Camera

-OpenGL에서 카메라의 이동은 Scene의 모든 오브젝트들을 반대방향으로 이동시킴으로 카메라가 움직이는 효과를 불러일으킴

Camera/View space

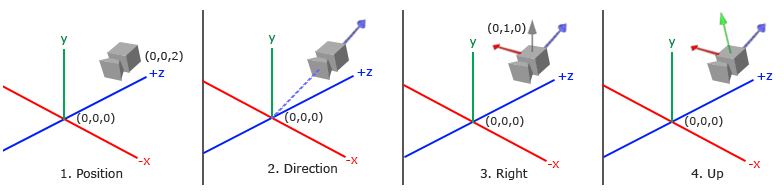
-카메라를 정의하기 위해서 필요한 내용

-카메라의 위치

-카메라가 보는 방향

-카메라의 오른쪽을 가리키는 벡터

-카메라의 위쪽을 가리키는 벡터

-4개의 요소가 모여서 하나의 작은 좌표계를 만드는 것

1. Camera Position

-카메라의 위치는 world space의 벡터임

|  |
| --- |
| glm::vec3 cameraPos = glm::vec3(0.0f, 0.0f, 3.0f); |

2. Camera Direction

-카메라가 scene의 원점 (0, 0, 0)을 가리키게 하기

-Camera Position – Target Position(0, 0, 0)

|  |
| --- |
| glm::vec3 cameraTarget = glm::vec3(0.0f, 0.0f, 0.0f);  glm::vec3 cameraDirection = glm::normalize(cameraPos - cameraTarget); |

-실제로 카메라가 보는 방향과는 반대방향을 가리키고 있음

3. Right

-카메라의 오른쪽을 나타내기 위해서 World Space의 위쪽을 가리키는 벡터와 카메라방향을 외적하면, 두 벡터에 수직인 벡터가 나옴으로 오른쪽을 가리키는 벡터를 얻

|  |
| --- |
| glm::vec3 up = glm::vec3(0.0f, 1.0f, 0.0f);  glm::vec3 cameraRight = glm::normalize(glm::cross(up, cameraDirection)); |

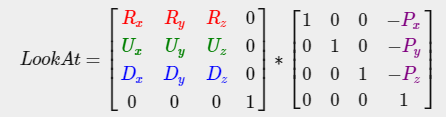
4. UP

-카메라 방향, 카메라의 오른쪽을 외적하면 위쪽 방향벡터를 얻을 수 있음

|  |
| --- |
| glm::vec3 cameraUp = glm::cross(cameraDirection, cameraRight); |

Look At

-3개의 직각인 축을 사용해서 좌표 space를 만들면, 이동 벡터를 만들어 좌표 space를 변환할 수 있음

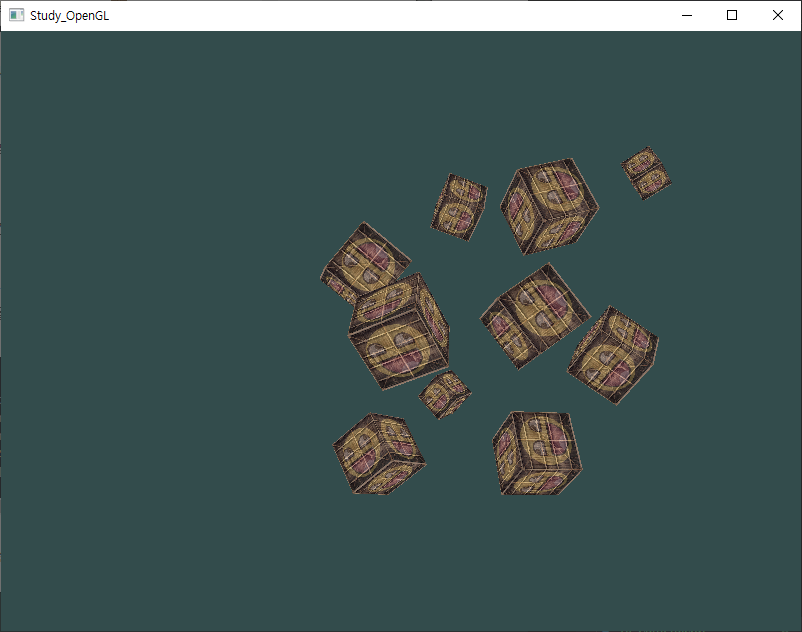
-R : 오른쪽벡터 U : 위쪽벡터 D : 방향벡터, P : 카메라 위치 벡터

