```
#ifndef _Ogre_H_
#include <ogre.h>
#endif
#include "OIS.h"
#include <CEGUI.h>
#include <CEGUIOgreRenderer.h>
#include <Terrain\include\OgreTerrainGroup.h>
#include <Terrain\include\OgreTerrain.h>
#include "btBulletDynamicsCommon.h"
#include "btHeightfieldTerrainShape.h"
#include "CEGUIManager.h"
using namespace Ogre;


   //function defs
   ManualObject* createCubeMesh(Ogre::String name, Ogre::String matName);


   // this pattern updates the scenenode position when it changes within the bullet simulation
   // taken from BulletMotionState docs page24
   class MyMotionState : public btMotionState {
   public:
      MyMotionState(const btTransform &initialpos, Ogre::SceneNode *node) {
         mVisibleobj = node;
         mPos1 = initialpos;
      }
      virtual ~MyMotionState() {    }
      void setNode(Ogre::SceneNode *node) {
         mVisibleobj = node;
      }
      virtual void getWorldTransform(btTransform &worldTrans) const {
         worldTrans = mPos1;
      }
      virtual void setWorldTransform(const btTransform &worldTrans) {
         if(NULL == mVisibleobj) return; // silently return before we set a node
         btQuaternion rot = worldTrans.getRotation();
         mVisibleobj->setOrientation(rot.w(), rot.x(), rot.y(), rot.z());
         btVector3 pos = worldTrans.getOrigin();
         // TODO **** XXX need to fix this up such that it renders properly since this doesnt know the scale of the node
         // also the getCube function returns a cube that isnt centered on Z
         mVisibleobj->setPosition(pos.x(), pos.y()+5, pos.z()-5);
      }
   protected:
      Ogre::SceneNode *mVisibleobj;
      btTransform mPos1;
   };


   class Application : public FrameListener, public OIS::KeyListener, public OIS::MouseListener
   {
   public:
      void go()
      {
        mContinue = true;
         createRoot();
         defineResources();
         setupRenderSystem();
         createRenderWindow();
         initializeResourceGroups();
         setupScene();
         createBulletSim();
         setupInputSystem();
         setupCEGUI();
         createFrameListener();
         startRenderLoop();

      }

      ~Application()
      {
         mInputManager->destroyInputObject(mKeyboard);
                                    mInputManager->destroyInputObject(mMouse);
         OIS::InputManager::destroyInputSystem(mInputManager);
```

```
        delete mRoot;

        // cleanup bulletdyanmics

       //cleanup in the reverse order of creation/initialization
       //remove the rigidbodies from the dynamics world and delete them
       for (i=dynamicsWorld->getNumCollisionObjects()-1; i>=0 ;i--)
       {
         btCollisionObject* obj = dynamicsWorld->getCollisionObjectArray()[i];
         btRigidBody* body = btRigidBody::upcast(obj);
         if (body && body->getMotionState())
         {
           delete body->getMotionState();
         }
         dynamicsWorld->removeCollisionObject( obj );
         delete obj;
       }

       //delete collision shapes
       for (int j=0;j<collisionShapes.size();j++)
       {
         btCollisionShape* shape = collisionShapes[j];
         collisionShapes[j] = 0;
         delete shape;
       }

       delete dynamicsWorld;
       delete solver;
       delete overlappingPairCache;
       delete dispatcher;
       delete collisionConfiguration;
    }

  private:
     Root *mRoot;
     SceneManager *mSceneMgr;
     OIS::Keyboard *mKeyboard;
                          OIS::Mouse *mMouse;
     OIS::InputManager *mInputManager;
                          CEGUIManager* ceguiManager;
     bool mContinue;
                Ogre::TerrainGlobalOptions* mTerrainGlobals;
     Ogre::TerrainGroup* mTerrainGroup;
     bool mTerrainsImported;
       // scene objects
                          ManualObject *cmo;


     // bullet dynamics
      int i;
      btDefaultCollisionConfiguration* collisionConfiguration;
      btCollisionDispatcher* dispatcher;
      btBroadphaseInterface* overlappingPairCache;
      btSequentialImpulseConstraintSolver* solver;
      btDiscreteDynamicsWorld* dynamicsWorld;
      btCollisionShape* groundShape;
      btAlignedObjectArray<btCollisionShape*> collisionShapes;


       // frame listener
     bool frameStarted(const FrameEvent &evt)
