Introduction to Sandy 3D 3.0

januari 2008, Steve Stomp

Inhoud

1. Introduction

Internet is growing up. New development is presented almost every day. One these developments are object oriented Actionscript 3D library, like Away3D, Papervision3D and Sandy3d. These 3D library enable developers to create 3D while programming in Flash or Flex. Because this is rather new development, not much documentation is available. That is why I am studying these libraries. This paper focuses on Sandy3D, and gives in introduction on the basics of Sandy3D. For the study I used Sandy3D 3.0.1 in conjunction with Flex Builder 2 and Adobe Flash CS3.

Chapter two discusses how to set up your computer system in order to get Sandy3D running properly. Chapter three give an introduction of a basic Sandy Actionscript file. This followed by a chapter that explains the basics of keyboard and mouse events. Chapter five explains the attributes and materials types of Sandy 3.0. Chapter six gives a brief introduction in sprites. And finally chapter seven which holds the conclusion.

2. Preparing Sandy3D for Adobe Flex

Start up Flex and create a new Actionscript project, RMB click (right mouse button) in the Navigator section, see figure 2.1. Now the wizard will open. Give the project a name, select folders location and
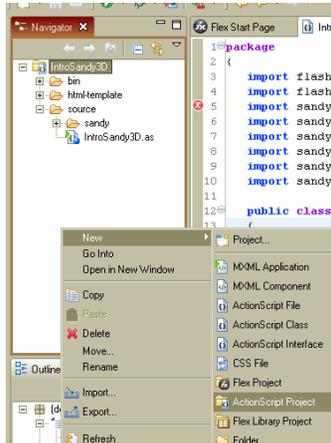

click on next. Create a directory by filling in a name in Main source folder section, for example "source", and click on Finish. Now open up the file explorer and browse to just created folder, and RMB click on the source folder. Select SVN checkout, a new window will open, see figure 2.2. For URL of repository fill in http://sandy.googlecode.com/svn/trunk/sandy/as3/trunk/src/ and click on OK. This will install the latest version on Sandy in project source folder. I use Subversion client, TortoiseSVN for the checkout. http://tortoisesvn.net/

Figure 2.1, Flex Navigator

Figure 2.2, SVN Checkout
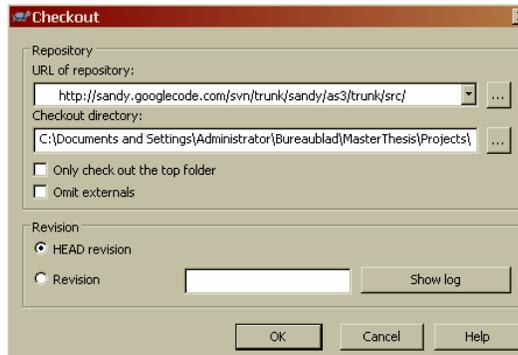
In the project folder, *IntroSandy* you can find all the project files. If you want the compile/execute one of these files out of Flex, they have to be application files. To make a file compliable right mouse click on the file in the navigator and select *Set as Default Application*. Now the actionscript file icon will change to . To run a file, select Run>Run [filename] or Ctrl + F11.

3. Basic Sandy Actionscript File

Every Actionscript file contains some basic elements which are necessary for creating a working Actionscript file. In order to get Flash and Sandy functionality their classes needed to be imported. This is done by declaring it at the top of the file. (see figure 3.1)

```
1 package
2 {
3     import flash.display.Sprite;
4     import flash.events.*;
5     import sandy.core.data.*;
6     import sandy.core.Scene3D;
7     import sandy.core.scenegraph.*;
8     import sandy.materials.*;
9     import sandy.materials.attributes.*;
10    import sandy.primitive.*;
```

Figure 3.1, Importing Flash and Sandy classes

The Sandy (3D) world is structured as a tree, and describes the relationship between the elements in the scene. It contain three types of nodes:

* Group   - general grouping node
* TransformGroup   - used for transformations to all its children
* Shape3D   - 3D object in the scene. (has no children)

In contrast to earlier versions of Sandy3D, it is not necessary to create transform groups in order to move objects. The Shape3D and TransformGroup node now extends ATransformable, which facilitates all kinds of movements[1]. The user looks at the scene through a Camera 3D object. This object can be added to any group node in the 3D world.

```
public class IntroSandy extends Sprite
{
    private var scene:Scene3D;
    private var camera:Camera3D;
    private var torus:Torus;

    public function IntroSandy()
    {
        // We create the camera
        camera = new Camera3D( 500, 500 );
        camera.z = -400;

        // We create the "group" that is the tree of all the
        var root:Group = createScene();

        // We create a Scene and we add the camera and the
        scene = new Scene3D( "scene", this, camera, root );

        // Listen to the heart beat and render the scene
        addEventListener( Event.ENTER_FRAME, loop3D );
```

Figure 3.2, class and function IntroSandy

---

[1] http://www.petitpub.com/labs/media/flash/sandy3/

In order to create a 3D scene we need to declare a variable which will contain all objects in the scene, and to view everything we need a camera object (see figure 3.2). In the method *IntroSandy* the following actions happen:

- The camera object is defined.
- All objects in the scene are grouped in a single variable. In this case the objects are defined by a private method, *createScene*.
- A scene group is defined, by four parameters:
  - The name of the scene
  - The container
  - The camera
  - The objects
- An event listener is called that will render the scene.

```
private function createScene():Group
{
    // Create the root Group
    var g:Group = new Group();

    // Create a cube so we have something to show
    torus = new Torus( "torus",50,50,50);

    // we define a new material
    var materialAttr:MaterialAttributes = new MaterialAttributes(
        new LineAttributes( 0.5, 0x2111BB, 0.4 ),
        new LightAttributes( true, 0.1));

    var material:Material = new ColorMaterial( 0x006633, 1, materialAttr );
    material.lightingEnable = true;
    var app:Appearance = new Appearance( material );

    torus.appearance = app;

    // Object placement
    torus.rotateX = 35;
    torus.rotateY = 35;
    torus.moveHorizontally(200);
    torus.moveUpwards(50);

    // We need to add the cube to the group
    g.addChild( torus );

    return g;
```

Figure 3.3, method createScene

In the method *createScene* a variable is created that will contain all objects (including their attributes, such as materials and translation). This variable will eventually be returned in order to create the scene. Then an object is created. The material for it are assigned by creating a variable of the type *Material*, and the attributes are defined by creating a variable of the type *MaterialAttributes*. This constructor holds a variable number of parameters, in this case two:

- *LineAttributes* hold three parameters:
  - Line thickness
  - Line color
  - Line opacity
- *LightAttributes* hold two parameters:
  - Enable/disable light
  - Level of ambient light

The material attributes are applied to a *ColorMaterial*. Other materials are *BitmapMaterial*, *MovieMaterial*, *VideoMaterial*, etc. More about materials in the chapter 5. To use the light source on the material we need to tell the rendering engine to use the light, *material.lightEnable = true;*. To map the material to the object we need to create variable of the type *Appearance* and assign the material parameter to it, *var app:Appearance = new Appearance (material);*. Finally we place the object in our scene. In this case the object is also rotated.

```
// The Event.ENTER_FRAME event handler tells the world to render
private function loop3D( event : Event ) : void
{
    torus.rotateY += 1; // object movement
    scene.render();
}
```

Figure 3.4, method loop3D

As method names implies this method will loop. Every time the scene is rendered the torus is rotate 1degree in y-axis.

