3Dアニメーション作成ツール

J3Wのすべて

水谷純
mizutani.jun@nifty.ne.jp
http://www.nk.rim.or.jp/~jun/

J3Wとは？

「3Dゲームを作りたい！」と思っている人はいませんか？J3Wシリーズは、まさにそんな人に向けて用意された、3次元アニメーションプログラムを作成するための環境です。ここではその中の1つ、「J3W for Linux Ver.6.42」について解説していきます。また、作成したアプリケーションは、同じシリーズの「J3W for Windows 95 Ver.4.42」を使って、Windows 95/98/NTでも実行可能となっています。

Linuxで3Dグラフィックス表示を伴うプログラムを作成するには、通常、Mesa（OpenGL互換フリーライブラリ）などの3次元ライブラリを使ってC言語で作成することになります。しかし、プログラミング初心者の方は、Cなどのプログラム言語を何とか理解できるようになって、雑誌や書籍を頼りにOpenGLのサンプルプログラムを動かせても、次に「フライトシミュレータ」や「格闘ゲーム」を作ろうとしたときに、どうしたらいいか分からなくなるのが普通ではないでしょうか？例えば、1つの物体を表示して回転させることは、3Dライブラリを使用しても比較的簡単に実現できます。しかし、独立して運動する複数の物体を表示するには、物体ごとに位置、速度、そのほかの状態を保持しておく必要があります。そのため、通常はまずデータ構造の設計に頭を悩ませることになります。

そこでJ3Wの出番です。J3Wを使用して3次元アニメーションのプログラムを作成する場合、3Dライブラリ（例えば、DirectX、OpenGL）を使用する上で必須となる、「三角関数」や「行列」を意識する必要は全くありません。J3Wでは物体ごとの位置、速度、姿勢といった情報は個々の物体自身が知っています。「1秒間で90度右を向く」、「瞬間的に30度上を向く」、「30秒間で100メートル前進する」といった物体の動きと速さを、Java風の文法を持った3次元アニメーション専用のオブジェクト指向言語「j3c」でプログラムとして作成します。j3cは3次元アニメーション専用言語のため、表示の方法や表示速度を気にすることなく、常に指定した時間で動作します。高速なコンピュータでは滑らかに、遅い環境でもそれなりに表示されるのです。3次元アニメーションというと速いマシンが必要と思われがちですが、J3WはXlibのみで作成されているため、100MHz版Pentium程度でも十分動かしたり遊んだりできます。

また動作の高速化のため、テクスチャマッピング、バンプマッピングなどリアルな3D CGを作成するために必要な機能はサポートしていませんが、リアルタイムに視点、光源、物体を自由に動かせる3次元アニメーションを作成できます。光源の種類を変更したり、スペキュラ強度を変更したりすることももちろん可能です。3Dプログラミングの入門や本格的な3Dアプリケーションを作成する前に、簡単にテストしてみる「3Dプロトタイピングツール」には有効でしょう。慣性速度、角速度、重力加速度を物体ごとに設定することが可能なので物理シミュレーションも作成できます。

そして、J3W付属の入門用ドキュメント中のサンプルを改造して試している間に、オリジナルのプログラムのアイデアが生まれてくるはずです。プログラミングは、書籍を読むより実際に試してみるほうが、ずっと簡単に習得できます。

「プログラムなんてしたことはないが、3次元アニメーションには興味がある」という人も、J3Wでオブジェクト指向プログラミングとアセンブラによるプログラミング、さらに3次元

アニメーションを同時にマスターできるかもしれません。オブジェクトが「すべて目に見える物体」となっているため、オブジェクト指向言語と聞いて尻込みしてしまう人でも、抽象的なオブジェクトで悩むことはありません。

またLinux版J3Wのライセンスは、Free Software FoundationのGPLを採用しています。J3W自身の使用や改造、また改造したものを配布することは、そのソースを公開する限り自由に行なって構いません。ライセンスの詳細はパッケージに含まれていますので、詳しくはそちらを参照してください。

実行例1　j3w-642ディレクトリの内容

```
$ ls
COPYING     README.euc    history.txt   j3     j3cc   j3opt.pl     sample.txt
Makefile    gpl.text      html   j3c_script   j3d    j3w_script   source
```

