Ogre Primer
What is OGRE?

- OGRE is an open source graphics engine.
- OGRE handles loading mesh files, lighting, basic shadows, texture mapping, animation, and a whole host of other features.
- It does *not* handle sound, physics, input, and other non-graphics features.
- The OGRE wiki (http://www.ogre3d.org/wiki) has a bunch of great articles and tutorials. It will be your best reference for the project.
Basic Ogre Classes

• Ogre Core Objects
  • Root
  • Render System
  • Scene Manager
  • Resource Group Manager
  • Entity
  • Materials
Basic Ogre Classes

• Ogre Core Objects
  • Root
  • Render System
  • Scene Manager
  • Resource Group Manager
  • Entity
  • Materials
OGRE!
Scene Graph

- Hierarchical data structure. Stores position + orientation of objects relative to parent
- All objects are attached to the scene graph
- Composed of SceneNodes
- Entities/Cameras/Lights don't actually have positions/orientations. They are attached to sceneNodes that keep track of these things
  - mSceneMgr->getRootSceneNode()
  - mSceneNode->createChildSceneNode()
  - mSceneNode->attachObject(Movable Object)
SceneManager

- This is the main class for creating objects in and adding them to the scene
- Handles
  - Static scene geometry
  - Cameras
  - Lights
  - Entities
  - Scene Nodes
- All of these objects are created through the scene manager.
Camera Example

• Creating Camera

```cpp
mCamera = mSceneMgr->createCamera("PlayerCam");
mCamera->setPosition(Vector3(0,10,500));
mCamera->lookAt(Vector3(0,0,0));
mCamera->setNearClipDistance(5);
```

• Setting up window viewport

```cpp
Viewport* vp = mWindow->addViewport(mCamera);
vp->setBackgroundColour(ColourValue(0,0,0));
mCamera->setAspectRatio(Real(vp->getActualWidth()) / Real(vp->getActualHeight()));
```
Light Example

```cpp
light = mSceneMgr->createLight("Light3");
light->setType(Light::LT_DIRECTIONAL);
light->setDiffuseColour(ColourValue(.25, .25, 0));
light->setSpecularColour(ColourValue(.25, .25, 0));
```
Scene Manager

- Scene Manager also handles static background geometry.
- Create using Root::createSceneManager()
- Many different types for different types of environments
  - ST_GENERIC: Doesn't load static terrain data
  - ST_EXTERIOR_CLOSE: Simple SM for loading terrains from cfg files. Terrain specified as heightmap
  - ST_INTERIOR: Loads interior environments from .bsp files
  - ST_EXTERIOR_REAL_FAR (Advanced): SM for large exterior scenes. Contains optimizations for paging parts of the world in/out
  - Many more
Entities

• All movable objects with meshes are considered entities

• mSceneMgr->createEntity("foo.mesh");

• Store pointer to mesh, animations, material, and other object specific properties.
Frame Listener

• Very important
• Only point of control after render loop has started. This is where you can update the scene
• Created by Root
• Abstract Base Class. You need to write your own subclass and override these two methods
  - frameStarted()
  - frameEnded()
Animation

- Rigid body animation can be done by transforming SceneNodes before each frame: translate, rotate, scale

- Animated OGRE meshes also store a list of their current animations, indexed by name.

```cpp
mAnimationState = ent->getAnimationState("Idle");
mAnimationState->setLoop(true);
mAnimationState->setEnabled(true);
mAnimationState->addTime(evt.timeSinceLastFrame);
```
Setting Up

• Setup done through root object

• Overview of startup sequence
  – Create the Root object.
  – Define the resources that Ogre will use.
  – Choose and set up the RenderSystem (that is, DirectX, OpenGL, etc).
  – Create the RenderWindow (the window which Ogre resides in).
  – Initialize the resources that you are going to use.
  – Create a scene using those resources.
  – Set up any third party libraries and plugins.
  – Create any number of frame listeners.
  – Start the render loop

• “Basic Tutorial 6” on the wiki has a more detailed description
Content
Loading Content

- On startup, specify directories for OGRE to look for resources
- OGRE will automatically load everything in those directories
- ResourceGroupManager::addResourceLocation();
- Often specified in resources.cfg, which is parsed during startup
- This is done for you in the example framework, but it probably loads more than you need
Meshes

- OGRE uses its own .mesh format for storing object models
  - UV mapped textures, many material settings, armatures and animations are exported
- Actually 2 formats
  - .mesh.xml: What is exported from 3D modeling programs. Cannot be read by OGRE
  - .mesh: binary format. Readable by OGRE
  - Convert Between using OgreXmlConverter
- The wiki has exporters for Blender, Maya, etc.
  - Specify animations here
Materials

- Example Material Script

```plaintext
material walls/funkywall1
{
  // first, preferred technique
technique
  {
    // first pass
    pass
    {
      ambient 0.5 0.5 0.5
      diffuse 1.0 1.0 1.0
    }

    // Texture unit 0
    texture_unit
    {
      texture wibbly.jpg
    }

    // Texture unit 1 (this is a multitexture pass)
texture_unit
    {
      texture wobbly.png
    }
  }
}
```
Materials

• OGRE has rich, expressive material support
• Specified as scripts: can be shared between objects. Even support inheritance!
• Nested Structure
  – Technique: ways of achieving a given visual effect. Define multiple for different levels of quality
  – Pass: each pass is a complete render of the material
  – Texture Unit: defines a texture to be applied to the object
Materials: Shaders

- References to shaders can be defined in material scripts

```plaintext
vertex_program myVertexProgram asm
{
  source myVertexProgram.asm
  syntax vs_1_1
}

vertex_program_ref myVertexProgram
{
  <params>
}

- Refer to OGRE manual for detailed description of materials
Effects
Render to Texture

- Ogre provides a great framework for rendering to textures
- This allows all kinds of interesting effects (mirrors, portals, etc)
Render to Texture Example

Ogre::TextureManager::getSingleton().createManual("RttTex",
    ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME,
    TEX_TYPE_2D, mWindow->getWidth(), mWindow->getHeight(), 0,
    PF_R8G8B8, TU_RENDERTARGET);

Ogre::RenderTexture *renderTexture = texture->getBuffer()-
    >getRenderTarget();

renderTexture->addViewport(mCamera);
renderTexture->getViewport(0)->setClearEveryFrame(true);
renderTexture->getViewport(0)->setBackgroundColour(ColourValue::Black);
renderTexture->getViewport(0)->setOverlaysEnabled(false);
renderTexture->setAutoUpdated(true);
Shadows

- Shadows become really simple in OGRE
- Try experimenting with different types of shadows

mSceneMgr->setShadowTechnique(SHADOWTYPE_STENCIL_ADDITIVE);
ent->setCastShadows(true);
Conclusion

• OGRE has a lot of nice features, but you might have to do some digging to discover them

• The OGRE wiki is your best resource for learning about OGRE. There are a lot of articles, tutorials and documentation.