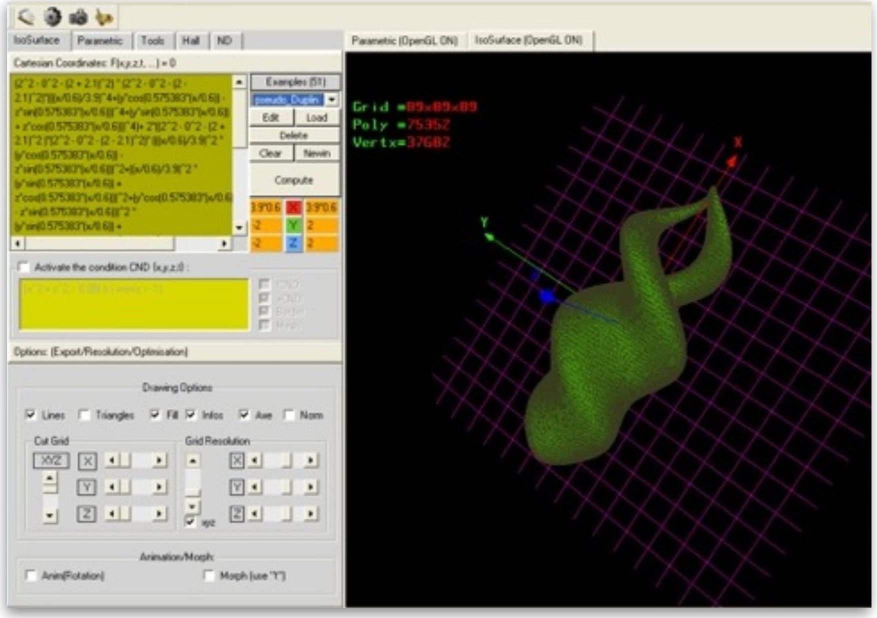
Software tools and web resources to generate math objects for 3D-printing

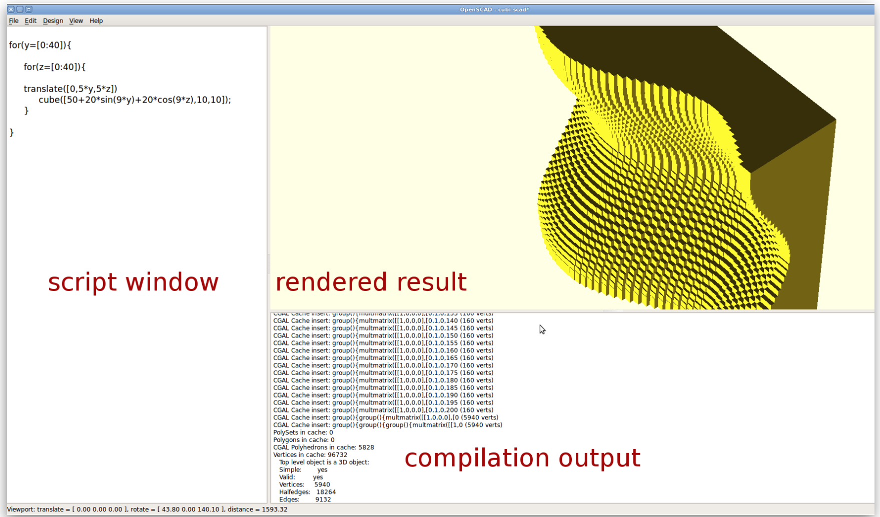
E. Canessa, G. Fior, C. Fonda - ICTP SciFabLab
Tarig Abdelgadir - ICTP Math Group

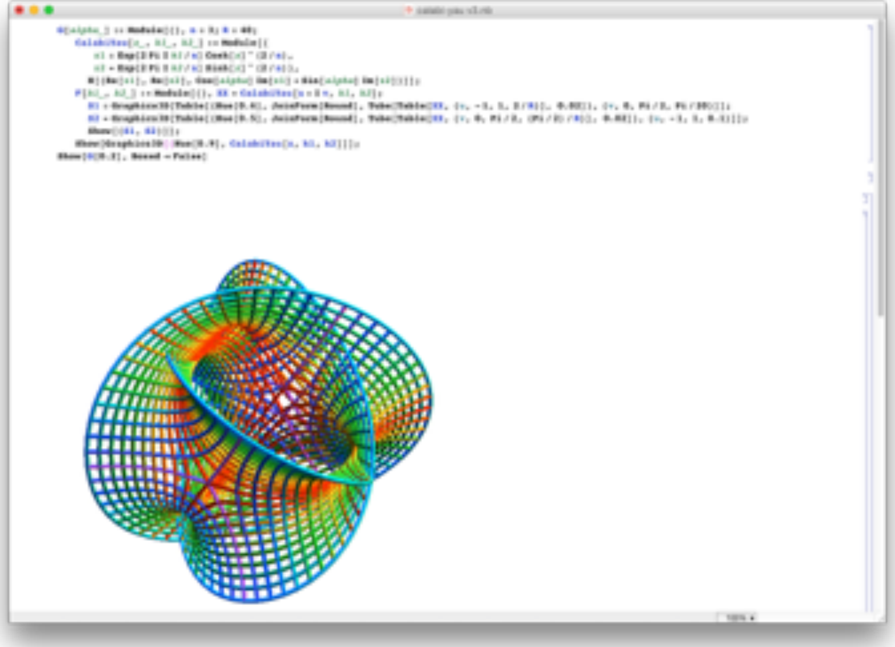
ICTP
The Abdus Salam
International Centre
for Theoretical Physics
50th Anniversary 1964-2014

Software

- From simple to more complex:
  - K3DSurf (k3dsurf.sourceforge.net)
  - OpenSCAD (openscad.org)
  - Mathematica™ (wolfram.com)

