Panda3D Reference

Quick reference to common functions and their most common parameters

Loading Models

only model:
```
obj1=loader.loadModel('models/model.egg')
```

model with animations:
```
obj2=Actor.Actor('model/model.egg',
                 {'anim1':'models/modelAnim1.egg',
                  'anim2':'models/modelAnim2.egg'})
```

control functions:
```
obj2.play('anim1')     obj2.loop('anim2')
obj2.stop('anim1')     obj2.pose('anim2', 0)
```

Intervals

Actors:
```
int1=actorInterval('anim1', loop=1, duration=100,
                   startFrame=2, endFrame=102,
                   playRate=.5)
```

Sounds:
```
int2=soundInterval('sound1' loop=1, duration=100,
                   volume=.5, startTime=.1)
```

Functions:
```
int3=Func(obj.setPos, Point3(0,0,0))
int4=Func(obj.hide)
```

Lerp: (linear interpolation)
```
int5=LerpFunc(obj.setX, 0, 1000,duration=10)
int5=obj.posInterval(duration=1, Point3(0,0,0),
                     Point3(1000,0,0))
```

Sequences & Parallels:
```
inOrder=Sequence(int1,int2,int3,int4,int5)
alltogether=Parallel(int1,int2,int3,int4,int5)
```

control functions:
```
int.start()  int.pause() int.setT(1)int.isPlaying()
int.finish() int.resume()int.getT() int.isStopped()
                         int.loop()
```

Messages

The DirectObject class has support for messages.
Classes created as subclasses of Direct Object can accept messages.

accepting & ignoring messages:
```
class Foo(DirectObject.DirectObject):
        self.accept('event1',self.fun1)
        self.acceptOnce('even2',obj.show)
        self.ignore('event2')
        self.ignoreAll()
```

sending messages:
```
messenger.send('event1')
interval.setDoneEvent('event2')
```

debugging messages:
```
messenger.toggleVerbose()
print messenger
messenger.clear()
```

Collision detection

handler:
```
collHandEvent=CollisionHandlerEvent()
collHandEvent.addInPattern ("enter%in")
```

traverser:
```
collTraverser=CollisionTraverser()
base.cTrav=collTraverser
collTraverser.getNumColliders()
collTraverser.clearColliders()
```

collision spheres:
```
collSphere=CollisionSphere( x, y, z, radius)
cSphereStr ='CollisionHull'
collSphereNode=CollisionNode( collSphereStr )
collSphereNode.addSolid(collSphere)
```

selective collisions:
```
collSphereNode.setIntoCollideMask(BitMask32.bit(1))
collSphereNode.setFromCollideMask(0)
collTraverser.addCollider(collSphereNode,
                          collHandEvent)
accept( 'enter' + collSphereStr, collideFnc)
```

debugging:
```
debugCollObj=(node.attachNewNode(collSphereNode))
debugCollObj.show()
```

on Collisions:
```
def collideFnc(collEntry):
        print  collEntry.getFromNode().getName()
        print  collEntry.getIntoNode().getName()
```

orientation of panda3d: (right, forward, up)

take camera control away from the mouse: `base.disableMouse()`

camera object name : `camera`

Transformations, Rotations, Scaling, etc.

to scale:
```
obj.setScale(VBase3(0,0,0)) or obj.setScale(0,0,0)
obj.setSx(0) obj.setSy(0) obj.setSz(0)
```

to move:
```
obj.setPos(VBase3(0,0,0)) or obj.setPoint(0,0,0)
obj.setX(0) obj.setY(0) obj.setZ(0)
```

to rotate:
```
obj.setHpr(VBase3(0,0,0)) or obj.setHpr(0,0,0)
obj.setH(0) obj.setP(0) obj.setR(0)
```

to color:
```
obj.setColor(VBase4(1,1,1,1))
or obj.setColor(1,1,1,1)
obj.clearColor()
```

Sounds

supported file types: midi, wav, mp3

to Load:
```
loader.loadSfx('sounds/sound.wav')
loader.loadMusic('sounds/sound.wav')
```

control functions:
```
sound.play()        sound.setTime(.5)
sound.stop()        sound.setVolume(0)
base.disableAllAudio()
base.enableAllAudio()
base.enableMusic(bEnableMusic)
base.enableSoundEffects(bEnableSoundEffects)
```

Tasks

Creating:
```
def exampleTask(task):
      if task.time < 2.0:
            return Task.cont
      print 'Done'
      return Task.done
```

Task Manager:
```
taskMgr.add(exampleTask,'aTask')
taskMgr.Remove('aTask')
taskMgr.doMethodLater(10, aFunc, 'otherTask')
```

Finite State Machines

Creating:
```
class NewStyle(FSM.FSM):
    def enterRed(self):
        print "enterRed(self,'%s','%s')" %
              (self.oldState, self.newState)
    def filterRed(self, request, args):
        if request == 'advance':
            return 'Green'
        return self.defaultFilter(request, args)

    def exitRed(self):
        print "exitRed(self,'%s','%s')" %
              (self.oldState, self.newState)
```

Using:
```
aFSM= NewStyle('a new FSM')
print aFSM.state
aFSM.request('Red')
aFSM.request('advance')
```

Lighting

Ambient Light:
```
aLight = AmbientLight( 'ambientLight' )
aLight.setColor( Vec4( 0.4, 0.4, 0.4, 1 ) )
```

Directional Light:
```
dLight = DirectionalLight( 'directionalLight' )
dLight.setPoint(Vec3(0,0,0))
dLight.setDirection(Vec3( -0.3, .2, .10 ))
dLight.setColor(Vec4( 0.4, 0.4, 0.4, 1 ))
```

Point Light:
```
pLight = PointLight( 'pointLight' )
pLight.setPoint(Vec3(0,0,0))
pLight.setColor(Vec4( 0.4, 0.4, 0.4, 1 ))
```

Spot Light:
```
spot = Spotlight( 'spotlight' )
spot.setPoint(Vec3(0,0,0))
spot.setColor( Vec4( 0.4, 0.4, 0.4, 1 ))
spot.setDirection(Vec3( -0.3, .2, .10 ))
spot.setLens(PerspectiveLens())
spot.getLens()
```

general stuff:
```
lightAttributes=LightAttrib.makeAllOff()
lightAttributes.addLight( aLight)
render.attachNewNode(aLight.upcastToPandaNode())
render.node().setAttrib(lightAttributes )
```