midterm_examples.js

```
import { Shader, readShaderFile } from './examples/WebGLSource/util/shader.js';
import { resizeAspectRatio } from './examples/WebGLSource/util/util.js';

/* ================= Backgrounds ================= */
// document, canvas, window 구분
/*
document | HTML 문서 전체를 나타내는 객체. DOM 조작, 이벤트 등록 등에 사용됨.
canvas | HTML 문서 내 <canvas> 태그에 해당하는 요소. 여기서는 WebGL 캔버스.
window | 브라우저 전체 창을 나타내는 객체. 전역 객체이자 이벤트 루프의 루트.
*/
/* ================= Mandatory Pipeline ================= */
const canvas = document.getElementById('glCanvas');
const gl = canvas.getContext('webgl2');

let vao = null;
let shader = null;
let isInitialized = false;

// DOMContentLoaded event
// 1) 모든 HTML 문서가 완전히 load되고 parsing된 후 발생
// 2) 모든 resource (images, css, js 등) 가 완전히 load된 후 발생
// 3) 모든 DOM 요소가 생성된 후 발생
// DOM: Document Object Model로 HTML의 tree 구조로 표현되는 object model
// 모든 code를 이 listener 안에 넣는 것은 mouse click event를 원활하게 처리하기 위해서임
// mouse input을 사용할 때 이와 같이 main을 call 한다.
document.addEventListener('DOMContentLoaded', () => {
    if(isInitialized) {
        console.log("Already Initialized");
        return;
    }
    // main() 함수는 프로그램 초기화 성공 여부 (success = true or false)를 return 한다.
    // success = true 이면 프로그램을 계속 실행
    // success = false 이면 프로그램을 종료
    // catch block은 success = true 인 경우 main()의 내부에서 발생하는 error를 처리하는 부분임
    // Call main function
    main().then(success => {
        if(!success) {
            console.log('Terminate the Program');
            return;
        }

        isInitialized = true;
    }).catch(error => {
        // 1. main() 함수 자체가 실행되지 못하는 경우
        // 2. then() 블록 내부에서 발생하는 에러
        // 3. 비동기 작업 중 발생하는 처리되지 않은 에러
        // 다음과 같은 예기치 않은 에러들을 처리:
        console.error('error occured:', error);
    });
});

// Initialize WebGL settings: canvas size, resizeAspectRatio, viewport, clearcolor
function initWebGL() {
    if (!gl) {
        console.error('WebGL 2 is not supported by your browser.');
        return false;
    }

    canvas.width = 700;
    canvas.height = 700;
    resizeAspectRatio(gl, canvas);

    //  viewport and clear color
    gl.viewport(0, 0, canvas.canvas.width, canvas.height);
    gl.clearColor(0.1, 0.2, 0.3, 1.0);
}

// Buffer Setup: vertices array, indices array, color array etc -> vao -> vbo -> set attribute pointers
function setupBuffers() {

}

// Initialize Shader
async function initShader() {
    const vertexShaderSource = await readShaderFile('shVert.glsl');
    const fragmentShaderSource = await readShaderFile('shFrag.glsl');
    shader = new Shader(gl, vertexShaderSource, fragmentShaderSource);
}

// Render loop
function render() {
    // gl.COLOR_BUFFER_BIT: 프레임 버퍼(화면)를 초기화하거나 지울 때 사용하는 상수
    gl.clear(gl.COLOR_BUFFER_BIT); // 현재 렌더링 중인 화면을 지움 (= 초기화)
    // Draw something here
    // shader use
    shader.use();

    // bind VAO each render() & draw
    gl.bindVertexArray(vao);
    gl.drawArrays(gl.TRIANGLES, 0, 6); // gl.drawArrays(mode, first, count)
}

async function main() {

    await initShader();

    setupBuffers();

    return true; // <- initWebGL에서 main().then 호출을 위해 boolean return
}

/* ================= WEEK 02 ================= */
// 01: key press event listener
function setupKeyboardEvents(){
```

```
    document.addEventListener('keydown', (event) => {
    console.log(`key Pressed: ${event.key}`);
    key = event.key;
    switch(key) {
        // ...
    }
});
}
// main()
async function main() {
    try {
        if (!initWebGL()) {
            // ...
        }

        // ...