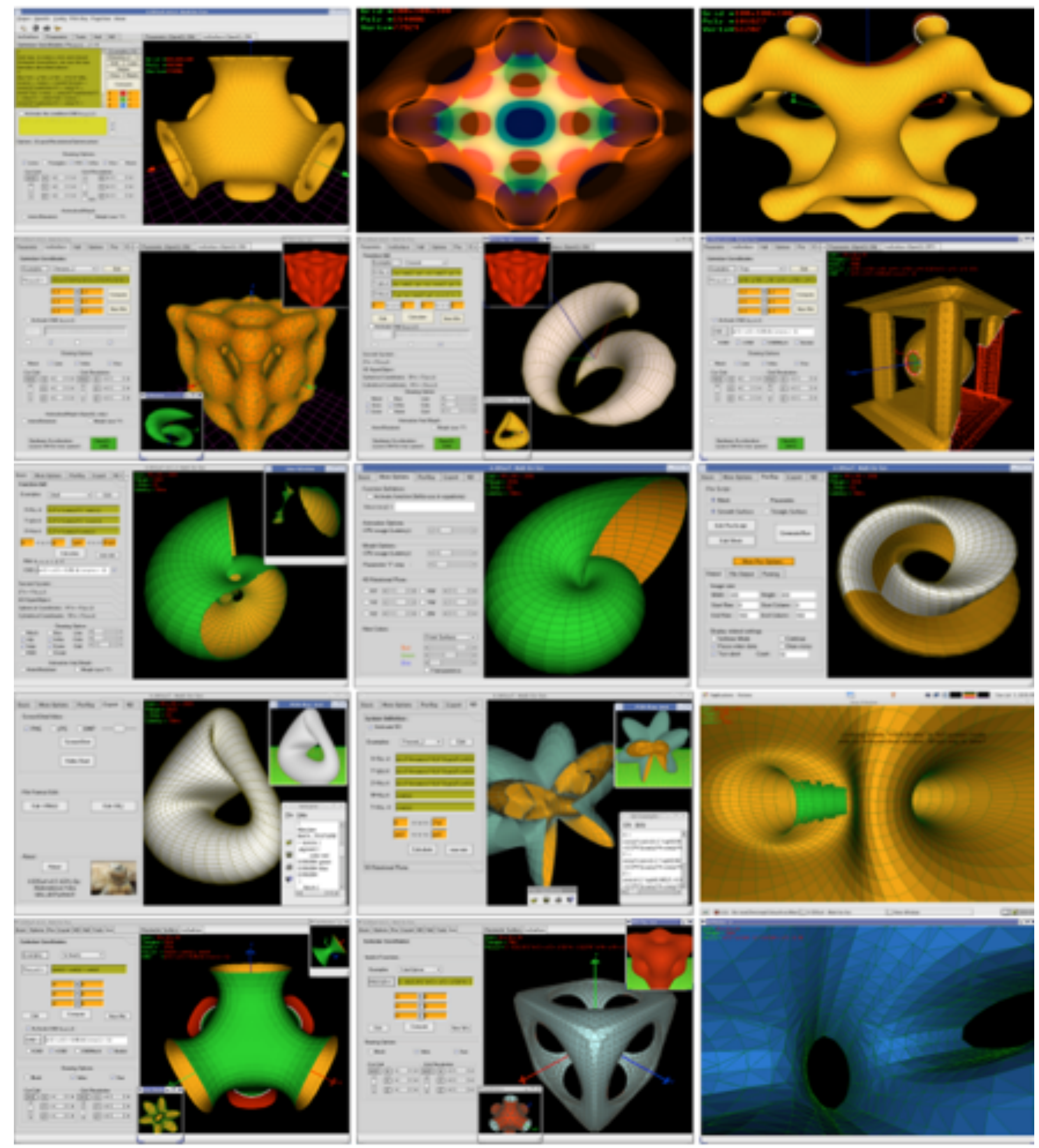


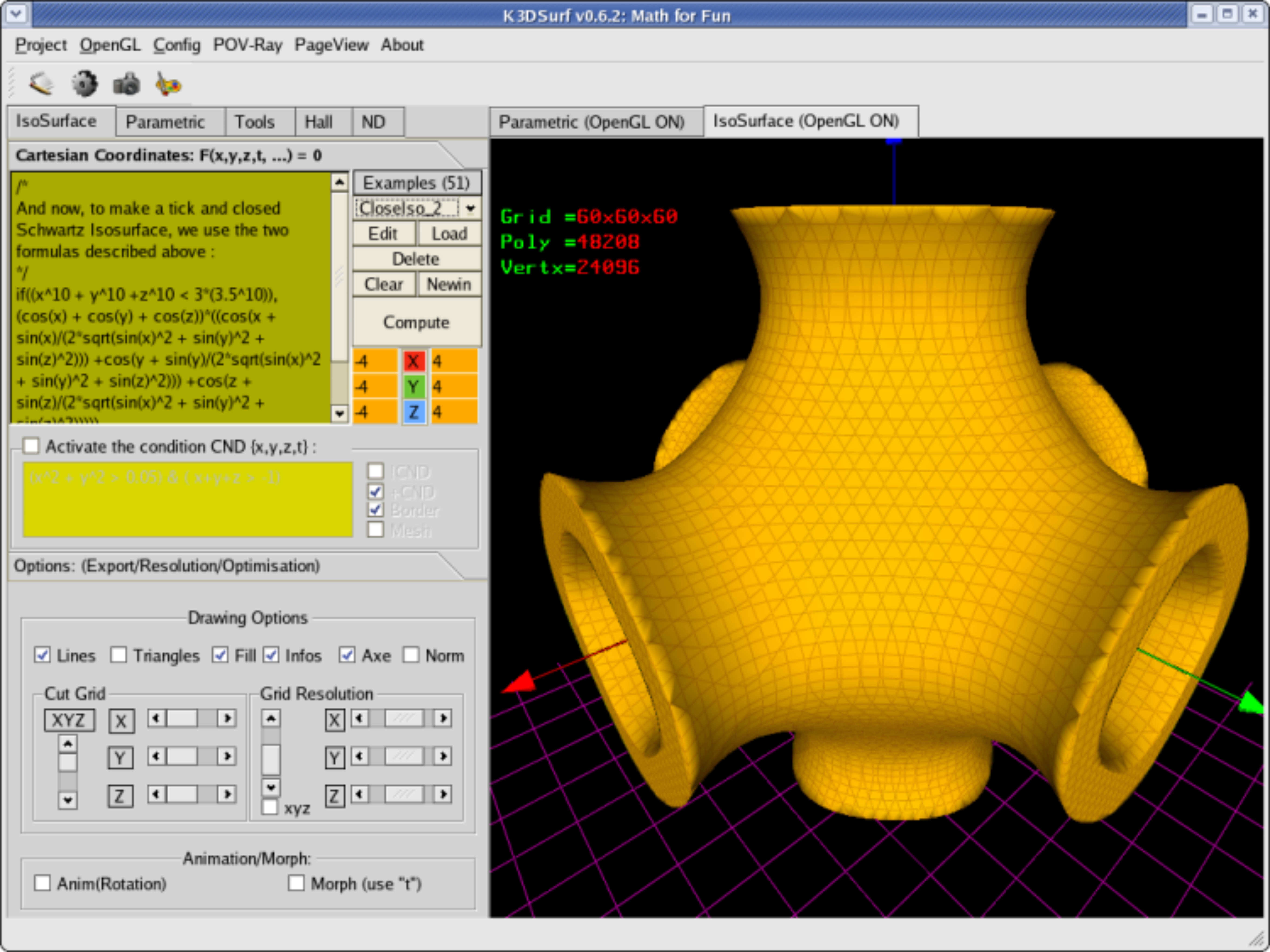


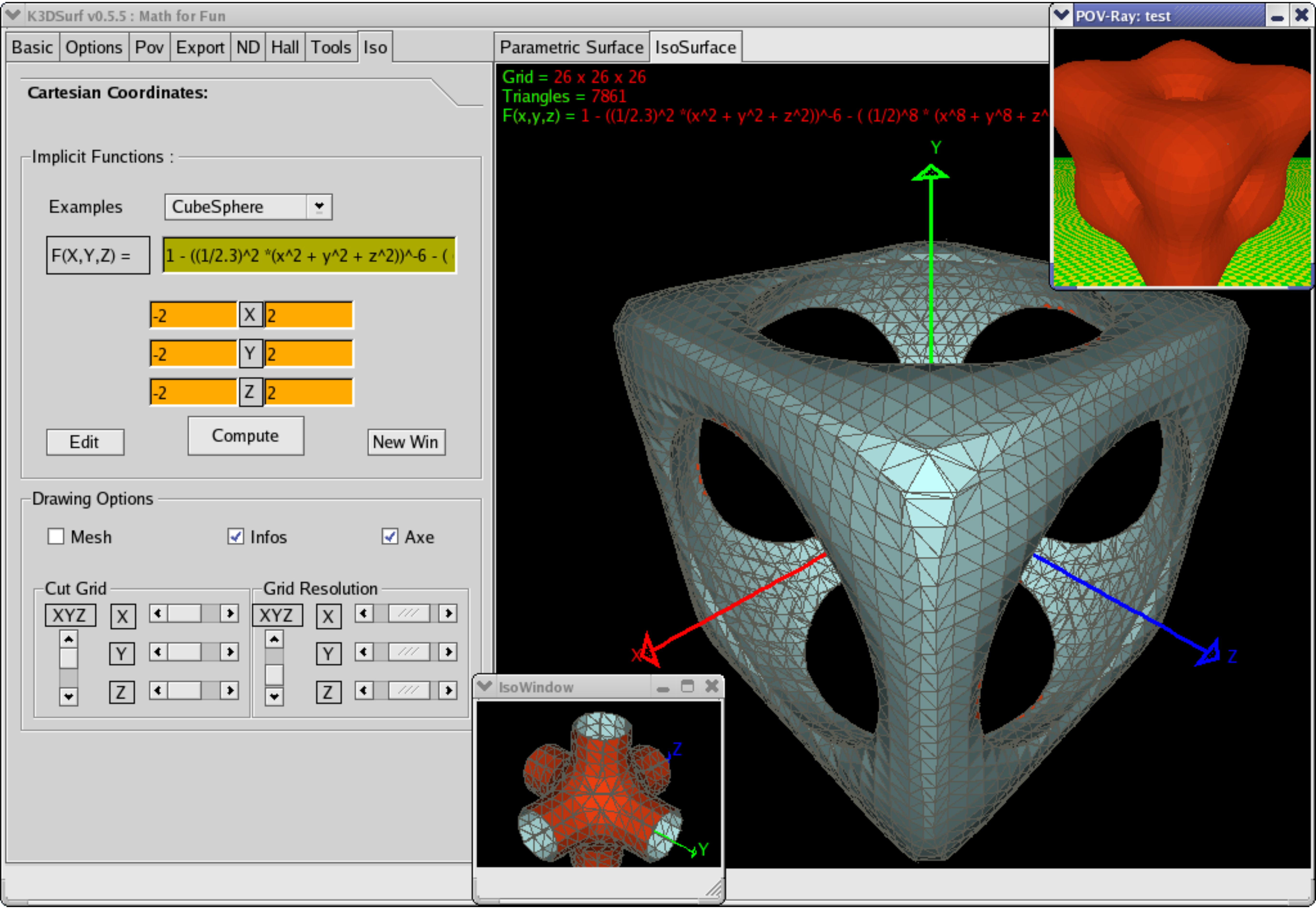


K3DSurf

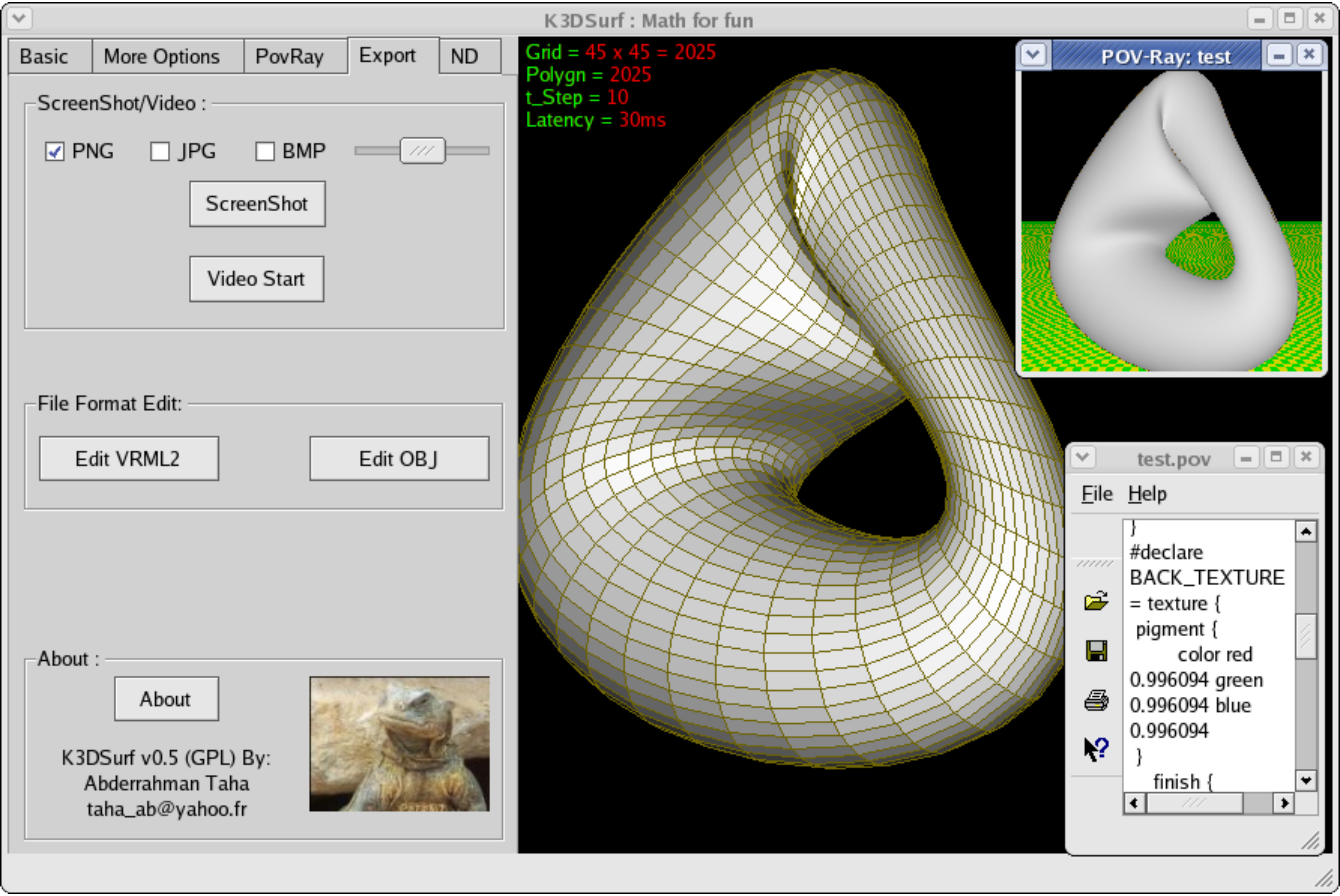
- K3DSurf is a program to visualize and manipulate mathematical models in 3, 4, 5 and 6 dimensions
- K3DSurf supports Parametric equations and Isosurfaces
- multiplatform (Win/OSX/Linux), free opensource software
- web: k3dsurf.sourceforge.net




K3DSurf v0.6.2: Math for Fun
Project
OpenGL
Config
POV-Ray
PageView
About
IsoSurface
Parametric
Tools
Hall
ND
Parametric (OpenGL ON)
IsoSurface (OpenGL ON)
Cartesian Coordinates: F(x,y,z,t, ...) = 0
/*
And now, to make a tick and closed
Schwartz Isosurface, we use the two
formulas described above :
*/
if((x^10 + y^10 +z^10 < 3*(3.5^10)),
(cos(x) + cos(y) + cos(z))*((cos(x +
sin(x)/(2*sqrt(sin(x)^2 + sin(y)^2 +
sin(z)^2))) +cos(y + sin(y)/(2*sqrt(sin(x)^2
+ sin(y)^2 + sin(z)^2))) +cos(z +
