8
External Models

In the previous chapter, the Utah teapot was Introduced to demo several shaders. Of course, we did not create the teapot by manually defining each triangle to form the shape of a teapot. Papervision3D allows us to work with models that are created and exported by a **3D modeling program**, which adds many possibilities, especially when the shape of the default primitives doesn't fit your needs.

With Papervision3D we can load several types of 3D models. Some of them support animations that have been created in the 3D modeling program. There are many programs that support the model formats Papervision3D reads, more than what we can cover in this book. We will have a look at the following three:

- **Autodesk 3ds Max**: One of the widely-known commercial 3D programs, which runs only on Windows
- **Sketchup**: A free 3D modeling program provided by Google. Designed to be a more intuitive 3D program, which runs on Max OS X and Windows
- **Blender**: An open source, platform-independent 3D modeling tool

The main focus of this chapter will be on how to get models from these programs into Papervision3D. The process of creating models in general is too program-specific and out-of-scope for this book. Therefore, only the creation of a simple 3D object per program will be discussed. However, some more complex preconfigured models are also provided.

This chapter covers the following:

- Modeling for Papervision3D
- Preparing for loading models
- Creating and loading models using Autodesk 3ds Max
- Loading an animation from Autodesk 3ds Max

- Creating and loading models using SketchUp
- Creating and loading models using Blender
- Controlling loaded materials

Let's start off by having a look at some general practices to keep in mind when modeling for Papervision3D.

Modeling for Papervision3D

As mentioned in the introduction, this chapter will not describe in detail how to create models. Each tool works differently and requires a different approach for creating suitable models.

In this section, we will discuss several techniques that relate to modeling for Papervision3D. As Papervision3D is commonly used for web-based projects, modeling requires a different mindset than modeling for an animated movie, visualization, or game. Most of the techniques discussed relate to improving performance. This section is especially useful for modelers who need to create models for Papervision3D.

Papervision3D Previewer

Papervision3D Previewer is a small program that should be part of every modeller's toolbox. This tool comes in handy for testing purposes. It allows a modeler to render an exported model in Papervision3D, and it displays some statistics that show how the model performs. At the time of writing, this tool was not compatible with Papervision3D 2.1, which could result in small problems when loading external models.

Papervision3D Previewer can be downloaded from `http://code.google.com/p/mrdoob/wiki/pv3dpreviewer`

Keep your polygon count low

Papervision3D is a cutting edge technology that brings 3D to the Flash Player. It does this at an amazing speed relative to the capabilities of the Flash player. However, performance of Papervision3D is just a fraction of the performance that can be achieved with hardware-accelerated engines such as used by console games. Even with hardware-accelerated games there is a limit to the number of polygons that can be rendered, meaning there is always a compromise between detail and performance. This counts even more for Papervision3D, so always try to model using as few polygons as possible.

Papervision3D users often wonder what the maximum number of triangles is that the Flash player can handle. There is no generic answer to this question, as performance depends on more factors than just the number of triangles. On average, the total triangle count should be no more than 3000, which equals 1500 polygons (remember that one polygon is made of two triangles).

Unlike most 3D modeling programs, Papervision3D is triangle based and not polygon based.

Add polygons to resolve artifacts

Although this seems to contradict the previous suggestion to keep your polygon count low, sometimes you need more polygons to get rid of texture distortion or to reduce z-sorting artifacts. z-sorting artifacts will often occur in areas where objects intersect or closely intersect each other. Subdividing polygons in those areas can make z-sorting more accurate. Often this needs to be done by creating new polygons for the intersecting triangles of approximately the same size.

There are several approaches to prevent z-sorting problems. The next chapter will discuss these in detail. Depending on the object you're using, it can be very time consuming to tweak and find the optimal amount and location of polygons. The amount of polygons you add in order to solve the problem should still be kept as low as possible. Finding the optimal values for your model will often result in switching a lot between Papervision3D and the 3D modeling program.

Keep your textures small

Textures used in the 3D modeling tool can be exported along with the model to a format that is readable for Papervision3D. This is a valuable feature as the texture will automatically be loaded by Papervision3D. However, the image, which was defined in the 3D authoring tool, will be used exactly as provided by Papervision3D. If you choose a 1024 by 1024 pixels image as the texture, for example the wheels of a car, Papervision3D loads the entire image and draws it on the wheel of a car that appears on screen at a size of 50 by 50 pixels for example. There are several problems related to this:

- It's a waste of bandwidth to load such a large image. Loading any image takes time, which should be kept as short as possible.
- It's a waste of capacity. Papervision3D needs to resize the image from 1024 by 1024 pixels to an image, which will be, for example, maximal 50 by 50 pixels on screen.

Always choose texture dimensions that make sense for the application using it, and keep in mind that they have to be power of two. This will enable mipmapping and smoothing, which come without extra performance costs.

Use textures that Flash can read

3D modeling programs usually read a variety of image sources. Some even support reading Adobe Photoshop's native file-format PSD. Flash can load only GIF, JPG, or PNG files at run time. Therefore, stick to these formats in your model so that you do not have to convert the textures when the model needs to be exported to Papervision3D.

Use UV maps

If your model is made up of several objects and textures, it's a good idea to use **UV mapping**, which is the process of unwrapping the model and defining all its textures into one single image. This way we can speed up initial loading of an application by making one request from Flash to load this image instead of loading dozens of images. UV mapping can also be used to tile or reuse parts of the image. The more parts of the UV-mapped image you can reuse, the more bandwidth you'll save. Always try to keep your UV-mapped image as small as possible, just as with keeping your normal textures small. In case you have a lot of objects sharing the same UV map and you need a large canvas to unwrap the UV map, be aware of the fact that the maximum image size supported by Flash Player 9 is 2880x2880 pixels. With the benefits of power of two textures in mind, the maximum width and height is 2048x2048 pixels.

Baking textures

Baking textures is the process of integrating shadows, lighting, reflection, or entire 3D objects into a single image. Most 3D modeling tools support this.

This contradicts what has been said about tiling images in UV maps, as baking results in images that usually can only be used once because of the baked information on the texture. However, it can increase the level of realism of your application, just like shading does, but without the loss of performance caused by calculating shading in real time. Never use them in combination with a tiling image, as repeated shading, for instance, will result in unnatural looking renders. Therefore, each texture needs to be unique, which will cause longer loading times before you can show a scene.

Use recognizable names for objects and materials

It is always a good convention to use recognizable names for all your objects. This counts for the classes, methods, and properties in your code, and also for the names of the 3D objects in your modeling tool.

Always think twice before renaming an object that is used by an application. The application might use the name of an object as the identifier to do something with it—for example, making it clickable. When working in a team of modelers and programmers, you really need to make this clear to the modelers as changing the name of an object can easily break your application.

