

Away3D Basics 5 – Primitives(Part 1)

本　　ド　　キ　　ュ　　メ　　ン　　ト　　は　　、
http://www.flashmagazine.com/Tutorials/detail/away3d_basics_5_-_primitives_part_1/で
公開されている「Away3D Basics 5 – Primitives(Part 1)」を、ヒム・カンパニー 永井勝則が自主的
に日本語に訳したものです。

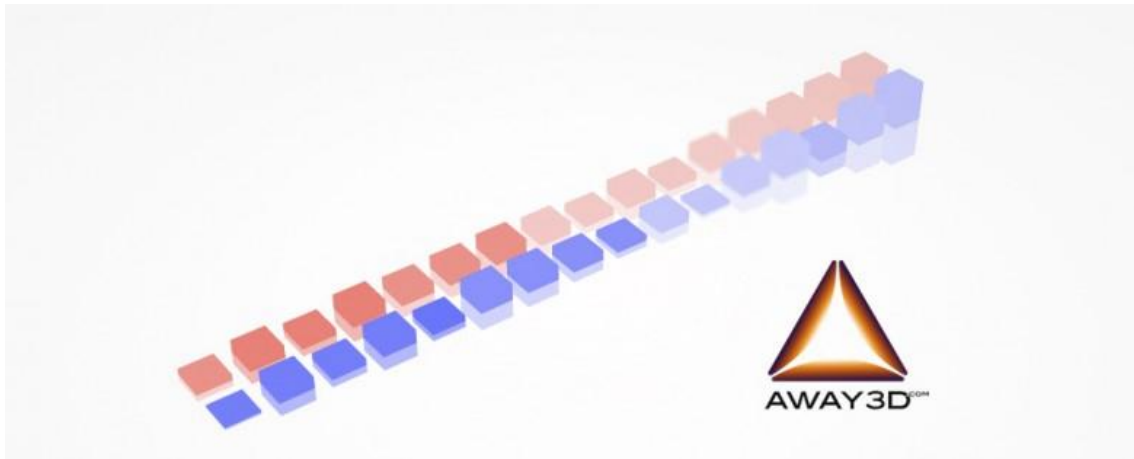
<http://www.himco.jp/>

knagai@himco.jp

(2009/11)

原文は、

http://www.flashmagazine.com/Tutorials/detail/away3d_basics_5_-_primitives_part_1/
で読むことができます。



Away3D の基本5: プリミティブ(パート1)

プリミティブは多くの 3D ソフトウェアに欠かせない要素です。Away3D には補助ツールとともに 17 のプリミティブが備わっています。このプリミティブに関するチュートリアルパート1ではこれらのプリミティブを使って、サウンドビジュアライザーや CMYK カラー分離ビューといった見栄えのする効果を実現する方法を探っていきます。本チュートリアルではまた、Away3D がサポートする2つのコーディングスタイルも紹介します。

まず第一に、プリミティブを軽くみはいけません。プリミティブは単純かもしれませんが、非常に役立つのです。たとえば単純な球体はパノラマビューにはなくてはならないものです。立方体のパノラマも作りたいですか？ これには Skybox プリミティブが使用できます。

プリミティブに関するチュートリアルはパート3まであります。パート1と2では多くの場合で有用に使用できるプリミティブを、パート3では海がめのような少し変わったプリミティブを見ていきます。海がめ？ そう海がめです。ほかの 3D エンジンには牛のプリミティブがあります。Away3D の場合には海がめのプリミティブがあるのです。

必要とされる予備知識

本チュートリアルはわれわれの[Away3D チュートリアル](#)上に構築されています。Flash 3D が初めての方は本チュートリアル以前のチュートリアルに一通り目を通されたほうがよいでしょう。各サンプルには完全なソースファイルをつけています。付随する ActionScript ファイルへのリンクをクリックするとそのサンプルのクラスファイルがダウンロードできます。また多くのサンプルでは Cover.as ファイルが必要です。これは多くの Flash ファイルを一度に表示しようとする際に発生する可能性のあるマシンのフリーズを避けるために使用します (Flash 3D は多くのマシンリソースを消費するので、一度に多くの 3D を再生するとマシンが固まる恐れがあります)。サンプルのクラスファイルの使用方法がよく分からないという方は、[このチュートリアル](#)にまず目を

通してください。

本チュートリアルサンプルの中にはテクスチャを使うものがあります。テクスチャとマテリアルについては別のチュートリアルでもっと詳しく見ていきますが、Flash CS3 や CS4 でソースファイルを作成したい場合には、[このチュートリアル](#)を先に読んでください。

三角形 (Triangle)