        initShader();
        setupBuffers();
        // ...
        setupKeyboardEvents(); // <- main 안에서 호출출
        return true;
    } catch (error) {
        //...
        return false;
    }
}

// 02: requestAnimationFrame();
// callback됨:
// animate함수 내에서 render 함수가 call 되고, animate 함수 파라미터가 time 종류라면,
// if requestAnimationFrame(animate): time(ms 단위) 받아서 다음 프레임을 request하는 구조
// render을 requestAnimationFrame(render) 이렇게 받는다면, 매 프레임에서 렌더 진행하는 방식으로 animate
// main 함수 내에서 requestAnimationFrame(render)을 호출하는 render()을 직접 부르거나,
// main().then에서 requestAnimationFrame(animate) 이런식으로 호출함

// 03: vertex array가 두개 이상의 정보를 포함할 경우 (예: position, colors)
function setupBuffers() {
    const vertices = new Float32Array([
        // positions      // colors
         0.5, -0.5, 0.0,  1.0, 0.0, 0.0,  // bottom right, red
        -0.5, -0.5, 0.0,  0.0, 1.0, 0.0,  // bottom left, green
         0.0,  0.5, 0.0,  0.0, 0.0, 1.0   // top center, blue
    ]);

    vao = gl.createVertexArray();
    gl.bindVertexArray(vao);

    const vertexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);

    //한 vertex의 데이터 구조 (위의 vertices array 참조))
    //[x y z r g b] = 6개의 float 값 (size = 6)
    //↑      ↑---- color 데이터 시작 (offset: 3 * Float32Array.BYTES_PER_ELEMENT)
    //↑---------- position 데이터 시작 (offset: 0)
    // Stride = 한 vertex 정보의 크기 = 한 정점에서 다음 정점까지 몇 바이트 뛰어넘을지
    shader.setVertexAttribPointer("a_position", 3, gl.FLOAT, false, 6 * Float32Array.BYTES_PER_ELEMENT, 0);
    shader.setVertexAttribPointer("a_color", 3, gl.FLOAT, false, 6 * Float32Array.BYTES_PER_ELEMENT,
                                3 * Float32Array.BYTES_PER_ELEMENT);

}

// 04: Uniform variable을 이용한 flip
/* 'shVert.glsl':
    ...
    uniform float verticalFlip;
    void main() {
        gl_Position = vec4(a_position[0], a_position[1]*verticalFlip, a_position[2], 1.0);
    } */
/* .js 내부:
    some logic -> verticalFlip = -verticlFlip; */

/* ================= WEEK 03 ================= */
// 01: 캔버스 좌표 - WebGL 좌표 - mouse click
// 캔버스 좌표: 캔버스 좌측 상단이 (0, 0), 우측 하단이 (canvas.width, canvas.height)
// WebGL 좌표 (NDC): 캔버스 좌측 하단이 (-1, -1), 우측 상단이 (1, 1)

// [a,b]로 normalize: x_normalized = (x - min) / (max - min) * (b - a) + a
function convertToWebGLCoordinates(x, y) {
    return [
        ((x / canvas.width) * 2 -1), // x/canvas.width 는 0 ~ 1 사이의 값, 이것을 * 2 - 1 하면 -1 ~ 1 사이의 값
        -((y / canvas.height) * 2 - 1) // y canvas 좌표는 상하를 뒤집어 주어야 하므로 -1을 곱함
    ]
}
/*
    browser window
    +---------------------------------------+
    | toolbar, address bar, etc.            |
    +---------------------------------------+
    | browser viewport (컨텐츠 표시 영역)       |
    | +-----------------------------------+ |
    | |                                   | |
    | |    canvas                         | |
    | |    +----------------+             | |
    | |    |                |             | |
    | |    |      *         |             | |
    | |    |                |             | |
    | |    +----------------+             | |
    | |                                   | |
    | +-----------------------------------+ |
    +---------------------------------------+

    *: mouse click position
```

```
    event.clientX = browser viewport 왼쪽 경계에서 마우스 클릭 위치까지의 거리
    event.clientY = browser viewport 상단 경계에서 마우스 클릭 위치까지의 거리
    rect.left = browser viewport 왼쪽 경계에서 canvas 왼쪽 경계까지의 거리
    rect.top = browser viewport 상단 경계에서 canvas 상단 경계까지의 거리

    x = event.clientX - rect.left  // canvas 내에서의 클릭 x 좌표
    y = event.clientY - rect.top   // canvas 내에서의 클릭 y 좌표
*/

function setupMouseEvents() {
    function handleMouseDown(event) {
        event.preventDefault(); // 이미 존재할 수 있는 기본 동작을 방지
