Technical University of Denmark (DTU)

02561 Computer Graphics

Project: Planar Reflector

1 December 2024

Gao, Tianrun, s241728

Table of contents

Link: https://trgao.github.io/02561-com-graphics/

Introduction

Reflections play a significant role in generating realistic computer-generated imagery. In ray tracing, where the behaviours of light rays are simulated as they travel through a scene, reflections are trivially handled as they are already factored into the calculations for light simulation. However, WebGL is a rasterization based rendering engine, which makes it more challenging to render reflections. To simplify the situation, this project aims to understand and implement planar reflections, using a 3D scene consisting of a Newell teapot and a textured ground quad. A combination of techniques will be used to render the scene, namely reflection matrices, blending, stencil buffer clipping, and oblique near plane clipping.

Method

The implementation of a planar reflector is broken into 4 key steps using different techniques:

1. Reflection transformation matrix: A second teapot is rendered and it mirrors the original teapot across the reflective plane, which is the ground quad. To achieve this, the model view matrix of the reflected teapot is multiplied by a reflection transformation matrix, which projects the vertices of the original teapot mirrored across the ground quad plane.
2. Blending: The ground quad is blended with the reflection of the teapot so that it is rendered as a translucent object and we can see the reflection through the ground quad, giving the illusion that the mirrored teapot is a reflection within the ground quad.
3. Stencil buffer clipping: The reflection of the teapot is clipped such that it will not be rendered if it moves below the ground quad, so that it does not appear outside of the ground quad. The stencil buffer is used to keep track of the vertices of the ground quad to create a mask on the reflected teapot and decide the boundary to render the teapot within.
4. Oblique near plane clipping: The reflection of the teapot is clipped such that it will not be rendered if the teapot moves below the ground quad, which causes the reflection to move above the ground quad. This is achieved through modifying the near plane of the view frustum in the projection of the reflected teapot

Rasterization transforms 3D geometry into a 2D image by projecting objects onto the screen using a camera perspective. It does not simulate the behaviours of the light rays at all, which means that to render reflections, it will have to be manually calculated and rendered into the scene. Thus, this combination of techniques creates the illusion of reflection without the computationally intensive calculations for simulating light ray behaviours.

Implementation

The project starts off from the final rendered scene in Worksheet 9, as can be seen in Figure 1. The teapot has shadows projected onto itself and the ground quad below it. Reflections will be added to the ground quad to make the scene even more realistic.
The camera is now changed to a 65 degrees field of view at position (0, 0, 1), looking at position (0, 0, -3). This can be seen in Figure 2, and will be the initial starting point of the project.

