CS 378: Computer Game Technology

Collision Detection and More Physics

Spring 2012

Initialize Configuration

```
mRoot = new Ogre::Root(mPluginsCfg);
// Load resource paths from config file
Ogre::ConfigFile cf;
cf.load(mResourcesCfg);
// Parse all sections & settings in the file
Ogre::ConfigFile::SectionIterator seci = cf.getSectionIterator();
Ogre::String secName, typeName, archName;
while (seci.hasMoreElements()) {
        secName = seci.peekNextKey();
        Ogre::ConfigFile::SettingsMultiMap *settings = seci.getNext();
        Ogre::ConfigFile::SettingsMultiMap::iterator i;
        for (i = settings->begin(); i != settings->end(); ++i) {
                typeName = i->first;
                archName = i->second;
                Ogre::ResourceGroupManager::getSingleton().
                        addResourceLocation(archName, typeName, secName);
        }
}
// Show the configuration dialog and initialize the system
if((mRoot->restoreConfig() || mRoot->showConfigDialog())) {
        //If initialized, create a render window
        mWindow = mRoot->initialise(true, "Demo App");
}
else { return false; }
```

Initialize Resource Mgr and Scene

```
// Set default mipmap level (note: some APIs ignore this)
Ogre::TextureManager::getSingleton().setDefaultNumMipmaps(5);
// initialise all resource groups
Ogre::ResourceGroupManager::getSingleton().initialiseAllResourceGroups();

// Create the SceneManager, in this case a generic one
mSceneMgr = mRoot->createSceneManager("DefaultSceneManager");

// Create a scene
mSceneMgr->setSkyBox(true, "Examples/MorningSkyBox", 5000, false);
PlayingField* bCourt = new PlayingField(mSceneMgr);
ball = new Ball(mSceneMgr);
bCourt->addChild(ball->getNode());
ball->setPlayingField(bCourt);
```

Setting up a Camera

```
// Create the camera
mCamera = mSceneMgr->createCamera("PlayerCam");
// Position it near front of the room
mCamera->setPosition(Ogre::Vector3(0.0f, 0.0f, 0.4f * bCourt->getLength()));
// Look at the origin
mCamera->lookAt(Ogre::Vector3(0,0,0));
mCamera->setNearClipDistance(5);
// create a default camera controller
mCameraMan = new OgreBites::SdkCameraMan(mCamera);
// Create one viewport, entire window
Ogre::Viewport* vp = mWindow->addViewport(mCamera);
vp->setBackgroundColour(Ogre::ColourValue(0,0,0));
// Alter the camera aspect ratio to match the viewport
mCamera->setAspectRatio(
Ogre::Real(vp->getActualWidth()) / Ogre::Real(vp->getActualHeight()));
```

Mouse and Keyboard Input

```
#include <OISEvents.h>
#include <OISInputManager.h>
#include <OISKeyboard.h>
#include <OISMouse.h>
#include <OgreRenderWindow.h>
#include <SdkTrays.h>
#include <SdkCameraMan.h>
class ControlListener : public OIS::KeyListener, public OIS::MouseListener {
protected:
        OIS::InputManager* inputManager;
        OIS::Mouse* mouse;
        OIS::Keyboard* keyboard;

        Ogre::RenderWindow* window;
        Ogre::Camera* camera;
        OgreBites::SdkTrayManager* trayManager;
        OgreBites::SdkCameraMan* cameraManager;
        OgreBites::ParamsPanel* detailsPanel;
        bool exitCmd;
```

Mouse and Keyboard Input

```
protected:
        // OIS::KeyListener
        virtual bool keyPressed( const OIS::KeyEvent &arg );
        virtual bool keyReleased( const OIS::KeyEvent &arg );
        // OIS::MouseListener
        virtual bool mouseMoved( const OIS::MouseEvent &arg );
        virtual bool mousePressed( const OIS::MouseEvent &arg, OIS::MouseButtonID id );
        virtual bool mouseReleased( const OIS::MouseEvent &arg, OIS::MouseButtonID id );
public:
            ControlListener(Ogre::RenderWindow* window);
            ~ControlListener();
            void detach();
            void update();
            void resizeWindow(unsigned int width, unsigned int height);
            void setCamera(Ogre::Camera* cam);
            void setTrayManager(OgreBites::SdkTrayManager* tM);
            void setCameraManager(OgreBites::SdkCameraMan* cM);
            void setDetailsPanel(OgreBites::ParamsPanel* pp);
            OIS::Mouse* getMouse() { return mouse; }
            bool exit() { return exitCmd; }
};
```

A Game Object

```
#include <Ogre.h>
#include "PlayingField.h"
class Ball {
protected:
        Ogre::SceneNode* rootNode;
        Ogre::Real bRadius;
        Ogre::Vector3 bDirection;
        Ogre::Real bSpeed;
        PlayingField* grounds;
public:
        Ball(Ogre::SceneManager* scnMgr);
        ~Ball();
        void move(const Ogre::FrameEvent& evt);
        Ogre::SceneNode* getNode() { return rootNode; }
        void setPlayingField(PlayingField * pf) { grounds = pf; }
};
```

A Game Object

```
#include <OgreEntity.h>
#include <OgreSceneManager.h>
#include "Ball.h"
Ball::Ball(Ogre::SceneManager* scnMgr) {
        Ogre::Entity* ball = scnMgr->createEntity("Sphere", "sphere.mesh");
        ball->setMaterialName("BallColor/CubeMap");
        ball->setCastShadows(true);
        rootNode = scnMgr->createSceneNode("Ball");
        rootNode->attachObject(ball);
        rootNode->scale(0.1f,0.1f,0.1f);
        bRadius = 10.0f;
        bDirection = Ogre::Vector3(1.0f, 2.0f, 3.0f);
        bDirection.normalise();
        bSpeed = 250.0f;
}
void Ball::move(const Ogre::FrameEvent& evt) {
        Ogre::Vector3 bPosition = rootNode->getPosition();
        if (bPosition.y < -grounds->getHeight()/2.0f + bRadius && bDirection.y < 0.0f) bDirection.y = -bDirection.y;
        if (bPosition.y > grounds->getHeight()/2.0f - bRadius && bDirection.y > 0.0f) bDirection.y = -bDirection.y;
        if (bPosition.z < -grounds->getLength()/2.0f + bRadius && bDirection.z < 0.0f) bDirection.z = -bDirection.z;
        if (bPosition.z > grounds->getLength()/2.0f - bRadius && bDirection.z > 0.0f) bDirection.z = -bDirection.z;
        if (bPosition.x < -grounds->getWidth()/2.0f + bRadius && bDirection.x < 0.0f) bDirection.x = -bDirection.x;
        if (bPosition.x > grounds->getWidth()/2.0f - bRadius && bDirection.x > 0.0f) bDirection.x = -bDirection.x;
        rootNode->translate(bSpeed * evt.timeSinceLastFrame * bDirection);
}
```

Rendering Loop

```
class MyFrameListener : public Ogre::FrameListener {

-----------------------------------------------------------

// Rendering loop
                mRoot->startRendering();

-----------------------------------------------------------

bool Assignment2::frameRenderingQueued(const Ogre::FrameEvent& evt) {
   if(mWindow->isClosed()) return false;
                mControls->update();
                if (mControls->exit()) return false;
                mGUI->frameRenderingQueued(evt);
                if (!mGUI->isDialogVisible()) {
                                // if dialog isn't up, then update the scene
                                mCameraMan->frameRenderingQueued(evt);
                                // Move the ball
                                ball->move(evt);
                }
   return true;
}
```