        event.stopPropagation(); // event가 상위 요소 (div, body, html 등)으로 전파되지 않도록 방지

        const rect = canvas.getBoundingClientRect(); // canvas를 나타내는 rect 객체를 반환
        const x = event.clientX - rect.left;  // canvas 내 x 좌표
        const y = event.clientY - rect.top;   // canvas 내 y 좌표

        if (!isDrawing && lines.length < 2) {
            // 1번 또는 2번 선분을 그리고 있는 도중이 아닌 경우 (즉, mouse down 상태가 아닌 경우)
            // 캔버스 좌표를 WebGL 좌표로 변환하여 선분의 시작점을 설정
            let [glX, glY] = convertToWebGLCoordinates(x, y);
            startPoint = [glX, glY];
            isDrawing = true; // 이제 mouse button을 놓을 때까지 계속 true로 둠. 즉, mouse down 상태가 됨
        }
    }

    function handleMouseMove(event) {
        if (isDrawing) { // 1번 또는 2번 선분을 그리고 있는 도중인 경우
            const rect = canvas.getBoundingClientRect();
            const x = event.clientX - rect.left;
            const y = event.clientY - rect.top;

            let [glX, glY] = convertToWebGLCoordinates(x, y);
            tempEndPoint = [glX, glY]; // 임시 선분의 끝 point
            render();
        }
    }

    function handleMouseUp() {
        if (isDrawing && tempEndPoint) {

            // lines.push([...startPoint, ...tempEndPoint])
            //   : startPoint와 tempEndPoint를 펼쳐서 하나의 array로 합친 후 lines에 추가
            // ex) lines = [] 이고 startPoint = [1, 2], tempEndPoint = [3, 4] 이면,
            //     lines = [[1, 2, 3, 4]] 이 됨
            // ex) lines = [[1, 2, 3, 4]] 이고 startPoint = [5, 6], tempEndPoint = [7, 8] 이면,
            //     lines = [[1, 2, 3, 4], [5, 6, 7, 8]] 이 됨

            lines.push([...startPoint, ...tempEndPoint]);

            if (lines.length == 1) {
                updateText(textOverlay, "First line segment: (" + lines[0][0].toFixed(2) + ", " + lines[0][1].toFixed(2) +
                    ") ~ (" + lines[0][2].toFixed(2) + ", " + lines[0][3].toFixed(2) + ")");
                updateText(textOverlay2, "Click and drag to draw the second line segment");
            }
            else { // lines.length == 2
                updateText(textOverlay2, "Second line segment: (" + lines[1][0].toFixed(2) + ", " + lines[1][1].toFixed(2) +
                    ") ~ (" + lines[1][2].toFixed(2) + ", " + lines[1][3].toFixed(2) + ")");
            }

            isDrawing = false;
            startPoint = null;
            tempEndPoint = null;
            render();
        }
    }

    // 이 때 event listener은 마우스 클릭 -> canvas
    canvas.addEventListener("mousedown", handleMouseDown);
    canvas.addEventListener("mousemove", handleMouseMove);
    canvas.addEventListener("mouseup", handleMouseUp);
}

function render() {
    gl.clear(gl.COLOR_BUFFER_BIT);

    gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);

    shader.use();

    // 저장된 선들 그리기
    let num = 0;
    for (let line of lines) {
        if (num == 0) { // 첫 번째 선분인 경우, yellow
            shader.setVec4("u_color", [1.0, 1.0, 0.0, 1.0]);
        }
        else { // num == 1 (2번째 선분인 경우), red
            shader.setVec4("u_color", [1.0, 0.0, 1.0, 1.0]);
        }
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(line), gl.STATIC_DRAW);
        gl.bindVertexArray(vao);
        gl.drawArrays(gl.LINES, 0, 2);
        num++;
    }

    // 임시 선 그리기
    if (isDrawing && startPoint && tempEndPoint) {
        shader.setVec4("u_color", [0.5, 0.5, 0.5, 1.0]); // 임시 선분의 color는 회색
```

```
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array([...startPoint, ...tempEndPoint]),
                      gl.STATIC_DRAW);
        gl.bindVertexArray(vao);
        gl.drawArrays(gl.LINES, 0, 2);
    }

    // axes 그리기
    axes.draw(mat4.create(), mat4.create()); // 두 개의 identity matrix를 parameter로 전달
}

/* ================= WEEK 04 ================= */
// 'gl-matrix-main.js': mat4 함수 제공.
