

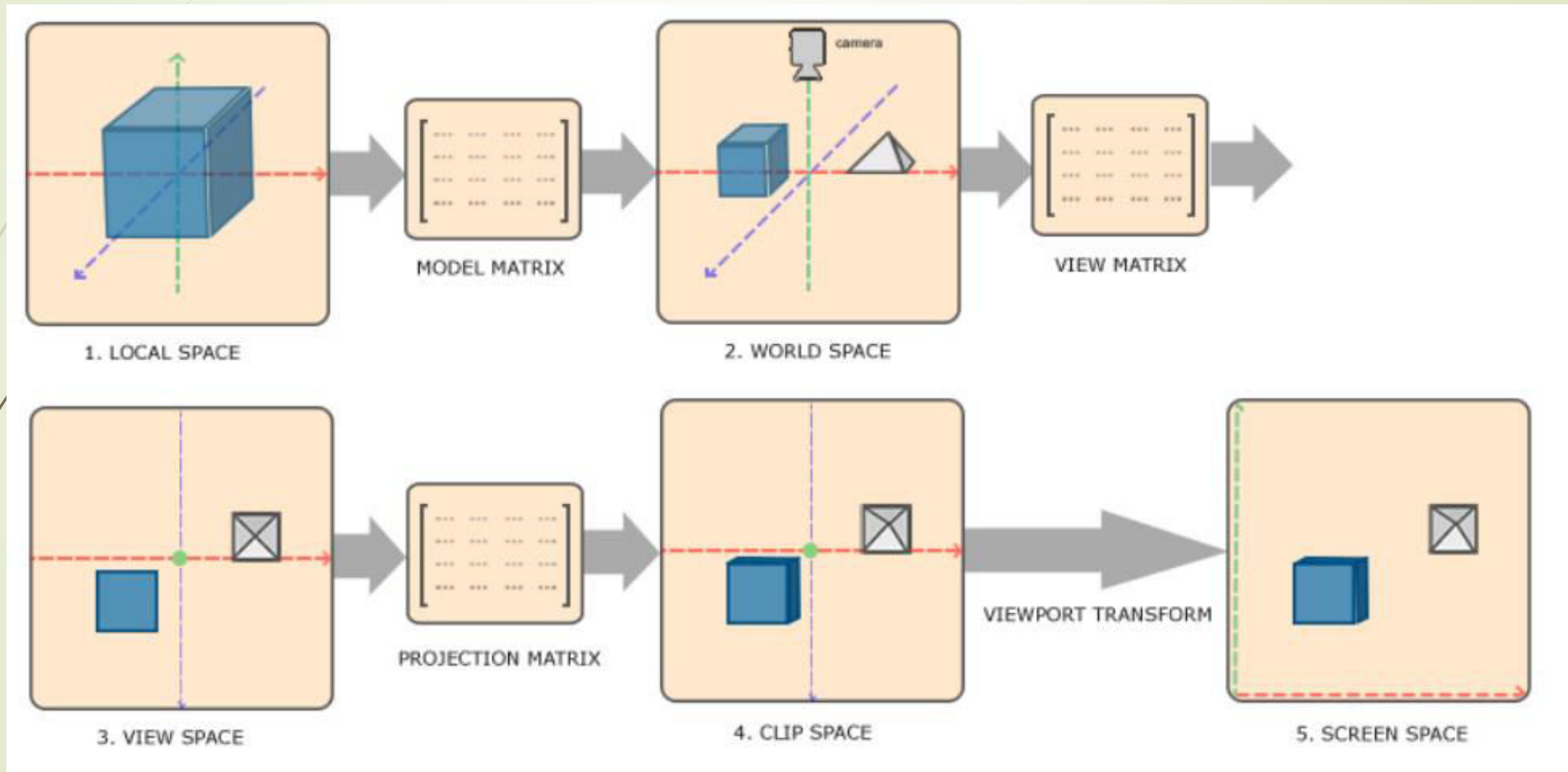
Model, View, Projection. Камера

Курс з 3D-програмування «Від трикутника до сцени».

Рекомендовані джерела:

1. learnopengl. Урок 1.8 — Системы координат
<https://habr.com/post/324968/>
2. learnopengl. Урок 1.8 – Камера <https://habr.com/post/327604/>
3. Tutorial 3 : Matrices <http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/>
4. Tutorial 6 : Keyboard and Mouse <http://www.opengl-tutorial.org/beginners-tutorials/tutorial-6-keyboard-and-mouse/>
5. Уроки по OpenGL с сайта OGLDev
<https://triplepointfive.github.io/ogltutor/>
6. OpenGL Camera http://www.songho.ca/opengl/gl_camera.html#lookat
7. Шпаргалка по 3D трансформациям средствами GLM: ps-group.github.io/opengl/glm_cheatsheet

Системы координат

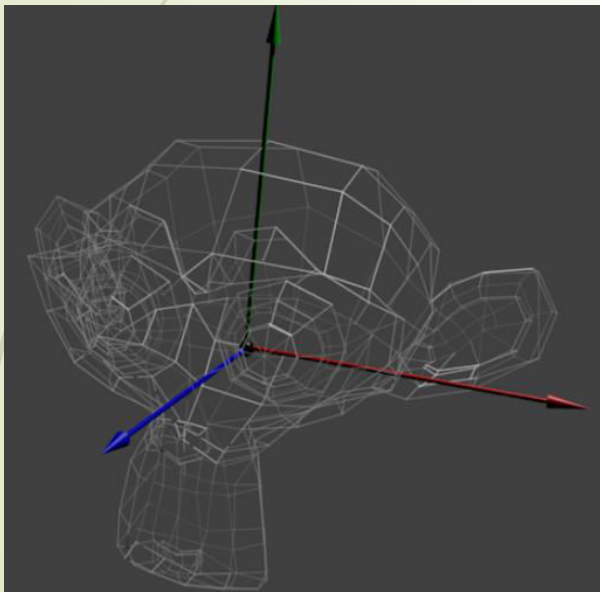


Системи координат

1. **Локальні координати** – координати об'єкта в межах від -1 до 1 відносно точки відліку (0,0). Матриця моделі.
2. **Світові координати** : перетворення локальних координат у координати ігрового світу, змінюються відносно глобальної точки відліку – **ЄДИНОЇ ДЛЯ ВСІХ ОБ'ЄКТІВ**.
3. **Координати камери**: світові координати трансформуються відносно точки огляду спостерігача (камери) у простір камери.
4. **Координати відсікання**: діють в діапазоні від -1 до 1 і визначають, які об'єкти потрапляють на екран.
5. **Екранні координати**: перетворення координат відсікання в область екранних координат, заданих **glViewport**.

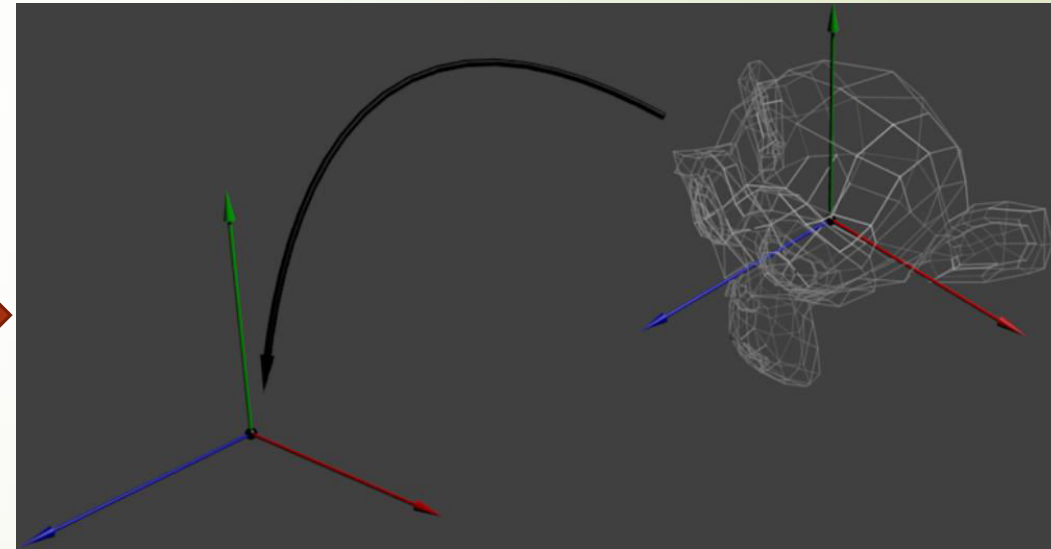
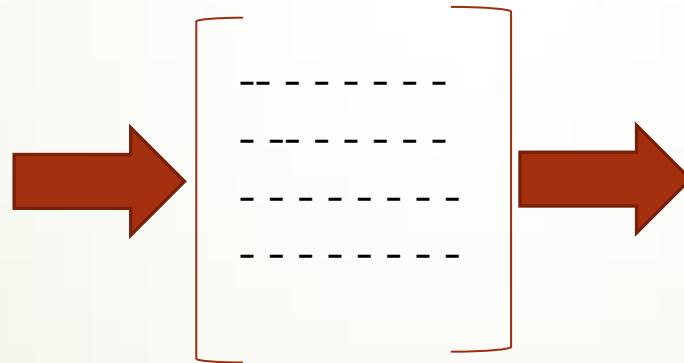
Матриця моделі (Model)