-View행렬로 사용할 수 있는 LookAt행렬 만들기

|  |
| --- |
| glm::mat4 view;  view = glm::lookAt(glm::vec3(0.0f, 0.0f, 3.0f), //카메라 위치  glm::vec3(0.0f, 0.0f, 0.0f), //타겟 위치  glm::vec3(0.0f, 1.0f, 0.0f)); //world space 위쪽 벡터 |

-scene주위를 카메라가 빙빙 도는것을 만들어보자, target은 (0, 0, 0)

|  |
| --- |
| float radius = 10.0f;  float camX = sin(glfwGetTime()) \* radius;  float camZ = cos(glfwGetTime()) \* radius;  glm::mat4 view;  view = glm::lookAt(glm::vec3(camX, 0.0, camZ), glm::vec3(0.0, 0.0, 0.0), glm::vec3(0.0, 1.0, 0.0)); |



Walk Around

-카메라가 알아서 돌아가는 것이 아닌, 직접 움직임을 제어하는 방법

-일단 기본 카메라 위치 설정

|  |
| --- |
| glm::vec3 cameraPos =glm::vec3(0.0f, 0.0f, 3.0f);  glm::vec3 cameraFront = glm::vec3(0.0f, 0.0f, -1.0f);  glm::vec3 cameraUp = glm::vec3(0.0f, 1.0f, 0.0f); |

-LookAt함수

|  |
| --- |
| view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp); |

-키 입력 함수에 W, A, S, D를 추가

|  |
| --- |
| void processInput(GLFWwindow \*window)  {  ...  float cameraSpeed = 0.05f;  if (glfwGetKey(window, GLFW\_KEY\_W) == GLFW\_PRESS)  cameraPos += cameraSpeed \* cameraFront;  if (glfwGetKey(window, GLFW\_KEY\_S) == GLFW\_PRESS)  cameraPos -= cameraSpeed \* cameraFront;  if (glfwGetKey(window, GLFW\_KEY\_A) == GLFW\_PRESS)  cameraPos -= glm::normalize(glm::cross(cameraFront, cameraUp)) \* cameraSpeed;  if (glfwGetKey(window, GLFW\_KEY\_D) == GLFW\_PRESS)  cameraPos += glm::normalize(glm::cross(cameraFront, cameraUp)) \* cameraSpeed;  } |

-오른쪽 벡터를 사용할 때, 정규화를 해줘야함, 그렇지 않으면 외적에 의해 크기가 바뀜으로 원하는 속도로 이동하지 않을 수 있기 때문임

-cameraSpeed로 카메라의 이동속도를 바꿀 수 있음

Movement Speed

-위에 이동속도는 상수로 설정했으나, 컴퓨터의 성능에 따라 프로세싱 속도가 다르기 때문에 컴퓨터마다 이동속도가 달라짐, 이를 동일하게 바꾸기 위해 설정하는 것

-deltaTime변수를 사용(마지막 프레임을 렌더링하는데 걸리는 시간을 저장)

-모든 속도들에 deltaTime을 곱하면 렌더링이 오래 걸리는 컴퓨터는 높은 숫자, 짧게 걸리는 컴퓨터는 낮은 숫자들을 가짐, 즉 동일한 속도를 가지게 됨

-deltaTime, lastFrame 전역변수 기록

|  |
| --- |
| float deltaTime = 0.0f;  float lastFrame = 0.0f; |

-각 프레임마다 새로운 deltaTime 계산

|  |
| --- |
| float currentFrame = glfwGetTime();  deltaTime = currentFrame - lastFrame;  lastFrame = currentFrame; |

-속도 변경

|  |
| --- |
| void processInput(GLFWwindow \*window)  {  float cameraSpeed = 2.5f \* deltaTime;  ...  } |

