# Developing a First Person Camera System with Panda3D and C++

by Fero63 a.k.a. [I won't tell you ;)]

## Introduction

First, I would like you to keep in mind both that I am from Germany, so I will probably make mistakes and that this is going to be my first tutorial on developing with Panda3D. So if you find any mistakes concerning either the content of this tutorial or even the grammar - tell me, so I can correct it!
As you can probably conclude from the title, I am going to explain how to program a 1st person camera system as it is known from First Person Shooters. The camera will thereby be rotated around the X- and the Z-axis by moving the mouse and moved by using the keys W, A, S and D.

I hope you will enjoy reading.

Fero63

## 1. The mathematics behind it

As you can imagine, this all will not work without a mathematical concept. Let's start off with the rotation:

So how do we rotate the Panda3D camera? First, you have to know that I do not actually use the camera itself, but its scene node stored in a **NodePath** variable. We can easily get the node of the camera by doing the following:

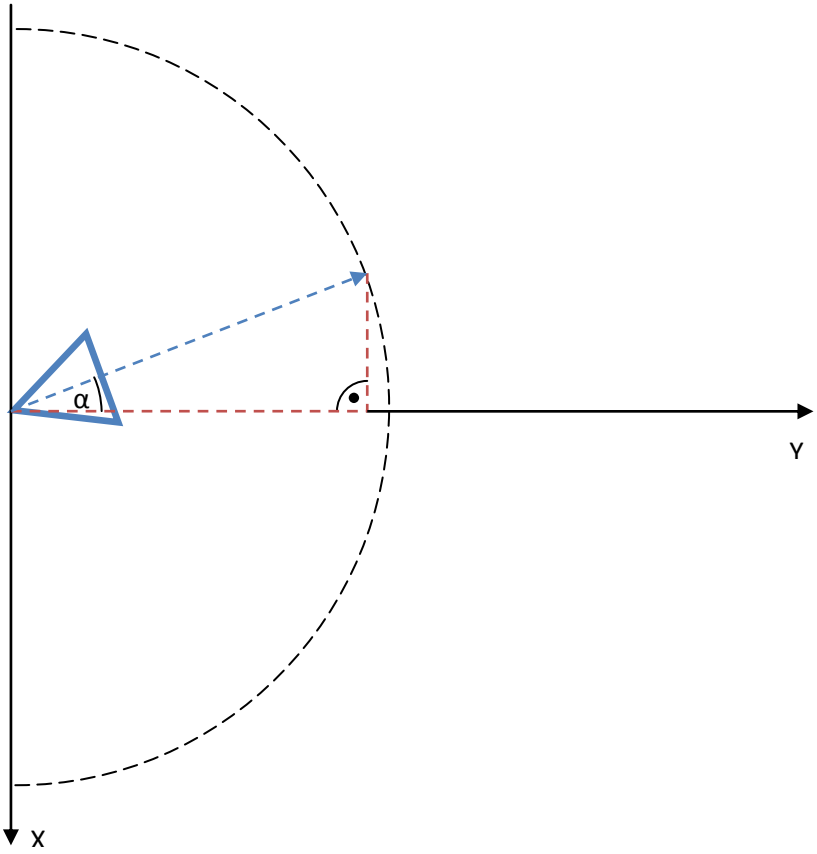
C++ Code:

```
NodePath camera = framework.get_window(0)->get_camera_group();
```

In this example, **framework** is a variable of the type **PandaFramework.**
Note that this method will only work, when you have already successfully created a window.
In order to rotate the camera now, we only have to call the **NodePath::set_hpr** function and pass as parameters the rotation around the X-, Y- and Z-axis in degrees.

Not much of maths, is it? But do not lean back and relax. The really mathematical part is following now, as we go for the movement of the camera:

Basically, we are calling the NodePath::set_pos function to translate the camera. But there is one problem: The direction in which the camera has to move depends on the rotation angle. To solve this problem, let's draw two little sketches:




1st sketch: Top view of the scene

The above sketch shows the scene from the top. The blue triangle is our camera. The angle α is its rotation angle. The red dotted lines are the X and Y components of the forward-movement, which is drawn as a blue dotted arrow. In order to move the camera into this direction, we have to calculate the X and Y components. As α is known, this can be done by using simple trigonometry:

Calculation:

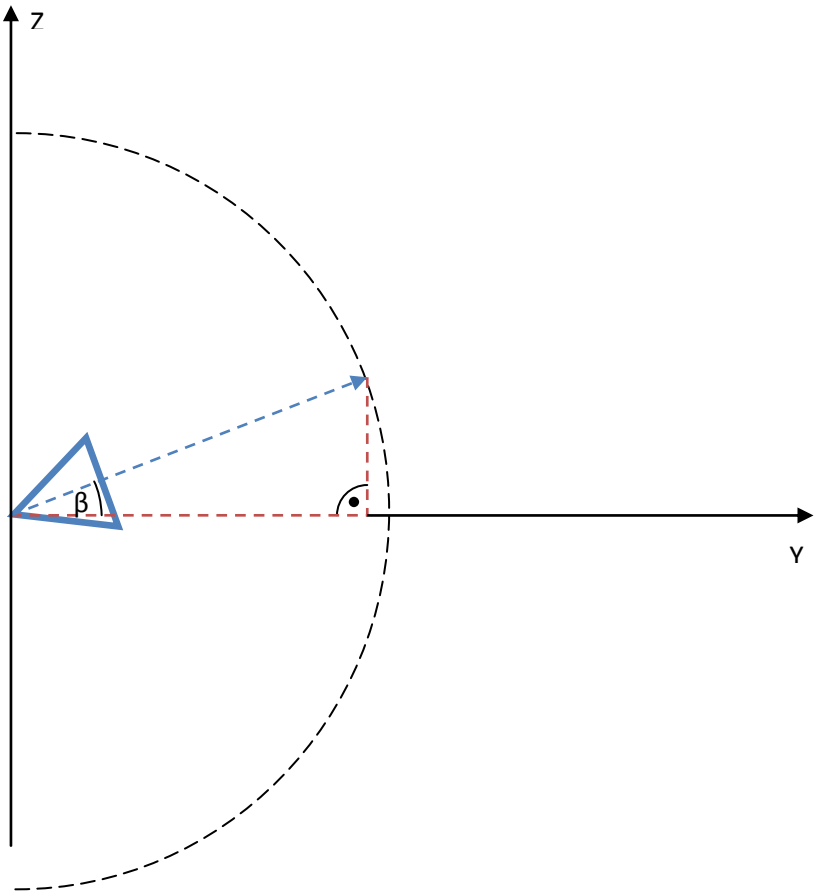
$$\sin\alpha = -\frac{x_{comp}}{m_{total}}$$
$$-x_{comp} = \sin\alpha \times m_{total}$$

The same thing goes for the Y component:

Calculation:

$$\cos\alpha = \frac{y_{comp}}{m_{total}}$$
$$y_{comp} = \cos\alpha \times m_{total}$$

Now we use a second sketch to find a formula for the Z component of the forward-movement.



2nd sketch: Side view of the scene

Actually, it is the same sketch as the first one except that we are looking at the scene from the side now. Doing this we can now use the same trigonometric concepts as before to now determine the Z component of the movement. Of course, there is again the Y component within the sketch, but there is no need to recalculate it.
The angle β is the rotation angle, just as α. The difference is that β is the rotation angle (in degrees) around the X-axis and α is, as you probably recall, the rotation angle around the Z-axis. Using β instead of α, it is quite easy to find a formula for the Z component of our total movement (By the way, you will not need a Z component when developing an FPS as long as the player cannot fly).

Calculation:

$$\sin\beta = \frac{z_{comp}}{m_{total}}$$
$$z_{comp} = \sin\beta \times m_{total}$$

The three formulas, we have just derived from the above sketches, describe the forward-movement of the camera. Now it is an easy task to deduce the corresponding formulas for the backward-, right- and left-movement. The results:

**1. Forward-movement**

$$x_{comp} = -\sin\alpha \times speed_{cam}$$

$$y_{comp} = \cos\alpha \times speed_{cam}$$

$$z_{comp} = \sin\beta \times speed_{cam}$$

**2. Backward-movement**

$$x_{comp} = \sin\alpha \times speed_{cam}$$

$$y_{comp} = -\cos\alpha \times speed_{cam}$$

$$z_{comp} = -\sin\beta \times speed_{cam}$$

**3. Left-movement**

$$x_{comp} = -\cos\alpha \times speed_{cam}$$

$$y_{comp} = -\sin\alpha \times speed_{cam}$$

**4. Right-movement**

$$x_{comp} = \cos\alpha \times speed_{cam}$$

$$y_{comp} = \sin\alpha \times speed_{cam}$$

As you can see, I have replaced $\mathbf{m_{total}}$ with $\mathbf{speed_{cam}}$, because it is actually the speed of the camera (the number of units it moves per frame) that we multiply the Sine/Cosine of the rotation angle with. Furthermore, you might have noticed that neither the right- nor the left-movement have a Z component. Thinking about it you will notice that those two movements can never move the camera along the Z-axis, as we are not rotating the camera around the Y-axis. Think about it – it is true!