- Локальні координати -> Матриця Моделі-> Світові координати



Локальний простір

`translation*rotation*scale;`

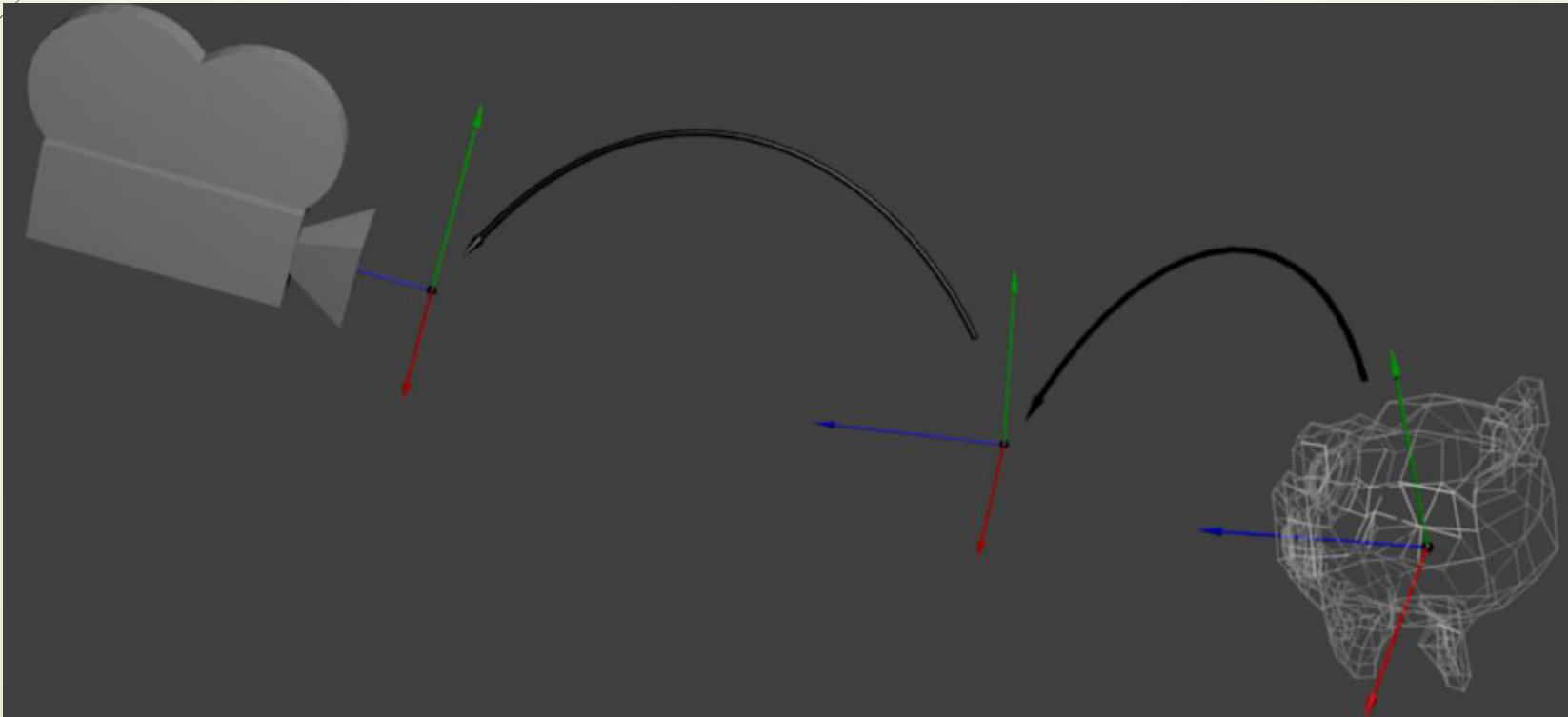


Світовий простір
(Сцена)

Матриця вигляду (View)

- Світові координати -> Матриця Вигляду-> Координати відносно камери

```
glm::mat4 ViewMatrix = glm::translate(glm::mat4(), glm::vec3(-3.0f, 0.0f, 0.0f));
```

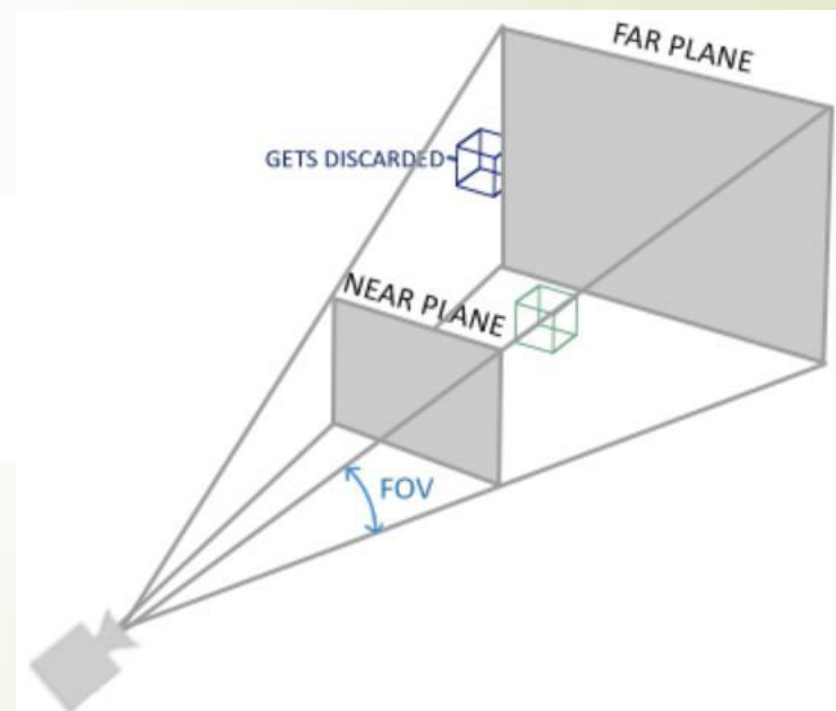


Перспектива. Простір відсікання

- За допомогою камери можна змінювати положення по x та y .
- Z – відстань до камери. Задається за допомогою перспективи та **матриці проєкції**.



