OpenSCAD语言

9 转变 (类似 librecad 的编辑功能)
9.1 scale 比例
9.2 resize 改变大小
9.3 rotate 旋转
9.4 translate 调动 (改变位置)
9.5 mirror 镜像
9.6 multmatrix 多点矩阵分布
9.7 color 颜色
9.8 minkowski 闵可夫斯基，
9.9 hull 去壳

10 CSG Modeling CSG 模型
10.1 union 联合
10.2 difference 剪切
10.3 intersection 求交
10.4 render 穿透(渲染)

11 Modifier Characters 特征修整
11.1 Background Modifier 背景修整
11.2 Debug Modifier 调试修整
11.3 Root Modifier 根 修整
11.4 Disable Modifier 无效修整

12 Modules 模块

13 Importing Geometry 导入几何
13.1 import 导入
13.2 import_stl 导入 stl

14 Include Statement include 描述

15 Other Language Features 其他的语言特点
15.1 Special variables 特殊变量
15.1.1 $fa, $fs and $fn
15.1.2 $t
15.1.3 $vpr and $vpt
15.2 User-Defined Functions 用户自定义函数
15.3 Echo Statements echo 描述
15.4 Render 渲染
15.5 Surface 表面
15.6 Search 搜索
15.6.1 Index values return as list 索引数值返回就像列表
15.6.2 Search on different column; return Index values 搜索在不同的行，返回索引数值
15.6.3 Search on list of values 搜索在数值列表上
15.6.4 Search on list of strings 搜索在字符串列表上

15.6.5 Getting the right results 获取正确的结果 pt

（根据官方 openSCAD user manual 用户手册为基础，以及个人领悟，并非完全官方直译，仅供参考，以官方教程为准。）

个人感觉：整体感觉模型画图的的方法就像是用 C 语言编写一段程序，然后编译，编译后，在某个第三方 3D／2D 环境下观看到模型，甚至模型的基础组件像，方 形，圆柱，等，就像是 C 语言的函数，后边配置用括号指定参数，多参数设置像 C 语言的数组，还有很多数学的调试功能，更像是 math.h 库的应用了，如果你 编写过用 math.h 库的 C 语言程序的话，这个软件对你的帮助将会更大，C 语言一般输出的就是数值，而这个软件把这些数值直接当成了 2D／3D 模型的输入 参数直接用 openGL，cga1， 软件转换成视觉模型出来，而不是数据模型，而是可视模型，而软件的"功能部分"，就像 2 个模型组合，联合，区分，交集，镜像，颜色，多矩阵，等，是依赖一 个单独的 3D 模型组合软件库 openCSG 来完成的，总体感觉简单，便捷，强悍，易学，教程完善，安装方便等特点集聚一身，不错的软件，希望更多的中国的 机械设计和其他的 3D 模型设计者 ，能够多学习和应用这个软件。

有问题可以探讨。

这里，我做了一些，我个人感觉应该的调整，做了一些数序的调整，并并把整个 OpenSCAD 语言这一章分成了 3 个部分。 1～7， 8～12，13～15， 其实，最后整理成 2 部分。

我的顺序是 8~12 是第一部分部分， 1～7， 13～15，是第二部分。因为第一部分属于画图部分，这行函数都是用于生成最基础的零件，组件，模块，而第二部分是根据第一部分的基础模型基础上加以编辑，功能强大，且不容易理解。所以我在翻译的时候，就调整了顺序。

这样的顺序，我感觉容易理解，内容上没有修改。

第一部分（简介，鼠标，键盘，和菜单）.

第二部分,（ 8~12）；

第三部分,(1～7) && (13～15);

第一部分（简介，鼠标，键盘，和菜单）.

1，介绍

OpenSCAD 是一个用于创建立体三维 CAD 对象的软件。它是免费软件，可用于 GNU/Linux，MSWindows 和苹果 OSX。

下载地址链接

http://www.openscad.org/downloads.html

下载地址链接

不同于大多数自由软件，用于创建 3D 模型（如著名的应用 Blender），OpenSCAD 不专注于艺术方面的 3D 建模，而是专注于 CAD 方面。因此，它可能当你正在寻找一个建造 3D 机械零件的应用，但可能不是你在期望的你非常感兴趣的在电脑动画电影。

OpenSCAD 不是一个交互建模。相反，它是在脚本文件中描述对象，并呈现从脚本文件中的 3D 模型，上面写着像一个三维的解释。这给了你（设计师）的建模过程的完全控制权，使您可以轻松地更改任何步骤在建模过程中，甚至设计所定义的配置参数

OpenSCAD 包括两个主要的建模技术：首先，建设性的立体几何（CSG），第二，二维轮廓映射。AutoCAD DXF(qcad,1ibrecad)文件作 为数据交换格式的二维轮廓。除了 2D 路径输出，但也可以从 DXF 文件读取设计参数。除了读取 DXF 文件，OpenSCAD 还可以读取和创建三维模型的 STL 和 OFF 文件格式。

此软件类此 openCASCADE,区别是:openCASCADE 是一个标准的 C++库，本身就是专业的 CAD 库;

而 openSCAD 是利用 openCSG 和 CGAL,这两个标准 C++库来完成 CAD， 主要以立体几何图像技术和

openGL 渲染完成 CAD 模型。

openCSG 基于图形的构建立体几何用 openGL 做渲染的软件库，使用 C++语言编写并在 windows/Linux 系统中支持大多数硬件.

CGAL 开源项目，目标是以 C++库的形式提供方便，高效，可靠的几何运算，CGAL 应用到在诸多方面有几何运算需要的地方，如：计算机图形学，科学可视化，计 算机辅助设计与建模，地理信息系统，分子生物学，医学成像，机器人技术和运动规划，网格生成，数值计算方法等等， 相见 http://www.cgal.org/

2，OpenSCAD 第一步

2.1 创建一个简单模型

2.2 打开一个已经存在的例子模型

2.3 位置布局（置位）于对象模型

2.4 更改对象模型的颜色

2.5 模型观看(三维)'

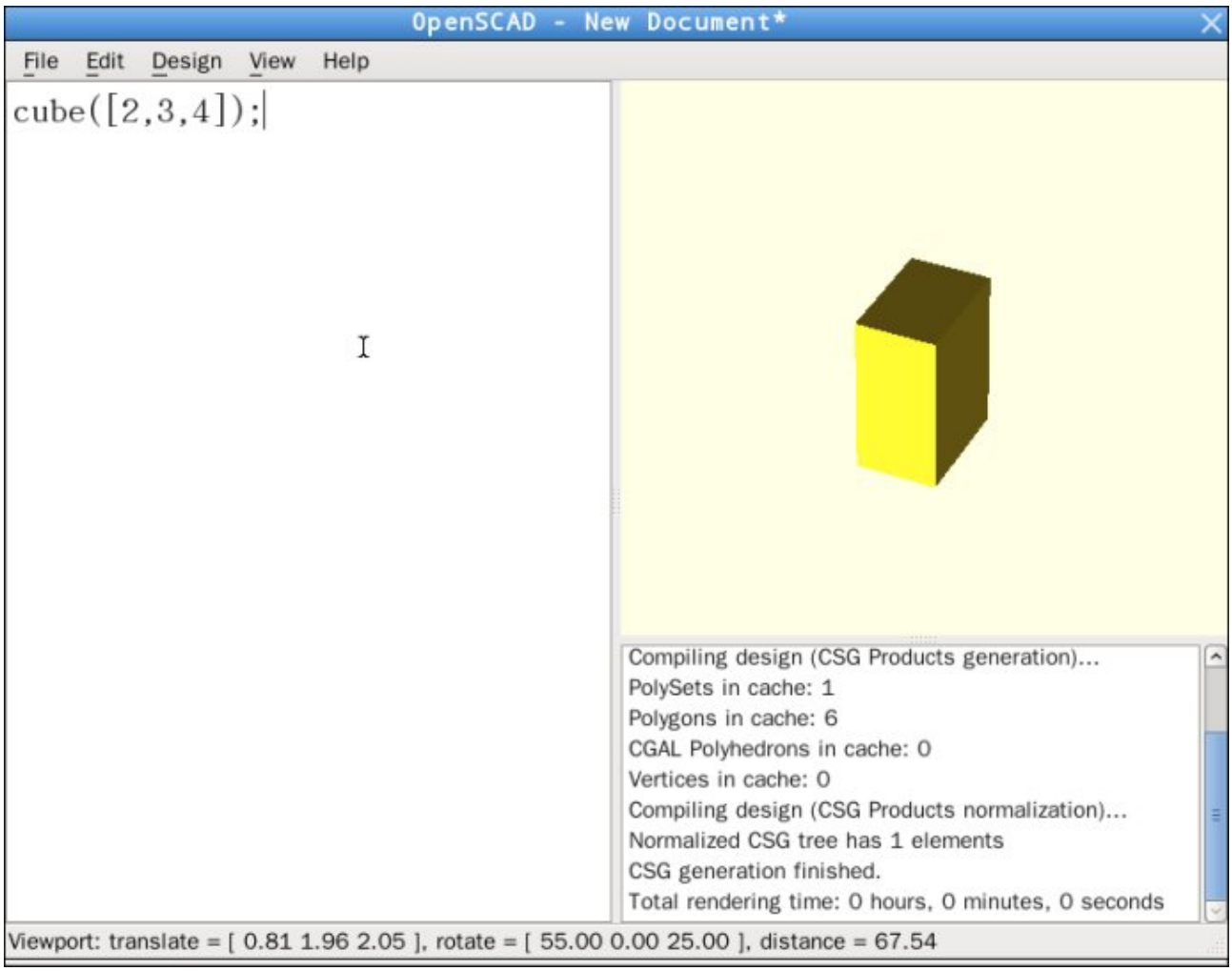
2.1 创建一个简单的模型

我们第一个模型是创建一个简单的 2＊3＊4 的立方提，在 openSCAD 的编辑器，输入一下命令：

```
cube([2,3,4]);
```

编译和透视我们的第一个模型立方体模型可以编译和透视，（当 openSCAD 编辑器在中心的时候）敲击 F6 键，是用 CGAL 模块透视和编译模型。然后点击 Design-> compile，或者直接键盘敲击 F5，表示把文本输入的命令转换编译成三维模型，到模型查看器，openGL 的模块。

截图 2-1：openSCAD 简单的立方体模型



菜单栏：

File, Edit, Design, View, Help

文件，编辑，设计，查看，帮助

左侧的编辑器，是 openSCAD 语言代码编辑的区域。

右侧上面的大块区域是图像模型生成区域。

右侧下部小块是系统编译源代码语言的状态信息。

最底下的状态栏，显示当前的视图角度和一些信息。

2.2 打开一个已存在的例子模型

打开一个或者多个模型是从 OpenSCAD 的 File->open （快捷键 Ctrl+O）就可以打开一个文件管理窗口，选择目标文件即可，也可以在终端中直接输入命令打开， openscad /home/user/xxx/example004.scad 命令是 openscad，文件加上文件的路径，或者在当前路径可直接输入文件名即可。