      {
       mKeyboard->capture();
                            mMouse->capture();
         // update physics simulation
       //dynamicsWorld->stepSimulation(evt.timeSinceLastFrame,10);
                            dynamicsWorld->stepSimulation(evt.timeSinceLastFrame);
        return mContinue;
      }

     // KeyListener
```

```
bool keyPressed(const OIS::KeyEvent &e) {
  switch (e.key) {
        case OIS::KC_ESCAPE:
          mContinue = false;
           break;
        default:
                                                        CEGUI::System &sys = CEGUI::System::getSingleton();
           sys.injectKeyDown(e.key);
           sys.injectChar(e.text);
          break;
  }
  return true;
}
          bool quit(const CEGUI::EventArgs &e){
                    mContinue = false;
                    return false;
          }

bool keyReleased(const OIS::KeyEvent &e) {
                    CEGUI::System::getSingleton().injectKeyUp(e.key);
                    return true;
          }

 void createRoot()
 {
    mRoot = new Root("plugins_d.cfg","ogre.cfg","Ogre.log");
 }

 void defineResources()
 {
    String secName, typeName, archName;
    ConfigFile cf;
    cf.load("resources_d.cfg");

    ConfigFile::SectionIterator seci = cf.getSectionIterator();
    while (seci.hasMoreElements())
    {
       secName = seci.peekNextKey();
       ConfigFile::SettingsMultiMap *settings = seci.getNext();
       ConfigFile::SettingsMultiMap::iterator i;
       for (i = settings->begin(); i != settings->end(); ++i)
       {
          typeName = i->first;
          archName = i->second;
          ResourceGroupManager::getSingleton().addResourceLocation(archName, typeName, secName);
       }
    }
 }

 void setupRenderSystem()
 {
    if (!mRoot->restoreConfig() && !mRoot->showConfigDialog())
       throw Exception(52, "User canceled the config dialog!", "Application::setupRenderSystem()");

    //// Do not add this to the application
   // RenderSystem *rs = mRoot->getRenderSystemByName("Direct3D9 Rendering Subsystem");
    //                                // or use "OpenGL Rendering Subsystem"
  // mRoot->setRenderSystem(rs);
   // rs->setConfigOption("Full Screen", "No");
   // rs->setConfigOption("Video Mode", "800 x 600 @ 32-bit colour");
 }

 void createRenderWindow()
 {
    mRoot->initialise(true, "Tutorial Render Window");


 }

 void initializeResourceGroups()
 {
    TextureManager::getSingleton().setDefaultNumMipmaps(5);
    ResourceGroupManager::getSingleton().initialiseAllResourceGroups();
```

```
    }

void initBlendMaps(Ogre::Terrain* terrain)
{   Ogre::TerrainLayerBlendMap* blendMap0 = terrain->getLayerBlendMap(1);
                        Ogre::TerrainLayerBlendMap* blendMap1 = terrain->getLayerBlendMap(2);
                        Ogre::Real minHeight0 = 70;
                        Ogre::Real fadeDist0 = 40;
                        Ogre::Real minHeight1 = 70;
                        Ogre::Real fadeDist1 = 15;
                        float* pBlend0 = blendMap0->getBlendPointer();
                        float* pBlend1 = blendMap1->getBlendPointer();
                        for (Ogre::uint16 y = 0; y < terrain->getLayerBlendMapSize(); ++y)
                        {
                                for (Ogre::uint16 x = 0; x < terrain->getLayerBlendMapSize(); ++x)
                                {
                                        Ogre::Real tx, ty;

                                        blendMap0->convertImageToTerrainSpace(x, y, &tx, &ty);
                                        Ogre::Real height = terrain->getHeightAtTerrainPosition(tx, ty);
                                        Ogre::Real val = (height - minHeight0) / fadeDist0;
                                        val = Ogre::Math::Clamp(val, (Ogre::Real)0, (Ogre::Real)1);
                                        *pBlend0++ = val;

                                        val = (height - minHeight1) / fadeDist1;
                                        val = Ogre::Math::Clamp(val, (Ogre::Real)0, (Ogre::Real)1);
                                        *pBlend1++ = val;
                                }
                        }
                        blendMap0->dirty();
                        blendMap1->dirty();
