Open Scene Graph DirectInput Joystick control

In this exercise you will add in joystick control using DirectInput. I have created a joystick control class to make it easier for you to use, but you can program directly yourself using DirectInput if you wish.

- Create a new Win32 console project called *OSG Joystick*
- Set the paths for the osg include and lib folders
- Add the following libraries to the linker input : osgd.lib osgviewerd.lib osgdbd.lib osgGAd.lib dinput8.lib
- Make sure you have downloaded the resources and put them in a "models" folder inside your project

Set up the headers and add include files

You will see some extra files in the resources folder. The *joystickControlCustom.h* file is one I have created for you to handle the joystick events and pass an event code back to your main program.

The *directInputRegistry.cpp* file is an OSG file which gives code to control a variety of devices. You don't need to use this file directly but it is called by the joystickControl code. You must include it (and the associated .h file) in your project folder and also ADD it to your source files in your project (if you are not sure how to do this, please ask).

- You will need to add the following header declarations to your project:

```
#include "stdafx.h"
#include <osgViewer/Viewer>
#include <osgDB/ReadFile>
#include <osg/Node>
#include <osg/PositionAttitudeTransform>
#include <osg/ref_ptr>

#include <InitGuid.h>
#include <dinput.h>

#include "joystickControl.h"
```

- Build the project before adding any other code to make sure that all the files are added correctly

- Next add the following code

Create geometry and display in the viewer

```
int _tmain(int argc, _TCHAR* argv[])
{
	// Declare a node which will serve as the root node
	// the rest of the scene "hangs" on this node

	osg::ref_ptr<osg::Group>  myRoot  = new osg::Group();

	//*************************  load an object in the scene
*********************//

	osg::ref_ptr<osg::Node> blueHex = osgDB::readNodeFile("models/lBlueHex.3ds");
	osg::ref_ptr<osg::PositionAttitudeTransform>  blueXform = new
osg::PositionAttitudeTransform();
	blueXform->setPosition(osg::Vec3(0,1,0));
	blueXform->addChild(blueHex.get());
	myRoot->addChild(blueXform.get()); // adds it all to the root node


	//*****************************************************************//



		//*******set up the custom viewer to display your
scene****************//

	CustomViewer viewer; // this viewer is defined in the joystickControl file. It
allows directInput to be recognised

	viewer.setSceneData( myRoot.get()); //assign the scene graph we created above
to this viewer

	viewer.setLightingMode(osg::View::LightingMode::HEADLIGHT); // give the scene
some lighting

	viewer.getCamera()->setClearColor(osg::Vec4(1.0,1.0,1.0,1.0)); // sets
background colour of scene

	viewer.getCamera()-
>setViewMatrixAsLookAt(osg::Vec3(0.0,0.0,0),osg::Vec3(0.0,1.0,0), osg::Vec3(0.0, 0.0,
1.0));


	//*************************************************************//


	viewer.realize(); // start the viewer


	//************ add a title to the window ***********************//

	typedef osgViewer::Viewer::Windows Windows;
	Windows windows;

	viewer.getWindows(windows);
	windows[0]->setWindowName("UoP osg direct input example");

	//****************************************//
```

```
	while (!viewer.done()) // until the end of the program
	{

		viewer.frame(); // update to next frame

	}


	return 0;
}
```

- Build and run the project. You should have a static scene with the blue hex box displayed.

Adding a joystick controller

Now we are going to add joystick control to the box. We will create a *joystickStateType* (which is defined in the *joystickControl* code) and add an event handler to respond to any joystick input events.

- Add the following code after the camera setting code

```
viewer.getCamera()->setViewMatrixAsLookAt(osg::Vec3(0.0,0.0,0),osg::Vec3(0.0,1.0,0),
osg::Vec3(0.0, 0.0, 1.0));


	//*************************************************************//


	//********************** set up direct input *************//

	joystickStateType *myJoystickState = new joystickStateType();
	JoystickHandler *myJoystickHandler = new JoystickHandler (myJoystickState);
    viewer.addEventHandler( myJoystickHandler );


	//*********************************************************//
```

- Test the code to make sure there are no compilation errors

At the moment the joystick does nothing. We need to create a function to make use of the joystick input. We will create a simple function which reads the code from the joystick event and moves or scales the box.

