Python Ogre

381

cube.mesh

Replace robot.mesh with cube.mesh for your assignment!

Lights, Camera, Input

```
import ogre.renderer.OGRE as ogre
import SampleFramework as sf


class TutorialApplication (sf.Application):

    def _createScene (self):
        pass

    def _createCamera (self):
        pass

    def _createViewports (self):
        pass


if __name__ == '__main__':
    ta = TutorialApplication ()
    ta.go ()
```

- You can create one or more Cameras
- You can create Viewports

You can still create a scene

Camera, camera node, movement

- "Camera" views the scene
- Like a sceneNode
  - position
  - yaw, roll, pitch
- You can attach to another sceneNode
- Camera position is relative to its parent's position
- Usage:
  - Create ONE camera
  - Attach to sceneNode - RTSNode (child of root)
  - move, orient RTSNode in response to UI

Switching between FP and RTS view

- Each entity in your game engine (381 Engine) has a corresponding python-ogre entity plus a python-ogre scene node (entNode)
- Create a child of entNode and call it FPSNode
- When you want to switch to First Person node attach camera to FPSNode.
  - Since FPSNode is child of entNode, it will move exactly with entNode
  - Now you have a FP view; as your camera renders what it sees (from the point of view of entity) to the screen
- When you want to switch to RTS view:
  - Detach camera from FPSNode
  - Attach camera to RTSNode

Camera creation and setup

```
self.camera =  self.sceneManager.createCamera ('PlayerCam')
# ask the sceneManager to create us a camera called
# PlayerCam
# cam = self.sceneManager.getCamera('PlayerCam')

self.camera.position = (0, 150, -500)
self.camera.lookAt ((0, 0, 0))
#lookAt makes it easy to make any sceneNode face and pos

self.camera.nearClipDistance = 5
# you don't see anything closer than 5 units
# you can also set far clip distance to stop rendering things that
# are very far away. Speeds up rendering.
```

Viewports and Render windows

The RenderWindow is where everything is displayed.
Your code must tell the RenderWindow which camera to use and what portion of the screen to use. "Portion" == Viewport
For now: One viewport, one camera, rendering to whole screen.

Usage:

- Create Camera
- Get Viewport from RenderWindow
- Set background color in viewport
- Set aspect ratio from information in viewport

Viewports, code

in _createViewports(self)

```
viewport = self.renderWindow.addViewport (self.camera)
# ask renderWindow (created by sample framework) to make a viewport using
# your newly created camera

viewport.backGroundColor = (0, 0, 0)
# set background color to black because ...

self.camera.aspectRatio = float (viewport.actualWidth) / float (viewport.actualHeight)
# tell ogre about the aspect ratio so rendered entities looks correctly dimensioned
```

Shadows - more gfx realism

Ogre currently has support for four types of Shadows:
1. Modulative Texture Shadows (SHADOWTYPE_TEXTURE_MODULATIVE) - The least intensive of the four. This creates a black and white render-to-texture of shadow casters, which is then applied to the scene.
2. Additive Texture Shadows (SHADOWTYPE_TEXTURE_ADDITIVE) - This technique is very much like the last one, but it is more more accurate than modulative shadows, as it correctly colors the shadows according to other light sources. However, it is more system intensive as it requires the number of lights + 2 passes.
3. Modulative Stencil Shadows (SHADOWTYPE_STENCIL_MODULATIVE) - This technique renders all shadow volumes as a modulation after all non-transparent objects have been rendered to the scene. This is not as intensive as Additive Stencil Shadows, but it is also not as accurate.
4. Additive Stencil Shadows (SHADOWTYPE_STENCIL_ADDITIVE) - This technique renders each light as a separate additive pass on the scene. This is very hard on the graphics card because each additional light requires an additional pass at rendering the scene.

Shadows - code

in _createScene

```
sceneManager = self.sceneManager
sceneManager.ambientLight = (0, 0, 0)          # black
sceneManager.shadowTechnique = ogre.
SHADOWTYPE_STENCIL_ADDITIVE

ent = sceneManager.createEntity ('Ninja', 'ninja.mesh')
# Yes there is a ninja mesh

ent.castShadows = True # False won't cast shadows - terrain for example

sceneManager.getRootSceneNode().createChildSceneNode ().attachObject (ent)
```

What are we going to cast shadows on?
What is the Ninja standing on?

