ECS762P

Computer Graphics: Lab 2

Sean Sanii Nejad

Contents

Exercise A

A1

Output

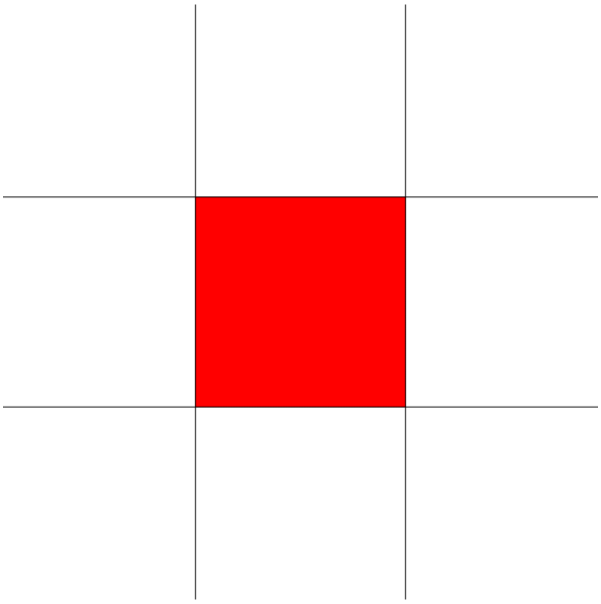

Figure 1: Output showing scaling and rotation transformation

JavaScript Code Snippet

```javascript

// A1 -- DEFINE THESE TWO 4x4 MATRICES PROPERLY
let pre_scale = [[0.5, 0, 0, 0],
                 [0, 0.5, 0, 0],
                 [0,   0, 1, 0],
                 [0,   0, 0, 1]];

let pre_rotate = [[Math.cos(theta), -Math.sin(theta), 0, 0],
                  [Math.sin(theta), Math.cos(theta), 0, 0],
                  [0,   0, 1, 0],
                  [0,   0, 0, 1]];

```

Figure 2: JavaScript Code Snippet from 'transforming-square.js' file

Vertex Shader Code Snippet

```glsl
// homogeneous cordinates [x,y,z,w]
vec4 point =  vec4(v0: vertex.x, v1: vertex.y, v2: 0.0, v3: 1.0);

// A3 -- DEFINE translate_inv HERE

// A1 -- ADD CODE HERE
point =  pre_scale * pre_rotate * point;

// A1, A2, A3, A4, A5 -- MODIFY HERE
gl_Position =  point;
```

Figure 3: GLSL Code Snippet from 'transforming-vert.glsl' file

Identity Matrix 4x4

$$
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Figure 4: Identity Matrix of size 4

Scale Matrix

$$
\boldsymbol{Sp} = \begin{bmatrix} s_0\,x \\ s_1\,y \\ s_2\,z \\ 1 \end{bmatrix} \qquad \boldsymbol{S} = \begin{bmatrix} s_0 & 0 & 0 & 0 \\ 0 & s_1 & 0 & 0 \\ 0 & 0 & s_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Figure 5: Scaling matrix that acts on homogenous coordinates

Rotation Matrix

$$
\boldsymbol{R}_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Figure 6: Rotation matrix in 2D xy plane

Explanation

For this exercise, we applied Euclidean and affine transformations. The first transformation was scaling. We scaled the size of the square by half. This is shown with the matrix pre_scale (figure 2) and outlined in figure 5. We modified the identity matrix (figure 4) which does not have any changes to the square. The first 0.5 represents the x coordinate and the second 0.5 represents the y coordinate. Z and w are both set to 1 and therefore, are unchanged

We then applied another affine transformation, rotation. This was performed by changing the identity matrix in pre-rotate to the rotation matrix (figure 6) as shown in figure 2.

References

https://www.brainvoyager.com/bv/doc/UsersGuide/CoordsAndTransforms/SpatialTransformationMatrices.html