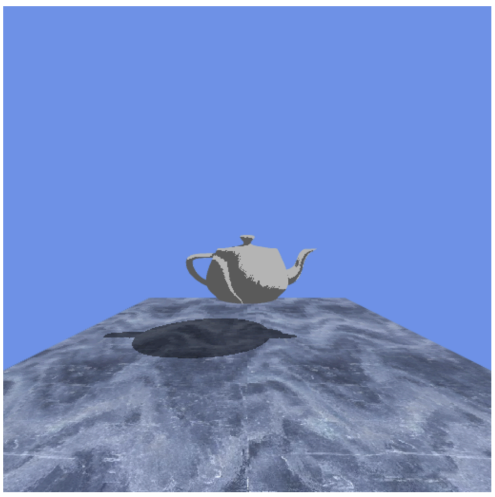

Figure 1. Worksheet 9 final results

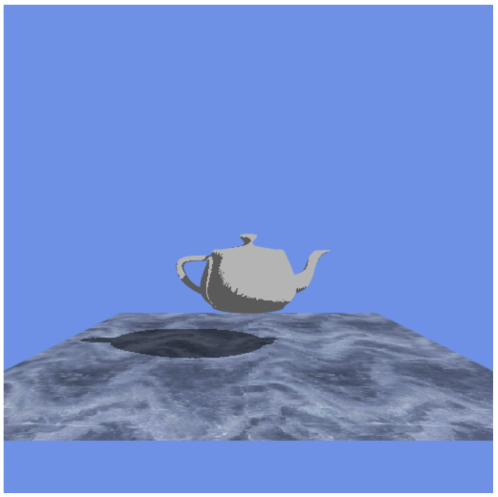

Figure 2: Initial starting point

Reflection Transformation Matrix

To create a reflection, the original teapot is rendered a second time, but mirrored across the ground plane. This is implemented by multiplying the model view matrix of the original teapot with a reflection transformation matrix, which is given by this formula:

$$R = \begin{pmatrix} 1-2V_x^2 & -2V_xV_y & -2V_xV_z & 2(P\cdot V)V_x \\ -2V_xV_y & 1-2V_y^2 & -2V_yV_z & 2(P\cdot V)V_y \\ -2V_xV_z & -2V_yV_z & 1-2V_z^2 & 2(P\cdot V)V_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Here, vector V refers to the normal vector to the plane representing the ground quad, and vector P refers to a point on the plane. This reflection transformation matrix and the model view matrix of the reflected teapot is implemented as shown below:

```
var p = vec3(-2.0, -1.0, -1.0);
var v = vec3(0.0, 1.0, 0.0);
var reflection = mat4(
    1 - 2 * v[0] * v[0], -2 * v[0] * v[1], -2 * v[0] * v[2], 2 * dot(p, v) * v[0],
    -2 * v[0] * v[1], 1 - 2 * v[1] * v[1], -2 * v[1] * v[2], 2 * dot(p, v) * v[1],
    -2 * v[0] * v[2], -2 * v[1] * v[2], 1 - 2 * v[2] * v[2], 2 * dot(p, v) * v[2],
    0, 0, 0, 1
);
var reflectionModelView = mult(reflection, teapotModelView);
```

As can be seen in Figure 3, the teapot and the reflected teapot are both rendered, and movement cannot be shown through images but the movement of the teapot and the reflected teapot are also mirrored. The ground quad is hidden to show both objects clearly.

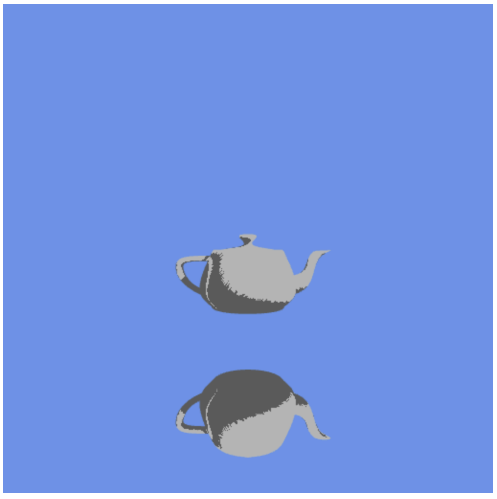

Figure 3: Teapot and its reflection

Blending

The ground quad is now rendered together with the 2 teapot objects. However, from Figure 4, we can see that the reflected teapot does not look like a reflection at all because the ground quad is opaque and blocking the view of the reflected teapot. To solve this problem, we use the blending technique to make the ground quad translucent so that the reflected teapot appears within the ground quad to achieve the illusion of a reflection. The colors of the reflected teapot and the ground quad are blended and rendered as the color of the ground quad using this formula:

$$color_{result} = (color_{source} \times factor_{source}) + (color_{destination} \times factor_{destination})$$

In this case, source color refers to the original color of the ground quad, and destination color refers to the color currently stored in the color buffer, which is the color of the reflected teapot. The colors are blended using a source factor of the alpha component of the source color, and a destination factor of 1 - alpha component of the source color. This is implemented as shown below:

```
gl.enable(gl.BLEND);
gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA);
```

As can be seen in Figure 5, the ground quad is now translucent, allowing us to see the reflected teapot beneath it. It is starting to resemble a proper reflection.