Ground / Water plane

- Create a plane entity
- Attach plane entity to a scene node
- Texture it
- Tell scene manager to cast shadows on plane

We do not have a plane mesh instead we generate a mesh using MeshManager! This is new

```
plane = ogre.Plane ((0, 1, 0), 0)#Perpendicular to Y, facing +Y, distance 0 from origin
meshManager = ogre.MeshManager.getSingleton ()
meshManager.createPlane ('Ground', 'General', plane, 1500, 1500, 20, 20, True, 1, 5, 5, (0, 0, 1))
```

1500x1500 plane called "Ground" in the "General" category of resources

Ground plane entity and scene node

Create entity

```
ent = sceneManager.createEntity ('GroundEntity', 'Ground')
sceneManager.getRootSceneNode().createChildSceneNode ().attachObject (ent)
```

Texture it:

```
ent.setMaterialName ('Examples/Rockwall')
```

Tell SceneManager that the ground plane does not itself cast shadows

```
ent.castShadows = False
```

Before we can see the ninja and its shadows, we need to add lights!

Ogre Lights types

1. Point (LT_POINT) - Point light sources emit light from them in every direction.

2. Spotlight (LT_SPOTLIGHT) - A spotlight works exactly like a flashlight does. You have a position where the light starts, and then light heads out in a direction. You can also tell the light how large of an angle to use for the inner circle of light and the outer circle of light (you know how flashlights are bright in the center, then darker after a certain point?).

3. Directional (LT_DIRECTIONAL) - Directional light simulates far away light that hits everything in the scene from a direction. Lets say you have a night time scene and you want to simulate moonlight. You could do this by setting the ambient light for the scene, but that's not exactly realistic since the moon does not light everything equally (neither does the sun). One way to do this would be to set a directional light and point in the direction the moon would be shining.

Ogre light properties

Diffuse color - matte
Specular color - shiny

Point light - emits in all directions

```
light = sceneManager.createLight ('PointLight')
light.type = ogre.Light.LT_POINT
light.position = (150, 300, 150)

light.diffuseColour = (.5, .0, .0)
light.specularColour = (.5, .0, .0)
```

If you run the tutorial now, you should be able to see the ninja and a shadow!

Note you cannot see the light source only the effects of the emitted light.

Directional light - far away directional source

```
light = sceneManager.createLight ('DirectionalLight')
light.type = ogre.Light.LT_DIRECTIONAL
light.diffuseColour = (.5, .5, .0)         # yellow
light.specularColour = (.75, .75, .75)# light grey

# no position (very far away) only direction

light.direction = (0, -1, 1)
```

```
 0   -> not along the x - axis
-1   -> along the -y axis
+1  -> along the +z axis
```

Spotlight

```
light = sceneManager.createLight ('SpotLight')
light.type = ogre.Light.LT_SPOTLIGHT
light.diffuseColour = (0, 0, .5)        #blue
light.specularColour = (0, 0, .5)

light.direction = (-1, -1, 0)
light.position = (300, 300, 0)

light.setSpotlightRange(ogre.Degree(35), ogre.Degree(50))
# two cones. Inner cone is brighter
```

End of T2

Terrain, sky, fog

```
import ogre.renderer.OGRE as ogre
import SampleFramework as sf


class TutorialApplication (sf.Application):

    def _createScene (self):
        pass

    def _chooseSceneManager (self):
        pass

if __name__ == '__main__':
    ta = TutorialApplication ()
    ta.go ()
```

SceneManager and TERRAIN

self.sceneManager = self.root.createSceneManager (ogre.ST_EXTERIOR_CLOSE, 'TerrainSM')