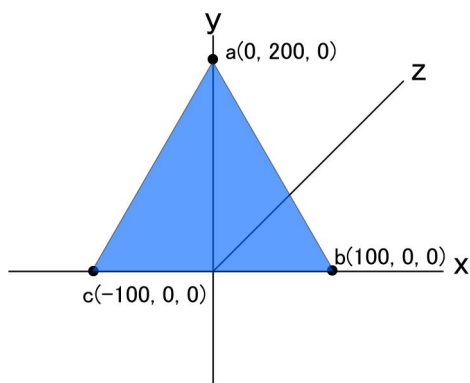
すべての 3D モデルの基本的な構成要素は三角形です。三角形そのものはさほど有用ではありませんが、多数の三角形を組み合わせることで、見事な結果を生み出すのです。Away3D では商用 3D ソフトウェアから複雑なモデルを読み込むことができるので、三角形からモデルを構築する必要はそう頻繁にはないでしょう。しかし三角形にはそれ自体の目的があります。三角形は最小の 3D 要素なので、パーティクル効果などのリソースインテンシブな操作によく使用されます。

単一の三角形を作成するには、3D 空間の点に a、b、c プロパティを設定します。この3つの各点は、次のように x、y、z 位置を持つ頂点で定義されます。

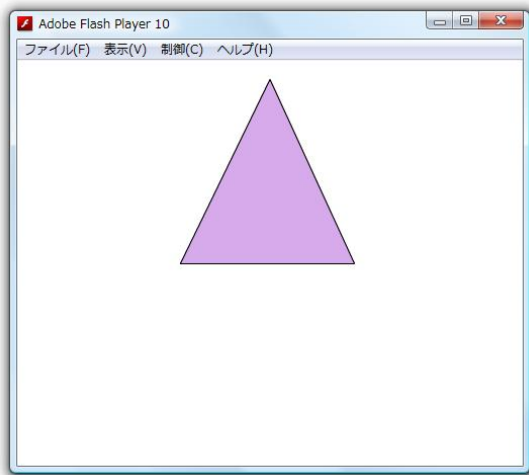
```
var tri:Triangle = new Triangle();
tri.a = new Vertex(0,200,0);
tri.b = new Vertex(100,0,0);
tri.c = new Vertex(-100,0,0);
tri.bothsides = true;
view.scene.addChild(tri);
```

訳者注：

上記頂点を座標で打っていくと、次の図のようになります。これはつまり z 方向に奥行きのない、y 軸で対称な二等辺三角形です。



以下はそのサンプルです。



ムービー: [Basic08_triangle.as](#)

ここでは `bothsides` (フェースの前面と裏面両方をレンダリングするかどうかを示す) プロパティを `true` に設定している点に注目してください。これを設定しないとこの三角形は片側しか見えなくなります。

Basic08_triangle.as

```
package {  
  
    import away3d.containers.View3D;  
    import away3d.core.base.Vertex;  
    import away3d.primitives.Triangle;  
  
    import flash.display.Sprite;  
    import flash.events.Event;  
  
    [SWF(width="500", height="400",  
        frameRate="60", backgroundColor="#FFFFFF")]  
    public class Basic08_triangle extends Sprite{  
  
        private var tri:Triangle;  
        private var view:View3D;
```

```

public function Basic08_triangle() {
    // ビューの作成
    view = new View3D({x:250,y:200});
    addChild(view);

    // 三角形を追加
    tri = new Triangle();
    tri.a = new Vertex(0,200,0);
    tri.b = new Vertex(100,0,0);
    tri.c = new Vertex(-100,0,0);
    tri.bothsides = true;
    view.scene.addChild(tri);

    this.addEventListener(Event.ENTER_FRAME, update);
}

// 三角形を回転させ、ビューをレンダリング
private function update(e:Event):void {
    tri.rotationY += 1;
    view.render();
}
}
}

```

平面 (Plane)

平面は奥行きを持たないフラットな矩形です。デフォルトで平面は2つの三角形から構成されますが、必要に応じて三角形の数を増やすことができます。平面はすべての Away3D プリミティブの中で2番めに軽量のプリミティブなので、多くの作業に利用できます。ビデオウォールを作成したいですか？ それならばひとつづきの平面を使って、それに MovieMaterial を設定します。平面はまた 3D シーンの地面を表すのに非常に便利です。例として[このデモ](#)を見てください。

最初に述べたように、Away3D は基本的な2つのコーディングスタイルをサポートします。オブジェクトを作成するとき、コンストラクタ内でプロパティを渡すこともできますし、オブジェクトを作成した後でプロパティを設定することもできます。この2つの方法から可能になることを見てください。

最初の方法では、“魔法のように自動的な”初期化オブジェクトを使用しています。初期化オブジェクトを使うと、オブジェクトのすべてのパラメータを渡すことができます。次のコードでは黒いワイヤーを持つ白のワイヤーフレームのマテリアルで平面を作成しています。

```
var myPlane:Plane = new Plane({material:"white#black",rotationX:-90});
View.scene.addChild(myPlane);
```

これは新しいオブジェクトを作成する際の非常にコンパクトな方法です。この方法を使うと簡単にワイヤーフレームのカラーが指定でき、Away3D がみなさんに代わって適切な WireColorMaterial を作成して適用します。またこれらのプロパティは、次のように1行で1つずつ宣言することもできます。

```
var mat:WireColorMaterial = new WireColorMaterial();
mat.color = 0xff0000;
mat.wirecolor = 0x000000;
var myPlane:Plane = new Plane();
myPlane.material = mat;
myPlane.rotationX = -90;
View.scene.addChild(myPlane);
```

