Panda3D による 3DCG プログラミング教育

箕原辰夫
e-mail: minohara@cuc.ac.jp
千葉商科大学政策情報学部

◎Key Words 3DCG，シーン・グラフ，プログラミング，Panda3D，Python

1. はじめに

これまで 3DCG プログラミング教育においては、Python[(1)] と OpenSceneGraph[(2)]の Python 移植版を使ってきましたが、2011 年度から、Panda3D[(3)]を使うことにしました。その理由は、基本的にはシーン・グラフ（あるいはシーン・ノード・グラフ：Scene Node Graph）を基本としたプログラミング方法は変わらないのですが、同じ内容を記述するのに Panda3D の方が簡潔に記述できるからです。また、PyODE[(4)]（物理エンジン）との連携も行なうことができるという利点もあります。更に、Panda3D は C/C++用のライブラリと共に Python 用のライブラリが基本的なモジュールとして開発されており、Python 用の環境が安定しているという理由もあります。OpenSceneGraph の場合は、外部のプロジェクトとして Python 移植版が作られていましたので、本家の開発と連動していないという問題点がありました。Panda3D は、この問題が解決されています。この報告では、このライブラリの導入の仕方、およびどのような記述方法であるかを説明した後、この環境を使った教育成果の評価、卒業研究における成果について報告します。

2. 導入

2.1 インストール

Python には、32bit 版と 64bit 版があるのですが、Panda3D はこの報告を記述している時点では、1.7.2 版がリリース版になっており、これは 32bit 版しかありません。そのため、Windows でも Macintosh でも、32bit 版で Python の環境を統一しておく必要があります。そのため、32bit 版のみがリリースされている Python 2.6.6 版をインストールする必要があります。Python 2.7 版以降になると 32bit 版と 64bit 版が混在するために、うまく動作しない可能性があるからです。

Windows 版では、バイナリ版が用意されていますので、それをインストールすれば環境設定ができてしまいます。非常に簡単に学生でもすぐにインストールができます。Macintosh 版では、ソースファイルからコンパイルしてインストールする必要があります。このとき、Panda3D が前提としているライブラリのうち、たとえば X11 ウィンドウのライブラリなど、不要なライブラリがないことをコンパイルのオプションとして指定することができます。ただし、Cg のライブラリがあった方が、シェーディングなどで影をつけるときやバンプマッピングなども機能しますので、nVidia の Web サイトから予め Cg Toolkit[(5)]をダウンロードしてインストールしておいた方が良いでしょう。ソースファイルをコンパイルした後は、インストールのアプリケーションができますので、これを使ってインストールを行ないます。

Panda3D の Python 用のライブラリは、通常の Python が置かれるライブラリとは別の場所に置かれます。そのため、Python のオリジナルの IDE（開発環境）のエディタから動かすためには、Windows でも Macintosh でも特定の環境設定をしなければなりません。それが煩わしい場合、特に学生がそのようなカスタマイズに精通していない場合は、Panda3D によってインストールされる実行フォルダの中に、カスタマイズされた Python のインタープリタ（ppython）がありますので、Windows であれば pyScripter[(6)]、Macintosh であれば Editra[(7)]などの外部開発環境を用いて、Python のスクリプト実行時に通常の python インタープリタの代替として、このインタープリタを指定して実行させるように設定しておくと簡単に利用することができます。

2.2 Panda3D の機能

Panda3D では、通常のシーン・グラフベースのライブラリができることは、一通り実現されています。モデルなども、読み込むファイル形式が限られているものの、変換ソフトウェアを用いて、変換することができ、多くの形式のファイルを読み込むことができます。標準では、Egg 形式と呼ばれるモデルのファイル形式をサポートしています。今回の卒業研究では、DAE 形式のファイルを利用することが多くなりました。

Panda3D では、OpenSceneGraph のようなプリミティブ図形の作成機能が含まれていません。例えば立方体を表示したければ、OpenGL のようにポリゴン 6 枚を貼り合わせて立方体にするか、立方体のモデルを読み込む必要があります。背景の設定もできないので、半球のような、背景となる大きな立体のモデルを背後に読み込み、そのモデルの内側に背景のテキスチャを貼り付けるという形でプログラムする必要があります。

2.3 Panda3D を用いた記述方法

Panda3D では、シーン・グラフを構成するためのいくつかの大域変数が用意されていて、その変数が指し示すオブジェクトを利用することにより、シーン・グラフを作っていきます。まず、下のスクリプトでは、ShowBase クラスと Loader クラスのオブジェクトを作成し、*render* と命名され