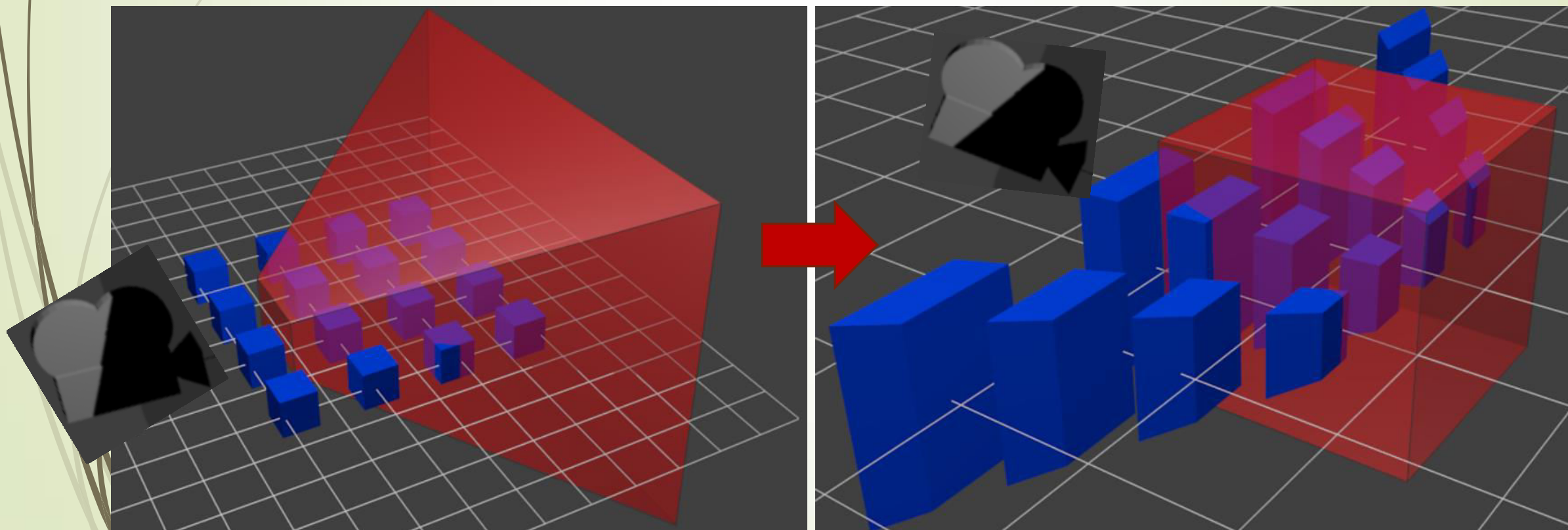
$$out = \begin{pmatrix} x/w \\ y/w \\ z/w \end{pmatrix}$$



Матриця проєкції

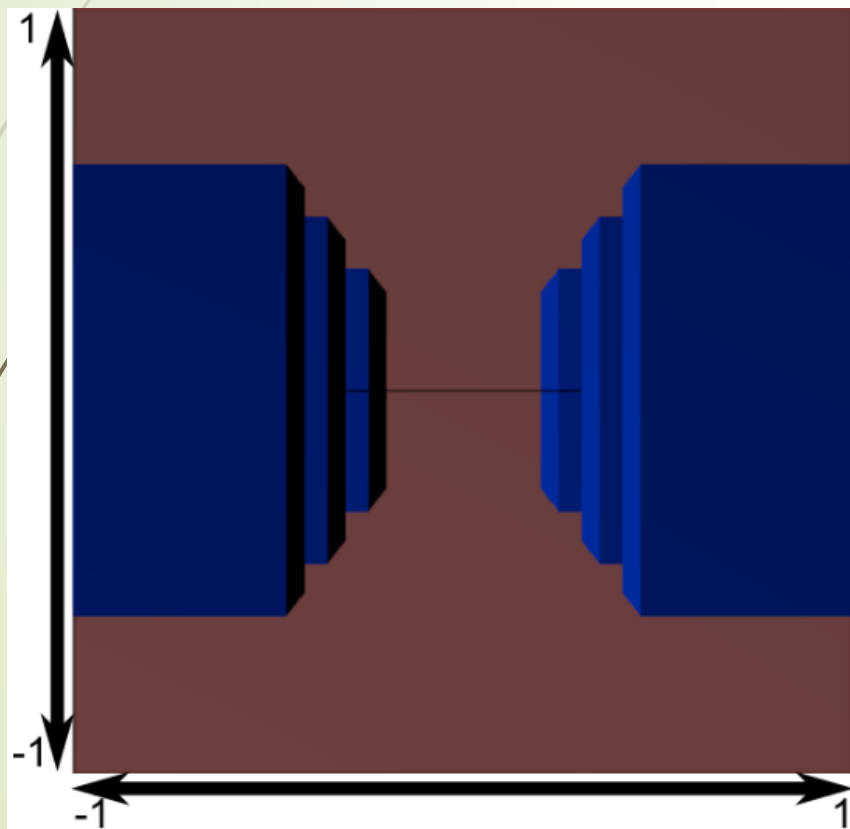
- Координати камери -> Матриця проєкції -> Гомогенні координати

```
glm::mat4 proj = glm::perspective( 45.0f, (float)width/(float)height, 0.1f, 100.0f);
```

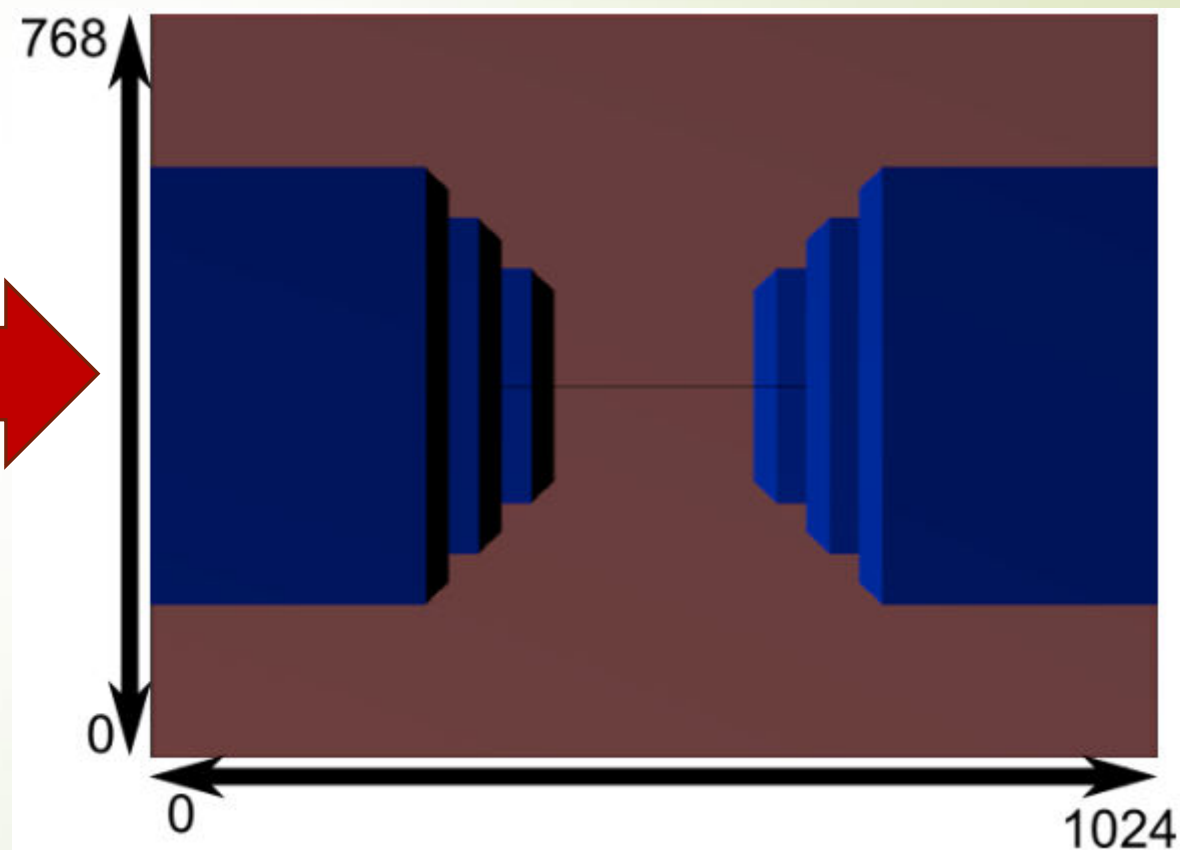


Екранний простір

Простір відсікання



Екранний простір



Обчислення трансформацій MVP

- У програмі на C++ в ігровому циклі створюємо матрицю трансформації:

```
// C++ : compute the matrix  
glm::mat4 MVPmatrix = projection * view * model; // Remember : inverted !
```