4. Camera and Motion

Standard all objects in a Sandy scene are manipulative. The class ATransformable gives all the possible operations. An overview can be found at

. These operations can be divided in two categories:

- Gobal operations   - operations around the global axis, such as rotation
- Local operations   - operations around the local axis, such as pan and tilt

The operations can assigned by user input. In the following section I will first discuss how to assign keyboard events to manipulate objects and then how to assign mouse events, especially how to control the camera using a mouse. To make everything more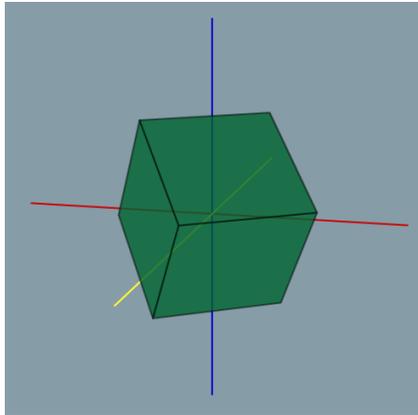 clear I created a xyz-axis, see figure 4.1and IntroSandy.as (appendix A). For this I created a three variables, *xLine, yLine* and *zLine*, which are all of the type Line3D. Then the materials are created (*var matAttrYLine MaterialAttributes = new MaterialAttributes(new LineAttributes(1, 0xFFFF33, 1));*) and assigned to the object (*yLine.appearance = new Appearance (new ColorMaterial (0, 1 , matAttrYLine));*).

Figure 4.1, xyz-axis

4.1. Keyboard Events

In the method *loop3D* we added an eventlistener to catch user input events. The moment a keyboard key is pressed it will call the method *keyPressed*. The (global) operations are assigned to keyboard buttons, see figure 4.2. In this method all operations are executed on the box variable. To change this to camera, the only thing that needs to be done is change the variable of the box variable to the camera variable (*[variable_name.type_of_operation]*).

The way method keypressed is implemented give the user the possibility of executing one operation a time. In other words the object can either be moved/rotated in x or in y direction, and not in both directions. In order to move the object in both directions I made use of an extra Actionscript class, Key.as. This class is created by Senolcular and can be found in the IntroSandy project folder or at http://www.kirupa.com/forum/showpost.php?p=2098269&postcount=319. It recreates the functionality of Key.isDown of AS1 and AS2. In order to use it, it has to be initialized, *Key.initialize(stage).* Now all keyboard events can assign using the following line of code:

- *if (Key.isDown(Keyboard.Up))*
        box.z += 6;

```
private function loop3D( event : Event ) : void
{
    stage.addEventListener(KeyboardEvent.KEY_DOWN, keyPressed);
    scene.render();
}

private function keyPressed( event : KeyboardEvent ) : void
{
    if (event.keyCode == Keyboard.NUMPAD_8)
        box.rotateX -= 6;
    if (event.keyCode == Keyboard.NUMPAD_2)
        box.rotateX += 6;
    if (event.keyCode == Keyboard.NUMPAD_6)
        box.rotateY += 6;
    if (event.keyCode == Keyboard.NUMPAD_4)
        box.rotateY -= 6;
    if (event.keyCode == Keyboard.UP)
        box.z += 6;
    if (event.keyCode == Keyboard.DOWN)
        box.z -= 6;
    if (event.keyCode == Keyboard.RIGHT)
        box.x += 6;
    if (event.keyCode == Keyboard.LEFT)
        box.x -= 6;
    if (event.keyCode == Keyboard.NUMPAD_ADD)
        box.y += 6;
    if (event.keyCode == Keyboard.NUMPAD_SUBTRACT)
        box.y -= 6;
}
```

Figure 4.2, keyPressed

```
public function IntroSandy(
{
    // We create the camera
    camera = new Camera3D(
    camera.x = 100;
    camera.y = 100;
    camera.z = -400;
    camera.lookAt(0,0,0);

    // We create the "group
    var root:Group = create

    // We create a Scene an
    scene = new Scene3D( "s

    Key.initialize(stage);
```

