Boosting Mechanical Design with the C++ OOML and Open Source 3D Printers

Juan Gonzalez-Gomez, Alberto Valero-Gomez, Mario Almagro, and Miguel A. Salichs

***Abstract*—In this paper we are presenting the C++ Object Oriented Library (OOML). The OOML is a WYGIWYM (what you get is what you mean) tool that allows the fast development of 3D objects for fabrication on a 3D Printer. Designing with a WYGIWYM paradigm help students to communicate and reason geometrically, by applying geometry in real-world settings, and by solving problems through the integrated study of number systems, geometry, algebra, data analysis, probability, and trigonometry. These designed objects can be converted to STL files. The STL file can be used in 3D printers for fast prototyping or with traditional mechanization processes. We have designed and evaluated with students the OOML. Results show that the OOML plus the 3D Printer boosts their creativity. As a non searched result, we have observed that OOML also helps students to understand better the Object Oriented Programming Paradigm.**

I. Introduction. About mechanical design and 3D printers.

Bradshaw et. al [1] have recently made a study on low-cost 3D printing. They briefly run through the history of 3D printing, beginning in the late 1970s. These more than thirty years have driven to affordable 3D printers for individuals [2], and allow them to print complex engineering parts entirely automatically from design files that it is straightforward to share over the Internet. While open source software development has been studied extensively, relatively little is known about the viability of the same development model for a physical object design. 3D printers are offering new possibilities of sharing physical objects. As they can be defined using code, researchers can share their own parts, evolve them and build them using 3D printers. This allows for a decentralized community to independently produce physical parts based on digital designs that are shared via the Internet. Apart from improving the device, dedicated infrastructures were developed by user innovators. As Bruijn shows in his master thesis [3], a considerable improvement of hardware are proposed by people sharing parts and having access to 3D printers. This hardware modifications are relatively easy for others to replicate. As it has been the case with software for many years, currently, there are also on-line repositories of parts, where people can download and upload their designs[1].

In figure I four of the most important open source 3D-printers are shown. The origin of these kind of printers was the RepRap project[2][4] started by Adrian Bowyer in 2004. The aim of this project was to develop an open-source self-replicating machine. In May 2007 the first prototype, called Darwin was finished and some days later, in May 29th the first replication was achieved. Since then, the reprap community (original reprap machines and derived designs) has been growing exponentially[3]. The current estimated population is around 4500 machines. The second reprap generation, called Mendel, was finished in September 2009. Some of the main advantages of the Mendel printers over Darwin are bigger print area, better axis efficiency, simpler assembly, cheaper, lighter and more portable.

Initially, both Darwin and Mendel were not designed for the general public but for people with some technical background. As the reprap project was open-source, small companies were created to start shelling these 3D printers, as well as derived designs. The first company was Makerbot Industries[3], who shipped a first batch of their Cupcake CNC in April 2009. By the end of 2009 they had shipped nearly 500 complete kits. After operating for a year they had sold about 1000 kits in April 2010. Their latest design is the Thing-O-Matic printer, announced in September 2010. It is really easy to build and use, and their cost is around 950d'.

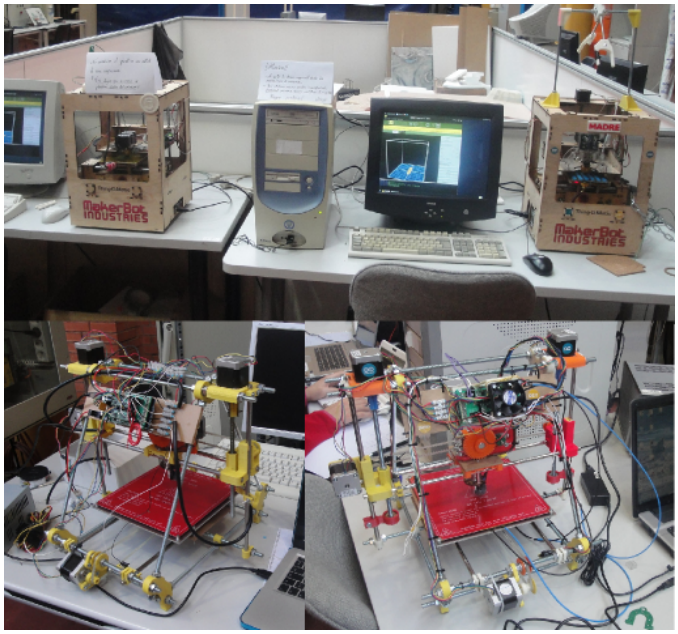


Figure 2. Our open source 3D printers at Carlos III University of Madrid. On the top: the Makerbot Thing-o-Matics. On the bottom: the Prusa Mendel prototypes, being built by the students

