The Soya HandBook

Bertrand 'blam' LAMY (blam@tuxfamily.org)
Jean-Baptiste 'Jiba' LAMY (jiba@tuxfamily.org)

Version 0.1

Part I

Introduction

1 What is Soya?

Soya is a Python 3D engine for use in games.

Soya is specialy designed to be:

- very high-level and object oriented
- light weight (start time should be as short as possible)
- generic: this means that you must be able to make any kind of games with it, from FPS, jump'n run, shoot them up to real-time strategy (RTS), or even non-game application (*e.g.* 3D modeler, chemical app,...). If some possibilities lack for making your game, please email us!

2 Why shall I code my game in Python?

Traditionnaly games are coded in C or C++. There are 2 reasons for this:

1. many coders know C/C++ but not Python,
2. many people believe that Python (or scripting langages in general) is too slow for games.

Surely Python is quite slow to execute compared to natively compiled langages, but as the rendering process of Soya is in C, only the top-level function calls are in Python. A little personal story: I was speaking with a coder that wanted to used a real physics engine in his MMORPG (coded in C). I tell him the physics engine will slow down his game and that he will get few advantages of it (a physics engine will be more interested for a race game ;-). He responds that it doesn't matter cause the power of the computer will increase. Here is my conclusion: if you accept to loose speed for a physics engine, you can accept to loose speed for Python, it's just the same.

Here are the advantages of using Python instead of C/C++:

1. Python is easier to learn and you will get all its cool features (garbage collector...),
2. Python is very fast to code (you will write much fewer lines of code compared to C/C++). This means you will soon get a playable version of your game (and this is very motivating),
3. Python is portable.

Part II

Soya Base Reference

3 Soya classes

3.1 Introduction

Basically there are 2 kinds of objects: Shape and 3D Element objects. The main differences are:

- 3D Element objects can be added into a soya3d.World as child whereas Shapes can be added into soya3d.World or soya3d.Volume as shape,
- 3D Element objects have a matrix and so are 3D objects that can be handled.

Shape objects	model.Shape, model.Bonus, land.Land
3D Element objects	soya3d.Volume, soya3d.World, soya3d.Light, soya3d.Portal, soya3d.Sprite, soya3d.CylinderSprite, particle.

3.2 3D Element objects

3.2.1 3D Element objects generic functions

All 3D Element objects have the following set of functions. Since there are a lot of functions to move 3D Element objects in Soya, some explanations are needed.

```
element.set_identity()
```

Reset the 3D Element position and orientation.

```
element.set_xyz(x, y, z)
```

It just does what you think (ie element.x = x, element.y = y, element.z = z). x, y and z must be floats.

```
element.translate(x, y, z)
```

Translate the 3D Element of `Vector(element.parent, x, y, z)`. x, y and z must be floats.

```
element.shift(x_factor, y_factor, z_factor)
```

Equivalent to 3 translations with the following vectors: (element, x_factor, 0.0, 0.0), (element, 0.0, y_factor, 0.0), (element, 0.0, 0.0, z_factor). Typical usage of this function is element.shift(0.0, 0.0, 1.0) which means that the element will advance along its own Z axis (usefull for camera movement).

x_factor, y_factor and z_factor must be floats.

```
element.scale(x_factor, y_factor, z_factor)
```

Respectively multiply the X, Y and Z axis vector units by x_factor, y_factor, z_factor.

x_factor, y_factor and z_factor must be floats.

```
element.rotate_lateral(angle)
```

Perform a rotation of angle degrees along the Y axis of element.parent.

angle must be a float.

```
element.rotate_vertical(angle)
```

Perform a rotation of angle degrees along the X axis of element.parent.

angle must be a float.

```
element.rotate_incline(angle)
```

Perform a rotation of angle degrees along the Z axis of element.parent.

angle must be a float.

```
element.turn_lateral(angle)
element.turn_vertical(angle)
element.turn_incline(angle)
```

