**HW3 보고서**

20212020 박민준

# 요구사항

본 수업에서 배포한 예제 프로그램을 기반으로 하여 **재미있고 복잡도가 있는 3차원 뷰잉 과정**을 설계하고, 이를 OpenGL API 함수 (Core Profile)를 사용하여 구현하라.

**1. 요구사항 및 구현 개요**

## (a) - i. 정적 기하 물체 5개 배치

**•** 요구사항:

* 좌표 데이터 폴더 (Data/static\_objects)에 있는 정적 객체 중 5개를 선정하여 장면 위에 배치한다.

**•** 선정한 객체:

1. Bike
2. Ant
3. Dragon
4. Ironman
5. WoodTower

(+ Two Cats)

**•** 구현 방법:

* Static\_Objects.cpp 내 각 클래스(Bike, Ant, Dragon, Ironman, WoodTower, Cat)의 define\_object() 함수에서

- 모델 파일(.geom)을 읽어 VAO/VBO를 초기화 한다.

- instances.emplace\_back()을 통해 장면에 인스턴스를 추가한다.

- ModelMatrix를 사용해 모델의 크기, 회전, 위치를 조정한다.

* Scene\_Definitions.cpp 내 Scene::build\_static\_world() 함수에서 위 객체들을 생성하고, static\_objects 벡터에 순서대로 push\_back한다.

**•** 배치 위치:

1. Bike: (35, 40, 0) 위치, 크기 8.2배, X축 90° 회전
2. Ant: (95, 20, 40) 위치, 크기 3.8배, Z축 90° 회전
3. Dragon: (95, 140, 30) 위치, 크기 0.6배, Y축 15° 회전
4. Ironman: (165, 148, 0) 위치, 크기 3.8배, 여러 축 복합 회전
5. WoodTower: (183, 70, –10) 위치, 크기 5배, X축 90° 회전

(+ Cat1: (180, 70, 32) 위치, 크기 8.2배, X축 90° 회전, Y축 90° 회전)

(+ Cat2: (160, 77, 9) 위치, 크기 8.2배, X축 90° 회전, Y축 -90° 회전)

**•** 특이 사항:

* 기존 프로젝트에 존재하던 table, light, teapot, frame 정적 물체들을 제거하지 않고 그대로 사용하였다.
* 정적 물체 5개 구현 완료 후, WoodTower와 Table 위에 각각 귀여운 고양이를 추가로 배치하였다.

**•** 확인 방법:

1. 프로그램 실행 후 주 카메라 모드(기본)에서 정적 객체가 올바르게 배치되어 wireframe으로 렌더링 되는지 화면을 관찰한다.
2. 카메라 시점 회전을 통해 다섯 객체가 전부 존재함을 확인한다.

**•** 자체 점수: 50/50

## (a) - ii. 동적 기하 물체 2개 배치

**•** 요구사항:

* 좌표 데이터 폴더 (Data/dynamic\_objects)에서 2개를 선택해 배치하고, 애니메이션 루프를 통해 지속적으로 움직이도록 구현한다.

**•** 선정한 객체:

1. Tiger (동적 호랑이)
2. Wolf (동적 늑대)

**•** 구현 방법:

1. Tiger\_D 클래스 (Dynamic\_Objects.cpp)

* define\_object()에서 12프레임 동적 모델을 로드하고, 각 프레임마다 동일한 스케일·재질을 설정한다.
* draw\_object() 내부 DYNAMIC\_OBJECT\_TIGER 케이스에서

- 미리 정의된 waypoints 배열(vector<glm::vec2> waypoints)에 따라 왕복 루프 경로를 생성한다.

- time\_stamp와 speedTiger를 곱해 현재 이동 거리(rawDist)를 계산하고, fmod로 루프 내 상대 거리(curDist)를 결정한다.

- 각 세그먼트 길이를 계산해, curDist가 속한 세그먼트를 찾고, segT를 통해 두 지점 간을 보간한다.

- 보간된 위치(curX, curY) 및 진행 방향(dirX, dirY)을 이용해 angleRad = atan2(dirY, dirX) 을 계산한다.

- ModelMatrix = Translate(curX, curY, 0) → RotateZ(angleRad+보정), 필요 시 추가 회전/스케일을 적용한다.