                        blendMap0->update();
                        blendMap1->update();
          }

                        void getTerrainImage(bool flipX, bool flipY, Ogre::Image& img)
                        {
      img.load("terrain.png", Ogre::ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME);
      if (flipX)
        img.flipAroundY();
      if (flipY)
        img.flipAroundX();
                        }

                        void defineTerrain(long x, long y)
                        {    char str1[50];
       sprintf(str1,"defineTerrain: x=%d,y=%d",x,y);
                  Ogre::LogManager::getSingleton().logMessage(str1);
       Ogre::String filename = mTerrainGroup->generateFilename(x, y);
       if (Ogre::ResourceGroupManager::getSingleton().resourceExists(mTerrainGroup->getResourceGroup(), filename))
        {
          mTerrainGroup->defineTerrain(x, y);
        }
        else
        {
         Ogre::Image img;
         getTerrainImage(x % 2 != 0, y % 2 != 0, img);
         mTerrainGroup->defineTerrain(x, y, &img);
         mTerrainsImported = true;
        }

                        }
                        void configureTerrainDefaults(Ogre::Light* light)
                        {
                                // Configure global
                                mTerrainGlobals->setMaxPixelError(8);
                                // testing composite map
                                mTerrainGlobals->setCompositeMapDistance(3000);

                                // Important to set these so that the terrain knows what to use for derived (non-realtime) data
                                mTerrainGlobals->setLightMapDirection(light->getDerivedDirection());
                                mTerrainGlobals->setCompositeMapAmbient(mSceneMgr->getAmbientLight());
                                mTerrainGlobals->setCompositeMapDiffuse(light->getDiffuseColour());
```

```
                        // Configure default import settings for if we use imported image
                        Ogre::Terrain::ImportData& defaultimp = mTerrainGroup->getDefaultImportSettings();
                        defaultimp.terrainSize = 513;
                        defaultimp.worldSize = 12000.0f;
                        defaultimp.inputScale = 600;
                        defaultimp.minBatchSize = 33;
                        defaultimp.maxBatchSize = 65;
                        // textures
                        defaultimp.layerList.resize(3);
                        defaultimp.layerList[0].worldSize = 100;
                        defaultimp.layerList[0].textureNames.push_back("dirt_grayrocky_diffusespecular.dds");
                        defaultimp.layerList[0].textureNames.push_back("dirt_grayrocky_normalheight.dds");
                        defaultimp.layerList[1].worldSize = 30;
                        defaultimp.layerList[1].textureNames.push_back("grass_green-01_diffusespecular.dds");
                        defaultimp.layerList[1].textureNames.push_back("grass_green-01_normalheight.dds");
                        defaultimp.layerList[2].worldSize = 200;
                        defaultimp.layerList[2].textureNames.push_back("growth_weirdfungus-03_diffusespecular.dds");
                        defaultimp.layerList[2].textureNames.push_back("growth_weirdfungus-03_normalheight.dds");
                }


                void buildTerrain(){
    Ogre::MaterialManager::getSingleton().setDefaultTextureFiltering(Ogre::TFO_ANISOTROPIC);
    Ogre::MaterialManager::getSingleton().setDefaultAnisotropy(7);

    Ogre::Vector3 lightdir(0.55, -0.3, 0.75);
    lightdir.normalise();

    Ogre::Light* light = mSceneMgr->createLight("LightSource");
    light->setType(Ogre::Light::LT_DIRECTIONAL);
    light->setDirection(lightdir);
    light->setDiffuseColour(Ogre::ColourValue::White);
    light->setSpecularColour(Ogre::ColourValue(0.4, 0.4, 0.4));

    mSceneMgr->setAmbientLight(Ogre::ColourValue(0.2, 0.2, 0.2));

    mTerrainGlobals = OGRE_NEW Ogre::TerrainGlobalOptions();

    mTerrainGroup = OGRE_NEW Ogre::TerrainGroup(mSceneMgr, Ogre::Terrain::ALIGN_X_Z, 513, 12000.0f);
    mTerrainGroup->setFilenameConvention(Ogre::String("ClassEngineTerrain"), Ogre::String("dat"));
    mTerrainGroup->setOrigin(Ogre::Vector3::ZERO);