Juan Gonzalez-Gomez, Alberto Valero-Gomez, Mario Almagro, and Miguel A. Salichs are with the Robotics Laboratory of Universidad Carlos III de Madrid, Spain. juan@iearobotics.com

[1]http://www.thingiverse.com

[2]http://reprap.org/wiki/Main_Page

[3]http://www.makerbot.com/

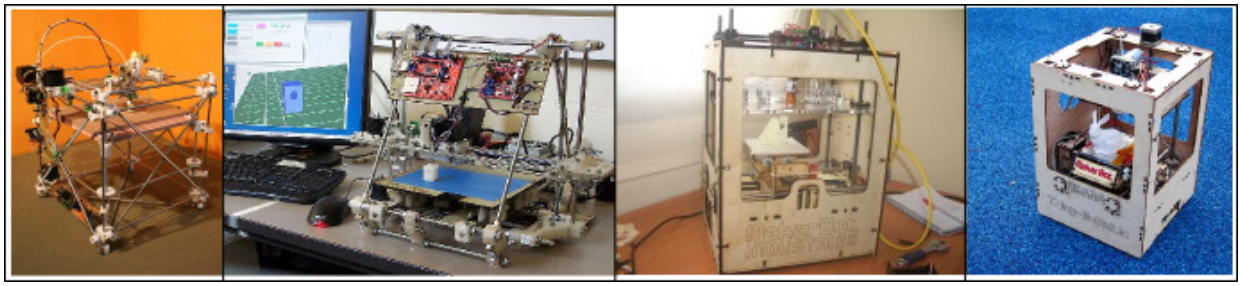


Figure 1. Pictures of some open source 3D printers. From left to the right: RepRap Darwin, the first generation (May, 2007); Reprap Mendel (Sep, 2009) the second generation; Makerbot Cupcake (April, 2009), the first commercial open-source 3D printer; Makerbot Thing-o-Matic (Sep, 2010), second version

Currently, at the System Engineering and Automatic Department of Carlos III University of Madrid we have two Thing-O-Matic Printers at the students disposal. They are shown in Figure 2. They were fully assembled by the students. Anyone has free access to them so that they can print whatever designs they want. Our main goal is to stimulate their imagination and enhance their creativity.

In addition we have started a project, called *Clone Wars*[4], in which a group of students are building their own reprap printers from the scratch. All the parts are being printed in our Thing-O-Matics. We have chosen the Prusa Mendel model as the design to build, because it is very well documented and it is rather easy to assemble. In figure 2(on the bottom) the first two prototypes are shown. In total the students are building 20 of them.

In the following sections we are describing the designing process of the mechanical parts to be printed. Being our aim, first of all educational, we would like that the students were able to start rapidly to have their designs done, and print them. As the "mechanization" process is now immediate thanks to the 3D printers, the designing process should not slow down their creativity. As we will present, thanks to the C++ Object Oriented Library, a student with a basic knowledge of C++ can design and fabricate relatively complex things after just one o two training hours. Having their designs made "real" has proved to motivate their creativity. In Section II we will introduce the designing paradigm in which the OOML is based. In Section III we will present the major characteristics of the OOML. Last Section will present some results and students' evaluation of the OOML.