- Tiger의 “눈 위치”를 로컬 좌표 (0, –17, 13)으로 가정하여 월드 좌표(eyeWorldPos)로 변환한다.

- Tiger-Eye 카메라(scene.camera\_list[camera\_ID\_mapper[CAMERA\_TIGER\_EYE]])의 cam\_view.pos, cam\_view.naxis, cam\_view.vaxis, cam\_view.uaxis 값을 갱신하다.

- 최종적으로 glm::lookAt()을 사용해 ViewMatrix를 재생성한다.

1. Wolf\_D 클래스 (Dynamic\_Objects.cpp)

* define\_object()에서 17프레임 동적 모델을 로드하고, 재질을 설정한다.
* draw\_object() 내부 DYNAMIC\_OBJECT\_WOLF 케이스에서

- Tiger와 유사한 방식으로 wolfWaypoints 배열을 정의하고, 왕복 루프 경로를 생성한다.

- time\_stamp와 speedWolf를 곱해 현재 이동 거리(rawDist) 계산, fmod로 루프 내 curDist 값을 결정한다.

- curDist가 속한 세그먼트를 찾은 뒤 보간을 통해 (curX, curY) 및 방향 벡터를 계산한다.

- ModelMatrix = Translate(curX, curY, 0) → RotateX(90°) → RotateY(angleRad+33°) → Scale(28)을 적용한다.

**•** 확인 방법:

1. 실행 직후 Tiger와 Wolf 객체가 미리 지정된 경로를 따라 반복 이동하는지 시각적으로 확인한다.
2. Tiger‐Eye 카메라 토글 시, 동적 객체의 전방 시점으로 장면이 보이는지 확인한다.

**•** 자체 점수: 100/100

## (b) - i. 주 카메라 이동 기능

**•** 요구사항: 주 카메라는 마우스 또는 키보드 등의 입력 장치에 반응하여 상/하/좌/우/앞/뒤로 움직여야 한다.

**•** 구현 방법 (4.9.2.Our\_House\_V\_0.55\_main.cpp/keyboard 함수):

* W/w: 상단 이동 → cam\_view.pos += (vaxis) \* moveSpeed
* S/s: 하단 이동 → cam\_view.pos += (-vaxis) \* moveSpeed
* A/a: 좌측 이동 → cam\_view.pos += (-uaxis) \* moveSpeed
* D/d: 우측 이동 → cam\_view.pos += (uaxis) \* moveSpeed
* Q/q: 전방 이동 → cam\_view.pos += (-naxis) \* moveSpeed
* E/e: 후방 이동 → cam\_view.pos += (naxis) \* moveSpeed
* 이동 속도(moveSpeed)는 상수 10.0f로 설정

**•** 확인 방법:

1. 키보드 Q/E/A/D/W/S를 누른 상태에서 주 카메라 뷰포트(큰 창)를 관찰하여, 물체가 카메라 반대 방향으로 이동하는지 확인한다.

**•** 자체 점수: 30/30

## (b) - ii. 주 카메라 회전 기능

**•** 요구사항: 주 카메라는 세 축을 중심으로 회전(Pitch, Yaw, Roll)할 수 있어야 한다.

**•** 구현 방법 (4.9.2.Our\_House\_V\_0.55\_main.cpp/keyboard 함수 및 speicalKeyboard함수):

1. 자동 업데이트 변수:

* activeCameraID가 CAMERA\_MAIN일 때, 아래 회전 키가 주 카메라에 적용된다.

1. Roll (Z/X 키) (keyboard):

* Z/z: 시계방향 회전 → axis = naxis, angle = +1°(라디안) → R = rotate(angle, axis) → uaxis, vaxis 갱신
* X/x: 반시계방향 회전 → axis = naxis, angle = –1°(라디안) → uaxis, vaxis 갱신

1. Pitch / Yaw (화살표 키) (specialKeyboard):

* Up Arrow: Pitch (↑) → axis = uaxis, angle = +1°(라디안) → naxis, vaxis 갱신
* Down Arrow: Pitch (↓) → axis = uaxis, angle = –1°(라디안) → naxis, vaxis 갱신
* Left Arrow: Yaw (←) → axis = vaxis, angle = +1°(라디안) → naxis, uaxis 갱신
* Right Arrow: Yaw (→) → axis = vaxis, angle = –1°(라디안) → naxis, uaxis 갱신

1. ViewMatrix 재생성 (keyboard 및 specialKeyboard 하단):

* glm::vec3 eye = cam\_view.pos;
* glm::vec3 center = eye - cam\_view.naxis;
* glm::vec3 up = cam\_view.vaxis;
* cam.ViewMatrix = glm::lookAt(eye, center, up);
* 이후 uaxis, vaxis, naxis, pos를 올바르게 업데이트한다.

