1 The Python API

Blender provides a Python[1] API (Application Programming Interface) for developers. At *Not A Number's* home page you find a special section called *Python & Plugins* at the *Discussion Server* dedicated to chat about Python and plugin programming. At the moment there is another API for building C plugins. With the C API you can program your own texture plugins or plugins for the sequence editor within Blender. The C API has several disadvantages (see section 2.1) and it would be good to replace this API through a Python API. So let's see what is there in the Python API right now (see section 1.1) and what can be done in the future (see section 1.2).

1.1 Current Status

One problem for developers is to figure out what is available in the Python API. You can use Python in a way that your programs and modules are a bit self–documenting.

Figure 1: Daniel Dunbar's Help Browser

Daniel Dunbar used this feature in his implementation of the Python API and

[1]Python is an (object–oriented) programming language like C or C++.

wrote a script to browse the documentation strings within Blender (see figure 1). I used the same technique for adding NURBS support.

So what is there right now? As you can see in figure 1 there are ten (with NURBS support eleven) modules and some functions to access data. Daniel used modules for different data but I think it's a little bit confusing and would like to change the API to reflect the object–oriented view on the data as you can see it in the OOPS (Object Oriented Programming System) window (see figure 2).

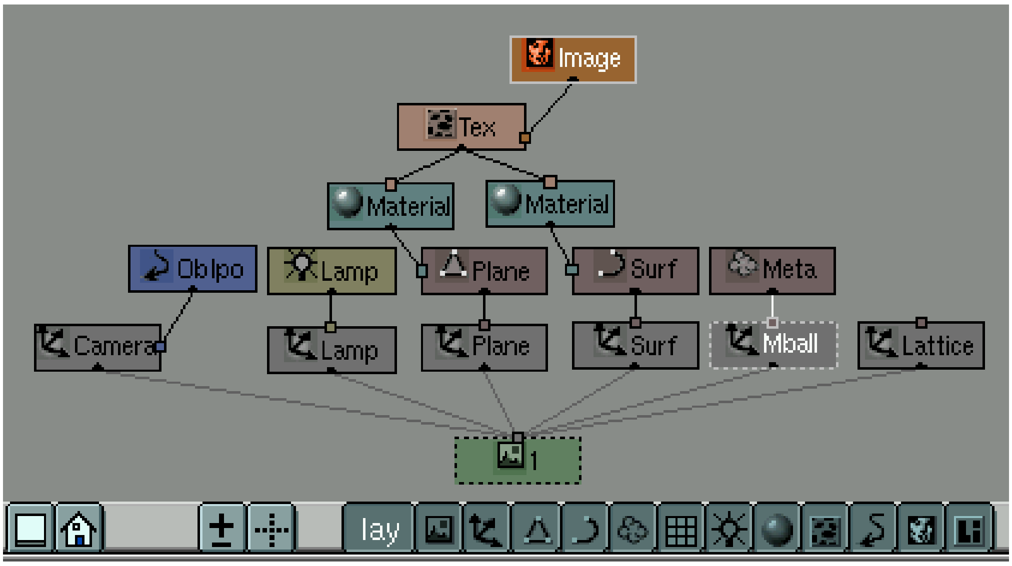


Figure 2: Object–Oriented Programming System

If you look at figure 2 you see that there is a single **Scene** and several objects connected to the scene. An **Object** has mainly a matrix connected to it and leaves more specific attributes to connected data like **Mesh**, **Curve**, **Surface**, **Font**[2], **Metaball**, **Lattice**, **Lamp**, **Material**, **Texture**, **Ipo** (acronym for Interpolation Curves), **Image**, and **Library**. I think this are good candidates for classes in Python to reflect what's going on in Blender. Additionally you will have functions to make connections between instances of this classes which are taking care that only types can be connected in a way Blender supports it. It would be nice — especially for the game engine — if the OOPS window would show **Scripts** and if the Python API would allow to share a Python script like you can share meshes or textures[3].