Size and positioning

Maintaining the same relative size for your modeled objects, as you would use for instantiating primitives in your scene, is a good convention. Although you could always adjust the scale property of a loaded 3D model, it is very convenient when both Papervision3D and your modeling tool use the same scale.

Remember that Papervision3D doesn't have a metric system defining units of a certain value such as meters, yards, pixels, and so on. It just uses units.

Another convention is to position your object or objects at the origin of the 3D space in the modeling tool. Especially when exporting a single object from a 3D modeling tool, it is really helpful if it is located at a position of 0 on all axes. This way you can position the 3D object in Papervision3D by using absolute values, without needing to take the offset into account. You can compare this with adding movie clips to your library in Flash. In most cases, it is pretty useful when the elements of a movie clip are centered on their registration point.

Finding the balance between quality and performance

For each project you should try to find the balance between lightweight modeling and quality. Because each project is different in requirements, scale, and quality, there is no rule that applies for all. Keep the tips mentioned in the previous sections in mind and try to be creative with them. If you see a way to optimize your model, then do not hesitate to use it.

Before we have a look at how to create and export models for Papervision3D, we will create a basic application for this purpose.

Creating a template class to load models

In order to show an imported 3D model using Papervision3D, we will create a basic application. Remember the basic mouse interaction example we created in Chapter 4? We created a grid of planes together with a camera that was orbiting around the grid with an eased animation. This is a good starting point. We can use the same class to rotate around the models, which we are going to create and load later in this chapter.

Based on that class we can create the following new class. Each time we load a new model we just have to alter the `init()` method. First, have a look at the following base code for this application:

```
package {

import flash.events.Event;

import org.papervision3d.materials.WireframeMaterial;
import org.papervision3d.materials.utils.MaterialsList;
import org.papervision3d.objects.DisplayObject3D;
import org.papervision3d.objects.primitives.Cube;
import org.papervision3d.view.BasicView;

   public class ExternalModelsExample extends BasicView
   {
          private var model:DisplayObject3D;
          private var rotX:Number = 0.1;
          private var rotY:Number = 0.1;
          private var camPitch:Number = 90;
          private var camYaw:Number = 270;
          private var easeOut:Number = 0.1;

          public function ExternalModelsExample()
          {
                 stage.frameRate = 40;

                 init();
                 startRendering();
          }

          private function init ():void
          {
                 model = new Plane();
                 scene.addChild(model);
          }
```

```
        private function modelLoaded(e:FileLoadEvent):void
        {
                //To be added
        }

            override protected function onRenderTick(e:Event=null):void
            {
                var xDist:Number = mouseX - stage.stageWidth * 0.5;
                var yDist:Number = mouseY - stage.stageHeight * 0.5;

                camPitch += ((yDist * rotX) - camPitch + 90)
                            * easeOut;
                camYaw += ((xDist * rotY) - camYaw + 270) * easeOut;

                camera.orbit(camPitch, camYaw);

                super.onRenderTick();
            }

    }
}
```

We have created a new plane using a wireframe as its material. The plane is assigned to a class property named `model`, which is of the `DisplayObject3D` type. In fact, any external model is a do3D. No matter what type of model we load in the following examples, we can always assign it to the `model` property. The classes that we'll use for loading 3D models all inherit from `DisplayObject3D`.

Now that we have created a default application, we are ready to create our first model in 3D Studio Max, export it, and then import it into Papervison3D.

Creating models in Autodesk 3ds Max and loading them into Papervision3D

Autodesk 3ds Max (also known as **3D Studio Max** or **3ds Max**) is one of the widely-known commercial 3D modeling and animation programs. This is a good authoring tool to start with, as it can save to two of the file formats Papervision3D can handle. These are:

- **COLLADA** (extension `*.dae`): An open source 3D file type, which is supported by Papervision3D. This is the most advanced format and has been supported since Papervision3D's first release. It also supports animations and is actually just a plain text XML file.
- **3D Studio** (extension `*.3ds`): As the name suggests, this is one of the formats that 3ds Max natively supports. Generally speaking it is also one of the most common formats to save 3D models in.

As of 3ds Max version 9, there is a built-in exporter plugin available that supports exporting to COLLADA. However, you should avoid using this, as at the time of writing, the models it exports are not suitable for Papervision3D.

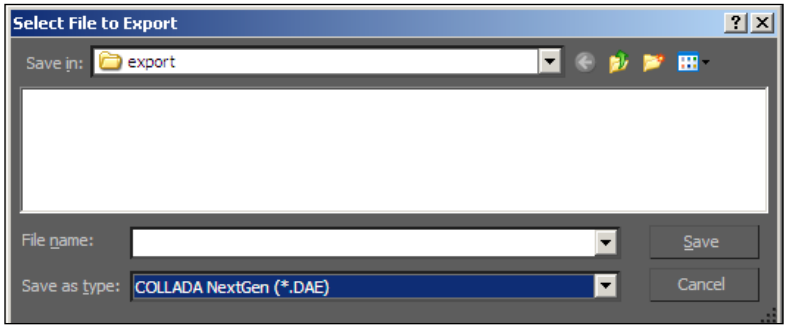
Don't have a license of 3ds Max and want to get along with the following examples? Go to `www.autodesk.com` to download a 30-day trial.

Installing COLLADA Max

An exporter that does support COLLADA files suitable for Papervision3D is called **COLLADA Max**. This is a free and open source exporter that works with all versions of 3ds Max 7 and higher.

Installing this exporter is very easy. Just follow the steps mentioned next:

1. Make sure you have installed 3ds Max version 7 or higher.
2. Go to `http://sourceforge.net/projects/colladamaya/`.
3. Click on **View all files** and select the latest COLLADA Max version. (At the time of writing this is **COLLADA Max NextGen 0.9.5**, which is still in beta, but is the only version that works with 3ds Max 2010).
4. Save the download somewhere on your computer.
5. Run the installer.
6. Click **Next**, until the installer confirms that the exporter is installed.
7. Start 3ds Max and double check if we can export using the **COLLADA** or **COLLADA NextGen** filetype, as shown in the following screenshot:



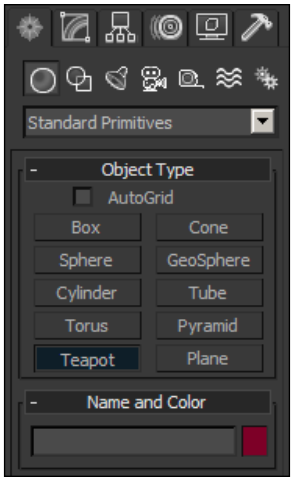
If the only COLLADA export option is **Autodesk Collada**, then something went wrong during the installation of COLLADA Max, as this is not the exporter that works with Papervision3D.

