

midterm_examples.js

```
1 import { Shader, readShaderFile } from './examples/WebGLSource/util/shader.js';
2 import { resizeAspectRatio } from './examples/WebGLSource/util/util.js';
3
4 /* ===== Backgrounds ===== */
5 // document, canvas, window 구분
6 /*
7 document | HTML 문서 전체를 나타내는 객체. DOM 조작, 이벤트 등록 등에 사용됨.
8 canvas | HTML 문서 내 <canvas> 태그에 해당하는 요소. 여기서는 WebGL 캔버스.
9 window | 브라우저 전체 창을 나타내는 객체. 전역 객체이자 이벤트 루프의 루트.
10 */
11 /* ===== Mandatory Pipeline ===== */
12 const canvas = document.getElementById('glCanvas');
13 const gl = canvas.getContext('webgl2');
14
15 let vao = null;
16 let shader = null;
17 let isInitialized = false;
18
19 // DOMContentLoaded event
20 // 1) 모든 HTML 문서가 완전히 load되고 parsing된 후 발생
21 // 2) 모든 resource (images, css, js 등) 가 완전히 load된 후 발생
22 // 3) 모든 DOM 요소가 생성된 후 발생
23 // DOM: Document Object Model로 HTML의 tree 구조로 표현되는 object model
24 // 모든 code를 이 listener 안에 넣는 것은 mouse click event를 원활하게 처리하기 위해서임
25 // mouse input을 사용할 때 이와 같이 main을 call 한다.
26 document.addEventListener('DOMContentLoaded', () => {
27   if(isInitialized) {
28     console.log("Already Initialized");
29     return;
30   }
31   // main() 함수는 프로그램 초기화 성공 여부 (success = true or false)를 return 한다.
32   // success = true 이면 프로그램을 계속 실행
33   // success = false 이면 프로그램을 종료
34   // catch block은 success = true 인 경우 main()의 내부에서 발생하는 error를 처리하는 부분임
35   // Call main function
36   main().then(success => {
37     if(!success) {
38       console.log('Terminate the Program');
39       return;
40     }
41
42     isInitialized = true;
43   }).catch(error => {
44     // 1. main() 함수 자체가 실행되지 못하는 경우
45     // 2. then() 블록 내부에서 발생하는 에러
46     // 3. 비동기 작업 중 발생하는 처리되지 않은 에러
47     // 다음과 같은 예가지 않은 에러들을 처리:
48     console.error('error occured:', error);
49   });
50 });
51
```

```
52 // Initialize WebGL settings: canvas size, resizeAspectRatio, viewport, clearcolor
53 function initWebGL() {
54   if (!gl) {
55     console.error('WebGL 2 is not supported by your browser. ');
56     return false;
57   }
58
59   canvas.width = 700;
60   canvas.height = 700;
61   resizeAspectRatio(gl, canvas);
62
63   // viewport and clear color
64   gl.viewport(0, 0, canvas.canvas.width, canvas.height);
65   gl.clearColor(0.1, 0.2, 0.3, 1.0);
66 }
67
68 // Buffer Setup: vertices array, indices array, color array etc -> vao -> set
  attribute pointers
  function setupBuffers() {
69
70
71 }
72
73 // Initialize Shader
74 async function initShader() {
75   const vertexShaderSource = await readShaderFile('shVert.glsl');
76   const fragmentShaderSource = await readShaderFile('shFrag.glsl');
77   shader = new Shader(gl, vertexShaderSource, fragmentShaderSource);
78 }
79
80 // Render loop
81 function render() {
82   // gl.COLOR_BUFFER_BIT: 프레임 버퍼(화면)를 초기화하거나 지울 때 사용하는 상수
83   gl.clear(gl.COLOR_BUFFER_BIT); // 현재 렌더링 중인 화면을 지움 (= 초기화)
84   // Draw something here
85   // shader use
86   shader.use();
87
88   // bind VAO each render() & draw
89   gl.bindVertexArray(vao);
90   gl.drawArrays(gl.TRIANGLES, 0, 6); // gl.drawArrays(mode, first, count)
91 }
92
93 async function main() {
94
95   await initShader();
96
97   setupBuffers();
98
99   return true; // <- initWebGL에서 main().then 호출을 위해 boolean return
100 }
101
102 /* ===== WEEK 02 ===== */
103 // 01: key press event listener
104 function setupKeyboardEvents(){
```

```

105 document.addEventListner('keydown', (event) => {
106   console.log(`key Pressed: ${event.key}`);
107   key = event.key;
108   switch(key) {
109     // ...
