

---

## CMSC 498M: Chapter 4b Game Engines and Ogre 3D

**Reading:** The material for this lecture is adapted from

- Online Ogre tutorials:  
[http://www.ogre3d.org/wiki/index.php/Ogre\\_Tutorials](http://www.ogre3d.org/wiki/index.php/Ogre_Tutorials).
- *Pro Ogre 3D Programming*, by G. Junker, Apress, Berkeley, CA, 2006.

**Overview:**

- More about Ogre

Chapter 4, Slide 1  
Copyright © David Mount and Amitabh Varshney

---

## Ogre Frame Listener

**Frame Listener:**

- This is where you include code that is to be processed for **each frame** that is rendered.
- It is an **interface** that allows you to provide two member functions:

```
bool frameStarted( const FrameEvent& event); // called before drawing  
bool frameEnded( const FrameEvent& event ); // called after drawing
```

- Returning **false** from either method causes Ogre to **terminate**.
- The variable **FrameEvent::timeSinceLastFrame** indicates the amount of time elapsed since the last **frameStarted**. Useful for updating your physical simulation of time.

Chapter 4, Slide 2  
Copyright © David Mount and Amitabh Varshney

## Ogre Frame Listener

### Frame Listener Registration:

- To **register** a FrameListener, **derive** your own class, say MyFrameListener, from FrameListener and **add** it to the Ogre root.

```
MyFrameListener* mFrameListener = new MyFrameListener( ... );  
mRoot->addFrameListener( mFrameListener );
```

- It is also possible to **remove** a FrameListener.

### Acquiring Input:

**Event-driven:** Register a **KeyboardListener** or **MouseListener**. These are not part of Ogre proper, but part of **OIS** (the Open Input System).

**Polling:** Check the state of each key with each new frame.

**For simplicity:** we will discuss only the polling method here, but for a serious application with multiple input sources, event-driven is better.

Chapter 4, Slide 3  
Copyright © David Mount and Amitabh Varshney

## Ogre Frame Listener

### Input Manager :

- You need to perform a number of **initializations** involving OIS. See [http://www.ogre3d.org/wiki/index.php/Using\\_OIS](http://www.ogre3d.org/wiki/index.php/Using_OIS) for sample code for a sample Input Manager.

```
OIS::Mouse* mMouse;           // for mouse input  
OIS::Keyboard* mKeyboard;     // for keyboard input  
OIS::JoyStick* mJoy;         // for joystick input
```

### Sample Keyboard Input:

- Add this to your **frameStarted** function:

```
mKeyboard->capture( );           // acquire keyboard input state  
                                // terminate on ESC  
if( mKeyboard->isKeyDown( OIS::KC_ESCAPE ) ) return false;
```

- For more OIS definitions see: **OgreSDK\include\OIS**.

Chapter 4, Slide 4  
Copyright © David Mount and Amitabh Varshney

## Ogre Frame Listener

### Sample Mouse Input:

- Add this to your **frameStarted** function:

```
mMouse->capture( );           // acquire keyboard input state
                               // process left mouse click
if ( mMouse->getMouseState( ).buttonDown( OIS::MB_Left ) ){
    // ... processing when left mouse button is down
}
```

- For more OIS definitions see: **OgreSDK\include\OIS**.

### Word of Caution:

- The above code tests the **current state** of the mouse and keyboard. If the user **holds down** the button or key, the event will be processed **repeatedly** until it is released.
- To process a single **key- or button-click**, store a boolean that holds the **current key state**, and process only on a **change** of state.

Chapter 4, Slide 5  
Copyright © David Mount and Amitabh Varshney

## Ogre Scene Manager

**Scene Manager:** Organizing all **scene content** for efficient rendering.

- Creating and placing **movable objects**, lights, and cameras.
- Loading and assembling **world geometry**.
- Answering **scene queries**: "Which objects are contained in this sphere?" "What is the height of the mesh at this point?"
- Determining which **objects are invisible**, for culling.
- Organizing and sorting (in increasing distance) **light sources**.
- Setup and rendering of **shadows**.
- Setup and rendering **all objects of the scene**, including any background or skybox.

