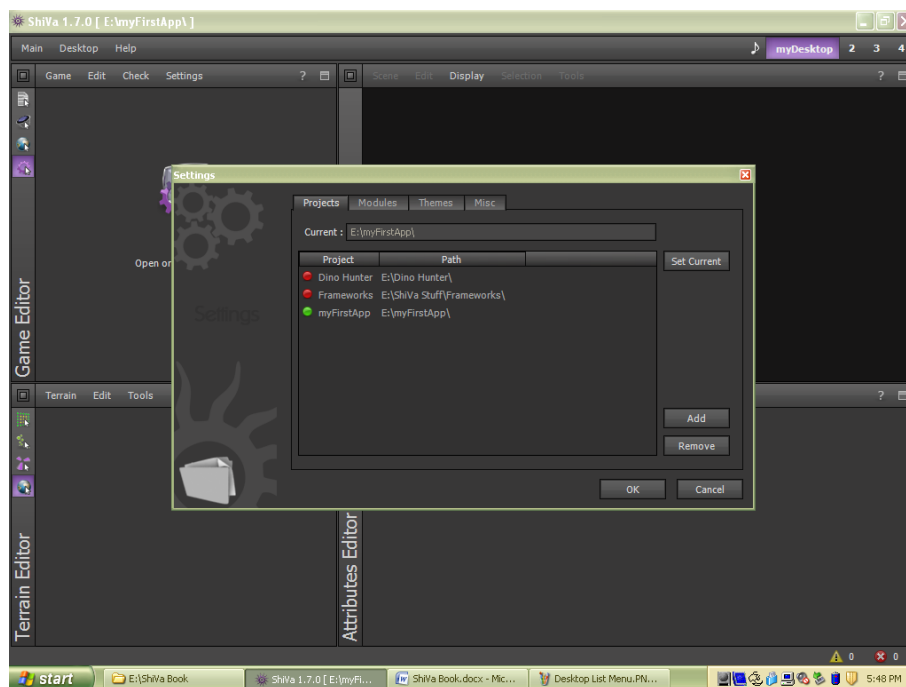


Your First ShiVa Application

In this chapter, I'm going to take you step-by-step through creating your very first ShiVa application. Hopefully, it will be easy to follow, and at the end you will have been able to go away and start creating your own applications.

CREATE YOUR PROJECT

Firstly you will have to create your new ShiVa project, this is done by selecting “Main” and then “Settings” from the desktop. “Main” can be found at the top left of the ShiVa desktop, underneath the title bar.



In the dialog box that is displayed, you need to left-click on the “Add” button, and select the root directory for your project. For example: “E:\myFirstApp”.

Tip!

ShiVa will automatically create the necessary file structure within your selected folder. So, you don't have to worry about that either, great huh?.

Once your directory structure has been created, ShiVa will automatically set this new project as your default project, and you can simply click on the “OK” button.

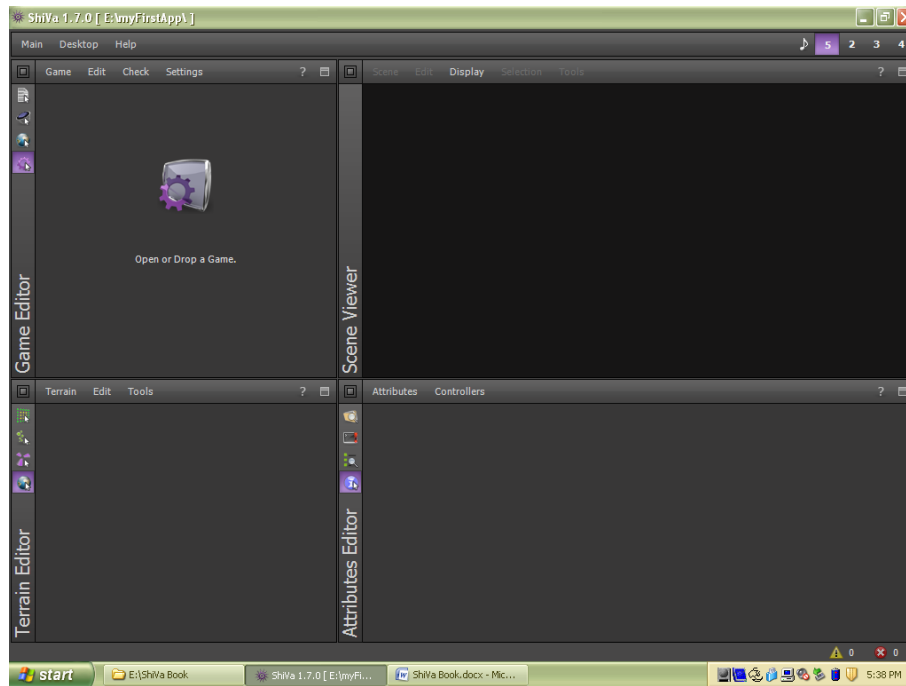
YOUR FIRST SHIVA APPLICATION

Tip!

To access previous projects, just select them in the “Setup” dialog, and either “double-click” on the project, or use the “Set Current” button. You will notice that the current default project has a green circle next to it.

To remove projects, just simply select them and press the “Remove” button. BE CAREFUL, ShiVa does NOT ask you if you want to delete the project, it just removes it!

Now, your screen should be similar to the screenshot below (pretty boring eh?)



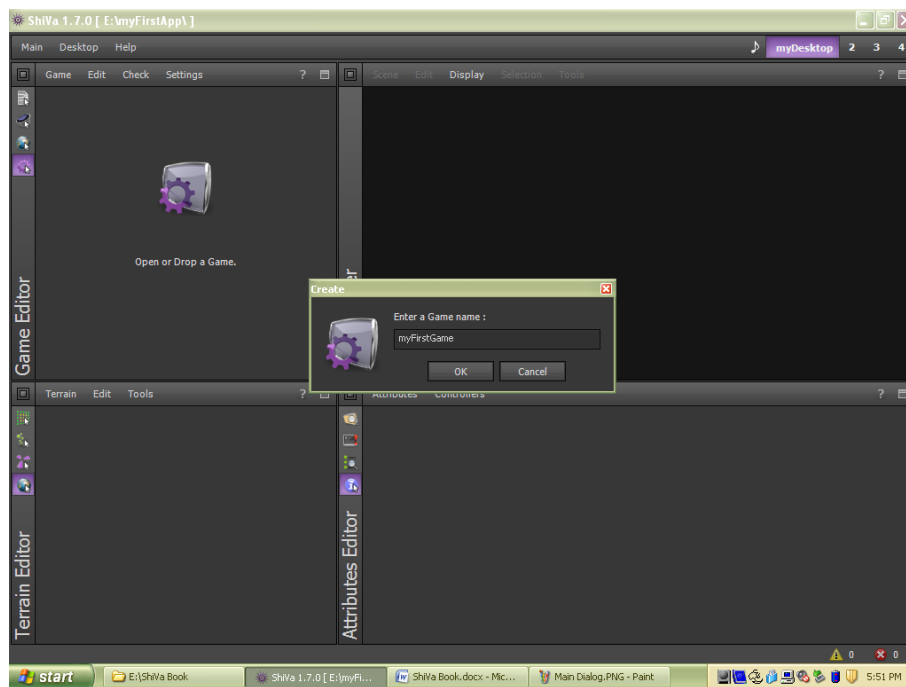
So, let's do something with it.....

CREATE YOUR GAME

In ShiVa, a Game is the basic resource representing the app. It doesn't mean that you can only create games with ShiVa though!, oh no, as I've previously stated, ShiVa can be used to create virtually any type of 3D app, and thanks to the various players (both standalone and web) provided by StoneTrip, these apps can be pretty much presented anywhere, even on the iPhone, now that is really cool. Just think, your app could be played standalone on a PC or Mac, on the web, or even on a mobile phone!!!

Enough of this I hear you say, let's just create the app....

To create the app simply select the Game Editor () and left-click “Game” and then “Create”:



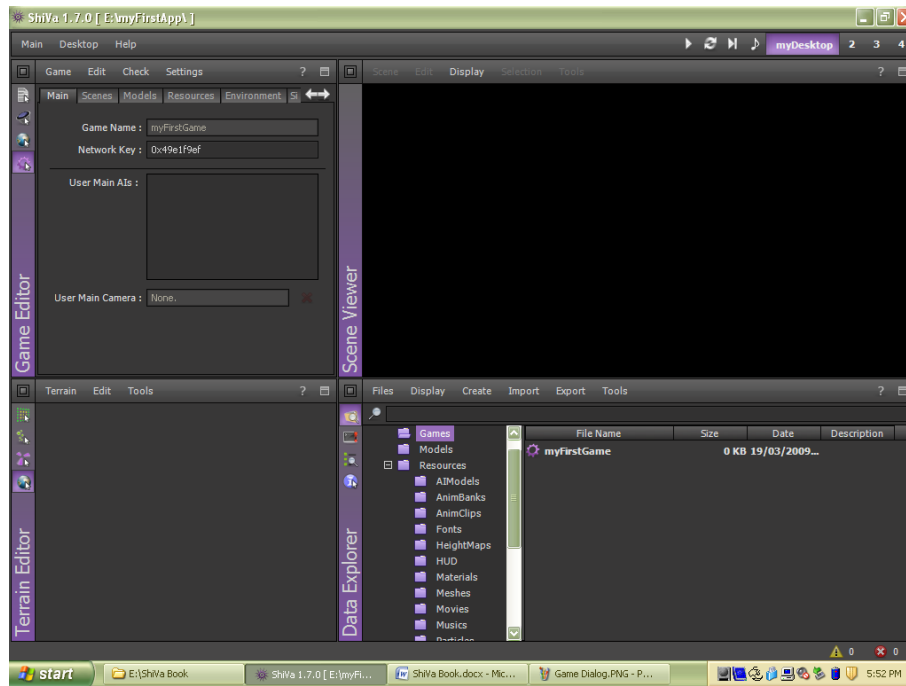
Now, simply type the name of your app (myFirstGame?), and click on the “OK” button. That's it! your app is created.....

