Unity 3D を用いた 3 次元プログラミング教育

箕原辰夫[*1]
Email: minohara@cuc.ac.jp

*1: 千葉商科大学政策情報学部

◎Key Words Unity, 3DCG, プログラミング

1. はじめに

Unity Technologies 社からフリーの開発環境としてリリースされている Unity 3D[(1)] (以下 Unity と略記する) を用いて、3 次元プログラミングの教育を卒業研究の学生に対して行なった。この環境は、半分は Blender[(2)] のような 3 次元のモデリング環境になっており、もう半分は JavaScript (C#や Python なども試用可能) のプログラミング環境になっている。これからのゲームエンジンは、このようにモデリングとプログラミングを合わせた形で開発していくのではないかと考えられる。この Unity はライセンスを購入することで、iOS や Android といったタブレットの開発環境としても利用することができる。また、標準でサーバとクライアントから構成されるネットワークゲームも作成することができる。この論文では、卒業研究で作成された作品を題材にしながら、Unity の可能性およびこれからの 3 次元プログラミング教育について考えていく。

2. Unity の考え方

Unity では、各オブジェクトが独自に動作する前提でプログラミングを行なっていかなければならない。これらのオブジェクトは、最初からシーンに配置しておくこともできるし、配置されたシーンの中のオブジェクトからスクリプトの中から新たに作り出すこともできる。これらのオブジェクトが並行に動き、その保持する関数をお互いに呼び出すという形で動作が進んでいく。

2.1 プロジェクトとシーン

全体はプロジェクトという形でまとめられている。また、その中で複数のシーンを持つことができる。1 つのシーンには、Main Camera という名前でそれを画面に投影するためのカメラが標準で付け加えられている。シーンの中に、必要なオブジェクトを配置することができる。Main Camera 以外にもカメラのオブジェクトを配置することも可能になっている。シーンの中は、Blender などの通常の 3 次元モデリングソフトウェアと同様な操作感で、編集することが可能になっている。

2.2 オブジェクト

オブジェクトは、GameObject クラスのオブジェクトとして配置される。1 つのオブジェクトの中には、複数のコンポーネントを持つことができる。このコンポーネントには、効果 (Effect) に関係するもの、レンダリングに関係するもの、物理特性に関連するもの、ネットワークに関連するものなどが含まれている。このコンポーネントの一種として、スクリプトが含まれており、スクリプトを特定のオブジェクトに付随させることによって、スクリプトがそのオブジェクトにおいて有効化される。

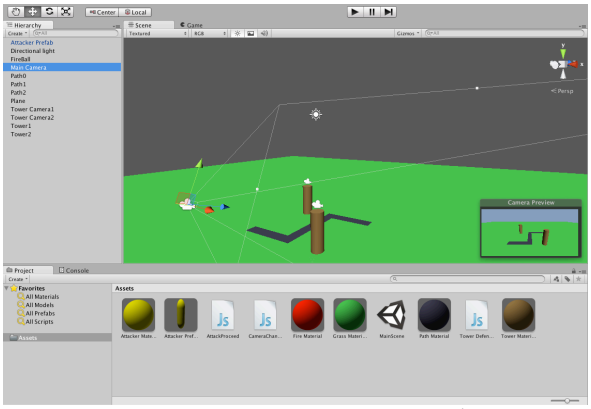

図 1 シーンでの Main Camera とオブジェクト

2.3 並行プログラミング

オブジェクトに付随されたスクリプトは、実行が始まると一斉に呼び出されることになる。これは、シーンに配置したオブジェクトすべてに対して行なわれる。そのため、なんらかイベントが起こらない限り、処理を必要としないオブジェクトは、毎時のレンダリングで呼び出される関数の中身を空にしておく。その替わりに、イベントに対してコールバックされる必要な関数を定義しておき、特定のイベントが起こったときだけ処理を行なう。オブジェクト間の同期は、お互いのオブジェクトを参照しておいて、互いの特定の属性値を見ながら行なうようにする。

2.4 アセット

Unity 有償版などでは、標準に用意されているオブジェクトだけでなく、より高機能のレンダリングのコンポーネントやスクリプトを持つオブジェクトが用意されている。また、無償あるいは有償で、特定の機能や形状を持つオブジェクトがアセットという形で提供されている。このアセットを商品として売ることが、Unity

