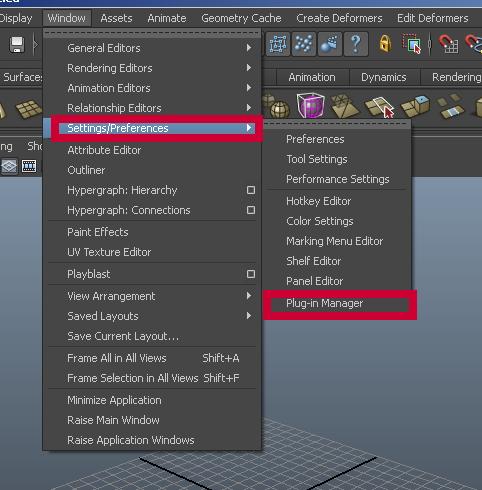
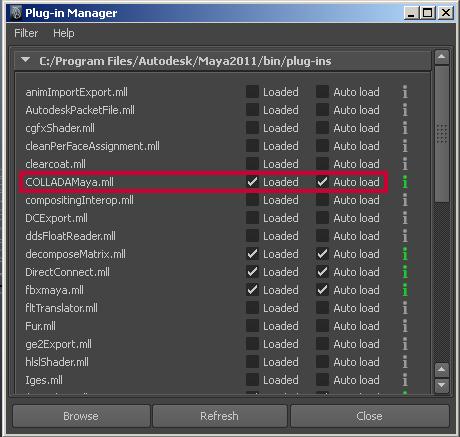
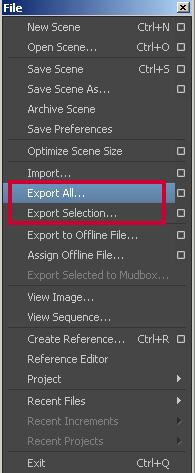
**Should set up Maya openCollda plugin,and then following my Pic Guide**

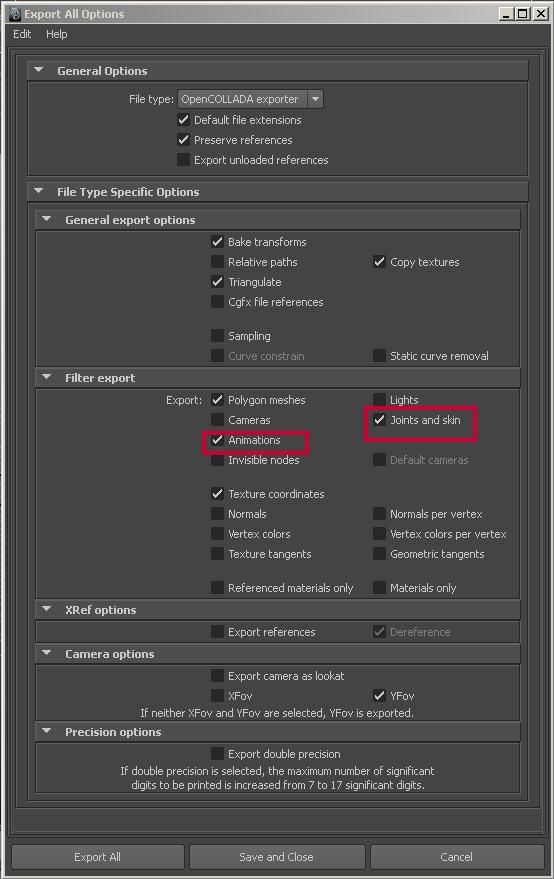




       After that, we can export DAE from Maya



If the DAE include animations or Joint and Skin info. choose them from the Option Menu;



**Optimize the standard material**

[StandardMaterial](http://www.virtuos.net/groups/flash-group/forum/topic/optimize-the-standard-material/)(diffuseMap:[TextureResource](http://www.virtuos.net/groups/flash-group/forum/topic/optimize-the-standard-material/), normalMap:[TextureResource](http://www.virtuos.net/groups/flash-group/forum/topic/optimize-the-standard-material/), specularMap:[TextureResource](http://www.virtuos.net/groups/flash-group/forum/topic/optimize-the-standard-material/) = null, glossinessMap:[TextureResource](http://www.virtuos.net/groups/flash-group/forum/topic/optimize-the-standard-material/) = null, opacityMap:[TextureResource](http://www.virtuos.net/groups/flash-group/forum/topic/optimize-the-standard-material/) = null)

The standard material include 5 parameters .2 of them are essential ,that is diffuseMap and normalMap, which means we should have at least 2 materials to show each models. That will double the number of the assets if all the model needs 2 kinds of material.