Time for me to go.....

Only joking! there's still quite a bit to do before this app is finished! I did tell you that 3D applications are pretty complex, didn't I?

You should now have a screen similar to that shown below:

YOUR FIRST SHIVA APPLICATION



If you can't see the various screens, just select them from the drop-down lists. You'll notice that ShiVa helpfully labels the various modules in use, though you may have to tilt your head through 90 degrees to read them!

You may have also noticed that some buttons have appeared in the top right of the screen. These buttons are used to control your app, by allowing you to:

- play and pause your app
- relaunch your app by rebooting the in-built ShiVa monitor
- step through your app one Frame at a time
- mute the speakers

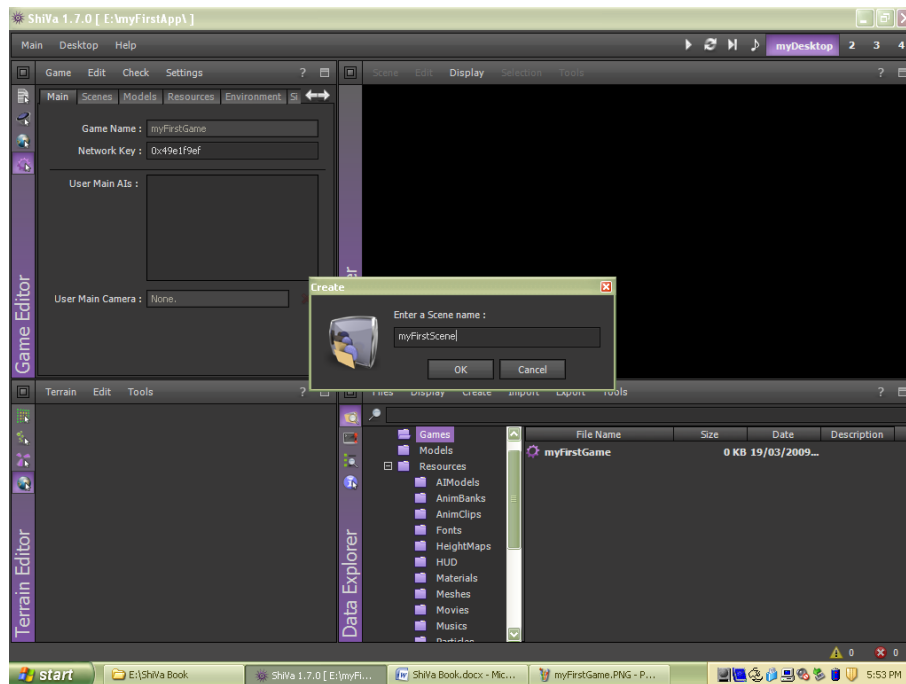
The app that you have just created is not really much of an application as it can't actually do anything. There are no Models to see, no Cameras to see them with, no scenery etc.

So, let's get on and create some content for your app.....

CREATING A SCENE (AND I MEAN THAT IN A 3D WAY!)

In ShiVa, your Scene is the 3D world in which you, or your Players, will interact. It is also the container for your Models, both static and dynamic.

To create an empty Scene, open the Data Explorer module and left-click on “Create” and then “Scene”. Give your Scene a name in the pop-up box (myFirstScene?) and click on the “OK” button.



Scenes can also be imported from modelling programs (as long as they can export in the Collada 1.4 format), by clicking on “Import” and then “Scene” and browsing to your modelled Scene. The import dialog box provides some specific options for importing (e.g.: Import Materials, Import Meshes, Import Cameras etc.) and more details on this can be found in the ShiVa docs.

Also, you can import Scenes from ShiVa archive files (.ste files) by clicking on “Import” and then “Archive”. We will be using this method when we come to the HLDL part of the book later.

When your Scene has been created, save it by clicking on “Scene” and then “Save” from within the Scene Viewer module.

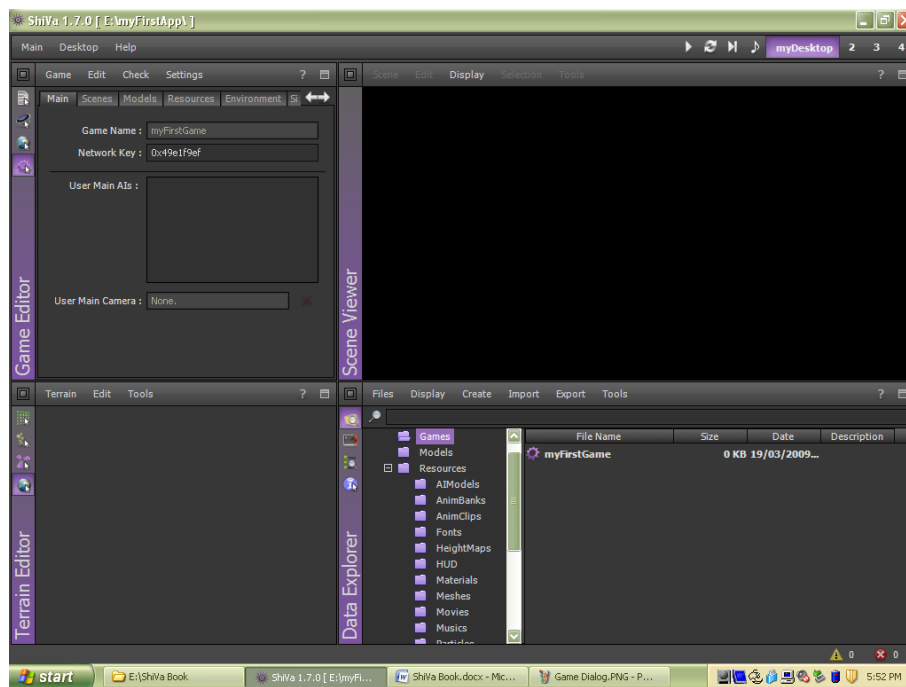
Hang on..... there is no option to save available in Scene Viewer??????

What I found when I first tried to use ShiVa is that you have to load things first. **This is VERY IMPORTANT.**

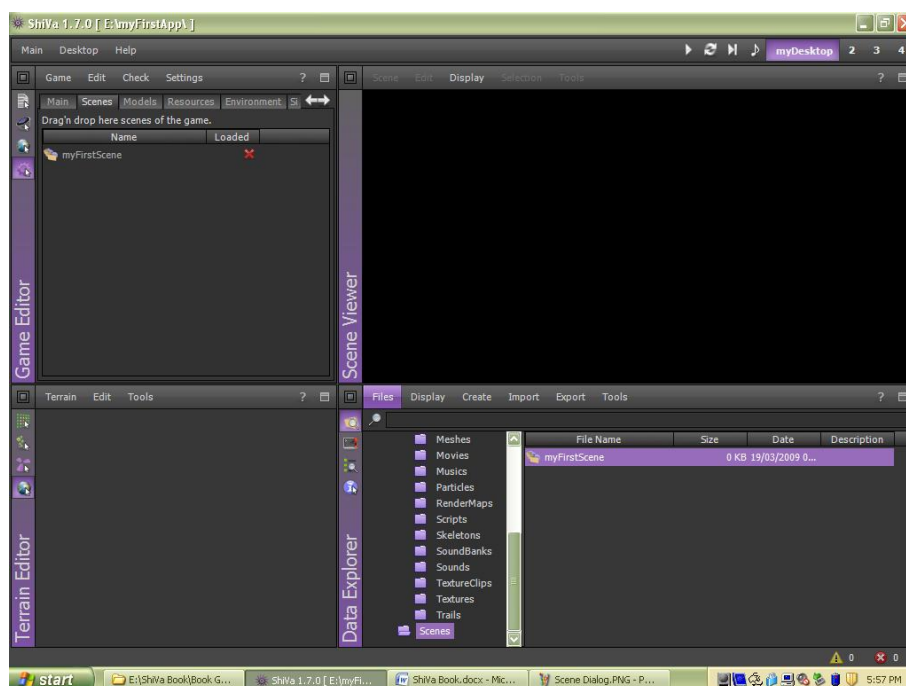
So, what do I have to do?

First, you have to actually load up your new app, by going to the Data Explorer module, clicking on the “Games” folder and then either double-clicking on “myFirstGame” (or whatever you called it), or dragging & dropping the file from the Data Explorer into the Game Editor.

YOUR FIRST SHIVA APPLICATION



Secondly, you have to load up the new Scene. So, switch to the Scenes tab in the Game Editor and then click on the Scene folder in the Data Explorer, and either drag and drop the file “myFirstScene” (or whatever you called it) into the Scenes tab of the Game Editor. You will notice that a big red **X** appears next to your Scene:



This big red **X** is basically telling you that the Scene is NOT loaded. In this way, you can keep track of all your Scenes used by your app, and know exactly which one you are currently working with.

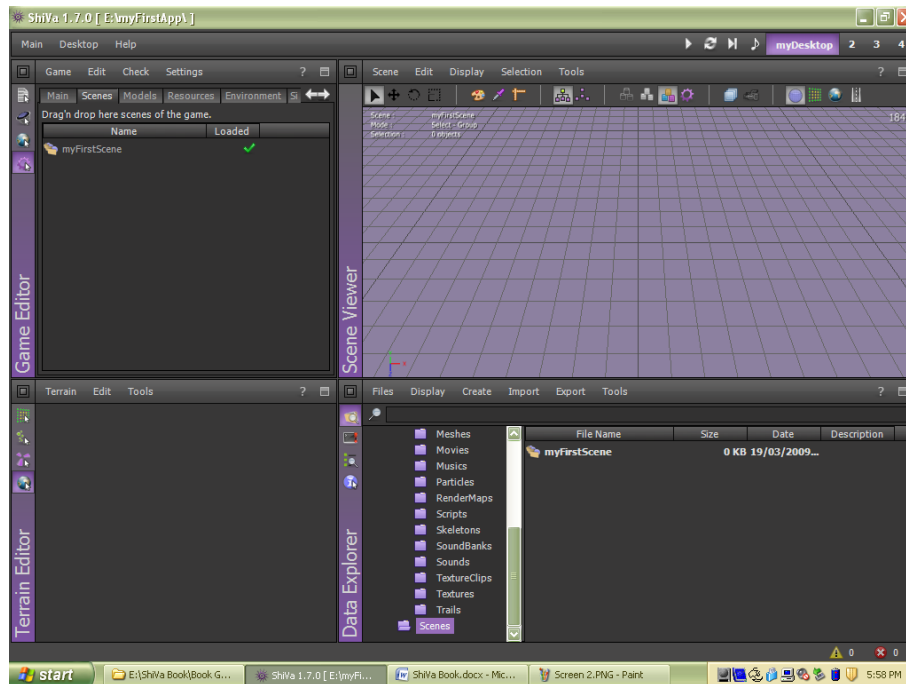
I know that all this switching between modules seems painful at first, but, once you get used to it, it feels quite natural. (For those of us old enough to remember, just think of the difference when we moved from DOS to Windows – Yes I am that old!, Windows soon became natural.)