sin(z)/(2*sqrt(sin(x)^2 + sin(y)^2 +
Examples (51)
CloseIso_2
Edit
Load
Delete
Clear
Newin
Compute
-4
X
4
-4
Y
4
-4
Z
4
Grid =60x60x60
Poly =48208
Vertx=24096
Activate the condition CND {x,y,z,t} :
(x^2 + y^2 > 0.05) & (x+y+z > -1)
!CND
+CND
Border
Mesh
Options: (Export/Resolution/Optimisation)
Drawing Options
Lines
Triangles
Fill
Infos
Axe
Norm
Cut Grid
XYZ
X
Y
Z
Grid Resolution
X
Y
Z
xyz
Animation/Morph:
Anim(Rotation)
Morph (use "t")



K3DSurf v0.5.5 : Math for Fun
Basic
Options
Pov
Export
ND
Hall
Tools
Iso
Cartesian Coordinates:
Implicit Functions :
Examples
CubeSphere
F(X,Y,Z) =
1 - ((1/2.3)^2 *(x^2 + y^2 + z^2))^-6 - (
-2
X
2
-2
Y
2
-2
Z
2
Edit
Compute
New Win
Drawing Options
Mesh
Infos
Axe
Cut Grid
XYZ
X
Y
Z
Grid Resolution
XYZ
X
Y
Z
Parametric Surface
IsoSurface
Grid = 26 x 26 x 26
Triangles = 7861
F(x,y,z) = 1 - ((1/2.3)^2 *(x^2 + y^2 + z^2))^-6 - ((1/2)^8 * (x^8 + y^8 + z^
POV-Ray: test
IsoWindow