Same as the rotate family functions but the rotation is performed along the X, Y or Z axis of element. Typically, the `turn_*` are used for cameras, and the `rotate_*` are used for other elements.

```
element.rotate_axe(angle, x, y, z)
```

Perform a rotation of angle degrees around an axis defined by Vector (element, x, y, z). angle, x, y, and z must be floats.

```
element.rotate(angle, x1, y1, z1, x2, y2, z2)
```

Perform a rotation of angle degrees around an axis defined by the 2 Points (element, x1, y1, z1) and (element, x2, y2, z2). angle, x1, y1, z1, x2, y2 and z2 must be floats.

```
element.look_at(object)
```

object must have x, y, z and parent attributes. object is often a Point or a Vector. If object is a Vector, this function orientate the Z axis of element in the vector direction. Else element Z axis is oriented into the direction of the vector resulting by (point - element origin).

```
element.distance_to(object)
```

Return the distance between element's origin and object. object must have x, y, z and parent attributes.

Part III

Soya game-specific features

4 Real-Time Strategie game

4.1 Fog of war

Part IV

Setting up a Soya-based app

5 Creating the directory structure

First, create a directory for your app (that's obvious), say `./app`. Soya needs some directories to store its data ; it use a different directory for each type of data (puting all the data in the same directory is NOT a good idea!). There is one directory for images (also called textures in 3D), one for materials (surface properties, including textures), one for models (World in Soya) and one for optimized models (Shape in Soya).

So the next step is to create these directories: `./app/images ./app/materials ./app/worlds ./app/shapes`. You may add `./app/sounds` if you plan to use sound support.

6 Initializing your app

Then you can create the first script of your app (`./app/app.py`), and add initialization code similar to this one:

```
import os, os.path, sys
import soya, soya.soya3d as soya3d, soya.model as model

HERE = os.path.dirname(sys.argv[0]) # get the directory were this script lies
model.Image   .PATH = os.path.join(HERE, "images")
```

```
model.Material.PATH = os.path.join(HERE, "materials")
model.Shape   .PATH = os.path.join(HERE, "shapes")
soya3d.World  .PATH = os.path.join(HERE, "worlds")
```

After this initialization, you can use `model.Image.get("my_image")`, `model.Material.get("my_material")`, `model.World.get("my_model")`, `model.Shape.get("my_model")` to load the corresponding object (if the object is already loaded the same object is returned, use `load` instead of `get` if you want a new one).

To save a material or a world, do: `object.filename = "my_object"; object.save()`. The object will be saved in the corresponding directory, with an additional `.data` extension. To delete an object, simply delete the corresponding file.

7 Drawing textures

Textures can be drawn in about any painting program, such as The Gimp. Preferred image file format for Soya is PNG (because it offers high quality, low weight, transparency and doesn't have any patent problem); TGA can be used too.

Simply put your textures in `./app/images`.

8 Creating materials

Materials can be easily created using the Soya Editor. Launch the editor by typing `soya_editor`. The first time you use it, you need to configure the editor by entering the pathes of the four directories created at the beginning, then click on Apply.

To create a new material, click the File▷New Material... menu. A new window appears. Use the `tex_filename` listbox to choose the texture (the list propose the texture available in the `./app/images` directory), and set the material's name in the `filename` textbox (e.g. `my_material`). See the Material reference for the other properties. When finished, click the Pickle▷Save menu to save the material.

To open an existing material, click the File▷Load... menu in the editor's first window, and then choose the desired file in the dialog box (*e.g.* `./app/materials/my_material.data`). You can also give files as command line arguments.

9 Designing models (=Worlds)

Different tools can be used for designing 3D models. For Soya-oriented modeling, we must consider 2 types of model : animated models (*e.g.* a 3D character), and non-animated models (*e.g.* a sword). For animated models, Soya actually uses Cal3D model, and uses the Cal3D file format. Only non-animated models can be created and modified by Soya.

