

Blender Analytical Geometry



© May 2004 – Stefano Selleri a.k.a. S68
Vers. 0.0.5
‘La Bufera e altro’

Table of Contents

INTRODUCTION	3
PACKAGE CONTENT	3
DEPENDENCIES	3
INSTALLATION	3
USAGE	3
SIMPLE USAGE	4
POWER USAGE	4
SYMBOLIC EVALUATION	6
DEMO FUNCTIONS	7
SPHERE	7
TORUS	7
SPIRAL	7
COIL	8
SHELL	8
XML FILE	8
<BAG>	9
<REFERENCE>	9
<LABEL>	9
<F1> <F2> <F3>	9
<PARU> <PARV>	9
<USERPARA>	9
<MACRO>	10

Disclaimer

This software is provided 'as it is' no guarantees expressed or implied are given, except that it will occupy some space on your disk(s). The software, bag.blend and this document are © 2004 Stefano <S68> Selleri and released under Blender Artistic license which you should have received with the package. If this is not the case a copy of said license is available at www.blender.org or directly from the author selleri@det.unifi.it.

Introduction

Blender Analytical Geometry (BAG) is a Python script designed to crunch user defined mathematical formula defining parametric surfaces.

Package Content

The **BAG** package should contain:

1. **BAG.pdf** – This file you are reading ☺
2. **BAG.blend** – An example Blender file with the script within
3. **BAG-0.0.5.py** – The script, as a separate text file.
4. **Licence.txt** – The Blender Artistic License under which this package is given.

Dependencies

BAG needs the following softwares to be correctly installed on your computer to work:

1. An operating system of your choice ☺.
2. Blender 2.33 and maybe later version (© The Blender Fund - www.blender.org).
3. Python 2.3.3 or whatever Python is recommended with your version of Blender (© - www.python.org)

Installation

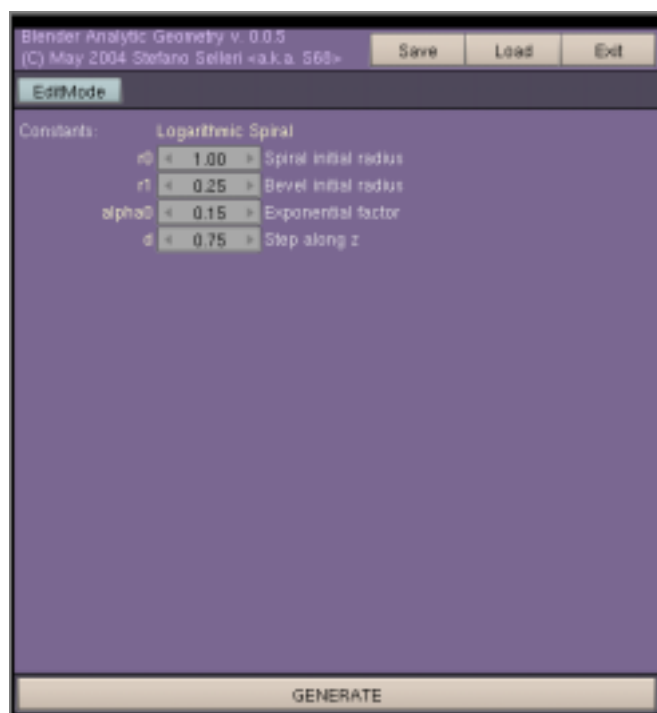
Unpack the package in a directory of your choiche.

Once you are done you can load the **BAG.blend** file of this package in Blender.

Usage

Place your mouse cursor on the left Blender window, the text window containing the **BAG** script and press **ALT-P**.

The Graphical User Interface (GUI) here on the right should appear. If not you have a problem.



The **GUI** is divided into four part:

1. A header with credits and three buttons: **Save**, **Load** and **Exit** – The three buttons does exactly what you are thinking of.
2. A top part, exhibiting a single button **EditMode** (For now!)
3. A central, big, part, which lets you define constants.
4. A lower part with just one button: **GENERATE** to launch execution.

Simple Usage

Simple usage means ‘With **EditMode** button **NOT** pressed’

In this configuration **BAG** is at its minimum power and allows you to load and save any analytic formula you, or someone else, have defined. Formula are stored in a XML file defined in a chapter of its own later on. A bunch of example is anyway provided, and described later.