110   }
111 });
112 }
113 // main()
114 async function main() {
115   try {
116     if (!initWebGL()) {
117       // ...
118     }
119     // ...
120   }
121 }
122   initShader();
123   setupBuffers();
124   // ...
125   setupKeyboardEvents(); // <- main 안에서 호출출
126   return true;
127 } catch (error) {
128   //...
129   return false;
130 }
131 }
132
133 // 02: requestAnimationFrame();
134 // callback됨:
135 // animate함수 내에서 render 함수가 call 되고, animate 함수 파라미터가 time 종류라면,
136 // if requestAnimationFrame(animate): time(ms 단위) 받아서 다음 프레임을 request하는 구조
137 // render를 requestAnimationFrame(render) 이렇게 받는다면, 매 프레임에서 렌더 진행하는 방식으로
  animate
138 // main 함수 내에서 requestAnimationFrame(render)을 호출하는 render()을 직접 부르거나,
139 // main().then에서 requestAnimationFrame(animate) 이런식으로 호출함
140
141 // 03: vertex array가 두개 이상의 정보를 포함할 경우 (예: position, colors)
142 function setupBuffers() {
143   const vertices = new Float32Array([
144     // positions // colors
145     0.5, -0.5, 0.0, 1.0, 0.0, 0.0, // bottom right, red
146     -0.5, -0.5, 0.0, 0.0, 1.0, 0.0, // bottom left, green
147     0.0, 0.5, 0.0, 0.0, 0.0, 1.0 // top center, blue
148   ]);
149
150   vao = gl.createVertexArray();
151   gl.bindVertexArray(vao);
152
153   const vertexBuffer = gl.createBuffer();
154   gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
155   gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
156
157   //한 vertex의 데이터 구조 (위의 vertices array 참조)

```

```

158   //[x y z r g b] = 6개의 float 값 (size = 6)
159   //↑ ↑---- color 데이터 시작 (offset: 3 * Float32Array.BYTES_PER_ELEMENT)
160   //↑----- position 데이터 시작 (offset: 0)
161   // Stride = 한 vertex 정보의 크기 = 한 정점에서 다음 정점까지 몇 바이트 뛰어넘을지
162   shader.setVertexAttribPointer("a_position", 3, gl.FLOAT, false, 6 *
Float32Array.BYTES_PER_ELEMENT, 0);
163   shader.setVertexAttribPointer("a_color", 3, gl.FLOAT, false, 6 *
Float32Array.BYTES_PER_ELEMENT,
3 * Float32Array.BYTES_PER_ELEMENT);
164 }
165
166 // 04: Uniform variable을 이용한 flip
167 /* 'shVert.glsl':
168   ...