オブジェクトを作成する方法はコードがかなり長くなりますが、コードが読みやすいという理由でこれを好む人は多くいます。どちらの方法を使うかはみなさん自身が決めることですが、多くの開発者はこのテーマに一家言持っています。Away3D ではこの両方のコーディングスタイルをサポートすることで、すべての使用者がハッピーになります。

おそらく気づいておられるでしょうが、上記の例では平面 (myPlane) を x 軸で-90 度回転させています。平面には奥行きがないので、見えるようにするにはこちら向きに回転させる必要があるのです。

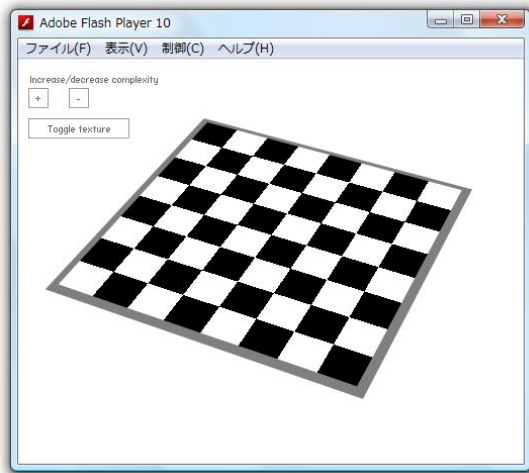
また平面の複雑性はセグメントのパラメータを設定することで調整できます。

```
myPlane.segmentsW = 4;
myPlane.segmentsH = 6;
```

訳者注:

Plane.segmentsH は平面を構成する垂直方向のセグメント数を定義し、Plane.segmentsW は平面を構成する水平方向のセグメント数を定義します。

これによって平面の三角形の密度を大きくすることができるので、三角形の数が少なすぎてテクスチャが正確に表示できない場合、テクスチャの歪曲を補正したときに便利です。この問題は次のサンプルで見ることができます。ボタンをクリックして試してみてください。



ムービー: [Textures.as](http://www.himco.jp/articles/images/resources.zip)

訳者注:

このチュートリアルシリーズでは画像などは一切提供されていないので、サンプル作成にあたっては訳者が手作りしています。chess.jpg や 次の サンプル の CMYK 画像は <http://www.himco.jp/articles/images/resources.zip> で用意しています。

Textures.as

```
package {  
  
    import away3d.containers.*;  
    import away3d.core.base.*;  
    import away3d.core.math.*;  
    import away3d.core.utils.Cast;  
    import away3d.events.*;  
    import away3d.materials.*;  
    import away3d.primitives.*;  
    import away3d.core.render.Renderer;  
  
    import com.bit101.components.*;
```

```

import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.display.Sprite;
import flash.events.Event;
import flash.events.MouseEvent;

[SWF(width="500", height="400",
     frameRate="30", backgroundColor="#FFFFFF")]
public class Textures extends Sprite {

    // チェスイメージが乗る平面
    private var plane:Plane;
    // チェスの台用の平面
    private var planeBorder:Plane;
    private var planeComplexity:Number = 1;
    private var my_material:IMaterial;
    private var cover:Cover;
    private var View:View3D;
    private var panel:Panel;
    private var toggle:Boolean = false;

    // Flex 用にテクスチャを埋め込む
    [Embed(source="resources/chess.jpg")]
    private var chessBitmap:Class;
    private var chessTexture:BitmapData;

    public function Textures() {
        // super()は記述しなくても ActionScript が代行してくれる
        super();

        View = new View3D({x:250, y:200});
        // 正確なレンダリングを保証するために、
        // ソート後三角形の順番づけをやり直す
        View.renderer = Renderer.CORRECT_Z_ORDER;
        addChild(View);
    }
}

```

```

// テクスチャを保持し、平面を作成
var bitmap:Bitmap = new chessBitmap() as Bitmap;
// チェスのビットマップデータ
chessTexture = bitmap.bitmapData;
my_material = new BitmapMaterial(
    Cast.bitmap(chessTexture));
makePlane();

// カメラの位置と見る方向
View.camera.position = new Number3D(400, 500, 400);
View.camera.lookAt( new Number3D(0, 0, 0) );

// コントロールの追加
addControls();

View.render();
cover = new Cover(this);
addChild(cover);

addEventListener(Event.ENTER_FRAME, onEnterFrame);
}

private function onEnterFrame(e:Event):void {
    update();
}

// 毎フレーム呼び出される
private function update(e:MouseEvent = null):void {
    if(!cover.visible)
    {
        // ビューをレンダリングし、2つの平面を回転
        View.render();
        plane.rotationY = planeBorder.rotationY += 1;
    }
}
}