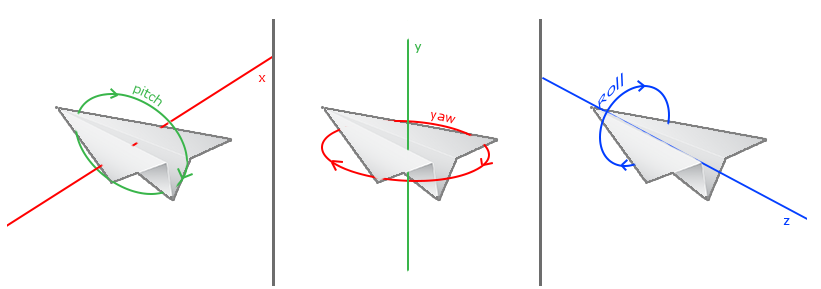
Look Around

-마우스에 의한 시점변화 추가하기!

-scene을 둘러볼려면 마우스 입력에 따라 cameraFront벡터를 수정해야함

-이에 관한 벡터 수정은 삼각법을 알고 있어야함

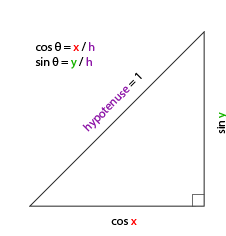
Euler Angles 오일러 각

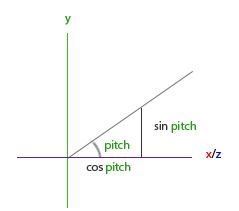
-카메라 시스템에는 roll값이 필요하지 않음

-yaw, pitch값을 방향벡터로 변환하는 방법

-아래 변의 길이가 cos x/h = cos x/1 = cos x

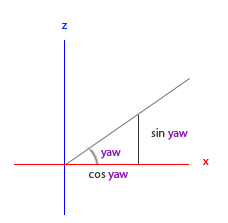
오른 변의 길이는 sin y/h = sin y/1 = sin y





-pitch값 계산

|  |
| --- |
| direction.y = sin(glm::radians(pitch));  direction.x = cos(glm::radians(pitch));  direction.z = cos(glm::radians(pitch)); |

-yaw값 추가

|  |
| --- |
| direction.x = cos(glm::radians(pitch)) \* cos(glm::radians(yaw));  direction.y = sin(glm::radians(pitch));  direction.z = cos(glm::radians(pitch)) \* sin(glm::radians(yaw)); |

-이 값이 주변을 둘러볼 수 있게 해주는 3차원 방향벡터로 바꿔주는 공식임

Mouse Input

-좌우 마우스 움직임은 yaw에 영향

-상하 마우스 움직임은 pitch에 영향

-마지막 프레임의 위치를 저장, 현재 프레임에서 움직임 정도를 계산해서 얼마나 움직였는지 값을 계산하는 것

-마우스를 window안에 한정하고, 커서를 안보이게 하는 코드

|  |
| --- |
| glfwSetInputMode(window, GLFW\_CURSOR, GLFW\_CURSOR\_DISABLED); |

-마우스 이벤트 콜백함수

|  |
| --- |
| void mouse\_callback(GLFWwindow\* window, double xpos, double ypos); |

-GLFW에 콜백 함수를 등록

|  |
| --- |
| glfwSetCursorPosCallback(window, mouse\_callback); |

-FPS스타일의 카메라 입력을 위한 단계

1. 마지막 프레임으로 부터 마우스 offset 계산

2. yaw, pitch값에 offset 더하기

3. pitch값에 최댓값/최솟값 설정

4. 방향 벡터를 계산

1. offset 계산

-마우스의 처음 위치는 화면크기/2

|  |
| --- |
| float lastX = 400, lastY = 300; //x800, y600 |

-콜백 함수에서 offset 계산

|  |
| --- |
| float xoffset = xpos - lastX;  float yoffset = lastY - ypos; // y 좌표의 범위는 밑에서부터 위로가기 때문에 반대로  lastX = xpos;  lastY = ypos;  float sensitivity = 0.05f;  xoffset \*= sensitivity;  yoffset \*= sensitivity; |