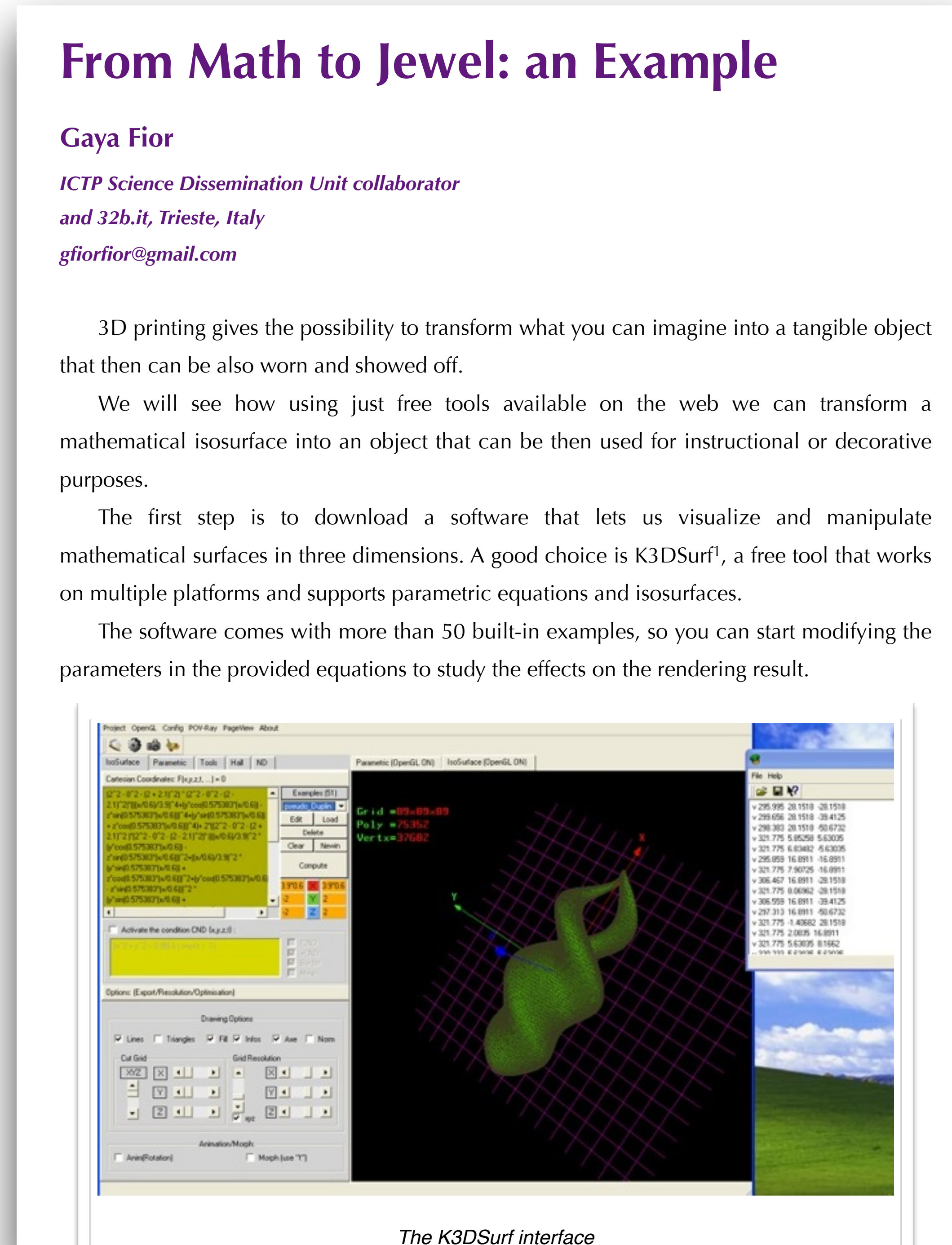

K3DSurf : Math for fun
Basic
More Options
PovRay
Export
ND
ScreenShot/Video :
PNG
JPG
BMP
ScreenShot
Video Start
File Format Edit:
Edit VRML2
Edit OBJ
About :
About
K3DSurf v0.5 (GPL) By:
Abderrahman Taha
taha_ab@yahoo.fr
Grid = 45 x 45 = 2025
Polygn = 2025
t_Step = 10
Latency = 30ms
POV-Ray: test
test.pov
File
Help
}
#declare
BACK_TEXTURE
= texture {
pigment {
color red
0.996094 green
0.996094 blue
0.996094
}
finish {

Tutorial: “Math to Jewel”

- See: **“From Math to Jewel: an Example”** an article by *Gaya Fior*

in the free open book on “Low-cost 3D Printing for Science, Education and Sustainable Development”

http://sdu.ictp.it/3D/book.html



From Math to Jewel: an Example

Gaya Fior

ICTP Science Dissemination Unit collaborator
and 32b.it, Trieste, Italy
gfiorfior@gmail.com

3D printing gives the possibility to transform what you can imagine into a tangible object that then can be also worn and showed off.

We will see how using just free tools available on the web we can transform a mathematical isosurface into an object that can be then used for instructional or decorative purposes.

The first step is to download a software that lets us visualize and manipulate mathematical surfaces in three dimensions. A good choice is K3DSurf[1], a free tool that works on multiple platforms and supports parametric equations and isosurfaces.

The software comes with more than 50 built-in examples, so you can start modifying the parameters in the provided equations to study the effects on the rendering result.