```

```

// コンストラクタとボタンから呼び出される
private function makePlane():void {
    var rot:Number;
    if(plane){
        // 平面の回転値をメモ
        rot = plane.rotationY;
        // 平面を解体してなくす
        destroy();
    }

    // 平面を新しいセグメント数で作成し直す
    plane = new Plane(
        {material:my_material,width:250,height:250,
        x:0,y:30,segmentsH:planeComplexity,
        segmentsW:planeComplexity});
    planeBorder = new Plane(
        {material:"grey",width:265,height:265,x:0,y:28,
        segmentsH:1, segmentsW:1});

    //平面をシーン追加
    View.scene.addChild(planeBorder);
    View.scene.addChild(plane);

    // メモしておいた回転値を代入して元に戻す
    if(rot){
        plane.rotationY = planeBorder.rotationY = rot;
    }
}

// コントロールを作成
private function addControls():void {
    var pad:Number = 10;
    var label1:Label = new Label(this, pad, pad);
    label1.autoSize = false;
    label1.width = 180 -(pad*2);
}

```

```

label1.text = "Increase/decrease complexity";

// 複雑性を増す+ボタン
var plusButt:PushButton = new PushButton(
    this, pad, label1.height+label1.y, "+", increaseComplexity);
plusButt.width = plusButt.height = 20;

// 複雑性を減らす-ボタン
var minButt:PushButton = new PushButton(
    this, 50, label1.height+label1.y, "-", decreaseComplexity);
minButt.width = minButt.height = 20;

// テクスチャを切り替えるボタン
var toggleButt:PushButton = new PushButton(
    this, pad, plusButt.height+plusButt.y+pad,
    "Toggle texture", wiresTextureToggle);
toggleButt.height = 20;
}

// 複雑性を減らす
private function decreaseComplexity(e:MouseEvent):void {
    // planeComplexity が 0 より大きいなら 1 減らし、
    // そうでないなら 1 にする
    planeComplexity > 0 ?
        planeComplexity -= 1 : planeComplexity = 1;
    makePlane();
}

// 複雑性を増す
private function increaseComplexity(e:MouseEvent):void {
    planeComplexity += 1;
    makePlane();
}

private function wiresTextureToggle(e:MouseEvent):void {
    trace("wiresTextureToggle"+toggle);
}

```

```

        if(!toggle){
            // マテリアルをワイヤーフレームにする
            my_material = new WireColorMaterial(
                {color:"blue", lighting:true});
            toggle = true;
        } else {
            // マテリアルをチェスのイメージにする
            my_material = new BitmapMaterial(
                Cast.bitmap(chessTexture));
            toggle = false;
        }
        makePlane();
    }

    // 平面を解体してないものとする
    private function destroy():void {
        if(plane){
            plane.material = null;
            View.scene.removeChild(plane);
            View.scene.removeChild(planeBorder);
            plane = null;
            planeBorder = null;
        }
    }
}

```

訳者注:

このクラスを実行するにはプロジェクトフォルダに MinimalComps コンポーネント、Cover.as を配置し、resources フォルダに chess.jpg を配置します。具体的には Flash CS4 では、MinimalComps コンポーネントの com と assets フォルダ、Cover.as を FLA ファイルのあるフォルダにコピーします。このクラスでは Flex コンパイラの機能を使用するので、ライブラリパスに flex.swc へのパス (\$(FlexSDK)/frameworks/libs/flex.swc など)を追加します。

Flex コンパイラを使用せず、Flash のライブラリに chess.jpg を読み込んで使用する場合には、読み込ん

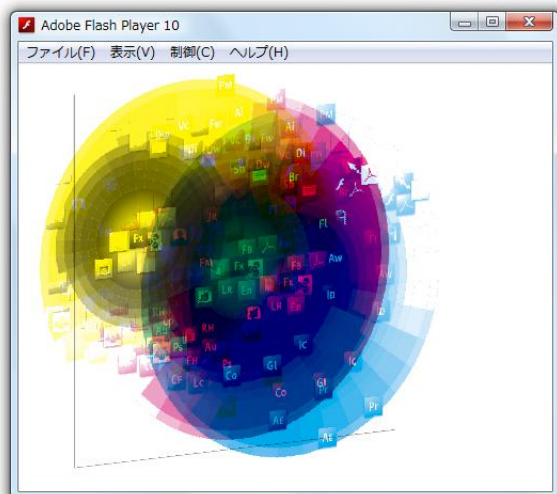
だ chess.jpg の[ビットマッププロパティ]ダイアログボックスで[ActionScript に書き出し]を選択し、[リンクエー
ジ]の[クラス]で ChessTexture を指定します。クラスのコードを次のように変更します。

```
// Flex 用にテクスチャを埋め込む
//[Embed(source="resources/chess.jpg")]
// private var chessBitmap:Class;
private var chessTexture:BitmapData;

...

// テクスチャを保持し、平面を作成
//var bitmap:Bitmap = new chessBitmap() as Bitmap;
// チェスのビットマップデータ
//chessTexture = bitmap.bitmapData;
chessTexture = new ChessTexture(0,0);
```

では平面を使った実際のコードを見ていきましょう。次のサンプルでは、[Adobe の John Nack のブログで公開されている CS3 アイコンの全セット](#)を見ることが出来ます。画面をクリックしドラッグすると、カラーが立体的に分離して見えます。