- У вершинному шейдері:

```
// Input vertex data, different for all executions of this shader.  
layout(location = 0) in vec3 vertexPosition_modelspace;  
  
// Values that stay constant for the whole mesh.  
uniform mat4 MVP;  
  
void main(){  
    // Output position of the vertex, in clip space : MVP * position  
    gl_Position = MVP * vec4(vertexPosition_modelspace,1);  
}
```

MVP у програмі

- 1. Матриця проєкції (порядок створення матриць неважливий):

```
// Projection matrix : 45° Field of View, 4:3 ratio, display range : 0.1 unit <-> 100 units  
glm::mat4 Projection = glm::perspective(glm::radians(45.0f), (float) width / (float) height, 0.1f, 100.0f);
```

- 2. Матриця вигляду

```
// Camera matrix  
glm::mat4 View = glm::lookAt(  
    glm::vec3(4,3,3), // Camera is at (4,3,3), in World Space  
    glm::vec3(0,0,0), // and looks at the origin  
    glm::vec3(0,1,0) // Head is up (set to 0,-1,0 to look upside-down)  
);
```

- 3. Матриця моделі

```
// Model matrix : an identity matrix (model will be at the origin)  
glm::mat4 Model = glm::mat4(1.0f);  
// Our ModelViewProjection : multiplication of our 3 matrices  
glm::mat4 mvp = Projection * View * Model; // Remember, matrix multiplication is the other way around
```

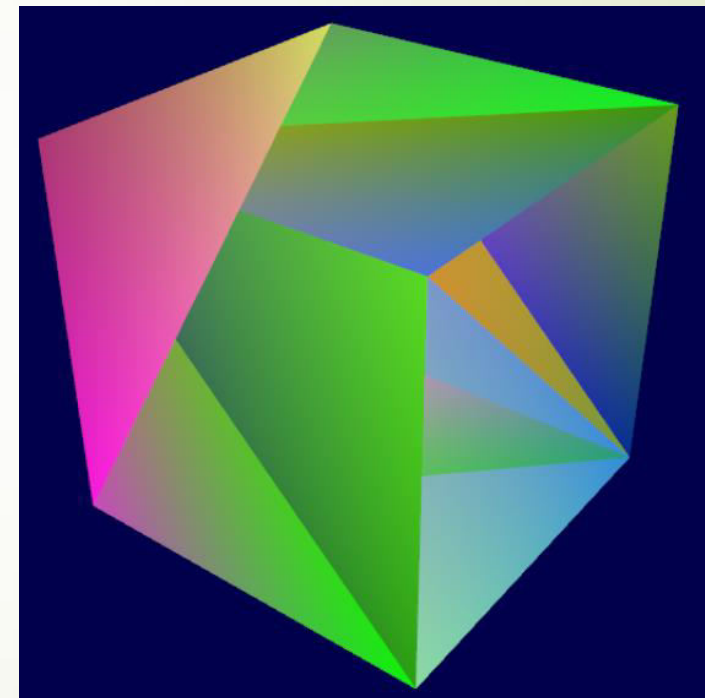
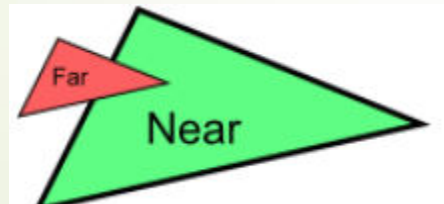
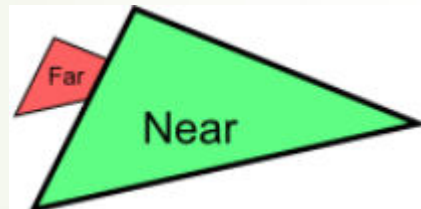
MVP у програмі

- 4. В ігровому циклі визначаємо положення uniform-змінної MVP і передаємо вказівник на матрицю трансформації до вершинного шейдера:

```
// Get a handle for our "MVP" uniform  
// Only during the initialisation  
GLuint MatrixID = glGetUniformLocation(programID, "MVP");  
  
// Send our transformation to the currently bound shader, in the "MVP" uniform  
// This is done in the main loop since each model will have a different MVP matrix  
glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &mvp[0][0]);
```

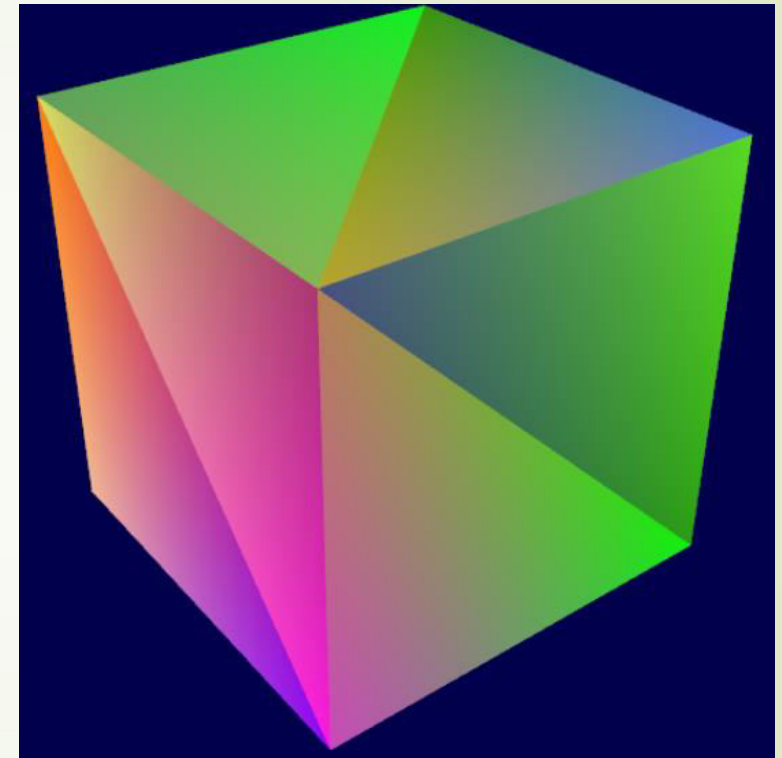
3D-фігури

- Координати усіх вершин можна взяти в тюторіалі
- Результат не зовсім очікуваний...



Z-буфер

- В ньому OpenGL зберігає інформацію про глибину
- GLFW створює цей буфер автоматично
- Глибина зберігається для кожного фрагмента (z-значення)
- При рендерингу виконується перевірка глибини
- Для активації перевірки в ігровому циклі використовується команда:
- **`glEnable(GL_DEPTH_TEST);`**
- Перед кожною ітерацією буфер глибини потрібно очищувати:
- **`gl_Clear(GL_COLOR_BUFFER | GL_DEPTH_BUFFER_BIT);`**



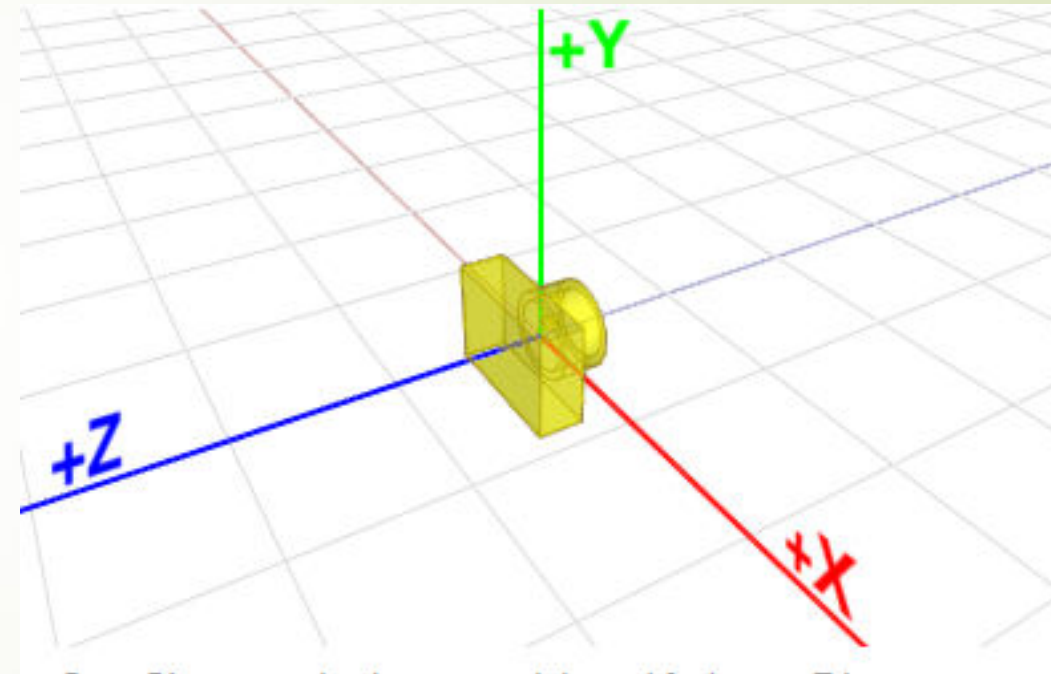
Камера

Двигуни не переміщують корабель. Він стоятиме на місці, а двигуни рухатимуть всесвіт навколо нього.