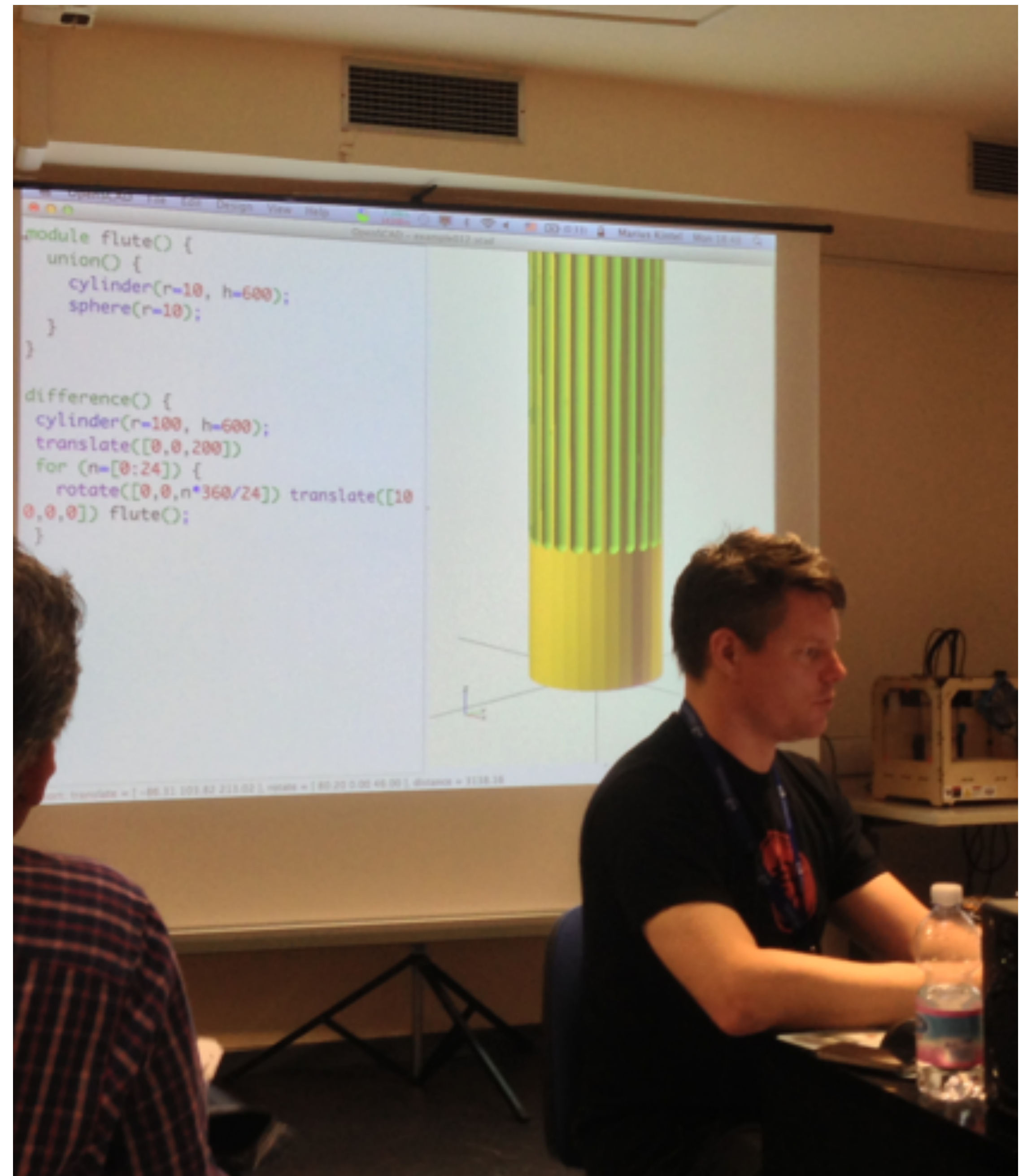
The K3DSurf interface

DEMO

K3DSurf for 3D printing

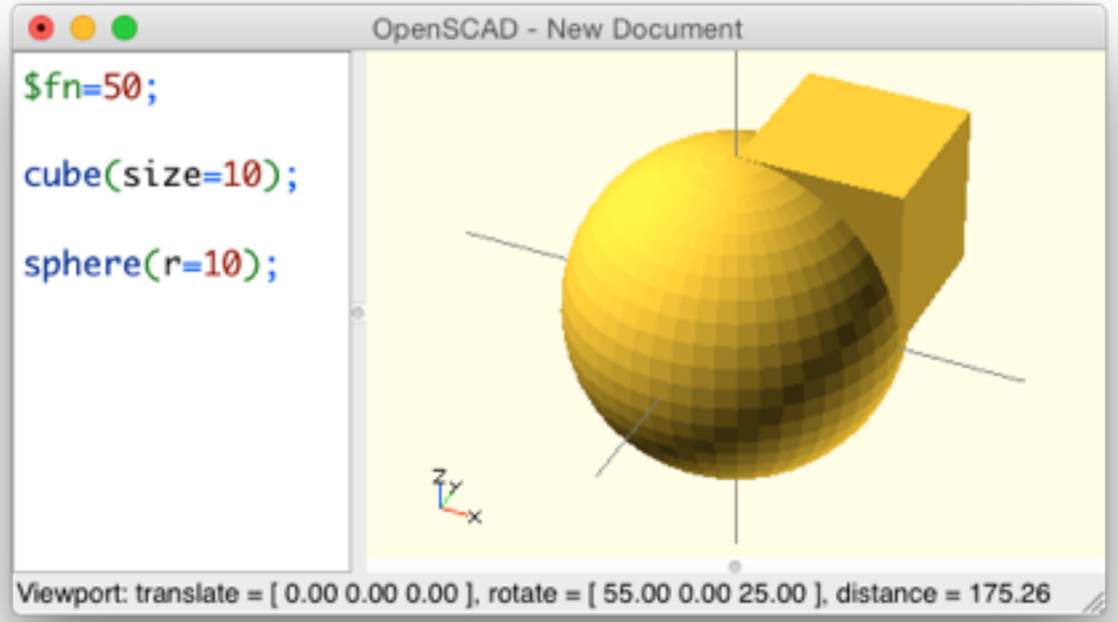
OpenSCAD

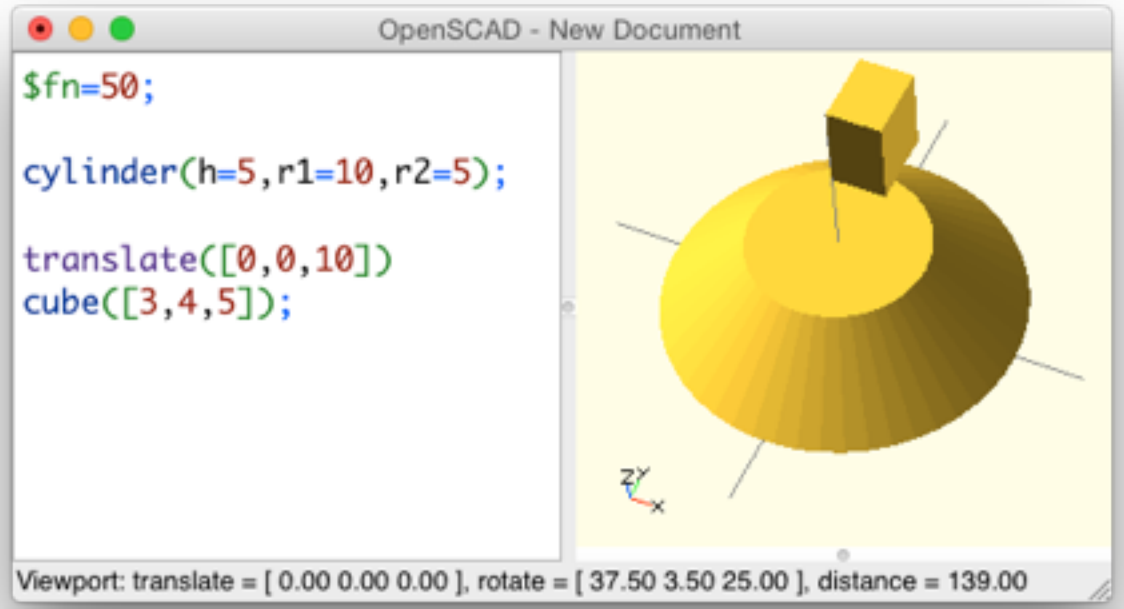
- OpenSCAD is a programming language for for creating solid 3D CAD models
- it's free, multiplatform (Win, OS X and Linux) and opensource, the lead author is Marius Kintel (see photo ;-)
- web: openscad.org



3D primitives

- **cube** (*size=10*) ;
- **sphere** (*radius=10*) ;
- **cylinder** (*h,r1, r2*) ;
- cube ([*width, height, depth*]) ;
- polygon ([points]) ;





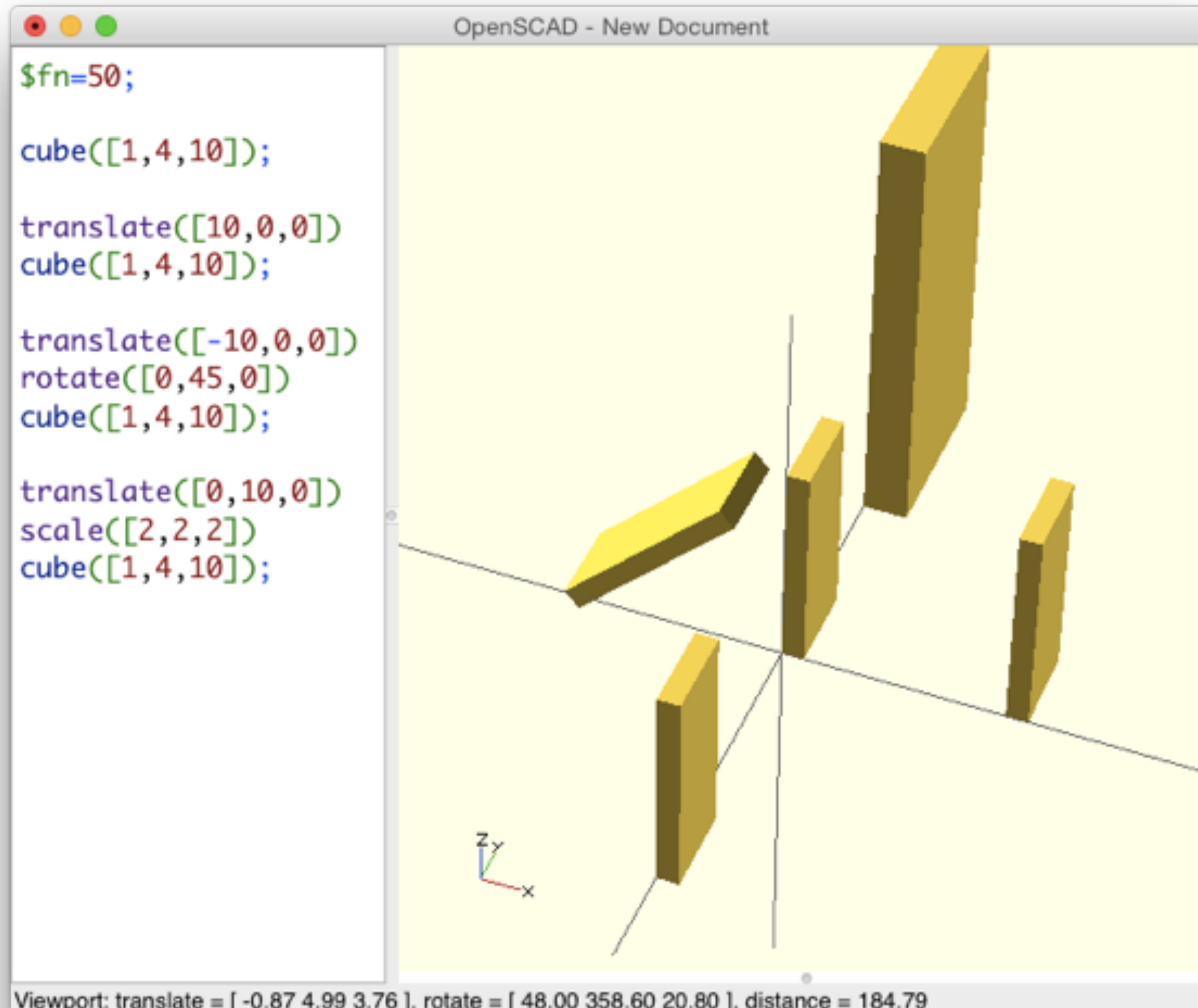
Transformations

- translate ([*x*,*y*,*z*])
  primitives ;

- rotate ([*x*,*y*,*z*])
  primitives ;