In order to solve this problem. We use the BitmapTextureResource as the normal TextureResource of the normalMap which is set as “0x7F80FF”,just the background color of normal material. We set it to some model that is not very important like the level ground.

So it's like

**var** \_normal:BitmapTextureResource = **new** BitmapTextureResource(**new** BitmapData(1, 1, **false**, 0x7F80FF));

                                                                   surface.material=**new** StandardMaterial(diffuse,\_normal);

which will save more loading period

### stage3d 上下文丢失解决方案

* 转自天地会：

Context3d，虽然我更喜欢通俗地译作3d环境，不过计算机编程的码农们似乎更偏爱上下文这种专业气息的叫法。

言归正传，我想大家也都知道，Stage3D的上下文是会丢失的，不仅是Stage3D，这一点在其它语言中也会出现。原因很简单，显存就那么大，通常当然要被一个程序独占了，那么切换程序时，原有的程序自然就会失去对显卡的控制权，导致资源被释放，上下文丢失了。

这是个麻烦又不麻烦的事件。

麻烦在于它随时可以出现，只要你ctrl+alt+delete（或者徽标+L）切换到系统页面，就可以让一切程序都丢失上下文了。对于web3D而言，切换窗口时同样也可能会导致其上下文丢失。

不麻烦在于，你可以收到新的context3d创建事件，以便重复之前的数据载入过程，也就是将网格数据、纹理数据还有着色器等上传到显存。

这个问题很多引擎都还没有解决的样子，minko在丢失后就一直丢了，而proscenium则是报个错，然后再恢复正常。

但不管怎么说，比起引擎，这个更应该由用户自行解决（毕竟引擎不知道你需要重载什么），而解决的方法很简单！

1、不要删掉注册的Event.CONTEXT3D\_CREATE事件，以便接受重载事件

2、在事件处理函数的最上面，先删除渲染循环事件（Event.ENTER\_FRAME），等上传完成后再恢复，以防止渲染

3、在渲染循环处理函数中，先对clear方法进行try catch，以判断是否上下文丢失

就是这样，你只需要根据自己的目的，分情况处理即可了。

[Quote](http://www.virtuos.net/groups/flash-group/forum/topic/stage3d-%e4%b8%8a%e4%b8%8b%e6%96%87%e4%b8%a2%e5%a4%b1%e8%a7%a3%e5%86%b3%e6%96%b9%e6%a1%88/#post-reply) | [#](http://www.virtuos.net/groups/flash-group/forum/topic/stage3d-%e4%b8%8a%e4%b8%8b%e6%96%87%e4%b8%a2%e5%a4%b1%e8%a7%a3%e5%86%b3%e6%96%b9%e6%a1%88/#post-2607)

* Avatar Image[Yuting Xiao](http://www.virtuos.net/members/xiaoyuting/) said 2 years, 11 months ago:    *@xiaoyuting*

It's very important, in Book "Packt Adobe Flash 11 Stage3D(Molehill) Game Programming Beginner's Guide" page 47 also mentions. So always, we should never write initialize codes in the constructor, instead, we should write them in the callback function of event "Event.CONTEXT3D\_CREATE".

So as my experience the initialization of Stage3D should be as following

public function AS3D() // constructor

{

            if (stage != null)  
                init();  
            else  
                addEventListener(Event.ADDED\_TO\_STAGE, init);

}

private function init(e:Event = null):void  
        {  
            if (hasEventListener(Event.ADDED\_TO\_STAGE))  
                removeEventListener(Event.ADDED\_TO\_STAGE, init);  
              
            // class constructor – sets up the stage  
            stage.scaleMode = StageScaleMode.NO\_SCALE;  
            stage.align = StageAlign.TOP\_LEFT;  
              
            // and request a context3D from Stage3d  
            stage.stage3Ds[0].addEventListener(  
                Event.CONTEXT3D\_CREATE, onContext3DCreate);  
            stage.stage3Ds[0].requestContext3D();  
        }  
          
        private function onContext3DCreate(event:Event):void  
        { …… }

1.how to create ADT enviroment

2.how to code ADT

3.how to package more assets use ADT

1------enviroment

The AIR Developer Tool (ADT) is a multi-purpose, command-line tool for developing AIR applications. You can use ADT to perform the following tasks:

■Package an AIR application as an .air installation file

■Package an AIR application as a native installer—for example, as a .exe installer file on Windows, .ipa on iOS, or .apk on Android

■Package a native extension as an AIR Native Extension (ANE) file

■Sign an AIR application with a digital certificate

■Change (migrate) the digital signature used for application updates

■Create a self-signed digital code signing certificate

■Remotely install, launch, and uninstall an application on a mobile device

■Remotely install and uninstall the AIR runtime on a mobile device

ADT is a Java program included in the AIR SDK. You must have Java 1.5 or higher to use it. The SDK includes a script file for invoking ADT. To use this script, the location of the Java program must be included in the path environment variable. If the AIR SDK bin directory is also listed in your path environment variable, you can type adt on the command line, with the appropriate arguments, to invoke ADT. (If you do not know how to set your path environment variable, please refer to your operating system documentation. As a further aid, procedures for setting the path on most computer systems are described in Path environment variables.)