Technologies 社の一つの財源になっている。アセットでは、予めスクリプトや形状が専門家によってデザインされているオブジェクト（これを Unity では Prefab と呼んでいる）が用意されており、これをプロジェクトのアセットが格納されているフォルダに移動すれば、すぐにそのプロジェクトで利用することが可能になる。

3. Unity のプログラミングの記述方法

以下に Unity を利用するときに、キーとなる JavaScript のプログラミングによるオブジェクトやネットワークなどの記述方法を紹介していく。

3.1 各オブジェクトのプログラミング

Unity の基本として、各オブジェクトに対して、最初に 1 回だけ呼び出される Start 関数と、毎回のレンダリングの際に呼び出される Update 関数の中に、ほぼ中心となる処理の記述を行なう形になっている。

```
function Start( ) {
    // 最初に 1 回だけ行なう処理
}
function Update( ) {
    // レンダリング時に毎回呼び出される処理
}
```

位置、姿勢制御のために、各オブジェクト（GameObject と呼ばれるクラスに属するオブジェクト）には、transform という属性が用意されている。これは、Transform クラスのオブジェクトになっていて、次のようなメソッドや属性を持っている。

```
transform.Translate (0, 10, 0);
transform.Rotate (0, 10, 0);
transform.LookAt(Vector3.zero, Vector3.up);
transform.position = new Vector3(0,10,0);
transform.rotation = Quaternion.identity;
```

レンダリングの時間間隔を得るために、Time クラスにおいて、deltaTime 属性の中に時間間隔が保持されている。これを次のように足しあわせることによって、起動時からどの程度時間が経過したのかを求めることができる。

```
var elapsedTime: float = 0;
    // 経過時間を保持する変数
function Update( ) {
    elapsedTime += Time.deltaTime;
}
```

3.2 他のオブジェクトへの参照

Unity においては、次のような記述をすることで、他の物体（GameObject クラスのオブジェクト）を参照することができる。

```
var otherObject =
    GameObject.Find(“他のオブジェクトの名前”);
```

また予め定義されている他の物体（GameObject クラスのオブジェクト）を新たに表示するには、Instatiate 関数を用いて、その作成を行なう。

```
var createdObject =
    Instatiate(“オブジェクトの名前”, 位置, 姿勢 );
```

3.3 テキスト表示や GUI

画面上にテキストを表示したい場合は、GUIText の GameObject を予め配置しておく。そして、Find でそのオブジェクトを見つけ出し、そこに書き込むことができる。下記は、GUIText というそのままの名前でテキストを表示するオブジェクトがある場合の記述となっている。

```
var guiTextObject = GameObject.Find( “GUIText” );
guiTextObject.guiText.text = “表示テキスト”;
```

Main Camera のコンポーネントとして、GUILayer を追加すると、OnGUI 関数を定義することにより、ボタン表示やテキスト入力がスクリプトから可能になる。下記のスクリプトでは、GUI クラスを用いているので、GUI の場所やサイズを指定する Rect クラスのオブジェクトを伴っているが、GUILayer クラスを用いた場合は、指定の必要はなく、自動的にレイアウトされる。

```
function OnGUI {
    if ( GUI.Button( Rect (25,25,100,30), "Click Me") ) {
        // ボタンが押されたときの処理
    }
    var result : String = “Default Answer”;
    var rect = Rect (25,25,100,30);
    result = GUI.TextField ( rect, result, 25);
}
```

3.4 物理的な挙動・衝突感知

物理的な挙動のために、各オブジェクト（GameObject）には、rigidbody という属性が用意されている。力学的な力を加えたり、現在の速度・角速度などを制御したりするために、次のようなメソッドや属性が用意されている。

```
rigidbody.AddForce (Vector3.up * 10);
rigidbody.AddForce (0, 10, 0);
rigidbody.AddTorque (Vector3.right * 10);
rigidbody.velocity = new Vector3(0,10,0);
rigidbody.rotation = Quaternion.identity;
if ( rigidbody.angularVelocity.magnitude > 5 ) {  }
```

各物理的なオブジェクトには衝突感知を行なうために、Collider クラスのサブクラス（BoxCollider や CapsuleCollider など）のオブジェクトになっている