Figure 4.3, Initialize Key.as

4.2.   Mouse Events

The file IntroSandy.as contains two mouse events which executes two methods, one operating on the box objects and one operating the camera object. In the method loop3D, four lines of code are added, see figure 4.4. Every time the mouse button is clicked the method *startCamera* will be executed and when the mouse cursor moves over the objects in the scene the method activate will be executed. These two methods both contain a Boolean variable, *active* and *start* which changes either to true or false. The methods mouseOver and moveCamera contain a if statement which are depended of the Boolean methods. Everytime the variable active becomes true the box objects start to rotate and everyime the start variable becomes true the camera will roll, see figure 4.5 and figure 4.6.

```
stage.addEventListener(MouseEvent.CLICK, startCamera);
stage.addEventListener(MouseEvent.MOUSE_OVER, activate);
stage.addEventListener(Event.ENTER_FRAME, mouseOver);
stage.addEventListener(Event.ENTER_FRAME, moveCamera);
```

Figure 4.4, mouse eventlisteners

```
private function activate (event:MouseEvent) : void
{
    active = !active;
}
private function mouseOver (event:Event) : void
{
    if (active)
    {
        box.rotateX++;
        tg.rotateZ++;
        tg.rotateX++;
        tg.pan += 3;
    }
}

private function startCamera (event:MouseEvent) : void
{
    start = !start;
}

private function moveCamera (event:Event) : void
{
    if (start)
    {
        camera.roll += 1;
    }
}
```

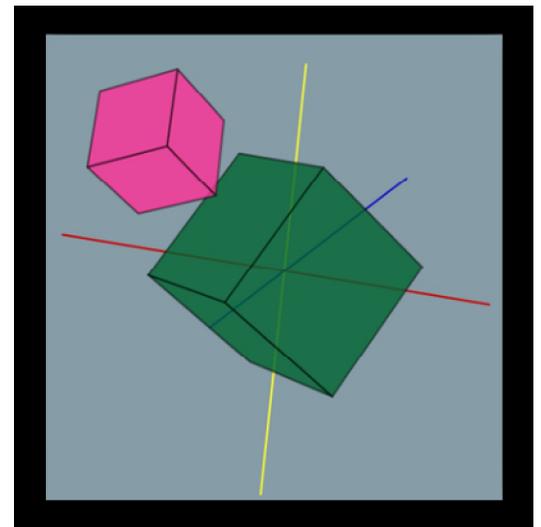

Figure 4.6, scene

Figure 4.5, Mouse methods

4.3.   Controlling Camera Using Mouse and Keyboard Events

In the file, *Fps.as*, two methods are created to navigate through a 3D environment. The method *navigate* uses *Key.as* to assign keyboard events to camera manipulations (see figure 4.6). The numbers represent buttons on the keyboard. In this case; a, d, w and s. For other codes you can visit, http://www.flash-creations.com/notes/asclass_key.php.

```
private function navigate(event:KeyboardEvent) : void {
    if (Key.isDown(65)) {
        camera.moveSideways(-20);
    }
    if (Key.isDown(68)) {
        camera.moveSideways(20);
    }
    if (Key.isDown(87)) {
        camera.moveForward(20);
    }
    if (Key.isDown(83)) {
        camera.moveForward(-20);
    }
}
```

Figure 4.6, function navigate

The method *mouseLook*[2] lets the camera pan and tilt. This is done on basis of the scene (stage.X and stage.Y). By default you can't view with the mouse, Boolean *activate* is set to false. This can be toggled by pressing spacebar. It has to be noted that this method is not written properly, so viewing doesn't work very good, but it is a start.

---

[2] The method is copied from a tutorial on http://www.flashsandy.org/tutorials/3.0/sandy_cs3_tut042.