    configureTerrainDefaults(light);

            mSceneMgr->setSkyDome(true, "Examples/CloudySky", 5, 8);

    for (long x = 0; x <= 0; ++x)
    for (long y = 0; y <= 0; ++y)
       defineTerrain(x, y);

     // sync load since we want everything in place when we start
     mTerrainGroup->loadAllTerrains(true);
                                if (mTerrainsImported)
     {
      Ogre::TerrainGroup::TerrainIterator ti = mTerrainGroup->getTerrainIterator();
      while(ti.hasMoreElements())
      {
       Ogre::Terrain* t = ti.getNext()->instance;
       initBlendMaps(t);
      }
     }

     mTerrainGroup->freeTemporaryResources();

                }
    void CreateCube(const btVector3 &Position, btScalar Mass,const btVector3 &scale,char * name){
                        // empty ogre vectors for the cubes size and position
        Ogre::Vector3 size = Ogre::Vector3::ZERO;
        Ogre::Vector3 pos = Ogre::Vector3::ZERO;
        SceneNode *boxNode;
                Entity *boxentity;
        // Convert the bullet physics vector to the ogre vector
        pos.x = Position.getX();
```

```
	pos.y = Position.getY();
	pos.z = Position.getZ();
	boxentity = mSceneMgr->createEntity(name, "cube.mesh");
		//boxentity->setScale(Vector3(scale.x,scale.y,scale.z));
	boxentity->setCastShadows(true);
	boxNode = mSceneMgr->getRootSceneNode()->createChildSceneNode();
	boxNode->attachObject(boxentity);
		boxNode->scale(Vector3(scale.getX(),scale.getY(),scale.getZ()));
	//boxNode->setScale(Vector3(0.1,0.1,0.1));
	Ogre::AxisAlignedBox boundingB = boxentity->getBoundingBox();
		//Ogre::AxisAlignedBox boundingB = boxNode->_getWorldAABB();
		boundingB.scale(Vector3(scale.getX(),scale.getY(),scale.getZ()));
		size = boundingB.getSize()*0.95f;
		btTransform Transform;
	Transform.setIdentity();
	Transform.setOrigin(Position);
		MyMotionState *MotionState = new MyMotionState(Transform,boxNode);
		//Give the rigid body half the size
	// of our cube and tell it to create a BoxShape (cube)
	btVector3 HalfExtents(size.x*0.5f,size.y*0.5f,size.z*0.5f);
	btCollisionShape *Shape = new btBoxShape(HalfExtents);
		btVector3 LocalInertia;
	Shape->calculateLocalInertia(Mass, LocalInertia);
		btRigidBody *RigidBody = new btRigidBody(Mass, MotionState, Shape, LocalInertia);

	// Store a pointer to the Ogre Node so we can update it later
	RigidBody->setUserPointer((void *) (boxNode));

   // Add it to the physics world
	dynamicsWorld->addRigidBody(RigidBody);
			collisionShapes.push_back(Shape);
		}
//-------------------------------------------------------------------------------------------------------------
   void setupScene()
	{
	   mSceneMgr = mRoot->createSceneManager(ST_GENERIC, "Default SceneManager");
	   Camera *cam = mSceneMgr->createCamera("Camera");
					Viewport *vp = mRoot->getAutoCreatedWindow()->addViewport(cam);
					//cam->setPosition(Ogre::Vector3(1683, 50, 2116));
					//cam->setPosition(Ogre::Vector3(1683, 60, 2116));
					cam->setPosition(Ogre::Vector3(1863, 60, 1650));
					cam->lookAt(Ogre::Vector3(2263, 50, 1200));
					cam->setNearClipDistance(0.1);
					cam->setFarClipDistance(50000);

	   if (mRoot->getRenderSystem()->getCapabilities()->hasCapability(Ogre::RSC_INFINITE_FAR_PLANE))
	   {
		cam->setFarClipDistance(0);   // enable infinite far clip distance if we can
	    }


	   mSceneMgr->setAmbientLight(ColourValue(0.25, 0.25, 0.25));
	   mSceneMgr->setShadowTechnique( SHADOWTYPE_STENCIL_ADDITIVE );
					cmo = createCubeMesh("mcube", "");
					cmo->convertToMesh("cube");

			buildTerrain();

	}
		CEGUI::MouseButton convertButton(OIS::MouseButtonID buttonID)
	{
					switch (buttonID)
					{
					case OIS::MB_Left:
						return CEGUI::LeftButton;

					case OIS::MB_Right:
						return CEGUI::RightButton;

					case OIS::MB_Middle:
						return CEGUI::MiddleButton;

					default:
```

```
                                        return CEGUI::LeftButton;
                                }
        }
                        bool mouseMoved( const OIS::MouseEvent &arg )
                        {
                                CEGUI::System &sys = CEGUI::System::getSingleton();
                                sys.injectMouseMove(arg.state.X.rel, arg.state.Y.rel);
                                // Scroll wheel.