YOUR FIRST SHIVA APPLICATION

Anyway, back to your Scene...

Now you need to load up our Scene. To do this you need to double-click on the Scene in the Game Editor, and the big red **X** will change to a big green **✓**. This tick shows you which Scene is currently selected.

Now, if you open up the Scene Viewer, you will see a grid pattern with the name of your Scene in the top left corner, along with some other text (which we won't worry about for now), and a number in the top right corner. The number corresponds to the frames per second (fps) that your video card is pushing through on this Scene.



As you can see, my video card (Mobile Intel 945GM) is pushing 180+ frames per second! That's pretty good, but, I do have one slight problem, there's actually nothing in the Scene, so it isn't really that surprising. Wait until a few items have been added, and watch the fps plummet! As long as you can keep the fps up around 60 your app should be perfectly playable. By the way, my desktop graphics card (HD3870 512MB) pushes over 1000 fps at this point!

OK, now you should be able to save the Scene. Using "Scene" and then "Save" from within the Scene Viewer.

Right, so you now have an empty Scene in your app. Next you will introduce one of the most important aspects of your app, the Camera, without which you would never be able to see a thing.

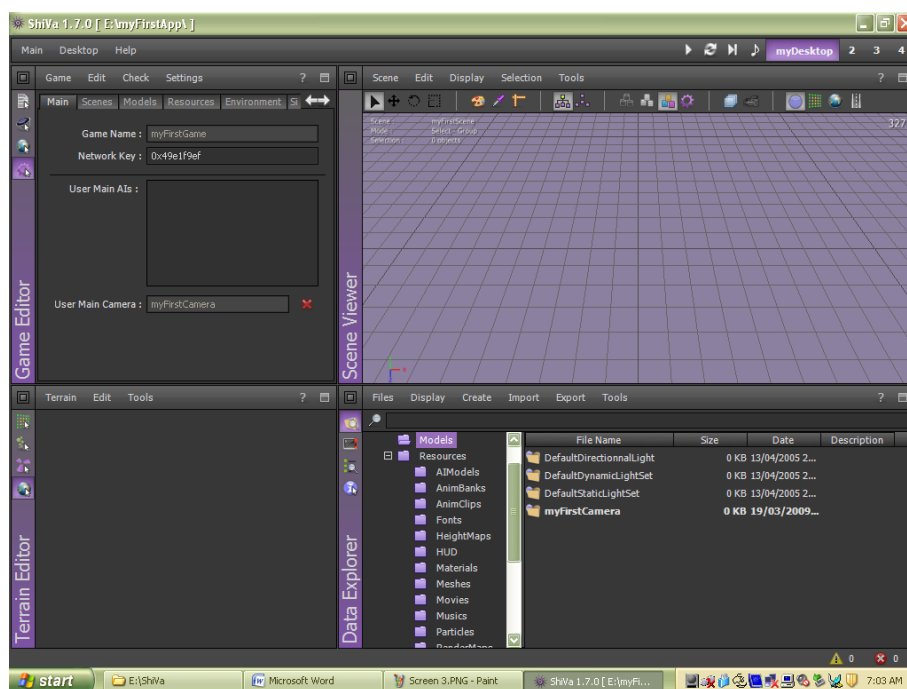
CREATING A CAMERA

A Camera is created in a very similar way to a Scene.

Firstly, go to the Data Explorer, and click on “Create”, then on “Model”, then on “Camera” and finally on “Simple”. Again, name your Camera (myFirstCamera??), and click on “OK”.

If you look in your Data Explorer window, you will now see a Camera sitting in the Models folder.

Now, all you have to do is drag and drop the newly created Camera into the Game Editor. To do this make sure that you are in the “Main” tab of the Game Editor and drag the Camera to the box labelled “User Main Camera”.



NOTE: The big red **X** next to the User Main Camera box is for removing the Camera, and does **NOT** denote whether the Camera is the active one or not.

Well, we are almost there (whew! I hear you say), just a little bit of scripting left, and you will have a functional app. (Well, sort of, there won't be any Models in there yet!)

CREATING YOUR SCRIPTS

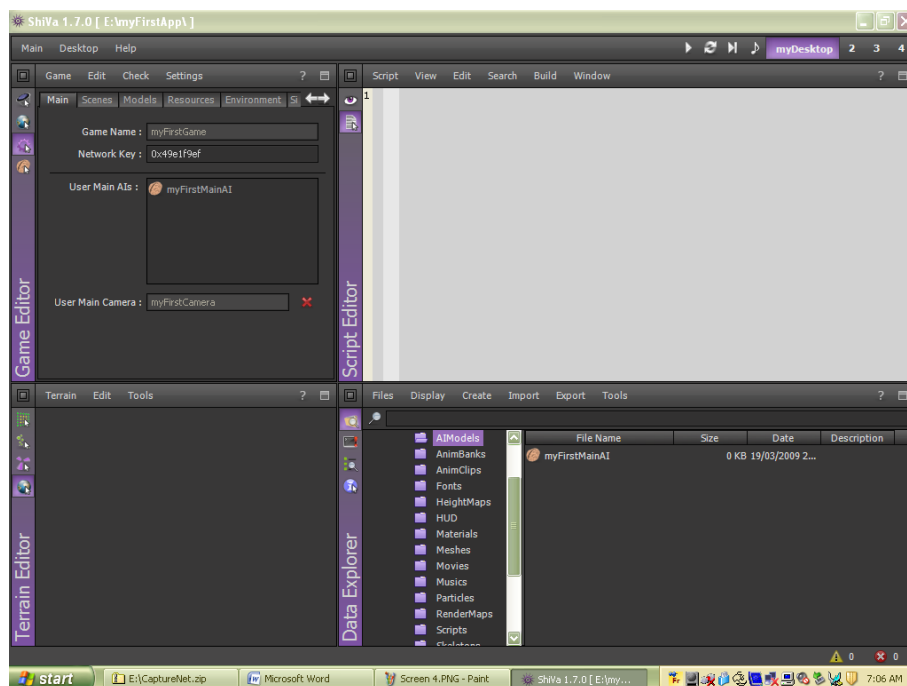
Scripts, unfortunately, are a necessary evil of almost every 3D development environment on the market. These Scripts can range from simple programs of a few lines (as you will see here) up to thousands of lines of code written in a programming language like C, C++ or Java.

Fortunately, for you, ShiVa has excellent scripting capability, and there are Scripts (that have been pre-written for you) that can be downloaded and installed into your app (which is exactly what you will be doing with the HLDL later in the book). Always check the Download section and the WIKI of the ShiVa website before writing Scripts (oh, and don't forget the Forum as it has some excellent examples), as it can save many hours of heartache to use something that someone else has made freely available, rather than writing your own from scratch.

ShiVa uses Scripts based around the AIModel Editor. This module is where we can start to put some form of intelligence into our app (and we all know that any self-respecting app needs to have some form of intelligence!).

To create your AIModel, go to the Game Editor, and click on “Edit”, then “User Main AI” and then “Create”. Again, give it a name (myFirstMainAI??) and click on “OK”.

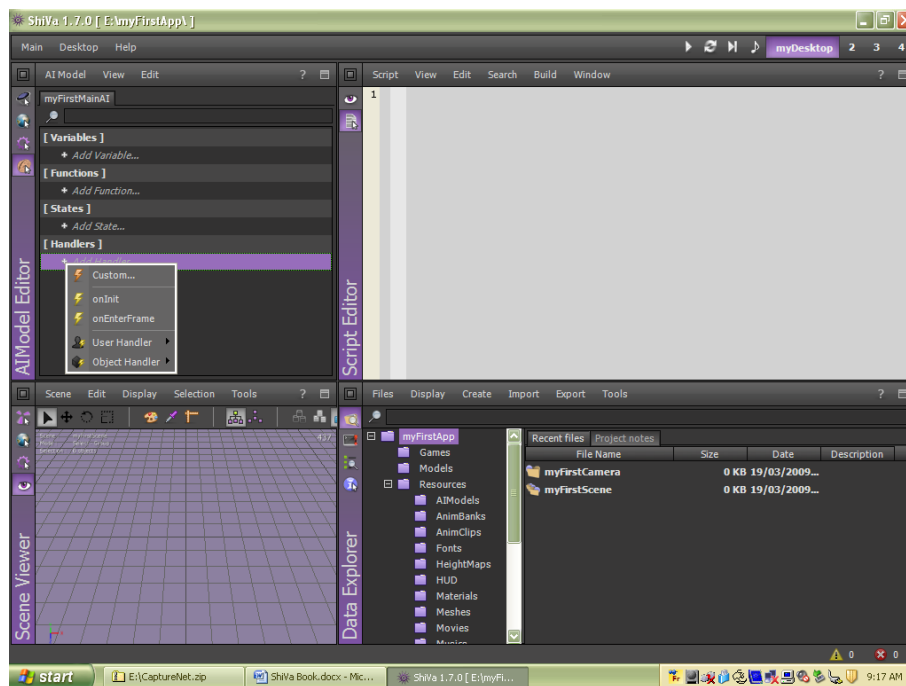
You will notice that your new Script is now showing in the Game Editor in the User Main AIs box. This box displays all the AI Scripts used by your app.