5. Materials and Textures

In Actionscript files *Basics.as* and *Materials.as* I used a 3d model created in 3D Studio Max 9 and exported to an Actionscript class by using the exporter created by shirotokoro (http://seraf.mediabox.fr/showcase/as3-geom-class-exporter-for-3ds-max-english/). On this website you can find a tutorial on how to set up 3D Studio Max in order to be able to export the model. In my project directory the actionscript classes, *Max.as* and *Maze.as* both contain a 3d object and it is located in the same directory as the application file. Therefore you don't have to import it. Sandy 3.0 contains the following materials[3]:

- BitmapMaterial - Displays a bitmap on the faces of a 3D shape.
- CelShadeMaterial   - Displays the faces of a 3D shape as a Cel Shaded Material (polygon based cel shading).
- ColorMaterial   - Displays a color with on the faces of a 3D shape.
- Material   - ABSTRACT CLASS - base class for all materials.
- MovieMaterial   - Displays a MovieClip on the faces of a 3D shape.
- OutlineMaterial - Displays the outline of a 3D shape in wireframe.
- VideoMaterial   - Displays a Flash video (FLV) on the faces of a 3D shape.
- WireframeMaterial   - Displays the faces of a 3D shape as a wire frame.
- ZShaderMaterial   - Displays a kind of Z shading of any object that this material is applied
       to.

In this paragraph we will review CelShadeMaterial, ColorMaterial, OutlineMaterial, WireFrameMaterial and ZShaderMaterial. The BitmapMaterial will be reviewed in paragraph 5.2 and the MovieMaterial and VideoMaterial in paragraph 5.3.

5.1   Sandy Materials and Attributes

In Chapter 2 was mentioned that these materials can be expanded with a number of attributes. In total there are four attribute types[4]:

- GouraurdAttributes   - Realize a Gouraud shading on a material.
- LightAttributes   - Realize a flat shading effect when associated to a material.
- LineAttributes   - Holds all line attribute data for a material.
- OutlineAttributes   - Holds all outline attributes data for a material.

In the project folder you can find file Materials.as. This file contains three object with different material attributes. By click on them you can see which attributes are used. If you want to stop the animation press spacebar. Sandy 3.0 contains the following materials:

- BitmapMaterial - Displays a bitmap on the faces of a 3D shape.
- CelShadeMaterial   - Displays the faces of a 3D shape as a cel shaded material (polygon based cel shading).
- ColorMaterial   - Displays a color with on the faces of a 3D shape.
- Material   - ABSTRACT CLASS - base class for all materials.
- MovieMaterial   - Displays a MovieClip on the faces of a 3D shape.
- OutlineMaterial - Displays the outline of a 3D shape in wireframe.

---

[3] http://sandy.googlecode.com/svn/trunk/sandy/as3/branches/3.0.1/docs/index.html
[4] http://sandy.googlecode.com/svn/trunk/sandy/as3/branches/3.0.1/docs/index.html

- VideoMaterial   - Displays a Flash video (FLV) on the faces of a 3D shape.
- WireframeMaterial   - Displays the faces of a 3D shape as a wire frame.
- ZShaderMaterial   - Displays a kind of Z shading of any object that this material is applied
                                                to.

The BitmapMaterial will be reviewed in paragraph 5.2 and the MovieMaterial and VideoMaterial in paragraph 5.3

The ColorMaterial is very basic material that is flat shaded and assigns a color to your object. Besides a color number as attribute you can also adjust alpha channel, which controls the transparency of the object.

5.2   Assigning textures

This section describes how to import external textures. For this I used a tutorial created by Petit, http://www.petitpub.com/labs/media/flash/sandy3/materials2.shtml. It a very handy tutorial, even for beginners. All steps taken are explained very deeply. The first step is to load the textures this is done in the method *importTextures()* (see figure 5.1). In this method a Loader variable is created to which the textures are assigned by doing a *URLReques*t.

```
public function importTextures() : void
{
    var importer:Loader = new Loader();
    importer.contentLoaderInfo.addEventListener(Event.COMPLETE, importCompletedMovie);
    importer.load(new URLRequest("../asset/logo.swf"));

    var importer2:Loader = new Loader();
    importer2.contentLoaderInfo.addEventListener(Event.COMPLETE, importCompleted);
    importer2.load(new URLRequest("../asset/vu.png"));

    var importer3:Loader = new Loader();
    importer3.contentLoaderInfo.addEventListener(Event.COMPLETE, importCompleted);
    importer3.load(new URLRequest("../asset/realisticCar.png"));
}
```