II. Designing 3D objects through code: What You Get Is What You Mean.

Designing mechanical parts has been traditionally made using graphical CAD programs, following the WYSIWYG paradigm (what you see is what you get). According to the Oxford English Dictionary, the term WYSIWYG is used in computing to describe a system in which content (text and graphics) displayed onscreen during editing appears in a form exactly corresponding to its appearance when printed or displayed as a finished product which might be a printed document, web page, or slide presentation. Diverging from the main stream of 3D design OpenSCAD was born. Unlike most 3D CAD applications OpenSCAD does not follow the WYSIWYG but a derivative called WYGIWYM (what you get is what you mean). In a WYGIWYM editor, the user writes the contents in a structured way, marking the content according to its meaning, its significance in the document, instead of designing its appearance. For example, in a WYGIWYM document the user might mark text as the title of the document, the name of a section, or the name of an author. This requires the structure of the document (contents semantics) to be known before editing it. The editor also needs a system for exporting the edited text to generate the final format of the document, following the indicated structure. The main advantage of this paradigm is the total separation of presentation and content: users can concentrate their efforts on structuring and writing the document, rather than concerning themselves with the appearance of the document, which is left to the export system. Another advantage is that the same content can more easily be exported in different formats. OpenSCAD follows this paradigm. OpenSCAD is something like a 3D-compiler that reads in a script file that describes the object and renders the 3D model from this script file, giving the possibility to export it to different standard formats for 3D objects (STL, OFF, DXF). This gives the designer full control over the modeling process and enables him to easily change any step in the modeling process or make designs that are defined by configurable parameters.

Designing 3D objects through code has opened a new possibility that until now was only present in the software community. As it has been the case with software for many years, currently, 3D objects can be shared on the internet, modified, improved and reused. The new affordable 3D printers (for example, Thing-O-Matic by Makerbot) enhance this design and sharing process, as designers can easily share, print, test, and evolve the objects. OpenSCAD is (to our knowledge) the only coding environment+language for modeling mechanical things, nonetheless, the idea of designing 3D objects (in this case for visualization and video editing) through code is also present in PovRAY [5].

The differences between both paradigms do not only affect the way the designer has to work, but also the required skills, or, as it can be said from the education point of view, the skills a student will develop using it. Lizabeth Arum has made interesting comparisons among some free CAD programs[5].

[4]http://asrob.uc3m.es/index.php/Proyecto:_Clone_wars

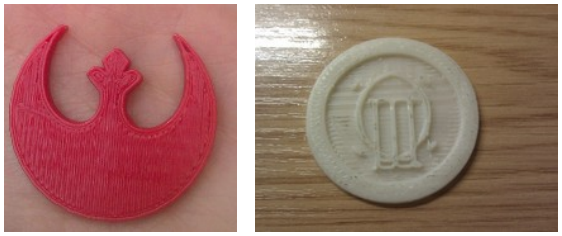
[5]http://lizarum.com/makerbot/index.html

The following table shows her analysis. The comparison is made in terms of the skills that are developed using/learning them.

Paradigm	Std 1	Std 2	Std 3	Std 4	Std 5	Std 6	Std 7
WYSIWYG	✔	✔			✔	✔	✔
WYGIWYM	✔	✔	✔		✔	✔	✔

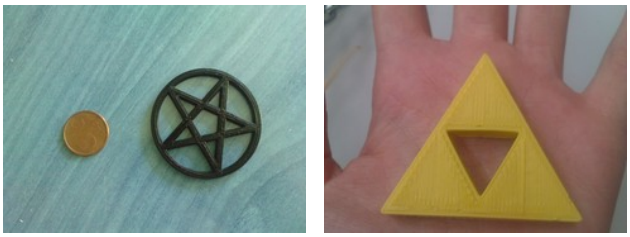
Table I
EVALUATION OF SEVERAL CAD PROGRAMS ACCORDING TO THE NYS LEARNING STANDARDS FOR MATHEMATICS, SCIENCE, AND TECHNOLOGY (HTTP://WWW.P12.NYSED.GOV/NYSATL/STANDARDS.HTML)

- Standard 1: Students will use mathematical analysis, scientific inquiry, and engineering design, as appropriate, to pose questions, seek answers, and develop solutions.
- Standard 2: Students will access, generate, process, and transfer information using appropriate technologies.
- Standard 3: Students will understand mathematics and become mathematically confident by communicating and reasoning mathematically, by applying mathematics in real-world settings, and by solving problems through the integrated study of number systems, geometry, algebra, data analysis, probability, and trigonometry.
- Standard 4: Students will understand and apply scientific concepts, principles, and theories pertaining to the physical setting and living environment and recognize the historical development of ideas in science.
- Standard 5: Students will apply technological knowledge and skills to design, construct, use, and evaluate products and systems to satisfy human and environmental needs.
- Standard 6: Students will understand the relationships and common themes that connect mathematics, science, and technology and apply the themes to these and other areas of learning.
- Standard 7: Students will apply the knowledge and thinking skills of mathematics, science, and technology to address real-life problems and make informed decisions.