**•** 확인 방법:

1. 화살표 키 (위/아래/좌/우)와 Z/X 키를 누르면서 주 카메라 뷰포트가 회전하는 방향이 올바른지 확인한다.

**•** 자체 점수: 30/30

## (b) - iii. 주 카메라 줌(zoom) 기능

**•** 요구사항: 주 카메라 화면을 확대/축소할 수 있어야 한다.

**•** 구현 방법 (4.9.2.Our\_House\_V\_0.55\_main.cpp/mousewheel 함수):

* activeCameraID == CAMERA\_MAIN일 때:

// ======= code example ===========

float& fovy = mainCam.cam\_proj.params.pers.fovy;

if (direction > 0) { fovy = max(fovy - zoomSpeed, 1° radian); }

else { fovy = min(fovy + zoomSpeed, 90° radian); }

mainCam.ProjectionMatrix = glm::perspective(fovy, aspect, n, f);

// ============================

* zoomSpeed는 1° × TO\_RADIAN으로 설정한다.
* 최소 FOV는 1°, 최대 FOV는 90°로 제한한다.

**•** 확인 방법:

1. 마우스 휠을 위/아래로 스크롤하면서 주 카메라 뷰포트가 확대(화면 오브젝트가 가까워짐) · 축소(멀어짐)되는지를 확인한다.

**•** 자체 점수: 20/20

## (b) – iv. 정적 CCTV 카메라 3개 구현

**•** 요구사항:

* 고정된 위치와 방향을 갖는 정적 CCTV 카메라 3개(CAMERA\_CC\_0, CAMERA\_CC\_1, CAMERA\_CC\_2)를 구현한다.

**•** 구현 방법 (Camera.cpp/Perspective\_Camera::define\_camera 함수):

1. CAMERA\_CC\_0

* 위치: (70, 150, 40)
* 방향: center = (100, 130, 30)
* Up: (0,0,1)
* FOV: 70°, 종횡비 1.0, near 1.0, far 2000.0
* ViewMatrix = glm::lookAt(eye, center, up)
* ProjectionMatrix = glm::perspective(70° radian, 1.0, 1.0, 2000.0)
* 뷰포트: x = window\_width – 200, y = window\_height × 7/10, w = 200, h = 200

1. CAMERA\_CC\_1

* 위치: (20, 70, 40)
* 방향: (100, 70, 20)
* FOV: 45°, 종횡비 1.0, near 1.0, far 2000.0
* 뷰포트: x = window\_width – 200, y = window\_height × 4/10, w = 200, h = 200

1. CAMERA\_CC\_2

* 위치: (220, 10, 40)
* 방향: (160, 120, 25)
* FOV: 60°, 종횡비 1.0, near 1.0, far 2000.0
* 뷰포트: x = window\_width – 200, y = window\_height × 1/10, w = 200, h = 200

**•** 특이 사항:

* 정적 CCTV는 flag\_move = false로 설정하여 회전·줌·이동 입력을 받지 않는다.
* showOrtho == false (카메라 모드)일 때에만 렌더링 대상에 포함된다.

**•** 확인 방법:

1. 기본 실행 시 주 카메라 뷰포트 우측에 세 개의 작은 뷰포트가 모두 표시되는지 확인한다.
2. ‘F’ 키를 눌러 ORTHO 모드로 전환하면 세 개 CCTV 뷰포트가 사라지고, 다시 ‘F’를 누르면 복원되는지 확인한다.

**•** 자체 점수: 45/45

## (b) – v. 동적 CCTV 카메라 1개 구현

**•** 요구사항:

* 고정된 위치를 갖되, 사용자가 카메라 방향(Pitch/Yaw/Roll)과 FOV(Zoom)만 조절할 수 있는 동적 CCTV(CAMERA\_DYN\_CCTV)를 구현한다.