Chapter 4, Slide 6  
Copyright © David Mount and Amitabh Varshney

## Ogre Scene Manager

**Scene Manager Types:** Ogre offers many different scene managers, optimized for various scenarios.

**ST\_GENERIC:** Generic scene manager.

- A simple and generic solution, works well for most scenes
- Can use the `StaticGeometry` class to accelerate large chunks of immovable geometry.

**ST\_EXTERIOR\_CLOSE:** `Terrain_Scene_Manager`

- Fast rendering of high-resolution terrain.
- Easy to generate terrain via heightmaps and terrain textures.

**ST\_EXTERIOR\_FAR:** Nature scene manager. (For bigger exteriors.)

**ST\_EXTERIOR\_REAL\_FAR:** `Paging_Scene_Manager`. (For super big exteriors.)

**ST\_INTERIOR:** BSP scene manager.

```
mSceneMgr = mRoot->createSceneManager( ST_GENERIC, "MySM" );
```

Chapter 4, Slide 7  
Copyright © David Mount and Amitabh Varshney

## Ogre Scene Manager

**Scene Nodes and Entities:** The objects stored in your scene graph are entities, which are attached to scene nodes.

```
Entity* ent = mSceneMgr->createEntity( "Robot", "robot.mesh" );  
SceneNode* node = mSceneMgr->getRootSceneNode( )->  
    createChildSceneNode( "RobotNode" );  
node->attachObject( ent );
```

**Transformations:**

- Scene nodes can be transformed. The transformation is applied to this node and all descendents.

```
node->translate( 100.0, 10.0, 0.0 );
```

- You may also define a vector and use as an argument.

```
node->translate( Vector3( -20, 0, 10 ) );
```

Chapter 4, Slide 8  
Copyright © David Mount and Amitabh Varshney

## Ogre Scene Manager

**Rotations:** Specified by the axis about which the rotation is made.

**Pitch:** Rotation about x.

**Yaw:** Rotation about y.

**Roll:** Rotation about z.

**Coordinate System:** Transformations are made with respect to a particular coordinate system.

**TS\_LOCAL:** Relative to the **node's own** coordinate system. (Default for translation.)

**TS\_PARENT:** Relative to the **node's parent's** coordinate system. (Default for rotation.)

**TS\_WORLD:** Relative to the **world** coordinate system. (May be preferred for yawing rotations.)

```
node->pitch( Degree( 45 ), Node::TS_LOCAL );
node->translate( 100.0, 10.0, 0.0, Node::TS_WORLD );
```

Chapter 4, Slide 9  
Copyright © David Mount and Amitabh Varshney

## Typical Ogre Application: General Structure

```
#include <Ogre.h> // standard Ogre includes
#include <OIS/OIS.h>
#include <CEGUI/CEGUI.h>
#include <OgreCEGUIRenderer.h>

using namespace Ogre;
// my frame listener
class MyFrameListener : public FrameListener {
// ...see below for details
};

class Application { // main application object
// ...see below for details
};

int main( ... ) { // main (note: This will differ for Windows)
Application app; // declare application object
try {
app.go(); // run it
} catch( Exception& e ) {
// ... An exception occurred. Output an error message.
}
return 0;
}
```

10  
shney

## Typical Ogre Application: FrameListener

### FrameListener Structure:

```
class MyFrameListener : public FrameListener {
public:
    bool frameStarted( const FrameEvent& evt ){
        mKeyboard->capture();
        mMouse->capture();
        /* ...process inputs... */
        return true;
    }
    bool frameEnded( const FrameEvent& event ) { ... }
};
```

- A lot of information has been hidden. See the Ogre tutorials for more information.

Chapter 4, Slide 11  
Copyright © David Mount and Amitabh Varshney

## Typical Ogre Application: Application Skeleton

```
class Application { // main application object
public:
    void go(){ // main entry point for application
        // ... see details below
    }
    ~Application(){ /* ...any final cleanup... */ }
private:
    Root* mRoot; // root object
    MyFrameListener* mListener; // frame listener object
    Camera* mCamera; // camera
    SceneManager* mSceneMgr; // scene manager
    RenderWindow* mWindow; // render window
    OIS::InputManager* mInputManager; // see tutorials for more info
    // ... other private data here
public:
    // ... your public methods here
};
```

Why all the "m"s? This seems to be a convention among the Ogre tutorials.