169   uniform float verticalFlip;
170   void main() {
171     gl_Position = vec4(a_position[0], a_position[1]*verticalFlip, a_position[2], 1.0);
172   } */
173   // .js 내부:
174   some logic -> verticalFlip = -verticalFlip; */
175
176   /* ===== WEEK 03 ===== */
177   // 01: 캔버스 좌표 - WebGL 좌표 - mouse click
178   // 캔버스 좌표: 캔버스 좌측 상단이 (0, 0), 우측 하단이 (canvas.width, canvas.height)
179   // WebGL 좌표 (NDC): 캔버스 좌측 하단이 (-1, -1), 우측 상단이 (1, 1)
180
181   // [a,b]로 normalize: x_normalized = (x - min) / (max - min) * (b - a) + a
182   function convertToWebGLCoordinates(x, y) {
183     return [
184       ((x / canvas.width) * 2 - 1), // x/canvas.width 는 0 ~ 1 사이의 값, 이것을 * 2 - 1 하
185       면 -1 ~ 1 사이의 값
186       -((y / canvas.height) * 2 - 1) // y canvas 좌표는 상하를 뒤집어 주어야 하므로 -1을 곱함
187     ]
188   }
189   //
190   //
191   browser window
192   +-----+
193   | toolbar, address bar, etc. |
194   +-----+
195   | browser viewport (컨텐츠 표시 영역) |
196   +-----+
197   | | | | |
198   | | canvas | |
199   | | +-----+ |
200   | | | | |
201   | | | * | |
202   | | | | |
203   | | +-----+ |
204   | | | | |
205   | | +-----+ |
206   +-----+
207
208   *: mouse click position

```

```

209 event.clientX = browser.viewport 왼쪽 경계에서 마우스 클릭 위치까지의 거리
210 event.clientY = browser.viewport 상단 경계에서 마우스 클릭 위치까지의 거리
211 rect.left = browser.viewport 왼쪽 경계에서 canvas 왼쪽 경계까지의 거리
212 rect.top = browser.viewport 상단 경계에서 canvas 상단 경계까지의 거리
213
214
215 x = event.clientX - rect.left // canvas 내에서의 클릭 x 좌표
216 y = event.clientY - rect.top // canvas 내에서의 클릭 y 좌표
217 */
218
219 function setupMouseEvents() {
220     function handleMouseDown(event) {
221         event.preventDefault(); // 이미 존재할 수 있는 기본 동작을 방지
222         event.stopPropagation(); // event가 상위 요소 (div, body, html 등)으로 전파되지 않도록
223
224         방지
225
226         const rect = canvas.getBoundingClientRect(); // canvas를 나타내는 rect 객체를 반환
227         const x = event.clientX - rect.left; // canvas 내 x 좌표
228         const y = event.clientY - rect.top; // canvas 내 y 좌표
229
230         if (isDrawing && lines.length < 2) {
231             // 1번 또는 2번 선분을 그리고 있는 도중이 아닌 경우 (즉, mouse down 상태가 아닌 경우)
232             // 캔버스 좌표를 WebGL 좌표로 변환하여 선분의 시작점을 설정
233             let [glX, glY] = convertToWebGLCoordinates(x, y);
234             startPoint = [glX, glY];
235             isDrawing = true; // 이제 mouse button을 놓을 때까지 계속 true로 둬. 즉, mouse
236             down 상태가 됨
237         }
238     }
239
240     function handleMouseMove(event) {
241         if (isDrawing) { // 1번 또는 2번 선분을 그리고 있는 도중인 경우
242             const rect = canvas.getBoundingClientRect();
243             const x = event.clientX - rect.left;
244             const y = event.clientY - rect.top;
245
246             let [glX, glY] = convertToWebGLCoordinates(x, y);
247             tempEndPoint = [glX, glY]; // 임시 선분의 끝 point
248             render();
249         }
250     }
251
252     function handleMouseUp() {
253         if (isDrawing && tempEndPoint) {
254             // lines.push(...startPoint, ...tempEndPoint)
255             // : startPoint와 tempEndPoint를 펼쳐서 하나의 array로 합친 후 lines에 추가
256             // ex) lines = [] 이고 startPoint = [1, 2], tempEndPoint = [3, 4] 이면,
257             // lines = [[1, 2, 3, 4]] 이 됨
258             // ex) lines = [[1, 2, 3, 4]] 이고 startPoint = [5, 6], tempEndPoint = [7, 8]
259             // lines = [[1, 2, 3, 4], [5, 6, 7, 8]] 이 됨
260             lines.push(...startPoint, ...tempEndPoint);
261         }
262     }
263 }