Once a formula is loaded (If no formula is loaded **BAG** shows logarithmic spiral by default) The middle part of the GUI presents a series of NumButtons. Each NumButton represents a parameter whose name is on the left in pale yellow and whose description is on the right in light gray.

You can play with NumButtons as you wish. Pressing the **GENERATE** button creates the surface.

The figure here on the right represent the default logarithmic spiral.



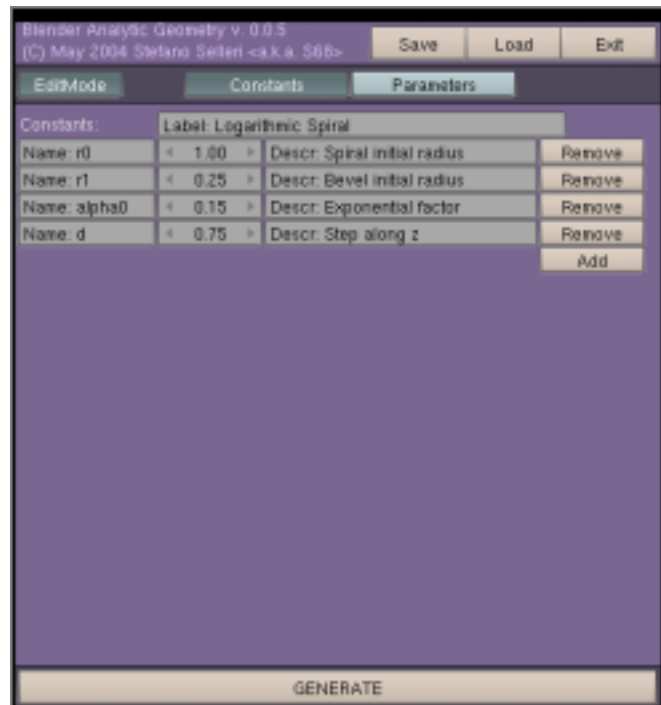
If you change any NumButton value, a save action will store the values in a file. Loading and Saving calls a standard blender File Window. Please note that **BAG** file extension is ***.bag** and that Blender does not set it by default.

As a premium BAG 0.0.5 generates UV coordinates for texture mapping. Since the surface is defined by *uv* parameters, **obviously**, to each vertex the pertinent *uv* values, scaled to [0,1] range are assigned.

Power Usage

To access Poer Utilization press the **EditMode** ToggleButton. The interface changes as shown here on the right.

Two more toggles appears in the top part. Allowing you to select two different Tab of the GUI. The first Tab define the constants.



Here you can not only change the constants values, but also the constants names and description.

You can add and remove a constant with the pertinent buttons

You can also change the label of the surface.

By pressing the **Parameters** ToggleButton the GUI switches to what is shown here on the right.

Here you can select the kind of reference system in which your surface is defined, either **Cartesian**, **Cylindrical** or **Spherical**.

Below the three ToggleButtons allowing this choice there are three TextButtons defining the parametric functions defining the surface.

Depending on which reference system is used: Cartesian (x,y,z) , cylindrical (ρ,ϕ,z) and spherical (r,θ,ϕ) , each surface is defined by three functions of the two parameters u and v , according to:

$$\begin{cases} x = f_1(u, v) \\ y = f_2(u, v) \\ z = f_3(u, v) \end{cases} \quad \begin{cases} \rho = f_1(u, v) \\ \phi = f_2(u, v) \\ z = f_3(u, v) \end{cases} \quad \begin{cases} r = f_1(u, v) \\ \theta = f_2(u, v) \\ \phi = f_3(u, v) \end{cases}$$

Each function must be a string, expressing the function according to Python syntax. The functions can contain, besides mathematical symbols, only the letters 'u' and 'v' which will be substituted by the current parametric values, and a number user defined constants and macros, as will be explained in the symbolic computation section. Valid examples are:

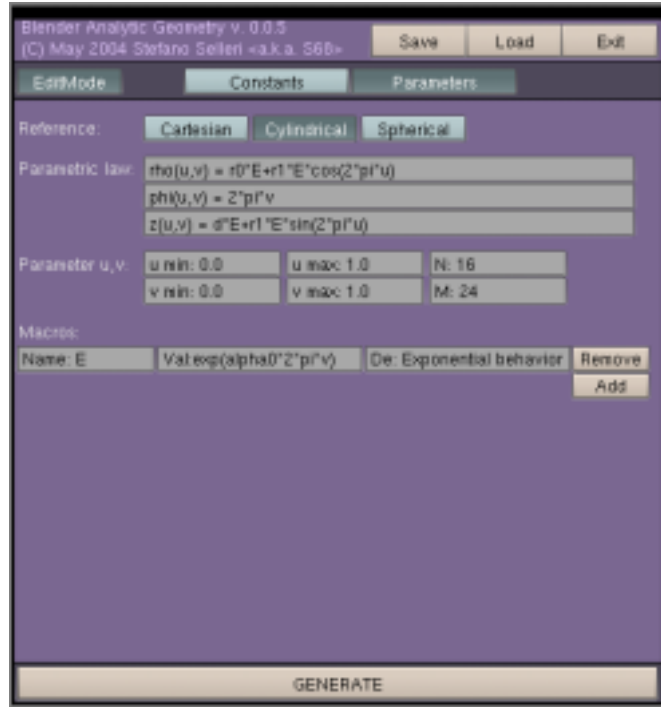
$$\begin{aligned} f_1(u, v) &= u * u \\ f_1(u, v) &= \cos(u) \\ f_1(u, v) &= 2 * u * \sin(v * v) \end{aligned}$$

parameters u and v have their range of variation and a number of points into which such range of variation is defined. These are defined in the GUI via $Umin$, $Umax$, $Vmin$ and $Vmax$ buttons.

Please note that $Umin$, $Umax$, $Vmin$ and $Vmax$ are strings and can be themselves parametric expression. These expression are again Python expression containing only math symbols and User defined constants.

A user defined constant is always implicitly defined, that is π . $\pi=3.141592\dots$

Parameters u and v are subdivided into N and M points, respectively, to create the surface. If one of these latter number is 1 then that parameter assumes the minimum value and does not vary. No surface is then generated, but rather a curve.



Again, N and M can be parametric expression which will be converted to integer values at computing time.

To help in function definition not only a variable number of user defined constants can be defined, as seen in the previous GUI Tab, but also an arbitrary set of Macro can be defined.

Macros are similar to Constants in their handling, they have a name, a value and a description and can be added and removed with the pertinent buttons. The key difference is that constants are *Numbers* while macros are *strings* and can be themselves functions.

This is quite flexible but must be understood to be used. The following section explain it.

Symbolic Evaluation

The surface is made by faces, defined by vertices in Blender's reference, which is Cartesian. These vertices are computed as follows.

The parametric expression are taken. These are 9 strings. The three defining the surface and the 2 groups of three defining u and v limit of variations and number of points. Let's name Γ one generic string defining one of these expressions. The algorithm is:

ONCE FOR ALL GENERATION

1. All macros are substituted in Γ one after the other *exactly in the order they are defined*, in RegExp language: foreach i in *NumberofMacros*: $\Gamma = \sim s/\text{MacroName}[i]/ \text{MacroValue}[i]/g$

Please note that this explicitly allows you to use later defined Macros within a Macro definition. The Shell demo function is a working example on this. Please also note that this can lead to errors

If Macro $M = a + b\ b$ then $M * c$ is $a + b * c$ and NOT $(a + b) * c$ as you might have expected, unless you explicitly use parenthesis in Macro definition.

2. All constants are substituted in Γ one after the other *exactly in the order they are defined*, in RegExp language: foreach i in *NumberofConstants*: $\Gamma = \sim s/\text{ConstantName}[i]/ \text{ConstantValue}[i]/g$

Since Constants are numbers this is easier to grasp.

3. At the end of the procedure u and v ranges of variations as well as N and M are computed. The two latter are also made integers

ONCE FOR EACH VERTEX COMPUTATION

4. The values of parameters u and v are computed as floats, this is indeed a pair of nested for loops.
5. The values of u and v are substituted in Γ as constants were in step 2.
6. The resulting expression is evaluated numerically.
7. The values are assigned to the pertinent coordinates
8. If necessary coordinates are transformed into Blender's Cartesian reference.