(a) Rogue Squadron Symbol (b) University Logo

(c) BatMan Symbol (d) Chalk Holder



(e) Pentacle (f) TriForce Kay Chain

Figure 3. Things designed with OpenSCAD after one teaching hour. Source Code is on http://www.thingiverse.com

A. The advantages of the WYGIWYM paradigm in education

It could be discussed which designing paradigm is more appropriated for building 3D things, in fact, data gives few space to discussion: tools like Catia, 3DStduio, SolidEdge, which are based on the WYSIWYG paradigm, are by far more used in engineering. Nonetheless, we are not dealing with tools for final engineering design, but we are focusing on the educational aspects of these tools. We can enumerate three advantages of tools like OpenSCAD or OOML over other more powerful commercial tools:

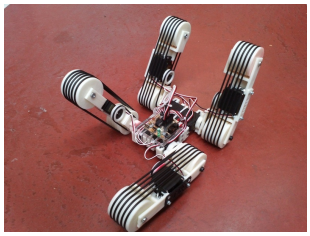
- The learning time is much shorter with WYGIWYM programs. Our experience over the last two years it that with a single lesson of two hours students can design relatively complex things. Figure 3 shows some things designed and printed by our students after one hour lesson.
- Designing through code forces students to understand the underlying geometry of the objects. Students will understand the mathematical relations of geometry (intersection, union, difference, rotation, translation), and become mathematically confident by communicating and reasoning mathematically, by applying mathematics in real-world settings, and by solving problems through the integrated study of number systems, geometry, algebra, data analysis, and trigonometry.
- Designing through code allows to easily share the designs. On the object code there is already disclosed its geometry (which does not happen on a WYSIWYG design). Objects are easy to understand, modify, and reshare.

Against the opinion that these tools, nonetheless, do not allow to design more complex objects and mechanisms we just show some examples of things developed by our students (Fig. 4).

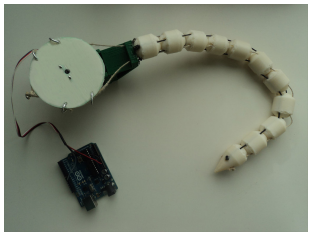
III. THE C++ OBJECT ORIENTED MECHANICS LIBRARY (OOML)

A. From OpenSCAD to OOML

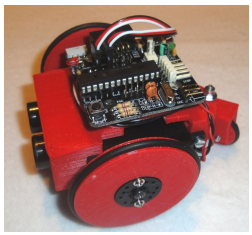
In academic year 2009/2010 we started applying a community oriented project based learning experience with engineering students of 2nd year (this teaching experience has been submitted as a full paper to this conference, in case it is accepted a Ref. will be placed here). Students were offered to follow voluntarily a course in which they would be taught to design robots, print them, design the electronics, place the sensors and finally program them. This course would consist of 2 hours per week plus their personal work. It would give them no "curriculum refund" and should be done out of the hours of their regular lessons. 12 students enrolled this initiative, that started in October 2009 and finished in May 2010. The resulting robots are shown in Figure 4.



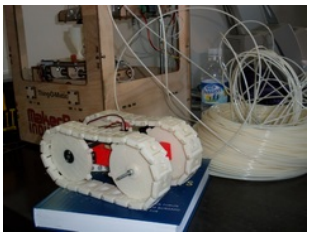
(a) 4-Track Robot



(b) Animatronic tail



(c) MiniSkyBot Robot



(d) Orugator Robot

Figure 4. Things designed with OpenSCAD. Source Code is on http://www.thingiverse.com

We taught them to design the mechanical parts using OpenSCAD. The electronics and the robotic agents were programmed using C/C++. Through the year we realized that the single parts designed by the students in OpenSCAD were not easy to share with each other. It was open source and it was code, but each student had its particular way of designing, which was different from the others. This made that the code of each student was hardly reused by the others. Another aspect that emerged was that after learning C++, students started to miss basic functionality that was not present in the OpenSCAD script, examples of this functionality were the while loops, the variables, etc. Also after learning C++ they started saying that it would be great to define physical objects as *programming objects*, and then convert the geometrical operations into methods of the objects.

After this feed-back the idea of the OOML was born and we started to develop it. The hard points of the idea were three.