ムービー: Basic08_plane.as

このサンプルでは、部分的に透明な4つの PNG ファイルを4つ (wheelC.png、wheelM.png、wheelY.png、wheelK.png) 作成し、シアン、マゼンタ、イエロー、ブラックの CMYK が紙上で組み合わせられて印刷されフルカラーになるカラー印刷の効果をシミュレーションしています。カメラには

HoverCamera3D を使用し、CMYK カラーを表す平面4つと、紙の役割を果たす白い1つの平面を回転させます。BlendMode.MULTIPLY(表示オブジェクトの要素カラーの値と背景色の要素カラーの値を乗算した後、0xFF で割って正規化し、色を暗くします)を適用すると各レイヤーのカラー値を組み合わせることができるので、新聞の印刷と似た結果が生まれます。

```
package {

    import away3d.cameras HoverCamera3D;
    import away3d.containers.*;
    import away3d.core.base.*;
    import away3d.core.light.*;
    import away3d.core.math.*;
    import away3d.events.*;
    import away3d.materials.*;
    import away3d.primitives.*;

    import flash.display.Bitmap;
    import flash.display.BlendMode;
    import flash.display.Sprite;
    import flash.display.Stage;
    import flash.events.Event;
    import flash.events.MouseEvent;

    [SWF(width="500", height="400", frameRate="30", backgroundColor="#ffffff")]
    public class Basic08_plane extends Sprite{

        private var View:View3D;
        private var swfStage:Stage;
        private var cover:Cover;
        private var waitFourFrames:Number = 0;

        // ホバーカメラの制御
        private var camera:HoverCamera3D;
        private var lastMouseX:Number;
        private var lastMouseY:Number;
        private var lastPanAngle:Number;
```

```

private var lastTiltAngle:Number;
private var move:Boolean = false;

// 埋め込みイメージ
[Embed (source="resources/wheelC.png")]
private var cTex:Class;
[Embed (source="resources/wheelM.png")]
private var mTex:Class;
[Embed (source="resources/wheelY.png")]
private var yTex:Class;
[Embed (source="resources/wheelK.png")]
private var kTex:Class;
[Embed (source="resources/wheelW.png")]
private var wTex:Class;

public function Basic08_plane() {
    super();

    camera = new HoverCamera3D(
        {zoom:18, focus:2000, distance:18000});
    View = new View3D({camera:camera,x:250, y:200});
    addChild(View);

    camera.lookAt( new Number3D(0, 0, 0) );
    camera.targetpanangle = camera.panangle = 0;
    camera.targettiltangle = camera.tiltangle = 0;
    camera.mintiltangle = -90;

    // シアン用平面の作成と追加
    var cPlane:Plane = new Plane(
        {material:cTex,width:200,height:200,
        bothsides:true,z:0,ownCanvas:true});
    cPlane.rotationX = 90;
    cPlane.rotationY = 180;
    // blendMode は Plane が継承する Object3D の機能
    cPlane.blendMode = BlendMode.MULTIPLY;

```

```
View.scene.addChild(cPlane);

// マゼンタ用平面の作成と追加
var mPlane:Plane = new Plane(
    {material:mTex,width:200,height:200,
     bothsides:true,z:-50,ownCanvas:true});
mPlane.rotationX = 90;
mPlane.rotationY = 180;
mPlane.blendMode = BlendMode.MULTIPLY;
View.scene.addChild(mPlane);

// イエロー用平面の作成と追加
var yPlane:Plane = new Plane(
    {material:yTex,width:200,height:200,
     bothsides:true,z:-100,ownCanvas:true});
yPlane.rotationX = 90;
yPlane.rotationY = 180;
yPlane.blendMode = BlendMode.MULTIPLY;
View.scene.addChild(yPlane);

// ブラック用平面の作成と追加
var kPlane:Plane = new Plane(
    {material:kTex,width:200,height:200,
     bothsides:true,z:-150,ownCanvas:true});
kPlane.rotationX = 90;
kPlane.rotationY = 180;
kPlane.blendMode = BlendMode.MULTIPLY;
View.scene.addChild(kPlane);

/// 白い紙用平面の作成と追加
var wPlane:Plane = new Plane(
    {material:wTex,width:200,height:200,
     bothsides:true,z:-200});
wPlane.rotationX = 90;
View.scene.addChild(wPlane);
```

```

View.render();
camera.hover();

cover = new Cover(this,500,400,
    "Adobe product icons, from John Nack's blog");

addEventListener(
    Event.ENTER_FRAME, onEnterFrame);
this.stage.addEventListener(
    MouseEvent.MOUSE_DOWN, MouseDown);
this.stage.addEventListener(
    MouseEvent.MOUSE_UP, MouseUp);
}

private function onEnterFrame(e:Event):void {
    if(!cover.visible || waitFourFrames < 4) {
        var cameraSpeed:Number = 0.3;

        if (move) {
            camera.targetpanangle =
                cameraSpeed*(stage.mouseX -
                    lastMouseX) + lastPanAngle;
            camera.targettiltangle =
                cameraSpeed*(stage.mouseY -
                    lastMouseY) + lastTiltAngle;
        }
        camera.hover();
        View.render();
        waitFourFrames++;
        if(waitFourFrames == 4){
            cover.refresh();
            addChild(cover);
        }
    }
}
}

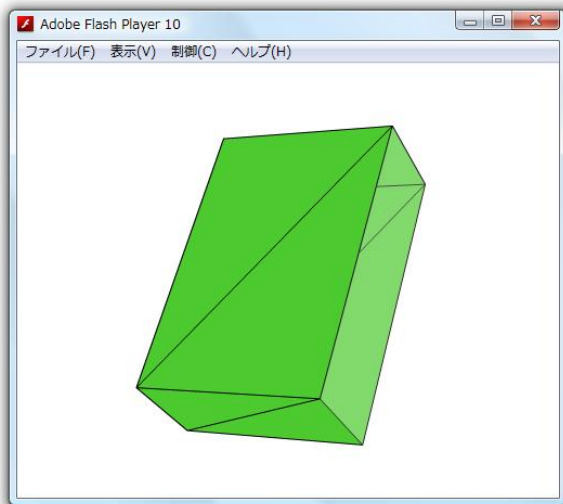
```

```
private function MouseDown(event:MouseEvent):void {
    lastPanAngle = camera.targetpanangle;
    lastTiltAngle = camera.targettiltangle;
    lastMouseX = stage.mouseX;
    lastMouseY = stage.mouseY;
    move = true;
}

private function MouseUp(event:MouseEvent):void {
    move = false;
}
}
}
```

