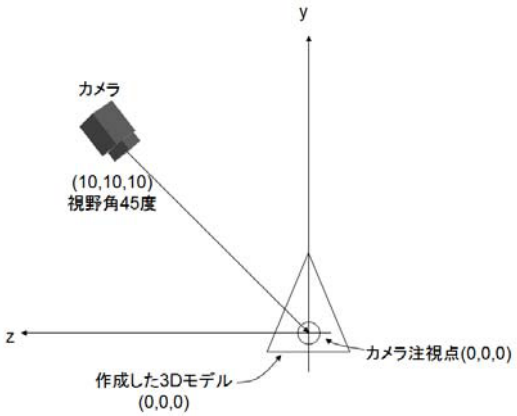
3DCGレンダリング(座標変換) †

ローカル座標系を定義し、モデルの向き・位置を調整できるようにする。

カメラ・物体の配置 †

以下のような配置にすること、ただし、モデルが画面内に収まらない場合は、カメラ位置を調節すること。



ライティング設定 †

前回は、ライティング設定を自前で行っていたが、XNAにはデフォルトのライト設定を行ってくれる、EnableDefaultLightingメソッドが用意されている。

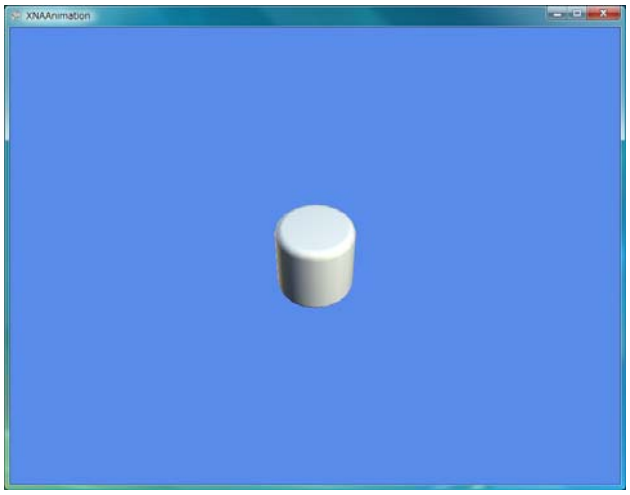
コードは以下のようになる。

```
private void DrawModel(Model m,
 Matrix world, Matrix view, Matrix projection, GameTime gameTime)
{
    Matrix[] trasnforms = new Matrix[m.Bones.Count];
    m.CopyAbsoluteBoneTransformsTo(trasnforms);

    foreach (ModelMesh mesh in m.Meshes)
    {
        foreach (BasicEffect effect in mesh.Effects)
        {
            effect.EnableDefaultLighting(); //追加

            effect.View = view;
```

EnableDefaultLightingメソッドで行ってくれるライト設定は、映像制作におけるライティングの基礎に基づいたものである。
詳細については、以下を参照すること。

- 『BasicEffectとは』 ひにけにXNA
- http://blogs.msdn.com/ito/archive/2007/03/19/basiceffect.aspx

EnableDefaultLightingメソッドを使用すると、例として以下のようなライティングとなる。



グリッドの表示 [†]

物体を動かす際、空間に基準となる表示がないと、物体の動きを把握しづらい。
そこで、別途アドレスに置かれているクラス(Grid3D.cs)を使用することで、空間にMetasequoiaと同等のグリッドを表示することができる。
(右クリックで[名前を付けて保存]、現在のプロジェクトへ追加すること)

Grid3D.csをプロジェクトに追加後、以下のように追記する。

```
using Microsoft.Xna.Framework.Storage;
using XNAHelper;//追加
```

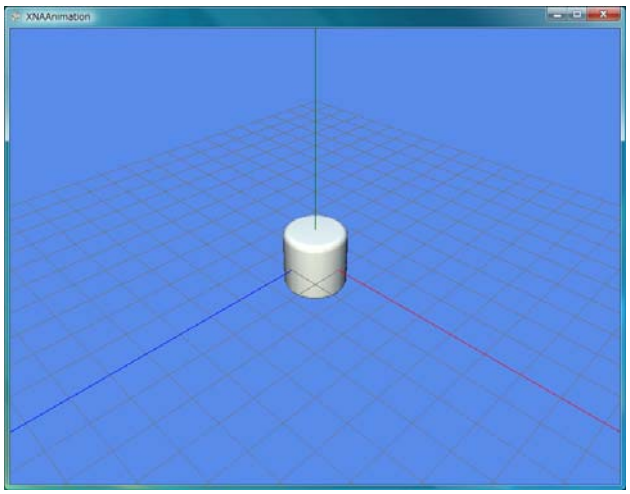
```
SpriteBatch spriteBatch;
Grid3D grid;//追加
```

```
public Game1()
{
    graphics = new GraphicsDeviceManager(this);
    Content.RootDirectory = "Content";

    grid = new Grid3D(this, 10.0f, 10.0f, 10.0f, 10, 10);//追加
    Components.Add(grid);//追加
```

