ANIMATOR

Service API for Rendering Software

FREDO6 – V1.1 – 26 MAR 2016

1. Introduction

Animator is a script dedicated to the animation of Sketchup models. It provides a parametric, interactive framework to control movements of objects and cameras along a timeline.

The animations, called ***'Clips'***, are built from a combination of unit transformations (called ***sequences***) such as translation, rotation, spin, scaling, explosion, etc, so that it is possible to **calculate accurately the exact position of objects and cameras for any frame of the video at any specified frame rate** (i.e. the animation is not based on the native Sketchup scene transition mechanism).

The **Service API for Renderers** provides various methods to drive the animation from an external script, in order to capture each frame of the animation for subsequent rendering.

2. Checking the presence of Animator

The API is implemented in the module **F6_Animator.** You can check the presence of Animator and its API by the following test:

```
defined?(F6_Animator.api_initialize)
```

There is a method to get the current version of Animator as a String, for instance "1.0a".

```
F6_Animator.get_version
```

3. Typical API workflow

The API is designed to be called from an external script without launching the Animator interactive tool itself. The API does not run as a Sketchup interactive Tool (i.e. Sketchup::Tool class). So the calling script can itself run as an interactive Sketchup::Tool.

As a prerequisite, you need two conditions:

- Not running the Animator interactive plugin
- the current model to include an animation clip (you can have several clips in the model)

The typical workflow includes the following steps:

1) **Create an Animator API object**: `@animator = F6_Animator.api_initialize`
   This method loads all information related to Animator clips from the model.

2) **Select a Clip and generation Options**: this is done via a **built-in dialog box** provided by the Animator API (`@animator.api_dialog_clip_options`).

3) **Processing the frames**: This is where you sequentially export the view and objects along the time line (`@animator.api_next_frame`). Alternatively, the external script can manage the calculation of frame timing and go through the animation by positioning it a given times by `@animator.api_goto_time` or `@animator.api_goto_frame`.

4) **Terminating the capture session**: `@animator.api_terminate`. This method is VERY IMPORTANT, as it puts back all objects in their original position and free up Animator. In normal processing, it is called implicitly however.

4. API Description

4.1. API Initialization

The first step is to get an API handle and load the Animator information from the model.

```
@animator = F6_Animator.api_initialize
```

The object returned is a class instance. It is to be used in all subsequent calls to the API.

Note that `api_initialize` returns `nil` in the following cases (displaying a message box):

- The current model is New (i.e. not saved)
- Animator interactive tool is currently running
- There are no playable clips in the model

4.2. API Termination

To finish the API session, you MUST call the Termination method. This will free up Animator and reset all objects to their initial position.

```
@animator.api_terminate
```

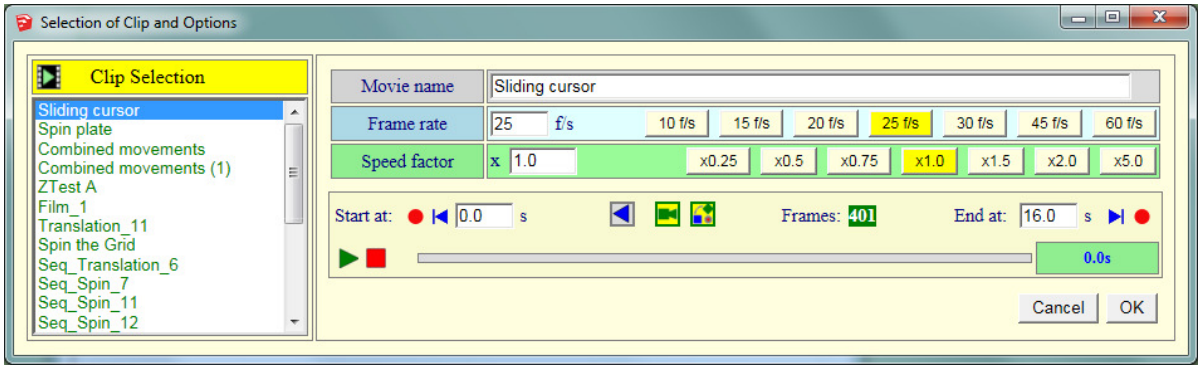
It is very important to call the termination method IN ALL CASES, in particular in case of errors, because all object movements are handled through the *move!* method, and therefore are not registered in the Undo stack.