たシーン・グラフのルートのオブジェクトに読み込んだオブジェクトを追加しています。あとは、ShowBase クラスのオブジェクトが自動的に、マウスからの入力を拾って、カメラの位置などを変える操作を行なってくれます。

```
from direct.showbase.ShowBase import *
from direct.showbase.Loader import *

base = ShowBase( )
loader = Loader( base )
model = loader.loadModel( "box.egg" )
model.reparentTo( render )
base.run( )
```

ところが、Panda3D では、この記述をさらに簡略化させるモジュールが用意されています。このモジュールを使うと、ShowBase クラスのオブジェクトが *base* という大域変数で、また Loader クラスのオブジェクトが *loader* という名前の大域変数で予め用意されていますので、上記のようにいちいち作成する必要がなく、同じことを記述するのに、*render* オブジェクトと共にそれらの変数を利用して以下のように短く記述することができます。

```
import direct.directbase.DirectStart

model = loader.loadModel( "box.egg " )
model.reparentTo( render )
run( )
```

上記の記述で読み込んで表示しているのは、標準で用意されている縦横幅がそれぞれ 1 unit の大きさの立方体のモデルです。なお、読み込んだモデルは、NodePath というクラスのオブジェクトになっており、以下のようなメソッドを利用して、移動・拡縮・回転させることが可能になっています。

```
model.setPos( 0, 0, 0 )     # X は左右 Y は前後 Z は上下
model.setScale( 0.5, 0.5, 0.5 )   # setScale( 0.5 )でも良い
model.setHpr( 90, 0, 0 )    # Hue / Pitch / Roll
```

これに、キー入力などのハンドラを組み込み、モデルの位置や角度を変えるようなスクリプトを記述してみると、次のようになります。

```
import direct.directbase.DirectStart
from panda3d.core import *

# モデルの読込みとシーン・グラフへの追加
model = loader.loadModel( "box.egg" )
model.reparentTo( render )

# カメラの位置・向きの設定
base.cam.setPos( 0, -10, 5 )
base.cam.lookAt( 0, 0, 0 )

# ハンドラとなる関数の定義
def forward( ): model.setY( model.getY( ) - 1    )
def backward( ): model.setY( model.getY( ) + 1    )
def turnright( ): model.setH( model.getH( ) - 10 )
def turnleft( ): model.setH( model.getH( ) + 10 )

# ハンドラの登録
base.accept( "arrow_up",    forward )
base.accept( "arrow_down", backward )
base.accept( "arrow_right", turnright )
base.accept( "arrow_left", turnleft )

run( )
```

簡単ながらも、これでユーザからの入力も受け付ける一通りのアプリケーションとして機能させることができます。

2.4　衝突感知・物理エンジンの組込み

Panda3D は、その内部に衝突感知用のモジュールが組み込まれています。これを利用して、物体同士が衝突したことを知ることができるのですが、この衝突感知の対象とするために、見えない物体オブジェクトを追加する必要があります。それらの物体オブジェクトの形は、衝突を受ける側と衝突する側でそれぞれ制限されていて、衝突を受ける側は、平面や立方体や球形などがありますが、衝突する側は球形の立体だけになっています。これらの物体オブジェクトの形は、今後増やされていく予定になっているようです。物体同士の衝突は、実際にはそれらの物体オブジェクトの衝突をハンドラで感知するという形式になります。