还可以把目标文件用 gedit, geany, leafpad, vi, nano 之类的文本编辑器 Text Editor 打开，然后将内容复制到 openSCAD 的文本编辑器中，

使用案例 1，– example004.scad:

```
difference() {
cube(30, center=true);
sphere(20);
}
translate([0, 0, 30]) {
cylinder(h=40, r=10);
}
```

完成复制例子代码后敲击 F5 键。

然后就可以预览你复制的代码的透视图。

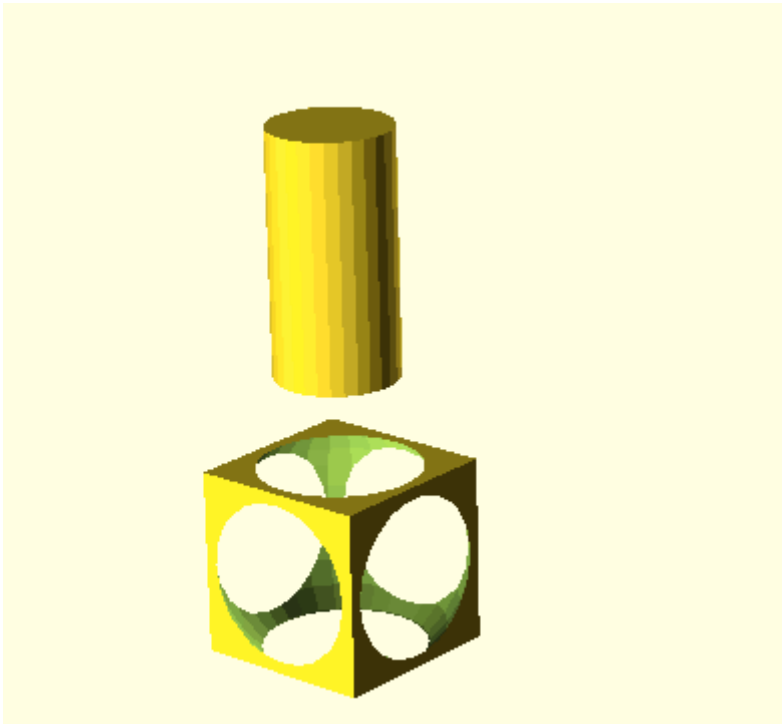
你可以有三种动态的预览框架：

1，按住鼠标左键，将光标移动到模型，可以旋转预览，底部的消息栏的 rotate = [22.22 000 333]，这些信息将随着你的鼠标选择的旋转方向的角度而改变角度。按住 shift 键盘可以旋转的方向。

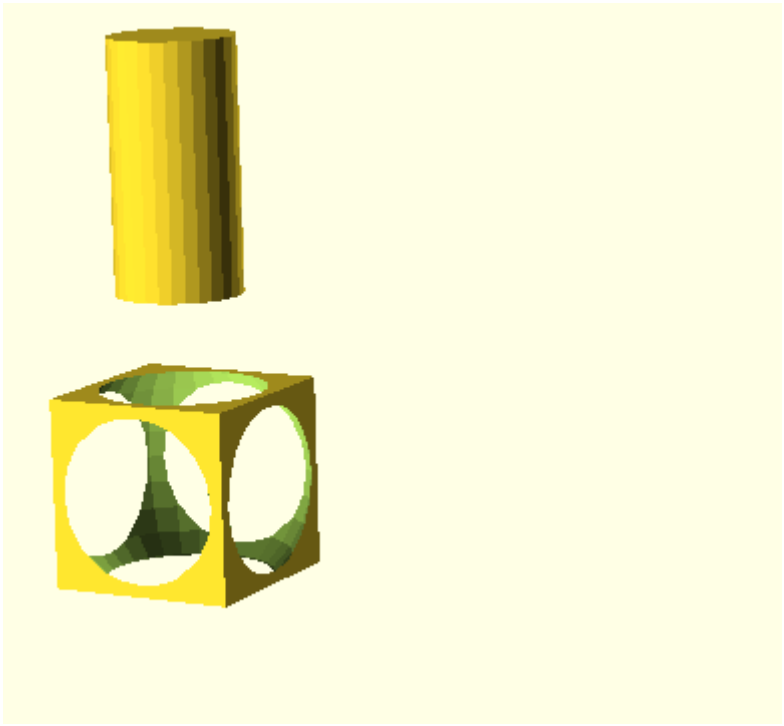
2，按住鼠标右键，将光标移动到模型，可以放置和移动预览观看，底部消息栏的 translate = [22, 222, 333] 的参数将随着鼠标的移动而改变。

3，使用鼠标的旋转滑轮（第三键）可以放大和缩小模型，也可以用+和-键，或者按住鼠标的右键/或者按住鼠标的第三键，同时按住键盘的 shift 键，可以放大和缩放，底部的信息栏同业在 distance = 555，的位置会伴随这放大和缩放的改变而改变。

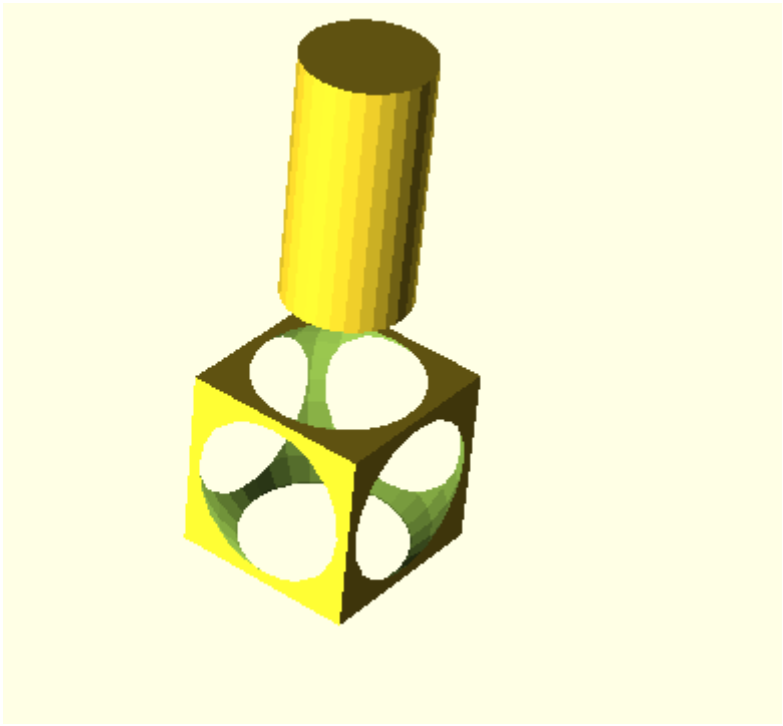
插图



2-2，旋转方向



2-3，位置移动



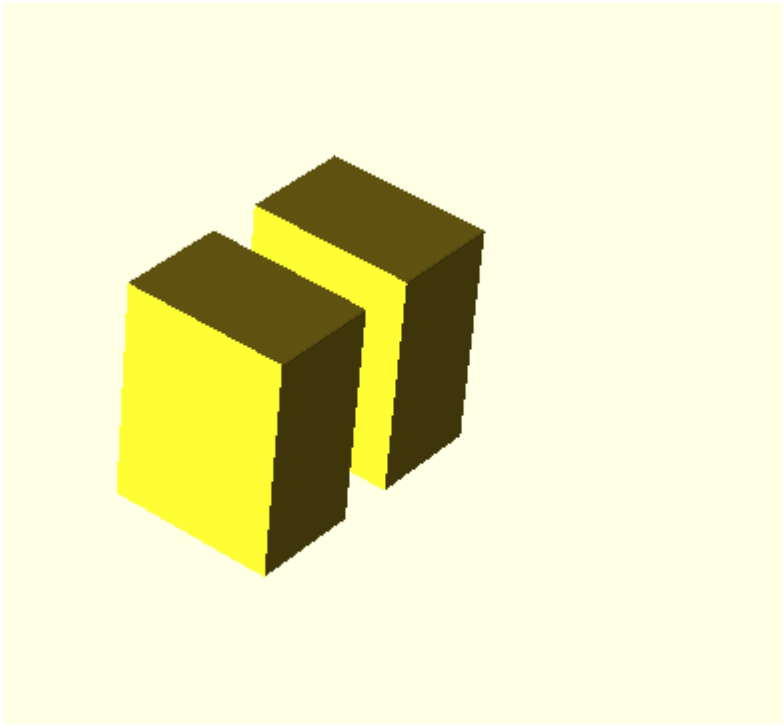
2-4，放大和缩放

2.3 位移一个对象模型 Positioning an object

我们已经看到了如何创建一个简单的立方体，我们下一段试着使用位置移动命令来从一个已经存在的立方体放置移动出另一个同样的立方体

使用例子 1- 位移放置一个对象模型

```
cube([2,3,4]);
translate([3,0,0]) {
cube([2,3,4]);
}
```



OpenSCAD 放置一个对象模型

在放置（translate）命令后的那里没有分号。

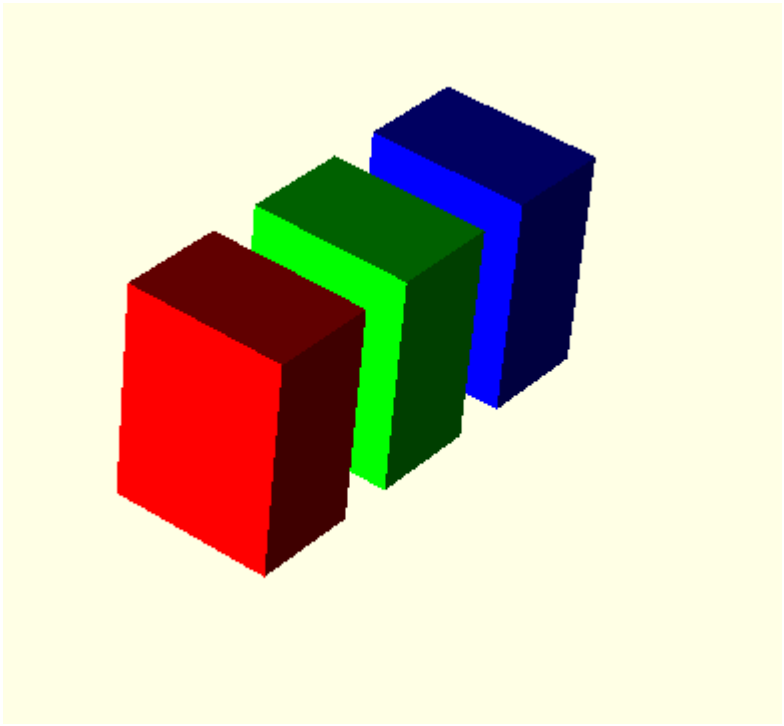
注意:那里没有分号跟随在放置位移命令。因为放置位移命令属于跟随的对象模型。如果分号没有被省略掉，然后放置位移的命令的效果将会终止，那么第二个立方体的位置将和第一个的位置是同样的，重叠的。

2.4 更改颜色到对象模型

我们可以改变一个对象模型的颜色源于给出的 RGB 值。之中包括传统的 RGB 值从 0-255 浮点数使用从 0.0 到 1.0.