Keeping these formulas in mind (or on a sheet of paper), we can now start programming!

## 2. The sample application's frame

We start by doing the basic things: Initializing the framework, creating a render window and loading a scene, in which we can test our camera. If you do not know how to do these, I suggest that you read the basic tutorials before going on. Otherwise, here is the code:

C++ Code:

```
#include <pandabase.h>
#include <pandaFramework.h>
#include <pandaSystem.h>
#include <GenericAsyncTask.h>
#include <AsyncTaskManager.h>

PandaFramework framework;
PT(AsyncTaskManager) taskMgr = AsyncTaskManager::get_global_ptr();

int main(int argc, char * argv[])
{
	WindowFramework * win;
	NodePath environ;

	//Open the framework & create a window
	framework.open_framework(argc, argv);
	framework.set_window_title("Panda3D Tests");
	win = framework.open_window();

	//Enable the keyboard
```

```
        win->enable_keyboard();

        //Load the environment
        environ = win->load_model(framework.get_models(),
                                  "models/environment");
        environ.reparent_to(win->get_render());
        environ.set_scale(0.25f);
        environ.set_pos(-8, 42, 0);

        //Enter the main loop
        framework.main_loop();

        //Close framework & quit
        framework.close_framework();
        return 0;
}
```

I will not explain this code, since it should be known to you. This is the frame in which we will insert the camera-relevant code.

## 3. Rotating the camera

Let us start with the easier part – the rotation. As there is no **mouse-move-event** inside Panda3D, we have to check every frame whether the cursor has moved and then use its movement to rotate the camera. To be able to calculate the amount of movement from the current cursor position, we need a fix point to where we set the cursor back every frame. I have chosen to use the middle of the window for this purpose, so we get the amount of movement by subtracting the half of the window width/height from the X/Y coordinate of the cursor.
We need to know the rotation angles α and β when moving the camera, so we have to hold track of them by storing them in two variables. We declare them as global signed floats:

C++ Code:

```
float alpha = 0.0f, beta = 0.0f;
```

Furthermore, we need to know the speed of the camera rotation and movement. So we define two macros:

C++ Code:

```
#define CAM_SPEED_ROT 0.05
#define CAM_SPEED_MOV 0.2
```

As I have already said, we need to check the cursor position every single frame. For this purpose, we set up a task and add it to the task manager. The code by now looks as follows (changes are printed **bold**):

```
#include <pandabase.h>
#include <pandaFramework.h>
#include <pandaSystem.h>
#include <GenericAsyncTask.h>
#include <AsyncTaskManager.h>

#define CAM_SPEED_ROT 0.05
#define CAM_SPEED_MOV 0.2

PandaFramework framework;
PT(AsyncTaskManager) taskMgr = AsyncTaskManager::get_global_ptr();
float alpha = 0.0f, beta = 0.0f;

AsyncTask::DoneStatus CamRotationTask(GenericAsyncTask * task, void * data)
{
	//Continue the task the next frame!
	return AsyncTask::DS_cont;
}

int main(int argc, char * argv[])
{
	WindowFramework * win;
	NodePath environ;

	//Open the framework & create a window
	framework.open_framework(argc, argv);
	framework.set_window_title("Panda3D Tests");
	win = framework.open_window();

	//Enable the keyboard
	win->enable_keyboard();

	//Load the environment
	environ = win->load_model(framework.get_models(),
				"models/environment");
	environ.reparent_to(win->get_render());
	environ.set_scale(0.25f);
	environ.set_pos(-8, 42, 0);

	//Add tasks to the task manager
	taskMgr->add(new GenericAsyncTask("Rotates the camera",
		&CamRotationTask, NULL));

	//Enter the main loop
	framework.main_loop();

	//Close framework & quit
	framework.close_framework();
	return 0;
}
```