A2

Output

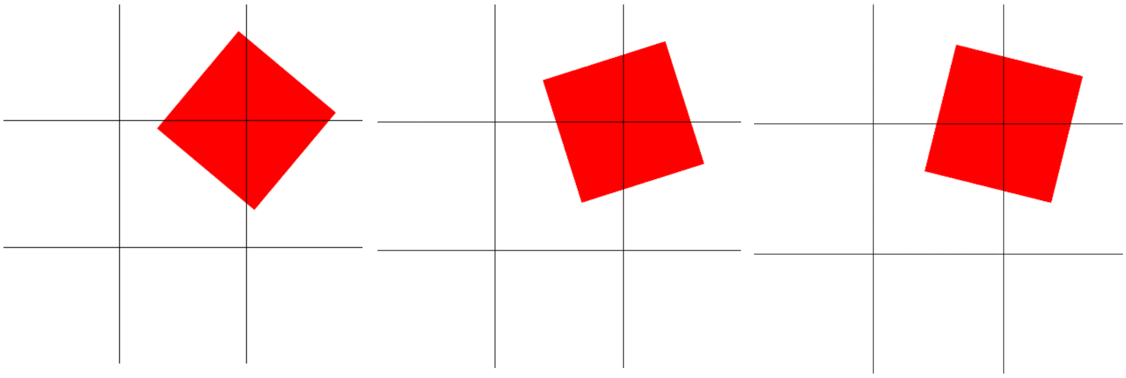

Figure 7: Output showing translate transformation

JavaScript Code Snippet

```javascript

// A2-5 ADD NEW DECLARATIONS
var translate_loc;
```

```javascript
    // get uniform locations
    pre_rotate_loc = gl.getUniformLocation(program, 'pre_rotate');
    pre_scale_loc = gl.getUniformLocation(program, 'pre_scale');
    rotate_loc = gl.getUniformLocation(program, 'rotate');
    rgb_loc = gl.getUniformLocation(program, 'rgb');

    // A2-5 GET NECESSARY UNIFORM LOCATIONS
    translate_loc = gl.getUniformLocation(program, 'translate');
```

```javascript
    let rotate = [[Math.cos(theta), -Math.sin(theta), 0, 0],
                  [Math.sin(theta),  Math.cos(theta), 0, 0],
                  [0, 0, 1, 0],
                  [0, 0, 0, 1]];
```

```javascript
    // A2-5 DEFINE NEW MATRICES
    let translate = [[1, 0, 0, side/2],
                     [0, 1, 0, side/2],
                     [0, 0, 1,    0   ],
                     [0, 0, 0,    1   ]];

    // set all transformations
    gl.uniformMatrix4fv(pre_rotate_loc, false, mat_float_flat_transpose(pre_rotate));
    gl.uniformMatrix4fv(pre_scale_loc, false, mat_float_flat_transpose(pre_scale));
    gl.uniformMatrix4fv(rotate_loc, false, mat_float_flat_transpose(rotate));
    gl.uniformMatrix4fv(translate_loc, false, mat_float_flat_transpose(translate));
```

Figure 8: JavaScript Snippet from 'transforming-square.js file

Vertex Shader Code Snippet

```
// A2 -- DECLARE UNIFORM TRANSLATION MATRIX HERE
uniform mat4 translate;
```

```
    // A1 -- ADD CODE HERE
    point =  pre_rotate * pre_scale * point;

    // A1, A2, A3, A4, A5 -- MODIFY HERE
    gl_Position =   translate * rotate * point;
```

Figure 9: Vertex Shader Snippet from 'transforming-vert.glsl file

Translation Matrix

$$\boldsymbol{Tp} = \begin{bmatrix} x + t_0 \\ y + t_1 \\ z + t_2 \\ 1 \end{bmatrix} \qquad \boldsymbol{T} = \begin{bmatrix} 1 & 0 & 0 & t_0 \\ 0 & 1 & 0 & t_1 \\ 0 & 0 & 1 & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 10: Translation matrix that acts on homogenous coordinates

Explanation

For this exercise, we transformed the square by a fixed amount to another location on the screen. In order to do this, we first created a uniform 4x4 matrix called translate in the vertex shader as shown in figure 9. We then created a global variable called translate_loc in JavaScript. We used one of WebGL's API methods to get the uniform location of the mat4 translate object. This is later used as the first parameter for uniformMatrix4fv method which is also found in WebGL's API.

Finally, we created a 4x4 translation matrix as shown in figures 8 and 10. This moved the square to the upper right corner of the screen.

References

https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/getUniformLocation
https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/uniformMatrix

A3

Output

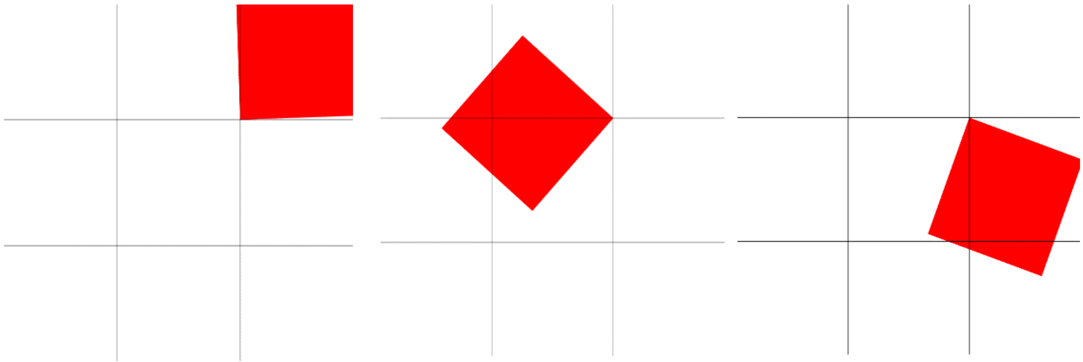

Figure 11: Output showing rotation around different point

Vertex Shader Code Snippet

```
// A3 -- DEFINE translate_inv HERE
mat4 translate_inv = translate;

translate_inv[3][0] = -translate_inv[3][0];
translate_inv[3][1] = -translate_inv[3][1];

// A1 -- ADD CODE HERE
point =  pre_rotate * pre_scale * point;

// A1, A2, A3, A4, A5 -- MODIFY HERE
gl_Position =  translate * rotate * translate_inv * point;
```

Figure 12: GLSL Snippet code from 'transforming-vert.glsl' file

Rotation around arbitrary point

- Let M be the matrix representation of this:

$$p' = Mp$$

- Let T be the translation by $[t_0,\ t_1,\ 0,\ 0]^\top$ hence:

$$M = TR_zT^{-1}$$

Explanation

For this exercise, we rotate the square from a different point. This is done by creating an additional translate 4x4 matrix with negative values of the original transform matrix. From here, we include another rotation matrix between the original translate and negative translate matrixes. This causes the desired rotation effect you see.