以下のスクリプトは、衝突しているかどうかを表示するように衝突を感知するオブジェクトに指定しています。ハンドラは、衝突感知のイベントを登録しておけば、定義した関数をハンドラとして指定することができます。また、衝突のための物体オブジェクト自体も表示するように指定していますので、半透明の白で表示され、衝突が近くなると半透明の黄色で表示されます。衝突した場合は、衝突したポイントが赤色のベクトルで表示されます。

```
import direct.directbase.DirectStart
from panda3d.core import *

# 衝突検知オブジェクト
base.cTrav = CollisionTraverser( "traverser" )
base.cTrav.showCollisions( render )

# ハンドラ関数の定義
def collisionhandler( entry ): print "Collision"
def separatehandler( entry ): print "Separate"

# 衝突検知イベントの登録
handlerevent = CollisionHandlerEvent( )
handlerevent.addInPattern( "into-%in" )
handlerevent.addOutPattern( "outof-%in" )

# イベントに対するハンドラ関数の登録
base.accept( "into-Smiley", collisionhandler )
base.accept( "outof-Smiley ", separatehandler )
```

```
# 衝突されるオブジェクト
smiley = loader.loadModel( "smiley" )
smiley.reparentTo(render)
smileyObj = CollisionNode( "Smiley" )
smileyObj.addSolid( CollisionBox(Point3(0,0,0), 1, 1, 1) )
smileyObj.setIntoCollideMask( BitMask32.bit(0) )
smileyCollision = smiley.attachNewNode( smileyObj )
smileyCollision.show( )

# 衝突するオブジェクトの設定
frowney = loader.loadModel( "frowney" )
frowney.reparentTo(render)
frowneyObj = CollisionNode( "Frowney")
frowneyObj.addSolid( CollisionSphere(0, 0, 0, 1.2) )
frowneyObj.setFromCollideMask( BitMask32.bit(0) )
frowneyCollision=frowney.attachNewNode(frowneyObj )
frowneyCollision.show( )
base.cTrav.addCollider( frowneyCollision, eventhandler )

# 衝突するオブジェクトのアニメーションの設定
frowney.posInterval(
      5, Point3(5, 0, 0), startPos=Point3(-5, 0, 0) ).loop( )

# カメラの位置と向きを設定
base.cam.setPos(20, -20, 3)
base.cam.lookAt(0, 0, 0)

run( )
```

物理エンジンのモジュールを使った記述は、ここでは分量が足りないのでスクリプトすべてを記述することは省略しますが、まず、次のように物理エンジンの世界を構築します。

```
from panda3d.ode import *

world = OdeWorld( )
world.setGravity( 0, 0, -9.81 )
space = OdeSimpleSpace( )
space.setAutoCollideWorld( world )
```

そして、各モデルに対して、物理的な本体と質量を設定します。質量の記述からもわかるように、慣性モーメントなども計算されます。

```
body = OdeBody( world )
body.setPosition( model.getPos( render ) )
body.setQuaternion( model.getQuat( render ) )
mass = OdeMass( )
mass.setBox(11340, 1, 1, 1)
body.setMass( mass )
```

そして、モデルの幾何図形的な設定を行ないます。

```
boxGeom = OdeBoxGeom( space, 1, 1, 1 )
boxGeom.setCollideBits( BitMask32.bit(1) )
boxGeom.setCategoryBits( BitMask32.bit(0) )
boxGeom.setBody( boxBody )
```

最後に、シミュレーションを行なう関数を定義し、それをアニメーションのタスクとして登録します。

```
def simulation( task ):
   space.autoCollide( )
   world.quickStep( globalClock.getDt( ) )
   box.setPosQuat(render, boxBody.getPosition(),
         Quat( boxBody.getQuaternion( ) ) )
   return task.cont

taskMgr.doMethodLater(0.5, simulation, " Simulation")
```

3. 卒業研究での使用

3.1 使用の経緯

今回卒業研究で Panda3D を使用した学生達には、３年次の研究会の春学期において、Python を用いた基本的なプログラミングの方法、および PyQt[8]を利用したプログラミングを学ばせていました。秋学期においては、OpenSceneGraph を Python から利用したプログラミングを学ばせていました。

最終作品には OpenSceneGraph を使う前提で、２次元版の作品を、PyQt を用いて試作していました。４年次の卒業研究の中間発表を終えた後の 11 月から急遽導入し、２ヶ月ぐらいの間、週１コマの時間だけライブラリを学びながら、自分達で作品を完成させました。中間発表のときに講評の先生方から出された問題点は、Panda3D のモジュールを使うことにほとんど解決できたようです。

3.2 作品の完成度

卒業研究の学生３人の作品のスナップショットを掲載しながら、作品の完成度について報告していきたいと覆います。まず、Panda3D には、DirectGUI というモジュールが用意されており、簡単な２次元上のボタンやポップアップ・メニューなどの GUI コンポーネントを容易に組み込むことができます。そのため、各シーンにおいて、ユーザからの入力を得るのも、シームレスに行なうことができました。

図１の作品は、車庫入れをうまく行なうためのトレーニングのためのソフトウェアです。この作品では、運転席から見たビューにミラーを３ヶ所入れています。衝突検知のために、表示をさせない小さな球のオブジェクトを用意し、それを車体の周りに貼り付けています。また、複数の視点から見えるようにしています。

図２に掲げた作品は、部屋のレイアウトを行なうためのシミュレータになっています。この作品では、Cg のレンダリング機能を用いて、陰影が入った形でのレンダリングをしています。また、図３のように、Panda3D の GUI を利用し、画像の入ったボタンを使ってユーザに家具などを選ばせるようにしています。