- Add the following function declaration at the beginning of the program, before the *main* function

```
#include "joystickControl.h"

void processJoystick(int inputCode, osg::ref_ptr<osg::PositionAttitudeTransform>
moveTransform );


int _tmain(int argc, _TCHAR* argv[])
{


viewer.frame(); // update to next frame

	}


	return 0;
}
```

- Now add this function definition at the end of the program, after the *main* function

```
void processJoystick(int inputCode, osg::ref_ptr<osg::PositionAttitudeTransform>
moveTransform ) // function to take some action in response to joystick input
{
	osg::Vec3 tempPos = moveTransform->getPosition(); // get the current object
position and place it in a temporary vector

	switch(inputCode)
	{
	case -998: // left
		tempPos[0] -= 0.01; // subtract from the x axis
		break;
	case 998:  // right
		tempPos[0] += 0.01; // add to the x axis
		break;
	case 999: // up
		tempPos[2] -= 0.01; // subtract from the z axis
		break;
	case -999: // down arrow key
		tempPos[2] += 0.01; // add to the z axis
		break;
	case 0:
		moveTransform->setScale(osg::Vec3(0.5,0.5,0.5));
		break;
	case 1:
		moveTransform->setScale(osg::Vec3(1,1,1));
		break;
		break;
	}
	moveTransform->setPosition(tempPos);
}
```

- You will also need to add a function declaration at the start of your code, **before** the main program loop

```
include <InitGuid.h>
#include <dinput.h>

#include "joystickControl.h"

void processJoystick(int inputCode, osg::ref_ptr<osg::PositionAttitudeTransform>
moveTransform );


int _tmain(int argc, _TCHAR* argv[])
{
```

- Finally we need to call the function every frame, so add this code inside the *while* loop which updates the frames

```
while (!viewer.done()) // until the end of the program
	{ if (myJoystickState->joystickInput)// if there has been a joystick event -
this gets checked every frame
		{
			processJoystick(myJoystickState->button, blueXform.get()); //
function to act on the input
			myJoystickState->joystickInput = false; // reset the state to
wait for next event
			myJoystickState->button = -1; // reset the variable for next
input
		}
		viewer.frame(); // update to next frame
```

- Build and run the project. You should now be able to move the box with the joystick, and also use button presses to resize the box.

Important note: Some of the joysticks we use are non-standard and don't map to the same buttons and controls. You will be familiar with customising input from your work in D-flow.

If you are using a non-standard joystick (and the controls above don't work as expected) you will need to customise the code for your joystick. I have written an amendment for you on this occasion, but it is not complete and will not map all the buttons for you.

If necessary, change your switch statement to use this code

```
            // The following cases are customised for the UoP APPVR controllers as
they are non-standard. Case 0 – 3 are the right joystick
      case 0: // up
            tempPos[2] += 0.01; // add to the z axis
            break;
      case 1: // right
          tempPos[0] += 0.01; // add to the x axis
          break;
      case 2: // down
          tempPos[2] -= 0.01; // subtract from the z axis
          break;
      case 3: // left
            tempPos[0] -= 0.01; // subtract from the x axis
            break;
      case 4: // left button 1 on front edge
            moveTransform->setScale(osg::Vec3(0.5,0.5,0.5));
            break;
      case 5: // right button 1 on front edge
            moveTransform->setScale(osg::Vec3(1,1,1));
            break;
            break;
      }
      moveTransform->setPosition(tempPos);
```

Experiment with changing the settings, adding other buttons etc.

Linking the joystick to a camera class

Until now we have only looked at very basic camera settings. Precise control of a camera is complex as it has six degrees of freedom, and as soon as it has rotated around at least one axis, calculations are needed to work out the new "forwards" "backwards" etc.

I have written a basic class to handle the complexities of camera control. I cannot guarantee it is bug-free! We are going to use it to link the joystick to the camera.

- Either work from the existing project, or create a new project as a duplicate of the joystick project

- Edit the library list in the linker to include the following libraries: osgd.lib osgviewerd.lib osgdbd.lib osgUtild.lib osgGAd.lib dinput8.lib

- Add the following header to your project

```
#include "joystickControl.h"

#include "manualCamera.h"
```

- Add the *manualCamera.cpp* file to the **source files** of your project and the *manualCamera.h* file to the **header files** of your project

- Now add the following line below the function definition to declare a new instance of our manual camera control

```
void processJoystick(int inputCode, osg::ref_ptr<osg::PositionAttitudeTransform>
moveTransform );

manualCamera* myCamControl; // camera control object to update the camera from manual
input

int _tmain(int argc, _TCHAR* argv[])
```