                                if (arg.state.Z.rel)
                                sys.injectMouseWheelChange(arg.state.Z.rel / 120.0f);
                                return true;
                        }
                        //-------------------------------------------------------------------------------------
                        bool mousePressed( const OIS::MouseEvent &arg, OIS::MouseButtonID id )
                        {
                                CEGUI::System::getSingleton().injectMouseButtonDown(convertButton(id));
                                return true;
                        }
                        //-------------------------------------------------------------------------------------
                        bool mouseReleased( const OIS::MouseEvent &arg, OIS::MouseButtonID id )
                        {
                                CEGUI::System::getSingleton().injectMouseButtonUp(convertButton(id));
                                return true;
                        }
//-------------------------------------------------------------------------------------
        void setupInputSystem()
        {
           size_t windowHnd = 0;
           std::ostringstream windowHndStr;
           OIS::ParamList pl;
           RenderWindow *win = mRoot->getAutoCreatedWindow();

           win->getCustomAttribute("WINDOW", &windowHnd);
           windowHndStr << windowHnd;
           pl.insert(std::make_pair(std::string("WINDOW"), windowHndStr.str()));
           mInputManager = OIS::InputManager::createInputSystem(pl);

           try
           {
               mKeyboard = static_cast<OIS::Keyboard*>(mInputManager->createInputObject(OIS::OISKeyboard, true));
                                        mMouse = static_cast<OIS::Mouse*>(mInputManager->createInputObject( OIS::OISMouse, true ));
               mKeyboard->setEventCallback(this);
                                        mMouse->setEventCallback(this);
               //mMouse = static_cast<OIS::Mouse*>(mInputManager->createInputObject(OIS::OISMouse, false));
              //mJoy = static_cast<OIS::JoyStick*>(mInputManager->createInputObject(OIS::OISJoyStick, false));
           }
           catch (const OIS::Exception &e)
           {
              throw new Exception(42, e.eText, "Application::setupInputSystem");
           }
        }

        void setupCEGUI()
        {

           // Other CEGUI setup here.
                                Ogre::TexturePtr tex = mRoot->getTextureManager()-
>createManual( "RTT",Ogre::ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME,
                        Ogre::TEX_TYPE_2D,512,512, 0,Ogre::PF_R8G8B8,Ogre::TU_RENDERTARGET);
                                 CEGUI::SubscriberSlot  evnt= CEGUI::Event::Subscriber(&Application::quit, this);
            ceguiManager = ceguiManager->getSingleton();
            ceguiManager->initialize( evnt,tex);
        }

        void createFrameListener()
        {
           mRoot->addFrameListener(this);
        }

        void startRenderLoop()
        {
           mRoot->startRendering();
        }
```

```
    void createBulletSim(void) {
        ///collision configuration contains default setup for memory, collision setup. Advanced users can create their own configuration.
        collisionConfiguration = new btDefaultCollisionConfiguration();

        ///use the default collision dispatcher. For parallel processing you can use a diffent dispatcher (see Extras/BulletMultiThreaded)
        dispatcher = new   btCollisionDispatcher(collisionConfiguration);

        ///btDbvtBroadphase is a good general purpose broadphase. You can also try out btAxis3Sweep.