Usage example 1 - changing the color of an object:
使用案例 1- 改变一个对象模型的颜色：

```
color([1,0,0]) cube([2,3,4]);
translate([3,0,0])
color([0,1,0]) cube([2,3,4]);
translate([6,0,0])
color([0,0,1]) cube([2,3,4]);
```



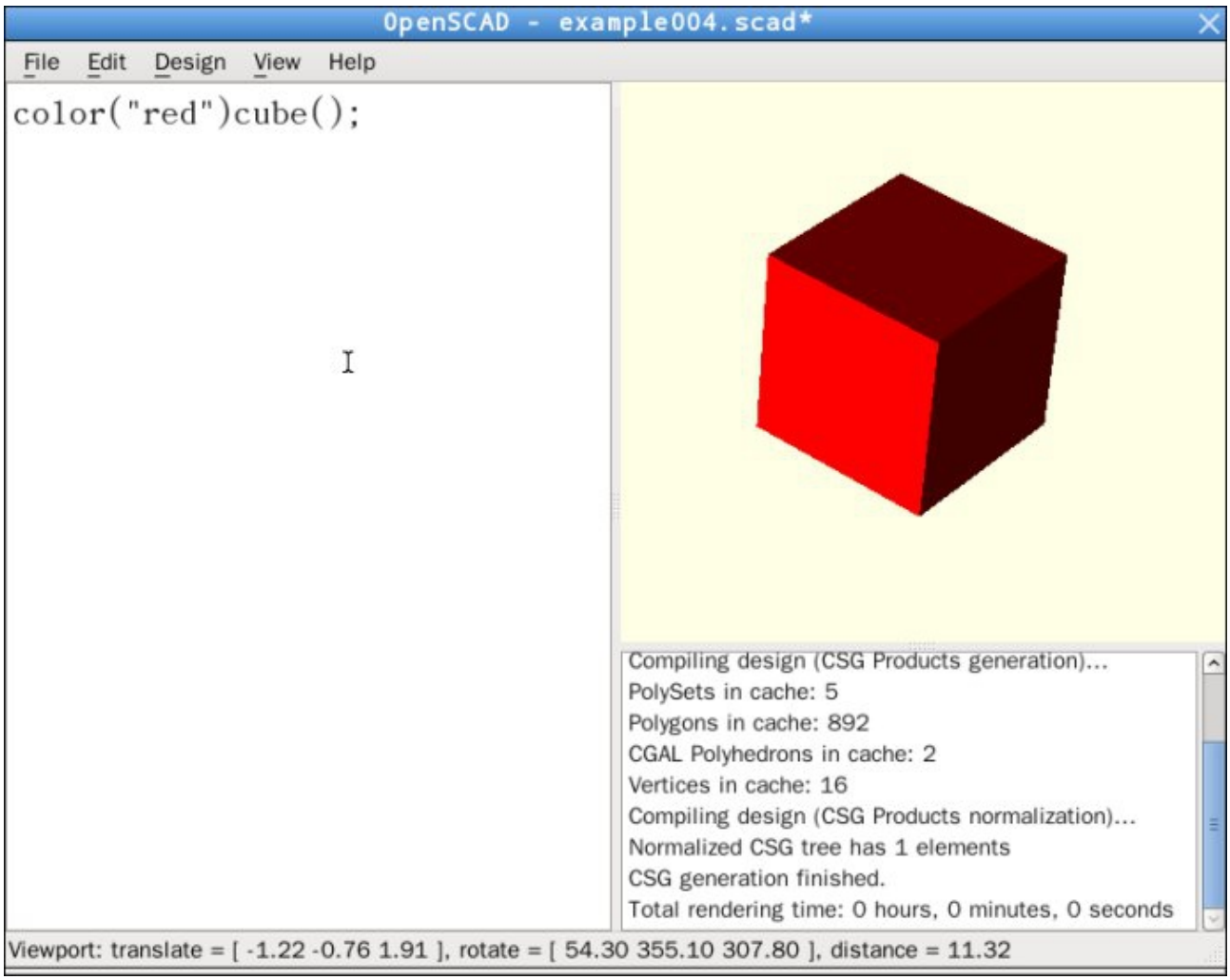
图片 2.4.1-OpenSCAD 改变一个对象模型的颜色

颜色的名字可以从 2011.12 版本（或者更新的版本）。名字就像是网页的颜色，例如：

```
color(“red”) cube();
```

更老的版本，颜色也可以用 RGB 数字来代替 。

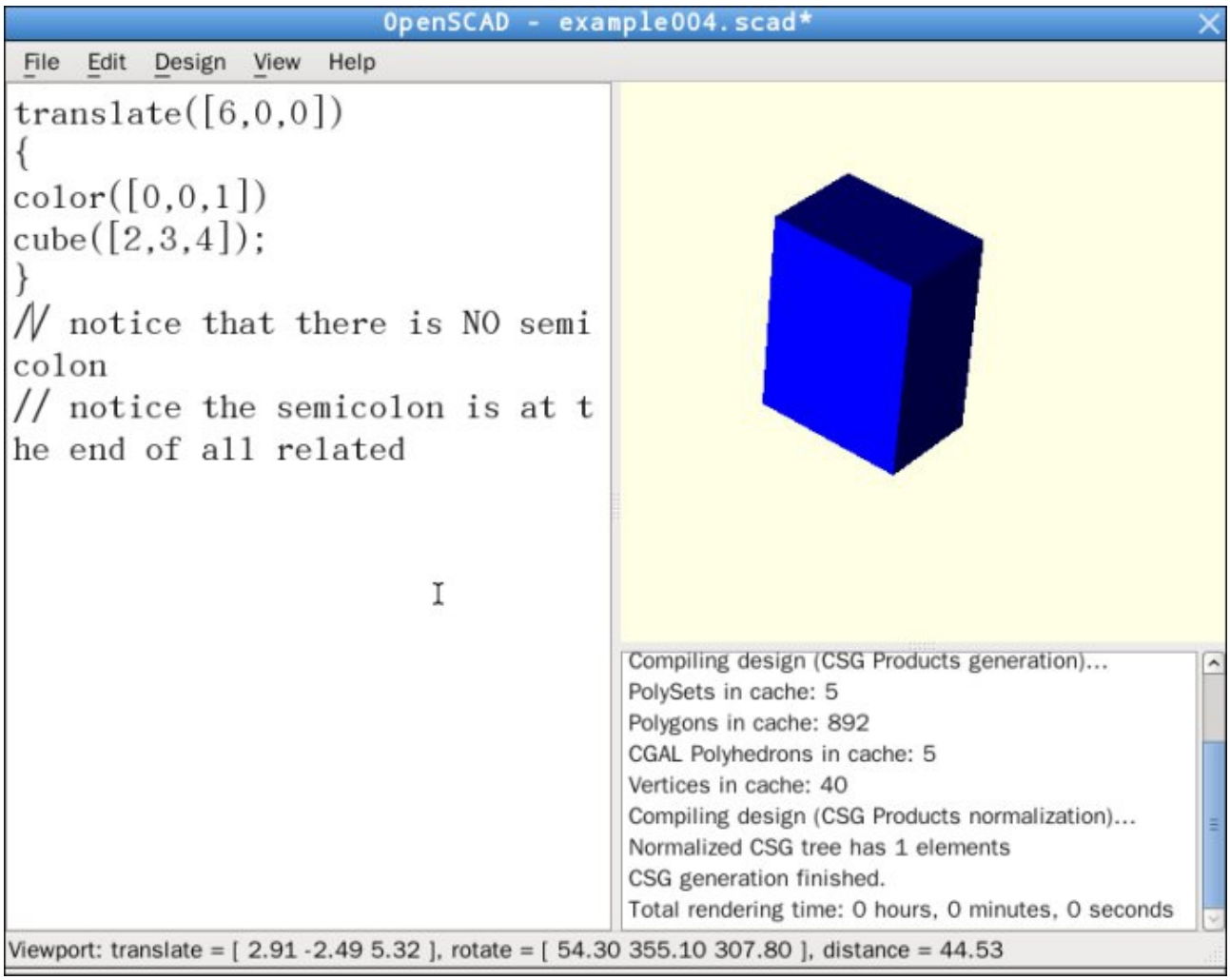
```
color([1, 0, 0]) cube(20);
```



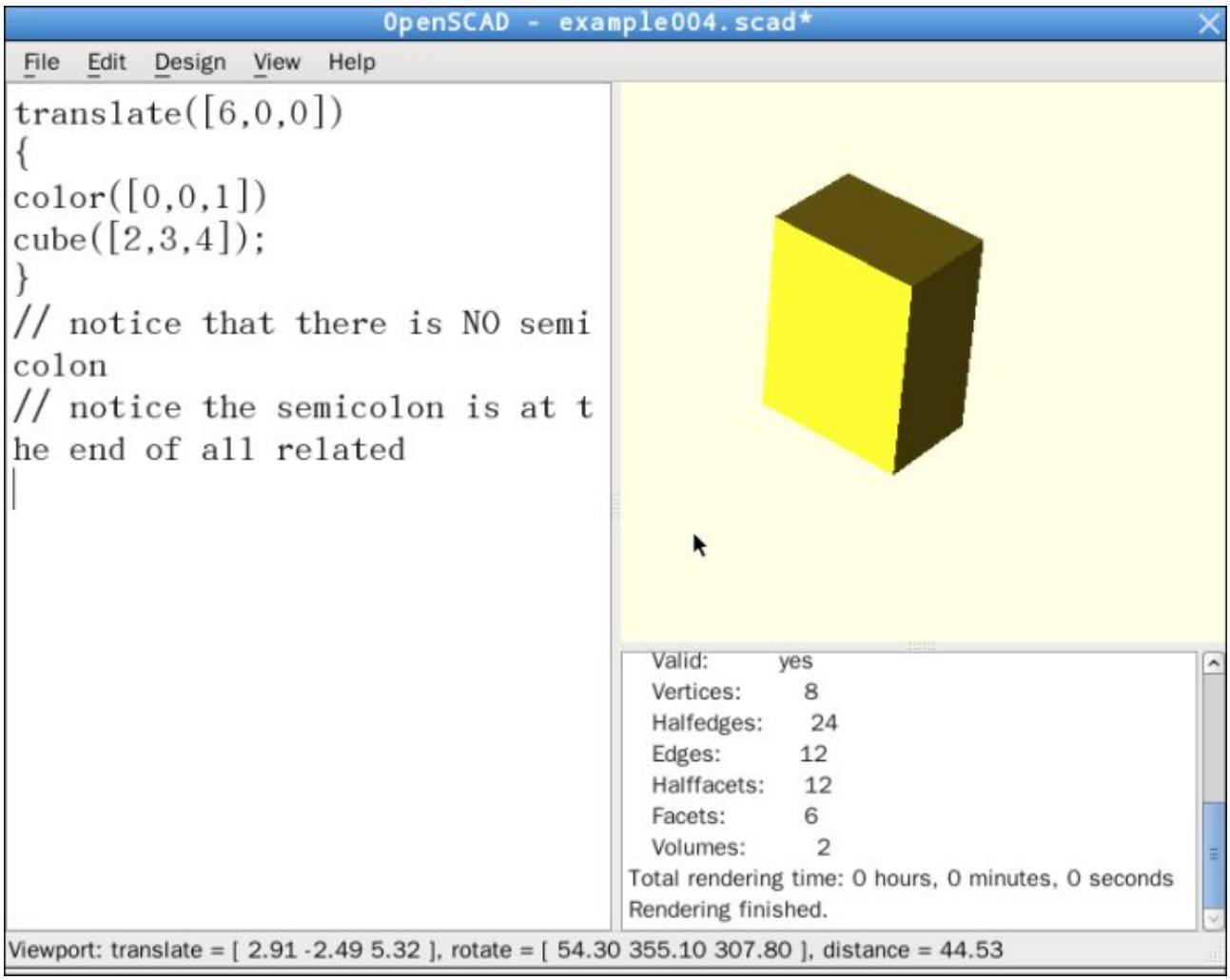
如果你想相关的命令就像是句子，然后，在句子中 color()就是一个"形容词"，可以描述"物体"(名词）。这样，物体就是 cube()将会创建。在句子中形容词的位置在名词的前面，就像：color() cube();.同样，translate()可以想象成一个"动词"可以是物体运动，就像：translate() color() cube();。

接下来的代码将产生同样的结果：

```
translate([6,0,0])