図１ 学生の作成した車庫入れ練習ソフトウェア

図２ 学生の作成した部屋のレイアウトエディタ

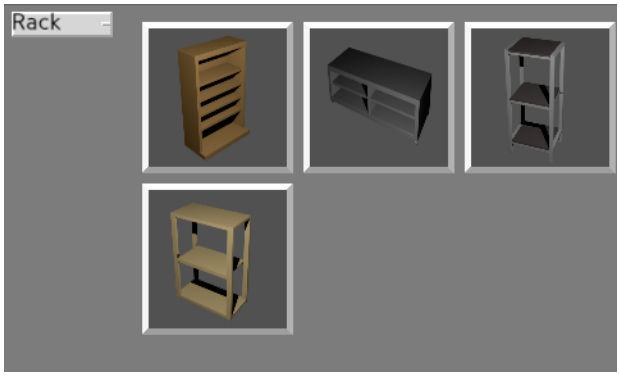

図３ Panda3D の GUI を用いて家具を選ばせるパネル

図４の作品は、ブロック落としゲームになっていますが、先に説明した Panda3D の物理エンジンを利用して、ブロックを上から落とし、積み重ねられるようにしています。この作品も複数の視点から見ることが可能になっています。

4. 教育教材としての評価

OpenSceneGraph を用いていた前年度の卒業研究の場合は、ブロックをいくつか組み合わせるようなソフトウェアが主体で終わっていました。これは、それまでの３年次までの研究会において、充分に Python や Python を用いた GUI

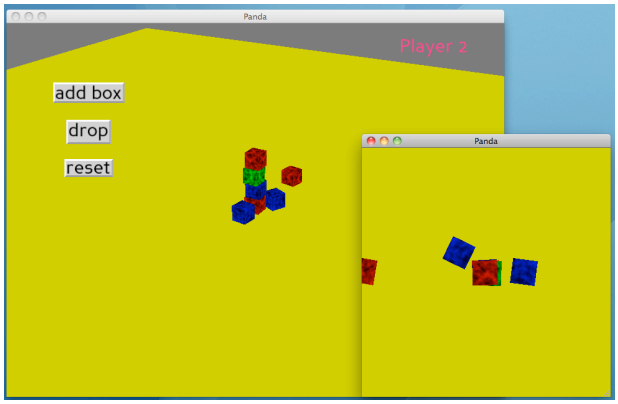

図４ 学生の作成したブロック落としゲーム

プログラミングを学ぶ態勢にしていなかったことにも起因しています。それに加えて、Panda3D では簡素ながらも GUI のライブラリもついており、OpenSceneGraph と PyQt を併用するよりはずっと楽に GUI を持つ 3 次元のプログラム作品を作りやすいということがわかりました。また Python と Panda3D の記述の簡潔さから、短期間で学生が習熟できた結果も見逃すことはできません。

今後は、iPad や Android タブレットなどの情報端末用のアプリケーションのリリースの需要も出てくるでしょう。OpenGL ES[9]を利用して、Unity3D[10]は JavaScript でスクリプトを記述できるようになっています。また、OpenSceneGraph は、C++からの利用だけに限られていますが、iOS/Android 版もリリースされています。残念ながら、Panda3D は、現在は iOS/Android 版のリリースはないようです。

5. おわりに

今回の Panda3D を利用した結果、学生の習熟度をアップさせるのにも Python と Panda3D の組み合わせは効果的であるということがわかりました。しかしながら、iPad や Android タブレットなどの情報端末用のアプリケーションを開発する需要がこれから起こりつつあるため、Panda3D の iOS あるいは Android クロス開発環境が今後リリースされることを期待すると共に、Unity3D などの開発環境と比較してみる必要があるでしょう。

参考文献

(1) “Python,” http://python.org, (viewed 2012).
(2) “OpenSceneGraph,” http://openscenegraph.org, (viewed 2012).
(3) “Panda3D,” http://panda3d.org, (viewed 2012).
(4) “PyODE,” http://pyode.org, (viewed 2012).
(5) “Cg Toolkit”, http://developer.nvidia.com/cg-toolkit, nVidia (viewed 2012).
(6) “pyScripter,” http://code.google.com/p/pyscripter, (viewed 2012).
(7) “Editra,” http://editra.org/, (viewed 2012).
(8) “PyQt”, http://www.riverbankcomputing.co.uk/software/pyqt, (viewed 2012)
(9) “OpenGL ES,” http://jp.khronos.org (viewed 2012).
(10) “Unity3D,” http://unity3d.com (viewed 2012).