Figure 5.1, importTextures

For the variables two methods are called *importCompletedMovie()* and *importCompleted()* (see figure 5.2), which assigns the texture to an object. The same method is called because it can identify the images by their filename, *name = name.substring(name.lastIndexOf("/")+1).split(".")[0]; .*

```
public function importCompleted (event:Event) : void
{
    //var i:Number = 0;
    var target = event.target;
    var name:String = target.url;
    name = name.substring(name.lastIndexOf("/")+1).split(".")[0];

    var texture = target.loader.content.bitmapData;
    var app:Appearance = new Appearance(new BitmapMaterial (texture));

    if (name == "realisticCar")
    {
        for (var i:Number = 2; i <  8; i++)
        {

            box.aPolygons[i].appearance = app;
        }
    }
    else
    {
        plane.appearance = app;
    }


}

public function importCompletedMovie (event:Event) : void
{
    var target = event.target;
    var motor = target.loader.content;
    var movie = new MovieClip();
    movie.addChild(target.loader.content);
    var material = new MovieMaterial (movie);
    box.aPolygons[0].appearance = new Appearance (material);
    box.aPolygons[1].appearance = new Appearance (material);
}
```

Figure 5.2, importCompletedMovie and importCompleted

The variable app is a BitmapMaterial that points to a texture file. Textures can either be assigned to the whole object or to certain polygons (faces of an object). In the method importCompleted I assigned image file, realisticCar.png, to 4 of the 6 polygon. This done checking if the file name is correct and then a for statement assigns the texture to selected polygons. Polygons[0] and polygon[1] have a different texture. In the other method a movie file is received. To assign this to an object we need to create a MovieClip variable, which will be a reference to the moviefile;

> *var movie = new MovieClip();*
> *movie.addChild(target.loader.content);*

5.3   Interacting with the Movie/Video Texture

The MovieMaterial used in section 5.1 can only handle a MovieClip. To create a material that can handle Flash video (flv) files you need to assign them to a VideoMaterial. In the file Movies.as I created a plane which displays different kinds of movies (both swf and flv files). The user can toggle between the movies by clicking on F1 – F5. To assign flv files as a texture you need a different method than the one used in *importCompletedMovie()*. I used the method *zapper()* (see figure 5.3). It

is copied from the tutorial by Petit[5]. The only difference is that it is expanded with toggle function to show the different movie files. Furthermore I created a method toggleFLV to pause the *.flv file which is projected onto the plane. I also tried to create a toggle method for a *.swf file, but I couldn't seem to get it to work.

```actionscript
private function zapper(event:KeyboardEvent) : void
{
    var flvClip:Video = new Video (800, 600);
    var material:VideoMaterial = new VideoMaterial (flvClip, 25);
    var app:Appearance = new Appearance (material);
    plane.appearance = app;

    var nc:NetConnection = new NetConnection();
    nc.connect(null);

    var ns:NetStream = new NetStream(nc);
    flvClip.attachNetStream(ns);

    var listener:Object = new Object();
    listener.onMetaData = function (event:Object) : void ();
    ns.client = listener;
    if (Key.isDown(Keyboard.F1))
    {
        importTextures();
    }
    else if (Key.isDown(Keyboard.F2))
    {
        ns.play("../asset/movingBlocks.flv");
    }
    else if (Key.isDown(Keyboard.F3))
    {
        ns.play("../asset/brazil.flv");
    }
    else if (Key.isDown(Keyboard.F4))
    {
        ns.play("../asset/car.flv");
    }
    else if (Key.isDown(Keyboard.F5))
    {
        ns.play("../asset/franchiseBrazil.flv");
    }
}
```