**•** 구현 방법 (Camera.cpp/Perspective\_Camera::define\_camera 함수):

1. CAMERA\_DYN\_CCTV 초기화

* 위치: (95, 15, 45)
* 초기 방향: center = (55, 40, 0)
* Up: (0,0,1)
* 초기 FOV: 45°
* 종횡비: win\_aspect\_ratio
* near: 1.0, far: 5000.0
* ProjectionMatrix = glm::perspective(45° radian, aspect, 1.0, 5000.0)
* 뷰포트: x = 0, y = window\_height × 7/10, w = window\_width/4, h = window\_height/4
* is\_dynamic = true, flag\_move = false (위치 고정)

1. 입력 처리 (keyboard, specialKeyboard, mouseWheel):

* Pitch/Yaw/Roll: 주 카메라와 동일하되, 회전 속도를 0.5° × TO\_RADIAN으로 조절한다.
* Zoom: 마우스 휠로 FOV 조절, fovy\_d도 최소 1°, 최대 90°로 제한한다.
* Translation: W/S/A/D/Q/E 입력 시 무시한다.

1. ViewMatrix 갱신 (keyboard/specialKeyboard 하단):

* glm::vec3 eye = cam\_view.pos; // 위치 고정
* glm::vec3 center = eye - cam\_view.naxis; // 전방 방향에 따라 갱신
* glm::vec3 up = cam\_view.vaxis;
* cam.ViewMatrix = glm::lookAt(eye, center, up);
* 이후 uaxis, vaxis, naxis 값을 재계산한다. (pos는 고정)

**•** 확인 방법:

1. 숫자 키 ‘2’를 눌러 활성 카메라를 “DYNAMIC CCTV”로 전환한다.
2. Arrow Keys, Z/X 키로 회전 여부를 확인한다.
3. 마우스 휠로 Zoom In/Out을 확인한다.
4. W/S/A/D/Q/E 키 입력 시 동적 CCTV가 이동하지 않는지 확인한다.

**•** 자체 점수: 30/30

## (b) – vi. 직교 투영 카메라 3개(정면·측면·상면도) 구현

**•** 요구사항:

* 정면도(CAMERA\_SIDE\_FRONT), 측면도(CAMERA\_SIDE), 상면도(CAMERA\_TOP)를 직교 투영으로 구현하고, 고정된 뷰포트에 표시한다.

**•** 구현 방법 (Camera.cpp/Orthographic\_Camera::define\_camera 함수):

1. 정면도 (CAMERA\_SIDE\_FRONT)

* 위치: eye = (125, –300, 25) (Y축 앞)
* center: (125, 80, 25) (물체 중심)
* up: (0,0,1)
* world AABB: [(0,0,0), (240,160,100)]
* AABB의 8개 꼭짓점을 ViewMatrix로 변환하여 화면좌표계 상의 xmin, xmax, ymin, ymax 계산
* Orthographic 매개변수:

left = xmin

right = xmax

bottom = ymin

top = ymax

near = 1.0

far = 1000.0

* ProjectionMatrix = glm::ortho(xmin, xmax, ymin, ymax, 1.0, 1000.0)
* 뷰포트: x = 0, y = window\_height × 7/10, w = window\_width/4, h = window\_height/4

1. 상면도 (CAMERA\_TOP)

* 위치: eye = (125, 80, 500) (높은 Z축)
* center: (125, 80, 25) (물체 중심)
* up: (0,1,0) (Y축을 위쪽으로)
* world AABB: [(0,0,0), (240,200,200)]
* 동일하게 xmin, xmax, ymin, ymax 계산 후 Orthographic 매개변수를 설정한다.
* ProjectionMatrix = glm::ortho(xmin, xmax, ymin, ymax, 1.0, 1000.0)
* 뷰포트: x = 0, y = window\_height × 4/10, w = window\_width/4, h = window\_height/4

1. 측면도 (CAMERA\_SIDE)

* 위치: eye = (300, 60, 25) (X축 옆)
* center: (125, 60, 25) (물체 중심)
* up: (0,0,1)
* world AABB: [(0,0,0), (240,200,100)]
* xmin, xmax, ymin, ymax 계산 후 Orthographic 매개변수를 설정한다.
* ProjectionMatrix = glm::ortho(xmin, xmax, ymin, ymax, 1.0, 1000.0)
* 뷰포트: x = 0, y = window\_height × 1/10, w = window\_width/4, h = window\_height/4

1. 세 카메라 공통 특징

* flag\_valid = true, flag\_move = false로 설정한다.
* showOrtho == true 모드에서만 렌더링 대상에 포함된다.
* Zoom/Rotation/이동 입력을 받지 않는다.