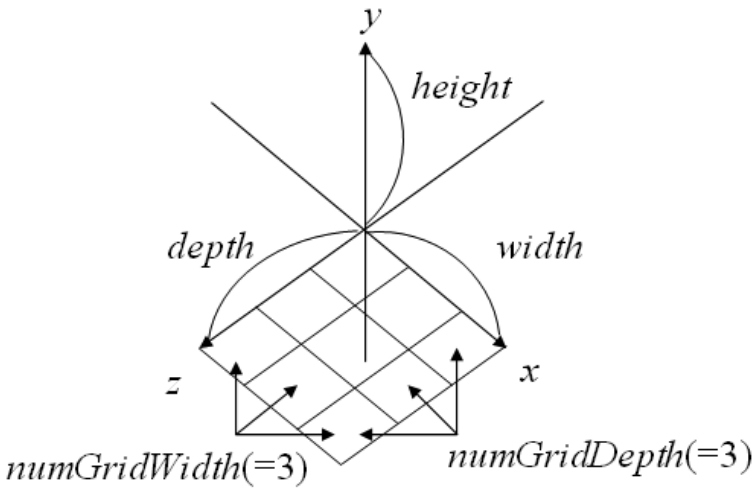
Drawメソッド内

```
view = Matrix.CreateLookAt(cameraPosition, Vector3.Zero, Vector3.Up);
grid.UpdateMatrices(world, view, projection);//追加
```

グリッドが表示されると、以下のようになる。



Grid3Dコンストラクタの各引数の意味を以下に示す。



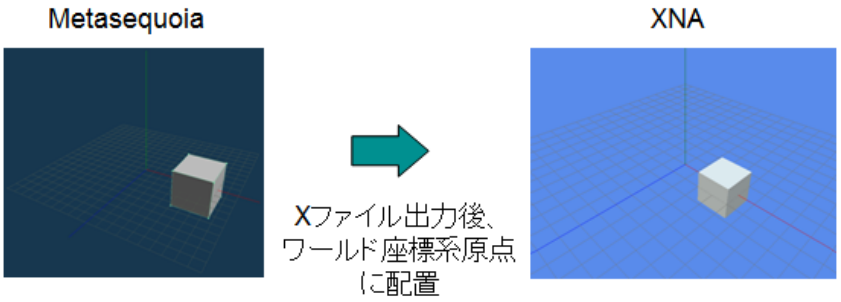
Grid3Dクラスの詳細については省略する。
興味のある者は、コードと共に、以下URLなどを参照すること。

- 方法：ポイントやラインなどの 3D プリミティブの描画
- http://msdn.microsoft.com/ja-jp/library/bb196414.aspx

モデル座標系 [†]

Metasequoia上で作成したモデルを、XNAワールド座標系における任意の位置**P**に配置する、ということは、
モデル座標系の原点を**P**にするという意味になる。
今回、モデル座標系の原点は、Metasequoia上での原点と対応する。

そのため、Metasequoia上で、原点からずれた位置に配置されたモデルをXファイルに出力、
XNA上のワールド座標系原点に配置しても、モデルは原点から離れた位置に配置されることになる。



今後、モデルをアニメーションさせる際、混乱の元となるので、
各自、Metasequoia上で、作成したモデルが原点中心となるように、位置を修正すること。

ローカル座標系 [†]