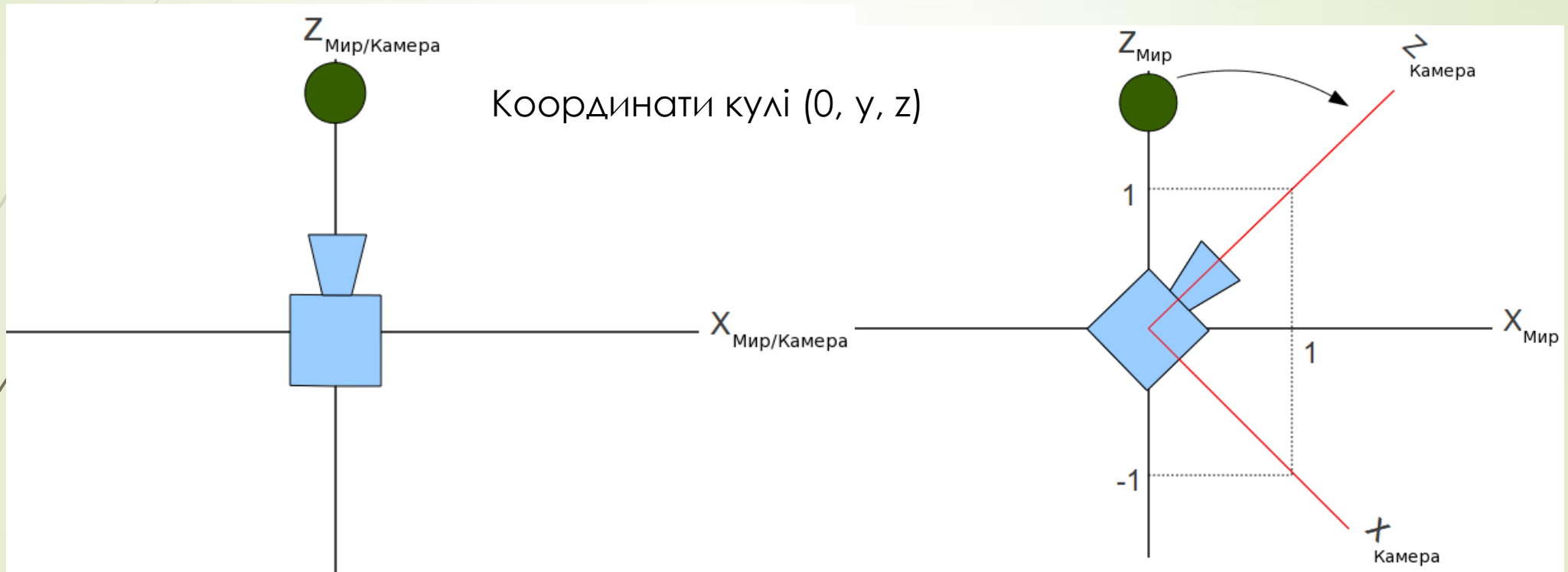
(Фурагама)

Камера: матриця вигляду

- Простір вигляду/камери – це те, як об'єкти виглядають з точки огляду камери.
- Камера повинна розміщуватися у початку світових координат (0,0,0) та дивиться вздовж протилежного напрямку вісі Z.
- Камера характеризується позицією та ціллю, визначеними у світових координатах.
- Координати сцени потрібно трансформувати у координати камери.



Система координат камери



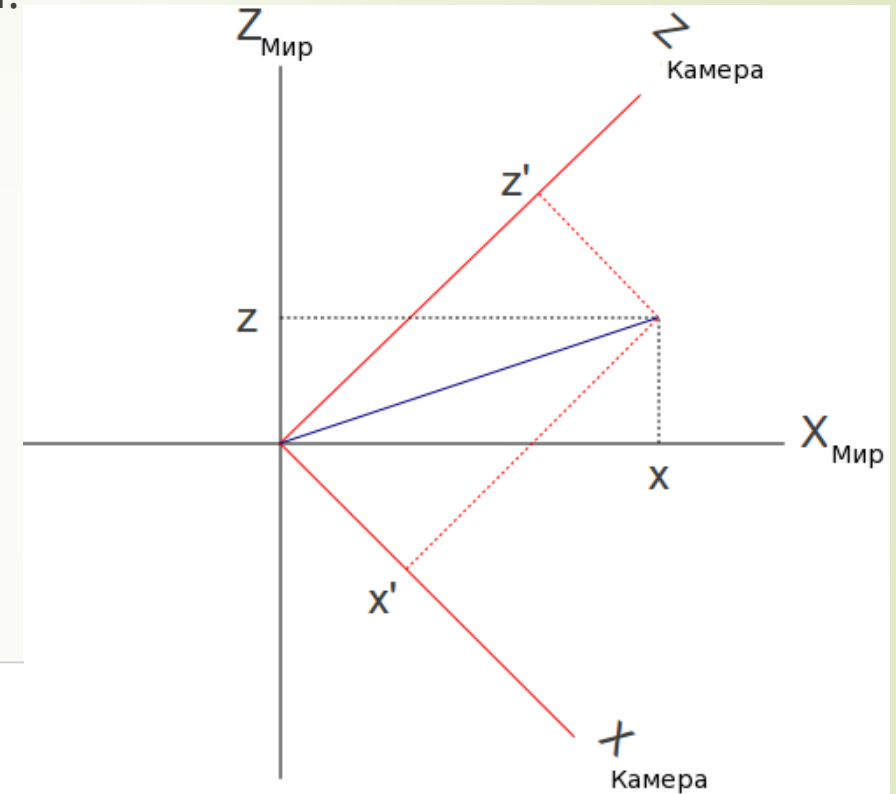
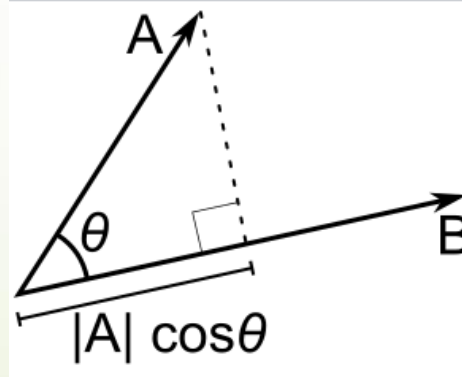
Система координат світу та камери збігаються

Система координат камери зміщена у світовому просторі

Потрібно врахувати координати кулі відносно положення камери

Перетворення у координати камери

- Система координат камери задана векторами:
 - $X (1, 0, -1)$
 - $Y (0, 1, 0)$
 - $Z (1, 0, 1)$
- Їх можна нормалізувати
- Маємо одиничні вектори, що позначають вісі і знаємо координати вектора в світі (x, y, z) .
 - Потрібно знайти координати цього вектора у системі координат камери – (x', y', z') .
 - Для цього використовуємо скалярну проекцію вектора на відповідну вісь:

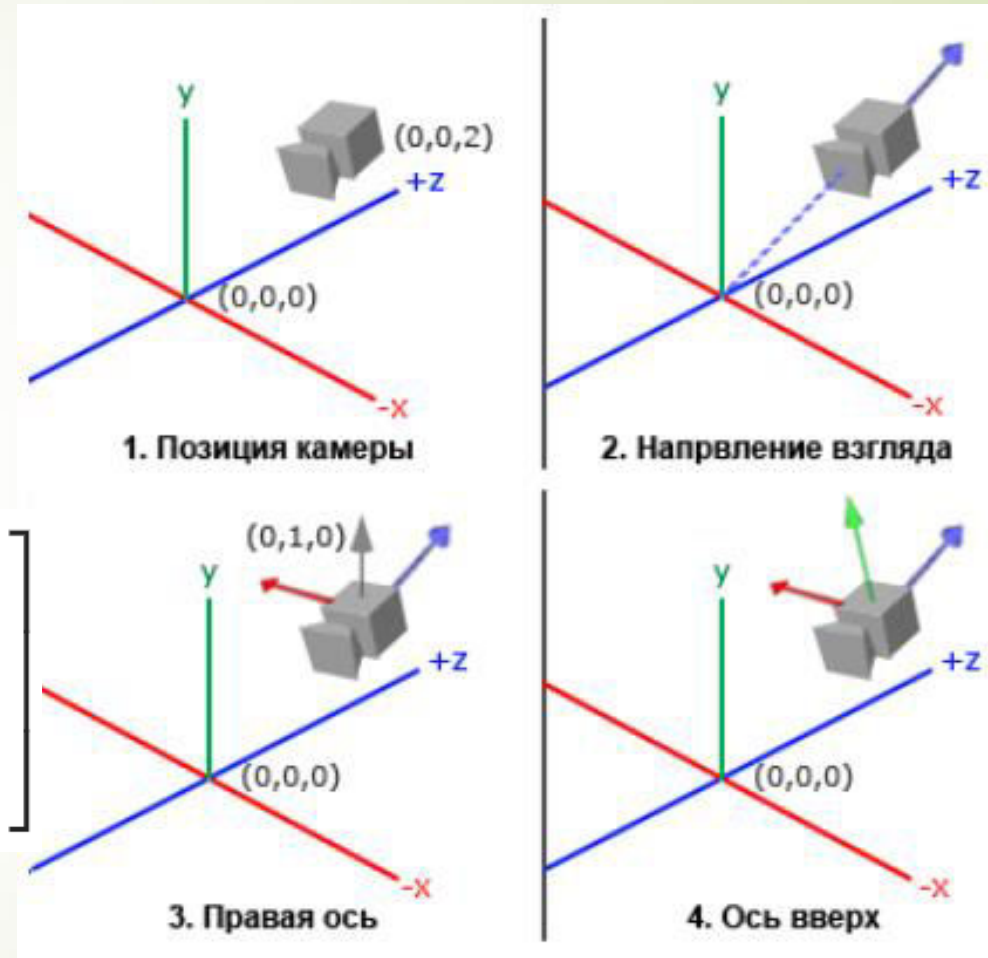


Простір камери

- D – вектор від камери до цілі (протилежний напрямку камери) – Z+
- U – вектор вгору, відповідає вісі Y+
- R – виходить з камери праворуч – вісь X+
- З векторів цих осей і додаткового вектора зміщення формується матриця LookAt:

$$LookAt = \begin{bmatrix} R_x & R_y & R_z & 0 \\ U_x & U_y & U_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

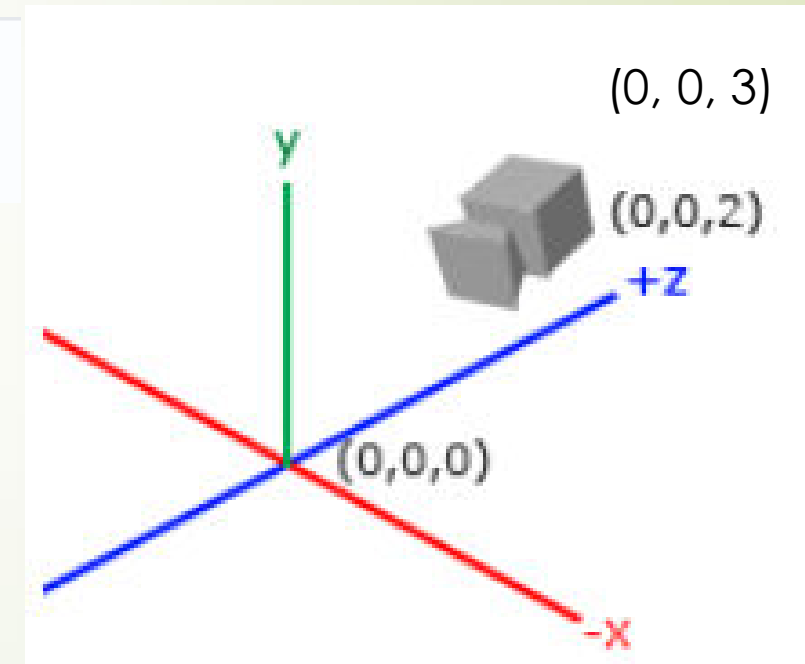
- Помноживши вектор вершини на цю матрицю перетворює його у координати простору камери.



1. Позиція камери

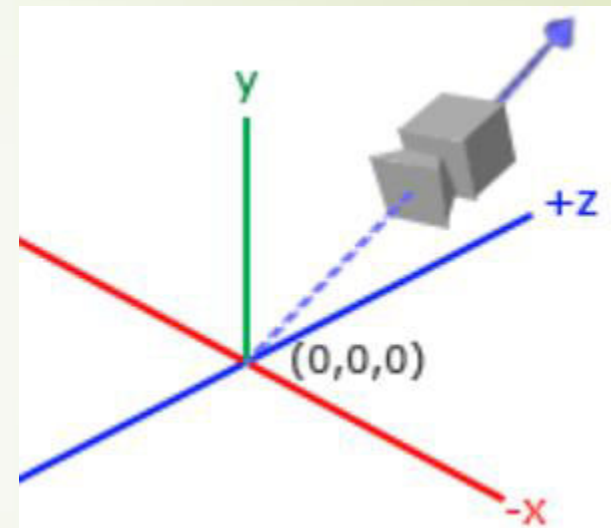
- Положення камери у світових координатах:

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



2. Напрямок камери

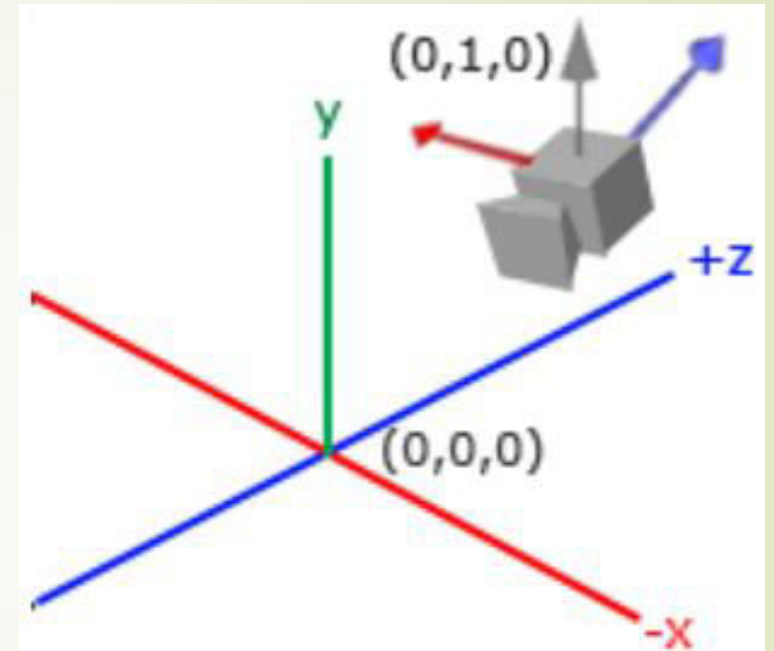
- Камеру націлено в центр сцени (0, 0, 0).
- Різниця вектору позиції і початку координат
- Дасть вектор до цілі (в напрямку Z+)