Now that 3ds Max is configured correctly for exporting a file format that can be read by Papervision3D, we will have a look at how to create a basic textured model in 3ds Max and export it to Papervision3D.

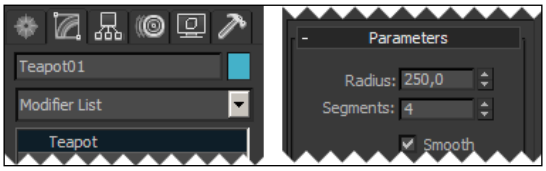
Creating the Utah teapot and export it for Papervision3D

If you already know how to work with 3ds Max, this step is quite easy. All we need to do is create the Utah teapot, add UV mapping, add a material to it, and export it as COLLADA. However, if you are new to 3ds Max, the following steps needs to be clarified.

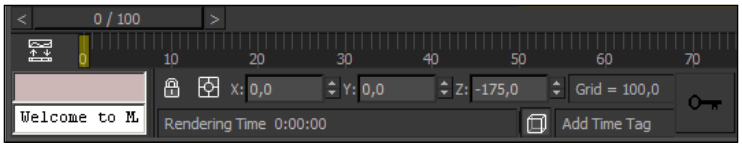
1. First, we start 3ds Max and create a new scene. The creation of a new scene happens by default on startup.
2. The Utah teapot is one of the objects that comes as a standard primitive in 3ds Max. This means you can select it from the default primitives menu and draw it in one of the viewports. Draw it in the top viewport so that the teapot will not appear rotated over one of its axes.



3. Give it a **Radius** of **250** in the properties panel on the right, in order to make it match with the units that we'll use in Papervision3D.



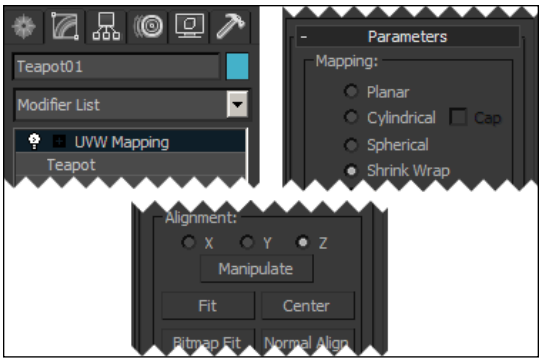
4. Position the teapot at the origin of the scene. You can do this by selecting it and changing the **x**, **y**, and **z** properties on the bottom of your screen. You would expect that you need to set all axes to 0, although this is not the case. In this respect, the teapot differs from other primitives in 3ds Max, as the pivot point is located at the bottom of the teapot. Therefore, we need to define a different value for the teapot on the z-axis. Setting it to approximately **-175** is a good value.



5. To map a material to the teapot, we need to define a UV map first.

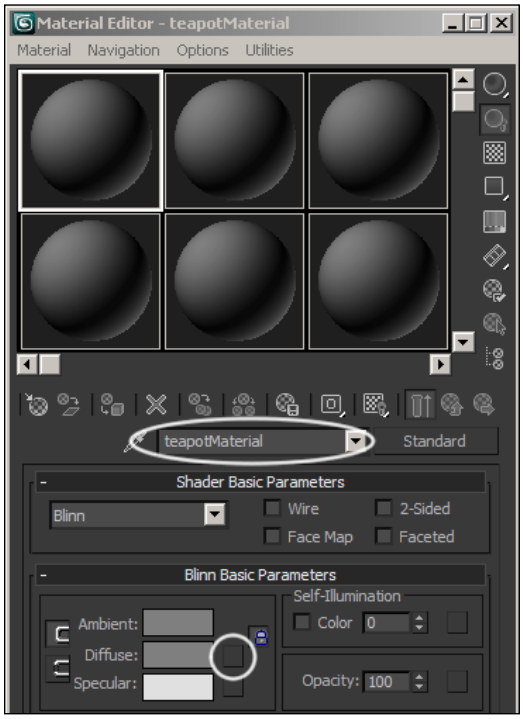
UV mapping is also known as **UVW mapping**. Some call it UV mapping and others call it UVW mapping. 3ds Max uses the term UVW mapping.

6. While having the teapot still selected, go to modify and then select **UVW Mapping** from the modifier list. Select **Shrink Wrap** and click **Fit** in the **Alignment** section. This will create a spherical UVW map for us.



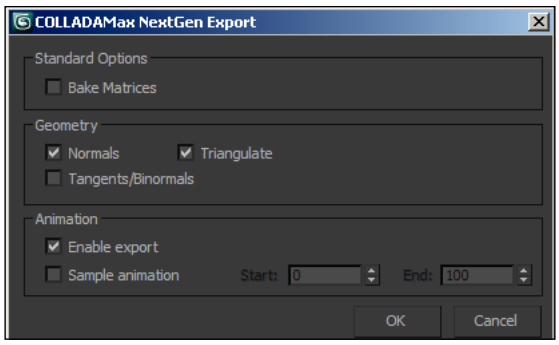
7. Open the material editor using keyboard shortcut *m*. Here we define the materials that we use in 3ds Max.
8. Give the new material a name. Replace **01 – Default** with a material name of your choice—for example, **teapotMaterial**.

9. Provide a bitmap as the diffuse material. You can do this by clicking on the square button, at the right of the **Diffuse** value within **Blinn Basic Parameters** section.



10. A new window called **Material/Map Browser** will open. Double-click **Bitmap** to load an external image. Select an image of your choice. We will use `teapotMaterial.jpg`, which has been provided as a file in the folder of this chapter.
11. The material editor will now update and show the selected material on an illustrative sphere. This is your newly-created material, which you need to drag on the created teapot.
12. The teapot model can now be exported. Depending on the version of the installed COLLADA exporter, select **COLLADA** or **COLLADA NextGen**. Note that you should not export using **Autodesk Collada**, as this exporter doesn't work properly for Papervision3D.
13. Give it a filename of your choice, for example **teapot**, and hit **Save**.

14. The exporter window will pop up. The default settings are fine for exporting to Papervision3D, so click **OK** to save the file.



Save the model in the default 3ds Max file format (`.max`) somewhere on your local disk, so we can use it later when discussing other ways to export this model to Papervision3D.

The model that we have created and exported is now ready to be imported by Papervision3D. Let's take a look at how this works.

Importing the Utah teapot into Papervision3D

To work with the just exported Utah teapot, we will use the `ExternalModelsExample` project that we created previously in this chapter.

Browse to the folder inside your project where you have saved your document class. Create a new folder called `assets` and copy to this folder, the created COLLADA file along with the image used as the material of the teapot.

The class used to load an external COLLADA file is called `DAE`, so let's import it.

```
import org.papervision3d.objects.parsers.DAE;
```

This type of class is also known as a parser, as it parses the model from a loaded file.