- Add the following code at the beginning of the main function to create an instance of manual camera and also to define the variables for the camera settings

```
int _tmain(int argc, _TCHAR* argv[])
{

	myCamControl = new manualCamera(); // create a new camera control

	osg::Vec3d eye; // sets the x y z position of the camera
	osg::Vec3d centre; // sets the "target" of the camera
	osg::Vec3d up; // sets the orientation of the camera

	// Declare a node which will serve as the root node
	// the rest of the scene "hangs" on this node

	osg::ref_ptr<osg::Group>  myRoot  = new osg::Group();
```

- Next we will add some terrain to our scene using the following code (make sure that the correct resources are in the models folder in your project folder)

```
osg::ref_ptr<osg::Group>  myRoot  = new osg::Group();


	osg::Node* sceneNode = osgDB::readNodeFile("models/pathHillFlat.3ds"); //read
in from file the scene to use
	myRoot->addChild(sceneNode);

	//*************************  load an object in the scene
*********************//

	osg::ref_ptr<osg::Node> blueHex = osgDB::readNodeFile("models/lBlueHex.3ds");
```

- Test the project to make sure the terrain loads correctly

- If you prefer you can change the lighting settings to give more global light)

```
viewer.setLightingMode(osg::View::LightingMode::SKY_LIGHT); // give the scene some
global lighting
```

- Now add a new function declaration at the top of your project.

```
void processJoystick(int inputCode, osg::ref_ptr<osg::PositionAttitudeTransform>
moveTransform );
void joystickCamControl(int inputCode);
```

- Add the following function at the end of your project

```
return 0;


}


void joystickCamControl(int inputCode) // function to control the camera with the
joystick
{
	switch(inputCode)
	{
		case -998: // left
		//updateYaw(-1); // swing to left
		myCamControl->updateYaw(-1);
		break;
		case 998:  // right
		//updateYaw(1);
		myCamControl->updateYaw(1);
		break;
		case 999: // up
		myCamControl->updatePitch(1); // pitch up 1 degree
		break;
```

```
                case -999: // down
                myCamControl->updatePitch(-1); // pitch down 1 degree
                break;
                case 0:
                //move forward along heading
                myCamControl->updateY(0.05);
                break;
                case 1:
                //move backward along heading
                myCamControl->updateY(-0.05);
                break;
                break;
        }
}
```

You will notice that this function is very similar to the one used to control the box with the joystick. This is because it is essentially doing the same thing – processing the joystick events and passing them on as control to another object.

IMPORTANT NOTE: If you are using one of the non-standard controllers you will need to change this switch statement in a similar way to the previous exercise. Try and work it out for yourself.

It is important that you don't have the joystick trying to control both the camera and the blue box or you will see very strange behaviour, so you may like to comment out or delete the previous functions before continuing.

- Amend the *while* loop in the *main* program to call the new function instead of the old one

```
if (myJoystickState->joystickInput)// if there has been a joystick event - this gets
checked every frame
                {
                        //processJoystick(myJoystickState->button, blueXform.get()); //
function to act on the input
                        joystickCamControl(myJoystickState->button); // function to
control the camera with  the joystick
                        myJoystickState->joystickInput = false; // reset the state to
wait for next event
                        myJoystickState->button = -1; // reset the variable for next
input
                }

                viewer.frame(); // update to next frame
```

- Build and test the code again.

You will find that **the joystick does not appear to control anything anymore**. This is because we need to tell the program to update the camera view to the new settings.

- Add the code to update the camera position and orientation after each input event

```
joystickCamControl(myJoystickState->button); // function to control the camera with
the joystick
                    myJoystickState->joystickInput = false; // reset the state to
wait for next event
                    myJoystickState->button = -1; // reset the variable for next
input
                    viewer.getCamera()->getViewMatrixAsLookAt(eye, centre, up);
                    myCamControl->setNewViewPos(sceneNode,eye, centre, up);
                    viewer.getCamera()->setViewMatrixAsLookAt(eye, centre, up);
            }

            viewer.frame(); // update to next frame
```

You will notice that we do not have full camera control yet in this function. See if you can work out how to add additional control on different buttons. Some commands you might find useful are:

myCamControl->updateX / updateY / updateZ / updateRoll

myCamControl->setTerrainFollow(*value for height above ground*)

mCamControl->noTerrainFollow()

myCamControl->setCollision(*value for hit distance*)