[3]*s); return oblique; }</pre> <p>The first argument is a vec4 with clip-plane equation coefficients (a, b, c, d), such that the clip plane is defined by</p> $ax + by + cz + d = 0.$ <p>The coefficients must be determined so that the plane is in eye space. The second argument is a 4-by-4 perspective projection matrix. Find the plane equation coefficients of the reflector in eye space, and use these with the given function to obtain a projection matrix with oblique near-plane clipping. Use this modified projection matrix when rendering reflected objects. [Lengyel 5.6]</p> <p>With oblique clipping, depth buffer values are inconsistent with the values produced by the original projection matrix. The depth buffer must therefore be cleared after the reflected objects have been drawn.</p>
---	---