Note that `api_terminate` is called implicitly at the end of the process based on `api_next_frame`. So the protection is really when the process fails unexpectedly.

For convenience, `api_terminate` can be safely called several times.

4.3. Setting the Clip and Options

The API provides a built-in method, `@animator.api_dialog_clip_options`, to let the user **select the Clip and related Options**.



Because web dialogs, even modal, are not blocking on Mac, the invocation of the dialog box uses a **callback** method (`return_proc(hinfo)`) to notify the caller when the dialog session has finished, either validating the input or cancelling the process.

So the call will look like:

```
@animator.api_dialog_clip_options() { |hinfo| return_proc(hinfo) }
```

…where the `hinfo` argument passed to the return method is a Hash Array or nil.

- `hinfo` gives information about the export parameters. The process could now start.
- `nil` if the user has cancelled the dialog box. The process should stop. Note that `api_terminate` is called implicitly.

`hinfo` is a Hash array with the following fields:

- `:video_name`: name given to the animation
- `:frame_rate`: frame rate (ex: 25, for 25 frames per second)
- `:number_frames`: number of frames to be processed
- `:frame_duration`: duration of a frame in second
- `:beg_time`: time of beginning of animation, as selected (in second)
- `:end_time`: time of end of the animation, as selected (in second)
- `:video_duration`: *nominal* total duration of the animation (in second), regardless of *beg_time* and *end-time*.
- `:speed_factor`: speed of the movie: 1 for normal speed, 0.5 for slower by 2, 2 for accelerated by 2.

Note that some parameters are for information and can be ignored when moving through the animation. However, the parameters `:number_frames` and `:frame_duration` are calculated taking into account the *frame_rate*, the *speed_factor* as well as the *beg_time* and *end_time*.

4.4. Processing the Frames in Sequence

Automatic Loop

Based on the selected options, you can go through each frame of the video by invoking `@animator.api_next_frame`, which returns:

- `[cur_time, inext_frame]`, current animation time and index of the current frame
- `nil` when no more frames need to be processed. The process should stop. Note that `api_terminate` is called implicitly.

Manual Loop

You can also go freely through each frame by invoking `@animator.goto_frame` or `@animator.goto_time`.

- `@animator.goto_frame(iframe)` will position the animation at the specified frame. Note that the **first frame is 1** (not zero), and that `iframe` should be comprised between 1 and `hinfo [:number_frames]`. With this method, frame 1 will correspond to the animation at *:beg_time* and the last frame at *:end_time*. If `iframe` is negative, it is counted from the end, that is, -1 means last frame, -2 the late before last, etc….
- `@animator.goto_time(anim_time)` will position the animation at the specified time. Note that the animation time `anim_time` should be comprised between 0 and `hinfo [:video_duration]`. Time is given in second. With this method, you are not bound to the specified *:beg_time* and *:end_time* parameters.

With manual methods, do not forget to call `api_terminate` when finished.

4.5. Summary of Full Process

Therefore, the whole rendering process could be done with the following pseudo-code:

```
def start_process
   @animator = F6_Animator.api_initialize
   return unless @animator
   @animator.api_dialog_clip_options() { |result| do_process(result) }
end


def do_process(result)
   #User cancelled the process in the dialog
   return unless result

   #Begin the Export process (ask other parameters, init progress bar, ….)
   …Start of the rendering process …

   #Processing the Frames in sequence
   while(@animator.api_next_frame)
      # at this point, view camera and objects are positioned in the current view
      begin
         …Export / Render the view and objects …
      rescue
         @animator.api_terminate
         …signal error …
         break
      end
   end
end
```

In practice however, **it is not advised to process the rendering as a *while* loop**, because it may not give a continuous visual feedback to the user (and update a progress bar)

It may be a good idea to use *UI.start_timer* or other asynchronous mechanisms to give back control to Sketchup so that the view is refreshed, as well as, to allow the user to interrupt the process, which can take quite long.

Note however that, in any case, after a call to `api_next_frame`, the view camera and objects are correctly positioned, whether this is or not reflected visually in the viewport.