- Physical objects are *programmed objects.*
- Geometric operations are the *methods of the object.*
- The parameters of the physical objects are the *attributes of the object.*

During the initial development of the OOML the idea of semantics arose. Geometric primitives are different from mechanical parts, and then, a part can be printed, but a package, like a castor wheel, is different from a part as each part must be printed separatedly. Nonetheless, a package is configured as a single entity. For example, the castor wheel can be configured giving the radius of the wheel and the total size, while the parameters of the single parts are calculated depending only on these two.

Finally, once semantics and object oriented things are designed, the concept of objects interface arises naturally, which standardizes the way in which primitives, parts, and packages are defined. These resulted in three base classes: PrimitiveObject, PrimitivePart, and PrimitivePackage, that making use of C++ polymorphism forces that all inherited objects include the same interface. The first preliminary version of OOML was released in July 2010, and a web page with all the required tutorials to start using it was created at http://iearobotics.com/oomlwiki

Summarizing, the C++ Object Oriented Mechanics Library (OOML) can be described in the following points:

- It includes all the geometry operations required to manipulate objects in space.
- It allows designers to create mechanical parts using the C++ language and applying the Object Oriented Programming Paradigm.
- The OOML is semantics oriented. It applies a semantic categorization of things, allowing to apply the WYGIWYM paradigm more easily and precisely.
- It generates OpenSCAD code. Using OpenSCAD designers may generate the STL of the parts.
- The OOML is Open Source, you can use it, modify it, share it.

We can conclude that the major advantage of OOML in relation with OpenSCAD does not lay in the designing paradigm, which is the same (WYGIWYM), but on the fact that the OOML is object oriented while OpenSCAD is not. An analogy could be found between C and C++. While for small programs C++ seems not to offer great benefit with relation to C, for big applications the benefits of the Object Oriented Methodology provided by C++ is out of the question in terms of code scalability and reusability.

A summarized comparison between both functionalities is shown in Table II:

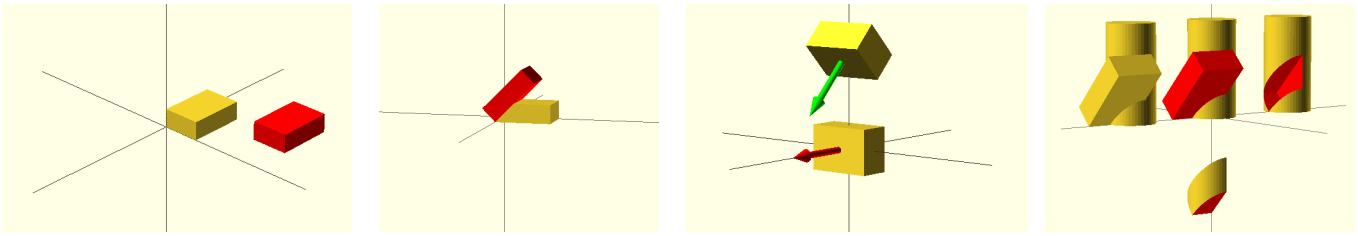
	OpenSCAD	**OOML**
Geometry Operations	✔	✔ (more than OpenSCAD)
Object Oriented	✗	✔
Semantics Oriented	✗	✔
Open Source	✔	✔
Primitive Objects	✔	✔(more than OpenSCAD)

Table II
OOML AND OPENSCAD

B. Geometry Operations

The OOML includes all the geometrical operations required for manipulating objects and creating new ones:

- Translation. Moves an object from a point to another, keeping its orientation.
- Rotation. Rotates an object around an axis.
- Movement. It moves one oriented point of an object to another point with a given orientation. Points and orientation are given by vectors. To do so it applies a translation and a rotation to the object.
- Intersection. Returns an object which is the result of the intersection of two objects.
- Addition. Returns an object resulting from adding other two objects.
- Difference. Substrates one object from another and returns the resulting object.
- Scaling. Scales an object with different (or equal) scale factors on each axis.



(a) Translation (b) Rotation (c) Movement (d) Original Objects (top center). Union (top left); Difference (top right); Intersection (bottom)

Figure 5. Geometric Operations

C. Object Oriented

The OOML applies the model of Object Oriented Programming to Mechanical Designs. Parts, designed as objects follow all the Object Oriented Programming properties:

- Encapsulation: Parts hide to the designer the attributes and methods that are not required for their definition, restricting access to some of the object's components. Doing so the OOML facilitates the bundling of data with the methods (or other functions) operating on that data.
- Dynamic dispatch – when a method is invoked on an part, the part itself determines what code gets executed by looking up the method at run time in a table associated with the object.
- Object inheritance (or delegation), parts can inherit from others following the *IS-A* relation. For example, a gears wheel *IS-A* wheel, inheriting its configuration methods and attributes, for example: set_radius(), set_thickness(), set_axis_radius().
- Reusability: Parts can be made of the addition or composition of other parts.

