CONVEXIFY 3D SHAPES

Service API

FREDO6 – 27 FEB 2015

1. Introduction

Convexify is a standalone script which decomposes arbitrary 3D shapes into Convex shapes.

The algorithm can however be invoked **as a service from an external Ruby script**, including from within an interactive Tool.

The Convex Decomposition is implemented in the class **Traductor:: SolidConvexify**, which is part of the **LibFredo6** library, **v6.7** and above (so FredoTools does not need to be installed). Therefore, you should protect the invocation of the API with a section:

```
if defined?(Traductor::Convexify)
  …
end
```

The second precaution is to make sure that you have committed (or aborted) any Sketchup operation before invoking the API. Convexify will operate within its own section:

```
Sketchup.active_model.start_operation "Convexify 3D Shapes"
 …
Sketchup.active_model.commit_operation
```

If there is an error, Convexify will trigger a `Sketchup.active_model.abort_operation`

As a result, you can cancel the whole convexify operation afterward with a single `Sketchup.undo` if it is successful**.**

2. Invoking Convexify

a) Creating the Convexifier

You first need to create an instance of the class Traductor::SolidConvexify. As an argument, encoded as a hash array, you indicate the method to be invoked when the Convexify operation is terminated:

```
hsh = { :notify_exit_proc => self.method("finish") }
@convexifier = Traductor::SolidConvexify.new hsh
```

The exit method, here called "finish", must have a single argument: `code`, which can take two value:

- `:exit` for normal termination
- `:abort` when an error occurred or the user aborted the operation

Note: it is important that you keep the convexifier instance in a class variable (so @convexifier) because the processing is asynchronous.

b) Executing the Convex Decomposition

There is a single call with 2 parameters:

```
@convexifier.convexify selection, hparams
```

`Selection` is the flat list of the shape geometry you wish to process. It is therefore a list of Sketchup entities such as **Faces**, **Groups** and **Component instances**. Other entities such as Edges, Guide points, etc… can be passed too but will simply be ignored. If you pass `nil`, then the current Sketchup selection will be taken, and if there is no current selection, the whole active model.

`hparams` is a Hash array containing the parameters for execution. For a 'silent execution', you probably wish to put the convex decomposition on a specified layer without touching the original shapes. The specified layer will be created if it does not exist.

So the recommended setting is:

```
hparams = { :use_layer => true, :layer_name => "my specified layer" }
```

Optionally you can indicate a Tolerance for concavity as the deviation in degree versus the flat angle. For instance, for a tolerance of 5.7 degree, `:tolerance_concave => 5.7`

If you wish to replace the original components, then just call the **`convexify`** method without a second argument or with { `:use_layer => false` }.

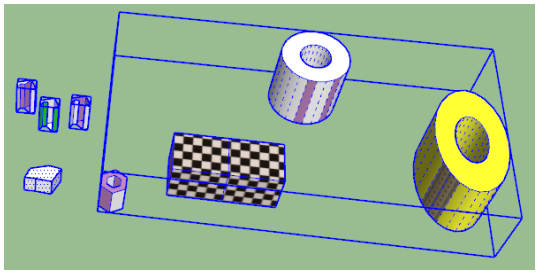
Note: If the processing takes long, a progress panel will be displayed, allowing the user to visualize the progression, but also to interrupt the operation.

c) Inspecting the resulting Convex Decomposition

When the processing is completed, your exit method (here **`finish()`**) is called with the parameter `code` indicating success or error / abort.

```
def finish(code)
   #Handling error and abort cases
   if code == :abort
      ….
      return
   end
   #Inspecting the convex decomposition
     Proceed with inspection, as explained in the following paragraphs
end
```

The results of the convex decomposition is provided as a list of **Compact Grouping structures**. A Compact Grouping represents a piece of the original selection including faces which are 'connected'. So an original group or Component Instance may correspond to several Compact groupings.



This selection corresponds to **8** Compact Groupings, out of

- A set of Faces at top level
- 3 Groups
- A Component instance embedding geometry and subgroups

A Compact Grouping includes the **correspondence between the original shapes and their convex decomposition**.

Internally it is described by

```
GroupingInfo = Struct.new :faces, :comp, :tr, :lst_convex, :parent
```

Typically, you would go through this information within a loop:

```
@convexifier.grouping_info.each do |grouping_info|
    …do what you have to do
end
```

Here is a description of the field of the GroupingInfo structure:

Field	Description
`:faces`	**List of the original faces making up the original Compact grouping**. These faces are normally valid since the original selection is untouched.
`:comp`	**Original group or component instance containing the Faces**. `comp` will be `nil` if the faces were at the top level of the active model.
`:tr`	**Top-level transformation of** `comp`, that is, the transformation of the internal <comp> coordinates to the active model. This is mainly useful for drawing in the viewport.
`:lst_convex`	**List of the Convex Groups corresponding each to a convex shape of the convex decomposition**. If the compact grouping was originally convex, then this is list is reduced to one element (which is a copy of the original faces of the compact grouping)
`:parent`	**Direct parent of the Convex groups**. All convex groups in `lst_convex` are 'brothers', i.e. at the same hierarchical level. Note that when the Convexify mode is to replace the original selection, then `parent` is equal to `comp`.

Normally, with the above information, you should be able to associate the original shapes to their convex decomposition, as well as to access the geometrical coordinates of the convex shapes created.

Note: the time of calculation, in second, is returned by a call to the method `@convexifier.calculation_time`.

d) Forgetting about the Convexify operation

Once you have all the information, you can restore Sketchup in its original state by simply calling `Sketchup.undo`.

3. Summary

To recap on the usage of the Service API, here is the condensed code that you shoud adapt to your need. This assumes that:

- you have checked that **`Traductor::SolidConvexify`** is defined
- you do not have a pending Sketchup operation.
- **`<selection>`** contains the list of original shapes in the parameters or is **`nil`** for the current Sketchup selection.

a) Invoking the Convexify operation

Using a separate layer

```
hsh = { :notify_exit_proc => self.method("finish") }
@convexifier = Traductor::SolidConvexify.new hsh
hparams = { :use_layer => true, :layer_name => "my specified layer" }
@convexifier.convexify selection, hparams
```

Replacing the original shapes

```
hsh = { :notify_exit_proc => self.method("finish") }
@convexifier = Traductor::SolidConvexify.new hsh
hparams = { :use_layer => false }
@convexifier.convexify selection, hparams
```

b) Post-Calculation method to review the results

```
def finish(code)
    #Handling errors or abort
    if code == :abort
       …probably report the error and stop
    end
    #Associate the original shapes and their convex decomposition
    @convexifier.grouping_info.each do |grouping_info|
       …use the GroupinInfo fields to build the mapping
    end
    #Restoring the original Sketchup state (optional)
    Sketchup.undo
end
```