Chapter 4, Slide 12  
Copyright © David Mount and Amitabh Varshney

## Typical Ogre Startup Sequence

### Startup Sequence for a Typical Ogre Program:

1. Create the **Root** object.
2. Define the **resources** that Ogre will use.
3. Select and set up the **RenderSystem** (e.g., DirectX, OpenGL).
4. Create the **RenderWindow** (the window which Ogre resides in).
5. Initialize the **resources** that you are going to use.
6. Create a **scene** using those resources.
7. Set up any third party **libraries** and **plugins**.
8. Create the desired number of **frame listeners**.
9. Start the **render loop** (event loop).

### Further Details:

1. Create the **Root** object.

```
mRoot = new Root();
```

Chapter 4, Slide 13  
Copyright © David Mount and Amitabh Varshney

## Ogre Startup Sequence

2. Define the **resources** that Ogre will use. This is done by loading the contents of the file resources.cfg.

```
resources.cfg
[General]
FileSystem=../../media
FileSystem=../../media/fonts
Zip=../../media/packs/skybox.zip

String secName, typeName, archName;
ConfigFile cf;
cf.load( "resources.cfg" ); // open configuration file
ConfigFile::SectionIterator seci = cf.getSectionIterator( );
while ( seci.hasMoreElements( ) ) {
    secName = seci.peekNextKey( ); // next section, e.g. "General"
    ConfigFile::SettingsMultiMap* settings = seci.getNext( );
    ConfigFile::SettingsMultiMap::iterator i;
    for ( i = settings->begin( ); i != settings->end( ); ++i ) {
        typeName = i->first; // read "foo=bar" pairs
        archName = i->second;
        ResourceGroupManager::getSingleton( ).
            addResourceLocation( archName, typeName, secName );
    }
}
addResourceLocation( "../../media", "FileSystem", "General" );
```

Chapter 4, Slide 14  
Copyright © David Mount and Amitabh Varshney

## Ogre Startup Sequence

### 3. Select and set up the **RenderSystem**.

- In the tutorials we let **Ogre select** the render system at startup (`mRoot->showConfigDialog( )`), but you can set this up manually.
- The following code **manually** sets up a Direct3D rendering system with a 800x600 window (not running full screen mode).

```
RenderSystem* rs = mRoot->getRenderSystemByName(
    "Direct3D9 Rendering Subsystem" );
mRoot->setRenderSystem( rs );
rs->setConfigOption( "Full Screen" , "No" );
rs->setConfigOption( "Video Mode" , "800 x 600 @ 32-bit colour" );
```

Chapter 4, Slide 15  
Copyright © David Mount and Amitabh Varshney

## Ogre Startup Sequence

### 4. Create the **RenderWindow** (the window which Ogre resides in).

```
mRoot->initialise( true, "My Game Window" );
```

### 5. Initialize the **resources** that you are going to use.

- Recall that resources include materials, meshes, skeletons, fonts, etc.
- For **large applications** they are usually loaded (and unloaded) **individually** as they are needed, or in **groups**.

```
ResourceManager::getSingleton().loadResourceGroup( "General" );
```

- For **small applications**, we can just load everything at **once**.

```
ResourceManager::getSingleton().initialiseAllResourceGroups();
```

Chapter 4, Slide 16  
Copyright © David Mount and Amitabh Varshney

## Ogre Startup Sequence

---

6. Create a **scene** using those resources.
  - This will involve creating a scene manager and may involve other elements such as creating a camera and viewport.

```
SceneManager* mgr = mRoot->createSceneManager(
    ST_GENERIC, "Default SceneManager" );
Camera* cam = mgr->createCamera( "MyCamera" );
Viewport* vp = mRoot->getAutoCreatedWindow( )->addViewport( cam );
```

7. Set up any third party **libraries** and **plugins**.
  - This may involve setting up input: **OIS** and a **FrameListener**.
  - Another useful tool is **CEGUI**, a graphical user interface for Ogre.
8. Create the desired number of **frame listeners**.

```
mListener = new MyFrameListener( ... );
mRoot->addFrameListener( mListener );
```

9. Start the **render loop** (event loop).

```
mRoot->startRendering( );
```

Chapter 4, Slide 17  
Copyright © David Mount and Amitabh Varshney

## Ogre Demo

---

Chapter 4, Slide 18  
Copyright © David Mount and Amitabh Varshney

## Summary

---

### Summary:

- More about Ogre

### What's Next?

- GPU architecture