References

https://www.brainvoyager.com/bv/doc/UsersGuide/CoordsAndTransforms/SpatialTransformationMatrices.html

A4

Output

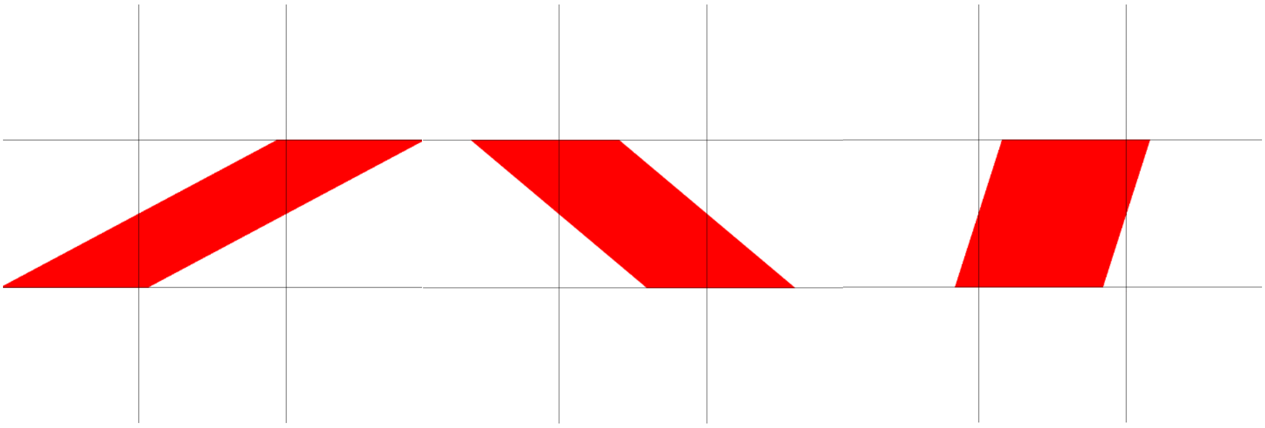

Figure 13: Output showing shear transformation

JavaScript Code Snippet

```javascript

// A2-5 ADD NEW DECLARATIONS
var translate_loc, shear_loc;

```

```javascript
// A2-5 GET NECESSARY UNIFORM LOCATIONS
translate_loc = gl.getUniformLocation(program, 'translate');
shear_loc = gl.getUniformLocation(program, 'shear');
```

```javascript
let shear = [[1, Math.tan(theta), 0, 0],
             [0, 1, 0, 0],
             [0, 0, 1, 0],
             [0, 0, 0, 1]];
```

```javascript
// A2-5 SET NECESSARY TRANSFORMATION UNIFORMS
gl.uniformMatrix4fv(translate_loc, false, mat_float_flat_transpose(translate));
gl.uniformMatrix4fv(shear_loc, false, mat_float_flat_transpose(shear));
```

```javascript
// A2 DISABLE YOUR TRANSFORMATIONS
gl.uniformMatrix4fv(translate_loc, false, mat_float_flat_transpose(identity));
gl.uniformMatrix4fv(pre_rotate_loc, false, mat_float_flat_transpose(identity));
gl.uniformMatrix4fv(pre_scale_loc, false, mat_float_flat_transpose(identity));
gl.uniformMatrix4fv(rotate_loc, false, mat_float_flat_transpose(identity));
gl.uniformMatrix4fv(shear_loc, false, mat_float_flat_transpose(identity));
```

Figure 14: JavaScript Code Snippet from 'transforming-square.js' file

Shear Matrix

$$Dp = \begin{bmatrix} x + d_{01}\, y \\ y \\ z \\ 1 \end{bmatrix} \qquad D = \begin{bmatrix} 1 & d_{01} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 15: Shear matrix that acts on homogeneous coordinates

Explanation

For this exercise, we sheared the square horizontally. In order to do this, we created another shear_loc, got the location of the uniform, set the uniform with uniformMatrix4fv and disabled the transformation for the grid as previously demonstrated. Most of this is shown in figure 13.

However, we created a shear matrix as shown in figure 14. This moves the shape horizontally only, creating the stretch effect.

References

https://www.brainvoyager.com/bv/doc/UsersGuide/CoordsAndTransforms/SpatialTransformationMatrices.html

