NCTU 3D Game Programming

Marching Robots

Zhang Zhexian; Student ID: 0545080; Assignment: 2; Email: zhangzhexian@outlook.com

[Yes, this is my own work ☺]

Introduction

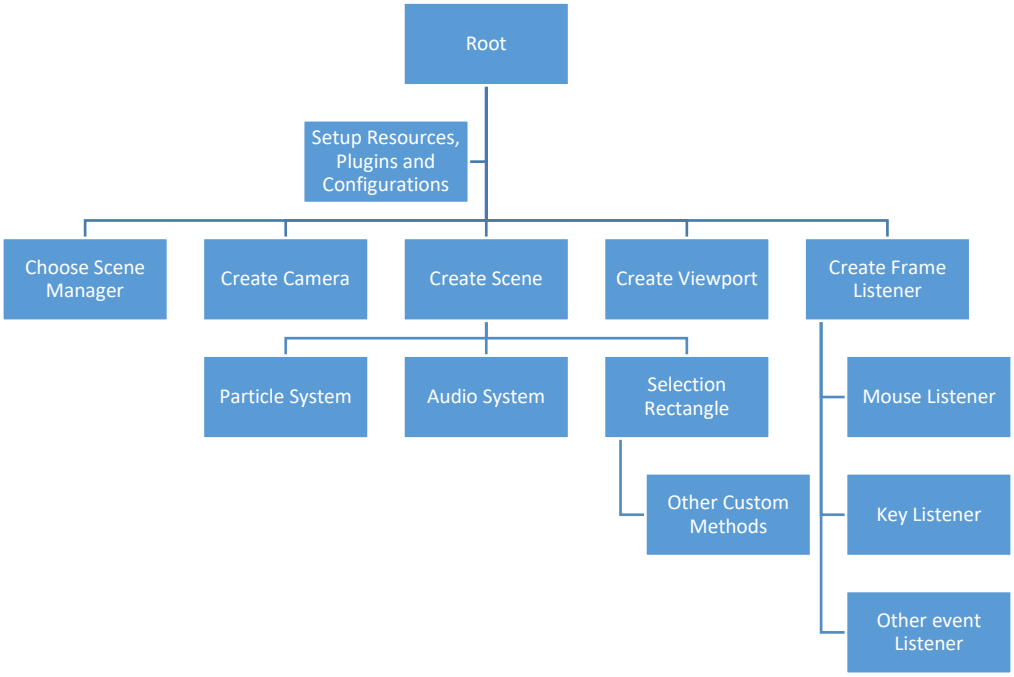
Compared to the previous static assignment, this assignment is much more challenging as it involves more advanced topics such as animation, collision detection, and special effect.

The objective is to create the following:

- In the main viewport:
  - Sky box
  - Fog
  - Ground
  - Two circles of robots, one of them is bigger than the rest
  - One central sphere
  - A rotating light that cast shadows on the ground
  - On left mouse click and drag, a selection rectangle is created
  - On left mouse release, the robots fall under the selection rectangle will show their bounding boxes
  - On right mouse click, the selected robots will move to the click position
  - The robot moves with the “walk” animation
  - Upon arrival, a sound will be played
  - When the bigger robot moves, a particle system is played
  - Collision detection is enabled between the robot and the central sphere, but not among the robots
- In the second viewport:
  - Yellow background
  - Ratio of height to width is 4:1
  - No sky or overlay
  - No fog

(word count: 181)

System Architecture



The Ogre project system is modularised and organized based on the standard **Model-View-Controller** (MVC) structure. The model is the basic scene, with the scene node and entities. The view decides how the scene is rendered, and is mostly done by scene manager, camera, and viewport settings. The controller decides how the animation will affect the game and other event listening processes.

(word count: 61)

Method

Task	**Implementation**
Show your name and ID on the top bar of the window	initialise(true, "NCTU 3D Game Programming Assignment 1; Author: Zhang Zhexian (ID: 0545080)");

Create a scene manager. Use it to create its own camera and viewport.	mSceneMgr = mRoot->createSceneManager(Ogre::ST_GENERIC);
	
	mSceneMgr->createCamera("PlayerCam");
	mWindow->addViewport(mCamera, z-index);

The viewport occupies the entire screen space	mWindow->addViewport(mCamera, z-index, 1, 1, 1, 1); or leave the last four parameters empty (default is 1)