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308

```

```

260 if (lines.length == 1) {
261     updateText(textOverlay, "First line segment: (" + lines[0][0].toFixed(2) +
262         ", " + lines[0][1].toFixed(2) +
263         ") ~ (" + lines[0][2].toFixed(2) + ", " + lines[0][3].toFixed(2) +
264         ")");
265     updateText(textOverlay2, "Click and drag to draw the second line
266     segment");
267 }
268 else { // lines.length == 2
269     updateText(textOverlay2, "Second line segment: (" + lines[1][0].toFixed(2)
270     + ", " + lines[1][1].toFixed(2) +
271     ") ~ (" + lines[1][2].toFixed(2) + ", " + lines[1][3].toFixed(2) +
272     ")");
273 }
274
275     isDrawing = false;
276     startPoint = null;
277     tempEndPoint = null;
278     render();
279 }
280
281 // 이 때 event listener은 마우스 클릭 -> canvas
282 canvas.addEventListener("mousedown", handleMouseDown);
283 canvas.addEventListener("mousemove", handleMouseMove);
284 canvas.addEventListener("mouseup", handleMouseUp);
285 }
286
287 function render() {
288     gl.clear(gl.COLOR_BUFFER_BIT);
289
290     gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
291
292     shader.use();
293
294     // 저장된 선들 그리기
295     let num = 0;
296     for (let line of lines) {
297         if (num == 0) { // 첫 번째 선분인 경우, yellow
298             shader.setVec4("u_color", [1.0, 1.0, 0.0, 1.0]);
299         }
300         else { // num == 1 (2번째 선분인 경우), red
301             shader.setVec4("u_color", [1.0, 0.0, 1.0, 1.0]);
302         }
303         gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(line), gl.STATIC_DRAW);
304         gl.bindVertexArray(vao);
305         gl.drawArrays(gl.LINES, 0, 2);
306         num++;
307     }
308
309     // 임시 선 그리기
310     if (isDrawing && startPoint && tempEndPoint) {
311         shader.setVec4("u_color", [0.5, 0.5, 1.0]); // 임시 선분의 color는 회색
312     }
313 }
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500

```

```

309 gl.bufferData(gl.ARRAY_BUFFER, new Float32Array([...startPoint, ...tempEndPoint]),
310               gl.STATIC_DRAW);
311 gl.bindVertexArrays(vao);
312 gl.drawArrays(gl.LINES, 0, 2);
313 }
314
315 // axes 그리기
316 axes.draw(mat4.create(), mat4.create()); // 두 개의 identity matrix를 parameter로 전달
317 }
318
319 /* ===== WEEK 04 ===== */
320 // 'gl-matrix-main.js': mat4 함수 제공.
321 // mat4.create, multiply
322 // mat4.lookAt(viewMatrix, COP, Target, Up vector): 파라미터들로부터 view matrix를 계산하여 첫
323 // 번째 parameter인 viewMatrix에 담음
324 // mat4.ortho(projMatrix(output임), left, right, bottom, top, near, far)
325 // mat4.perspective(projMatrix, fov_angle, width_height_ratio, near, far)
326 function getTransformMatrices() {
327     const T = mat4.create();
328     const R = mat4.create();
329     const S = mat4.create();
330
331     mat4.translate(T, T, [0.5, 0.5, 0]); // translation by (0.5, 0.5)
332     mat4.rotate(R, R, rotationAngle, [0, 0, 1]); // rotation about z-axis
333     mat4.scale(S, S, [0.3, 0.3, 1]); // scale by (0.3, 0.3)
334
335     return { T, R, S };
336 }
337
338 function applyTransform(type) {
339     finalTransform = mat4.create();
340     const { T, R, S } = getTransformMatrices();
341
342     const transformOrder = {
343         'TRS': [T, R, S],
344         'TSR': [T, S, R],
345         'RST': [R, T, S],
346         'RST': [R, S, T],
347         'STR': [S, T, R],
348         'SRT': [S, R, T]
349     };
350
351     type은 'TRS', 'TSR', 'RTS', 'RST', 'SRT', 'STR' 중 하나
352     array.forEach(...) : 각 type의 element T or R or S 에 대해 반복
