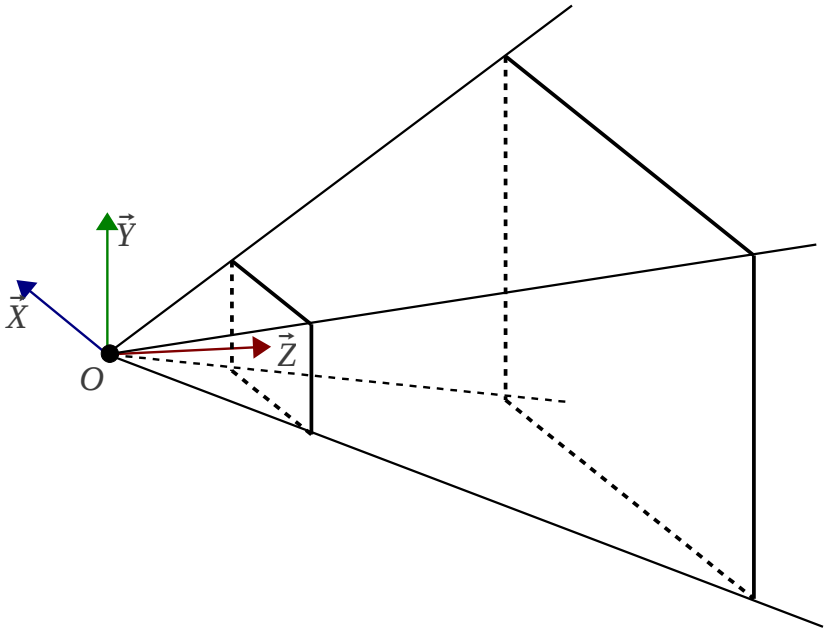
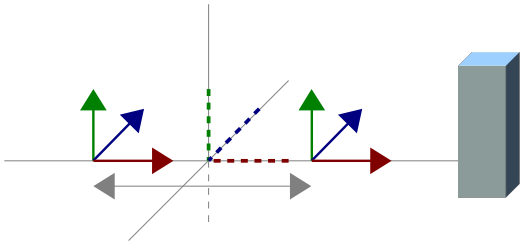
1 (Practice 3) Camera

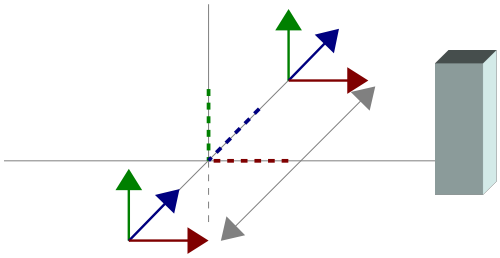
The goal of this practice is to implement camera movements and to drive them thanks to the keyboard. First download the materials associated to this practice and take a look to 'camera.h'. As you can see in the C structure, the camera is defined with a frame (O, $\vec{X}$, $\vec{Y}$, $\vec{Z}$) where O is the origin of the camera and $\vec{Z}$ the direction you are looking at.



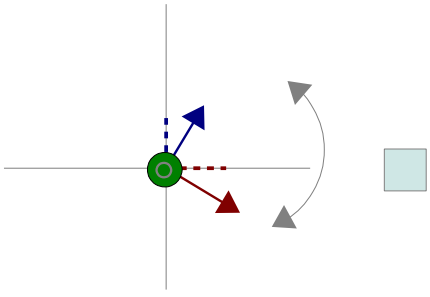
Here is the list of the translation/rotation we will implement:



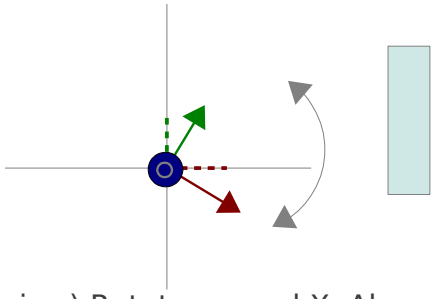
Translate along Z. (walking towards or backwards)



Translate along X. (walking right or left)



(Top view) Rotate around Y. Also called 'yaw' (turning left or right)



(Right view) Rotate around X. Also called 'pitch' (turning up or down)