self.root IS ROOT of the ogre class hierarchy

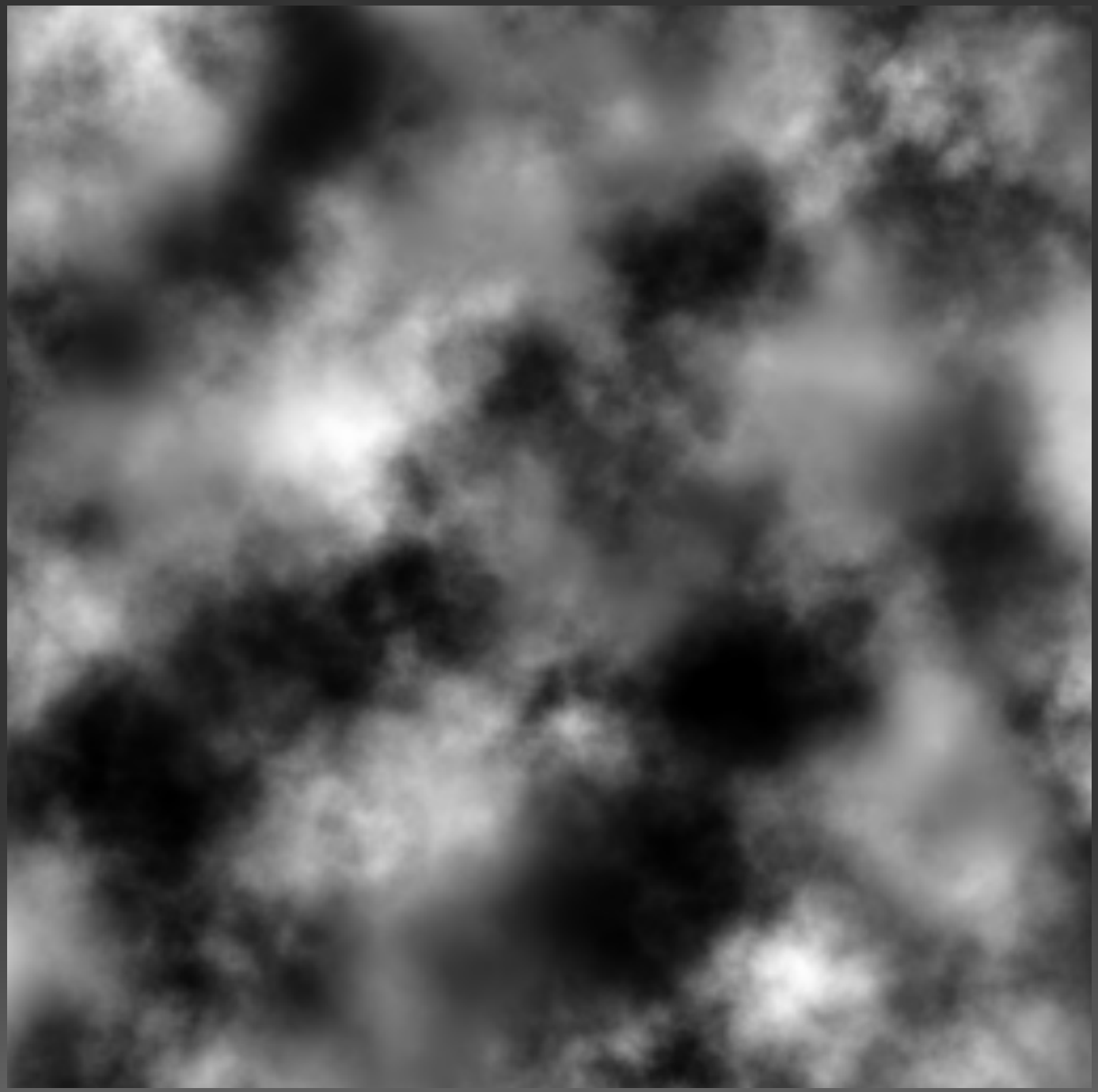
Sample Framework uses ogre.ST_GENETIC type scene manager which is the OctreeSceneManager.
Different scene managers can be specified in the plugins.cfg file.

self.root.getSceneManager('TerrainSM') gets you the sceneManager previously created above and named 'TerrainSM'

self.sceneManager.setWorldGeometry ("terrain.cfg")
uses information in terrain.cfg to create and texture terrain.

```
# Heightmap source
PageSource=Heightmap
# Heightmap-source specific settings
Heightmap.image=terrain.png
```

What is terrain.png and how does it determine terrain?



Terrain texture

Try them out

Skyboxes - a cube that contains everything

- Texture the INSIDE of the cube
self.sceneManager.setSkyBox (True, "Examples/SpaceSkyBox")
self.sceneManager.setSkyBox (False, "") turns of skybox

self.sceneManager.setSkyBox (True, "Examples/SpaceSkyBox", 5000, False)
#draw it 5000 units away and draw it last

self.sceneManager.setSkyDome (True, "Examples/CloudySky", 5, 8)
- spherical projection on cube's INSIDE
- bottom of the cube is untextured! make sure you have a ground plane/terrain
- 5 is curvature - you can experiment with values from 2 - 65
- 8 is texture tiling and is a float
- two more params, distance and drawFirst(T/F)

SkyPlane - only useful if you have high walls and cannot see out to world edge

plane = ogre.Plane ((0, -1, 0), -1000)
self.sceneManager.setSkyPlane (True, plane, "Examples/SpaceSkyPlane", 1500, 75)

self.sceneManager.setSkyPlane (True, plane, "Examples/CloudySky", 1500, 40, True, 1.5, 150, 150)

The seventh parameter allows you to specify the curvature of the SkyPlane, so that we are no longer using a Plane, but a curved surface instead. We also have to now set the number of x and y segments used to create the SkyPlane (initially the SkyPlane was one big square, but if we want curvature we need to have the plane made up of smaller squares). The eighth and ninth parameters to the function are the number of x and y segments, respectively.

sceneManager.setSkyPlane(False, ogre.Plane(), "")'

Fog

```
fadeColour = (0.9, 0.9, 0.9)
self.renderWindow.getViewport (0).backgroundColour = fadeColour
```

- Set fog color to very light grey
- Set the viewport (veiwport(0) because we only have one viewport) background color to be this color

```
self.sceneManager.setFog (ogre.FOG_LINEAR, fadeColour, 0, 50, 500)
self.sceneManager.setWorldGeometry ("terrain.cfg")
```

Linear fade light grey fog from 50 to 500. Under 50 - clear, Over 500 cannot see at all.