Setup an ambient light and enable linear white fog with parameters (0, linearStart=1400, linearEnd= 1600). Set the fog color to white.	mSceneMgr->setAmbientLight(ColourValue(0.9, 0.9, 0.9));
	Ogre::ColourValue fadeColour(1, 1, 1);
	mSceneMgr->setFog(Ogre::FOG_LINEAR, fadeColour, 0.0, 1400, 1600);

Create a skybox.	mSceneMgr->setSkyDome(true, "Examples/CloudySky", 5, 8);
Enable shadows. Use stencil additive shadow type.	mSceneMgr->setShadowTechnique(SHADOWTYPE_STENCIL_ADDITIVE);

Create a plane, set a material to it. Make sure that the plane does not cover the entire viewport. The background of the viewport can be seen.	void BasicTutorial_00::createDefaultPlane(void){
	Ogre::Plane plane(Ogre::Vector3::UNIT_Y, 0);
	Ogre::MeshManager::getSingleton().createPlane("ground",
	Ogre::ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME,
	plane, 1500, 1500, 20, 20, true, 1, 5, 5, Ogre::Vector3::UNIT_Z);
	}
	
	Ogre::Entity* entGround = mSceneMgr->createEntity("GroundEntity",
	"ground");
	
	mSceneMgr->getRootSceneNode()->createChildSceneNode()->attachObj
	ect(entGround);
	
	entGround->setMaterialName("Examples/BumpyMetal");

Create two circles of robots in a space of dimension 600x600.	void BasicTutorial_00::createCircleOfObjects(Ogre::String objectMesh,
	Ogre::Real numObjects, float circleRadius, Ogre::SceneManager*
	mSceneMgr) {
	Ogre::Real numCubes = numObjects;
	float PI = 3.141592654;
	float function;
	float height;
	float xPosition;
	float zPosition;
	float scaleFactor;
	for (int i=0; i<numCubes; ++i){
	Ogre::String name="robotCircle"+
	Ogre::StringConverter::toString(i);
	Ogre::Entity *ent = mSceneMgr->createEntity(name,
	objectMesh);
	
	ent->setMaterialName("Examples/SphereMappedRustySteel");
	Ogre::AxisAlignedBox boundingBox =
	ent->getBoundingBox();
	Ogre::Real cubeSize = boundingBox.getMaximum().x -
	boundingBox.getMinimum().x;

	Ogre::SceneNode *sceneNode = mSceneMgr->getRootSceneNode()->createChildSceneNode();
	sceneNode->attachObject(ent);
	function = i/(double)(numCubes-1);
	height = (1+sin(function*PI*4))*50;
	xPosition = circleRadius*cos(function*PI*2);
	zPosition = circleRadius*sin(function*PI*2);
	scaleFactor = 1.0/cubeSize/numCubes*255*0.8;
	sceneNode->scale(scaleFactor, height/cubeSize, scaleFactor);
	sceneNode->setPosition(xPosition, 50, zPosition);
	}
	}
	
	Ogre::Real numRobots = 30;
	BasicTutorial_00::createCircleOfObjects("robot.mesh", numRobots, 300, mSceneMgr);

Set one robot to be the largest, e.g., scale(2, 2, 2).	mSceneMgr->getRootSceneNode()->getChild(0)->scale(2, 2, 2);

Create a sphere with radius 70.0 and set it the center.	Ogre::Entity* entSphere = mSceneMgr->createEntity("sphere", "sphere.mesh");
	
	entSphere->setCastShadows(true);
	
	Ogre::SceneNode* ogreNodeSphere = mSceneMgr->getRootSceneNode()->createChildSceneNode(Ogre::Vector3(0, 50, 0));
	
	ogreNodeSphere->setScale(Ogre::Vector3(0.7,0.7,0.7));
	
	ogreNodeSphere->attachObject(entSphere);
	
	entSphere->setMaterialName("Examples/SphereMappedRustySteel");

The sphere cannot be selected.	entSphere->setQueryFlags(0);

Create one light and rotate it around the scene periodically. While the light is rotating, the shadows casted by the robots are changing accordingly.	Ogre::Light* rotatingLight = mSceneMgr->createLight("AnimLight");
	rotatingLight->setType(Light::LT_POINT);
	rotatingLight->setDiffuseColour(ColourValue(1,1,1));
	rotatingLight->setSpecularColour(ColourValue(1,1,1));
	rotatingLight->setCastShadows(true);
	