To edit your Script, open the AIModel Editor. If your Script is not displayed, double-click on it in either the Game Editor or the Data Explorer. Also, you can drag & drop the Script from the Data Explorer to the AIModel Editor.

YOUR FIRST SHIVA APPLICATION

Now, to have some action occur at the loading of the AI Model, you need to amend the “onInit” function. To do this you need to access the “Handler” for the AI Model, by left-clicking on the “+” next to “*Add Handler...*” under the “[Handlers]” heading, and navigating to the “onInit” Handler:



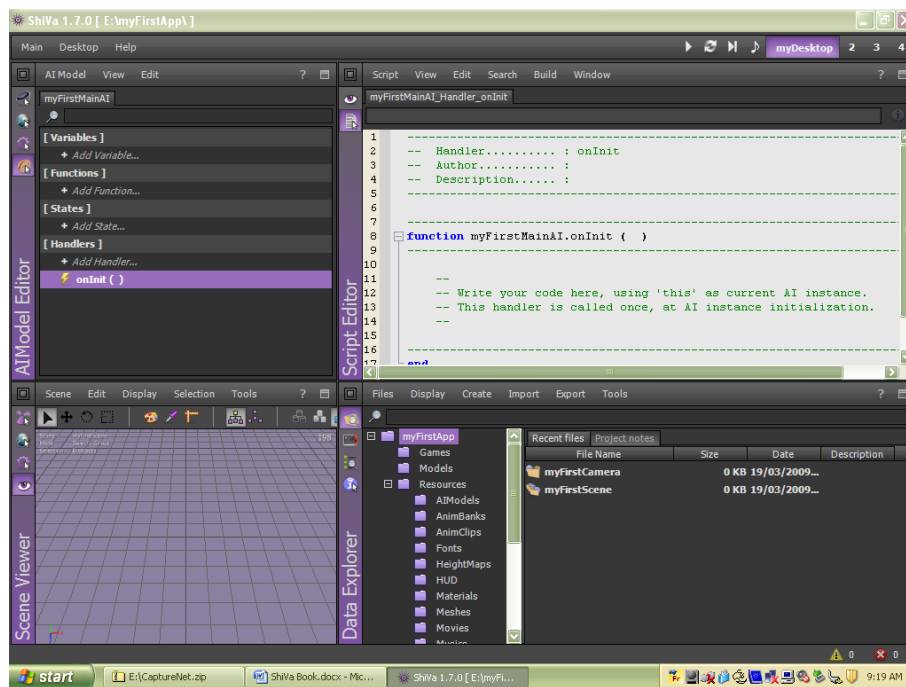
“Handlers” are just another term for a specific type of Script that “Handles” some Event, in this case the “onInit”, or “On Initialisation” Event.

Once you've clicked on the “onInit” Handler, you will see that the Handler has been added below the “[Attributes]” heading in the AI Model Editor, and also that the Script has been opened in the Script Editor. If you can't see the Script Editor, open it up now.

In the Script Editor, you will notice that a dummy Script has been opened up for you. This Script has a header that allows you to put in your name and the description of the Script, and also a preformed body that has one Function – `myFirstMainAI.onInit`. Note that the name of this Function is taken from the AI Model's name (`myFirstMainAI`) and the selected Handler (`onInit`).

It is **VERY IMPORTANT** that you do not change anything other than lines beginning with “--” or blank lines. If you do, the chances are that your Script will not work.

YOUR FIRST SHIVA APPLICATION



Now to your very first line of StoneScript.....

In the Script Editor, write the following line of code:

application.setCurrentUserScene (“myFirstScene”)

If you called your Scene something else, replace “myFirstScene” with whatever you named your Scene.

This line of code basically tells your application to open the Scene in the brackets whenever it first starts up.

You will notice that, whilst typing, the Script Editor tries to save you time by offering you options to complete what you are typing. This can be very handy for beginners, as it will prompt them with possible options, and possibly help prevent any typing errors from creeping in (which can be very annoying when debugging a non-working Script!).

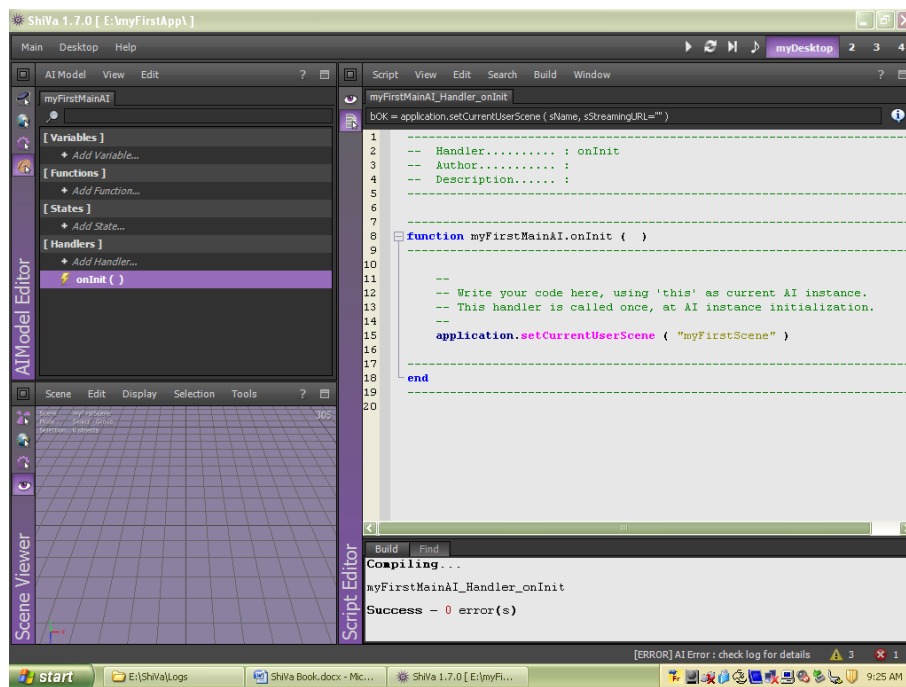
Now, all you have to do is save your Script, by either clicking on “Script” and “Save”, or by using “Ctrl-S” on your keyboard.

Tip!

ShiVa has many keyboard shortcuts – check the docs for a full listing.

Finally, press the “F7” key on your keyboard. This key validates your code, and displays any errors, or otherwise, in the “Output Window” of the Script Editor.

YOUR FIRST SHIVA APPLICATION



That's it! Your first Script is created, and you are ready to run your app for the very first time, but first take another look at the Script. You will notice that ShiVa has very kindly colour-coded the script. This is done to help you quickly locate specific types of code, and also helps in debugging when something may stand-out as looking wrong.

The colour-coding used by ShiVa is as follows:

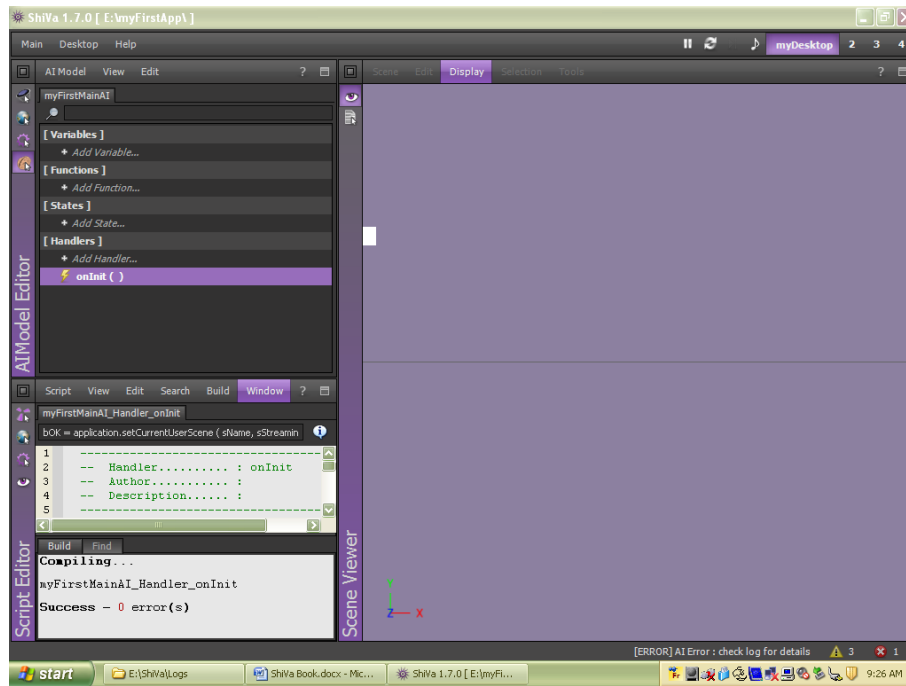
Standard Lua Functions
ShiVa Specific Function Type
ShiVa Specific Function Name
ShiVa Specific Constant
Comments
Shiva Names

So, the time has come, your first app is ready for its first run.....

Navigate to the Scene Viewer and either click on the Play button () or press “F9” on your keyboard.

ShiVa will ask if you want to save your Scene (if you haven't already done so), click on “Yes” and then you will see your app running....

YOUR FIRST SHIVA APPLICATION



Not very exciting is it? But what did you expect?

Click on the Pause button (⏸) and then on the Reboot (🔄) button, and you will now see a plain black screen in the Scene Viewer.

The main reasons that this run wasn't exactly exciting was that the Camera was placed at point (0, 0, 0) in your Scene, and, more importantly, that there wasn't anything to look at!

Now, we will proceed to add some intelligence to the Camera, and maybe even a Model to look at?