ワールド座標系内で、モデルの位置・向きを指定するには、ローカル座標系を定義する必要がある。
ローカル座標系は、実際には4x4行列により定義される。

まずは、ローカル座標系を定義するためのMatrixを用意する。

```
Model model;
Matrix mLocal; //追加
```

Updateメソッド内で、先ほど用意したMatrixに、まずは単位行列を代入する。

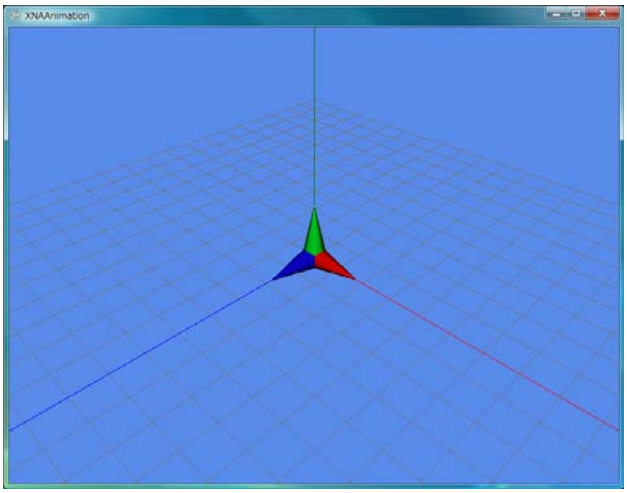
```
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
```

```
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back
          == ButtonState.Pressed)
        this.Exit();

    // TODO: Add your update logic here
    mLocal = Matrix.Identity;//追加

   base.Update(gameTime);
}
```

Drawメソッド内を以下のように変更する。

```
world = Matrix.Identity;
projection = Matrix.CreatePerspectiveFieldOfView(
               MathHelper.ToRadians(45.0f), aspectRatio, 1.0f, 1000.0f);
view = Matrix.CreateLookAt(cameraPosition, Vector3.Zero, Vector3.Up);

grid.UpdateMatrices(world, view, projection);

world = mLocal * world;//変更点

DrawModel(model, world, view, projection, gameTime);
```

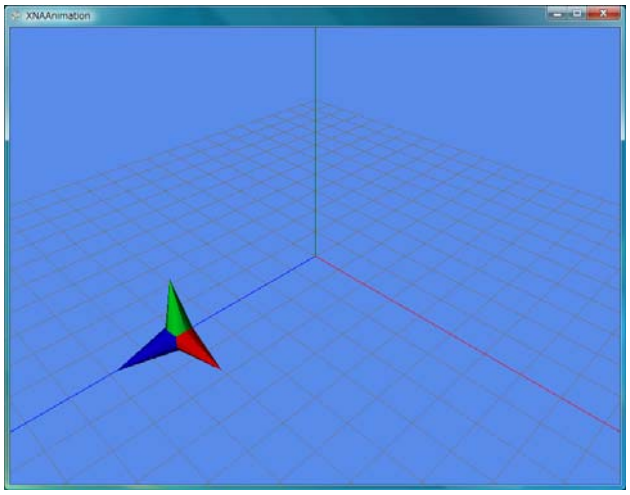
この時点で、モデルの表示位置に変化はない。



次に、Updateメソッド内で、以下のように行列の内容を設定する。

```
Vector3 vZ = Vector3.UnitZ;
Vector3 vY = Vector3.UnitY;
Vector3 vX = Vector3.UnitX;
Vector3 vP = new Vector3(0, 0, 5.0f);

mLocal.M11 = vX.X; mLocal.M12 = vX.Y; mLocal.M13 = vX.Z; mLocal.M14 = 0.0f;
mLocal.M21 = vY.X; mLocal.M22 = vY.Y; mLocal.M23 = vY.Z; mLocal.M24 = 0.0f;
mLocal.M31 = vZ.X; mLocal.M32 = vZ.Y; mLocal.M33 = vZ.Z; mLocal.M34 = 0.0f;
mLocal.M41 = vP.X; mLocal.M42 = vP.Y; mLocal.M43 = vP.Z; mLocal.M44 = 1.0f;
```

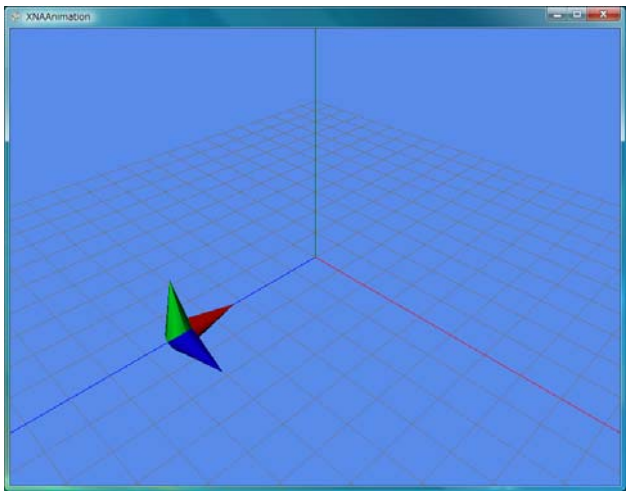
行列の4行3列にvP.z(=5.0f)が代入されていることに注意する。
実行すると、以下のように、モデルが(0, 0, 5)の位置に表示される。



