EULER ANGLES

HJALMAR BASILE
JAN 2019

Euler angles are three angles, PITCH, YAW and ROLL, which can be used to represent rotations in 3D.

PITCH	rotation around the x-axis
YAW	rotation around the y-axis
ROLL	rotation around the z-axis

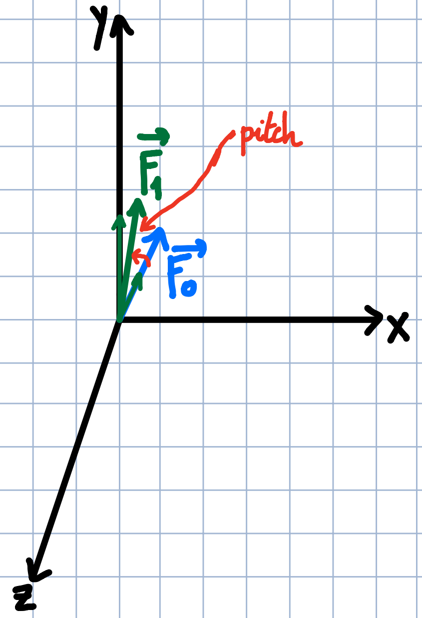
The intent of this note is to focus on pitch and yaw to explain the implementation of the following method:

```

void Camera::UpdateCameraVectors()
{
        m_CameraFront.x = -cos(glm::radians(m_Pitch)) * sin(glm::radians(m_Yaw));
        m_CameraFront.y = sin(glm::radians(m_Pitch));
        m_CameraFront.z = -cos(glm::radians(m_Pitch)) * cos(glm::radians(m_Yaw));
        /* No need to normalize, since the magnitude is already 1 by construction */

        m_CameraRight = glm::normalize(glm::cross(m_CameraFront, m_WorldUp));
}
```

Given pitch and yaw values, we build the camera forward vector obtained by applying pitch first and yaw later. Let's start by describing the pitch transform.



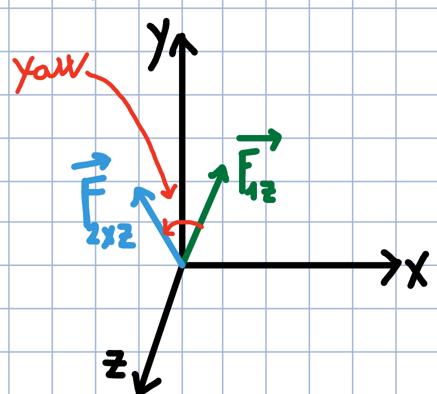
We will consider as default forward versor $\vec{F}_0 = (0,0,-1)$, while $\vec{F}_1$ is the versor obtained applying pitch to $\vec{F}_0$.

[NOTE: We will use the terms yaw and pitch to refer both to the rotations and to the corresponding angles]

We get $F_{1y} = \sin(\text{pitch})$, also the magnitude of the $\vec{F}_1$ component lying on the xz-plane is $\cos(\text{pitch})$

(N.B. We are enforcing $-\frac{\pi}{2} < \text{pitch} < \frac{\pi}{2}$)

Let's apply yaw now: what we notice is that the y component is not gonna change, so we can focus on the xz-plane:



We get $F_{2z} = -|\vec{F}_{1z}| \cdot \cos(\text{yaw})$

and $F_{2x} = -|\vec{F}_{1z}| \cdot \sin(\text{yaw})$

Putting all together we get the final result $\vec{F}$ $(= \vec{F}_2)$

$$\vec{F} = \begin{pmatrix} -\cos(pitch) \cdot \sin(yaw) \\ \sin(pitch) \\ -\cos(pitch) \cdot \cos(yaw) \end{pmatrix}$$

Notice that $|\vec{F}|$ will always be 1, also for $pitch = yaw = 0$ we get the default forward vector $\vec{F} = (0, 0, -1)$.

Final note about mouse input handling:

```
void Camera::ProcessMouseMovement(float xoffset, float yoffset)
{
        xoffset *= MOUSE_HORIZONTAL_SENSITIVITY;
        yoffset *= MOUSE_VERTICAL_SENSITIVITY;

        float pitchOffset = yoffset;
        float yawOffset = -xoffset;

        m_Pitch = std::clamp(m_Pitch + pitchOffset, -80.0f, 80.0f);
        m_Yaw += yawOffset;

        UpdateCameraVectors();
}
```

If we move the mouse to the right we will get $\Delta x = xoffset > 0$, but in our conventions yaw would be negative, that's why we set $\Delta yaw = -\Delta x$.