Collider コンポーネントが付属している。このコンポーネントが付属している場合、OnCollisionEnter 関数と OnCollisionExit 関数をスクリプトに定義することによって、衝突がどのように起こったかを衝突時に知ることができる。

```
function OnCollisionEnter( collsionInfo: Collision ) {
        guiTextObject.guiText.text = “衝突発生”;
}
```

3.5 クライアント・サーバの構成

ネットワークを介した制御では、1 つの Unity で作成されたアプリケーション（あるいはプロジェクト）が、サーバの役割を担う。サーバの役割を持ったアプリケーションは、次のような関数などを用意して、サーバを立ち上げる。

```
function LaunchServer ( ) {
    Network.incomingPassword = "HolyMoly";
    var useNat = !Network.HavePublicAddress( );
    Network.InitializeServer(32, 25000, useNat);
}
```

クライアント側は、サーバに対して、接続を要求する。

```
    Network.Connect("127.0.0.1", 25000);
```

接続されたサーバでは、OnPlayerConnected 関数が定義されていれば、その関数が自動的に呼び出され、クライアントの情報を得ることができる。

```
var playerCount: int = 0;
function OnPlayerConnected( player: NetworkPlayer )
{
    Debug.Log( "Player " + playerCount +
    " connected from " + player.ipAddress + ":" +
    player.port );
    playerCount++;
}
```

クライアント側からサーバ側に処理を行なわせる場合は、遠隔手続き呼出し（RPC: Remote Procedure Call）を行なう。これは、@RPC タグを伴って、クライアント側とサーバ側に同じ関数を定義するもので、クライアント側には関数の雛形だけがあれば良い。実質的な処理はサーバ側で記述する。

クライアント側の記述

```
var   viewid : NetworkViewID;
    viewid = Network.AllocateViewID( );
    networkView.RPC( "MoveIt",
RPCMode.AllBuffered, viewid, transform.position );

@RPC
function MoveIt( viewID : NetworkViewID, location :
Vector3 ) {   }
```

サーバ側の記述

```
@RPC
function MoveIt( viewID : NetworkViewID, location :
Vector3 ) {
    var   clone : Transform;
    clone = Instantiate( cube, location,
        Quaternion.identity ) as Transform;
    var   nView : NetworkView;
    nView = clone.GetComponent( NetworkView );
    nView.viewID = viewID;
}
```

上記のスクリプトの記述からわかるように、サーバ側にもクライアント側にも、動作の直列実行性を保証するために、Network View コンポーネントをスクリプトが添付されている Main Camera などに用意しておく必要がある。なお、サーバ側からクライアント側に情報を返すときは、クライアント側に@RPC タグのついた関数を用意し、それを上記の networkView.RPC 関数を使ってサーバ側から呼び出す。この場合、サーバ側にも@RPC タグのついた雛形関数を同じように用意する。

4. 卒業研究における指導

卒業研究においての指導は、2012 年度に行なったが、特に学生側も教員側も初めての開発環境なので、いろいろな試行錯誤を伴った。しかしながら、概して一からプログラムで組んでいく環境よりも、より複雑な作品内容を実現することができた。

4.1 指導の方法

最初は、簡単なプロジェクトを JavaScript の勉強も兼ねて作ってみた。学生は、Java は既習だったが、JavaScript は知らない状態だったので、その差異の説明もした。しかし、記述例を見てもわかる通り、Unity の JavaScript は、通常の JavaScript とは異なり、変数などの型をコロンで区切って指定することが可能になっている。この部分で型指定ができるので、コンパイルして、通常の JavaScript よりも高速な実行をすることを可能にしたようだ。この部分は、通常の JavaScript よりも Java に近いので、学生には、受け入れやすかったのではないかと考えられる。

4.2 作品例

指導した学生の作品の中から、いくつかの作品をその特徴と共にピックアップしていきたい。

★大きなシーンと数多くのオブジェクトを用いて、緊急時の行動のためのシミュレーションを行なった作品

この作品では、壁や自動車などが突発的に動くものになっている。最後のステージでは、画面にあるように、大きな球が上から振ってくるようなゲーム的な要素も持っている。学生自ら積極的にシーンを作成し、一般に出しても通用するようなレベルにまで仕上がった。