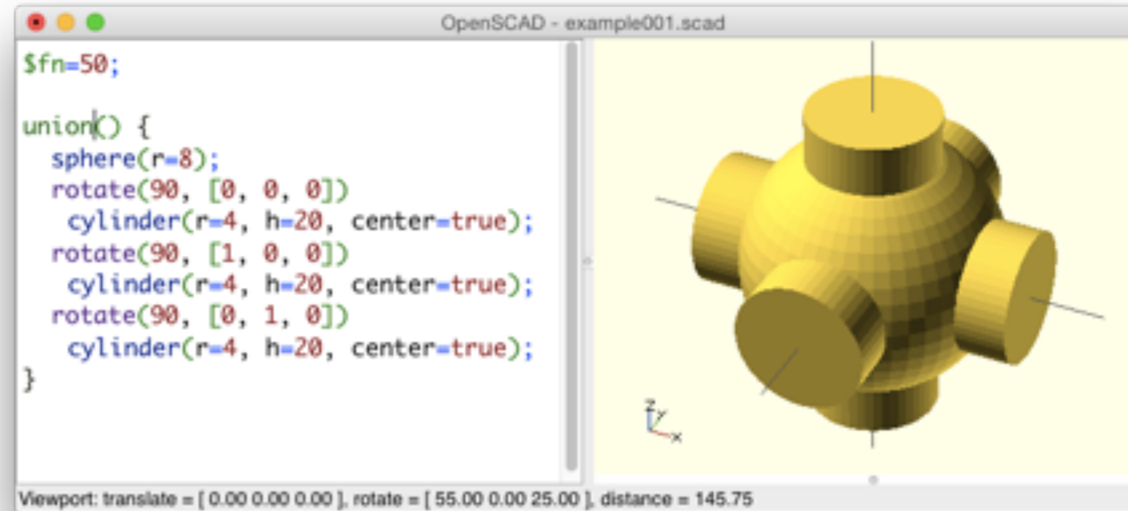
- scale ([*x*,*y*,*z*])
  primitives ;

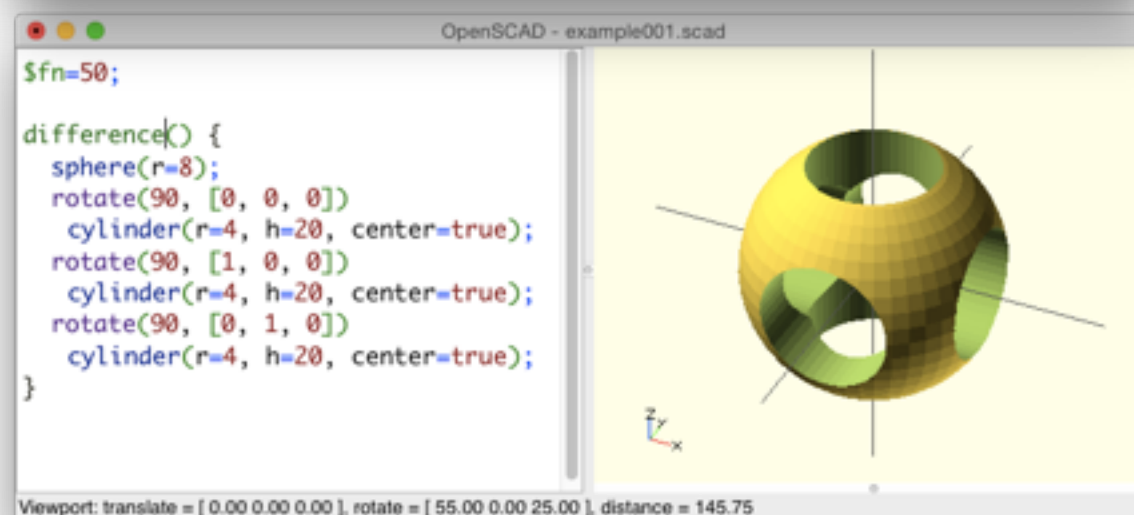
- mirror ([*x*,*y*,*z*])
  primitives ;

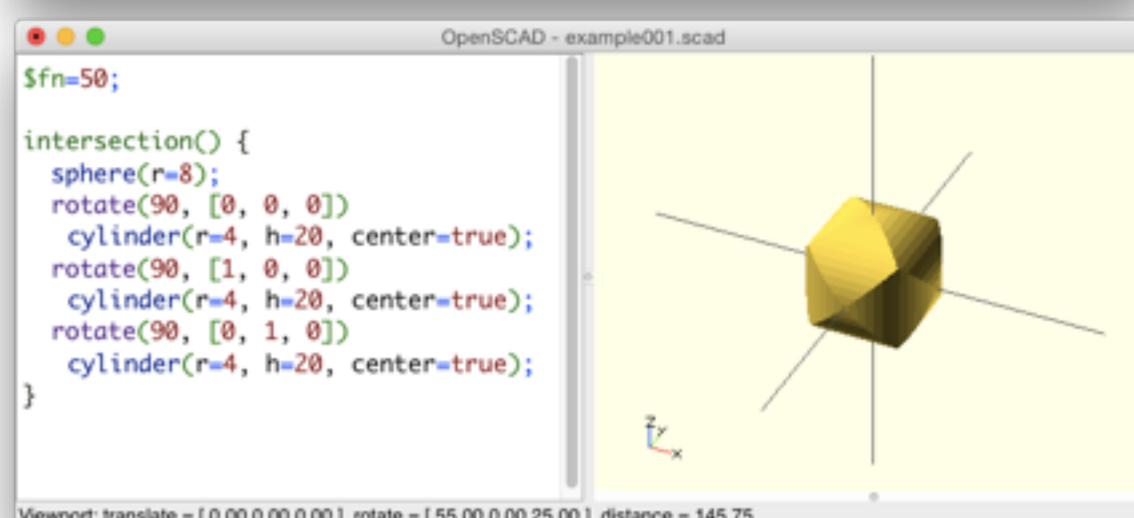


Boolean operations

- **union** ()
- **difference** ()
- **intersection** ()