D. Semantics Oriented

While in OpenSCAD all objects have the same semantic meaning (they are just objects), in OOML objects are divided in three categories:

- Primitive Objects: They are geometrical entities, without a mechanical meaning, like a Cylinder or a Sphere.
- Parts: They have a mechanical meaning in a complex *thing*, like a wheel or a chain.
- Packages: They are a set of Parts making a more complex *thing*. Like a Castor Wheel, which contains a Wheel and a support. They cannot be printed in one time.

This division makes possible to add different functionality to the different kind of objects. On PrimitiveObjects the designer can apply all the basic geometry operations; Parts include *links* , which allow to join different parts together with a simple attach operation; Packages provide functions generate separately

Applying the Object Oriented methodology all these objects share the same programming interface, so existing libraries are easy to use.

IV. Teaching Experience

Up to know we have presented the revolutionary contribution of open source 3D printers on education. Now, the mechanical design must not finish by necessity on the object renderization, but students may print the objects they design, closing the fabrication process. We have afterward explained the differences between two designing paradigms and shown how the WYGIWYM paradigm might be more suitable for educating students in 3D design. In the last section we have explained how the OOML was born from the students feed-back and presented the major contributions of our C++ Objects Oriented Library. In this section we are describing our experience with students teaching them to design and print things with OOML. Some preliminary results will be presented. Finally we are listing some feed-back received from our students.

As we said before, last year we taught 12 students to design using OpenSCAD. This year we count with 40 volunteer students following this course. We have also launched an official master seminar on 3D design with OOML, which counts with 13 students. After showing how to use OpenSCAD we have introduced the OOML library. After teaching OOML to the students, those already knowing how to program in C++ rapidly started to design things and found OOML more intuitive and powerful than OpenSCAD. Instead, those that cannot yet program in C++ should face preliminary difficulties such as installing a compiler, linking with the OOML, building the program, etc which are not required when designing with OpenSCAD. Due to this, starting to design with OOML was slower and a little bit discouraging. Nevertheless, once the OOML was installed and configured we observed that students that had not programmed before with the C++ language found it easy to understand and were rapidly familiar with the Objects Oriented Paradigm. From this point of view we can summarize that OOML seems more appropriate for students that can already program in C++ (even if they are not experienced), while OpenSCAD is still easier for students without prior knowledge about programming. Nonetheless, our experience suggests that the OOML could be appropriated also for courses introducing the Object Oriented Programming paradigm.

Another of the features that was very well received by the students using OOML was the possibility to extend the Objects Library. Thanks to inheritance in C++, it is straightforward

to design and share new PrimitiveObjects, PrimitiveParts, and Packages. When we released the OOML only the Cube, Cylinder and Sphere were defined. In two months the rest of primitives shown in Figure 6 were developed by the students. They also developed and printed Primitive Parts, like the Battery Holder of Figure 7.

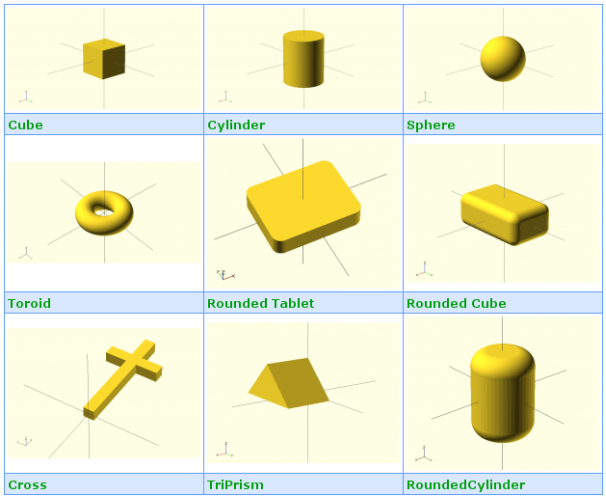
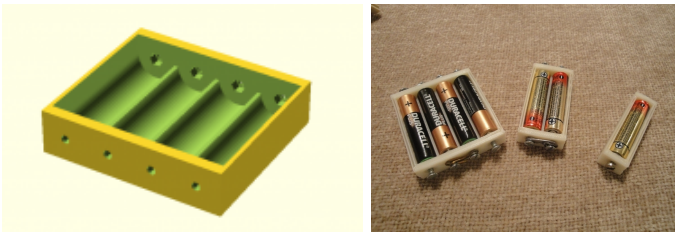