Figure 5.3, zapper()

[5] *First we create a Video object, on which we will project the video. We pass this to the constructor of a new VideoMaterial, which we use as material in an Appearance, and we set this to be appearance of our everlasting box. We create a NetConnection and call the connect() method on it. Passing null makes the connection work locally. On the connection we then create a NetStream, which we attach to the display object, our Video.*

The NetStream needs a client, that listens for meta data events, such as specific flags in the stream, called cue points. The listener also needs a meta data handler. To make it simple, it does nothing in this case. A disclaimer is prudent here - the set up of the connection and stream is done as simply as possible. If we want the application to be robust, we need to catch possible errors and do something about them. If one day you have the time and urge to become experts on video in Flash, there are lots web sites and books to read.

We tell the net stream to start playing our FLV as soon as the local buffer is full. We are at the end of the tutorial document, so we don't want the video to start playing immediately and reach the end, before our reader gets to see it, so we call the pause method. The user will start and pause the video by clicking on it, so we reuse the toggleMovie handler from the MovieMaterial application above.

6. Sprites

6.1   Sprite2D and Sprite3D

Sprites are two-dimensional images or animations. Sandy 3.0 support two kinds, Sprite2D and Sprite3D. Sprite2D are normal sprites what we are used to. In other words the image stays the same no matter the orientation of the camera.

A Sprite3D is actually a MovieCip containing a maximum of 360 frames. If the camera moves a different frame will be shown, thus mimicking a 3D effect. In this example I used a 3D package (Maya 8.5) to create two models (see figure 6.1) and render out the different frames. When you have all the frames you

Figure 6.1, 3D objects

must create a MovieClip out of them. For this you have different options, you can either use only Adobe Flash or a combination of a video editing program like Adobe After Effects or Adobe Premiere and Flash. To create a MovieClip start Adobe Flash and start a new *.fla scene. Import the frames images or video file into the library and export it as a MovieClip (File>Export>Export Movie). Remember to add the Sandy 3.0 library to Flash. Go to http://www.flashsandy.org/tutorials/3.0/installing_sandy_flash_cs3 for a tutorial on how to set Sandy for Flash CS3.

6.2   Flash and Sandy

In contrast to the other examples I used Adobe Flash CS3 instead of Adobe Flex 2 for the creation of the Sprite3D scenes. For some odd reason I couldn't get the file to properly load in Flex. Whereas the same file did properly load in Flash. I posted the problem on a Sandy forum and am awaiting an answer.

The folder *SpritesProjects* contain all project files used in this chapter.
- Carsprite.as & Sprites.as   - Actionscript logic
- Carsprite.fla & Sprites.fla   - Flash scenes
- Carsprite.swf & Sprites.swf   - Flash movies
- Assets>car.swf & Skull.swf   - 360° MovieClip

For the most part the actionscript files are the same as the ones used in Flex. Only differences are in the methods *Sprites(), Carsprites()* and *createSceene()*. To use the 360° MovieClip files in the Flash scene, they have to be uploaded. The class LoaderQueue does this for use. First a new LoaderQueue

object is created. Then *add( )* creates a new request for the object. When the *add( )* action is completed the method *init( )* is called, which creates the scene. And finally the *start( )* loads all resources in the queue (see figure 6.2). In the method creatScene() a new Sprite3D object is created which will reference to 360° MovieClip. Now the MovieClip(Sprite3D object) has the same features as normal Sandy primitive.

```
public function Sprites():void {
    queue = new LoaderQueue();
    queue.add( "skull", new URLRequest("assets/Skull.swf"));
    queue.addEventListener(SandyEvent.QUEUE_COMPLETE, init);
    queue.start();
}

    s = new Sprite3D("skull",queue.data["skull"],2);
```