When you have a closer look at the source files of Papervision3D and its model parsers, you will probably find out about the `Collada` class. This might be a little confusing as we use the `DAE` parser to load a COLLADA file and we do not use the `Collada` parser. Although you could use either, this book uses the `DAE` parser exclusively, as it is a more recent class, supporting more features such as animation. There is no feature that is supported by the `Collada` parser, and is not supported by the DAE parser.

Replace all code inside the `init()` method with the following code that loads a COLLADA file:

```
model = new DAE();
model.addEventListener(FileLoadEvent.LOAD_COMPLETE,modelLoaded);
DAE(model).load("assets/teapot.DAE");
```

Because `model` is defined as a `DisplayObject3D` class type, we need to cast it to `DAE` to make use of its methods so that we can call the `load()` method.

An event listener is defined, waiting for the model to be completely loaded and parsed. Once it is loaded, the `modelLoaded()` method will be triggered. It is a good convention to add models only to the scene once the model is completely loaded. Add the following line of code to the `modelLoaded()` method:

```
scene.addChild(model);
```

{Ex.} COLLADA Utah Teapot Example {Ex.}

Publishing this code will result in the teapot with the texture as created in 3ds Max.

In real-world applications it is good practice to keep your models in one folder and your textures in another. You might want to organize the files similar to the following structure:

- Models in `/assets/models/`
- Textures in `/assets/textures/`

By default, textures are loaded from the same folder as the model is loaded from, or optionally from the location as specified in the COLLADA file. To include the `/assets/textures/` folder we can add a file search path, which defines to have a look in the specified folder, to see if the file is located there, in case none can be found on the default paths. This can be defined like:

```
daeModel.addFileSearchPath("assets/textures");
```

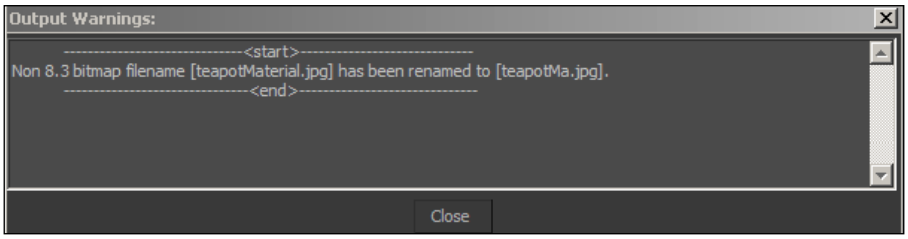
You can call this method multiple times, in order to have multiple folders defined. Internally, in Papervision3D, it will loop through an array of file paths.

Exporting and importing the Utah teapot in 3ds format

Now that we have seen how to get an object from 3ds Max into a Papervision3D project, we have a look at another format that is supported by both 3ds Max and Papervision3D. This format is called **3D Studio**, using a **3ds** extension. It is one of the established 3D file formats that are supported by most 3D modeling tools.

Exporting and importing is in fact very similar to COLLADA. Let's first export the file to the 3D Studio format.

1. Open the Utah teapot, which we've modeled earlier in this chapter.
2. Leave the model as it is, and go straight to export. This time we select **3D Studio (*.3DS)** as the file type. Save it into your project folder and name it **teapot**.
3. Click **OK** when asked whether to preserve Max's texture coordinates.
4. If your model uses `teapotMaterial.jpg` from the book downloads as material, or an image with more than eight characters in its filename, the exporter will output a warning.



   You can close this warning, but you need to be aware of the output message. It says that the bitmap filename is a non-8.3 filename, that is, a maximum amount of 8 characters for the filename and a 3-character extension. The 3D Studio file is an old format, released at the time when there was a DOS version of 3ds Max. Back then it was an OS naming convention to use short filenames, known as 8.3 filenames. This convention still applies to the 3D Studio format, for the sake of backward compatibility. Therefore, the reference to the bitmap has been renamed inside the exported 3D Studio file.
5. Because the exported 3D Studio file changed only the reference to the bitmap filename internally and it did not affect the file it refers to, we need to create a file using this renamed file reference. Otherwise, it won't be able to find the image. In this case we need to create a version of the image called `teapotMa.jpg`. Save this file in the same folder as the exported 3D Studio file.

As you can see, it is very easy to export a model from 3ds Max to a format Papervision3D can read. Modeling the 3D object is definitely the hardest and most time consuming part, simply because creating models takes a lot of time. Loading the model into Papervision3D is just as easy as exporting it.

First, copy the 3D Studio file plus the renamed image to the `assets` folder of your project. We can then alter the document class in order to load the 3ds file. The class that is used to parse a 3D Studio file is called `Max3DS` and needs to be imported.

```
import org.papervision3d.objects.parsers.Max3DS;
```

In the `init()` method you should replace or comment the code that loads the COLLADA model from our previous example, with the following:

```
model = new Max3DS();
model.addEventListener(FileLoadEvent.LOAD_COMPLETE,modelLoaded);
Max3DS(model).load("assets/teapot.3ds", null, "./assets/");
```

As the first parameter of the `load` method, we pass a file reference to the model we want to load. The second parameter defines a materials list, which we will not use for this example. The third and final parameter defines the texture folder. This folder is relative to the location of the published SWF. Note that this works slightly different than the DAE parser, which loads referenced images from the path relative to the folder in which the COLLADA file is located or loads images as specified by the `addFileSearchPath()` method.

{Ex.} Max 3ds Utah Teapot Example {Ex.}

Publish the code and you'll see the same teapot. However, this time it's using the 3D Studio file format as its source.

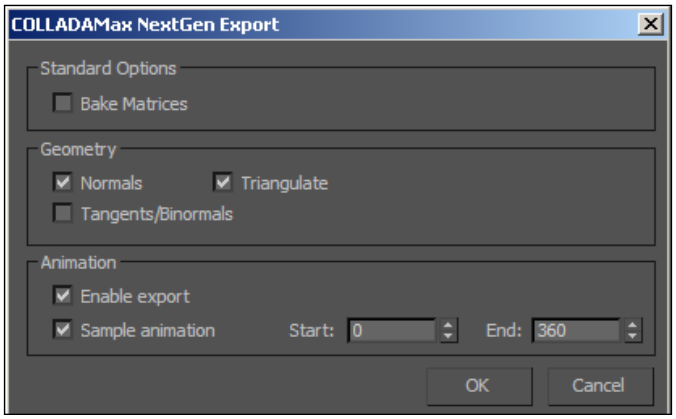
Importing animated models

The teapot is a static model that we exported from a 3D program and loaded into Papervision3D. It is also possible to load **animated models**, which contain one or multiple animations. 3ds Max is one of the programs in which you can create an animation for use in Papervision3D. Animating doesn't require any additional steps. You can just create the animation and export it. This also goes for other modeling tools that support exporting animations to COLLADA.