{
color([0,0,1])
cube([2,3,4]);
}
// notice that there is NO semicolon
// notice the semicolon is at the end of all related
```



F5 openCSG render



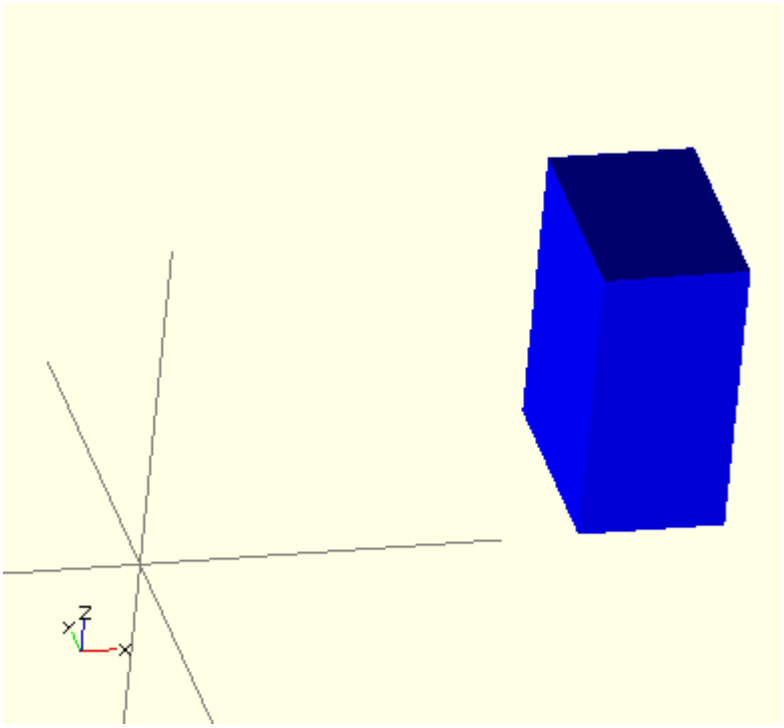
F6, CGAL render.

颜色的更改仅仅局限于预览模式（F5），透视图模式（F6)不支持配置的颜色。

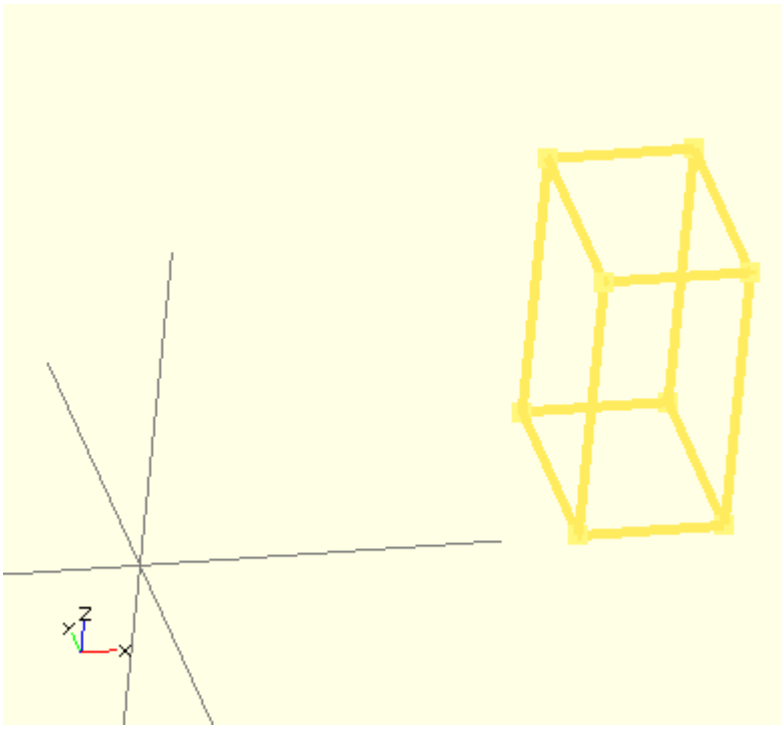
2.5 模型查看
The text in its current form is incomplete.
当前文本格式不太完善。
目录
2.5.1 表面查看
2.5.2 仅 CGAL 网格
2.5.3 The OpenCSG 查看
2.5.4 仅网格查看那
2.5.5 投影一起查查看

2.5.1 表面查看

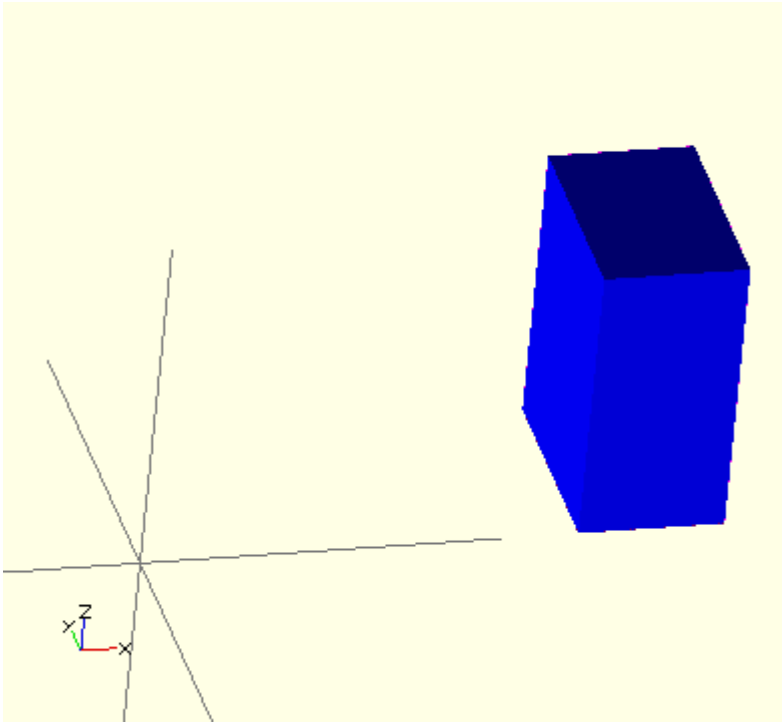
表面查看是初始模型查看产生，当模型代码第一次透视。



仅 CGAL 网格
仅 CGAL 网格查看仅表示在边面之下的”脚手架一样的框架“，想象埃菲尔铁塔。

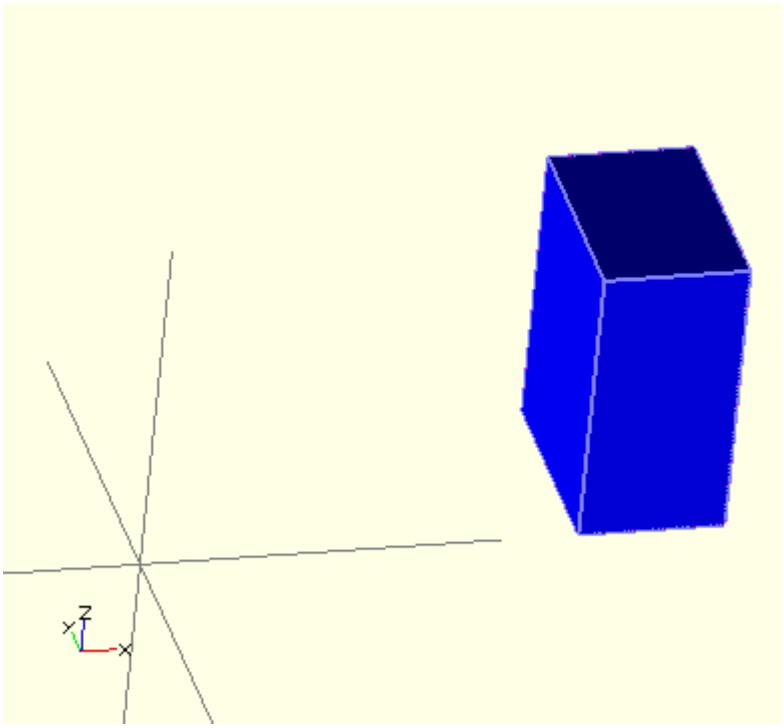


OpenCSG 查看
这个查看模式的功能源自 open 结构立体几何软件库，生成的 openGL 功能的模型预览。如果 openCSG 库不可用或者显卡或者驱动不知 openGL，这个查看将无法产生输出图形。