**•** 확인 방법:

1. ‘F’ 키를 눌러 ORTHO 모드로 전환한다.
2. 주 카메라 뷰포트(큰 창)와 동시에 좌측 세 군데 작은 뷰포트(정면·상면·측면)가 나타나는지 확인한다.
3. ‘F’ 키를 다시 눌러 CAMERA+CCTV 모드로 전환 시, 세 직교 카메라 뷰포트가 사라지는지 확인한다.

**•** 자체 점수: 45/45

## (b) – vii. 카메라 프레임(RGB 좌표축) 토글

**•** 요구사항:

* 직교 투영 카메라(정면도/측면도/상면도)를 제외한 모든 카메라(주 카메라, 정적 CCTV 0/1/2, 동적 CCTV, Tiger‐Eye)의 위치·방향을 시각적으로 표시하는 좌표축(Origin at camera position, R= X축, G= Y축, B= Z축)을 생성·해제하는 기능을 구현한다.

**•** 구현 방법 (4.9.2.Our\_House\_V\_0.55\_main.cpp/display 함수 하단, keyboard 함수):

1. 토글 플래그: 전역 변수 static bool showCameraFrames = false;
2. 키 입력 (keyboard):

else if (key == 't' || key == 'T') {

showCameraFrames = !showCameraFrames;

printf(">>> Camera Frames %s\n", showCameraFrames ? "ON" : "OFF");

glutPostRedisplay();

return;

}

1. 렌더링 코드 (display 함수):

* 모든 카메라별 뷰포트를 렌더링한 후(scene.draw\_world()), showCameraFrames == true이면 다음을 수행한다:

1. glDisable(GL\_DEPTH\_TEST)
2. 메인 뷰포트 (현재 cam.camera\_id == CAMERA\_MAIN인 카메라):

- 다른 카메라 리스트를 순회하며,

• flag\_valid == true이고

• camera\_id != CAMERA\_MAIN 이며

• (CAMERA\_SIDE\_FRONT, CAMERA\_TOP, CAMERA\_SIDE)가 아닌 카메라만 선택한다.

- scene.draw\_camera\_frame(other)를 호출한다.

1. 서브 뷰포트 (메인 카메라가 아닌 경우):

- 현재 그려진 카메라(cam)가 아닌 다른 카메라(list 순회) 중, 정면·측면·상면도를 제외한 카메라를 그린다.

- scene.draw\_camera\_frame(other)를 호출한다.

1. glEnable(GL\_DEPTH\_TEST)를 호출하여 Depth Test를 다시 켠다.
2. draw\_camera\_frame(const Camera& cam) (Scene\_Definitions.cpp):

* 모델 행렬:

1. Translate to cam.cam\_view.pos
2. Rotation 행렬 R:

R[0] = (uaxis.x, uaxis.y, uaxis.z, 0)

R[1] = (vaxis.x, vaxis.y, vaxis.z, 0)

R[2] = (naxis.x, naxis.y, naxis.z, 0)

R[3] = (0, 0, 0, 1)

1. Scale by axisLength = 20
2. MVP = ProjectionMatrix \* ViewMatrix \* ModelMatrix

* 셰이더 호출: shader\_simple\_ptr->loc\_ModelViewProjectionMatrix에 MVP를 전달한다.
* VAO(axis\_object.VAO) 바인딩 후, RGB 좌표축 선분을 그린다.

**•** 특이 사항:

* Tiger-Eye 카메라 구현 후, 해당 카메라에도 프레임(RGB 좌표축)을 추가하였다.

**•** 확인 방법:

1. ‘T’ 키를 눌러 “Camera Frames ON” 상태로 전환한다.
2. 주 뷰포트나 서브 뷰포트 상에서 카메라 위치에 작은 RGB 좌표축이 나타나는지 확인한다.
3. ‘T’ 키를 다시 눌러 “Camera Frames OFF” 시, 좌표축이 모두 사라지는지 확인한다.