A5

Output

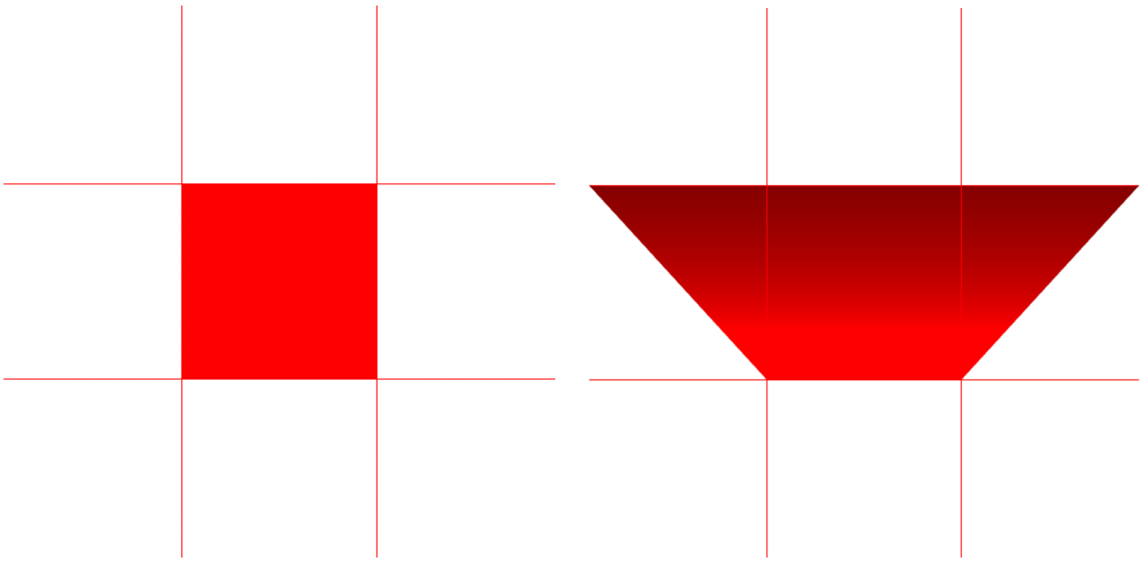

Figure 16: Output showing projective transformation

JavaScript Code Snippet

```
// A2-5 ADD NEW DECLARATIONS
var translate_loc, shear_loc, projective_loc, projective_inv_loc;
```

```
    // A2-5 GET NECESSARY UNIFORM LOCATIONS
    translate_loc = gl.getUniformLocation(program, 'translate');
    shear_loc = gl.getUniformLocation(program, 'shear');
    projective_loc = gl.getUniformLocation(program, 'projective');
    projective_inv_loc = gl.getUniformLocation(program, 'projective_inv');
```

```
    let projective = [[4/(2+side), 0, 0, 0],
                      [0, 1, 0, ((side-2)*side)/(2*(side+2))],
                      [0, 0, 1, 0],
                      [0, (2*(side-2))/(side*(side+2)), 0, 1]];

    let projective_inv = [[2+side, 0, 0, 0],
                          [0, Math.pow(2+side,2)/(2*side), 0, 1-Math.pow(side,2)/4],
                          [0, 0, 4, 0],
                          [0, 4/Math.pow(side,2)-1, 0,  Math.pow(2+side,2)/(2*side)]];
```


```
    let projective_inv = [[2+side, 0, 0, 0],
                          [0, Math.pow(2+side,2)/(2*side), 0, 1-Math.pow(side,2)/4],
                          [0, 0, 4, 0],
                          [0, 4/Math.pow(side,2)-1, 0,  Math.pow(2+side,2)/(2*side)]];
```


```
    // A2 DISABLE YOUR TRANSFORMATIONS
    gl.uniformMatrix4fv(translate_loc, false, mat_float_flat_transpose(identity));
    gl.uniformMatrix4fv(pre_rotate_loc, false, mat_float_flat_transpose(identity));
    gl.uniformMatrix4fv(pre_scale_loc, false, mat_float_flat_transpose(identity));
    gl.uniformMatrix4fv(rotate_loc, false, mat_float_flat_transpose(identity));
    gl.uniformMatrix4fv(shear_loc, false, mat_float_flat_transpose(identity));
    gl.uniformMatrix4fv(projective_loc, false, mat_float_flat_transpose(identity));
    gl.uniformMatrix4fv(projective_inv_loc, false, mat_float_flat_transpose(identity));
```

Figure 17: JavaScript Code Snippet from 'transforming-square.js' file

Vertex Shader Code Snippet

```
// 4x4 matrices
uniform mat4 pre_scale, pre_rotate, rotate, shear, projective, projective_inv;
```

```
    // A1, A2, A3, A4, A5 -- MODIFY HERE
    gl_Position =  projective_inv * projective * point;
```

Or

```
    // A1, A2, A3, A4, A5 -- MODIFY HERE
    gl_Position =  projective * point;
```

```
    // pass uniform colour to fragment shader varying
    // A5 -- MODIFY HERE
    colour = vec4(v0: gl_Position.w, v1: 0.0, v2: 0.0, v3: 1.0);
```

Figure 18: Vertex Shader Code Snippet from 'transforming-vert.glsl' file

Explanation

For this exercise, we created a trapezium with a matrix transformation called projective and then reversed it with the inverse of that matrix. This process is shown in figures 15 and 16 and is the same as the previous exercises demonstrated except for the given matrix.

Note that this is no longer an affine transformation. This is because parallelism has not been preserved. Instead, this is a projective transformation.

The Or code snippet in Vertex Shader creates either of the two outputs.

We also changed the value of colour. This illustrated that the w value is lower at the top of the shape.

Colour = vec4(gl_Position.w, 0.0, 0.0, 1.0)

References

https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial/Using_shaders_to_apply_color_in_WebGL

Exercise B

B1

Output

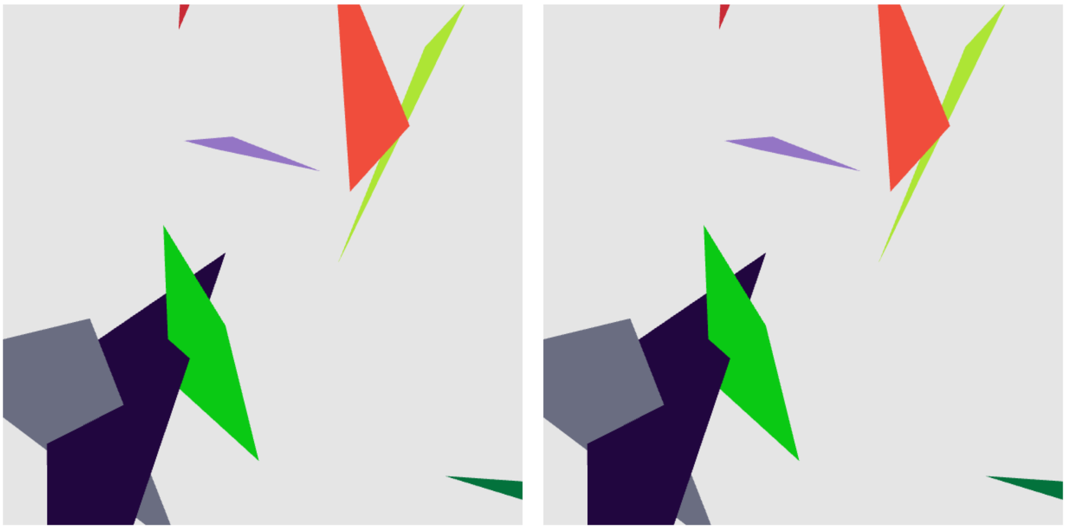

Figure 19: Output showing two canvases

JavaScript Code Snippet

```
    // B1: INSERT CODE HERE
    ctx.drawImage(gl.canvas, 0, 0);
```

```
    // B1: INSERT CALLBACK CODE HERE
    window.setTimeout(render_control, 1000/60);
```

Figure 20: JavaScript Snippet from 'projection.js'

Explanation

For this exercise we copied the initial render into another canvas. This will give us two points of view which we will implement later.