直方体 (Cube)

直方体には奥行きと高さ、幅があります。また各側面にはマテリアルが設定できます。



[Basic08_cube.as](#)

直方体も、Away3D のほかのオブジェクトと同様に、2つの方法で作成できます。

```
cube = new Cube({width:200,height:100,depth:300});
```

1行ずつ作成する方法もあります。この方が読みやすくなります。

```
var cube:Cube = new Cube();  
cube.width = 200;  
cube.height = 100;  
cube.depth = 300;
```

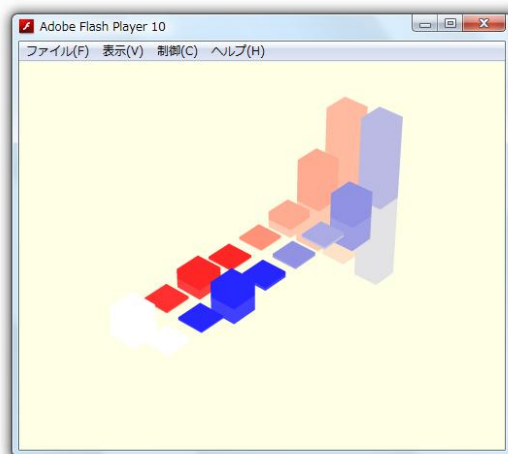
直方体にマテリアルを設定するには、`cubeMaterials` プロパティを使用します。これによって上面、底面、左面、右面、正面、背面のマテリアルが設定できます。

```
cube.cubeMaterials.left = new ColorMaterial(0xffffffff,{alpha:.3});
```

すべての 3D オブジェクトは、位置と回転に関する `x`、`y`、`z` 軸で、次のように操作できます。

```
cube.x = 200;  
cube.rotationY = 45;
```

オブジェクトの移動と操作については、前の「[Away3D の基本4: 3D オブジェクトの操作](#)」を参照してください。プロジェクトで使用する直方体を見ていきましょう。次のサンプルは、再生中の音楽を分析し、そのサウンド波形をもとに高さとカラー、フィルタを調整するサウンドビジュアライザーです。



ムービー: [Basic08_cube_music.as](#)