The next thing we are going to do is filling the **CamRotationTask** function with some more code. Because we know the camera rotation speed and how we get the relative cursor movement since the last frame, this is quite easy:

C++ Code:

```
AsyncTask::DoneStatus CamRotationTask(GenericAsyncTask * task, void * data)
{
	int cursorX, cursorY;
```

```
        WindowFramework * win = framework.get_window(0);
        NodePath cam = win->get_camera_group();

        //Get cursor position
        cursorX = win->get_graphics_window()->get_pointer(0).get_x();
        cursorY = win->get_graphics_window()->get_pointer(0).get_y();

        //Subtract half the window size to retrieve
        //the relative cursor movement
        cursorX -= (int)(win->get_graphics_window()->get_x_size() / 2);
        cursorY -= (int)(win->get_graphics_window()->get_y_size() / 2);

        //Update the rotation angles
        alpha -= cursorX * CAM_SPEED_ROT;
        beta  -= cursorY * CAM_SPEED_ROT;

        //Apply rotation to camera
        cam.set_hpr(alpha, beta, 0.0f);

        //Reset the cursor to its default position (middle of the window)
        win->get_graphics_window()->move_pointer(0,
          (int)(win->get_graphics_window()->get_x_size() / 2),
          (int)(win->get_graphics_window()->get_y_size() / 2));

        //Continue the task the next frame!
        return AsyncTask::DS_cont;
}
```

If there is any function you do not know you can look it up in the c++ reference. Anything else should become clear by reading the comments.

Now, that is everything there is to the rotation. Let's go for the movement!

## 4.1. Moving the camera – Getting keystrokes

As we want to move the camera with the keys W, A, S and D, we have to keep track of whether these keys are pressed or not. For this purpose, we declare a global boolean array, which we call **keys_pressed** and initialize all elements with **false**:

C++ Code:

```
bool keys_pressed[4] = {false, false, false, false};
```

Furthermore, we declare an enumeration to not have to bear in mind which index of the array keeps track of which key:

C++ Code:

```
enum CAMERA_KEY{CKEY_W = 0,
                CKEY_A = 1,
                CKEY_S = 2,
                CKEY_D = 3};
```

We get keystrokes by using Panda3D's event system. Due to that the manual currently has no explanation for using events with C++, it took me a while to find something useful. But here you go. As you can see from this thread, we first need an event handler, which sets the values

in our **keys_pressed** array according to the particular event. Then, we can specify keys to call this handler via the **PandaFramework::define_key** function. I am going to use only one event handler, which determines the event by getting the name of it (**Event::get_name**) and then reacts accordingly by setting the corresponding array element to **true** or **false**.

Let's take a look at the current code:

C++ Code:

```
#include <pandabase.h>
#include <pandaFramework.h>
#include <pandaSystem.h>
#include <GenericAsyncTask.h>
#include <AsyncTaskManager.h>

#define CAM_SPEED_ROT 0.05
#define CAM_SPEED_MOV 0.2

enum CAMERA_KEY{CKEY_W = 0,
                CKEY_A = 1,
                CKEY_S = 2,
                CKEY_D = 3};

PandaFramework framework;
PT(AsyncTaskManager) taskMgr = AsyncTaskManager::get_global_ptr();
float alpha = 0.0f, beta = 0.0f;
bool keys_pressed[4] = {false, false, false, false};

void KeyEventHandler(const Event * e, void * data)
{
      //Key pressed or released?
      if(e->get_name() == "w")
            keys_pressed[CKEY_W] = true;
      else if(e->get_name() == "w-up")
            keys_pressed[CKEY_W] = false;
      else if(e->get_name() == "s")
            keys_pressed[CKEY_S] = true;
      else if(e->get_name() == "s-up")
            keys_pressed[CKEY_S] = false;
      else if(e->get_name() == "a")
            keys_pressed[CKEY_A] = true;
      else if(e->get_name() == "a-up")
            keys_pressed[CKEY_A] = false;
      else if(e->get_name() == "d")
            keys_pressed[CKEY_D] = true;
      else if(e->get_name() == "d-up")
            keys_pressed[CKEY_D] = false;
}

AsyncTask::DoneStatus CamRotationTask(GenericAsyncTask * task, void * data)
{
      int cursorX, cursorY;
      WindowFramework * win = framework.get_window(0);
      NodePath cam = win->get_camera_group();

      //Get cursor position
      cursorX = win->get_graphics_window()->get_pointer(0).get_x();
      cursorY = win->get_graphics_window()->get_pointer(0).get_y();

      //Subtract half the window size to retrieve
```

```
        //the relative cursor movement
        cursorX -= (int)(win->get_graphics_window()->get_x_size() / 2);
        cursorY -= (int)(win->get_graphics_window()->get_y_size() / 2);

        //Update the rotation angles
        alpha -= cursorX * CAM_SPEED_ROT;
        beta  -= cursorY * CAM_SPEED_ROT;

        //Apply rotation to camera
        cam.set_hpr(alpha, beta, 0.0f);

        //Reset the cursor to its default position (middle of the window)
        win->get_graphics_window()->move_pointer(0,
                    (int)(win->get_graphics_window()->get_x_size() / 2),
                    (int)(win->get_graphics_window()->get_y_size() / 2));

        //Continue the task the next frame!
        return AsyncTask::DS_cont;
}


int main(int argc, char * argv[])
{
        WindowFramework * win;
        NodePath environ;

        //Open the framework & create a window
        framework.open_framework(argc, argv);
        framework.set_window_title("Panda3D Tests");
        win = framework.open_window();

        //Enable the keyboard
        win->enable_keyboard();

        //Load the environment
        environ = win->load_model(framework.get_models(),
                                  "models/environment");
        environ.reparent_to(win->get_render());
        environ.set_scale(0.25f);
        environ.set_pos(-8, 42, 0);

        //Define keys
        framework.define_key("w", "", &KeyEventHandler, NULL);
        framework.define_key("s", "", &KeyEventHandler, NULL);
        framework.define_key("a", "", &KeyEventHandler, NULL);
        framework.define_key("d", "", &KeyEventHandler, NULL);

        framework.define_key("w-up", "", &KeyEventHandler, NULL);
        framework.define_key("s-up", "", &KeyEventHandler, NULL);
        framework.define_key("a-up", "", &KeyEventHandler, NULL);
        framework.define_key("d-up", "", &KeyEventHandler, NULL);

        //Add tasks to the task manager
        taskMgr->add(new GenericAsyncTask("Rotates the camera",
                     &CamRotationTask, NULL));

        //Enter the main loop
        framework.main_loop();

        //Close framework & quit
        framework.close_framework();
        return 0;
}
```

Within the **KeyEventHandler** function we get the name of the Event and compare it against the name of the key events e.g. **"w"** and **"w-up"**. Depending on the result of the comparisons we set the corresponding element of the **keys_pressed** array to either **true** or **false**. Note that the comparison of the strings with the **== operator** only works, because **Event::get_name** returns an **std::string** for which the **== operator** has been overloaded. Normal **char arrays** cannot be compared this way (before you shout at me - of course they can. But, as the more experienced programmers know, it does something else than what most newbies assume it does).

## 4.2 Moving the camera – Let's move it!

Diiingggg! Final round, guys! We are almost done with our camera system.

The last thing to do is to use the knowledge we got in the maths section of the tutorial and the information stored in the **keys_pressed** array to implement a task which, every frame, moves the camera into the right direction.

First, we code a new task function, like the **CamRotationTask** function we used to rotate the camera via the mouse, and add it to the task manager by calling:

C++ Code:

```
taskMgr->add(new GenericAsyncTask("Moves the camera",
             &CamMovementTask, NULL));
```

Now, let's code the task function and fill it with some life:

C++ Code:

```
AsyncTask::DoneStatus CamMovementTask(GenericAsyncTask * task, void * data)
{
	//Move the camera
	NodePath cam = framework.get_window(0)->get_camera_group();
	float newX, newY, newZ;

	newX = cam.get_x();
	newY = cam.get_y();
	newZ = cam.get_z();

	if(keys_pressed[CKEY_W])
	{
		newX -= sin(DEGTORAD(alpha)) * CAM_SPEED_MOV;
		newY += cos(DEGTORAD(alpha)) * CAM_SPEED_MOV;
		newZ += sin(DEGTORAD(beta))  * CAM_SPEED_MOV;
	}
	else if(keys_pressed[CKEY_S])
	{
		newX += sin(DEGTORAD(alpha)) * CAM_SPEED_MOV;
		newY -= cos(DEGTORAD(alpha)) * CAM_SPEED_MOV;
		newZ -= sin(DEGTORAD(beta))  * CAM_SPEED_MOV;
	}
```

```
        if(keys_pressed[CKEY_A])
        {
                newX -= cos(DEGTORAD(alpha)) * CAM_SPEED_MOV;
                newY -= sin(DEGTORAD(alpha)) * CAM_SPEED_MOV;
        }
        else if(keys_pressed[CKEY_D])
        {
                newX += cos(DEGTORAD(alpha)) * CAM_SPEED_MOV;
                newY += sin(DEGTORAD(alpha)) * CAM_SPEED_MOV;
        }

        //Translate the camera
        cam.set_pos(newX, newY, newZ);

        return AsyncTask::DS_cont;
}
```