This was done by including ctx.drawImage(gl.canvas, 0,0); which is found in the API.

drawImage(image, dx, dy)

Image is an element to draw, in this case it is our gl.canvas. dx and dy represents the x-axis and y-axis coordinates. Finally, we include:

Window.setTimeout(render_control, 1000/60)

To call the functon again in case anything has changed.

References

https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/drawImage

B2

Output

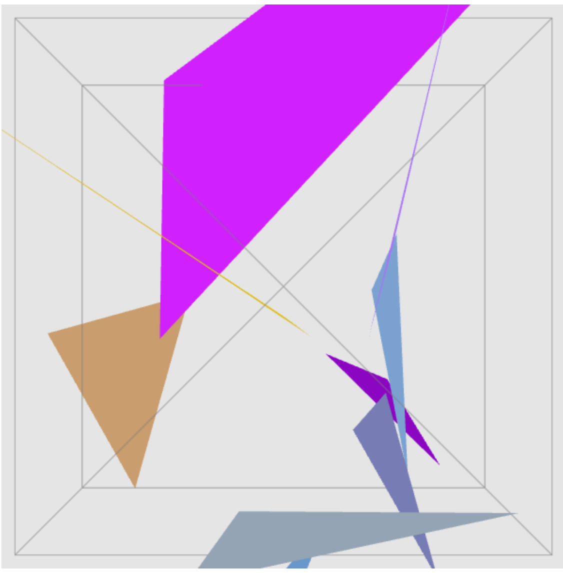

Figure 21: Output showing clipping planes of frustum

JavaScript Code Snippet

```
    // camera position for backed-off view (remember +z is behind camera)
    let eye = [0.0, 0.0, 50.0];

    // target point and up direction of camera
    let at = [0.0, 0.0, -max_depth];
    let up = [0.0, 1.0, 0.0];

    // transformation to apply to scene
    modelview = mat_lookat(eye,at,up);

    // camera matrix with adjusted clipping planes (so that we see everything)
    projection = mat_perspective(70, aspect, 1, 500);

    // B2, B3, B4 -- MODIFY RENDERING CONTROL
    render_triangles = true;
    render_near_plane = true;
    render_far_plane = true;
    render_side_planes = true;
    render_near_edges = true;
    render_far_edges = true;
    render_side_edges = true;
    render(rotation, translation);
```

```
    if(render_near_plane || render_far_plane || render_side_edges)
        console_log('Drawing frustum...');

    let offset = 0;

    if(render_far_plane) {
        gl.drawElements(gl.TRIANGLE_STRIP, 4, gl.UNSIGNED_SHORT, offset*UNSIGNED_SHORT_size);
        console_log('   far plane @' + offset);
    }
    offset += 4;

    if(render_near_plane) {
        gl.drawElements(gl.TRIANGLE_STRIP, 4, gl.UNSIGNED_SHORT, offset*UNSIGNED_SHORT_size);
        console_log(' near plane @' + offset);
    }
    offset += 4;
```

```
    if(render_near_edges) {
        gl.drawElements(gl.LINE_LOOP, 4, gl.UNSIGNED_SHORT, offset*UNSIGNED_SHORT_size);
        console_log(' near edges @' + offset);
    }
    offset += 4;

    if(render_far_edges) {
        gl.drawElements(gl.LINE_LOOP, 4, gl.UNSIGNED_SHORT, offset*UNSIGNED_SHORT_size);
        console_log('  far edges @' + offset);
    }
    offset += 4;

    for(let k = 0; k < 4; k++) {
        if(render_side_edges) {
            gl.drawElements(gl.LINES, 2, gl.UNSIGNED_SHORT, offset*UNSIGNED_SHORT_size);
            console_log('  side edge @' + offset);
        }
        offset += 2;
    }
}
```

Figure 22: JavaScript Snippet from 'projection.js'

Explanation

For this exercise we want to outline the frustum in 3D. We do this by creating a camera eye, its parameters, direction, coordinates and perspective and finally setting the rendering control parameters.

These rendering control parameters are used later as shown in figures 22, to draw additional elements.