	Ogre::SceneNode* ogreNodeParentLight = mSceneMgr->getRootSceneNode()->createChildSceneNode("parentLight",
	Ogre::Vector3(0, 0, 0));

	Ogre::SceneNode* ogreNodeLight = ogreNodeParentLight->createChildSceneNode(Ogre::Vector3(0, 0, 0));
	
	ogreNodeLight->attachObject(rotatingLight);
	
	ogreNodeLight->translate(Ogre::Vector3(-1000,500,0));
	
	--- 	
	In frameStarted:
	
	mSceneMgr->getRootSceneNode()->getChild("parentLight")->yaw(Ogre::Degree(0.1));

A rectangle region is defined by dragging the mouse while the LEFT MOUSE KEY is pressed. It must work for the following five cases: 1) Lower left to upper right; 2) upper left to lower right; 3) lower right to upper left; 4) upper right to lower left; and 5) click and release at the same position.	mSelectionRect = new SelectionRectangle("selectionRect");
	
	mSceneMgr->getRootSceneNode()->createChildSceneNode()->attachObject(mSelectionRect);
	
	mSelectionRect->setLightMask(0);
	
	mSelectionRect->setCastShadows(false);
	
	Then define functions in
	
	- mouseMoved
	
	- mouseReleased
	
	- mousePressed

The robots are selected if they overlap with the rectangle region when the LEFT MOUSE KEY is released.
 	
&
 	
Show the bounding volume of a robot if the robot is selected.
	bool BasicTutorial_00::mouseReleased(const OIS::MouseEvent &arg, OIS::MouseButtonID id)
	{
	isMousePressed = false;
	if (left==right && top==bottom) {
	return BaseApplication::mouseReleased(arg, id);
	
	}
	
	//USING mTrayMgr=============
	Real nleft = left;
	Real nright = right;
	Real ntop = 1+top;
	Real nbottom = 1+bottom;
	Ray topLeft = mTrayMgr->screenToScene(mCamera, Vector2(nleft, ntop));
	Ray topRight = mTrayMgr->screenToScene(mCamera, Vector2(nright, ntop));
	Ray bottomLeft = mTrayMgr->screenToScene(mCamera, Vector2(nleft, nbottom));

```
        Ray bottomRight = mTrayMgr->screenToScene(mCamera,
Vector2(nright, nbottom));
        //End Using mTrayMgr=============

        // The plane faces the counter clockwise position.
        PlaneBoundedVolume vol;
        int np = 100;
        vol.planes.push_back(Plane(topLeft.getPoint(3),
topRight.getPoint(3),                    bottomRight.getPoint(3)));
// front plane
        vol.planes.push_back(Plane(topLeft.getOrigin(),
topLeft.getPoint(np),    topRight.getPoint(np)));      // top plane
        vol.planes.push_back(Plane(topLeft.getOrigin(),
bottomLeft.getPoint(np),         topLeft.getPoint(np)));     // left plane
        vol.planes.push_back(Plane(bottomLeft.getOrigin(),
bottomRight.getPoint(np),        bottomLeft.getPoint(np)));  // bottom
plane
        vol.planes.push_back(Plane(bottomRight.getOrigin(),
topRight.getPoint(np),   bottomRight.getPoint(np)));    // right plane

        PlaneBoundedVolumeList volList;
        volList.push_back(vol);
        mVolQuery->setVolumes(volList);

        SceneQueryResult result = mVolQuery->execute();

        SceneQueryResultMovableList::iterator itr =
result.movables.begin();


        // Get the results, set the camera height
        // We are interested in the first intersection. It is ok to traverse
all the results.

        for (itr = result.movables.begin(); itr != result.movables.end();
++itr)
        {
                if (*itr)
                {
                        mCurrentObject = (*itr)->getParentSceneNode();
                        bool flgShow =
mCurrentObject->getShowBoundingBox();
                        mCurrentObject->showBoundingBox(!flgShow);
                }
        }

        return BaseApplication::mouseReleased( arg, id );
```

	}

Press RIGHT MOUSE KEY to define a target position and make the selected robot(s) walk to the destination and then stop there.	// Define right click differently
	if (id == OIS::MB_Left)
	{}
	else if (id == OIS::MB_Right)
	{}

If the robot(s) is/are not walking, set it/them to idle.	mAnimationState[i] = mEntityArr[i]->getAnimationState("Idle");
	mAnimationState[i]->setLoop(true);
	mAnimationState[i]->setEnabled(true);