J3W の構成

全体的な流れは、図1で示すように通常のプログラミングと変わりません。コンパイルは「j3c」、アセンブルは「j3dasm」、そして実行は「j3w」という流れです。

コマンドとしてのj3wは、仮想CPUのエミュレータとなっています。この仮想CPUは3次元空間で形を持つことができ、さらに自己増殖、自己消滅できる特殊なCPUとなっています。そのCPUの命令をj3dasmというアセンブラが生成し、そのアセンブラ用のコンパイラとしてj3cがあります。

「3次元空間に存在する物体は、それぞれコンピュータにコントロールされている。その空間を表示するコマンドがj3wである。それらのコンピュータの動作を指定するためにJ3C言語を使う」ということです。図1中のABC.j3?はファイル名で拡張子の「?」の部分が異なっていることに注意してください(ファイル名ABCは単に例として挙げているだけです)。下の例で最終的に生成されたABC.j3dは、32bitの16進数が並んだテキストファイルとなります。このABC.j3dを j3wで実行することにより、3次元アニメーションが実行されます。実行時にはデータである「???.j3d」と「j3wコマンド」以外は必要ありません。また用語の使い方としては、「J3W」がJ3Wシステム全体を指し、「j3w」はj3wコマンドを示します。「J3C」はJ3C言語を意味し、「j3c」はコンパイラのコマンド名を表わしていることにします。

インストール

J3W をインストールするには、ソースをコンパイルする必要があります。基本的にはmake、そして「make install」と2つのコマンドを実行するだけと、非常に簡単ですから、自分でコンパイルしたことのない人もぜひ挑戦してみましょう。意外と手軽に行なえることがすぐに分かるはずです。どのLinuxディストリビューションでもインストール作業は同じですが、手元にインストール直後の状態の「LASER5 Linux 6.0 Rel2.0」があったので、実際にインストールした手順に沿って説明します。ディストリビューションが違っていても、シェルプロンプトの表示が異なるだけですので、あまり気にする必要はありません。またここではユーザー「jun」のホームディレクトリ/home/jun/で作業を行なっていることにしますが、どのディレクトリで作業しても構いません。

① まず、「j3w642.tar.gz」というファイルを解凍(展開)します。

```
$ tar zxf j3w642.tar.gz
```

② 解凍されると「j3w-642」というディレクトリができますから、そこへ移動しましょう。

```
$ cd j3w-642/
```

③ ディレクトリ内のパッケージ内容を一応確認してみます。lsコマンドを打ち込むと、実行例1のように表示されます。

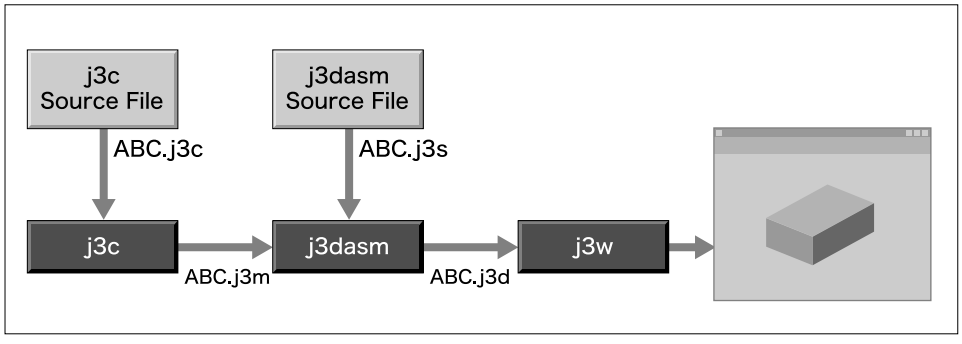


図1
J3Wのプログラミングの流れ

④「README.euc」というドキュメントにインストール方法と簡単な使用法が書いてあります。htmlディレクトリ内にはさらに詳細なドキュメントがあります。「index.html」がJ3Wの紹介と総目次になっていますので、必ず読んでください。