References

n/a

B3

Output

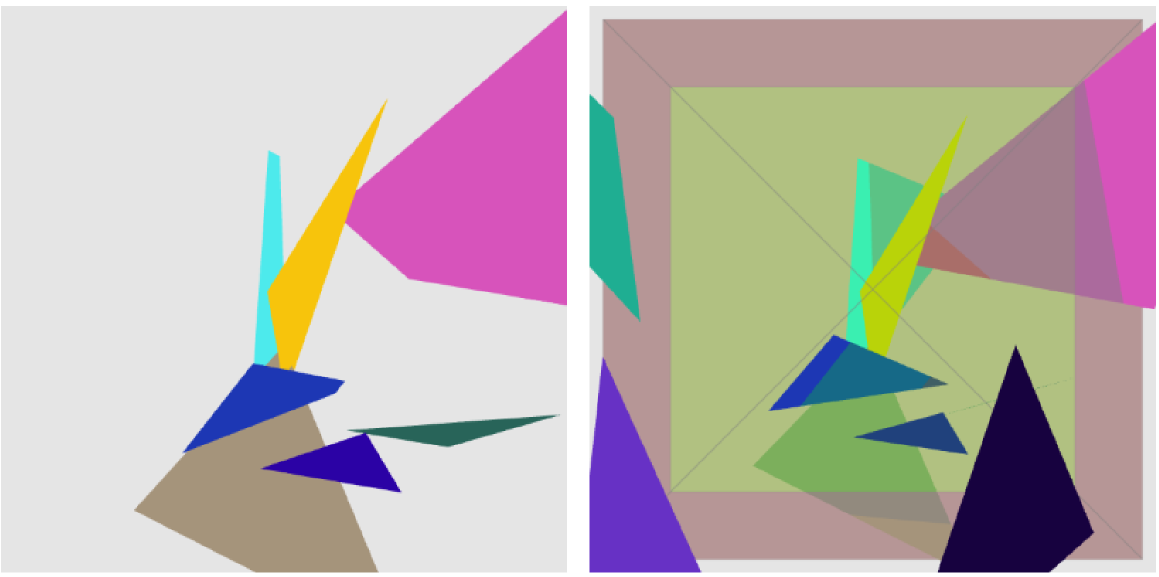

Figure 23: Output showing clipping and not clipping examples using the frustum

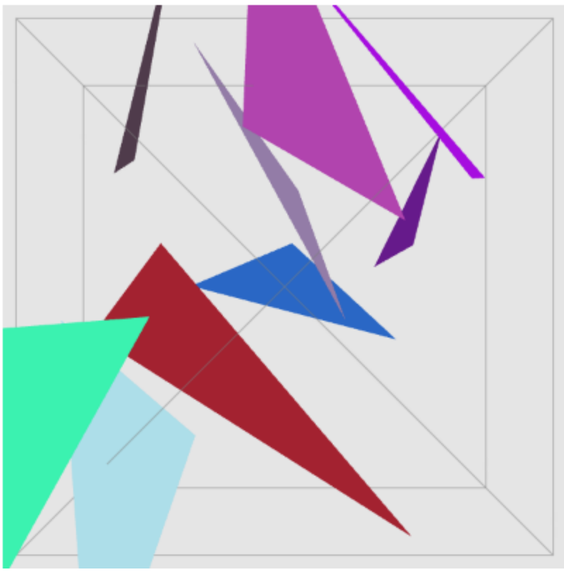

Figure 24: Another output showing before the changes

JavaScript Code Snippet

```
    // B2, B3, B4 -- MODIFY RENDERING CONTROL
    render_triangles = true;
    render_near_plane = true;
    render_far_plane = true;
    render_side_planes = true;
    render_near_edges = true;
    render_far_edges = true;
    render_side_edges = true;
```

Figure 25: JavaScript Snippet from 'projection.js'

Explanation

We can see by these two canvases, that primitives in full colour are outside the frustum in the second image and have been clipped (removed) in the first canvas.

This is because there are Boolean checks to call drawElements() later in the code. As these parameters are set to false in the first call of Render(), the primitives are not drawn unlike the second time it is called.

I refreshed the page to screenshot another output to follow what was ask for B3i-ii.png image.

References

[WebGLRenderingContext.drawElements() - Web APIs | MDN (mozilla.org)]