Attach a particle system to the largest robot.	ParticleSystem* sunParticle = mSceneMgr->createParticleSystem("Sun", "Examples/GreenyNimbus"); mSceneNodeArr[0]->attachObject(sunParticle);
Create another camera and another viewport. The background color of the viewport is YELLOW.	Ogre::Viewport* viewportSmall = mWindow->addViewport(mCameraSmall,1, 0.75, 0, 0.25, 0.25); viewportSmall->setBackgroundColour(Ogre::ColourValue(1,1,0));

(word count: 520)

Discussion

1. **What do you draw the selection rectangle?**

   When the mouse is pressed, the initial (x, y) values are stored.

   Depending on the direction of mouse movement, the initial (x, y) will be used as the anchor point.

Mouse movement direction	Y	X
Top left to bottom right	Top	Left
Top right to bottom left	Top	Right
Bottom right to top left	Bottom	Right
Bottom left to top right	Bottom	Left

   When the mouse is released, the new (x, y) will be used to define the other corner of the selection rectangle

2. **How do you compute the intersection point between a ray and the plane?**

   // get selection ray from viewport mouse position

   Ogre::Ray selectRay =
   mCamera->getCameraToViewortRay((Ogre::Real)mouseX/renderTarget->getWidth(),
   (Ogre::Real)mouseY/renderTarget->getHeight());

```
// get intersection point on selection ray
Ogre::Real point = ray.intersects(Plane).second;


// get world coordinates of intersection point
Ogre::Vector3 position = ray.getPoint(point);
```

3. **How do you disable skies for the second viewport?**

   In the setSkyDome method, the first parameter is a Boolean that determines if the sky is enabled. Setting it to false will disable the sky.

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

4. **How do you do so that the robots walk to the target position?**

   At every frame, change the robot's position from the previous position by the unit vector of the movement direction.

   To check if the robot has reached the destination, we may use dot product between the vector of "starting point to current robot position" and the vector of "current robot position to the destination position". This is because if robot has yet to reach the destination, the two vector will have the same direction, and hence the dot product will be POSITIVE. If the robot has reached, the dot product will be NEGATIVE.

5. **The background color of the first viewport is set to black. However, why does the background color of the first viewport appear to be white?**

   It appears to be white because the fog lays on top of the background. The fog's fading color is white, thus it masks the background's original black color.