Figure 6. Primitive Objects in OOML



(a) Design (b) Printed

Figure 7. Battery Holder Designed with OOML

A. Students Feed-back

We asked three of our students to evaluate the OOML. Their impressions, even if statistically short, certainly give a hint on the OOML possibilities and encourage us to keep on working on it. Here we attach their answers. They were in the 4th course of Electronics and Automation Engineering. They could already program in C++, and they had used previously OpenSCAD.

a) Student 1. : "I am not an experienced programmer. In my first contact with the OOML these are the fundamentals I would highlight:

- OOML provides a great flexibility. You have at your disposal all the functions and libraries already present in C++. You can use classes, structs, etc. obtaining great performance.

- With OOML you can easily reuse the code of others as it is organized in classes. The programmer can create his own parts, and provide them with attributes and functions that makes the part easy to reuse when you are designing other things.

- It is really easy to use. It is even more intuitive than OpenSCAD. For translating or rotating you call functions of the object which help you to understand what are you rotating or translating.

- As you can use objects oriented programming the code is more compact.

- It is not the facilities that it offers currently, but all the possibilities that it opens. I can imagine hundreds of improvements that could be implemented in order to make object design easier and more powerful."

b) Student 2.: "After using OpenSCAD and OOML I find the OOML objects libraries quite useful. It is more intuitive than OpenSCAD if you know C++ and Object Oriented Programming. The great advantage I'd like to highlight is that the OOML has helped me to understand better the object oriented programming paradigm. Then the C++ language give the programmer much more possibilities than the OpenSCAD script.

Another advantage of the OOML is that I can use whatever code editor (IDE) I want. OpenSCAD editor is quite poor. Then the wiki tutorials are easy to follow. I could install the OOML and program my things just following the online information without major problems. Examples are clear and well explained."

c) Student 3.: The OOML is a good initiative for designing things using C++. It maybe lacks of some advanced functionality when positioning objects. For example, you move obstacles in absolute coordinates, it would be nice to move objects in relative coordinates respect other objects. If you are experienced using C++ there are some strange things, like the fact than when you move an object it generates a new moved object instead of moving the object itself. But once you have understood this particularity it is really easy. Then, as it is C++ I do not need to learn any other programming language, as in the case of OpenSCAD.

Online tutorials are easy to follow and with little previous C++ knowledge you can start designing things. In comparison with other CAD programs I find OOML very easy, and thanks to 3D printers you can pass from the idea to the thing in few hours. Currently the OOML has few objects in the library, but with time it will surely have more.

REFERENCES

[1] S. Bradshaw, A. Bowyer, and P. Haufe, "The Intellectual Property Implications of Low-Cost 3D Printing," *SCRIPTed 5*, 2010. [Online]. Available: http://www.law.ed.ac.uk/ahrc/script-ed/vol7-1/bradshaw.asp

[2] A. Bowyer, "The Self-replicating Rapid Prototyper, Manufacturing for the Masses," in *8th National Conference on Rapid Design, Prototyping & Manufacturing*, June 2007.

[3] E. de Bruijn, "On the viability of the open source development model for the design of physical objects. Lessons learned from the RepRap project," November 2010.

[4] R. Jones, P. Haufe, E. Sells, P. Iravani, V. Olliver, C. Palmer, and A. Bowyer, "RepRap-the replicating rapid prototyper," *Robotica*, vol. 29, no. 01, pp. 177–191, Jan. 2011. [Online]. Available: http://www.journals.cambridge.org/abstract_S026357471000069X

[5] T. D. Fenn, D. Ringe, and G. A. Petsko, "*POVScript+*: a program for model and data visualization using persistence of vision ray-tracing," *Journal of Applied Crystallography*, vol. 36, no. 3 Part 2, pp. 944–947, Jun 2003. [Online]. Available: http://dx.doi.org/10.1107/S0021889803006721