では、コンパイルといきましょう。これはmakeコマンド一発でOKです。

```
$ make
```

コンパイルは数分以内に終了するはずです。次の操作は特に実行する必要はありませんが、バイナリ(コマンド)のサイズが小さくなります。

```
$ make strip
```

インストール先の/usr/local/binディレクトリは、一般ユーザーに対して書き込み権限を許可していませんから、インストールするにはスーパーユーザー(root)権限が必要になります。suコマンドでスーパーユーザーに変身しましょう。

```
$ su
Password:(パスワードを入力する)
```

ここでパスワードを入れてリターンキーを押すとOKです。これでスーパーユーザーになれたので、さっそくインストールのためのコマンドを実行することにします。

```
# make install
```

これで/usr/local/bin/に「j3cc」、「j3」、「j3c」、「j3dasm」、「j3w」、「j3opt.pl」の6つのファイルがインストールされます。もしJ3Wが不要になった場合には、これらの6つのファイルを削除するだけでアンインストールが完了します。Red Hat系のディストリビューションであっても、/usr/local/bin以下に自分でインストールしたファイルはRPMの管理外なので、勝手にコンパイルしてインストールしても問題ないでしょう。

筆者が利用している、LASER5 Linuxの場合にはインストール作業はこれで終了です。実際に使ってみるために、再び元のユーザーに戻ります。

```
# exit
```

解凍したj3w-642ディレクトリには、サンプルのソースやJ3C用のライブラリ、J3Wに関する解説などJ3Wを使う上で有用な情報が含まれています。J3Wが不要になるまで残しておいたほうがいいでしょう。しかしJ3Wのインストール作業の中間結果などは不要ですから、次のコマンドを実行しておくと少しディスクが節約できます。

```
$ make clean
```

ディストリビューションによっては、コマンドの検索パスに/usr/local/binが含まれていない場合があります。コマンドの検索パスは「echo $PATH」で確認できます。

```
$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/
home/jun/bin
```

LASER5 Linuxでは/usr/local/binが含まれているのですぐに実行可能です。もし/usr/local/binが含まれていない場合は、

```
$ export PATH=/usr/local/bin:$PATH
```

を実行しましょう。あるいは、毎回検索パスを指定するのが面倒な場合は~/.bashrcなどに

```
export PATH=/usr/local/bin:$PATH
```

を追加してください。

サンプルの実行

オリジナル3Dアニメーションを作成するためには、自分でプログラムを行なう必要がありますが、その前にまずJ3Wで何ができるかサンプルで確認してみましょう。j3w-642/j3dディレクトリには、すぐに実行できるサンプルが多く用意されています。ちょっとのぞいてみましょう(実行例2)。

拡張子がj3dのファイルが48個あります。ここでもしXサーバ(以下X)を起動していない場合は、以下のような感じで、なるべく多くの色が使えるように指定して起動しておきます。

```
$ startx -- -bpp 16
```

この理由は、動作しているXの色数で表示品質が決まるためです。8bpp(256色)の画面モードでも実行できますが、J3Wのウィンドウをアクティブにするとほかのウィンドウの色が変わります。また8bppではスペキュラが無効になってしまいます。

実行するにはj3wコマンドを使用しますが、サンプルの*.j3dでは日本語を使用しています。「kterm」または「rxvt」などの日本語が表示できるターミナルウィンドウから実行してください。「xterm」や「Gnome」のターミナルでは出力される文字が化けます。ktermが開いていない場合はターミナルウィン

```
$ cd j3d
$ ls
FRAME_1.DAT flight.j3d   list12.j3d   list9.j3d    list_4.j3d   pose.j3d
FRAME_2.DAT hand.j3d     list13.j3d   list_10.j3d  list_5.j3d   random.j3d
FRAME_3.DAT hsort.j3d    list2.j3d    list_11.j3d  list_6.j3d   taco.j3d
ball.j3d    human2.j3d   list3.j3d    list_12.j3d  list_7.j3d
testcalc.j3d
bcolor.j3d  line.j3d     list4.j3d    list_13.j3d  list_8.j3d   tubo.j3d
body.j3d    line2.j3d    list5.j3d    list_14.j3d  list_9.j3d   while.j3d
color.j3d   list1.j3d    list6.j3d    list_15.j3d  manual2.j3d
cube.j3d    list10.j3d   list7.j3d    list_2.j3d   multiprc.j3d
falcon.j3d  list11.j3d   list8.j3d    list_3.j3d   note.j3d
```