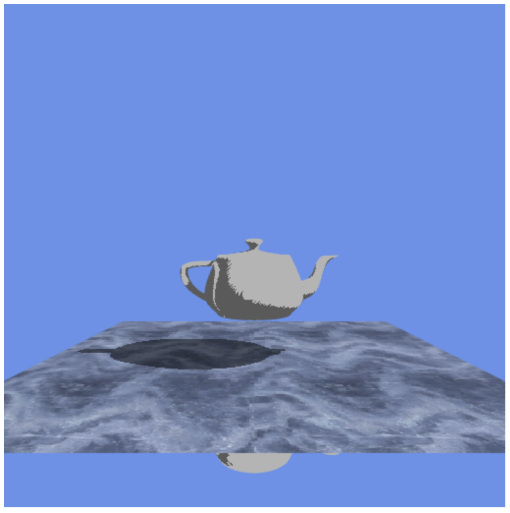

Figure 4: Ground quad with 2 teapot objects

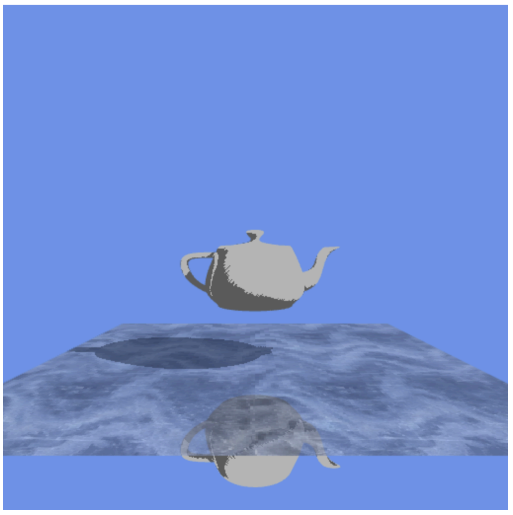

Figure 5: Blending ground with reflected teapot

Stencil buffer clipping

From Figure 5, it can be seen that when the teapot moves upwards, the reflected teapot will move downwards and possibly move below the ground quad, which ruins the illusion of reflection. To solve this, stencil buffer clipping is used to clip away the parts of the reflected teapot that moves below the ground quad to keep the reflected teapot bounded within the ground quad. The ground quad mask is obtained by drawing the ground quad into the stencil buffer, setting bits at positions of the ground quad vertices to 1, while setting every other bit to 0. The color and depth buffers are temporarily disabled to ensure that we do not actually render this ground quad yet which could cause issues with blending from the previous part. This is implemented as shown below:

```
gl.enable(gl.STENCIL_TEST);
gl.clear(gl.STENCIL_BUFFER_BIT);
gl.stencilFunc(gl.ALWAYS, 1, 0xFF);
gl.stencilOp(gl.KEEP, gl.KEEP, gl.REPLACE);
gl.colorMask(false, false, false, false);
gl.depthMask(false);
```

The reflected teapot can now be rendered. Stencil operations are used to draw the vertices of the reflected teapot only when the positions of the teapot pass the stencil testing. In this case, the vertices of the reflected teapot are only rendered when they coincide with the positions of the ground quad vertices. This is implemented as shown below:

```
gl.stencilFunc(gl.EQUAL, 1, 0xFF);
gl.stencilOp(gl.KEEP, gl.KEEP, gl.KEEP);
```

Thus, this ensures that the reflected teapot is not rendered when it goes beyond the boundary of the ground quad, as shown in Figure 6.