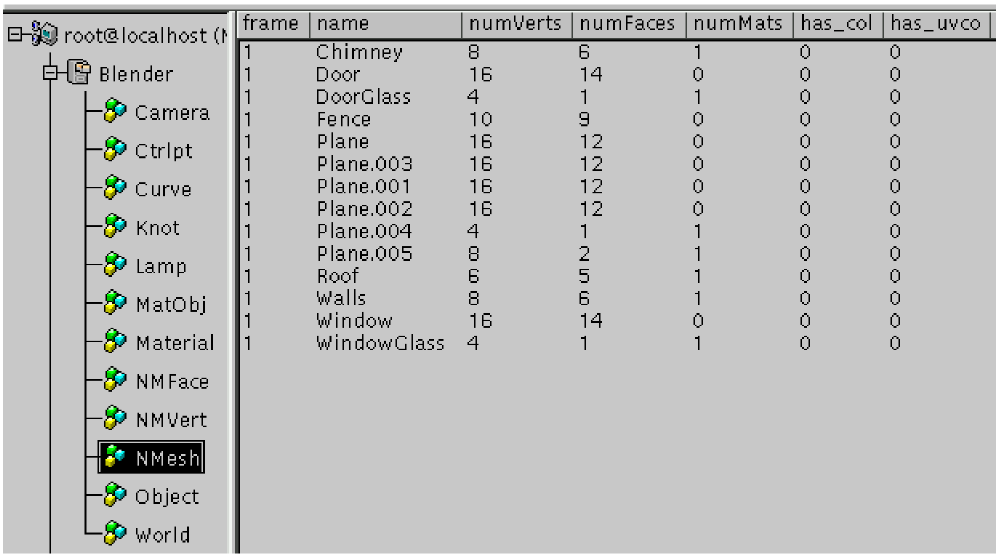
Because I want to change the Python API I introduced the so–called *Database Approach* (see section 1.1.1).

[2] **Curve**, **Surface**, and **Font** are not distinguished in the OOPS window.

[3] Look at figure 2 where two materials share one texture.

1.1.1 The Database Approach

The *Database Approach* is a method to save the developers of Python scripts from changes in the API. This does not work for all changes but if you have a good database with tables reflecting the internal Blender structure well enough than new attributes[4] added to the API will not necessarily mean that the developers working entirely on the database have to rewrite their Python scripts. I promised that I will take care of the connection between Blender and the external database. So all changes of the API should be reflected in the database.



frame	name	numVerts	numFaces	numMats	has_col	has_uvco
1	Chimney	8	6	1	0	0
1	Door	16	14	0	0	0
1	DoorGlass	4	1	1	0	0
1	Fence	10	9	0	0	0
1	Plane	16	12	0	0	0
1	Plane.003	16	12	0	0	0
1	Plane.001	16	12	0	0	0
1	Plane.002	16	12	0	0	0
1	Plane.004	4	1	1	0	0
1	Plane.005	8	2	1	0	0
1	Roof	6	5	1	0	0
1	Walls	8	6	1	0	0
1	Window	16	14	0	0	0
1	WindowGlass	4	1	1	0	0

Figure 3: A external database

In figure 3 you see the database approach I published with my Python scripts for version 2.02 of Blender. I think I will change the structure again[5] but the main idea is to find a structure for the database which will work for future enhancements. There are tables which will represent later classes for the new Python API e.g. **Lamp** or **Object** and there are tables for connections between different classes e.g. **MatObj**[6].

[4]This are rows in a table of the database.

[5]Because I don't like names like **NMesh** or **NMFace**.

[6]This was introduced for materials bound to objects. Unfortunately you can bind materials in Blender to meshes or objects. This makes sense because you can share meshes but maybe you want to have individual materials for each instance of this mesh. The new Python API will handle situations like this easily.

1.2 Future Work

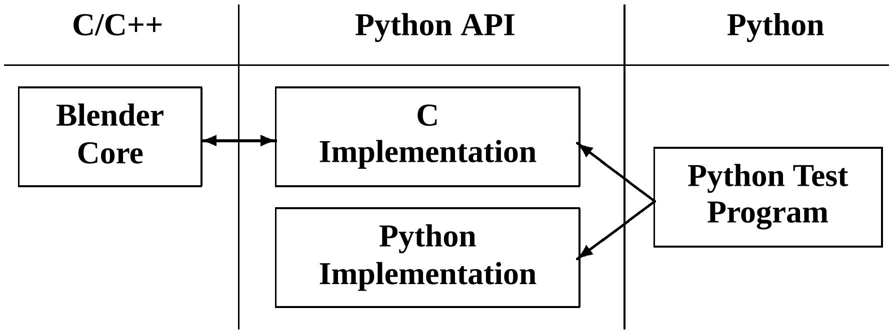


Figure 4: Test

2 The C API

The C API comes with the Blender distribution and is located in a subdirectory called *plugins*.

2.1 Disadvantages of the C API

The C API has several disadvantages:

- First of all there is no documentation of the C API shipping with Blender.
- Only a few people seem to use the C API.
- Most important: If you use the C API not accurately[7] you can crash Blender. And people tend to blame *Not a Number* for it.
- You have to compile your C code into a *shared object* or a *shared library*. This is operating system dependent and even when *Not a Number* gives developers assistance how to do it for several operating systems the developers have to actually compile it for each platform and operating system. This means that they need several computers with several operating systems and development tools installed on it just to be operating system independent like Blender is.

[7]Maybe because of the missing documentation.

Nevertheless the C API is very useful as long as there is no replacement for it through the Python API.