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

3. Права вісь камери

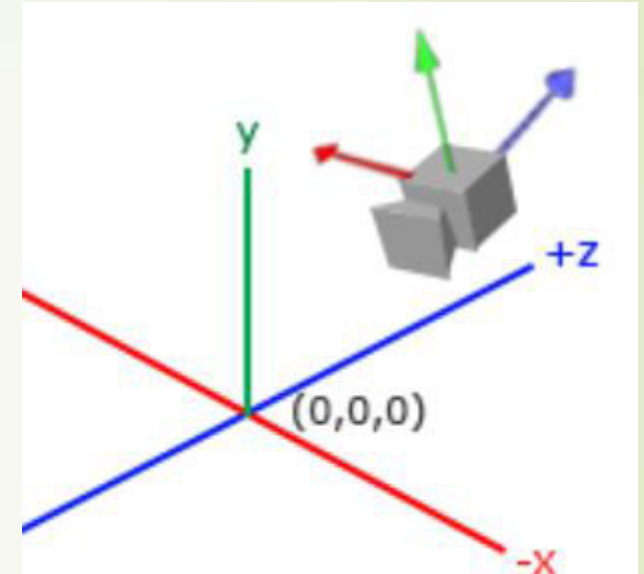
- Вектор, що дивиться праворуч від камери в напрямку $X+$
- Для обчислення:
- Задамо вектор, що вказує вгору.
- Використаємо обчислений перед цим вектор напрямку `cameraDirection`.
- Їх векторний добуток дасть вектор, перпендикулярний вихідним векторам, що вказує в напрямку $X+$:



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


4. Вісь вгору

- Як знайти вектор вісі Y камери у світових координатах?
- Знайти векторний добуток отриманих раніше векторів Z та X:
-



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

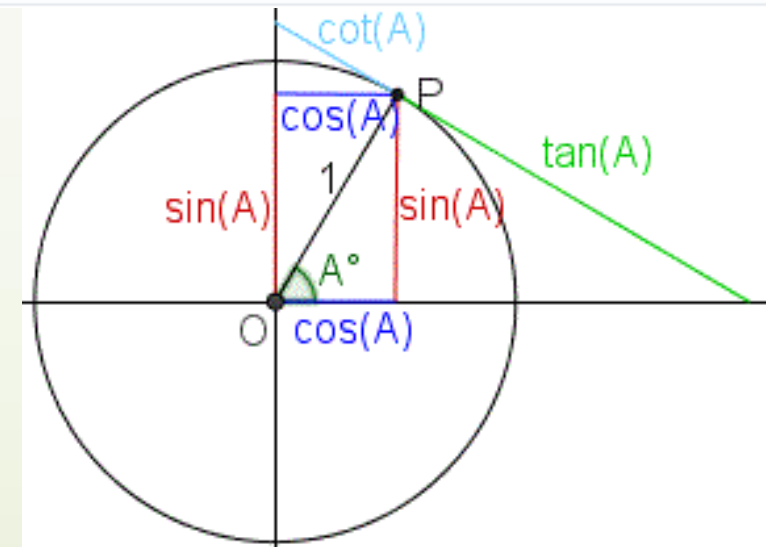
Функція LookAt

- Функція `lookAt` бібліотеки `glm` виконує розглянуті перетворення і формує матрицю вигляду.

```
glm::mat4 view;  
view = glm::lookAt(glm::vec3(0.0f, 0.0f, 3.0f), // позиція камери  
                  glm::vec3(0.0f, 0.0f, 0.0f), // координати цілі  
                  glm::vec3(0.0f, 1.0f, 0.0f)); // вектор вгору
```

Рух камери по колу

```
GLfloat radius = 10.0f;  
GLfloat camX = sin glfwGetTime() * radius;  
GLfloat 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));
```



Пересування камери

- При переміщенні камери важливо, щоб вона завжди була спрямована на ціль.
- Нехай положення камери задане такими параметрами:

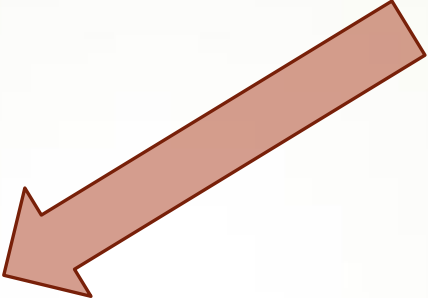
```
glm::vec3 cameraPos    = glm::vec3(0.0f, 0.0f, 3.0f);  
glm::vec3 cameraFront  = glm::vec3(0.0f, 0.0f, -1.0f);  
glm::vec3 cameraUp     = glm::vec3(0.0f, 1.0f, 0.0f);
```

- Тоді функція трансформації сцени `lookAt` матиме такий вигляд:

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

Керування камерою за допомогою клавіатури

```
bool keys[1024];  
void do_movement()  
{  
    // Керування камерою  
    GLfloat cameraSpeed = 0.01f;  
  
    if(keys[GLFW_KEY_W]) cameraPos += cameraSpeed * cameraFront;  
    if(keys[GLFW_KEY_S]) cameraPos -= cameraSpeed * cameraFront;  
    if(keys[GLFW_KEY_A]) cameraPos -=  
        glm::normalize(glm::cross(cameraFront, cameraUp)) * cameraSpeed;  
    if(keys[GLFW_KEY_D]) cameraPos +=  
        glm::normalize(glm::cross(cameraFront, cameraUp)) * cameraSpeed;  
}
```



GLFW_KEY_UP
GLFW_KEY_DOWN
GLFW_KEY_LEFT
GLFW_KEY_RIGHT

Швидкість руху камери

Дві глобальні змінні:

```
GLfloat deltaTime = 0.0f; // Час між останнім та поточним кадрами
```

```
GLfloat lastFrame = 0.0f; // Час виводу останнього кадру
```

```
while (!glfwWindowShouldClose(window))
```

```
{ // Обчислюємо deltatime для кожного поточного кадру
```

```
GLfloat currentFrame = glfwGetTime();
```

```
deltaTime = currentFrame - lastFrame;
```

```
lastFrame = currentFrame;
```

У тілі функції void do_movement() :

```
GLfloat cameraSpeed = 5.0f * deltaTime;...
```

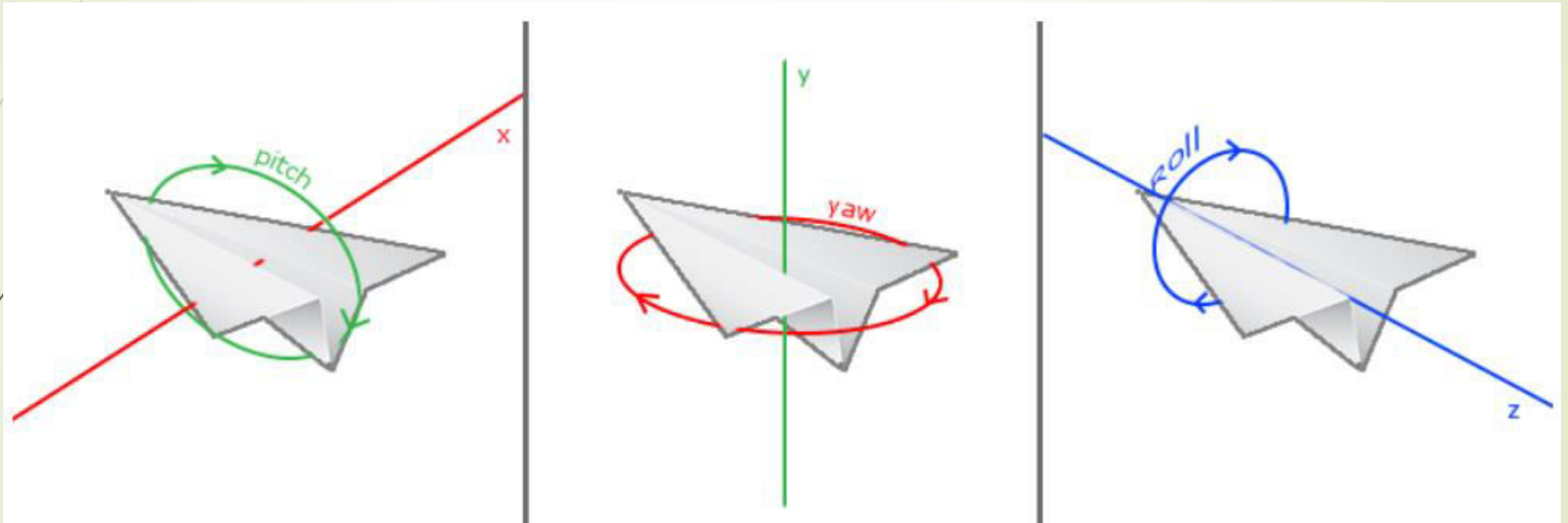
// В ігровому циклі перевіряємо настання подій (натискання клавіші, рух миші, тощо

```
glfwPollEvents();
```

```
do_movement();
```