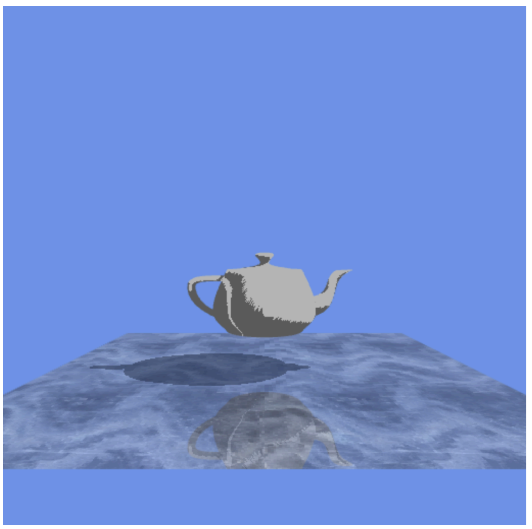

Figure 6: Reflected teapot does not stick out under the ground quad

Oblique near plane clipping

Stencil buffer clipping only ensures that the reflected teapot will not be rendered if it moves below the ground quad. However, if the teapot moves downwards until it is submerged below the ground quad, the reflected teapot actually appears above the ground quad as shown in Figure 7, which ruins the illusion of reflection. To prevent this, oblique near plane clipping is used to clip the reflected teapot such that it is not rendered when it moves above the ground plane, by changing the projection matrix of the reflected teapot. The projection matrix for the reflected teapot is changed to achieve this, which is given by this formula:

$$\mathbf{M}'_3 = \frac{-2Q_z}{\mathbf{C}\cdot\mathbf{Q}}\mathbf{C} + \langle 0,0,1,0\rangle.$$

M refers to the new projection matrix, C refers to the equation of the clip plane $ax + by + cz + d = 0$ in the form of a 4 dimension vector (a, b, c, d) projected into camera space, and Q refers to the furthest point that lies opposite the near plane of the view frustum and lies on the far plane of the view frustum. This is implemented below:

```
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;
}
var groundPlane = mult(
      transpose(inverse(mult(reflection, cameraModelView))),
      vec4(0.0, 1.0, 0.0, 1.0)
);
var reflectionProjection = modifyProjectionMatrix(groundPlane, cameraProjection);
```

The ground plane has an equation of $y = -1$, and is represented by the 4 dimensional vector (0, 1, 0, 1). The ground plane is transformed into camera space coordinates by multiplying it with the inverse transpose of the model view matrix of the reflected teapot. The projection matrix of the reflected teapot can then be modified using the modifyProjectionMatrix function that implements the formula mentioned above. As can be seen in Figure 8, this now produces the desired effect of clipping the reflected teapot once it moves above the ground quad.