        overlappingPairCache = new btDbvtBroadphase();

        ///the default constraint solver. For parallel processing you can use a different solver (see Extras/BulletMultiThreaded)
        solver = new btSequentialImpulseConstraintSolver;

        dynamicsWorld = new btDiscreteDynamicsWorld(dispatcher,overlappingPairCache,solver,collisionConfiguration);
        dynamicsWorld->setGravity(btVector3(0,-100,0));
                                    {
        ///create a few basic rigid bodies
        // start with ground plane, 1500, 1500
                                    Ogre::Terrain * pTerrain=mTerrainGroup->getTerrain(0,0);
                                    float* terrainHeightData = pTerrain->getHeightData();
        Ogre::Vector3 terrainPosition = pTerrain->getPosition();
                                    float * pDataConvert= new float[pTerrain->getSize() *pTerrain->getSize()];
                                    for(int i=0;i<pTerrain->getSize();i++)
                                                memcpy(
                                                                                pDataConvert+pTerrain->getSize() * i, // source
                                                                                terrainHeightData + pTerrain->getSize() * (pTerrain->getSize()-i-
1), // target
                                                                                sizeof(float)*(pTerrain->getSize()) // size
                                                                                );

                                                float metersBetweenVertices = pTerrain->getWorldSize()/(pTerrain->getSize()-1); //edit: fixed 0 -> 1 on
2010-08-13
                                                btVector3 localScaling(metersBetweenVertices, 1, metersBetweenVertices);

                                                btHeightfieldTerrainShape* groundShape = new btHeightfieldTerrainShape(
                                                                                pTerrain->getSize(),
                                                                                pTerrain->getSize(),
                                                                                pDataConvert,
                                                                                1/*ignore*/,
                                                                                pTerrain->getMinHeight(),
                                                                                pTerrain->getMaxHeight(),
                                                                                1,
                                                                                PHY_FLOAT,
                                                                                true);

                                                groundShape->setUseDiamondSubdivision(true);
                                                groundShape->setLocalScaling(localScaling);

                                                btRigidBody * mGroundBody = new btRigidBody(0, new btDefaultMotionState(), groundShape);

                                                mGroundBody->getWorldTransform().setOrigin(
                                                                                btVector3(
                                                                                                terrainPosition.x,
                                                                                                terrainPosition.y + (pTerrain->getMaxHeight()-pTerrain-
>getMinHeight())/2,
                                                                                                terrainPosition.z));

                                                mGroundBody->getWorldTransform().setRotation(
                                                                                btQuaternion(
                                                                                                Ogre::Quaternion::IDENTITY.x,
                                                                                                Ogre::Quaternion::IDENTITY.y,
                                                                                                Ogre::Quaternion::IDENTITY.z,
                                                                                                Ogre::Quaternion::IDENTITY.w));