Soya's conventions for model orientation is to use X axis for the right direction, Y axis for upward direction and -Z axis for front direction (and so Z for back. Why -Z for front? because Z would yield an indirect coordinate system!).

9.1 Blender

Blender is a free software 3D modeler; it is also the preferred modeler for Soya. It can be used for both animated and non-animated models. You need at least Blender 2.28 for a complete support.

Blender conventions is to use Y axis as front direction and Z as upward direction; you should use Blender's conventions since the export scripts will transform the model from Blender to Soya conventions.

XXX add a short Blender lesson here? Or at least a link to a good one?

9.1.1 Non-animated models

To export non-animated models, open a text editor in Blender (by clicking on the button at the lower left coin of a view) and load the `import_blender.py` script in Blender (this script is available in the Soya's source directory). Then modify the parameters at the beginning of the script. Parameters are :

`IMAGE_PATH`, `MATERIAL_PATH`, `WORLD_PATH`, `SHAPE_PATH` these are the four directories defined at the beginning.

`FILENAME` is the model's filename (*e.g.* `my_model`).

`EDIT` if set to 1, the model will be edited in the Soya Editor instead of being saved.

Save the model (to save the modified script together), and run the script (by taping Meta/Alt-p). The model is saved in `./app/worlds` (*e.g.* `./app/worlds/my_model.data`), and new materials (if there is any) are saved in `./app/materials`. The materials are named accordingly to the texture ; you can then edit them to set advanced material properties (Blender doesn't offer material properties corresponding to Soya one).

9.1.2 Animated models

To export animated models, you need to download the `blender2cal3d.py` export script on `http://oomadness.tuxfamily.org/en/blender2cal3d` (the script may be included in future versions of Blender, probably 2.29). Then load the script in Blender (see above), and modify the parameters at the beginning of the script. The interesting parameters are :

`EXPORT_FOR_SOYA` Setting this to 1 will rotate the model according to Soya's conventions. This is highly advised.

`SAVE_TO_DIR` is the directory were the model will be saved. Cal3D models are saved in a directory and not in a file ; since the Cal3D separates the model in several files.This directory must be a subdirectory of the shape directory, *e.g.* `./app/shapes/my_model`, where `my_model` is the name the Soya's shape name.

`RENAME_ANIMATIONS` is a dictionary mapping Blender's IPO names to Soya's animation names. This is needed because the Blender's action/animation names cannot be accessed yet. Typical Blender's IPO names are Action, Action.001, Action.002,... For helping you to find the right mapping, `blender2cal3d.py` prints animation names and durations. An exemple of mapping can be `{"Action" : "my_first_animation", "Action.001" : "my_second_animation"}`.

Now, run the script ! The model is now saved, though if you want to use material properties, you must create a material *with the same filename that the texture* (except the extension), for exemple if the texture is `./app/images/my_texture.png`, the material must be called `my_texture`.

9.2 The Soya Editor

Soya features its own editor ; though it is buggy, limited and unfinished... This editor is nice for quickly looking at the model or debugging Soya; it is not good at modeling :-(

Moreover, animated character cannot be designed with the Soya editor.

9.3 3D Studio Max, Milkshake

These modelers have pluggins for exporting in the Cal3D format, so you can use them to design animated models ; though they are not Free Software, and they have never been tested ! You cannot use them for non-animated models.

10 Optimizing (compiling) models (=Shapes)

Animated models don't need compilation (actually they are already compiled). Other model are automatically compiled when needed ; *e.g.* if you have a model saved as `./app/worlds/my_model.data`, and you ask for shape (=compiled model) `my_model` (by calling `soya.model.Shape.get("my_model")`), the non-compiled version is loaded, compiled, and the result is saved in `./app/shapes` (provided that the directory is writable). Futher call would read the compiled version until the non-compiled one has changed. This way of compilation is similar to the Python one.

So the basic sentence is "don't care about model compilation".

11 Sounds

Supported sound formats are Wave (for sound) and Ogg Vorbis (for music). Put both sound and music files in `./app/sounds`.