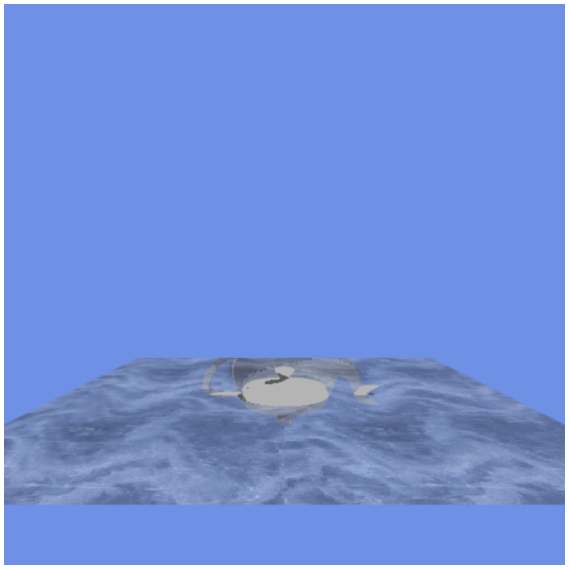

Figure 7: Reflected teapot appears above ground

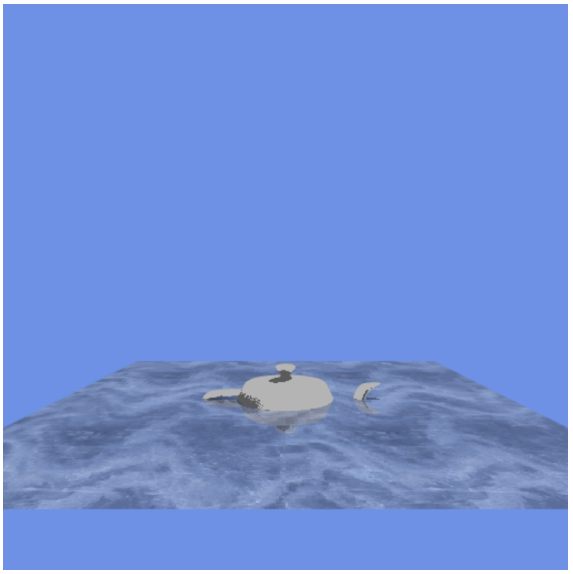

Figure 8: Reflected teapot is clipped

Results

Link to implemented project: https://trgao.github.io/02561-com-graphics/project/project.html

With the 4 techniques implemented, the illusion of reflection is now complete. The reflected teapot does not move above or below the ground quad, maintaining the illusion that it is just a reflection on the surface of the ground quad, when in reality it is just another rendered object blended with the ground quad.

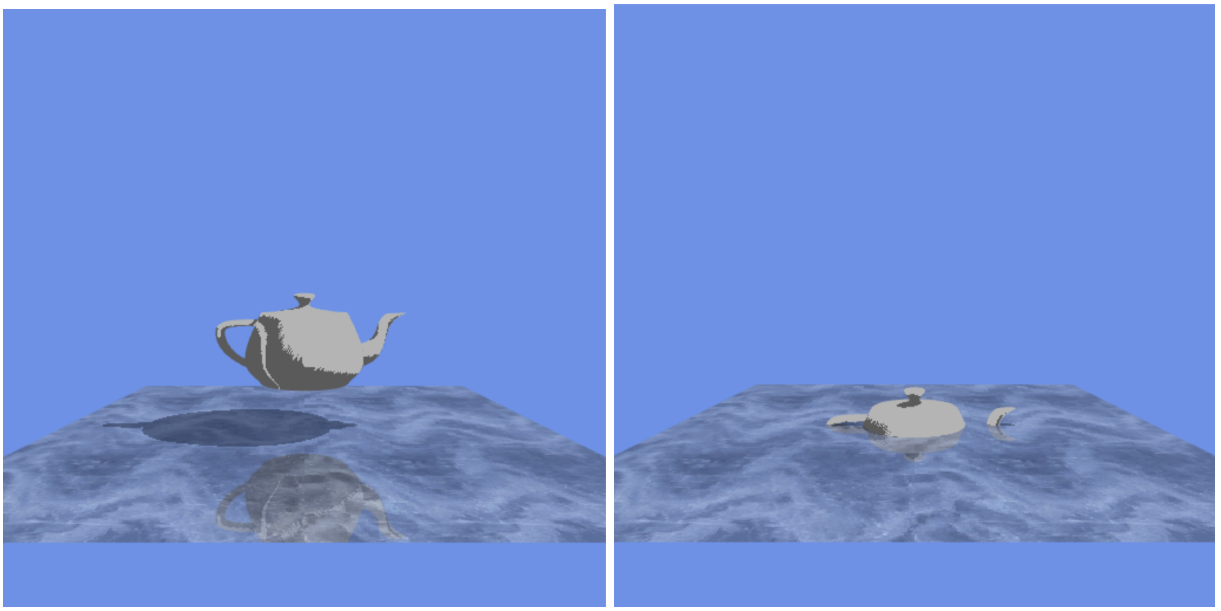

Figure 9 and 10: Results from the complete implementation

Discussion

Through this project, I have learnt about rendering a planar reflector in WebGL through the use of different techniques. I have learnt that rendering a reflection in WebGL does not actually involve simulation of light rays to render the reflection, but rather through the use of techniques that merely produce an illusion of reflection. However, these techniques can only be used to render reflections in a flat surface, and does not cover the more complicated scenarios of reflections in complex objects and curves.