// mat4.create, multiply
// mat4.lookAt(viewMatrix, COP, Target, Up vector): 파라미터들로부터 view matrix를 계산하여 첫 번째 parameter인 viewMatrix에 담음
// mat4.ortho(projMatrix(output임), left, right, bottom, top, near, far)
// mat4.perspective(projMatrix, fov_angle, width_height_ratio, near, far)
function getTransformMatrices() {
    const T = mat4.create();
    const R = mat4.create();
    const S = mat4.create();

    mat4.translate(T, T, [0.5, 0.5, 0]);  // translation by (0.5, 0.5)
    mat4.rotate(R, R, rotationAngle, [0, 0, 1]); // rotation about z-axis
    mat4.scale(S, S, [0.3, 0.3, 1]); // scale by (0.3, 0.3)

    return { T, R, S };
}

function applyTransform(type) {
    finalTransform = mat4.create();
    const { T, R, S } = getTransformMatrices();

    const transformOrder = {
        'TRS': [T, R, S],
        'TSR': [T, S, R],
        'RTS': [R, T, S],
        'RST': [R, S, T],
        'STR': [S, T, R],
        'SRT': [S, R, T]
    };

    /*
      type은 'TRS', 'TSR', 'RTS', 'RST', 'STR', 'SRT' 중 하나
      array.forEach(...) : 각 type의 element T or R or S 에 대해 반복
    */
    if (transformOrder[type]) {
        transformOrder[type].forEach(matrix => {
            mat4.multiply(finalTransform, matrix, finalTransform);
        });
    }
}

let lastTime = 0;


function animate(currentTime) {

    if (!lastTime) lastTime = currentTime; // if lastTime == 0
    // deltaTime: 이전 frame에서부터의 elapsed time (in seconds)
    const deltaTime = (currentTime - lastTime) / 1000;
    lastTime = currentTime; // 기록록

    if (isAnimating && currentTransformType) {
        // 2초당 1회전, 즉, 1초당 180도 회전
        rotationAngle += Math.PI * deltaTime;
        applyTransform(currentTransformType);
    }
    render();

    requestAnimationFrame(animate);
}

/* ================= WEEK 05 ================= */
// 01: Viewing
// vertex shader에 전달해줄 viewMatrix, projMatrix
let viewMatrix = mat4.create();
let projMatrix = mat4.create();
let startTime;

function render() {
    const currentTime = Date.now() // 명시적인 시간 지정
    const elapsedTime = (currentTime - startTime) / 1000.0;

    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    gl.enable(gl.DEPTH_TEST);

    const modelMatrix = mat4.create();

    mat4.rotateX(modelMatrix, modelMatrix, glMatrix.toRadian(3 * elapsedTime));
    mat4.rotateZ(modelMatrix, modelMatrix, glMatrix.toRadian(3 * elapsedTime));

    shader.use();
    // uniform setter
    shader.setMat4('u_model', modelMatrix);
    shader.setMat4('u_view', viewMatrix);
    shader.setMat4('u_projection', projMatrix);
    Object.draw(shader);

    requestAnimationFrame(render);
}
async function main() {
    try {
        if (!initWebGL()) {
            throw new Error('WebGL 초기화 실패');
        }

        await initShader();
```

```
        // View transformation matrix (move the cube to (0, 0, -4))
        // default camera is at the origin and looking at the infinite in the -z axis
        // invariant in the program
        mat4.translate(viewMatrix, viewMatrix, vec3.fromValues(0, 0, -4));

        // Projection transformation matrix (invariant in the program)
        mat4.perspective(
            projMatrix,
            glMatrix.toRadian(60),  // field of view (fov, degree)
            canvas.width / canvas.height, // aspect ratio
            0.1, // near
            1000.0 // far
        );

        // starting time (global variable) for animation
        startTime = Date.now();

        // call the render function the first time for animation
        requestAnimationFrame(render);

        return true;

    } catch (error) {
        console.error('Failed to initialize program:', error);
        alert('Failed to initialize program');
        return false;
    }
}

// 02: Camera Circle
const cameraCircleRadius = 5.0;
const cameraCircleHeight = 2.0;
const cameraCircleSpeed = 90.0;

function render() {
    // Viewing transformation matrix
    let camX = cameraCircleRadius * Math.sin(glMatrix.toRadian(cameraCircleSpeed *
elapsedTime));
    let camZ = cameraCircleRadius * Math.cos(glMatrix.toRadian(cameraCircleSpeed *
elapsedTime));
    mat4.lookAt(viewMatrix,
        vec3.fromValues(camX, cameraCircleHeight, camZ), // camera position
        vec3.fromValues(0, 0, 0), // look at origin
        vec3.fromValues(0, 1, 0)); // up vector
}
```