Figure 6.2, LoaderQueue and Sprite3D

In the project Sprites I used a skull with which can be interacted by pressing the left and right cursor or by moving the mouse over the object. In the other project I tried to mimic a driving car, but that didn't go that well. When you press forward or backward (cursor-up and –down) the car moves over the wrong axis, On the other hand turning the car did work.
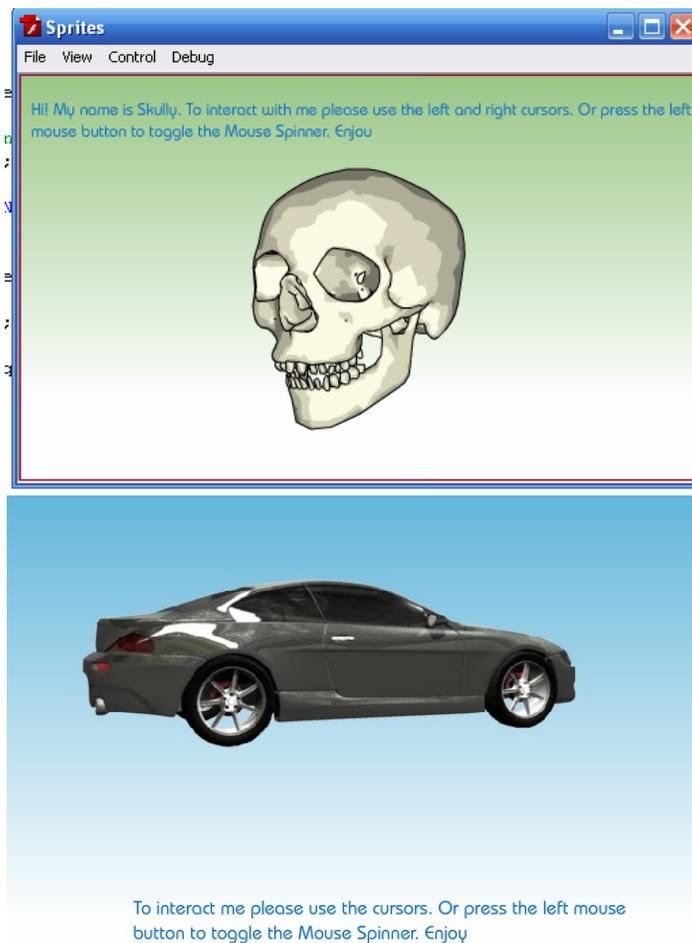

Figure 6.3, Sprites.swf and Carsprite.swf

7. Conclusion

For me Sandy 3D gave web development a whole new way to look at it. Especially since my focus used to be only on 3D computer graphics. By providing users with these 3D libraries a lot of nice possibilities come to hand. Certainly if it is combined with Flex. In this paper I only used Flex for writing and compiling the Actionscript codes. For future studies I hope to combine Flex components to the Sandy Actionscipt files to created a better feeling of interactivity and more functional "webapps". During studying Sandy I also found out that there are a lot more extensions to Actionscript and thus Sandy, such as animation libraries (Tweener) and physics libraries (Ape). So the next step is to have a look at these libraries to see what can be created in web3D.

8. Bibliography

http://sandy.googlecode.com/svn/trunk/sandy/as3/trunk/src/

http://tortoisesvn.net/

http://www.flashsandy.org/tutorials/3.0/

http://www.petitpub.com/labs/media/flash/sandy3/

http://sandy.googlecode.com/svn/trunk/sandy/as3/branches/3.0.1/docs/index.html

http://www.kirupa.com/forum/showpost.php?p=2098269&postcount=319

http://www.senocular.com/flash/actionscript.php

http://www.flash-creations.com/notes/asclass_key.php

http://seraf.mediabox.fr/showcase/as3-geom-class-exporter-for-3ds-max-english/

http://www.petitpub.com/labs/media/flash/sandy3/materials2.shtml

http://www.flashsandy.org/tutorials/3.0/installing_sandy_flash_cs3