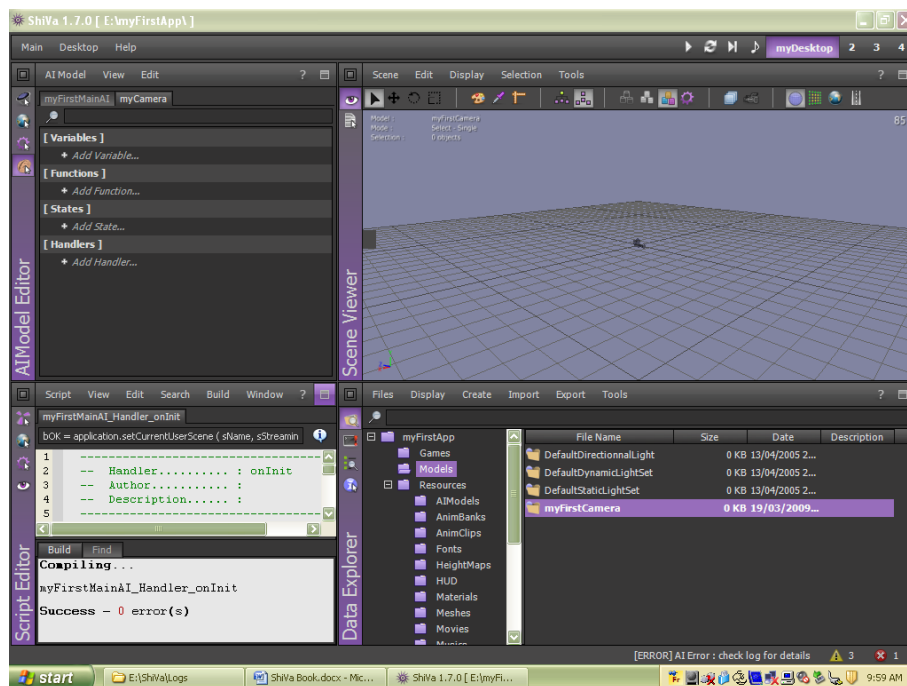
THE CAMERA SCRIPT

To make things more interesting, you need to create a Script to tell the Camera to do something. So, create a new AIModel (name it something like “myCamera”).

Next, you need to attach it to your Camera. To do this you need to open the Camera in Model mode by double-clicking the Camera (“myFirstCamera”) in the Data Explorer. Your Scene will reappear in the Scene Viewer, and you'll see a big Camera sitting there.

Important!

Do NOT drag your Camera to the Scene Viewer with a Scene open as this will just add an instance of the Camera to the Scene.



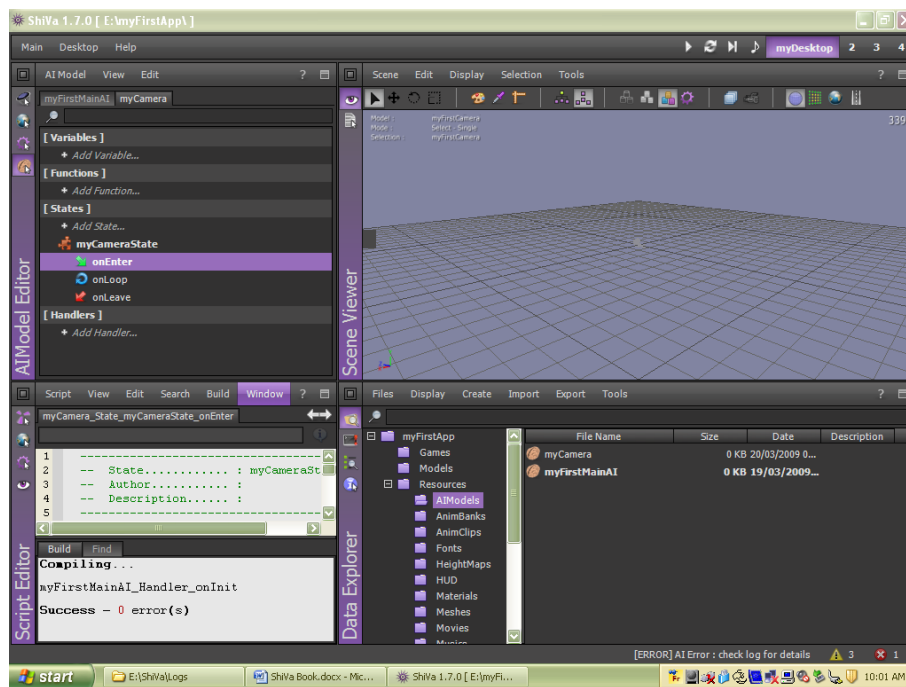
To attach your AI Model to the Camera, select the Camera in the Scene Viewer by clicking on it (it will turn light grey), then drag & drop the newly created AI Model onto the Camera.

Now you can save your Scene by clicking “Scene” and then “Save” in the Scene Viewer.

You will now give the Camera some intelligence, by getting it to rotate around the central point (0, 0, 0). To enable the Camera to do this, you must open the “myCamera” AI Model in the AIModel Editor, and add a State to the Model. This is done in much the same way as the “onInit” Handler, only this time you will need to select “Add State” rather than “Add Handler”. Name the new State (“myCameraState”??), and click on “OK”.

You will now see that three States have appeared in the AIModel Editor underneath the name you gave the State above.

YOUR FIRST SHIVA APPLICATION



The three States are:

- onEnter
- onLoop
- onLeave

These States are pretty self-explanatory, as they basically run when you start (onEnter), loop (onLoop), or stop (onLeave) the AIModel. The one's you'll be using for this part of the book are the “onEnter” and “onLoop” States.

So, open up the Script Editor, and you should have the “onEnter” Script template visible. Add the following couple of lines of code:

```
object.setTranslation ( this.getObject ( ), 3, 3, 3, object.kGlobalSpace )  
object.lookAt ( this.getObject ( ), 0, 0, 0, object.kGlobalSpace, 1)
```

Important!

All code **MUST** be written between the function header:

```
function myCamera.myCameraState_onEnter ( )
```

and the “end” line .

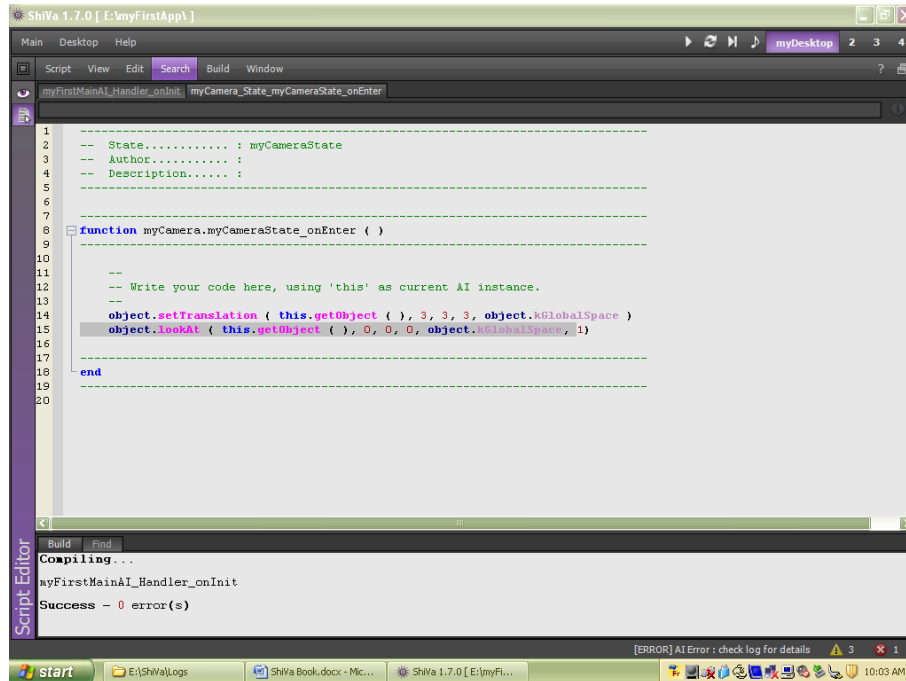
The first line of this Script moves the Camera to the point (3, 3, 3) in GLOBAL space, and the second line makes the Camera look at the point (0, 0, 0) in GLOBAL space.


YOUR FIRST SHIVA APPLICATION

Before we go any further, I think I'd better explain the difference between "LOCAL" and "GLOBAL":

LOCAL: Any changes made in LOCAL space will be applied directly to the Model, and will NOT affect the Model's position in GLOBAL space.

GLOBAL Any changes made in GLOBAL space WILL affect the Model's position in the 3D environment.



Note that I have maximised the Script Editor, by clicking on the () button, as I could not read the entire line of Script on my laptop's screen. These buttons in the top right of each Module are identical in operation to the standard window maximise and restore buttons.

Don't worry too much about the scripts at the moment, as we'll go into them in a lot more depth when we create our Dino Game.

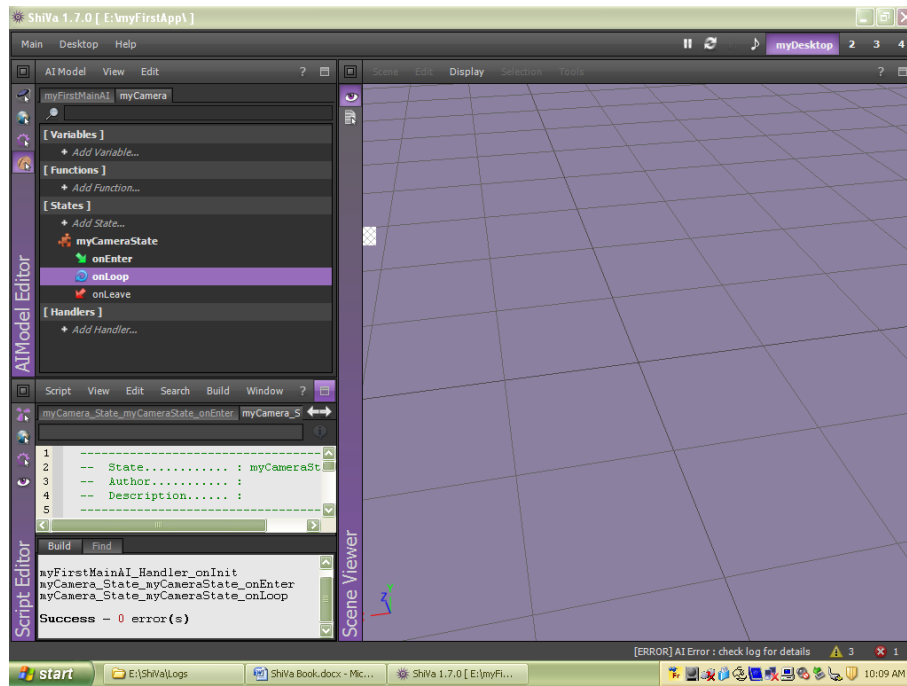
Next, change to the "onLoop" Script by double-clicking on it in the AIModel Editor, and enter the following two lines of code:

```
object.rotateAround ( this.getObject ( ), 0, 0, 0, 0, 1, 0, object.kGlobalSpace )  
object.lookAt ( this.getObject ( ), 0, 0, 0, object.kGlobalSpace, 1)
```

The first line rotates the Camera around the point (0, 0, 0), and the second ensures it looks at the point (0, 0, 0).