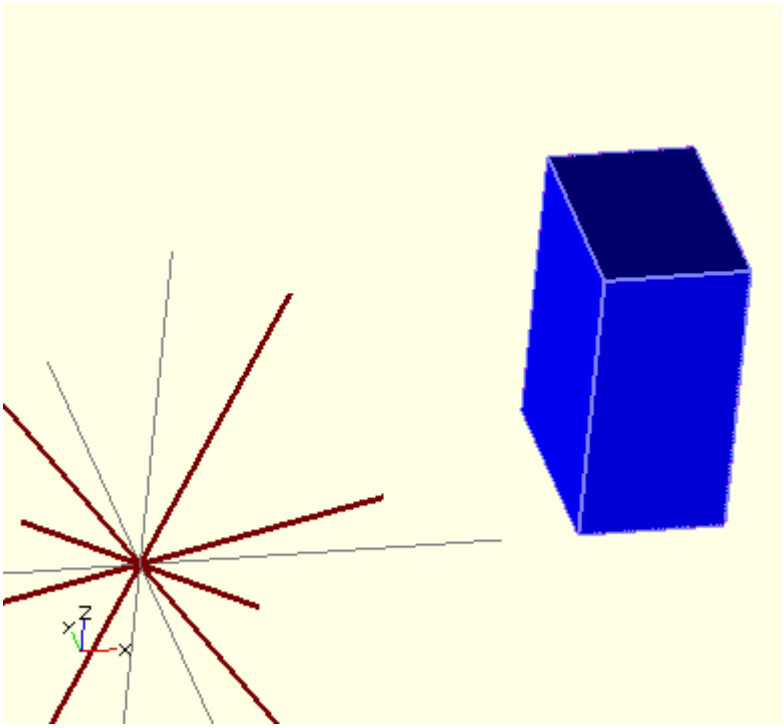


The grid only view
只有视图显示网格线弥补的对象，也被称为线框。
线框是一个三维或物理对象的视觉呈现。使用线框模型允许底层设计结构的三维模型的可视化。由于线框效果图是比较简单和快速计算，他们经常使用的情况下，高画面的帧速率是必要的（例如，当使用一个特别复杂的 3D 模型，或在实时系统中的车型外观现象）。当需要更大的图形细节，
表面纹理可以被自动添加完成后的初始呈现线框。这允许设计师快速审查 chansolids，或旋转对象，新的期望长期拖延的看法，并没有更逼真的 rendering.The 线框格式也是非常适合广泛应用于刀具路径编程 DNC（直接数字控制）机床。线框模型也被用来作为输入 CAM（计算机辅助
制造）。线框是最抽象的和最现实的三个主要的 CAD 模型。这种建模方法，包括仅定义一个对象的边缘的线，点和曲线。
从维基百科（http://en.wikipedia.org/wiki/Wire-frame_model）

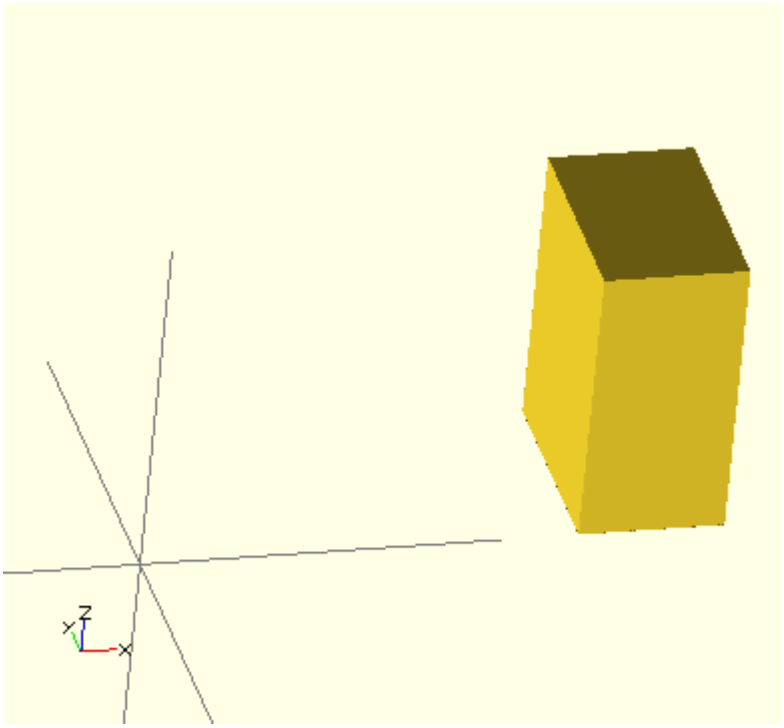
下图分别是查看选项中，选项 View-> show edge / show axes /show crosschair，
最下方的图是 openGL 模式的产看。



Show Grid. Ctrl+1.



Show Grid and Cross. Ctrl+2.



F6, the CGAL render.

3, The OpenSCAD 用户界面

目录

查看导视

可视面积主要是使用鼠标导视：

•用鼠标左键拖动旋转视图沿轴的可视面积。它保留垂直轴方向。
•当按下 shift 键，用鼠标左键拖动旋转视图沿垂直轴和轴指向用户。
•拖动右侧或鼠标中键移动查看区域。
•缩放，有三种方法：
•使用滚轮
•右或中间的鼠标按钮，按住 Shift 键拖动
•键+和 -
旋转可以重置使用快捷键 Ctrl+0。运动可以使用快捷键 Ctrl+ P 重置

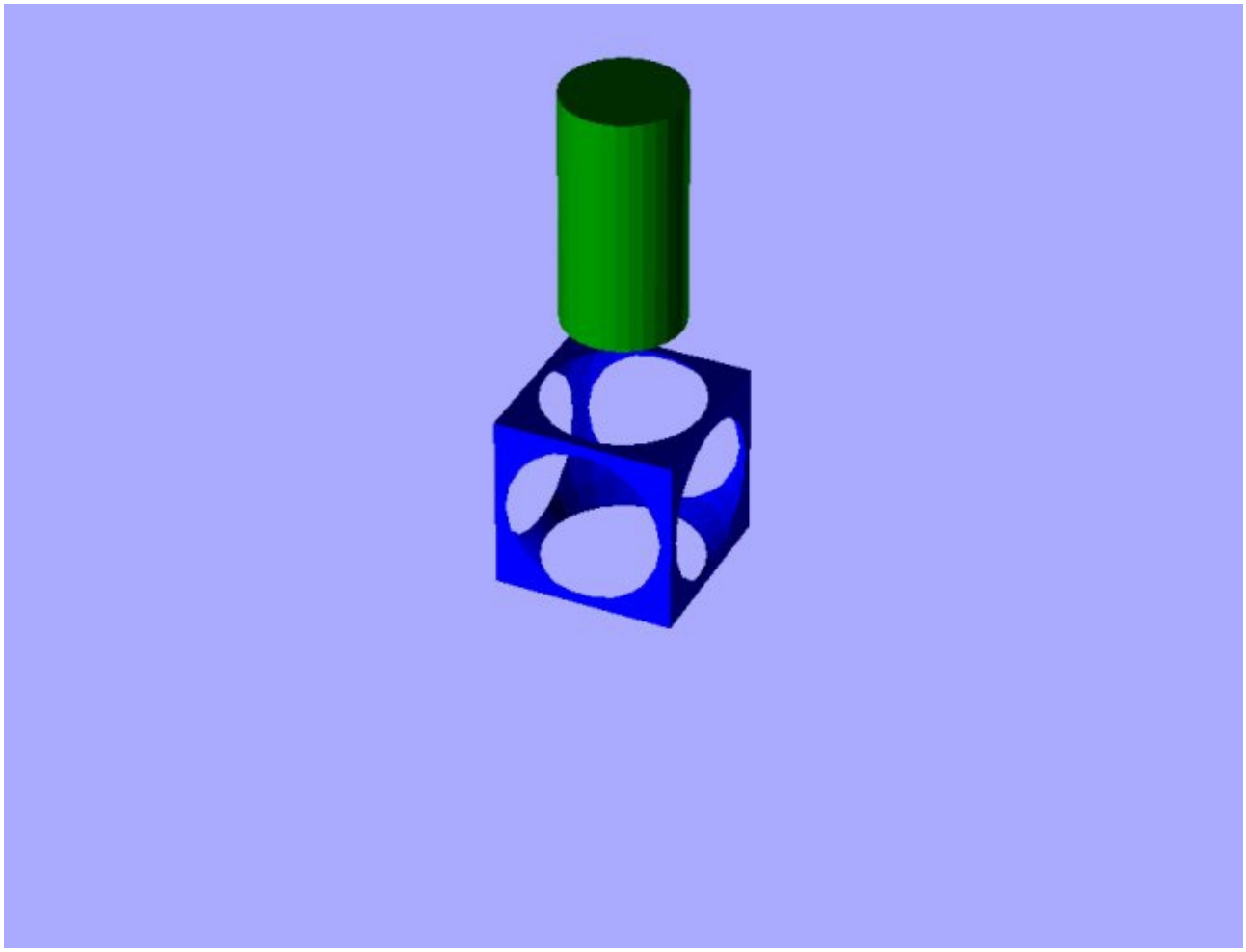
查看设置
可视面积可以被配置为使用不同的渲染方法和使用其他选项
查看菜单。这里描述的可供选择的方案大多是使用快捷键。

渲染模式

OpenCSG（F9）
OpenCSG 模式, OpenCSG 库，用于产生可见的模型。此库使用先进的 OpenGL（2.0）功能，如 Z 缓冲和不要求明确说明导最终的网格。，它跟踪的对象是如何被合并。例如，当渲染球形凹痕在一个立方体，它首先将显卡渲染立方体，然后渲染球体，而是使用 Z 缓冲器隐藏部分的球体覆盖的立方体，它会这些仅渲染部分的球体，在视觉上产生的一个立方体的球形凹痕。
这种方法产生的瞬时结果，但具有低帧率工作时高度非凸对象。Example :

```
color("blue")
difference() {
cube(30, center=true);
sphere(20);
}
color("green")
translate([0, 0, 30]) {
cylinder(h=40, r=10);
```