353     */
354     if (transformOrder[type]) {
355         transformOrder[type].forEach(matrix => {
356             mat4.multiply(finalTransform, matrix, finalTransform);
357         });
358     }
359 }
360
361 let lastTime = 0;

```

```

362 function animate(currentTime) {
363
364     if (!lastTime) lastTime = currentTime; // if lastTime == 0
365     // deltaTime: 이전 frame에서부터의 elapsed time (in seconds)
366     const deltaTime = (currentTime - lastTime) / 1000;
367     lastTime = currentTime; // 기록
368
369     if (isAnimating && currentTransformType) {
370         // 2초당 1회전, 즉, 1초당 180도 회전
371         rotationAngle += Math.PI * deltaTime;
372         applyTransform(currentTransformType);
373     }
374     render();
375
376     requestAnimationFrame(animate);
377 }
378
379 /* ===== WEEK 05 ===== */
380 // 01: Viewing
381 // vertex shader에 전달해줄 viewMatrix, projMatrix
382 let viewMatrix = mat4.create();
383 let projMatrix = mat4.create();
384 let startTime;
385
386 function render() {
387     const currentTime = Date.now() // 명시적인 시간 지정
388     const elapsedTime = (currentTime - startTime) / 1000.0;
389
390     gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
391     gl.enable(gl.DEPTH_TEST);
392
393     const modelMatrix = mat4.create();
394
395     mat4.rotateX(modelMatrix, modelMatrix, gl.Matrix.toRadian(3 * elapsedTime));
396     mat4.rotateZ(modelMatrix, modelMatrix, gl.Matrix.toRadian(3 * elapsedTime));
397
398     shader.use();
399     // uniform setter
400     shader.setMat4('u_model', modelMatrix);
401     shader.setMat4('u_view', viewMatrix);
402     shader.setMat4('u_projection', projMatrix);
403     Object.draw(shader);
404
405     requestAnimationFrame(render);
406 }
407
408 async function main() {
409     try {
410         if (!initWebGL()) {
411             throw new Error('WebGL 초기화 실패');
412         }
413
414         await initShader();
415

```

```

416 // View transformation matrix (move the cube to (0, 0, -4))
417 // default camera is at the origin and looking at the infinite in the -z axis
418 // invariant in the program
419 mat4.translate(viewMatrix, viewMatrix, vec3.fromValues(0, 0, -4));
420
421 // Projection transformation matrix (invariant in the program)
422 mat4.perspective(
423     projMatrix,
424     glmatrix.toRadian(60), // field of view (fov, degree)
425     canvas.width / canvas.height, // aspect ratio
426     0.1, // near
427     1000.0 // far
428 );
429
430 // starting time (global variable) for animation
431 startTime = Date.now();
432
433 // call the render function the first time for animation
434 requestAnimationFrame(render);
435
436 return true;
437
438 } catch (error) {
439     console.error('Failed to initialize program:', error);
440     alert('Failed to initialize program');
441     return false;
442 }
443 }
444
445 // 02: Camera Circle
446 const cameraCircleRadius = 5.0;
447 const cameraCircleHeight = 2.0;
448 const cameraCircleSpeed = 90.0;
449
450 function render() {
451     // Viewing transformation matrix
452     let camX = cameraCircleRadius * Math.sin(glMatrix.toRadian(cameraCircleSpeed *
elapsedTime));
453     let camZ = cameraCircleRadius * Math.cos(glMatrix.toRadian(cameraCircleSpeed *
elapsedTime));
454     mat4.lookAt(viewMatrix,
455         vec3.fromValues(camX, cameraCircleHeight, camZ), // camera position
456         vec3.fromValues(0, 0, 0), // look at origin
457         vec3.fromValues(0, 1, 0)); // up vector
458 }

```