ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Εισαγωγή στα Γραφικά Υπολογιστών

Εργαστήριο 4

Σ.σ. θέασης/κάμερας

- Προσομοιάζουμε την μετακίνηση της κάμερας, μετακινώντας όλα τα αντικείμενα στην αντίθετη κατεύθυνση

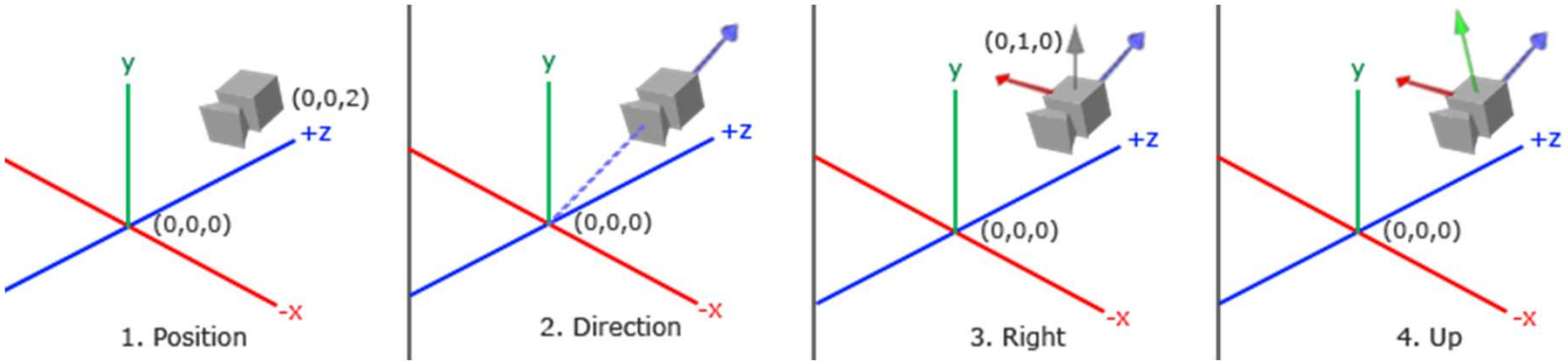
- Σ.σ. κόσμου → σ.σ. κάμερας: με τον πίνακα θέασης (view matrix)

- Μετασχηματισμός όλων των συντεταγμένων ώστε στο καινούριο σύστημα συντεταγμένων η κάμερα:
  - θα βρίσκεται στο κέντρο των αξόνων
  - θα έχει κατεύθυνση προς τον αρνητικό Z άξονα

Κάμερα

- Για να ορίσουμε το σύστημα συντεταγμένων της κάμερας, αρκούν:

- Η θέση της στο σ.σ. του κόσμου
- Το διάνυσμα κατεύθυνσης της κάμερας
- Το διάνυσμα κατεύθυνσης προς τα δεξιά
- Το διάνυσμα κατεύθυνσης προς τα πάνω



Κάμερα

- Η θέση της κάμερας:

glm::vec3 cameraPos = glm::0. vec3(0.0f, 0f, 3.0f);

- Η κατεύθυνση προς τα εμπρός:

Το διάνυσμα θα δείχνει προς την αντίθετη κατεύθυνση από την κατεύθυνση στην οποία 'κοιτάζει' η κάμερα.

glm::vec3 cameraTarget = glm::vec3(0.0f, 0.0f, 0.0f);

glm::vec3 cameraDirection = glm::normalize(cameraPos - cameraTarget);

Κάμερα

- Η κατεύθυνση προς τα δεξιά:

Το εξωτερικό γινόμενο του κανονικού διανύσματος που δείχνει 'πάνω' στο σ.σ. του κόσμου και του διανύσματος κατεύθυνσης της κάμερας.

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

- Η κατεύθυνση προς τα πάνω:

Το εξωτερικό γινόμενο του διανύσματος κατεύθυνσης της κάμερας και του διανύσματος προς τα δεξιά.

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

Κάμερα

- Από τα 4 παραπάνω διανύσματα μπορούμε να δημιουργήσουμε τον πίνακα που θα μετασχηματίζει τις κορυφές στο σ.σ. της κάμερας

- Ονομάζεται **LookAt matrix**

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

Κάμερα

- Η βιβλιοθήκη GLM διαθέτει τη συνάρτηση lookAt(), η οποία δημιουργεί έναν πίνακα LookAt:

```
glm::mat4 view;
view = glm::lookAt(glm::vec3(0.0, 0.0, 3.0), glm::vec3(0.0, 0.0, 0.0),
glm::vec3(0.0, 1.0, 0.0));

// view = glm::lookAt(cameraPos, cameraTarget, worldUp);
```