実行例2
j3w-642/j3dディレクトリ内のサンプルファイル

ドウから次のように入力してktermのウィンドウを開きます。

```
$ kterm &
```

ktermのウィンドウでフライトシミュレータを実行してみましょう。

```
$ j3w flight.j3d
```

実行が始まると、ktermウィンドウに簡単な使用法が表示され、新たに開いたウィンドウでフライトシミュレータが表示されます(画面1)。最初はオートパイロットモードとなっていて、MキーとESCキー以外の入力を受け付けない状態になっています。また視点は空中に固定された位置にあり、飛行速度は意識的に遅くしてあります。Mキーを押してマニュアルモードに変更するとキー操作できるようになります(画面2)。

Y、U、H、G、J、N、Tキーでカメラの位置が変わり、Fキーで位置固定の視点に切り替わります。カーソルキーとK、Lキーで方向転換、Sキーがキャノン発射で、弾が飛行船に命中するとスコアが上がり、標的の飛行船が消滅します。終了するには ESCキーを押してください(画面2)。詳細は「flight.j3c」というファイルを参照してください。

グラフィックが表示されているウィンドウをXボタンなどで強制的に閉じたような場合や、ターミナルウィンドウでCtrl+Cキーで強制終了した場合、まれにターミナルで文字入力を受け付けないような状態になる場合があります。このときはEnterキーを押したあとに、画面上で見えなくても

```
j3w(このあとEnterキー)
```

と入力して、引数なしでj3wを起動してください。

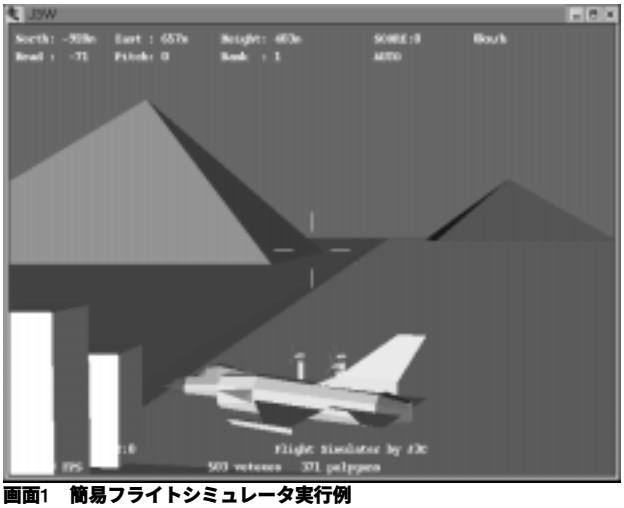
```
J3W ver.6.42 2000 Jun Mizutani
usage : j3w file
```