For the sake of simplicity, this example will make use of a model that is already animated in 3ds Max. The model contains two animations, which together make up one long animation on a shared timeline. We will export this model and its animation to COLLADA, load it into Papervision3D, and play the two animations.

1. Open **animatedMill.max** in 3ds Max. This file can be found in the book downloads.
2. You can see the animation of the model directly in 3ds Max by clicking the play button in the menu at the bottom right corner. This will animate the blades of the mill. The first 180 frames animate the blades from left to right. Frames 181 to 360 animate the blades from right to left.

3. As the model is already animated, we can go ahead with exporting, without making any changes to the model. Export it using the COLLADA filetype and save it somewhere on your computer.
4. When the COLLADA Max exporter settings window pops up, we need to check the **Sample animation** checkbox. By default **Start** and **End** are set to the length of the timeline as it is defined in 3ds Max. In case you just want to export a part of it, you can define the start and end frames you want to export. For this example we leave them as they are: **0** and **360**.



By completing these steps you have successfully exported an animation in the COLLADA format for Papervision3D. Now, have a look at how we can load the animated model into Papervision3D.

First, you need to copy the exported COLLADA and the applied material—`Blades.jpg`, `House.jpg`, and `Stand.jpg`—to the `assets` folder of your project.

To load an animated COLLADA, we can use the DAE class again. We only need to define some parameters at instantiation, so the animation will loop.

```
model = new DAE(true,null,true);
model.addEventListener(FileLoadEvent.LOAD_COMPLETE,modelLoaded);
DAE(model).load("assets/animatedMill.dae");
```

Take a look at what these parameters stand for.

	Parameter	Data type	Default value	Description
1	`autoPlay`	`Boolean`	`true`	When an animation is found in the COLLADA file and this value is set to `true`, Papervision3D will automatically play the animation. If a COLLADA contains multiple animations it will play all of them.
2	`name`	`String`	null	An optional name, which can be handy to find a reference to the model.
3	`loop`	`Boolean`	`false`	Defines whether or not to loop the animation that is found in the COLLADA file. When set to `false` it will play the animation only once.

Publish the project in order to see the mill with animated blades.

Animation clips

Once the project is published, you will notice that it is playing the two animations—left rotating blades and right rotating blades. This is because we have not defined the animation as two separate animation clips. Right now we just play the entire animation. In order to choose in which direction the blades rotate, we have to define the **animation clips** first. This is nothing more than splitting one long animation into two or more animations that make part of one animation timeline.

The `AnimationClip3D` class takes care of defining an animation clip. The following table lists its parameters at instantiation:

	Parameter	Data type	Default value	Description
1	`name`	`String`	—	A unique name for the animation clip. This is used as a reference to play an animation.
2	`startTime`	`Number`	0.0	Defines the start time of an animation. It should not be higher than the end time of the total animations. The total time of an animation can be found at the `daeModel.animation.endTime` animation property of a DAE instance.
3	`endTime`	`Number`	0.0	Defines the end time of an animation. It should not be higher than the end time of all animations.

Exported animations are time based instead of frame based as in 3ds Max. All animations in 3ds max should be kept at 30 frames per second.

The `AnimationClip3D` class is imported as follows:

```
import org.papervision3d.core.animation.clip.AnimationClip3D;
```

In this example we want to define two animations—animation right and animation left. The right rotating animation is defined from frame 0 to frame 180, which is half the total animation length. The entire animation is 360 frames at 30 frames per second, which is 360 / 30 = 12 seconds. Therefore, the end time of our right rotating animation will be 6. This will be also the start time of the left rotating blades animation.

Defining the animation clips can be done only when the model is completely loaded, so let's add the following code to the `loadComplete()` method:

```
var animationRight:AnimationClip3D = new AnimationClip3D
                                     ("right",0,6);
var animationLeft:AnimationClip3D = new AnimationClip3D("left",6,12);
```

This defines the time span of the two animation clips, but it still doesn't have any link with the animation in the loaded model. The animation clips need to be registered as playable animation clips on the `DAE` instance:

```
DAE(model).animation.addClip(animationRight);
DAE(model).animation.addClip(animationLeft);
```

Now the animation clips are defined and ready to be played. Playing an animation makes use of the name that is provided as the first parameter at instantiation of the animation clip. For example, when we want to play the right rotating animation we can call the `play` method on a `DAE` instance.

```
DAE(model).play("right");
```

When you want to stop your animation, you can call:

```
DAE(model).stop();
```

 Animated COLLADA Mill Example 

That's easy, isn't it? The most difficult part is definitely making an animated model. There are a lot of online tutorials available that explain how to animate in 3ds Max.

It is important to note that we've exported our model in the COLLADA format. At the time of writing, it is not possible to load animations using the 3D Studio format. COLLADA and **MD2** (which will not be covered by this book) are the only formats for which Papervision3D supports animation.

Creating and loading models using SketchUp

SketchUp is a free 3D modeling program, which was acquired by Google in 2006. It has been designed as a 3D modeling program that's easier to use than other 3D modelling programs. A key to its success is its easy learning curve compared to other 3D tools.

Mac OS X, Windows XP, and Windows Vista operating systems are supported by this program that can be downloaded from `http://sketchup.google.com/`. Although there is a commercial SketchUp Pro version available, the free version works fine in conjunction with Papervision3D.

An interesting feature for non-3D modelers is the integration with Google's 3D Warehouse. This makes it possible to search for models that have been contributed by other SketchUp users. These models are free of any rights and can be used in commercial (Papervision3D) projects.

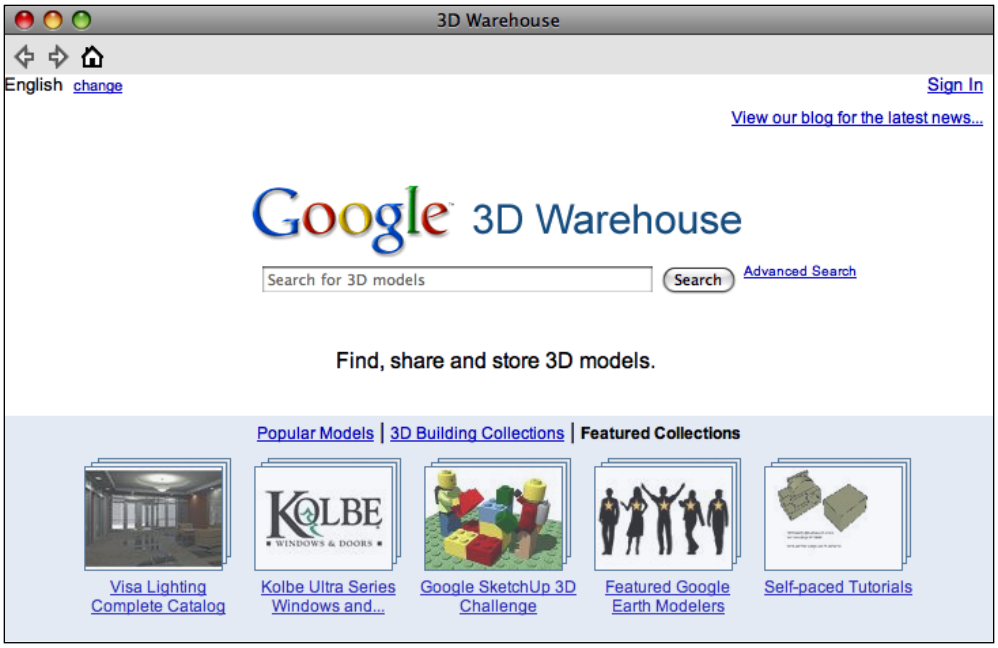
Exporting a model from Google's 3D Warehouse for Papervision3D