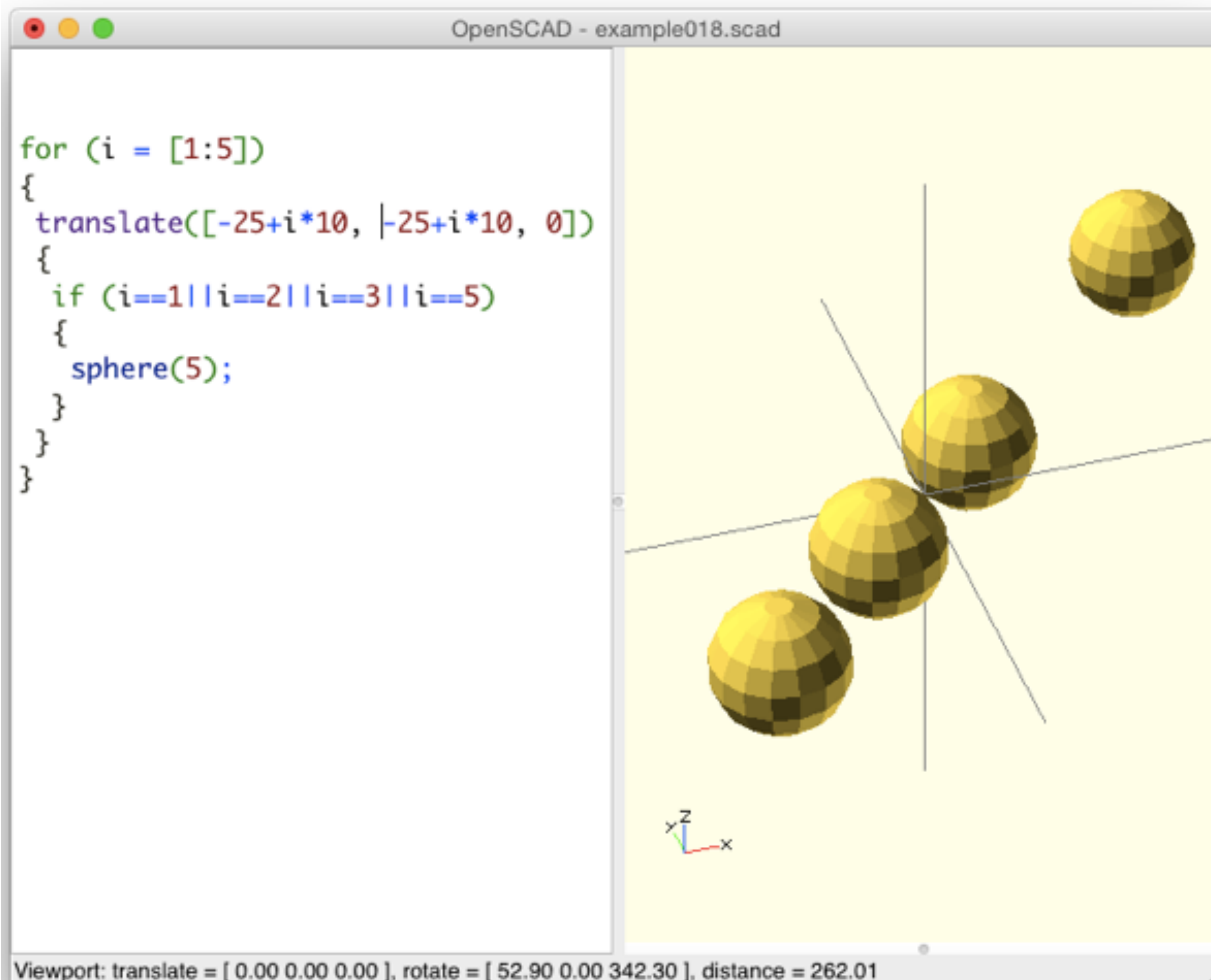






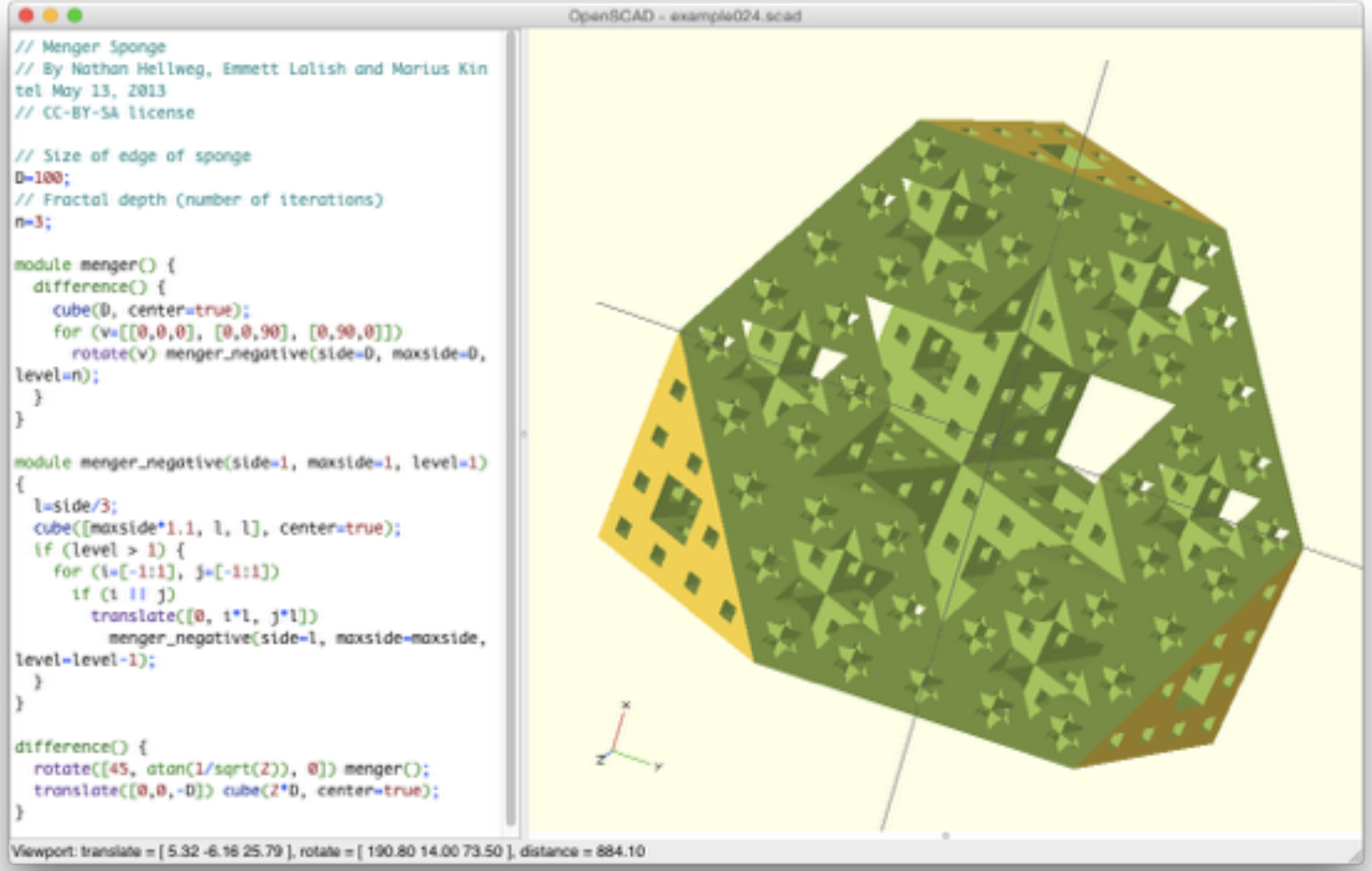
Loops & conditions

- **for** (i = [start:end])
  { … }
- **if** (condition)
  { … }
- … other usual stuff



Others

- many mathematical functions are available: power, root, trigonometrical, log, exponential, etc…
- functions, modules, include
- 2D primitives, extrusion
- export of 3D models as STL



A few simple commands

OpenSCAD CheatSheet

Syntax

```
var = value;
module name(…) { … }
name();
function name(…) = …
name();
include <….scad>
use <….scad>
```

2D

```
circle(radius)
square(size,center)
square([width,height],center)
polygon([points])
polygon([points],[paths])
```

3D

```
sphere(radius)
cube(size)
cube([width,height,depth])
cylinder(h,r,center)
cylinder(h,r1,r2,center)
polyhedron(points, triangles, convexity)
```

Examples

```
cylinder(10,5,5);
cylinder(h=10,r=5);
```

Transformations

```
translate([x,y,z])
rotate([x,y,z])
scale([x,y,z])
mirror([x,y,z])
multmatrix(m)
color("colorname")
color([r, g, b, a])
hull()
minkowski()
```

Boolean operations

```
union()
difference()
intersection()
```

Modifier Characters

```
*   disable
!   show only
#   highlight
%   transparent
```

Mathematical

```
abs
sign
acos
asin
atan
atan2
sin
cos
floor
round
ceil
ln
len
log
lookup
min
max
pow
sqrt
exp
rands
```

Other

```
echo(…)
str(…)
for (i = [start:end]) { … }
for (i = [start:step:end]) { … }
for (i = […,…,…]) { … }
intersection_for(i = [start:end]) { … }
intersection_for(i = [start:step:end]) { … }
intersection_for(i = […,…,…]) { … }
if (…) { … }
assign (…) { … }
search(…)
import("….stl")
linear_extrude(height,center,convexity,twist,slices)
rotate_extrude(convexity)
surface(file = "….dat",center,convexity)
projection(cut)
render(convexity)
```