と表示した後に正常な状態に戻ります。

では、さらにもうひとつサンプルを実行してみましょう。

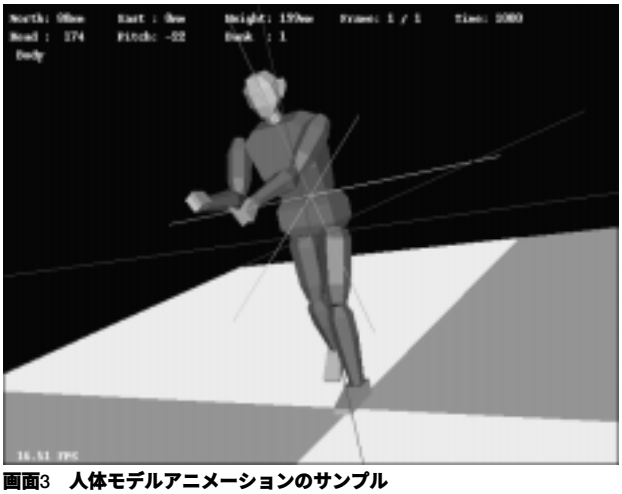
```
$ j3w pose.j3d
```

ポーズエディタが起動します。「pose.j3d」は人体モデルのアニメーションのサンプルです(画面3)。人体モデルのポーズ



画面1　簡易フライトシミュレータ実行例

画面2　マニュアルモードでの操縦キー設定



画面3　人体モデルアニメーションのサンプル

を作成してアニメーションとして再生させ、ファイルに保存できます(画面4)。pose.j3dを起動して2を押したあと、Mを押してください。サンプルのアニメーションが起動します。どうですか？　これで、格闘ゲームを作ってみたくありませんか？　J3Wで必要な知識を身につければ、それは夢ではありません。pose.j3dはESCキーで終了します。使用法の詳細はpose_man.htmlを参照してみましょう。

以上のサンプルを、450MHz版Celeron、RivaTNT、XFree86 4.0といった組み合わせの環境で実行すると、1秒間に150フレーム以上の表示速度になります。J3Wはインタプリタ系の環境ですが、速度の心配はまったく不要というわけです。

画面4
アニメーション設定・操作画面

j3cのプログラムの作成と実行

ここからはもう少し突っ込んだ話をすることにして、「prog.j3c」というソースファイルをコンパイルして実行するまでを解説していきましょう。

Java風のプログラミング言語、J3Cで書かれたプログラム(prog.j3cとします)を、まずコンパイラであるj3cコマンドでコンパイルします。

```
$ j3c prog.j3c
```

これで「prog.j3m」というファイルができます。これは仮想CPU用のアセンブリ言語のファイルなのでj3dasmというアセンブラでアセンブルします。

```
$ j3dasm prog.j3m
```

これでj3w コマンドから実行可能な「prog.j3d」が生成されます。あとは

```
$ j3w prog.j3d
```

で実行できます。しかし、J3C言語のプログラムを修正するたびにコンパイル、アセンブル、実行では面倒です。そこでj3ccというシェルプログラムが用意されています。コンパイルとアセンブルを同時に行ないたい場合は次のように指定します。

```
$ j3cc prog.j3c
```

これでprog.j3mとprog.j3dが生成されます。さらに、コンパイルとアセンブル、j3wの実行まで行ないたい場合は

```
$ j3cc -r prog.j3c
```

とします。すると、j3w prog.j3dの実行まで自動的に行なわれ、j3cのソースを直接実行できるような環境となります。j3cのソースに構文エラーがある場合、またj3dasmのアセンブリソースにエラーがある場合は、エラーを見つけた段階でエラーを表示してj3ccスクリプトは終了します。

Perlが使用できる環境ならばj3opt.plでj3cの出力ファイル*.j3mを最適化(10%〜15%)して、「*.j3o」に変換することができます。このj3oという拡張子のファイルは、j3mやj3sと同じくj3dasmの入力に使用できます。j3opt.plはPerlスクリプトです。/usr/bin/perlディレクトリ以外の場所にPerlがインストールされている環境では、j3opt.plの1行目を変更してください。またj3dファイルも小さくすることができます

が、この場合は、実行速度にはほとんど影響はありません。あまり挑戦する必要はないでしょう。

ちなみに、j3のほうは「Tcl/Tk」で実験的に作成したj3dasmとj3wのGUIフロントエンドとなっています。

最小のプログラム

ここでは、実際にJ3C言語でプログラムを作成してみましょう。条件分岐や代入文などの文法は既存のプログラム言語とよく似たものになっているため、C、Pascal、Java、Visual Basicなどでプログラミングの経験を持っている人ならば習得は容易です。J3Cは、それらの中でも、文法的にはJavaに最も近いものになっています。J3Cが3Dアニメーション専用となっているのは、複数のクラスが目に見える物体として同時に動作できる点と、3Dアニメーション専用の組み込み関数が多く用意されている点です。

以下に示す例はJ3Cの最小のプログラムで、実行しても何もしない無限ループとなっています。終了させるためにはCtrl+Cキーなどで強制的に終了させる必要があります。あえて実行する必要もありませんが、この形を覚えておいてください(リスト1)。