There are several ways to load a model, coming from Google 3D Warehouse, into Papervision3D. One of them is by downloading a SketchUp file and exporting it to a format Papervision3D works with. This approach will be explained.

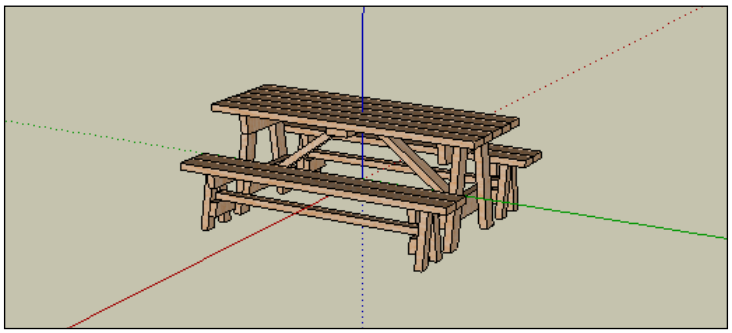
The strength of Google 3D Warehouse is also its weakness. Anybody with a Google account can add models to the warehouse. This results in a variety of quality of the models. Some are very optimized and work fluently, whereas others reveal problems when you try to make them work in Papervision3D. Or they may not work at all, as they're made of too many polygons to run in Papervision3D. Take this into account while searching for a model in the 3D warehouse.

For our example we're going to export a picnic table that was found on Google 3D Warehouse.

1. Start Sketch Up.
2. Choose a template when prompted. This example uses **Simple Template – Meters**, although there shouldn't be a problem with using one of the other templates.
3. Go to **File | 3D Warehouse | Get models** to open 3D Warehouse inside SketchUp.



4. Enter a keyword to search for. In this case that will be **picnic table**.
5. Select a model of your choice. Keep in mind that it has to be low poly, which is something you usually find out by trial and error.
6. Click on **Download Model**, to import the model into SketchUp and click **OK** when asked if you want to load the model directly into your Google SketchUp model.
7. Place the model at the origin of the scene. To follow these steps, it doesn't have to be the exact origin, approximately is good enough.
8. By default, a 2D character called **Sang** appears in the scene, which you do not necessarily have to remove; it will be ignored during export.
9. Because the search returned a lot of picnic tables varying in quality, there is a SketchUp file provided with this book. This file has a picnic table already placed on the origin. Of course, you could also choose another picnic table, or any other object of your choice.



10. Leave the model as it is and export it. Go to **File | Export | 3D Model**. Export it using the **Google Earth** (`*.kmz`) format and save it in your `assets` folder.

The file format we're exporting to originally was meant to display 3D objects in Google Earth. The file ends with `.kmz` as its extension, and is actually a ZIP archive that contains a COLLADA file and the texture images. In the early days of Papervision3D, it was a common trick to create a model using SketchUp and then get the COLLADA file out of the exported Google Earth file, as the Google Earth KMZ file format wasn't still supported then.

Importing a Google Earth model into Papervision3D

Now that we have successfully exported a model from SketchUp, we will import it into Papervision3D. This doesn't really differ from loading a COLLADA or 3D Studio file.

The class we use for parsing the created `PicnicTable.kmz` file is called `KMZ` and can be found in the `parsers` package. Add the following line to the import section of your document class:

```
import org.papervision3d.objects.parsers.KMZ;
```

Replace or comment the code that loads the animated COLLADA model and defines the animations from the previous example. In the `init()` method we can then instantiate the `KMZ` class, assign it to the `model` class property, and load the KMZ file. Make sure you have saved `PicnicTable.kmz` file into the `assets` folder of your project.

```
model = new KMZ();
model.addEventListener(FileLoadEvent.LOAD_COMPLETE,modelLoaded);
KMZ(model).load("assets/PicnicTable.kmz");
```

{Ex.} SketchUp Picnic Table Example {Ex.}

That looks familiar, right? Now let's publish the project and your model should appear on the screen.

Notice that in many cases, the downloaded and exported model from Google 3D Warehouse might appear very small on your screen in Papervision3D. This is because they are modeled with other metric units than we use in Papervision3D. Our example application places the camera at a 1000 units away from the origin of the scene. Many 3D Warehouse models are made using units that define meters or feet, which makes sense if you were to translate them to real-world units. When a model is, for example, 1 meter wide in SketchUp, this equals to 1 unit in Papervision3D. As you can imagine, a 1 unit wide object in Papervision3D will barely be visible when placing the camera at a distance of 1000. To solve this you could use one of the following options:

- Use other units in Papervision3D and place your camera at a distance of 5 instead of 1000. Usually you can do this at the beginning of your project, but not while the project is already in progress, as this might involve a lot of changes in your code due to other objects, animations, and calculations that are made with a different scale.
- Scale your model inside SketchUp to a value that matches the units as you use them in Papervision3D. When the first option can't be realized, this option is recommended.
- Scale the loaded model in Papervision3D by changing the scale property of your model.

```
model.scale = 20;
```

Although this is an option that works, it's not recommended. Papervision3D has some issues with scaled 3D objects at the time of writing. It is a good convention to use the same units in Papervision3D and your 3D modeling tool.

If you want to learn more about modeling with SketchUp, visit the support page on the SketchUp web site `http://sketchup.google.com/support`. You'll find help resources such as video tutorials and a help forum.

Creating and loading models using Blender

Blender is an open source, platform-independent 3D modeling tool, which was first released in 1998 as a shareware program. It went open source in 2002. Its features are similar to those in commercial tools such as 3ds Max, Maya, and Cinema4D. However, it has a reputation of having a difficult learning curve compared to other 3D modeling programs. Blender is strongly based on usage of keyboard shortcuts and not menus, which makes it hard for new users to find the options they're looking for. In the last few years, more menu-driven interfaces have been added.