1.1 Implement the camera movements and key callback

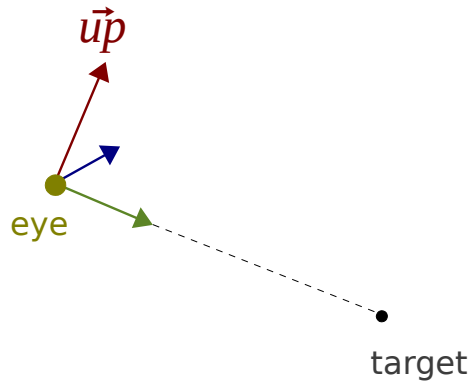
Before using the keys implement these functions in camera.c:

```
// Compute the camera frame according to eye, up and target
Camera camera_init(Vec4 eye, Vec4 up, Vec4 target);

// call gluLookAt() with the correct parameters
void camera_place(Camera c);

void camera_get_eye   (Camera c, Vec4 res_eye   );
void camera_get_up    (Camera c, Vec4 res_up    );
void camera_get_target(Camera c, Vec4 res_target);
```

camera_init() parameters are represented below:



From this parameters camera_init() must compute the frame (O, $\vec{X}$, $\vec{Y}$, $\vec{Z}$) of the camera and return it in the structure 'Camera'. Here is the relation between eye, target, up and the camera frame:

$$O = eye$$
$$\vec{Y} = \frac{\vec{up}}{\|\vec{up}\|}$$
$$\vec{Z} = \frac{target - eye}{\|target - eye\|}$$
$$\vec{X} = cross(\vec{Z}, \vec{Y})$$

(use the functions in vector4.h)

Declare a global variable for the camera 'Camera _cam;' and initialize it with the parameters you currently pass to 'gluLookAt()' inside your 'display()' loop.

Replace 'gluLookAt()' with 'camera_place()' and check that your camera hasn't moved. When implementing 'camera_place()' you can use camera_get_xxx() to translate the frame to the 'gluLookAt()' parameters.

$$eye = O$$
$$\vec{up} = \vec{Y}$$
$$target = O + \vec{Z}$$

1.1.1 Translations

Implement the following functions:

```
// Translation left and right
Camera camera_walksideway(Camera c, float step);
// Translation front and back
Camera camera_walk(Camera c, float step);
```

Walking only modifies the origin of the camera frame. Just translate the point 'O' along the correct camera axis (use the functions in vector4.h).

$$O' = O + \vec{axis} * step$$

Now add the control keys to walk sideways with the arrow keys right/left. Also add control to walk towards/backwards with the arrows up/down.

Unlike numbers and letters, arrow events are handle with a different callback. You can set up a glut callback with void 'glutSpecialFunc(void (*func)(int key, int x, int y));' which handle arrows.

The 'key' parameter can take these values :

- GLUT_KEY_UP (up arrow)
- GLUT_KEY_RIGHT (right arrow)
- GLUT_KEY_LEFT (left arrow)
- GLUT_KEY_DOWN (down arrow)

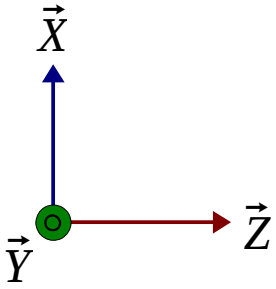
1.1.2 Rotations

It's time to implement these functions:

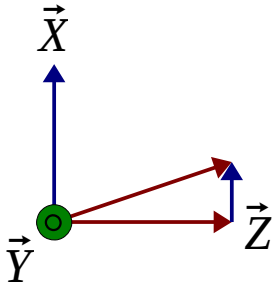
```
// Rotation Up and down
Camera camera_pitch(Camera c, float step);
// Rotation left and right
Camera camera_yaw(Camera c, float step);
```

We suggest to use the keys 'i' 'k' 'j' 'l' for respectively the rotation up and down and the rotation left and right.

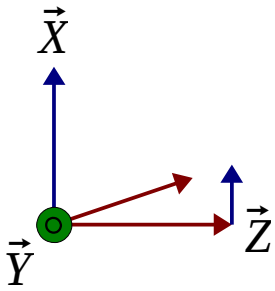
Rotation is slightly more complex to implement. We could use a rotation matrix and rotate every vector of our frame to get a new frame turned around the $\vec{X}$ or $\vec{Y}$ axis. However there is an easier way to do that. Because camera movements will be always relatively small we can simply add vectors to achieve the rotation:



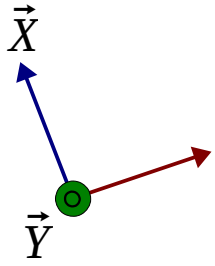
Camera frame in top view



Adding a small vector toward X, rotates Z around Y.



Don't forget to normalize



X must be updated with a cross product to keep the vectors orthogonal

Here is an example to rotate around the $\vec{X}$ axis:

$$\vec{Z}' = \vec{Z} + \vec{Y} * step$$

you need to normalize $\vec{Z}'$:

$$\vec{Z}' = \frac{\vec{Z}'}{\|\vec{Z}'\|}$$

Each axis needs to be orthogonal between each others:

$$\vec{Y}' = cross(\vec{X}, \vec{Z})$$