Άσκηση

- Περιστροφή της κάμερας γύρω από το κέντρο:

```
float radius = 30.0f;
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));
```

Κίνηση στο χώρο

- Για να στήσουμε την κάμερα χρειαζόμαστε τις μεταβλητές:

```
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);
```

- Άρα η συνάρτηση lookAt θα γίνει:

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

// lookAt(cameraPos, cameraTarget, worldUp);
```

Κίνηση στο χώρο

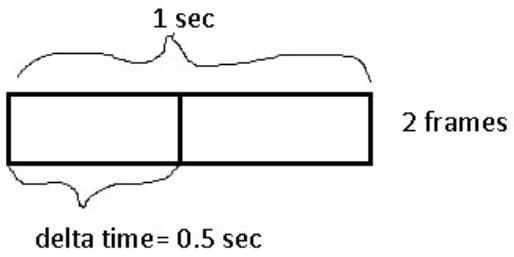
```
float cameraSpeed = 0.01f;

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;
```

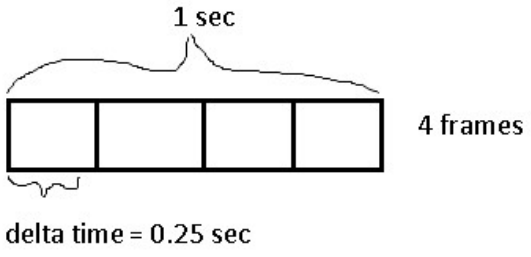
Ταχύτητα

- Η ταχύτητα της κάμερας δεν είναι ομοιόμορφη.
- Εξαρτάται από την υπολογιστική δύναμη του κάθε συστήματος.
- Για ομοιόμορφη ταχύτητα, χρησιμοποιούμε μια μεταβλητή **deltaTime**, η οποία αποθηκεύει το χρόνο που πέρασε από το τελευταίο frame.
- Στη συνέχεια, όλες οι ταχύτητες πολλαπλασιάζονται με την τιμή της deltaTime.

Ταχύτητα (μη ρεαλιστικό παράδειγμα)

- Έστω ότι σε ένα μηχάνημα η υπολογιστική του δύναμη οδηγεί σε 2 frames/s και σε ένα άλλο σε 4 frames/s και έστω ότι τα frames έχουν ακριβώς την ίδια διάρκεια.





- Τότε αν η μεταβλητή speed έχει τιμή 4 πολλαπλασιασμένη με το delta time θα δίνει βήματα των 2 στην πρώτη περίπτωση ενώ στην δεύτερη περίπτωση βήματα των 1.
- Παρατηρούμε ότι και στις 2 περιπτώσεις μέσα σε ένα δευτερόλεπτο θα πραγματοποιηθεί ακριβώς η ίδια μετατόπιση (κατά 4) ανεξάρτητα από τα fps.

Ταχύτητα

- Για τον υπολογισμό της deltaTime χρειάζονται δυο μεταβλητές:

```
float deltaTime = 0.0f; // Time between current frame and last frame
float lastFrame = 0.0f; // Time of last frame
```

- Σε κάθε frame γίνονται οι υπολογισμοί:

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

Ταχύτητα

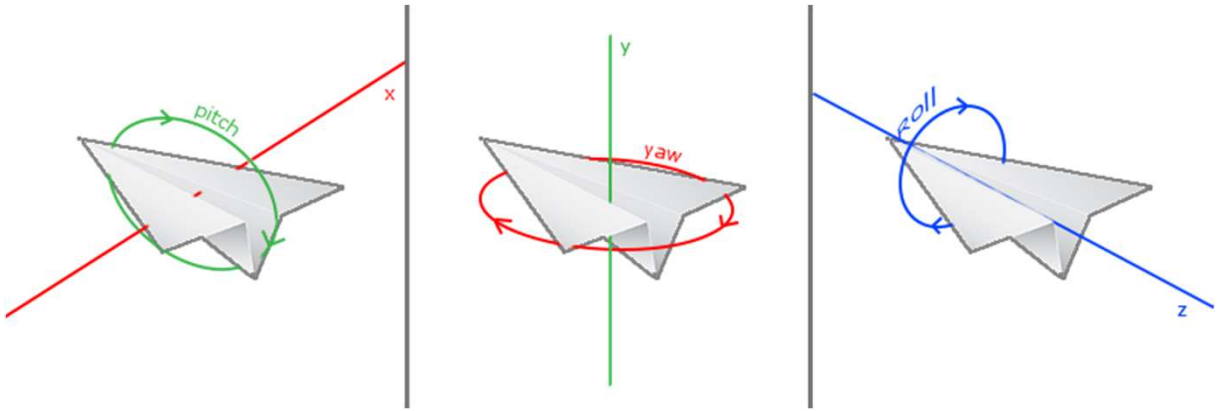
- Λαμβάνουμε την deltaTime υπόψη μας όταν υπολογίζουμε τις ταχύτητες:

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

Περιστροφή στο χώρο

- Για να κοιτάζουμε ελεύθερα στο χώρο, πρέπει να αλλάζουμε το διάνυσμα cameraFront ανάλογα με την είσοδο που δίνεται από το ποντίκι

- **Γωνίες Euler**: περιγράφουν οποιαδήποτε 3Δ περιστροφή

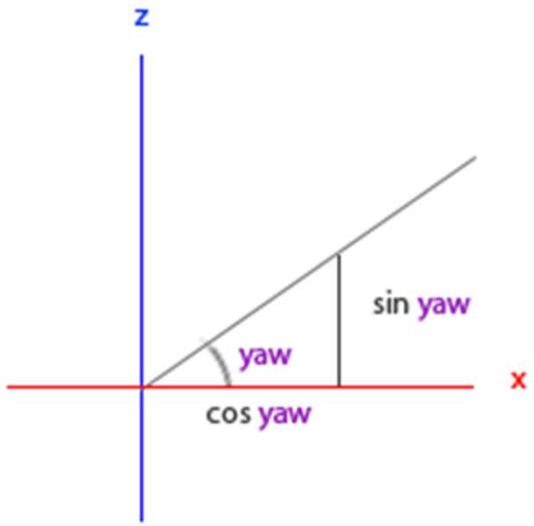


Περιστροφή στο χώρο

- **Yaw**:

direction.x = cos(glm::radians(yaw));

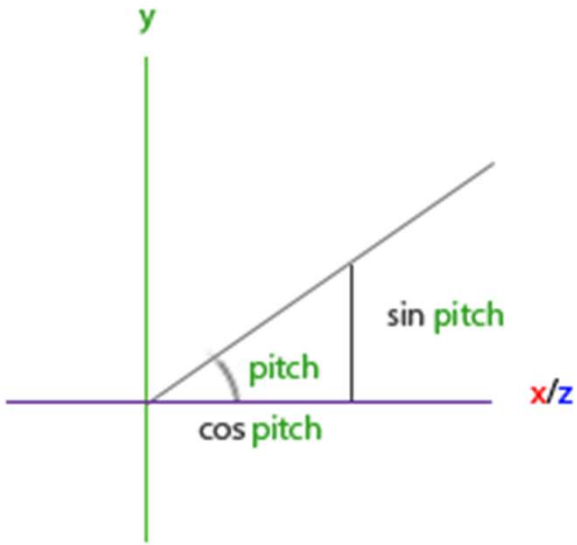
direction.z = sin(glm::radians(yaw));



Περιστροφή στο χώρο

- **Pitch**:

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



Περιστροφή στο χώρο

- Για να χρησιμοποιήσουμε την είσοδο του ποντικιού χρειαζόμαστε μια συνάρτηση callback:

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

- Δηλώνουμε τη συνάρτηση αυτή στην GLFW στη main:

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

Περιστροφή στο χώρο

Για να υπολογίσουμε το διάνυσμα της κατεύθυνσης της κάμερας:

- Υπολογισμός της διαφοράς της θέσης του ποντικιού από το προηγούμενο frame
- Προσθήκη της διαφοράς στις μεταβλητές yaw και pitch
- Προσθήκη περιορισμών στις μεταβλητές
- Υπολογισμός του καινούριου διανύσματος κατεύθυνσης

Περιστροφή στο χώρο

- Για τον υπολογισμό της αλλαγής της θέσης του ποντικιού:

```
float lastX = 400, lastY = 300; //middle of the screen
```

- Στη συνάρτηση callback υπολογίζουμε την αλλαγή σε σχέση με το προηγούμενο frame:

```
float xoffset = xpos - lastX;
float yoffset = lastY - ypos; // reversed since y-coordinates range from bottom to top
lastX = xpos;
lastY = ypos;

float sensitivity = 0.05f; //multiply by a sensitivity value, or else too strong
xoffset *= sensitivity;
yoffset *= sensitivity;
```

Περιστροφή στο χώρο

- Δημιουργούμε τις global μεταβλητές pitch και yaw:

```
float yaw   = -90.0f; // yaw is initialized to -90.0 degrees
float pitch =  0.0f;
```

- Προσθέτουμε τις αλλαγές στις μεταβλητές pitch και yaw:

```
yaw   += xoffset;
pitch += yoffset;
```

- Θέλουμε να περιορίσουμε την κάμερα ώστε να μην έχουμε περίεργες κινήσεις:

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

Περιστροφή στο χώρο

- Το τελευταίο βήμα είναι να υπολογίσουμε το διάνυσμα της κατεύθυνσης από τις τιμές των yaw και pitch:

```
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);
```

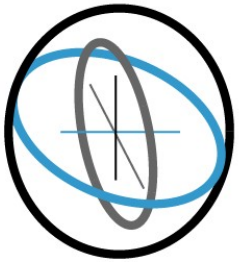
- Έτσι, το διάνυσμα περιέχει όλες τις περιστροφές που υπολογίσαμε από την κίνηση του ποντικιού.

Περιστροφή στο χώρο

- Μπορούμε να κρύψουμε τον κέρσορα μέσα στην εφαρμογή:

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

- Ένα πρόβλημα που υπάρχει με τις γωνίες Euler ονομάζεται Gimbal Lock:



three-gimbals

gimbal lock

Περιστροφή στο χώρο

- Η πρώτη θέση του ποντικιού είναι συνήθως μακριά από το κέντρο της οθόνης → μεγάλη αρχική κίνηση

- Θέτουμε μια Boolean μεταβλητή:

```
bool firstMouse = true;
```

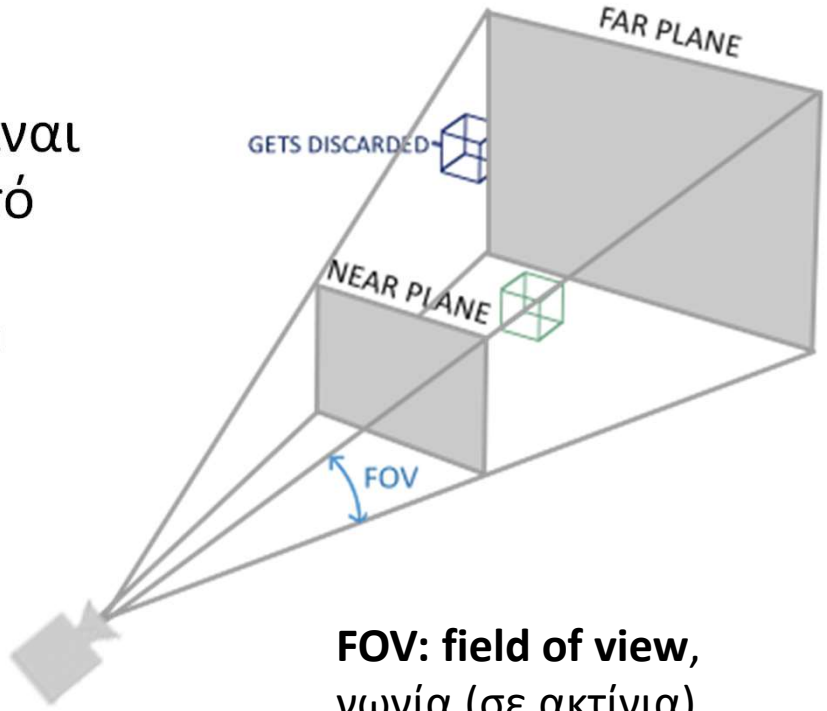
- Έλεγχος στη συνάρτηση callback:

```
if(firstMouse)
{
    lastX = xpos;
    lastY = ypos;
    firstMouse = false;
}
```

Zoom

- Το πεδίο όρασης (Field of View) καθορίζει πόση από τη σκηνή είναι ορατή. Όταν μειώνεται, το ορατό μέρος της σκηνής μειώνεται, δίνοντας την αίσθηση του zoom

- Η default τιμή του fov είναι 45 μοίρες, για το zoom το fov θα μεταβάλλεται μεταξύ 1 και 45 μοιρών



FOV: field of view, γωνία (σε ακτίνια)

Zoom

- Για να κάνουμε zoom θα χρησιμοποιήσουμε τον τροχό του ποντικιού:

```
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset)
{
  fov -= (float)yoffset;
  if(fov < 1.0f)
        fov = 1.0f;
  if(fov > 45.0f)
        fov = 45.0f;
}
```

Zoom

- Πρέπει να ενημερώσουμε τον πίνακα projection:

projection = glm::perspective(glm::radians(fov), screenWidth / screenHeight, 0.1f, 100.0f);

- Και τέλος να δηλώσουμε την συνάρτηση callback στην GLFW:

glfwSetScrollCallback(window, scroll_callback);

Camera Class

- Θα ενσωματώσουμε την κλάση που βρίσκεται στο αρχείο 'Camera.h' στον κώδικα μας.