Special variables

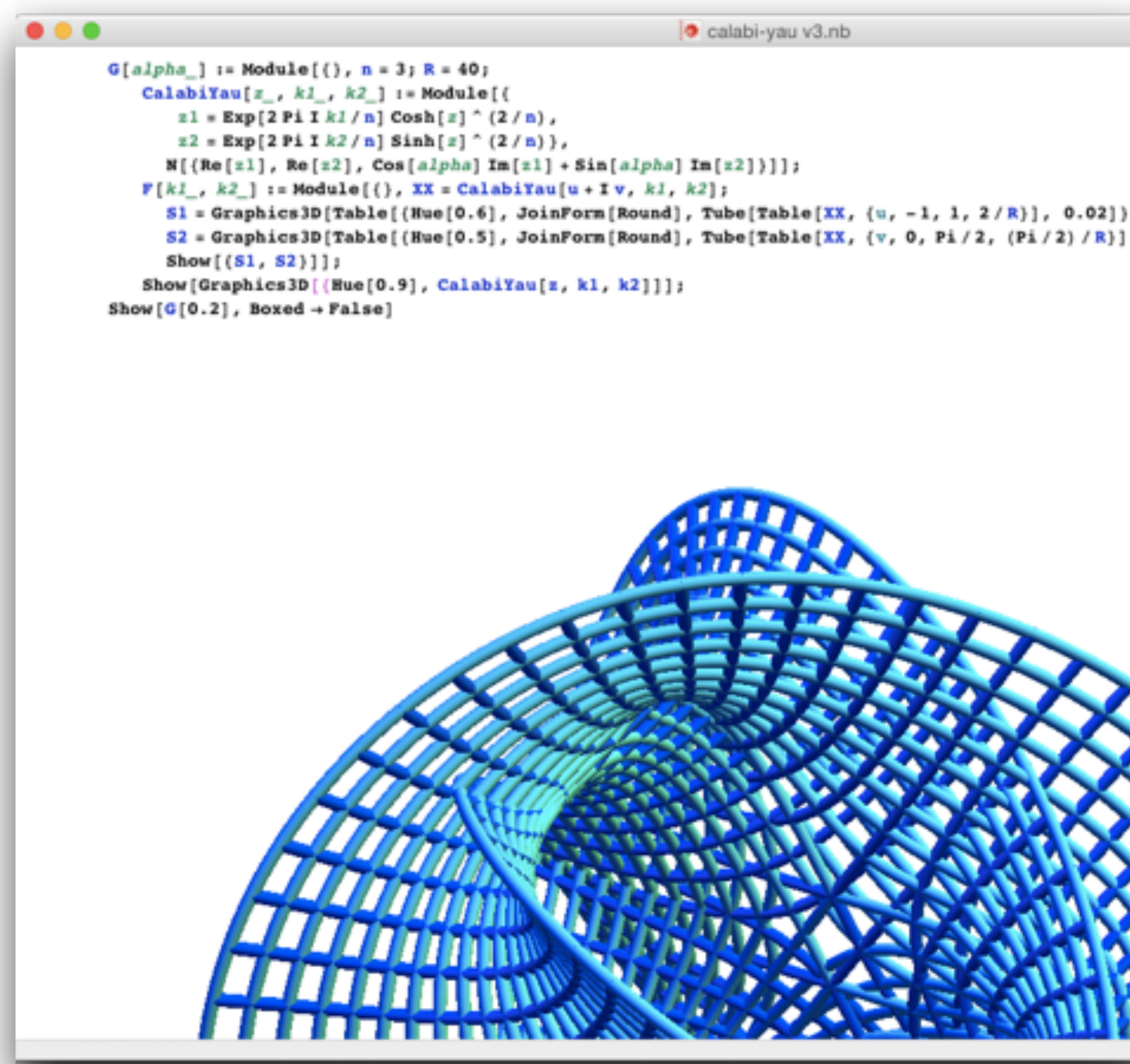
```
$fa minimum angle
$fs minimum size
$fn number of fragments
$t  animation step
```

DEMO

OpenSCAD: Menger sponge (example024)

Mathematica™

- Mathematica™ doesn't need any introduction, is the most powerful tool for mathematics
- it can generate and export 3D models (as STL, but it isn't always a correct manifold)
- www.wolfram.com/mathematica



- `Import` and `Export` fully support the STL file format.

Import and Export

- `Import["`*file*`.stl"]` imports an STL file as a `Graphics3D` object.
- `Export["`*file*`.stl", `*expr*`]` exports a `Graphics3D` expression to a binary STL file.

- `Import["`*file*`.stl"]` returns an expression of the form `Graphics3D[GraphicsComplex[`*vertices*`, `*primitives*`, `*opts*`]]`.
- `Export["`*file*`.stl", Graphics3D[`...`]]` creates an STL file representing a solid physical model.

- `Import["`*file*`.stl", `*elem*`]` imports the specified element from an STL file.
- `Import["`*file*`.stl", {`*elem*`, `sub_a`, `sub_b`, `...`}]` imports a subelement.
- `Import["`*file*`.stl", {{`$elem_1$`, `$elem_2$`, `...`}}]` imports multiple elements.
- The import format can be specified with `Import["`*file*`", "STL"]` or `Import["`*file*`", {"STL", `*elem*`, `...`}]`.

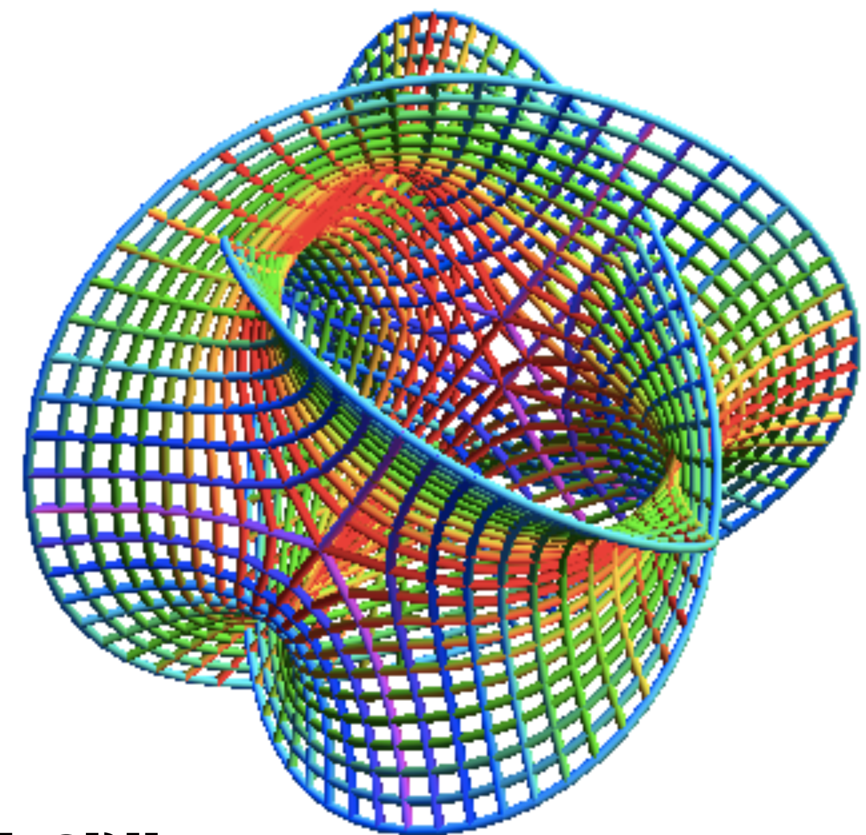
- `Export["`*file*`.stl", `*expr*`, `*elem*`]` creates a binary STL file by treating *expr* as specifying element *elem*.
- `Export["`*file*`.stl", {`$expr_1$`, `$expr_2$`, `...`}, {{`$elem_1$`, `$elem_2$`, `...`}}]` treats each $expr_i$ as specifying the corresponding $elem_i$.
- `Export["`*file*`.stl", `*expr*`, `opt_1` -> `val_1`, `...`]` exports *expr* with the specified option elements taken to have the specified values.
- `Export["`*file*`.stl", {`$elem_1$` -> `$expr_1$`, `$elem_2$` -> `$expr_2$`, `...`}, "Rules"]` uses rules to specify the elements to be exported.

DEMO

Mathematica™: Calabi-Yau

Calabi-Yau manifold

(used in string theory)



```
G[alpha_] := Module[{}, n = 3; R = 40;
  CalabiYau[z_, k1_, k2_] := Module[{
    z1 = Exp[2 Pi I k1/n] Cosh[z]^(2/n),
    z2 = Exp[2 Pi I k2/n] Sinh[z]^(2/n)},
   N[{Re[z1], Re[z2], Cos[alpha] Im[z1] + Sin[alpha] Im[z2]}]];
  F[k1_, k2_] := Module[{}, XX = CalabiYau[u + I v, k1, k2];
   S1 =
    Graphics3D[
     Table[{Hue[0.6], JoinForm[Round],
       Tube[Table[XX, {u, -1, 1, 2/R}], 0.02]}, {v, 0, Pi/2, Pi/20}]];
   S2 =
    Graphics3D[
     Table[{Hue[0.5], JoinForm[Round],
       Tube[Table[XX, {v, 0, Pi/2, (Pi/2)/R}], 0.02]}, {u, -1, 1, 0.1}]];
   Show[{S1, S2}]];
  Show[Graphics3D[{Hue[0.9], CalabiYau[z, k1, k2]}]];
Show[G[0.2], Boxed -> False]
```


Ultimaker