Save and Compile your Scripts, and when you run your app the Camera should now rotate around the centre point.

YOUR FIRST SHIVA APPLICATION



Over the next couple of pages, I'll go into a scripting a bit more, so feel free to skip this and come back later if you want to get on with creating your first app.

MORE ABOUT SCRIPTING

StoneScript is based on the LUA language and, as is common with most languages, identifiers can only consist of letters, numbers and the “_” character. Also in common with other languages there are some words that CANNOT be used as identifiers. These words are:

and
break
do
else
elseif
end
false
for
function
if
in
local
not
or
repeat
return
then
true
until
while

Another feature common to a lot of languages is that StoneScript is case sensitive, so “player” and “Player” are NOT the same.

Tip!

Always name your identifiers in a meaningful way. For instance, naming the main character “Player” is much more meaningful than naming it “aaa”.

Finally, identifiers beginning with “_” and followed by UPPER case letters (e.g.: _VERSION) are reserved for internal variables used by StoneScript, and should not be used.

Other tokens used within StoneScript are:

+ - * / ^ = ~= <= >= < > == () { } [] ; : , . ..

and, numbers can be written in many ways:

3 3.0 3.1416 314.16e-2 0.31416E1 etc.

One very important aspect of all programming is to COMMENT your code (makes it a lot easier to read for others, and also for yourself when you come back to it a year later!). Commenting in ShiVa can be done in two ways depending on how long your comment is going to be:

Short Comment use the double hyphen (--). This allows you to comment to the end of the current line

Long Comment again, use the double hyphen, but this time place “[[” immediately after it. This then allows you to comment until the code comes across “]]”. This is useful for commenting out blocks of code whilst testing/debugging etc.

As this is only a very quick intro to scripting, I'm just going to mention a couple more things, and then you can get on with the business at hand.... Creating your first app!

Firstly, StoneScript has two types of Variables available, LOCAL and MEMBER:

LOCAL LOCAL Variables are dynamically typed, so you don't have to specify what type of value they contain. There are five LOCAL Variable types (**nil, boolean, number, string & handle**).

NB: before the first assignment is made to a LOCAL Variable, its value is “nil”.

MEMBER MEMBER Variables are statically typed, and cannot be created at runtime, There are six MEMBER Variable types (**boolean, number, string, object, table & hashtable**).

NB: To obtain the value of a MEMBER Variable you have to use the following syntax:

local a = this.nVar()

and to set a value, you have to use this syntax:

this.nVar(12)

Finally, I would like to mention string concatenation, i.e.: the joining of strings...

String concatenation is denoted by using two dots (..) as shown below:

```
local a = "my"
local b = "first"
local c = "string"
local result = a..b..c                      -                      "myfirststring"
local result = a.." " ..b.." " ..c -                      "my first string"
```

Now for some examples:

- 1) Rotate an Object by 45 degrees around the GLOBAL Y axis:

object.rotate (this.getObject (), 0, 45, 0, object.kGlobalSpace)

- 2) Change the opacity of the Mesh attached to an Object:

shape.setMeshOpacity (this.getObject (), 0.5)

- 3) Play the first Sound attached to an Object:

```
sound.play ( this.getObject ( ), 0, 1, false, 0.5 )
```

- 4) Send a message to an Object:

```
object.sendEvent ( hTargetObject, "AIModelName", "onHandlerName", ... )
```

- 5) Obtain a reference to the current Scene:

```
local s = application.getCurrentUserScene
```

There is one last thing I should mention with regards to scripting in ShiVa, and that is the three types of scripts available (Functions, States and Handlers).

All scripts are really Functions, but some are called Handlers as they are usually attached to an Object or a User, and there is a special case for States.

Functions are “self-contained software routines that carry out a specific task”. Values can be passed in to Functions and a value may, or may not, be returned.

States are used to define particular AI actions (such as “eating”, “sleeping”, “walking”, “running” etc.) and, in ShiVa, have three possible scripts:

| | | |
|----------------|---|--|
| <i>onEnter</i> | - | executed once, when the State is first called. |
| <i>onLoop</i> | - | continually executed after the “ <i>onEnter</i> ” script, until the State is exited. |
| <i>onExit</i> | - | executed once, when the State exits. |

Handlers come in four distinct flavours (three predefined, and one User), and are used to respond to Events:

Common Predefined

These are common to both an Object and a User, and consist of:

| | | |
|---------------------|---|---|
| <i>onInit</i> | - | executed once at AIModel initialisation for EACH instance of the AIModel. |
| <i>onEnterFrame</i> | - | executed once per Frame for EACH instance of the AIModel. |

Object Specific Predefined

These are available ONLY if the AIModel is linked to an Object, and consist of:

| | | |
|--------------------------|---|---|
| <i>onSensorCollision</i> | - | executed when a Sensor attached to an Object collides with a Sensor attached to another Object. |
| <i>onActivate</i> | - | executed when an Object is activated. |
| <i>onDeactivate</i> | - | executed when an Object is deactivated. |

User Specific Predefined

These are available ONLY if the AIModel is linked to a User, and consist of:

| | | |
|---------------------------|---|--|
| <i>onJoypadStickMove</i> | - | executed when the stick of a Joypad is moved. |
| <i>onJoypadButtonDown</i> | - | executed when a Button on a Joypad is pressed. |
| <i>onJoypadButtonUp</i> | - | executed when a Button on a Joypad is released. |
| <i>onJoypadMove</i> | - | executed when a Joypad moves. |
| <i>onJoypadIRMove</i> | - | executed when a Joypad's IR receptor moves. |
| <i>onKeyboardKeyDown</i> | - | executed when a Key on the Keyboard is pressed. |
| <i>onKeyboardKeyUp</i> | - | executed when a Key on the Keyboard is released. |
| <i>onMouseMove</i> | - | executed when the Mouse is moved. |
| <i>onMouseWheel</i> | - | executed when the Wheel of the Mouse is moved. |
| <i>onMouseButtonDown</i> | - | executed when a Button on the Mouse is pressed. |
| <i>onMouseButtonUp</i> | - | executed when a Button on the Mouse is released. |
| <i>onUserEnterSession</i> | - | executed when a User connects to a Server Session. |
| <i>onUserLeaveSession</i> | - | executed when a User leaves the same Server Session. |
| <i>onUserEnterScene</i> | - | executed when a User enters the current Scene. |
| <i>onUserLeaveScene</i> | - | executed when a User leaves the current Scene. |

Custom

Custom Handlers can be called from Scripts in various ways depending on the type of AIModel they are linked to. These types of Handler can be called using:

| | | |
|--|---|-------------------------------|
| <i>object.postEvent</i> or <i>object.sendEvent</i> | - | AIModel attached to an Object |
| <i>user.postEvent</i> or <i>user.sendEvent</i> | - | AIModel attached to a User. |

All three types of script can access the Variables for the AIModel. However, there are some limitations as to how they interact with each other:

Functions can call States directly, but can only call Handlers using “sendEvent” or “postEvent”.

Handlers can call both Functions and States directly.

States can call Functions directly, but can only call Handlers using “sendEvent” or “postEvent”.

But wait! What are “Events”?

Events in the ShiVa world, are the basic means of passing information between Functions, States and Handlers. An Event is just an action that has been caused to occur by either ShiVa (such as updating the Scene etc.), or the user (such as moving the mouse etc.).

Well, that's enough for this introduction to scripting, so let's get on with the app.....

YOUR FIRST SHIVA APPLICATION

ON WITH THE SHOW.....

Now it is time to add something to look at in your Scene.

Since ShiVa doesn't have its own modelling tool, you will have to find your own. However, there are plenty of free modelling tools available over the net, such as Blender, Google Sketchup, TrueSpace, Wings 3D & the XSI Mod Tool. All of which support the Collada format used by ShiVa.

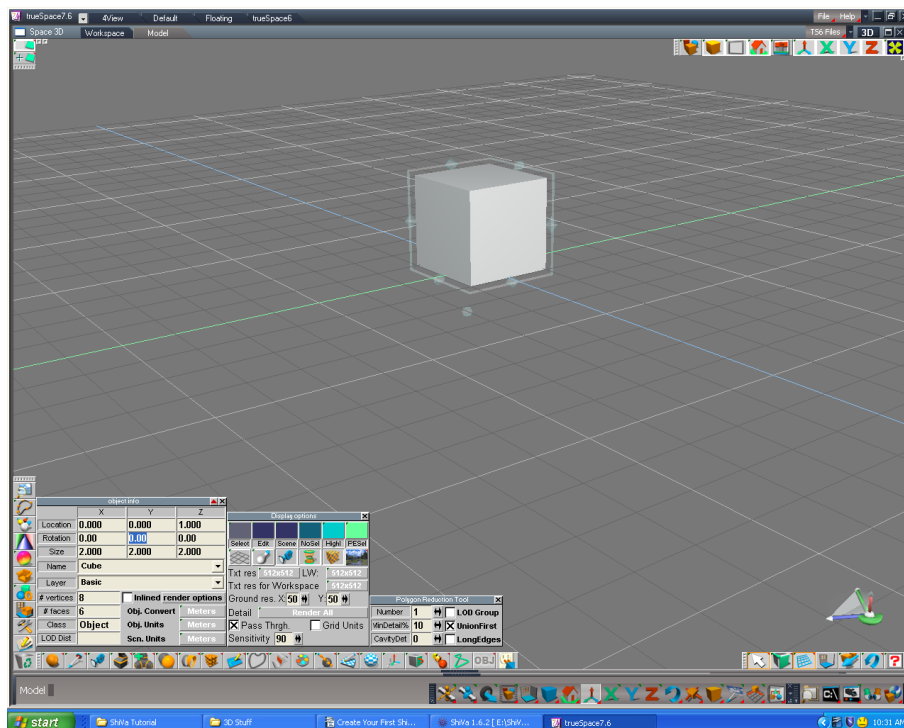
For this Model we will use TrueSpace, though it'll be pretty much the same in each of the others.