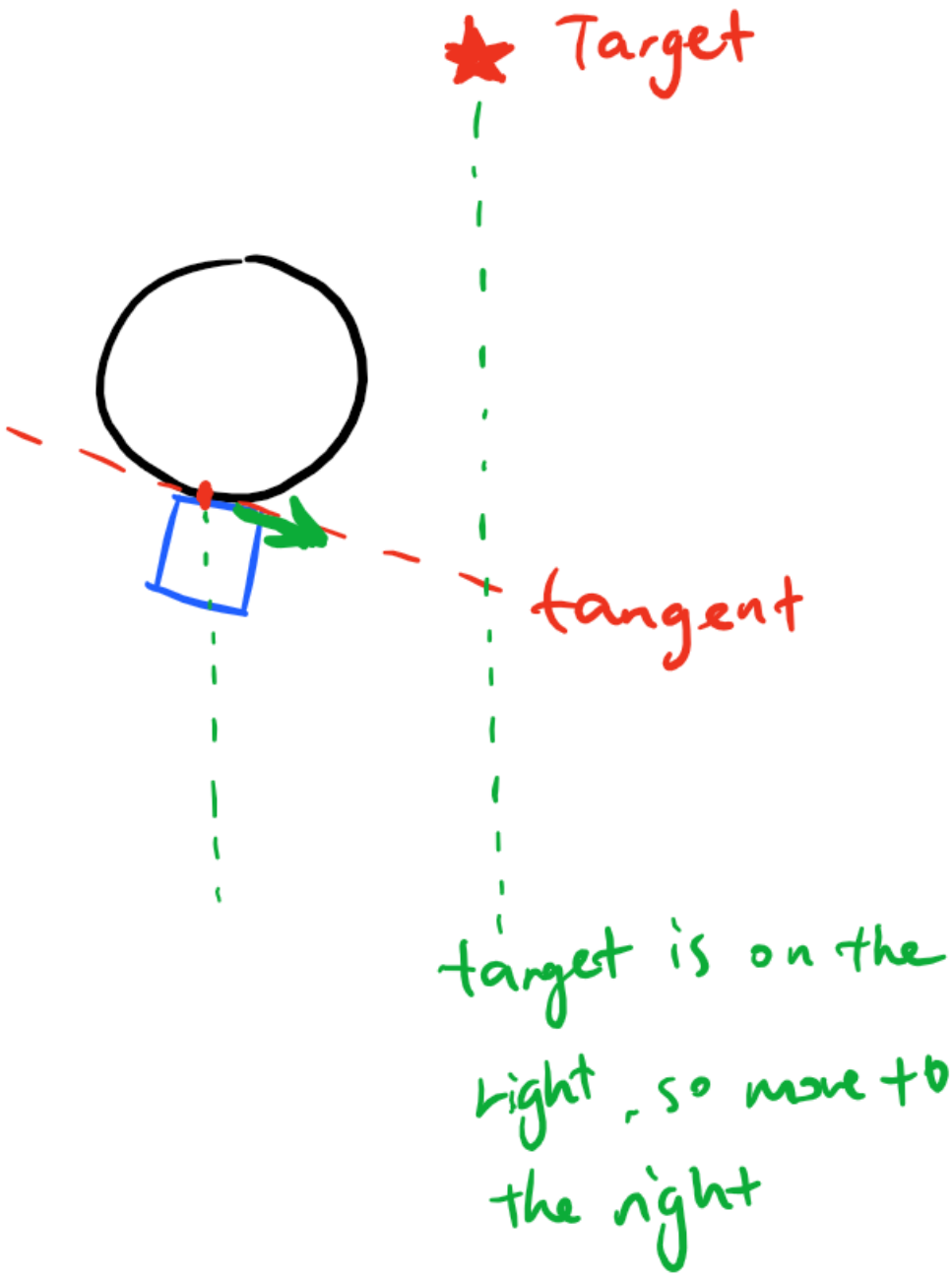
6. **How do you do collision handling for the moving robots and the sphere? Draw a figure for illustration.**

   The collision handling is done by first analysing the position of the collision point between two bounding boxes (or one bounding box one bounding sphere, or between two spheres, etc).

   At the collision point, the tangent to the colliding object (in this case it is the central sphere) is drawn, and the moving object will move along the tangent for 1 unit. The direction of movement depends on the position of the target destination (if the target is on the left, then move to the left along the tangent for 1 unit).

   If collision occurs again, repeat the above process; if no more collision, move to the target directly.

   Illustration:



7. Other interesting observations:

- [Robot Direction] At the start, all robots created are facing to one direction; even if yaw is applied by a fixed amount, they will still face one fixed direction. If using function to define the facing direction, it will not be intuitive and very troublesome. The solution is to set the lookAt property for all robots, so they face the central point automatically.
- [Single Robot Scaling] In order to scale only one robot, get scene node by name is first tried, but using entity name does not work for getting the correct scene node. The correct way is to get scene

node by index (in this case, index 0 is used for convenience). This works because the task only requires a single robot to be scaled larger, but the identity of the robot is not fixed.

- [Rotating Light] Even though light is not an entity, it can still be attached to a scene node to be animated. The rotating light effect may be done by defining a circular path for the light to travel, and modify the light position in each frame along the path. However, this is a troublesome procedure. A better way is to translate the light at some distance away from the centre of the circular path (the position of the parent scene node), and then do a yaw on the parent scene node.
- [Light Type and Shadow] Initially I realised that the shadow is not rendered on the ground. I thought it is the problem of the ground material, either that the setCastShadow need to be set as false, or the material need to be a shadow receiver instead of caster. It turns out that the problem is that I used SPOTLIGHT instead of POINTLIGHT, and the difference made the shadow rendering different.

(word count: 757)

Conclusion

This is so far the most challenging assignment. Even though I was worried while doing the project, partially because of the difficulty level, and partially because of the tight timeline, I feel extremely proud of what I have leanred simply by completing the assignment. Many of the tasks looks easy but are in fact hard to implement. The practically skills learned are transferrable and will be beneficial to my other projects in the future.

Having said this, I will hope that the next assignment will have a longer time for us to attempt. Given the difficulty level, and the fact that the course material is only taught one week ago, the current time given is not sufficient. Despite that, it is a fun experience to code for 12 hours straight without moving much ☺

(word count: 134)