At least 2GB of computer memory is required to use ADT. If you have less memory than this, ADT can run out of memory, especially when packaging applications for iOS.

Assuming both Java and the AIR SDK bin directory are both included in the path variable, you can run ADT using the following basic syntax:

adt -command options

2------iOS packages

On iOS, ADT converts the SWF file byte code and other source files into a native iOS application.

1.Open a command shell or a terminal and navigate to the project folder of your iPhone application.

2.Next, use the ADT tool to create the IPA file, using the following syntax:

adt     -package

    -target [ipa-test | ipa-debug | ipa-app-store | ipa-ad-hoc]

    -provisioning-profile PROFILE\_PATH

    SIGNING\_OPTIONS

    TARGET\_IPA\_FILE

    APP\_DESCRIPTOR

    SOURCE\_FILES

    -extdir extension-directoryChange the reference adt to include the full path to the adt application. The adt application is installed in the bin subdirectory of the AIR SDK.

Select the -target option that corresponds to the type of iPhone application you want to create:

■-target ipa-test—Choose this option to quickly compile a version of the application for testing on your developer iPhone.

■-target ipa-debug—Choose this option to compile a debug version of the application for testing on your developer iPhone. With this option, you can use a debug session to receive trace() output from the iPhone application.

You can include one of the following -connect options (CONNECT\_OPTIONS) to specify the IP address of the development computer running the debugger:

■-connect—The application will attempt to connect to a debug session on the development computer used to compile the application.

■-connect IP\_ADDRESS—The application will attempt to connect to a debug session on the computer with the specified IP address. For example:

-target ipa-debug -connect 192.0.32.10■-connect HOST\_NAME—The application will attempt to connect to a debug session on the computer with the specified host name. For example:

-target ipa-debug -connect bobroberts-mac.example.comThe -connect option is optional. If not specified, the resulting debug application will not attempt to connect to a hosted debugger.

If a debug connection attempt fails, the application presents a dialog asking the user to enter the IP address of the debugging host machine. A connection attempt can fail if the device is not connected to wifi. It can also occur if the device is connected but not behind the firewall of the debugging host machine.

For more information, see Debugging a mobile AIR application.

■-target ipa-ad-hoc—Choose this option to create an application for ad hoc deployment. See the Apple iPhone developer center

■-target ipa-app-store—Choose this option to create a final version of the IPA file for deployment to the Apple App Store.

Replace the PROFILE\_PATH with the path to the provisioning profile file for your application. For more information on provisioning profiles, see iOS setup.

Replace the SIGNING\_OPTIONS to reference your iPhone developer certificate and password. Use the following syntax:

-storetype pkcs12 -keystore P12\_FILE\_PATH -storepass PASSWORDReplace P12\_FILE\_PATH with the path to your P12 certificate file. Replace PASSWORD with the certificate password. (See the example below.) For more information on the P12 certificate file, see Converting a developer certificate into a P12 keystore file.

Replace the APP\_DESCRIPTOR to reference the application descriptor file.

Replace the SOURCE\_FILES to reference the main SWF file of your project followed by any other assets to include. Include the paths to all icon files you defined in the application settings dialog box in Flash CS5 or in a custom application descriptor file. Also, add the initial screen art file, Default.png.

Use the -extdir extension-directory option to specify the directory that contains the ANE files (native extensions) that the application uses. If the application uses no native extensions, do not include this option.

Important: Do not create a subdirectory in your application directory named Resources. The runtime automatically creates a folder with this name to conform to the IPA package structure. Creating your own Resources folder results in a fatal conflict.

Creating an iOS package for debugging

To create an iOS package for installing on test devices, use the ADT package command, setting the target type to ios-debug. Before running this command, you must have already obtained a development code signing certificate and provisioning profile from Apple.

adt     -package

    -target ipa-debug

    -storetype pkcs12 -keystore ../AppleDevelopment.p12

    -provisioning-profile AppleDevelopment.mobileprofile

    -connect 192.168.0.12

    myApp.ipa

    myApp-app.xml

    myApp.swf icons Default.pngType the entire command on a single line; line breaks in the above example are only present to make it easier to read. Also, the example assumes that the path to the ADT tool is on your command-line shell’s path definition. (See Path environment variables for help.)