It's not in the scope of this book to teach you everything about the modeling tools that can be used with Papervision3D. This also counts for Blender. There are many resources such as online tutorials and books that cover how to work with Blender.

Blender 3D—architecture, buildings, and scenery

Packt Publising has released a book called *Blender 3D: Architecture, Buildings, and Scenery*, which is targeted at modeling for architecture—a good starting point for learning how to model in Blender . It describes the most important things you need to know about modeling for Papervision3D.

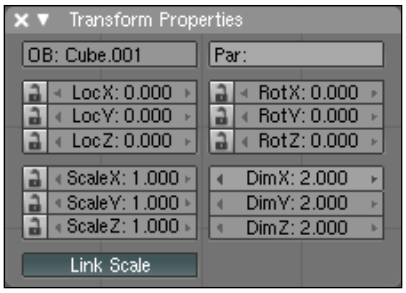
A link to the Blender installer download can be found on its web site: `www.blender.org`.

Exporting a textured cube from Blender into Papervision3D

For this example, we're going to create a textured and UV mapped cube to show how we can export a model from Blender to Papervision3D properly.The most recent versions of Blender have an integrated COLLADA exporter. Where the integrated exporter in 3ds Max has issues with Papervision3D, this exporter works fluently with Papervision3D.

Let's see how we have to model and texture a cube in Blender:

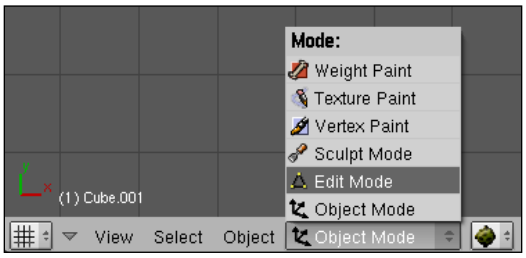
1. Start Blender. By default it opens a minimal scene, made up of a camera, a 3D object, and a light source.
2. Go to **Add | Mesh | Cube** in the top menu. This will add a new cube to the scene.
3. Place the cube on the origin of the scene. You can do this by dragging the gizmo or by changing the transform properties. These can be opened by using the **Object** menu (located at the bottom of the viewports) and by going to **Transform Properties** (shortcut *n*). Set **LocX**, **LocY**, and **LocZ** to **0**, which will set the cube's position to zero on all axes. Double-clicking the current values makes them editable.



Selecting objects

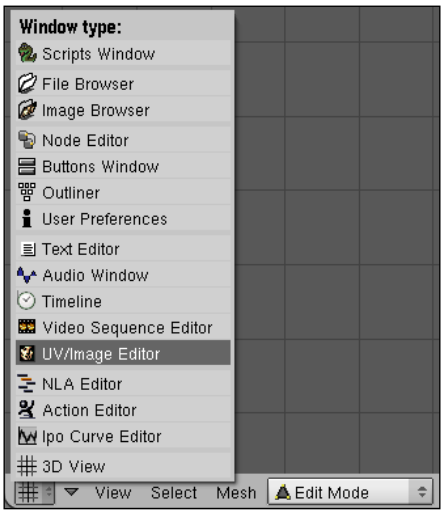
In case you lose the selection of an object in a Blender scene, you can re-select it by right-clicking it.

4. Scale the object, so it will match the units that we'll use in Papervision3D. This can be achieved in the **Transform Properties** panel. A dimension of 500 on all axes is a good value. You can set either the **Scale** values to `250` or the **Dim** values to `500`. When you select the **Link Scale** button, you have to change these values only for one axis, as it will constrain its proportions.
5. Scroll your mouse wheel to zoom out and see the whole cube again.
6. Change from **Object Mode** to **Edit Mode**.

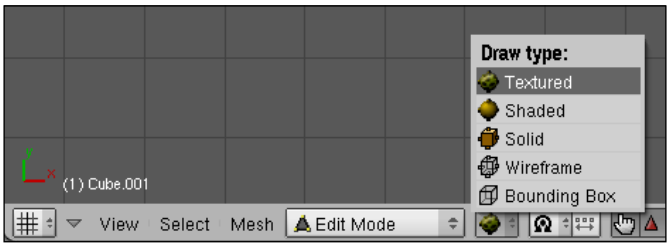


7. The **Object** menu will be replaced with **Mesh** menu. Collapse it and select **UV Unwrap** (shortcut *U* when in edit mode). Click the bottom option called **Unwrap (smart projections)** and click **OK** when a new window shows up. This will create a UV map for us.

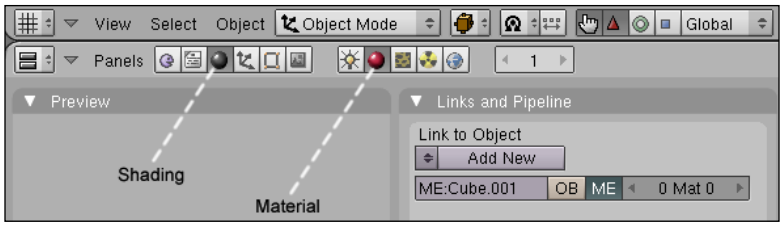
8. The unwrapped map of the surface can be found in the **UV/Image Editor**. You can change your view from **3D View** to **UV/Image Editor** by clicking the window icon at the bottom left corner of the 3D view of the scene.



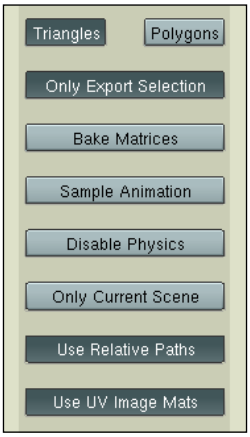
9. You might want to take this into account when selecting the image.
10. The UV/image editor will replace the 3D view and some new menu items show up. Go to the **Image** menu and select **Open** (shortcut *Alt* + *O*) to open up the image you want to use as texture. An example image can be found in the book downloads. In order to use relative paths in the exported model, we will save the model once it is finished to the same folder as the texture, which we have selected in this step.
11. The selected image should appear on your screen.
12. Exit the UV/image editor by selecting the **3D View** window type.
13. Change the **Draw type** from **Solid** to **Textured**, using the button next to the Mode selection, which we've previously set to **Edit Mode**.