**•** 자체 점수: 30/30

## (c) - 추가 구현 사항. Tiger-Eye 카메라 구현

**•** 구현 방법:

1. 카메라 ID 및 인스턴스 추가

* Camera\_ID 열거형에 CAMERA\_TIGER\_EYE가 이미 정의되어 있다.
* Camera\_Data 구조체에 Perspective\_Camera cam\_tiger\_eye{ CAMERA\_TIGER\_EYE };가 선언되어 있으며, Scene::create\_camera\_list()에서 다음과 같이 초기화하여 camera\_list에 추가한다.

camera\_data.cam\_tiger\_eye.define\_camera(win\_width, win\_height, win\_aspect\_ratio);

camera\_ID\_mapper[CAMERA\_TIGER\_EYE] = camera\_list.size();

camera\_list.push\_back(camera\_data.cam\_tiger\_eye);

1. 초기 설정 (Perspective\_Camera::define\_camera)

* CAMERA\_TIGER\_EYE 분기에서 초기 ViewMatrix와 ProjectionMatrix, 뷰포트를 설정한다.

1. Tiger 객체 애니메이션 로직

* Tiger\_D::define\_object()에서 12프레임 애니메이션 모델을 로드하고, 각 프레임마다 동일한 스케일·재질을 설정한다.
* Tiger\_D::draw\_object()의 DYNAMIC\_OBJECT\_TIGER 케이스에서

1. 웨이포인트(waypoints)를 초기화한다.
2. 현재 경로 위치를 계산한다.
3. 보간 위치 및 방향 벡터를 설정한다.
4. ModelMatrix를 생성한다.
5. Tiger‐Eye 카메라 뷰를 업데이트한다.
6. ProjectionMatrix: reshape 단계에서 이미 (45° radian, aspect, near=1, far=500)으로 설정되어 있으므로 여기서는 변경하지 않는다.
7. 키 동작 및 뷰포트 배치

* 키 입력 토글 코드를 작성한다. (Our\_House\_V\_0.55\_main.cpp keyboard 함수)

else if (key == '3' || key == '3') {

showTigerEye = !showTigerEye;

printf(">>> Tiger Eye Camera %s\n", showTigerEye ? "ON" : "OFF");

glutPostRedisplay();

return;

}

* display() 함수 내 Tiger‐Eye를 렌더링한다.
* 뷰포트를 설정한다. (reshape 함수)

**•** 확인 방법:

1. 프로그램 실행 후, 기본 모드에서 Tiger 객체가 경로에 따라 애니메이션 되는지 확인한다.
2. 숫자 키 ‘3’을 눌러 “Tiger Eye Camera ON” 상태로 전환한다.
3. 화면 왼쪽 아래에 작은 1/4 크기 뷰 포트가 나타나고, Tiger의 눈 위치 시점을 wireframe으로 렌더링한다.
4. Tiger가 이동 또는 회전할 때마다 Tiger‐Eye 뷰포트에서 실시간으로 시점이 변화하는지 확인한다.
5. 다시 숫자 키 ‘3’을 눌러 “Tiger Eye Camera OFF” 상태로 전환 후 뷰 포트가 사라지는지 확인한다.

**2. 프로그램 사용법**

**• ‘1’ → 주 카메라 작동 활성화(기본).**

**• ‘2’ → 동적 CCTV 작동 활성화.**

**• ‘3’ → Tiger-Eye Camera ON/OFF.**

**• ‘F’ → 시점 모드 토글 (Main Camera + Dynamic/Static CCTV ⇔ ORTHO(Main Camera + 정면도/측면도/상면도)).**

**• ‘T’ → 카메라 프레임(RGB 좌표축) 표시/해제.**

**• ‘C’ → 면 컬링(Cull Face) 모드 토글.**

**• ‘V’ → 폴리곤 렌더 모드(Polygon Mode) 토글.**

**• ‘B’ → 깊이 테스트(Depth Test) 토글.**

**• W/A/S/D/Q/E → (주 카메라) 이동, (다른 카메라) 이동 불가.**

**• 화살표 키, Z/X → (주 카메라 & 동적 CCTV) 회전, (Tiger -Eye) 무시.**

**• 마우스 휠 → (주 카메라 & 동적 CCTV) Zoom In/Out, (나머지) 무시.**

**• ESC → 프로그램 종료.**