So, fire up your modelling program and let's get started.....

All you need to do for your first app is to create a simple shape. I'm going to create a red cube.

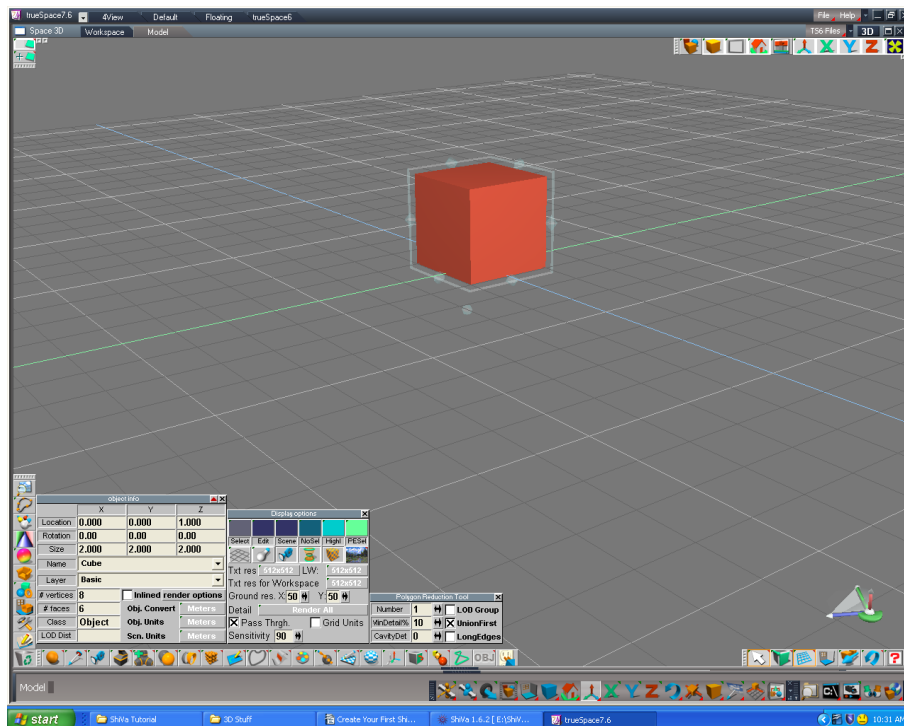
OK, let's go...

I'm not going to explain how TrueSpace works, as that would take a book in itself! Basically all you have to do is add the cube primitive to a new scene. It should be positioned at 0, 0, 1:

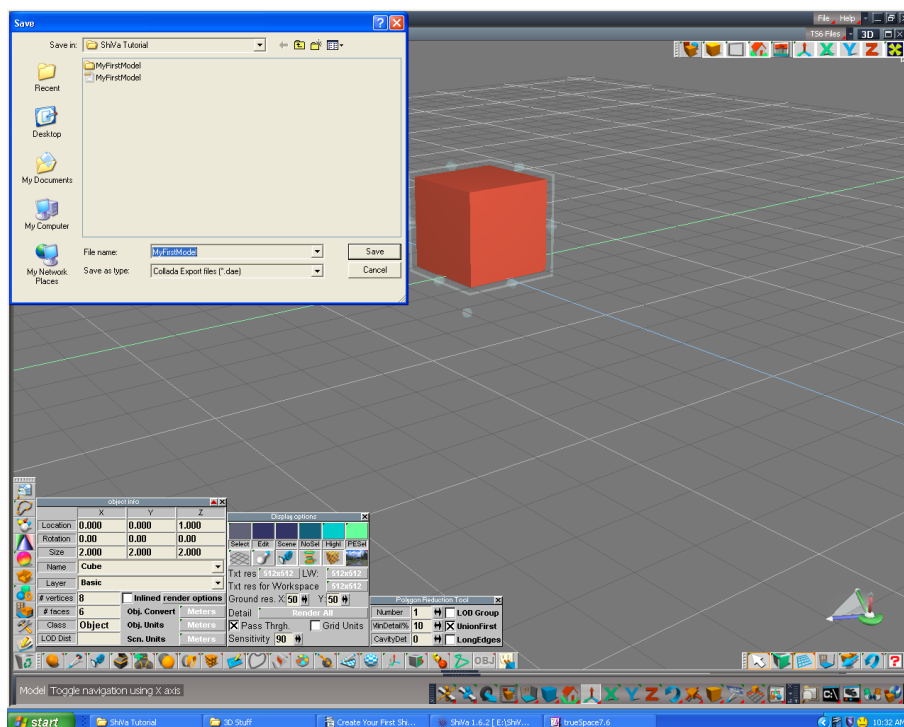


Now we have our base Model, I'll add the colour so it doesn't look too boring.

YOUR FIRST SHIVA APPLICATION



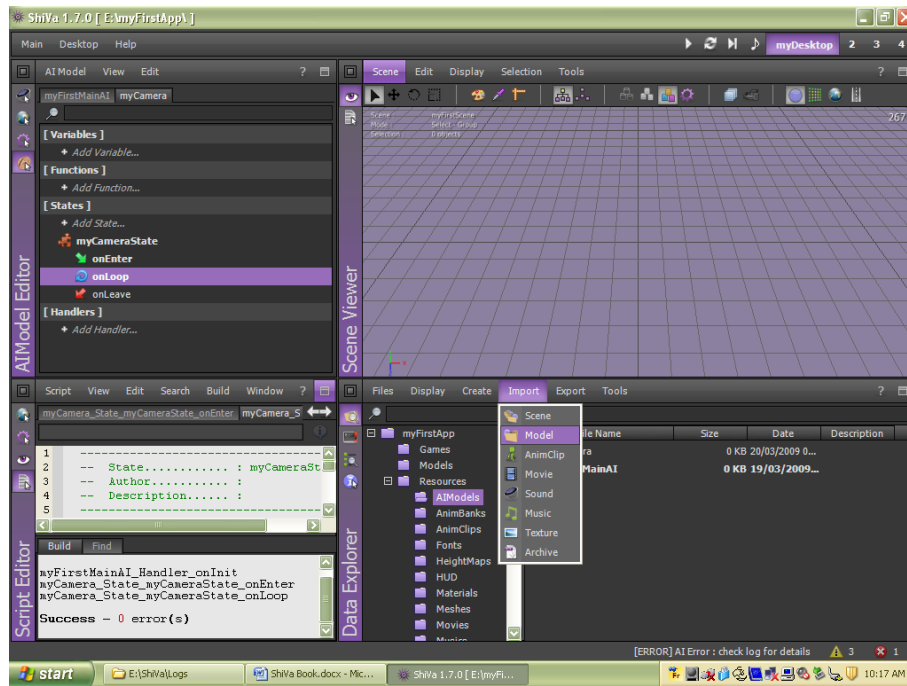
Now we need to save the Model, I'll call mine “myFirstModel”, as a Collada Model.



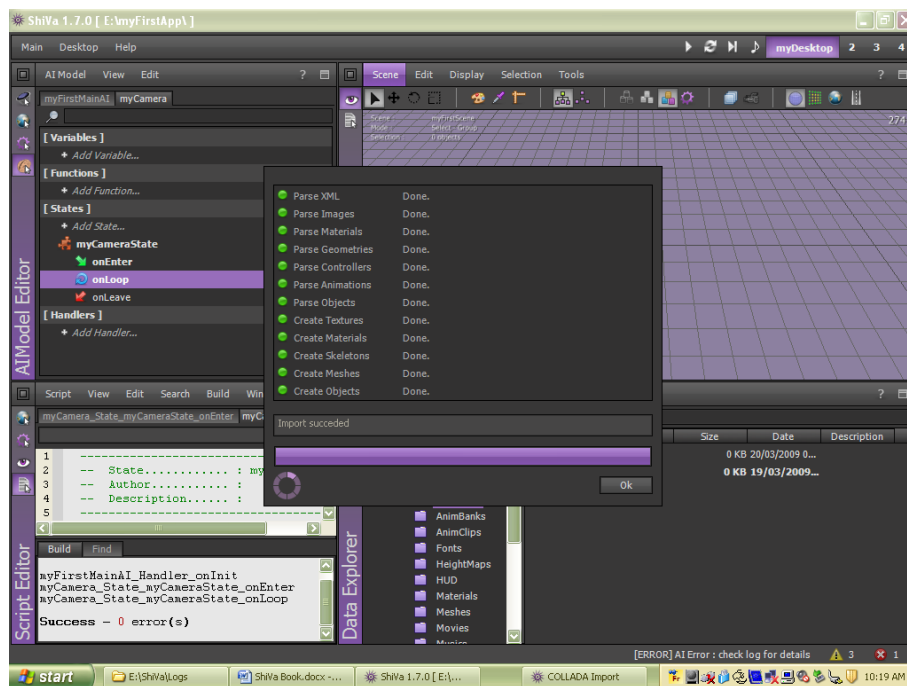
We should now have a Model sitting in a directory somewhere on our computer.