14. In the **Buttons Window**, which is the bottom window by default, select the **Shading** and **Material** button.



15. You will see the **Links and Pipeline** panel, as shown in the previous image, press **Add New** to link the material to the cube.
16. After linking the object, a new panel called **Material** will show up. Select the **TexFace** button.
17. Save your file in the Blender format into the same directory as you have saved the used texture. This enables us to export using relative paths.
18. Make sure you have the cube still selected and go to **File | Export | COLLADA 1.4 (.dae)**. This will open the COLLADA exporter.
19. Enter location to save the file and make sure you have selected:
    - **Triangles**
    - **Only Export Selection** because we do not want to export any other objects in the scene
    - **Use Relative Paths**, so we do not have to change the path to the material manually in the COLLADA file
    - **Use UV Image Mats**, so the model will make use of the UV map



20. Press **Export and Close** or just **Export**, to save the COLLADA.

Once these steps are completed successfully, the cube is ready for loading in Papervision3D. Loading this model works exactly the same as loading COLLADA, which is created by 3ds Max.

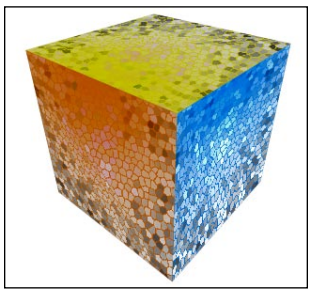
Copy the created model and texture to the `assets` folder of your project before loading the model. Loading requires you to have the `DAE` parser imported.

Change the `init()` method so that it will load the COLLADA model instead of the model from our previous example.

```
model = new DAE();
model.addEventListener(FileLoadEvent.LOAD_COMPLETE,modelLoaded);
DAE(model).load("assets/BlenderCube.dae");
```

{Ex.} COLLADA Blender Example {Ex.}

Publish your project and you should see the cube we created in Blender.



Keeping control over your materials

It is very convenient that the materials defined inside a 3D modeling tool can be used in Papervision3D. On the other hand, there are situations where you want to have control over materials. To name a few:

- When you want to use another material type such as a movie clip or streaming video
- When you want to change a material property such as `interactive`, `precise`, and `smooth`
- When you want to apply a shader

The moment you call the `load()` method using any of the 3D model parsers, you can pass a material list, specifying the materials you want to use. This works in a similar way to specifying a material list at instantiation of a cube and looks as follows:

```
var materials:MaterialsList = new MaterialsList();
materials.addMaterial( new ColorMaterial(0x0000FF),
"materialDefinition");
var model:DAE = new DAE();
model.load("model.dae",materals);
```

The `materialDefinition` string in the materials list refers to the unique ID of a material that was automatically set during export of a model. As you do not have any control over setting the ID yourself, you have to find it either by opening the COLLADA file in a text editor or trace it once the model has been loaded. The latter approach will be explained in a while.

The final example in this chapter shows how to change properties of a material once the object and materials are loaded. We're going to make the material interactive and rotate the object on each click. The previous example with the Blender-created cube will be used as a starting point.

Create a new project out of the previous example. For this new project we will use Tweener, so add the Tweener library to your **Build Path** (Flex Builder), **Classpath** (Flash CS3) or **Source Path** (Flash CS4), like we've seen in Chapter 4.

In the `init()` method we can set viewport interactivity to `true` and add an event listener to the model, waiting for `FileLoadEvent.LOAD_COMPLETE` to be dispatched.

```
viewport.interactive = true;
model.addEventListener(FileLoadEvent.LOAD_COMPLETE, modelLoaded);
```

Next, we define the `modelLoaded()` method. Once this method is triggered, we will have full access to the loaded model and its child objects.

```
private function modelLoaded(e:FileLoadEvent):void
{
```

As we're going to change the materials applied to the model, it might be helpful to trace `model.materials`, to find out their name(s). Some exporters automatically define a material name or add a suffix to the name, which was defined in the modeling program.

```
    trace("Used materials by this model: " + model.materials);
```

In our case this would trace `BlenderCube_jpg`. This string can be used to get an object's material, allowing you to set its properties.

```
model.getMaterialByName("BlenderCube_jpg").interactive = true;
```

Note that if you want to set `BitmapMaterial`-specific properties such as `smooth`, you first need to cast the material to `BitmapMaterial`.

Next, we define a listener waiting for clicks on an object nested inside the model. This needs to be set for every child object you want to be clickable. Therefore, we use `model.getChildByName` and search for a nested object. You can set the second parameter recursive to `true`, in order to search for a nested object inside a nested object. In fact, the model is a child object of the `DAE` class that was used to load it.

```
    model.getChildByName("Cube_001", true).addEventListener(Interactiv
eScene3DEvent.OBJECT_CLICK, click);
}
```

The name `Cube_001` was automatically defined inside the modeling tool. You can also see this name when Papervision3D parses the object and traces its name in the output window.

```
INFO: DisplayObject3D: Cube_001
```

To see this you can publish the previous project that also loads the cube created in Blender.

In the final part of this example, we set up the `click()` method that will be triggered each time the cube is clicked. Tweener will be used to animate the cube.

```
private var targetRotationX:Number = 0;

private function click(e:InteractiveScene3DEvent):void
{
    targetRotationX+=90;
    Tweener.addTween(model, {localRotationX:targetRotationX, time:1.5,
transition:"easeOutElastic"});
}
```

{Ex.} Controlling Materials Example {Ex.}

Publish this code and you will see the same cube as in the Blender example. However, clicking on it will rotate the cube over its local x-axis, with an elastic transition.

As this example shows, we can change material properties at run time. If you like, you can even change the material at run time, just like you would replace a material on primitive cube. As we have seen in previous chapters a primitive cube also works with a material list.

Summary

Modeling is a very broad topic as there are so many 3D programs, each with numerous features. When you want to display custom objects besides the built-in primitives, you can load models created by 3D programs.

This chapter showed how to create basic models in 3ds Max, SketchUp, and Blender, and how to export them for Papervision3D. To do this we've used three different file formats:

- COLLADA (`.dae`): An open source 3D model file type, which has been supported since the early releases of Papervision3D. This is the most developed file type, which also supports animation and animation clips.
- 3D Studio (`.3ds`): An established 3D file format that is supported by most 3D modeling programs.
- SketchUp (`.kmz`): A format that is used by Google Earth, which can be created by a free program called SketchUp.

Creating models for use in Papervision3D has some requirements and conventions to take into account:

- Keep a low polygon count
- Add polygons to problematic parts of your model to prevent z-sorting artifacts or texture distortion
- Keep your texture small
- Use textures Flash can read
- Use UV maps
- Bake textures
- Use recognizable names for objects and materials
- Use the same metrics as in Papervision3D
- Find balance in optimization

Models that are loaded in Papervision3D automatically load images that are defined as materials in the 3D modeling program. At the end of this chapter we've seen how we can have access to these materials. The way we can access a model's material doesn't differ from accessing a material on a primitive cube.

The next chapter discusses z-sorting. We're going to have a look at how Papervision3D draws its renders to the viewport and the issues with determining which object should be drawn first. The mill that was used in this chapter will be part of an example on how to solve z-sorting issues.