}

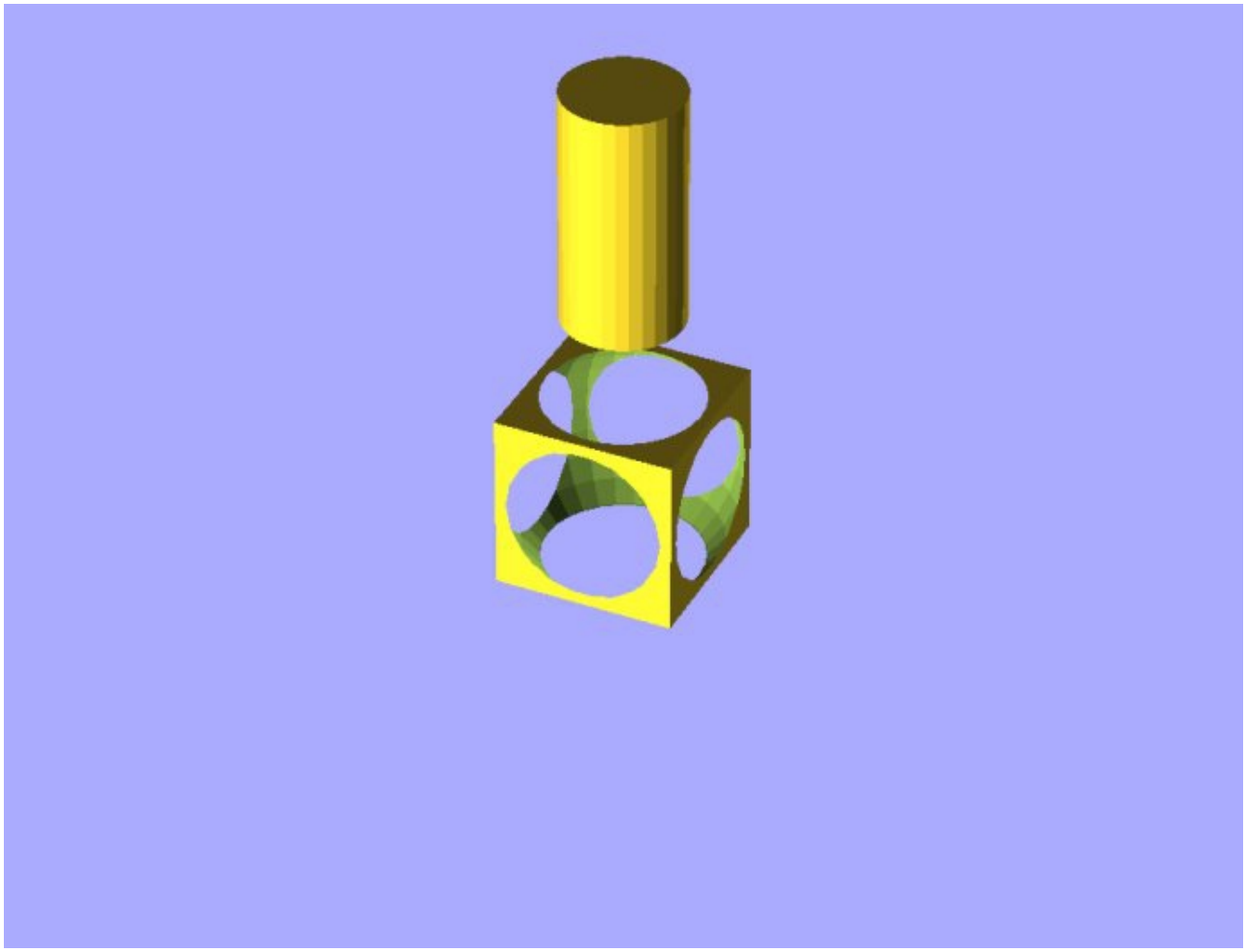


该选择说明 OpenCSG 模式 使用 F9 切换到最终的产生 OpenCSG 视图，
但不会重新的编译源代码。你可能要使用的编译功能（F5 发现
“样式”菜单），以重新评估的源代码，建立 OpenCSG 对象，然后切换到
OpenCSG 视图。

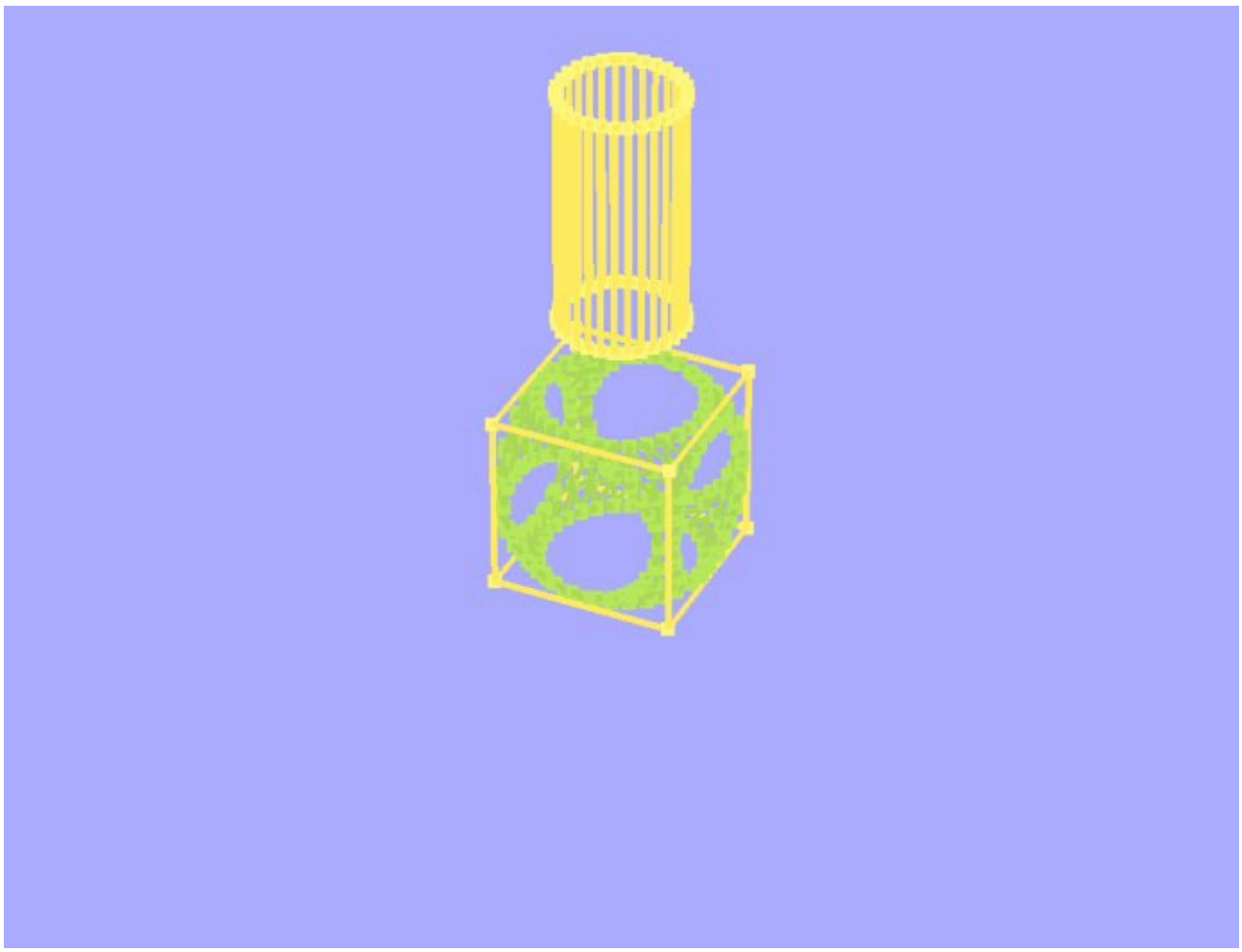
CGAL (Surfaces and Grid, F10 and F11)

首字母缩写的 CGAL,指到开源的计算几何算法库。
CGAL 模式，CGAL 库是用来计算最基础对象的网格，然后使用简单的 OpenGL 显示。
这种方法可能需要一些时间，当第一次使用一个新的程序，但届时将有更高帧频.
OpenCSG，F10 和 F11 之前只启用 CGAL 显示模式和不更新相关对象，编译和使用的渲染功能（F6，在“Design”菜单中找到）。
这两种显示方法的优点结合起来，你可以用你的选择性部分程序在渲染功能，并迫使他们是生成网状甚至 OpenCSG 模式启用。

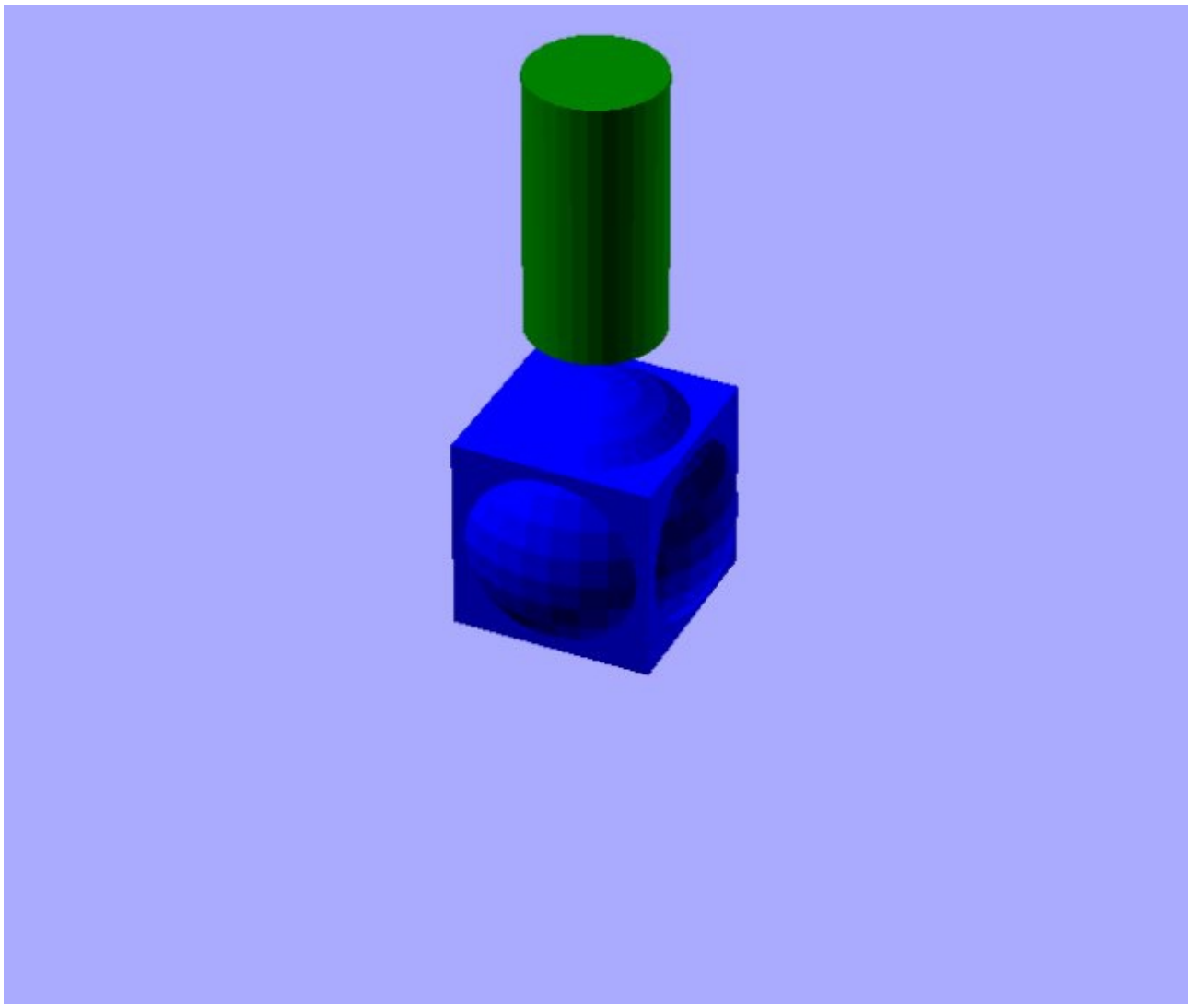
Example 302, F10 output view:



Example 303, F11 output view:

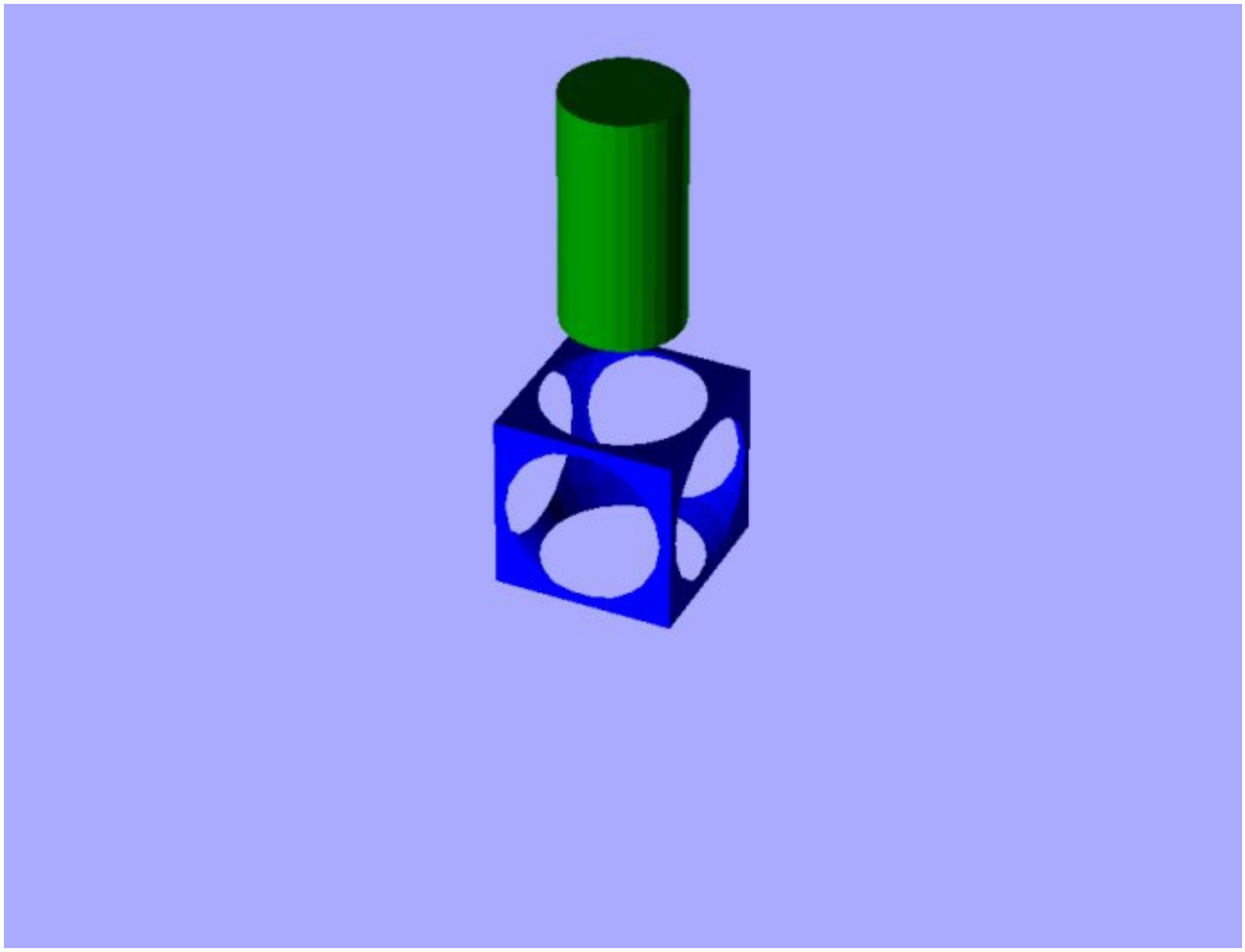


Example 304, F12 output view:

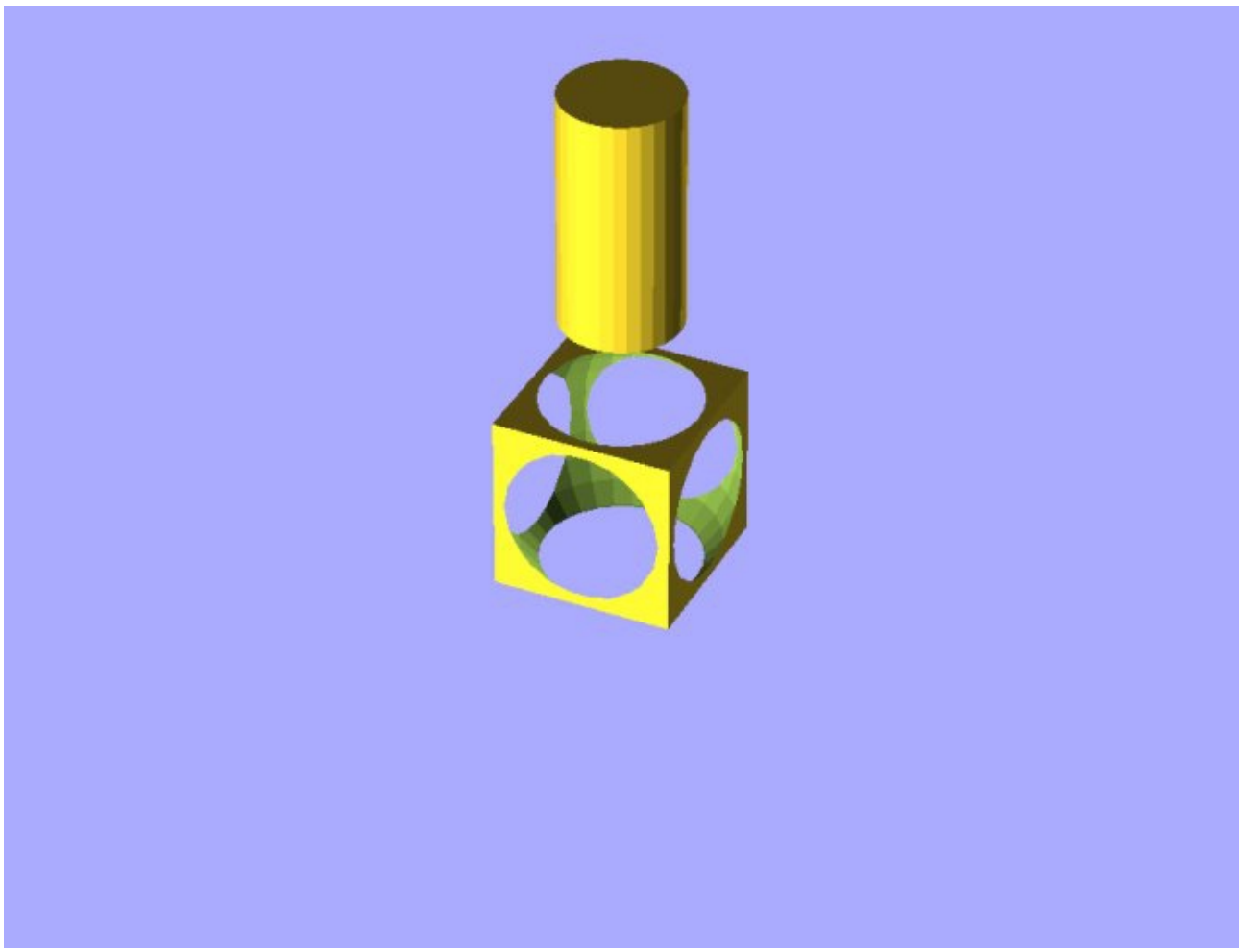


查看选项
显示边缘（按 Ctrl+1）
CGAL 和 OpenSCAD 之间的方法差异就是用布尔代数运算建立的可以产看的边缘。。
如果启用显示边缘，OpenCSG 和 CGAL 模式将呈现边缘以及面孔，CGAL 甚至会出现顶点。 CGAL 在网格模式下，这个选项没有任何作用。
启用该选项将显示 OpenCSG 和 CGAL 之间的差异相当清楚：虽然在 CGAL 模式，你看到无处不在绘制“属于”边缘.OpenCSG 不会从布尔运算显示边缘– 因为这是 来没有明确的计算但只是哪个对象的 Z 裁剪开始或结束。