Basic08_cube_music.as

```
package {

    import away3d.cameras HoverCamera3D;
    import away3d.containers View3D;
    import away3d.materials.BitmapMaterial;
    import away3d.materials.ColorMaterial;
    import away3d.primitives.Cube;
    import away3d.primitives.RegularPolygon;
    import away3d.core.utils.Cast;

    import flash.display.Bitmap;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.filters.BlurFilter;
    import flash.media.Sound;
    import flash.media.SoundChannel;
    import flash.media.SoundMixer;
    import flash.net.URLRequest;
    import flash.utils.ByteArray;
    import flash.utils.Timer;

    [SWF(width="500", height="400", frameRate="60", backgroundColor="#ffffff")]
    public class Basic08_cube_music extends Sprite {

        private var cam:HoverCamera3D;
        private var lastKey:uint;
        private var keyIsDown:Boolean = false;
        private var view:View3D;
        private var cover:Cover;

        // 伸び縮みするキューブの数
        private var numCubes:Number = 16;
        // 振幅(キューブの高さ)
        private var amplitude:Number = 70;
        private var cubes:Array;
        private var cubesMirrored:Array;
```

```
private var cubeColors:Array;
private var mp3:Sound;
private var soundBytes:ByteArray;
private var soundPlaying:Boolean = false;
private var soundValues:Array;
private var pausePosition:int = 0;
private var channel:SoundChannel;
private var highlightFilter:Array;
private var maxLev:Number = 0;

private var t:Timer;

[Embed (source="resources/greyish.jpg")]
private var greyTexture:Class;

public function Basic08_cube_music()    {

    cubes = new Array();
    cubesMirrored = new Array();
    cubeColors = new Array();
    soundBytes = new ByteArray();
    soundValues = new Array();

    // mp3 の設定とロード
    mp3 = new Sound();
    mp3.load(new URLRequest("redrum.mp3"))
    channel = new SoundChannel();

    // 配列をあらかじめ 0 で埋めておく
    for(var p:int = 0; p<numCubes;p++){
        soundValues.push(0);
        cubeColors.push(0);
    }

    // ホバーカメラの設定
    cam = new HoverCamera3D();
```

```
cam.z = -1000;
cam.panangle = 20;
cam.tiltangle = 20;
cam.targetpanangle = 20;
cam.targettiltangle = 20;
cam.mintiltangle = -90;
cam.zoom = 16;

// ビューを作成して追加
view = new View3D({camera:cam,x:250,y:200});
addChild(view);

// キューブを作成
var i:int; var zPos:int;
// i を 0 から numCubes まで変化させて、zPos と xPos を決める
for(i = 0; i < numCubes ; i++){
    if(i < (numCubes/2)){
        zPos = 15;
    } else {
        zPos = -15;
    }
    // 30 ずつ間をとって xPos を決める
    var xPos:Number =
        (i%(numCubes/2)*30)-(((numCubes/2)*30)/2)+15;

    // y=5 の位置にキューブを配置
    var cubeMaterial:ColorMaterial =
        new ColorMaterial(0xffffffff);
    var cube:Cube = new Cube(
        {material:cubeMaterial,width:20,height:1,
        depth:20,x:xPos,z:zPos,y:5,ownCanvas:true});
    view.scene.addChild(cube);
    cubes[i] = cube;

    // y=-5 の位置にキューブを配置
    var cubeMaterialM:ColorMaterial =
```

```

        new ColorMaterial(0xffffffff);
        var cubeM:Cube = new Cube(
            {material:cubeMaterialM,width:20,height:1,
            depth:20,x:xPos,z:zPos,y:-5,alpha:1,
            ownCanvas:true});
        view.scene.addChild(cubeM);
        cubesMirrored[i] = cubeM;
    }

    // テクスチャをつけた平面を追加
    var planeMat:BitmapMaterial = new BitmapMaterial(
        Cast.bitmap(greyTexture),{alpha:0.5});
    var plane:RegularPolygon = new RegularPolygon(
        {material:planeMat,radius:500,sides:14,ownCanvas:true});
    plane.ownCanvas = true;
    view.scene.addChild(plane);

    // キューブをぼかすフィルタの作成
    highlightFilter = new Array( new BlurFilter() );

    cam.hover();
    view.render();

    cover = new Cover(this,500,400,
        "Ugress - Redrum (1 minute excerpt)");
    addChild(cover);

    this.addEventListener(
        Event.ENTER_FRAME,onEnterFrame);

    //カバーからのイベントを監視
    cover.addEventListener(Cover.COVERD, playSound);
    cover.addEventListener(Cover.UNCOVERD, stopSound);
}

private function onEnterFrame(e:Event):void {

```

```

if(!cover.visible) {
    // サウンドから 512 個の振幅データを取得し、
    // バイト配列に保持
    SoundMixer.computeSpectrum(soundBytes,true);
    var tempValue:Number = 0;
    var newValues:Array = new Array();
    // キューブ 1 個が担当する振幅データの数
    var valuesToMerge:Number =
        Math.floor(512/numCubes);
    var i:uint;

    // 512 個のサンプルからボリュームの平均値を得る
    for (i = 0; i < 512; i++) {
        // バイト配列からレベルを読み取る
        var lev:Number =
            soundBytes.readFloat() * amplitude;
        tempValue += lev;
        if(i%valuesToMerge == valuesToMerge-1){
            var avg:Number =
                tempValue/numCubes;
            newValues.push(avg);
            tempValue = 0;
        }
    }

    // キューブを更新
    var l:int = newValues.length;
    var newSoundValues:Array = new Array();
    for (i = 0; i < l; i++) {
        var currentCube:Cube = cubes[i] as Cube;
        var currentCubeMirror:Cube =
            cubesMirrored[i] as Cube;
        // 値か前の値より大きい場合には、白くする
        if(newValues[i] > soundValues[i]){
            newSoundValues[i] = newValues[i];
            cubeColors[i] = 128;
        }
    }
}