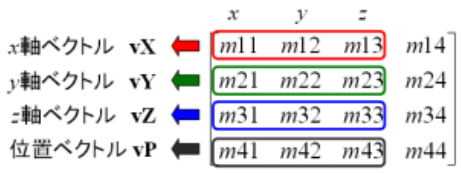
次に、以下のように書き換え、実行する。

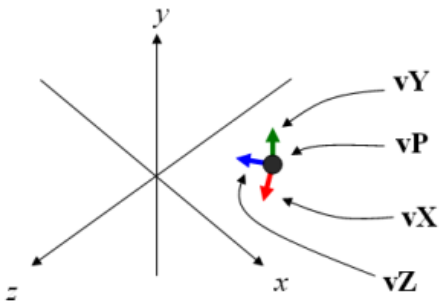
```
Vector3 vZ = new Vector3(1.0f, 0, 0);
Vector3 vY = Vector3.UnitY;
Vector3 vX = new Vector3(0, 0, -1.0f);
```

以下のような結果になる。



以上のことから、行列に指定した値は、以下のような意味を持つことが分かる。





ローカル座標系の各軸の向き、位置を、ワールド座標系での値で指定することになる。
ここで、**vX**、**vY**、**vZ**は正規化されたベクトル(長さ=1のベクトル)であり、
それぞれ直交している必要があることに注意すること。
また当然ながら、右手座標系である必要もある。

また、**vZ**、**vY**が決まれば、**vX**は外積から算出できるので、
ローカル座標系を定義する行列を作成する場合、通常、**vZ**、**vY**、**vP**を指定する。
XNAには、これと同等のメソッド、Matrix.CreateWorldメソッドが用意されている。
引数の、positionが**vP**、forwardが**vZ**、upが**vY**に相当する。

引数の名前から分かるとおり、慣例的、分かりやすさの点から、モデルの正面(モデル座標系のZ方向)をZ軸方向(**vZ**)、上方向をY軸方向(**vY**)とする事が多い。

ここで述べた、変換行列の話題に関して、深く知りたい者は、
以下の書籍などを参考に学習(復習)すること。

- プログラミングのための線形代数
- http://www.amazon.co.jp/dp/4274065782/

課題1 [+]

現在、Matrix.CreateWorldメソッドは現在(2009/06/12)、バグにより正しい結果を得られない。
そこで、Matrix.CreateWorldと同等のメソッドを実装せよ。

Matrix.CreateWorldは、前述の通り、モデルの正面(モデル座標系のZ方向)が引数forwardベクトルの方向を向き、引数positionベクトルにモデルを配置するためのローカル座標系を定義する行列を返すメソッドである。

実装する際は、以下の条件を満たす必要があるとする。

- 引数forward、upに正規化されていないベクトルが指定されても正しく動作する
- 引数forward、upベクトルが直交していなくても正しく動作する

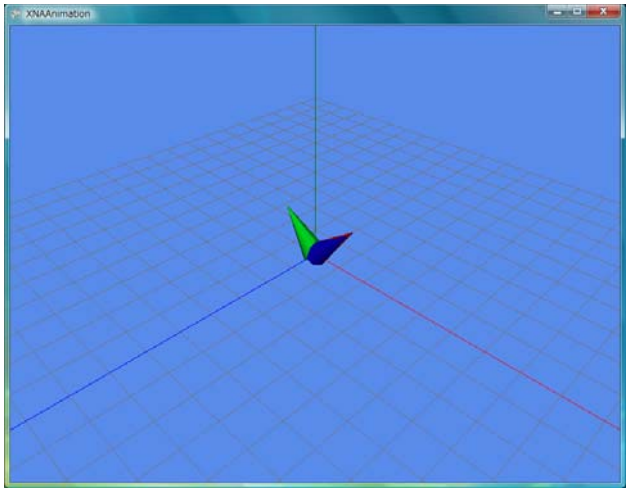
実装するCreateWorldメソッドの雛形を以下に示す。

```
private Matrix CreateWorldMatrix(Vector3 position, Vector3 forward, Vector3 up)
{

}
```