このリスト1で示されるように、J3Cのプログラムではmainクラスが必須であり、各クラスにはINIT、RUN、EVENTの3つのメソッドが必須となります。J3Cではクラスが実行の単位で、物体の1つ1つがクラス内で定義されたように動作します。ここで定義された関数をメソッドと呼びます。

任意の名前のクラス、メソッドを宣言できますが、プログラムには1つのmainクラスが必要で、各クラスは必ずINIT、RUN、EVENTメソッドが定義されている必要があります。mainクラスとINIT、RUN、EVENTメソッドは特別な意味を持つと覚えておいてください。クラスは継承することができるため、基底クラスで宣言されていれば、すべてのクラス宣言でINIT、RUN、EVENTの3つのメソッドを記述する必要はありません。今回はページ数が限られていますため、ここではこれ以上説明は行ないません。詳細はパッケージ中のj3c_spec.htmlを参考にしてください。

J3Cでは、mainクラスは最初に自動的にインスタンスとして生成され、INITメソッドが最初に実行されます。INITメソッドが終了すると次にRUNメソッドが繰り返し実行され、ほかのクラスインスタンスからのメッセージを受信した場合にかぎってEVENTメソッドが実行されます。上記の例では、「INIT()、RUN()、RUN()、RUN()……」というように強制終了されるまで実行します。

通常のプログラムでは、mainクラスのINIT、RUNメソッド、またはそれらから呼ばれたメソッドが、別のクラスのインスタンスを生成し、生成されたインスタンスは独立して自分のクラス宣言の定義にしたがって「INIT()、RUN()、RUN()、RUN()……」などを実行します。J3Cのプログラムの実行中は複数のインスタンスがRUNを実行し続けます。メッセージを受信した場合だけEVENTを実行して、その後RUNを繰り返し実行します。

リスト1　J3Cの最小のプログラム例

```
class main {          // クラス宣言
    int INIT() {}     // メソッド宣言
    int RUN() {}      // メソッド宣言
    int EVENT() {}    // メソッド宣言
}
```

Hello World

次のプログラムは、「Hello World」という文字列をターミナルウィンドウに表示するだけのプログラムです。グラフィックウィンドウは開きません(リスト2)。

「String」は文字列表示用の組み込み関数です。Char(13)は改行文字を出力します(Linux版では13[CR]は10[LF]に内部で変換されます)。Stringで「Hello World」を表示し、Char(13)で改行してStopで終了します。

/*と*/に囲まれた部分と//から行末まではコメントなので、入力する必要はありません。短いプログラムなのでお気に入りのエディタで実際に入力して、「list_2.j3c」というファイル名で保存してください。直接実行するために次のように入力します。

```
$ J3cc -r list_2.j3c
```

もしなんらかのエラーで中断する場合は、入力したプログラムとリスト2を何度もよく見比べてみましょう。実は、「俺は間違っているかもしれない」と悟るのが、プログラミング上

リスト2　「Hello World」プログラム例

```
class main {
    int INIT() {
        String("Hello World");
        Char(13);
        Stop();
    }
    int RUN() {}
    int EVENT() {}
}
```

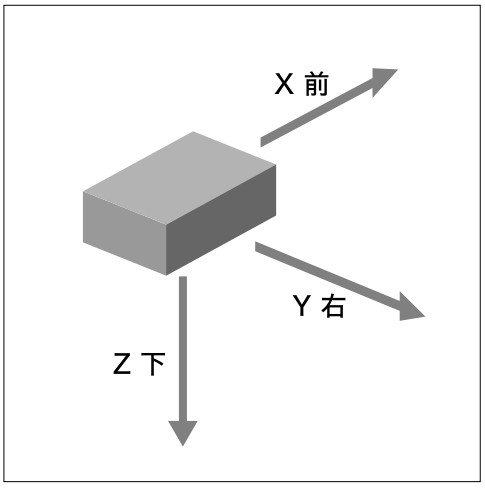


図2
J3Wの座標系

達のコツなのです。

リスト2中のStop()は、すべてのインスタンスを終了させる組み込み関数です。インスタンスとは、クラスの定義に基づいて実際に動作している部分です。J3Cでは、多くの仮想CPUが、複数のクラスからなる1つのJ3Cプログラムを同時に実行します。プロセスやスレッドという概念に近いものと考えてもらうといいでしょう。J3Cでは、すべてのインスタンスが終了した時点でプログラムが終了します。この場合は1つのインスタンスしか存在しないので、自分を終了させるdelete()組込み関数でもプログラムは終了します。しかし通常は、ほかのインスタンスの数も寿命も分からないため、プログラムを終了させる場合には、プログラムを終了させる責任のある(例えばキー入力を処理する)インスタンスがStop()を実行するようにしたほうが良いでしょう。