Обертання у 3D-просторі

► Кути Ейлера

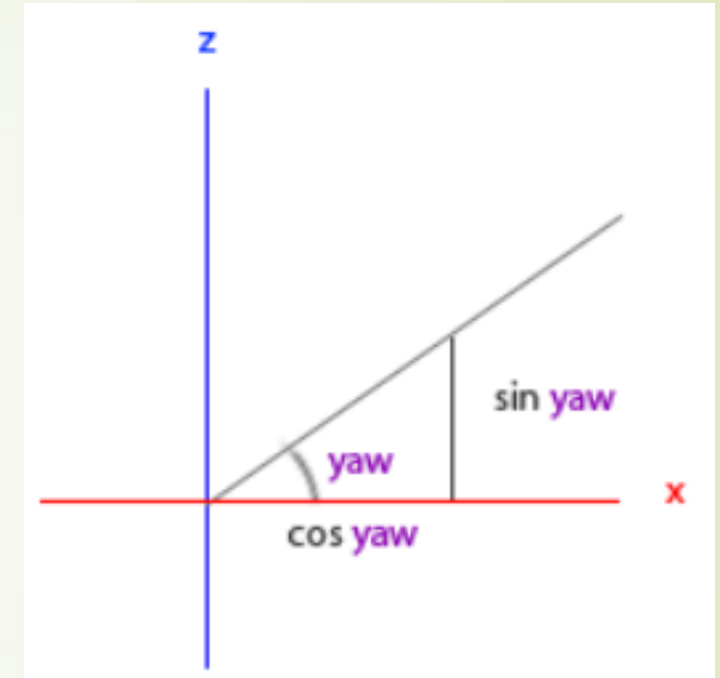
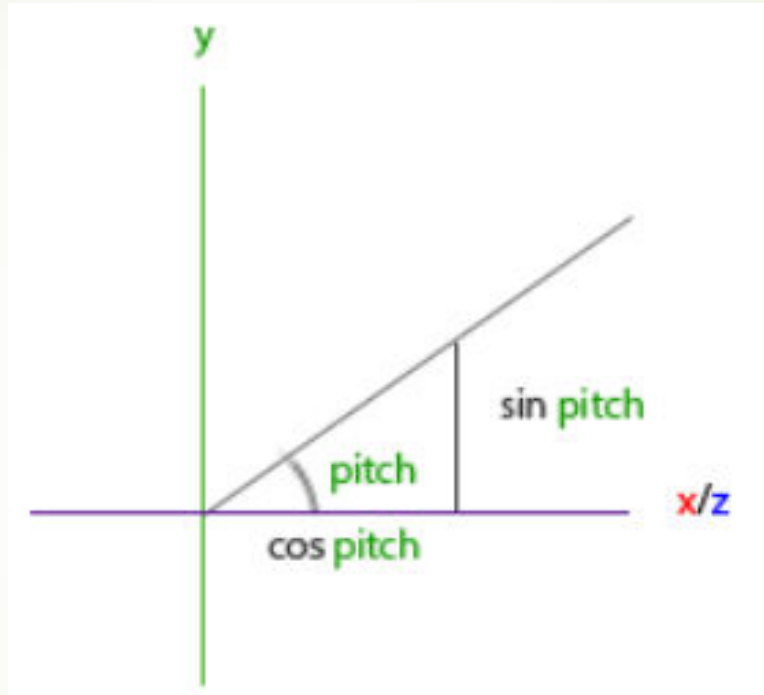
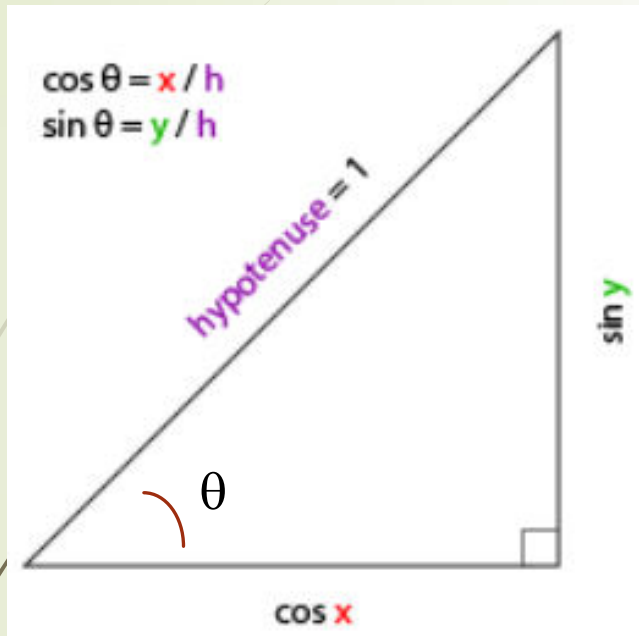


Вертикальне обертання-
кут **pitch**

Горизонтальне обертання-
кут **yaw**

Обертання навколо вісі Z -
кут **roll**

Обертання камери навколо Y та X



Вектор напрямку камери:

- `DirectionX = cos(glm::radians(pitch)) * cos(glm::radians(yaw));`
- `DirectionY = sin(glm::radians(pitch));`
- `DirectionZ = cos(glm::radians(pitch)) * sin(glm::radians(yaw));`

Керування камерою за допомогою миші

1. Захоплення і приховування курсору

```
glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);
```

2. Прототип функції для зчитування рухів миші:

```
void mouse_callback(GLFWwindow* window, double xpos, double ypos);
```

3. Реєстрація функції для взаємодії з вікном OpenGL:

```
glfwSetCursorPosCallback(window, mouse_callback);
```

4. Для реалізації функції нам знадобляться такі глобальні змінні:

Координати центру екрана (в прикладі для 800 на 600):

```
GLfloat lastX = 400, lastY = 300;
```

Кути обертання камери:

```
GLfloat yaw = -90.0f;
```

```
GLfloat pitch = 0.0f;
```

Керування камерою за допомогою миші

В тілі функції `void mouse_callback()`:

5. **Визначаємо зміщення миші з моменту останнього кадру:**

```
GLfloat xoffset = xpos - lastX;
```

```
GLfloat yoffset = lastY - ypos;
```

```
lastX = xpos;
```

```
lastY = ypos;
```

6. **Додаємо зміщення до кутів pitch і yaw камери (визначивши чутливість):**

```
GLfloat sensitivity = 0.05;
```

```
xoffset *= sensitivity;
```

```
yoffset *= sensitivity;
```

```
yaw += xoffset;
```

```
pitch += yoffset;
```

Керування камерою за допомогою миші

7. Обмежуємо максимальні/мінімальні значення кутів обертання:

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

8. Обчислюємо вектор напрямку камери cameraFront:

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

Завдання

- 1. Реалізувати матрицю трансформації MVP та LookAt для перетворення параметрів фігури.
- 2. Створити декілька об'ємних фігур, задавши для кожної свої параметри зміщення, масштабування та повороту в світовому просторі.
- 3. Реалізувати динамічний огляд сцени за допомогою камери, використавши клавіатуру та мишку.
- 4. Створіть камеру, яка обертається навколо об'єкта ($\text{position} = \text{ObjectCenter} + (\text{radius} * \cos(\text{time}), \text{height}, \text{radius} * \sin(\text{time}))$); Пов'яжіть радіус/висоту/час до клавіатури/миші.
- 5. **Реалізуйте створену програму за допомогою класів:** Шейдер, Камера, Фігура, Головна програма, тощо, забезпечивши принципи ООП, перевірку правильності введення початкових параметрів, мінімальний користувацький інтерфейс.

Вітаю!!! Ми завершили базовий курс з OpenGL

35

Далі буде...