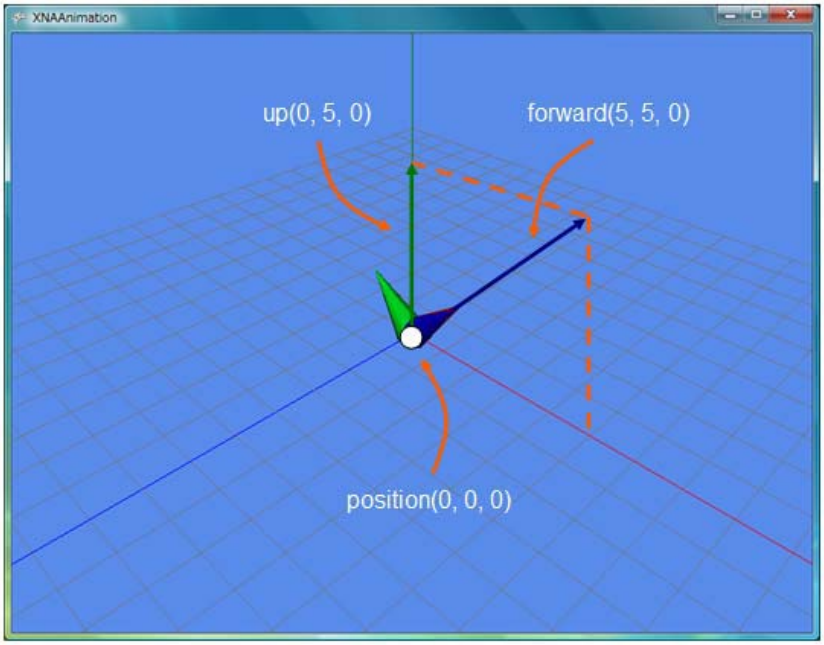
正しく実装されれば、Updateメソッド内で、以下のように記述したとき、

```
Vector3 vZ = new Vector3(5.0f, 5.0f, 0);
Vector3 vY = new Vector3(0f, 5.0f, 0);;
Vector3 vP = new Vector3(0, 0, 0);
mLocal = CreateWorldMatrix(vP, vZ, vY);
```

以下のように表示される。

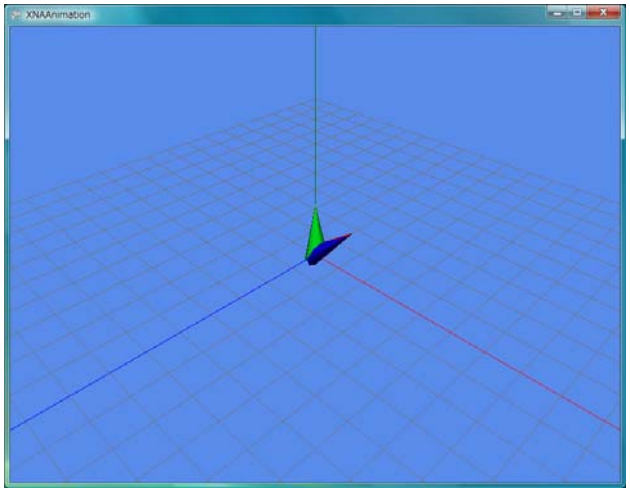


引数に指定された各ベクトルを以下に示す。
モデルの正面(Z軸方向)がforward方向へ向くようにローカル座標系が設定されていることが分かる。

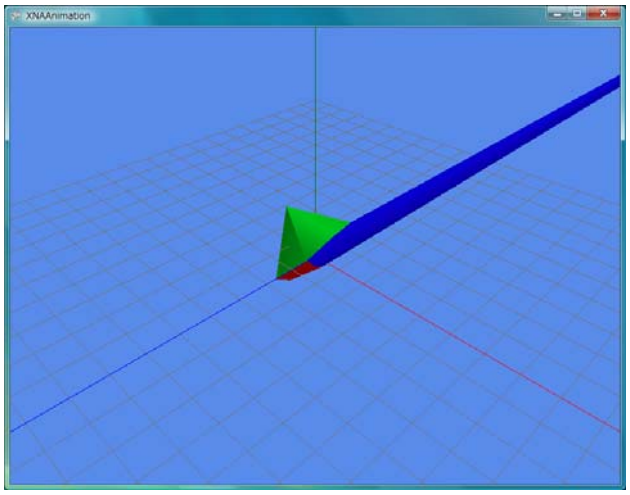


以下に、実装失敗例を示す。

- Y軸が歪んでいる



- スケールが正しくない



いずれも、次のような原因で起こる。

- X、Y、Z軸のいずれかが正規化されていない
- X、Y、Z軸のいずれかがそれぞれ直交していない