2. yaw, pitch값에 더하기

|  |
| --- |
| yaw += xoffset;  pitch += yoffset; |

3. 최대 최솟값 설정 (사용자의 이상한 움직임, 혹은 버그를 예방함)

-위로 89도 이상으로 바라보기, 아래 –89도 이하로 바라보기를 막음

|  |
| --- |
| if(pitch > 89.0f)  pitch = 89.0f;  if(pitch < -89.0f)  pitch = -89.0f; |

4. 방향 벡터 계산

|  |
| --- |
| glm::vec3 front;  front.x = cos(glm::radians(pitch)) \* cos(glm::radians(yaw));  front.y = sin(glm::radians(pitch));  front.z = cos(glm::radians(pitch)) \* sin(glm::radians(yaw));  cameraFront = glm::normalize(front); |

-바로 코드 실행하면 마우스 커서가 포커스를 윈도우로 받을 때 마다 크게 움직임을 볼 수 있음, 윈도우로 들어갈 때의 마우스 위치가 xpos, ypos로 들어가기 때문에 일어나는 현상

-이를 막기 위해 bool변수를 선언해 막음

|  |
| --- |
| if(firstMouse) //firstMouse는 TRUE로 초기화 되어있어야함  {  lastX = xpos;  lastY = ypos;  firstMouse = false;  } |

Zoom 확대 축소

-fov(Field of View)가 작아지면 scene projected space는 작아짐, zoom in되는 효과

-마우스 스크롤 콜백함수

|  |
| --- |
| void scroll\_callback(GLFWwindow\* window, double xoffset, double yoffset)  {  if(fov >= 1.0f && fov <= 45.0f)  fov -= yoffset;  if(fov <= 1.0f)  fov = 1.0f;  if(fov >= 45.0f)  fov = 45.0f;  } |

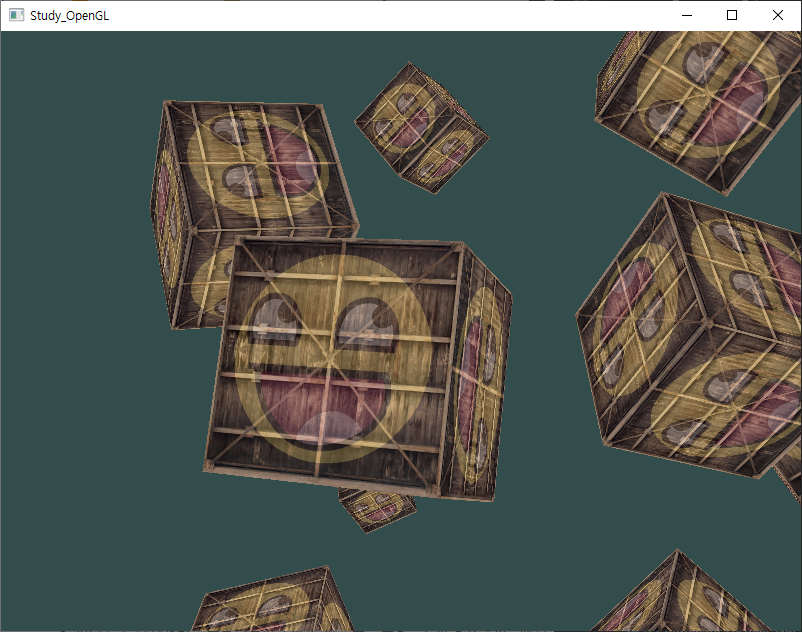
-zoom level을 1.0f ~ 4.5f로 제한

-루프마다 perspective projection 행렬을 알려줘야함

|  |
| --- |
| projection = glm::perspective(glm::radians(fov), 800.0f / 600.0f, 0.1f, 100.0f); |

-스크롤 콜백함수 등록

|  |
| --- |
| glfwSetScrollCallback(window, scroll\_callback); |

-이동, 확대, 보는 방향 변경이 작동하는 것을 확인할 수 있음

-카메라 관련 코드가 길기 때문에, 다른 파일로 빼서 사용(shader.h)와 같이 변경하여 사용