You must run the command from the directory containing the application files. The application files in the example are myApp-app.xml (the application descriptor file), myApp.swf, an icons directory, and the Default.png file.

You must sign the application using the correct distribution certificate issued by Apple; other code signing certificates cannot be used.

When you specify the -connect option, the application will attempt to initiate a debug session with the Flash Debugger (FDB) running on the specified IP or host name. See Connecting to the Flash debugger for more information.

Creating an iOS package for Apple App Store submission

To create an iOS package for submission to the Apple App store, use the ADT package command, setting the target type to ios-app-store. Before running this command, you must have already obtained a distribution code signing certificate and provisioning profile from Apple.

adt     -package

    -target ipa-app-store

    -storetype pkcs12 -keystore ../AppleDistribution.p12

    -provisioning-profile AppleDistribution.mobileprofile

    myApp.ipa

    myApp-app.xml

    myApp.swf icons Default.pngType the entire command on a single line; line breaks in the above example are only present to make it easier to read. Also, the example assumes that the path to the ADT tool is on your command-line shell’s path definition. (See Path environment variables for help.)

You must run the command from the directory containing the application files. The application files in the example are myApp-app.xml (the application descriptor file), myApp.swf, an icons directory, and the Default.png file.

You must sign the application using the correct distribution certificate issued by Apple; other code signing certificates cannot be used.

Important: Apple requires that you use the Apple Application Loader program in order to upload applications to the App Store. Apple only publishes Application Loader for Mac OS X. Thus, while you can develop an AIR application for the iPhone using a Windows computer, you must have access to a computer running OS X (version 10.5.3, or later) to submit the application to the App Store. You can get the Application Loader program from the Apple iOS Developer Center.

Creating an iOS package for ad hoc distribution

To create an iOS package for ad hoc distribution, use the ADT package command, setting the target type to ios-ad-hoc. Before running this command, you must have already obtained the appropriate ad hoc distribution code signing certificate and provisioning profile from Apple.

adt     -package

    -target ipa-ad-hoc

    -storetype pkcs12 -keystore ../AppleDistribution.p12

    -provisioning-profile AppleDistribution.mobileprofile

    myApp.ipa

    myApp-app.xml

    myApp.swf icons Default.pngType the entire command on a single line; line breaks in the above example are only present to make it easier to read. Also, the example assumes that the path to the ADT tool is on your command-line shell’s path definition. (See Path environment variables for help.)

You must run the command from the directory containing the application files. The application files in the example are myApp-app.xml (the application descriptor file), myApp.swf, an icons directory, and the Default.png file.

You must sign the application using the correct distribution certificate issued by Apple; other code signing certificates cannot be used.

Creating an iOS package for an application that uses native extensions

To create an iOS package for an application that uses native extensions, use the ADT package command with the -extdir option. Use the ADT command as appropriate for the target (ipa-app-store, ipa-debug, ipa-ad-hoc, ipa-test). For example:

adt     -package

    -target ipa-ad-hoc

    -storetype pkcs12 -keystore ../AppleDistribution.p12

    -provisioning-profile AppleDistribution.mobileprofile

    myApp.ipa

    myApp-app.xml

    -extdir extensionsDir

    myApp.swf icons Default.pngType the entire command on a single line; line breaks in the above example are only present to make it easier to read.

Regarding native extensions, the example assumes that the directory named extensionsDir is in the directory in which you run the command. The extensionsDir directory contains the ANE files that the application uses.

3--------package more assets

Package only the main files and an images subdirectory:

adt –package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.html AIRAliases.js imagesPackage an HTML-based application and all files in the HTML, scripts, and images subdirectories:

adt –package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml index.html AIRALiases.js html scripts imagesPackage the application.xml file and main HTML file located in a working directory (src):

adt –package -storetype pkcs12 -keystore cert.p12 myApp.air src/myApp.xml –C src myApp.html Package assets from more than one place in your build file system. In this example, the application assets are located in the following folders before packaging:

/devRoot

    /myApp

        /release

            /bin

                myApp.xml

                myApp.html

    /artwork

        /myApp

            /images

                image-1.png

                ...

                image-n.png

    /libraries

        /release

            /libs

                lib-1.js

                ...

                lib-n.js

                AIRAliases.jsRunning the following ADT command from the /devRoot/myApp directory:

adt –package -storetype pkcs12 -keystore cert.p12 myApp.air release/bin/myApp.xml

    –C release/bin myApp.swf

    –C release/bin myApp.html

    –C ../artwork/myApp images

    –C ../libraries/release libs