```

```

        currentCube.filters =
            highlightFilter;
        currentCubeMirror.filters =
            highlightFilter;
    } else {
// 低い場合にはフェードを維持
        var dampedValue:Number =
            soundValues[i]-3;
        if(dampedValue<1){
            dampedValue = 1; }
        cubeColors[i] -= 10;
        newSoundValues[i] =
            dampedValue;
        currentCube.filters = null;
        currentCubeMirror.filters = null;
    }

// キューブと反射キューブの高さを調節
currentCube.height = newSoundValues[i];
currentCube.y = newSoundValues[i]/2;
currentCubeMirror.height =
    newSoundValues[i];
currentCubeMirror.y =
    newSoundValues[i]/2*-1;

// カラーを調整
var adjustedColor:uint = cubeColors[i];
var colString:String;
var colStringMirror:String;
var increase:Number = 1.7;
// 反射キューブをどれだけ白くするか
if(i<1/2){
    // 青のキューブ
    colString =
        "0x"+ensureTwoDigits(
            adjustedColor)+

```

```

        ensureTwoDigits(
            adjustedColor)+"ff";
        colStringMirror =
            "0x"+ensureTwoDigits(
                adjustedColor*increase)+
                ensureTwoDigits(
                    adjustedColor*increase)+"ff";
    } else {
        // 赤のキューブ
        colString =
            "0xff"+ensureTwoDigits(
                adjustedColor)+
                ensureTwoDigits(adjustedColor);
        colStringMirror =
            "0xff"+ensureTwoDigits(
                adjustedColor*increase)+
                ensureTwoDigits(adjustedColor*increase);
    }

    // マテリアルを読み取り更新
    var col:ColorMaterial =
        currentCube.material as ColorMaterial;
    var colM:ColorMaterial =
        currentCubeMirror.material as ColorMaterial;
    col.color = uint(colString);
    colM.color = uint(colStringMirror);
}

// soundValues を新しい値で更新
soundValues = newSoundValues.concat();
// カメラをキューブの周りで周回させる
cam.targetpanangle += .5;
cam.hover();
view.render();
}
}

```

```

// 16 進数値を調整
private function ensureTwoDigits(n:uint):String{
    var s:String;
    if(n > 255){ n = 255;}
    if(n < 1){ n = 1;}
    if(n < 16){
        s = "0"+n.toString(16);
    } else {
        s = n.toString(16);
    }
    return s;
}

// カバーからのイベントで引き起こされる
private function playSound(e:Event):void {
    if(!soundPlaying){
        soundPlaying = true;
        channel = mp3.play(pausePosition);

if( !channel.hasEventListener(Event.SOUND_COMPLETE) ){
            channel.addEventListener(
                Event.SOUND_COMPLETE, soundCompleteHandler);
        }
    }

private function stopSound(e:Event):void {
    pausePosition = channel.position;
    channel.stop();
    soundPlaying = false;
}

// mp3 の再生が終了したら再スタート
private function soundCompleteHandler(e:Event):void {
    pausePosition = 0;

```

```
        soundPlaying = false;
        playSound(e);
    }
}
}
```

訳者注:

オリジナルのサンプルコードのままでは動作しなかったため、次のように変更しています。

// オリジナルのサンプルコードの次の2行を

```
cover.addEventListener(Event.ACTIVATE, stopSound);
cover.addEventListener(Event.DEACTIVATE, playSound);
```

```
cover.addEventListener(Cover.COVERD, playSound);
cover.addEventListener(Cover.UNCOVERD, stopSound);
```

に変更し、さらに Cover.as に、

// 定数を追加し、

```
public static const COVERD:String = "coverd";
public static const UNCOVERD:String = "uncoverd";
```

hideCover()と showCover()に dispatchEvent()の行を追加(太字のコード)。

```
private function hideCover(e:MouseEvent):void
{
    trace("hideCover "+e.stageX);
    if(e.stageX < stageW && e.stageX > 0 && e.stageY < stageH && e.stageY > 0){
        this.visible = false;
        dispatchEvent(new Event(COVERD));
    }
}
private function showCover(e:Event = null):void
{
    if(!this.visible){
        refresh();
    }
}
```

```
    }  
    // Turn on cover  
    this.visible = true;  
    dispatchEvent(new Event(UNCOVERD));  
}
```

訳者注:

このクラスのコードは相当に複雑で、訳者の理解を超えています。コンストラクタのキューブを作成する for ループの最初では、zPos と xPos という変数を使って、配置するキューブの z と x を決めようとしています。この部分は何をやっているのか、式を見ただけでは分かりませんが、trace() を使って出力してみると、想像はつきます。

```
var numCubes:uint= 16;  
var i:int;  
var zPos:int;  
  
for (i = 0; i < numCubes; i++) {  
    if (i < (numCubes/2)) {  
        zPos=15;  
    } else {  
        zPos=-15;  
    }  
    var xPos:Number = (i%(numCubes/2)*30)-(((numCubes/2)*30)/2)+15;  
  
    trace("i : " + i);  
    trace("xPos : " + xPos);  
    trace("zPos : " + zPos);  
    trace("-----")  
}
```

この xPos と zPos を 3D 座標に置くと次のようになります。

以降も相当に難しいコードがつづいているので、解読されたい方は自力で挑戦してみてください。