Next, switch back to ShiVa and import the Model, using “Import” and “Model” from the Data Explorer toolbar:

YOUR FIRST SHIVA APPLICATION

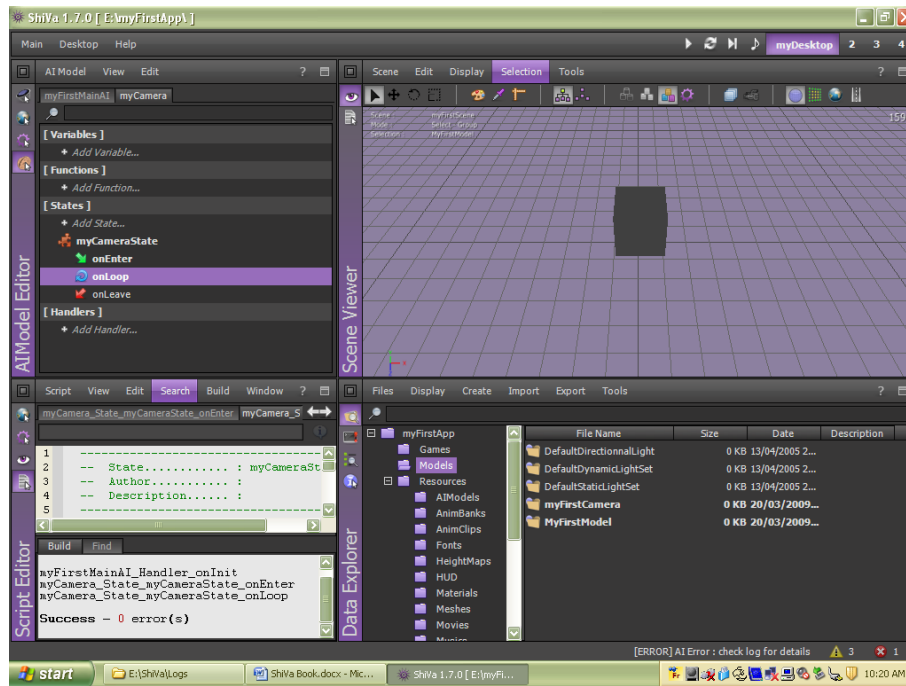


ShiVa will pop-up a dialog box showing the import process and, assuming all goes well, will let you know that the import succeeded:



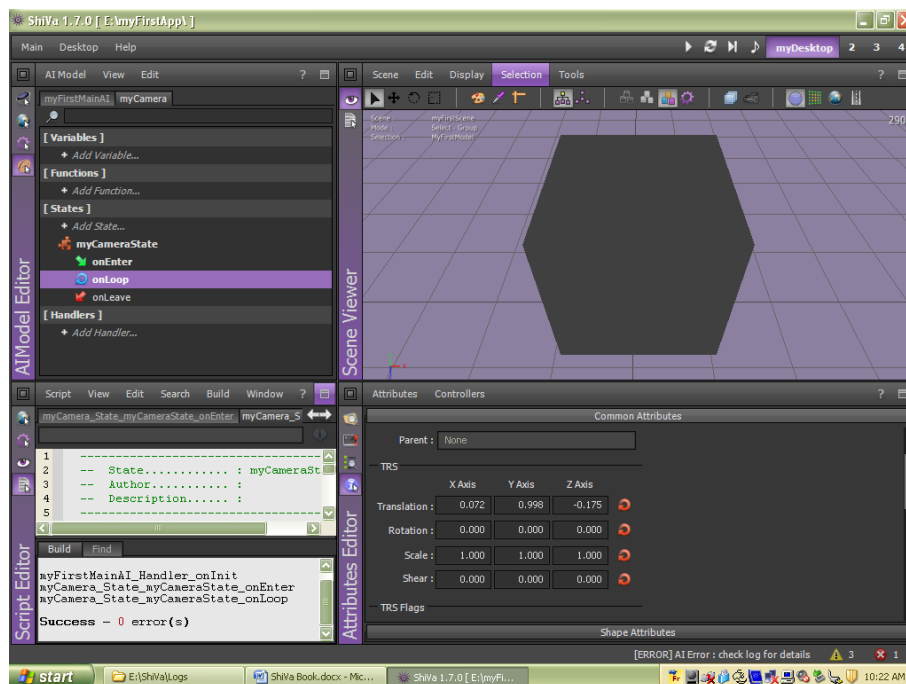
Finally, we need to add the Model to the Scene. This is done by dragging & dropping the Model straight into the Scene Viewer.

YOUR FIRST SHIVA APPLICATION



Unfortunately, it seems that the exporter in TrueSpace hasn't exported the colour! This can be a problem with some modelling programs, and the only way to tell is by trial and error, or searching the forums to see if others have had any problems and maybe come up with solutions! However, for our purposes, you can always add the colour in ShiVa, which you will be doing later on.

Firstly though, you need to check the Attributes of the Model so that they are correct. Open the Attribute Editor and scroll down to the section marked TRS. This will show the current Translation, Rotation & Scaling of your Model (i.e.: its Position, Rotation & Size):



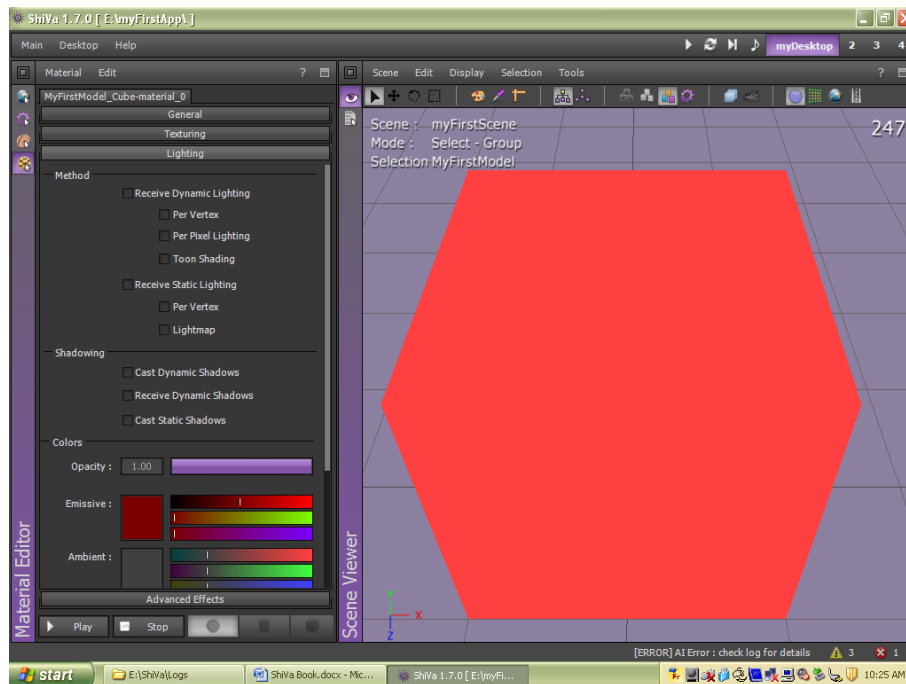
In my case, all looks OK, except for the Translation, which is showing 0.072, 0.998, -0.175, when I want 0, 1, 0! So, click in the X Axis box and change the value to 0, then in the Y Axis box and change this to 1, and finally change the Z Axis box to 0. Your Model should now be in the centre of your Scene. You can also use the “Tab” key to move between boxes.

YOUR FIRST SHIVA APPLICATION

Save your Scene, and you will be ready for the next part, the colouring of your Model.

To colour your Model, you will need to load up the Material Editor, and double-click on the Material for the Model in the Data Explorer (MyFirstModel_Cube-material_0):

In the Material Editor, select the “Lighting” tab, and change the RED “Emissive” value to approx. half-way:

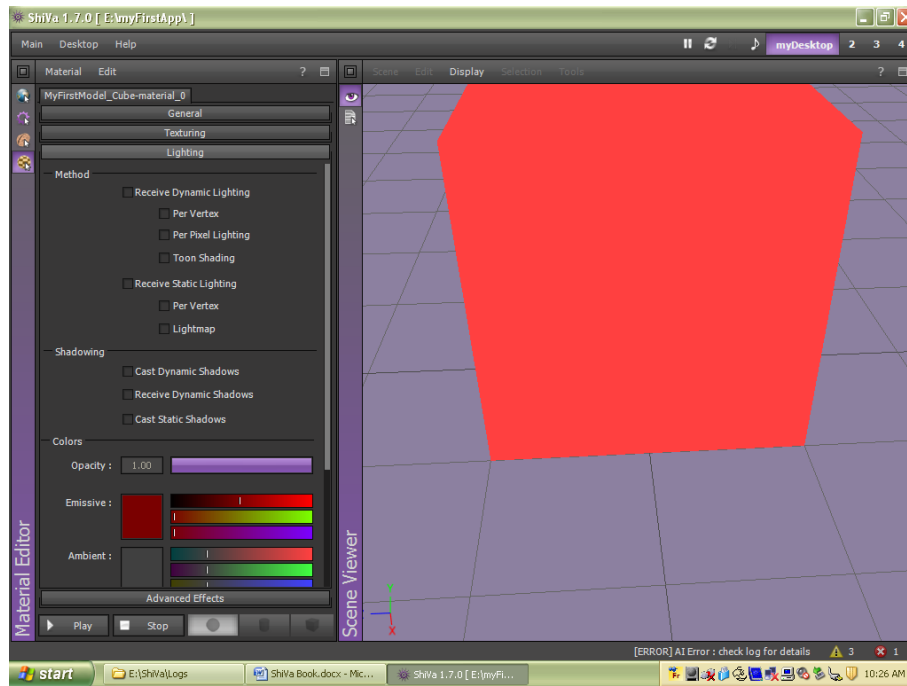


This is a bit of a cheat, but for now it will do!

You should now have a nice red cube sitting in the middle of your Scene.

Try running your app again..... (Don't forget to save your Scene!)

YOUR FIRST SHIVA APPLICATION



You should now have a Camera that rotates around your Model.

After testing your app, you will have to be able to turn it off. This is done by clicking the “Pause” button, followed by the “Reboot” button.

Important!

It is vital to always use the “Reboot” button. If you don't then your app does NOT reset itself. What this means is that if you edit one of your Scenes whilst running your app (nearly always the case) then you will lose your original Scene setup. Many hours can be lost in recreating your Scene following such an occurrence.

Well, that pretty much wraps up this part of the book. In the next Chapter I'll be introducing “Egg_Beak”, an animated Model courtesy of stoneTrip.