IF POINT 3 or 5 FAILS

Either because the expression is incorrect, values are undefined etc. the script assumes some pre-defined value for u and v limits and division, or the value '0' for the coordinate. A full warning is issued on the GUI.



Demo Functions

Five demo functions are currently shipped with the script. They are saved in *.bag files in the **demo** directory.

Sphere

The formula contains one user defined constant, the sphere radius R . The formula, in Spherical coordinates, is very simple. This duplicates blender's built-in Uvsphere.

$$\begin{cases} r = R \\ \theta = \pi u \\ \phi = 2\pi u \end{cases}$$

Torus

The formula contains two user defined constant, the radius R_a of the 'ring' and the radius R_b of the ring section. The formula, in Cylindrical coordinates, is:

$$\begin{cases} \rho = R_a + R_b \cos(2\pi u) \\ \phi = 2\pi v \\ z = R_b \sin(2\pi u) \end{cases}$$

Spiral

A logarithmic spiral ($\rho = e^{a\phi}$) made solid by imposing a circular section. Such a section has itself an exponential behavior. This is the default surface when **BAG** is launched:

$$\begin{cases} \rho = e^{au} (r_0 + r_1 \cos(2\pi v)) \\ \phi = 2\pi u \\ z = e^{au} (d + r_1 \sin(2\pi v)) \end{cases}$$

Coil

A complex mingle between a torus and a logarithmic spiral

$$\begin{cases} \rho = R_0 + R_1 e^{\alpha_0 v} \cos(2\pi n_1 v + \phi_1) + R_2 e^{\alpha_1 v} \cos(2\pi n_2 u + 2\pi n_1 v + \phi_1) \\ \phi = 2\pi v \\ z = R_1 e^{\alpha_0 v} \sin(2\pi n_1 v + \phi_1) + R_2 e^{\alpha_1 v} \sin(2\pi n_2 u + 2\pi n_1 v + \phi_1) \end{cases}$$

Shell

A very complex function to produce shells.

XML File

A *.bag file is a XML file like:

```
<BAG version="0.0.5">
<!-- #####
      DEMO BAG FILE
      #####

      A Logarithmic Spiral

-->
<Reference>Cylindrical</Reference>
<Label>
Logarithmic Spiral
</Label>
<F1>
r0*E+r1*E*cos(2*pi*u)
</F1>
<F2>
2*pi*v
</F2>
<F3>
d*E+r1*E*sin(2*pi*u)
</F3>
<ParU Min="0.0" Max="1.0">
16
</ParU>
<ParV Min="0.0" Max="1.0">
24
</ParV>
<UserPara Name="r0" Value="1.0" Index="0">
Spiral initial radius
```



```

</UserPara>
<UserPara Name="r1" Value="0.25" Index="1">
Bevel initial radius
</UserPara>
<UserPara Name="alpha0" Value="0.15000000596" Index="2">
Exponential factor
</UserPara>
<UserPara Name="d" Value="0.75" Index="3">
Step along z
</UserPara>
<Macro Name="E" Value="exp(alpha0*2*pi*v)" Index="0">
Exponential behavior
</Macro>
</BAG>

```

Tags are:

<BAG>

Encloses the whole data, its parameter, Version, is used to check against script's version.

<Reference>

Defines the reference used, can enclose the values Cartesian, Cylindrical or Spherical. If an error occurs, Spherical type is assumed.

<Label>

The surface title label, if any.

<F1> <F2> <F3>

These three tags encloses the pertinent parametric function

<ParU> <ParV>

Defines the parameters, it has parameters Min and Max defining the range of variation. The number of subdivisions is the content of the tag.

<UserPara>

An arbitrary number of UserPara tags are possible. Parameters are Name, Value and Index. The index tag, starting from 0, defines the order in which constants are shown. The content of this tag is a string describing the Constant, which will be used as a tooltip. Please note that this is read-only. You cannot define tooltips in the GUI.

<Macro>

An arbitrary number of Macro tags are possible. This tag is defined as a UserPara but values are strings.