例えばこの例では、Stop()がないと、INIT実行後にRUNが繰り返し実行されます。RUNの中にStop()を置くこともできます。INIT実行後、最初のRUNの実行時にプログラムは終了します。

リスト3　物体の表示・動作プログラム例

```
class Triangle {
    final int color1 = 6;        // 面の色指定
    final int color2 =14;
    int INIT() {
        NewObject(6, 2);         // オブジェクトの生成　6頂点 2面
        DefPoint(0, 0,   500);   // 頂点0 三角形の頂点座標を登録
        DefPoint(0, 500,-500);   // 頂点1
        DefPoint(0,-500,-500);   // 頂点2
        DefPoint(0, 0,   500);   // 頂点3 頂点0と同じ
        DefPoint(0, 500,-500);   // 頂点4 頂点1と同じ
        DefPoint(0,-500,-500);   // 頂点5 頂点2と同じ
        DefPlane(color1, 3, 0, 1, 2); //面の定義 表から見て左回りに指定
        DefPlane(color2, 3, 5, 4, 3);
        ClearRegisters();        // RX - RBに0を代入
        SetPosition();           // 初期位置と角度の設定
     }
     int RUN() {
        RotHead(6000, 2880);     // 6秒間で360度回転
        DeleteObject();          // オブジェクト
        Stop();                  // 全インスタンス終了
     }
     int EVENT() {}              // メソッド宣言
}

class main {
    int INIT() {
        new(Triangle, 50);
        NewObject(0, 0);          // 視点用オブジェクトの生成
        ClearRegisters();         // RX - RBに0を代入
        RX = -2000;               // 位置の指定 20メートル後方
        SetPosition();            // 位置と角度を視点に設定
        See();                    // このオブジェクトが見る
        BackgroundColor(0x000);   // 背景色を黒に設定
        GraphMode();              // グラフィックウィンドウを開く
     }
     int RUN() {
        Wait();
     int EVENT() {}               // メソッド宣言
    }
}
```

物体の表示

では、いよいよ3次元空間で物体を動かしてみましょう(リスト3)。3次元の世界で物体を表示する場合は、空間内に「物体」と「その物体を見ている物体(視点)」が存在する必要があります。また、「1つのインスタンス」は「1つの物体」になることができます。従って、ここでは最低でも2つのインスタンスが必要になってくるわけです。さらに、視点と物体では動作が異なるため、クラスも2つ必要になります。以下の例では、自動的に生成されるmainクラスのインスタンスが空の(形のない)物体を生成し、位置をSetPosition組み込み関数で設定して、See組み込み関数で視点を取得します。このインスタンスが「目」となり、2番目のインスタンス(Triangle)は三角形のオブジェクトとなります。

次に、位置を指定するために座標を指定します。すべての物体はそれぞれ自分の向いている方向を基準として、前がX、右がY、下がZの増加する方向です。これを座標系と呼びます。回転はZ軸回りの回転(右を向く)をヘッド回転、Y軸回りの回転(上を向く)をピッチ回転、X軸回りの回転(首を右に傾ける)をバンク回転と呼びます(図2)。

J3Wの中には、1つだけ常に存在する固定された特別な座標系があり、すべての物体の座標系の基準となっています。北を向いて立ち、北がX、東がY、下がZとなるわけです。これを「ワールド座標系」と呼び、クラスごとに重力を設定できますが、重力は物体の姿勢に関係なくワールド座標系の下方向(Zが正の方向)に働きます。