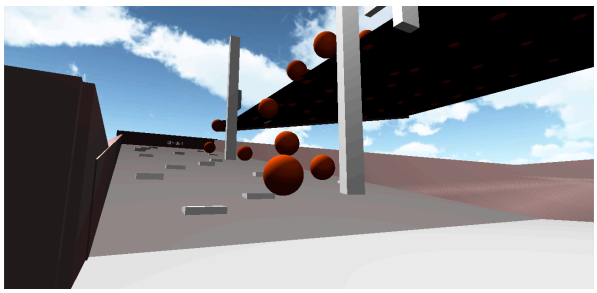

図2　緊急時の行動の最後のステージの試作場面

★RPC を用いて、クライアント・サーバの構成で複数人が参加できるようなゲーム作品

前節で説明した RPC を使って各ユーザのアバターが鬼ごっこをする作品だが、複数の参加者でプレイすることができるように仕上がっている。

図3 鬼ごっこで崖の上に登ったアバター

★チュートリアルとして用意されたレーシングゲームのアセットを用いて、ユーザが走行する車以外の車を道路の形状やユーザの走行に合わせて自動的に走行させるようにした作品

図4　他車が自動運転されるレーシング作品

Unity Technologies 社が用意したチュートリアルでは、ユーザの操作する車だけがサーキットを走行する形になっているが、これに他車も加えて、レーシングゲームにした作品だが、他車を自動的に動かすのに、サーキットの座標などを指標にするのではなく、衝突が起こったら方向を切り替えたり、ユーザの走行車のベクトルなどを参照したりして自動運転をさせた。この作品の制作を通して、一般的な車の自動運転が非常に大変であるということを学生も教員も認識させられた。

ちなみに、このチュートリアルを元にしたような作品が、ディズニー映画の公式サイトの Wreck-it Ralph のゲーム[3]として公開されている。この作品でも、ユーザ以外の車の走行の安定に苦労しているのがわかる。

5.　Panda3D, OSG とのプログラミングの比較

Panda3D と Open Scene Graph（OSG）は、1 つのプログラムの中心でレンダリングループを形成し、そこから派生的に各オブジェクトに対しての操作を呼び出す形になる。各オブジェクトはシーン・グラフの中に置かれれば、自動的にシーンの中でレンダリングされるという形になる。それに対して Unity では、中心となるループは既にランタイム・システム側で用意されており、プログラムはシーンに配置されたオブジェクトにスクリプトのコンポーネントを付属させることによって行なわれる。そのスクリプトにおいて、コールバックされる関数を定義しておき、そこではオブジェクトに対する操作がプログラムの記述の中心になっている。

6.　おわりに

既に 3DCG を一から記述していくプログラミングのパラダイムは終焉に差し掛かっているのかも知れない。Unity のパラダイムは、もちろん、基礎的なプログラミングの能力は必要とするが、それよりも、3 次元のシーンの中にある各オブジェクトの動きやロジックに集中したプログラミングを行なうことができる。また、現実世界と同様に、シーンの中においては、並行プログラミングが基本になっており、同期などもプログラムする必要はあるが、その仕組みがすぐにクライアント・サーバ型の環境に敷衍することが可能になっている。

まずこのような開発環境は、従来のライブラリ型の環境に比べて、3 次元シーンのエディタや様々なコンポーネントの管理をする部分が入ってくるので、開発環境自体の開発に非常に時間が掛かるだろう。しかしながら、Unity のようにでき上がってしまえば、その開発環境を利用して、アプリケーションの開発者は割と手間取らずに RAD（Rapid Application Development）が可能になるだろう。

このような RAD を前提とした環境では、オブジェクト指向型のスクリプト言語（JavaScript や Python、あるいは Lua など）が今後の主流になってくると思われる。C/C++や Java でプログラミングの基本を学んだ学生は、スクリプト言語を用いて、社会に出てからもこのような開発環境で Web/iOS/Android のアプリケーションを作り続けるものと考えられる。

参考文献

(1) Unity 3D, http://www.unity3d.com/unity（2013）.
(2) Blender, http://www.blender.org（2013）.
(3) Sugar Rush, http://disney.go.com/wreck-it-ralph/#/games/sugar-rush (2013).