B4

Output

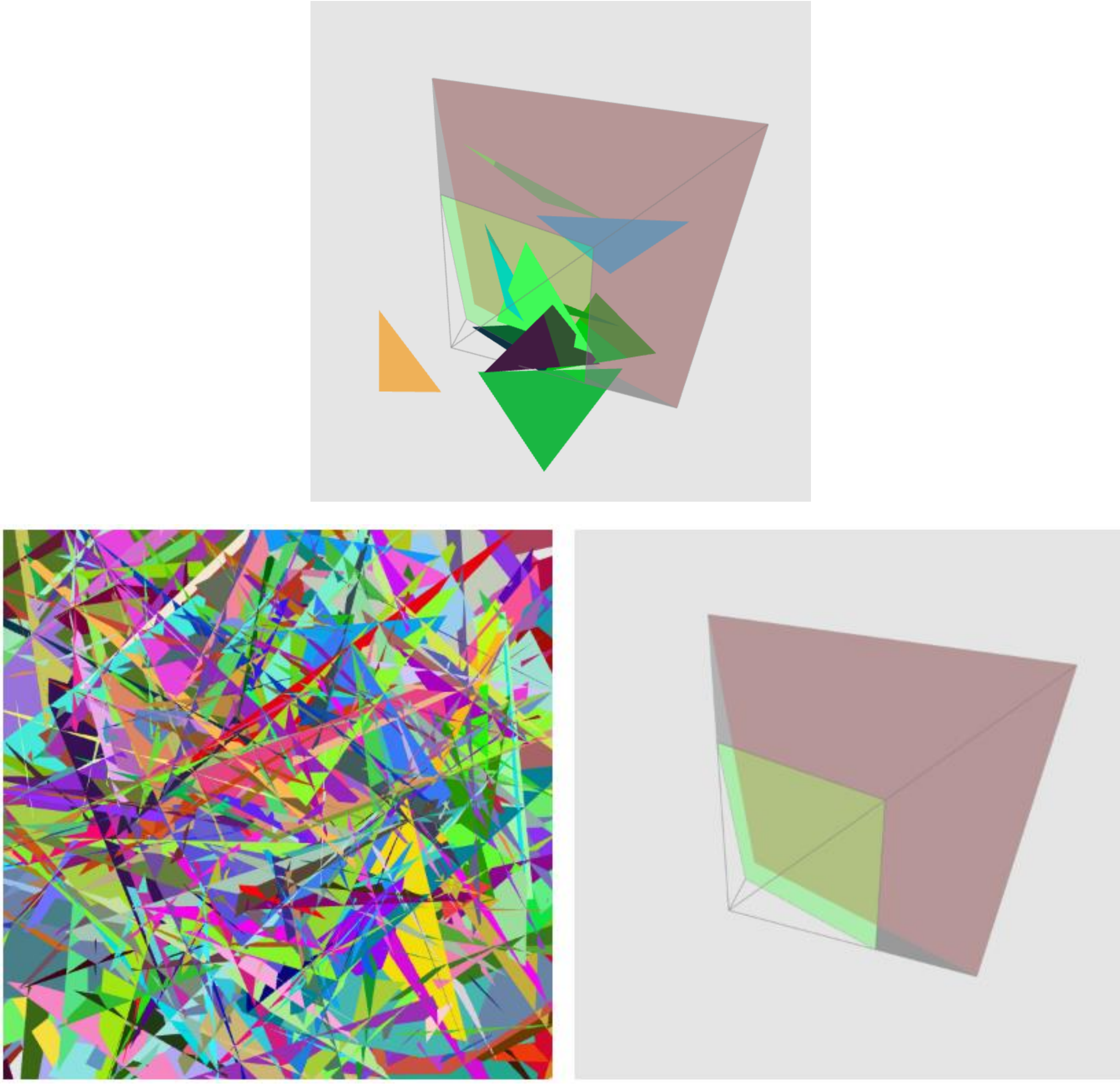

Figure 26: Output showing the change in view. Also includes 1000 triangles and removing triangle render

JavaScript Code Snippet

```javascript
    // camera position for backed-off view (remember +z is behind camera)
    let eye = [110.0, 140.0, 70.0];
```

```javascript
// B4 MODIFY SCENE PARAMETERS
var num_triangles = 1000;
```

Figure 27: JavaScript code Snippet from 'projection.js' file

Explanation

For this exercise, we changed the perspective of the camera, increased the number of triangles to 1000 and removed the rendering of the triangles so we can see the frustum more clearly. This is shown in the code snippet and the three canvases displayed in output.