ここで使用している座標系は航空力学で使用されるNED系(X:North[前]、Y:East[右]、Z:Down[下])で、3次元グラフィックでよく使われる座標系とは異なりますが、地上の建造物のように広がりのある空間では高さ情報を最後に持つ座標系のほうが読みやすいと思います。座標値は符号つき32bit整数の範囲(±20億)が可能です。とりあえず、最初のうちはあまり難しく考えず、「前がX、右がY、下がZ」とだけ覚えておくといいでしょう。

クラスmainでは、自動的に実行されるINITメソッドの冒頭で、クラスTriangleのインスタンスをnew()組み込み関数を使って生成しています。これでクラスTriangleのインスタンスは実行を始めます。2つのインスタンスが同時に動き始めますが、文章では視点となるmainクラス(のインスタンス)の動作から説明します。

視点には頂点も面も必要ないので、頂点と面の数を0として「見えない物体」を生成しています。ただし、表示されなくてもSetPositionで設定された位置と角度の情報を持ちます。また、引数をレジスタ変数(RX、RY、RZ、RH、RP、RB)で渡す必要のある組み込み関数がいくつもあります。レジスタ変数とは仮想CPUのレジスタを直接扱う変数です。組み込み関数は仮想CPUの命令に1対1に対応しているため、レジスタを直接操作する変数があるほうが都合が良かったので採用しています。

SetPositionには、位置座標(RX,RY,RZ)と角度(RH,RP,RB)をレジスタ変数に設定して呼び出すことで、視点の位置と角度を指定します。ClearRegisters()はRX、RY、RZ、RH、RP、RBをすべて0にする関数です。

See()を実行したインスタンスから見た光景がウィンドウに表示されます。実際に表示するためにはGraphMode組み込み関数を実行して、グラフィックウィンドウを開く必要があります。この組み込み関数は、どのインスタンスが実行しても構いませんが、複数のウィンドウを開くことはできません。

INITメソッドの仕事は以上です。この例では視点は移動しないためRUNメソッドでは終了イベントを待っているだけです。

クラスTriangleのインスタンスは、INITメソッド中でNewObject()組み込み関数を使って、Triangleに形状を生成するための用意をします。NewObject()の引数には最大頂点数と最大面数を与えます。頂点数、面数は実際に使用する数より多くても問題ありませんが、少ないとエラーが発生します。形状の定義はDefPoint組み込み関数でX、Y、Z座標を指定して1つずつ頂点を登録します。頂点番号はオブジェクトごとに0から登録順に付けられます(VertexRelative組み込み関数で変更可能)。

そして、DefPlane組み込み関数で色、頂点数、頂点番号列を指定することで面を定義します。面には表と裏があり、DefPlane組み込み関数1つでは、裏からは不可視になります。ここでは頂点を6個登録して、三角形の裏表の面を作成しています。

1つの平面の表と裏を表示するために、頂点を表と裏の面で共有して面を設定することができますが、なるべく1つの平面の表と裏の頂点を共有させないでください。リスト3では同じ頂点を2回ずつ登録しています。1平面の表と裏ではなく、例えば立方体の頂点のような場合には、頂点の共有は全く問題ありません。しかし、1つの平面の表と裏で頂点を共有すると、頂点ごとに法線ベクトルを与える必要がある環境では問題となる可能性があるためです。すでにWindowsのDirectX版(obsolete)で問題になりました。Linuxでも、今後Mesa(OpenGL)版を作成するときのための予防線です;-)。

三角形の物体の運動はRotHead(6000, 2880)で指定しています。この場合6秒間で360度ヘッド回転(Z軸回り)して終了します。回転速度を変更して実際に試してください。さらに、Up、Forwardなどの組み込み関数を順次実行することで、いろいろな動きを与えることができます。

終わりに

今回は、J3Wの特徴、インストール、サンプルの実行方法とJ3C言語のプログラミング例について簡単に解説しました。J3W パッケージには1Mbytesほどのサンプルソースとドキュメントが付属しています。プログラミングの経験者ならば文書よりプログラムソースを読んだほうが理解が早いかもしれません。

J3WのWebページ、また添付ドキュメントでは、J3C言語に関してもう少し詳しく解説しています。どんどん勉強して、3Dアニメーション作品や3Dゲームの制作を目指しましょう!