I guess there is some need of explanation, now. At first you may wonder, why I initialize **newX**, **newY** and **newZ** with the current camera position. Go back and take a look at the two sketches again that we have made to develop the formulas for the forward-movement. You will notice that the X, Y and Z components of the movement are relative to the current camera position. That is why I initially set the new position components to the current position and then add (or subtract – depends on the formula for the movement) the respective component to it. Thus, I get the absolute new position of the camera.

Moreover, you could ask what **DEGTORAD** is. It is a macro I defined. It simply does what it says: Convert degrees to radians. This is especially useful, when working with the C++ trigonometric functios **sin**, **cos**, **tan** and so on, because they expect their parameters to be in radians, while we humans can better work with degrees. However, **DEGTORAD** is defined as:

C++ Code:

```
#define DEGTORAD(x) x * 0.01745329252
```

The explanation for this is that we have to multiply a value in degrees with $\frac{\pi}{180}$ to convert it into radians. And $\frac{\pi}{180} \approx 0{,}01745329252$.

Taking a look at the formulas again, there should be no need of further explanation. If there is: Tell me!

## Epilogue

Now, that is everything I have to say about “Developing a First Person Camera System with Panda3D and C++”. I hope that you found it helpful or, at least, that you have not the impression that this tutorial wasted your time ;)

Feel engaged to comment on my work posting your opinion here.

Look at Appendix B to get the whole source code.