                                                dynamicsWorld->addRigidBody(mGroundBody);
                                                collisionShapes.push_back(groundShape);
                                                CreateCube(btVector3(2623, 500, 750),1.0f,btVector3(0.3,0.3,0.3),"Cube0");
                                                CreateCube(btVector3(2263, 150, 1200),1.0f,btVector3(0.2,0.2,0.2),"Cube1");
                                                CreateCube(btVector3(2253, 100, 1210),1.0f,btVector3(0.2,0.2,0.2),"Cube2");
                                                CreateCube(btVector3(2253, 200, 1210),1.0f,btVector3(0.2,0.2,0.2),"Cube3");
                                                CreateCube(btVector3(2253, 250, 1210),1.0f,btVector3(0.2,0.2,0.2),"Cube4");
            //CreateCube(btVector3(1963, 150, 1660),1.0f,btVector3(0.2,0.2,0.2),"Cube1");
```

```
        }

    }
};


#if OGRE_PLATFORM == PLATFORM_WIN32 || OGRE_PLATFORM == OGRE_PLATFORM_WIN32
#define WIN32_LEAN_AND_MEAN
#include "windows.h"

INT WINAPI WinMain(HINSTANCE hInst, HINSTANCE, LPSTR strCmdLine, INT)
#else
int main(int argc, char **argv)
#endif
{
    try
    {
        Application app;
        app.go();
    }
    catch(Exception& e)
    {
#if OGRE_PLATFORM == PLATFORM_WIN32 || OGRE_PLATFORM == OGRE_PLATFORM_WIN32
        MessageBoxA(NULL, e.getFullDescription().c_str(), "An exception has occurred!", MB_OK | MB_ICONERROR | MB_TASKMODAL);
#else
        fprintf(stderr, "An exception has occurred: %s\n",
            e.getFullDescription().c_str());
#endif
    }

    return 0;
}



// make a cube, no cube primiatives in ogre
// yanked from betajaen
//http://www.ogre3d.org/forums/viewtopic.php?p=301318&sid=ce193664e1d3d7c4af509e6f4e2718c6
ManualObject* createCubeMesh(Ogre::String name, Ogre::String matName) {

    ManualObject* cube = new ManualObject(name);

    cube->begin(matName);

    cube->position(0.5,-0.5,1.0);cube->normal(0.408248,-0.816497,0.408248);cube->textureCoord(1,0);
    cube->position(-0.5,-0.5,0.0);cube->normal(-0.408248,-0.816497,-0.408248);cube->textureCoord(0,1);
    cube->position(0.5,-0.5,0.0);cube->normal(0.666667,-0.333333,-0.666667);cube->textureCoord(1,1);
    cube->position(-0.5,-0.5,1.0);cube->normal(-0.666667,-0.333333,0.666667);cube->textureCoord(0,0);
    cube->position(0.5,0.5,1.0);cube->normal(0.666667,0.333333,0.666667);cube->textureCoord(1,0);
    cube->position(-0.5,-0.5,1.0);cube->normal(-0.666667,-0.333333,0.666667);cube->textureCoord(0,1);
    cube->position(0.5,-0.5,1.0);cube->normal(0.408248,-0.816497,0.408248);cube->textureCoord(1,1);
    cube->position(-0.5,0.5,1.0);cube->normal(-0.408248,0.816497,0.408248);cube->textureCoord(0,0);
    cube->position(-0.5,0.5,0.0);cube->normal(-0.666667,0.333333,-0.666667);cube->textureCoord(0,1);
    cube->position(-0.5,-0.5,0.0);cube->normal(-0.408248,-0.816497,-0.408248);cube->textureCoord(1,1);
    cube->position(-0.5,-0.5,1.0);cube->normal(-0.666667,-0.333333,0.666667);cube->textureCoord(1,0);
    cube->position(0.5,-0.5,0.0);cube->normal(0.666667,-0.333333,-0.666667);cube->textureCoord(0,1);
    cube->position(0.5,0.5,0.0);cube->normal(0.408248,0.816497,-0.408248);cube->textureCoord(1,1);
    cube->position(0.5,-0.5,1.0);cube->normal(0.408248,-0.816497,0.408248);cube->textureCoord(0,0);
    cube->position(0.5,-0.5,0.0);cube->normal(0.666667,-0.333333,-0.666667);cube->textureCoord(1,0);
    cube->position(-0.5,-0.5,0.0);cube->normal(-0.408248,-0.816497,-0.408248);cube->textureCoord(0,0);
        cube->position(-0.5,0.5,1.0);cube->normal(-0.408248,0.816497,0.408248);cube->textureCoord(1,0);
    cube->position(0.5,0.5,0.0);cube->normal(0.408248,0.816497,-0.408248);cube->textureCoord(0,1);
    cube->position(-0.5,0.5,0.0);cube->normal(-0.666667,0.333333,-0.666667);cube->textureCoord(1,1);
    cube->position(0.5,0.5,1.0);cube->normal(0.666667,0.333333,0.666667);cube->textureCoord(0,0);

    cube->triangle(0,1,2);      cube->triangle(3,1,0);
    cube->triangle(4,5,6);      cube->triangle(4,7,5);
    cube->triangle(8,9,10);      cube->triangle(10,7,8);
    cube->triangle(4,11,12);    cube->triangle(4,13,11);
    cube->triangle(14,8,12);    cube->triangle(14,15,8);
```

```
  cube->triangle(16,17,18);   cube->triangle(16,19,17);
  cube->end();

  return cube;
}
```