References

n/a

B5

Output

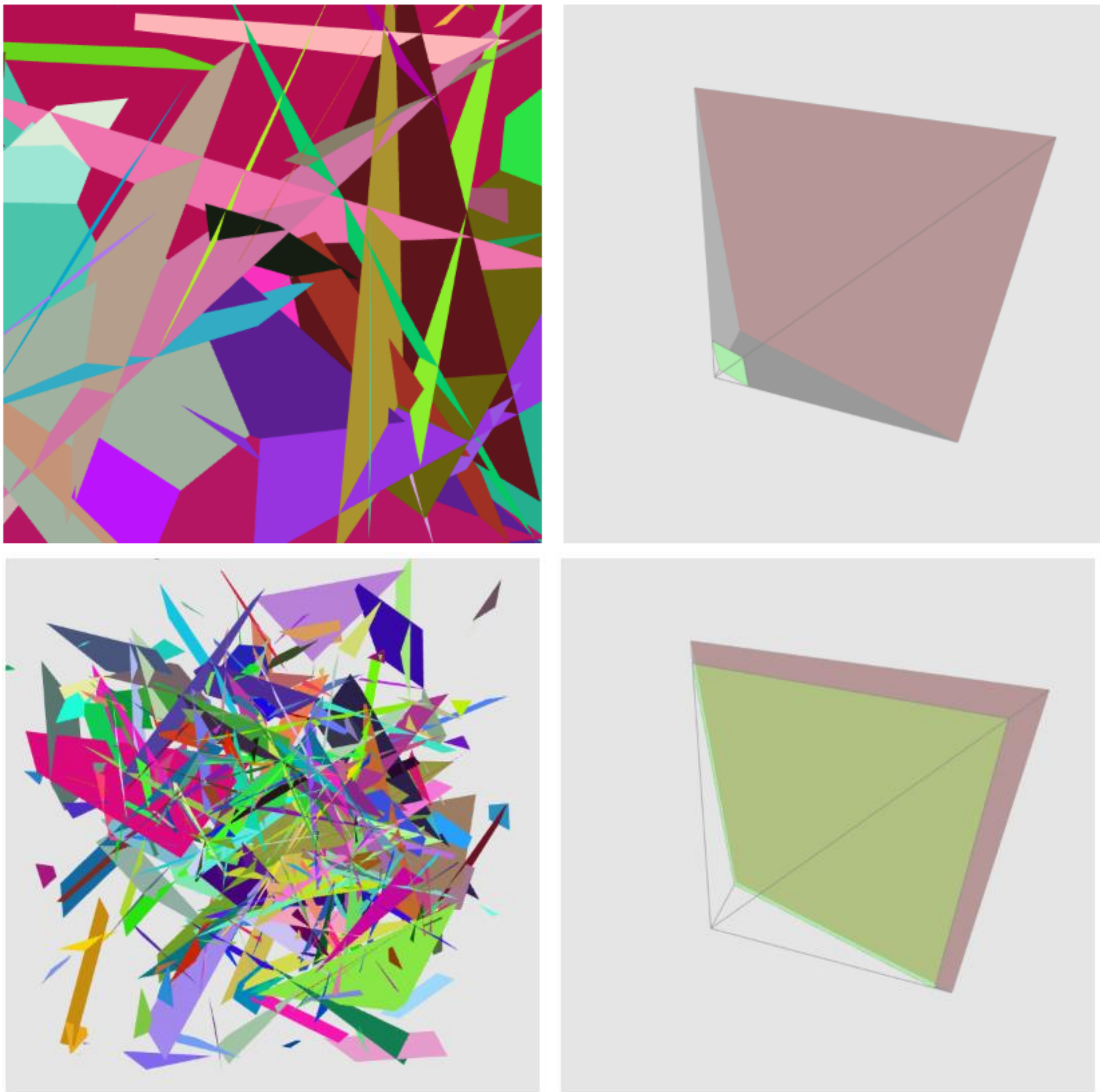

Figure 28: Output showing the change in near

JavaScript Code Snippet

```
// B5 MODIFY CAMERA PARAMETERS
let vert_fov = Math.PI/2;
let near = 10;
let far = 100;
let aspect = 1;
```

```
// B5 MODIFY CAMERA PARAMETERS
let vert_fov = Math.PI/2;
let near = 90;
let far = 100;
let aspect = 1;
```

Figure 29: JavaScript Code Snippet from 'projection.js' file

Explanation

For this exercise, we just adjusted the near plane forward and backwards inside the frustum. This affected what primitives were inside the frustum and therefore changed which primitives were rendered as shown in the two canvases. We can also point out that the triangles are smaller with near 90 than near 10. This is because the triangles are further away.

References

n/a

Exercise C

C1

Output

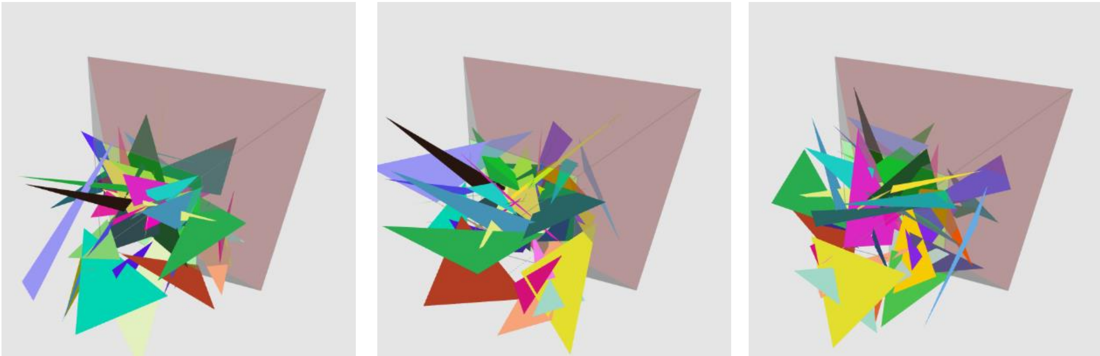

Figure 30: Output showing rotation inside cluster of triangles

JavaScript Code Snippet

```
    // C1: GET ROTATION AND TRANSLATION LOCATIONS HERE
    rotation_loc = gl.getUniformLocation(program, 'rotation');
    translation_loc = gl.getUniformLocation(program, 'translation');
```

```
    // C1: DEFINE ROTATION AND TRANSLATION HERE
    let rotation = [[Math.sin(theta),  0, Math.cos(theta),  0],
                    [0,                1,         0,        0],
                    [-Math.cos(theta), 0, Math.sin(theta),  0],
                    [0,                0,         0,        1]];


    let translation = [[1, 0, 0,  0],
                       [0, 1, 0,    0],
                       [0, 0, 1,    -max_depth/2],
                       [0, 0, 0,    1  ]];
```

```
    // C1: SET ROTATION AND TRANSLATION HERE
    gl.uniformMatrix4fv(rotation_loc, false, mat_float_flat_transpose(rotation));
    gl.uniformMatrix4fv(translation_loc, false, mat_float_flat_transpose(translation));
```

```
    // C1: DISABLE ROTATION AND TRANSLATION HERE
    gl.uniformMatrix4fv(rotation_loc, false, mat_float_flat_transpose(identity));
    gl.uniformMatrix4fv(translation_loc, false, mat_float_flat_transpose(identity));
```

Figure 31: JavaScript Code Snippet from 'projection.js'

Vertex Shader Code Snippet

```
    // C1: DEFINE INVERSE TRANSLATION MATRIX HERE
    mat4 translation_inv = translation;
    translation_inv[3][2] = abs(x: translation_inv[3][2]);

    // convert to homogeneous coordinates
    vec4 point = vec4(v0: vertex.x, v1: vertex.y, v2: vertex.z, v3: 1.0);

    // C1: USE ROTATION AND TRANSLATION MATRICES HERE
    point =   translation * rotation * translation_inv  * point;

    // transform and then project -- note that division is performed later
    gl_Position = projection * modelview * point;
```

Figure 32: Vertex Shader Code Snippet from 'projection-vert.glsl' file

Rotation Matrix Axis y

$$R_y = \begin{bmatrix} \cos\psi & 0 & \sin\psi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\psi & 0 & \cos\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 33: Rotation matrix y axis

Explanation

For this exercise, we rotate the triangles around a vertical axis through the centre of the cluster. This was achieved by creating a rotation and translation matrix. The rotation matrix follows the layout of figure 33.

I changed the cos / sin around to rotate in the opposite direction (as shown in the video).

I then get, set and disabled the transformations as shown previously in exercises A. This is shown in the JavaScript Code Snippet.

Finally, I created the inverse of the translation matrix in the Vertex Shader Code Snippet image.

Altogether, this translates the cluster to the centre, rotates on that position and then translates back to their original position.

References

n/a