## Appendix A – The formulas again:

### 1. Forward-movement

$$x_{comp} = -\sin\alpha \times speed_{cam}$$

$$y_{comp} = \cos\alpha \times speed_{cam}$$

$$z_{comp} = \sin\beta \times speed_{cam}$$

### 2. Backward-movement

$$x_{comp} = \sin\alpha \times speed_{cam}$$

$$y_{comp} = -\cos\alpha \times speed_{cam}$$

$$z_{comp} = -\sin\beta \times speed_{cam}$$

### 3. Left-movement

$$x_{comp} = -\cos\alpha \times speed_{cam}$$

$$y_{comp} = -\sin\alpha \times speed_{cam}$$

### 4. Right-movement

$$x_{comp} = \cos\alpha \times speed_{cam}$$

$$y_{comp} = \sin\alpha \times speed_{cam}$$

## Appendix B – The whole source code:

```
#include <pandabase.h>
#include <pandaFramework.h>
#include <pandaSystem.h>
#include <GenericAsyncTask.h>
#include <AsyncTaskManager.h>

#define CAM_SPEED_ROT 0.05
#define CAM_SPEED_MOV 0.2
#define DEGTORAD(x) x * 0.01745329252

enum CAMERA_KEY{CKEY_W = 0,
                CKEY_A = 1,
                CKEY_S = 2,
                CKEY_D = 3};

PandaFramework framework;
PT(AsyncTaskManager) taskMgr = AsyncTaskManager::get_global_ptr();
float alpha = 0.0f, beta = 0.0f;
bool keys_pressed[4] = {false, false, false, false};

void KeyEvent(const Event * e, void * data)
{
      //Key pressed or released?
      if(e->get_name() == "w")
            keys_pressed[CKEY_W] = true;
      else if(e->get_name() == "w-up")
            keys_pressed[CKEY_W] = false;
      else if(e->get_name() == "s")
            keys_pressed[CKEY_S] = true;
      else if(e->get_name() == "s-up")
            keys_pressed[CKEY_S] = false;
```

```
	else if(e->get_name() == "a")
		keys_pressed[CKEY_A] = true;
	else if(e->get_name() == "a-up")
		keys_pressed[CKEY_A] = false;
	else if(e->get_name() == "d")
		keys_pressed[CKEY_D] = true;
	else if(e->get_name() == "d-up")
		keys_pressed[CKEY_D] = false;
}

AsyncTask::DoneStatus CamRotationTask(GenericAsyncTask * task, void * data)
{
	int cursorX, cursorY;
	WindowFramework * win = framework.get_window(0);
	NodePath cam = win->get_camera_group();

	//Get cursor position
	cursorX = win->get_graphics_window()->get_pointer(0).get_x();
	cursorY = win->get_graphics_window()->get_pointer(0).get_y();

	//Subtract half the window size to retrieve
	//the relative cursor movement
	cursorX -= (int)(win->get_graphics_window()->get_x_size() / 2);
	cursorY -= (int)(win->get_graphics_window()->get_y_size() / 2);

	//Update the rotation angles
	alpha -= cursorX * CAM_SPEED_ROT;
	beta  -= cursorY * CAM_SPEED_ROT;

	//Apply rotation to camera
	cam.set_hpr(alpha, beta, 0.0f);

	//Reset the cursor to its default position (middle of the window)
	win->get_graphics_window()->move_pointer(0,
			(int)(win->get_graphics_window()->get_x_size() / 2),
			(int)(win->get_graphics_window()->get_y_size() / 2));

	//Continue the task the next frame!
	return AsyncTask::DS_cont;
}

AsyncTask::DoneStatus CamMovementTask(GenericAsyncTask * task, void * data)
{
	//Move the camera
	NodePath cam = framework.get_window(0)->get_camera_group();
	float newX, newY, newZ;

	newX = cam.get_x();
	newY = cam.get_y();
	newZ = cam.get_z();

	if(keys_pressed[CKEY_W])
	{
		newX -= sin(DEGTORAD(alpha)) * CAM_SPEED_MOV;
		newY += cos(DEGTORAD(alpha)) * CAM_SPEED_MOV;
		newZ += sin(DEGTORAD(beta))  * CAM_SPEED_MOV;
	}
	else if(keys_pressed[CKEY_S])
	{
		newX += sin(DEGTORAD(alpha)) * CAM_SPEED_MOV;
		newY -= cos(DEGTORAD(alpha)) * CAM_SPEED_MOV;
		newZ -= sin(DEGTORAD(beta))  * CAM_SPEED_MOV;
	}
```

```
	if(keys_pressed[CKEY_A])
	{
		newX -= cos(DEGTORAD(alpha)) * CAM_SPEED_MOV;
		newY -= sin(DEGTORAD(alpha)) * CAM_SPEED_MOV;
	}
	else if(keys_pressed[CKEY_D])
	{
		newX += cos(DEGTORAD(alpha)) * CAM_SPEED_MOV;
		newY += sin(DEGTORAD(alpha)) * CAM_SPEED_MOV;
	}

	//Translate the camera
	cam.set_pos(newX, newY, newZ);

	return AsyncTask::DS_cont;
}


int main(int argc, char * argv[])
{
	WindowFramework * win;
	NodePath environ;

	//Open the framework & create a window
	framework.open_framework(argc, argv);
	framework.set_window_title("Panda3D Tests");
	win = framework.open_window();

	//Enable the keyboard
	win->enable_keyboard();

	//Load the environment
	environ = win->load_model(framework.get_models(),
				"models/environment");
	environ.reparent_to(win->get_render());
	environ.set_scale(0.25f);
	environ.set_pos(-8, 42, 0);

	//Define keys
	framework.define_key("w", "", &KeyEvent, NULL);
	framework.define_key("s", "", &KeyEvent, NULL);
	framework.define_key("a", "", &KeyEvent, NULL);
	framework.define_key("d", "", &KeyEvent, NULL);

	framework.define_key("w-up", "", &KeyEvent, NULL);
	framework.define_key("s-up", "", &KeyEvent, NULL);
	framework.define_key("a-up", "", &KeyEvent, NULL);
	framework.define_key("d-up", "", &KeyEvent, NULL);

	//Add tasks to the task manager
	taskMgr->add(new GenericAsyncTask("Rotates the camera",
			&CamRotationTask, NULL));
	taskMgr->add(new GenericAsyncTask("Moves the camera",
			&CamMovementTask, NULL));

	//Enter the main loop
	framework.main_loop();

	//Close framework & quit
	framework.close_framework();
	return 0;
}
```