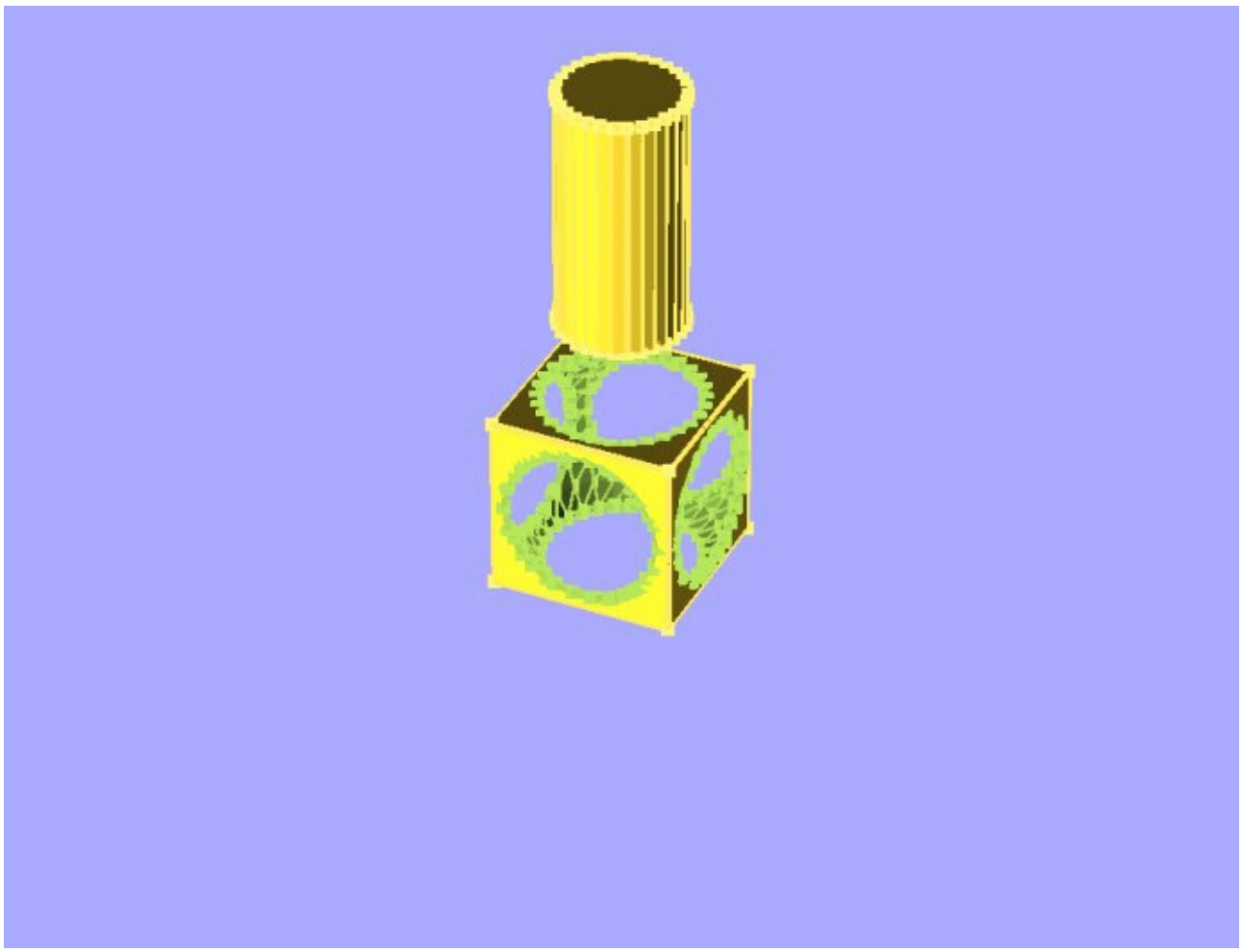
F9 output showing, not with Ctrl+1 and with Ctrl+1 both same.



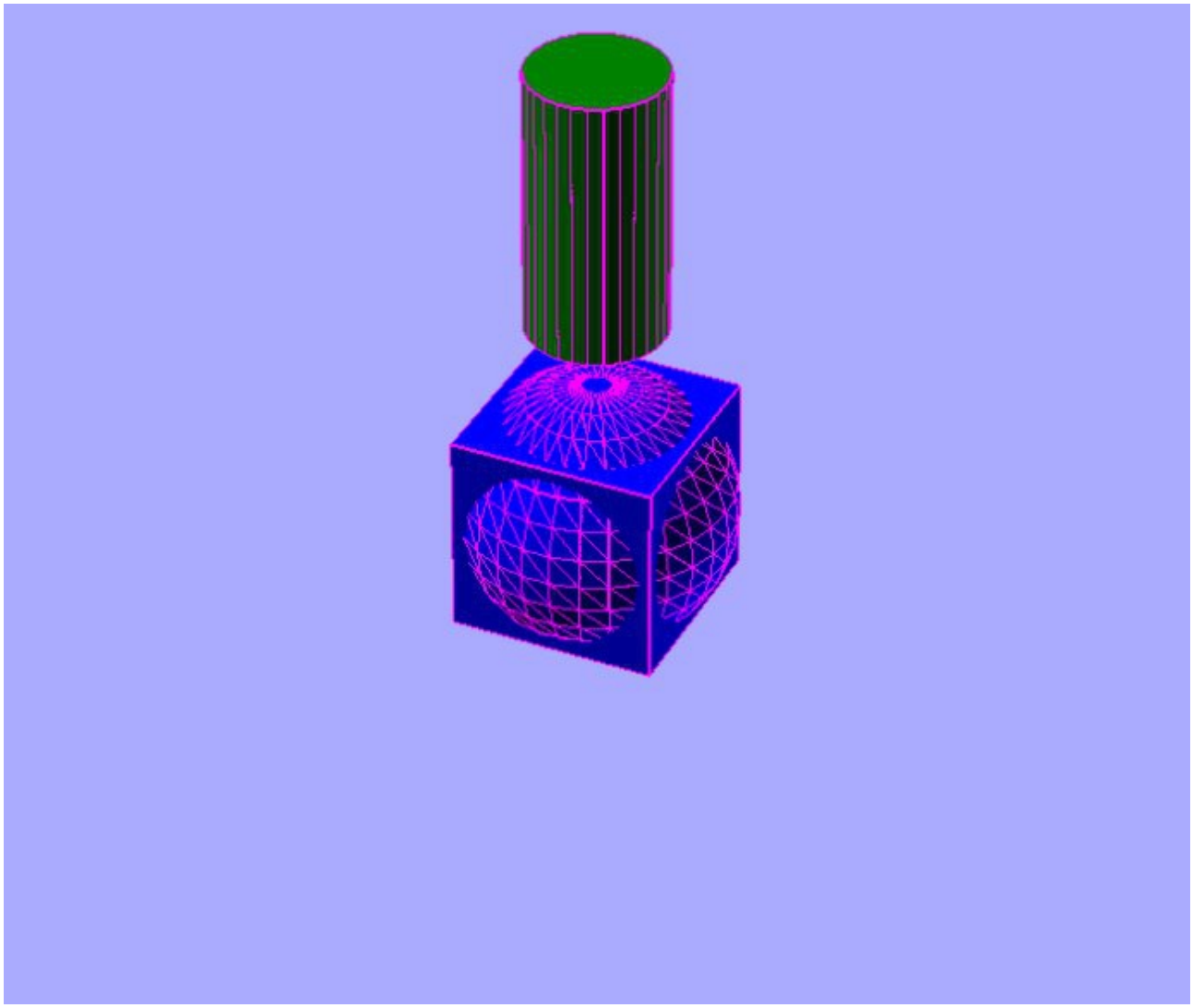
F10 output view , not with Ctrl+1:



F10 output view, with Ctrl+1: showing the grid:

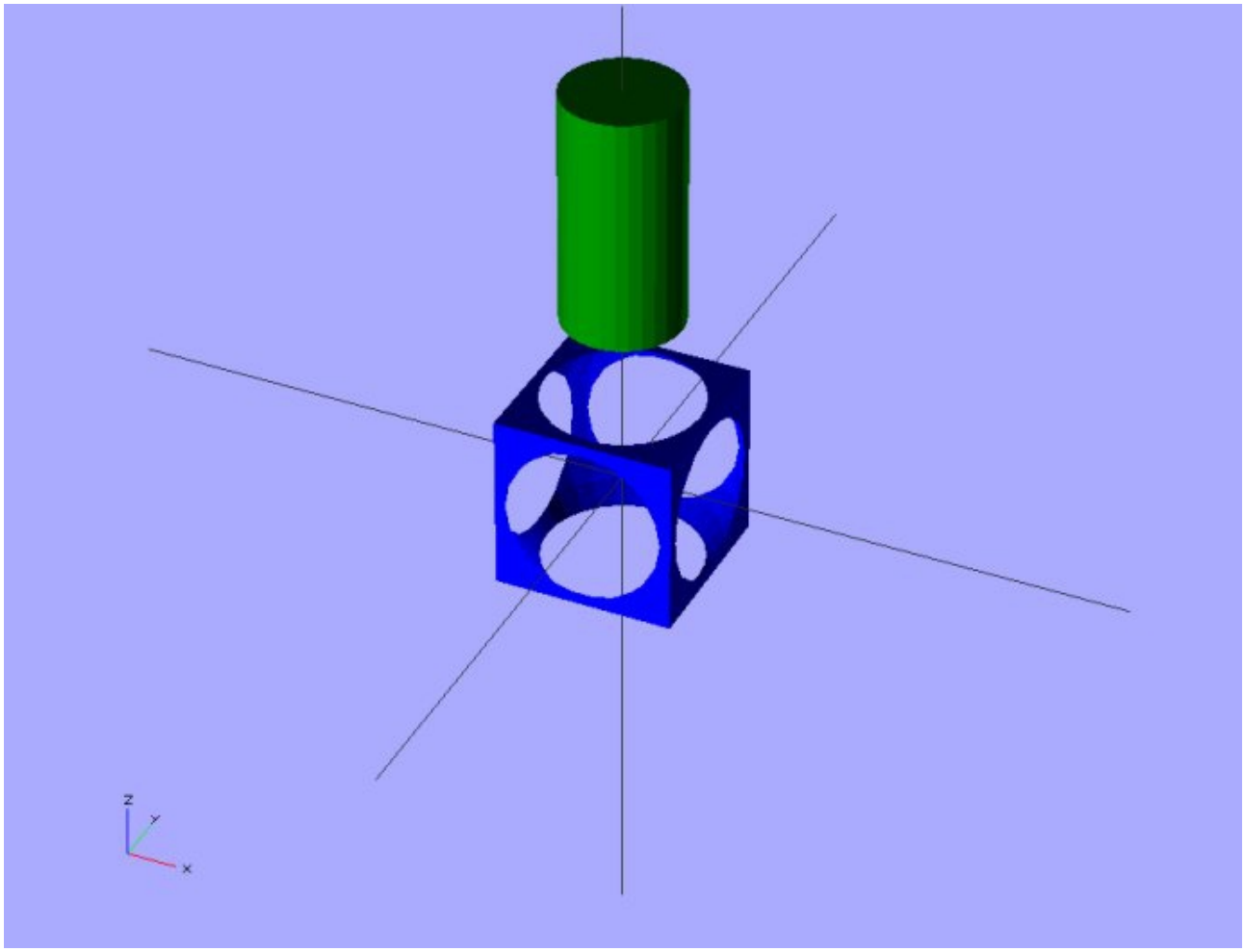


F12, output view, with Ctrl+1:



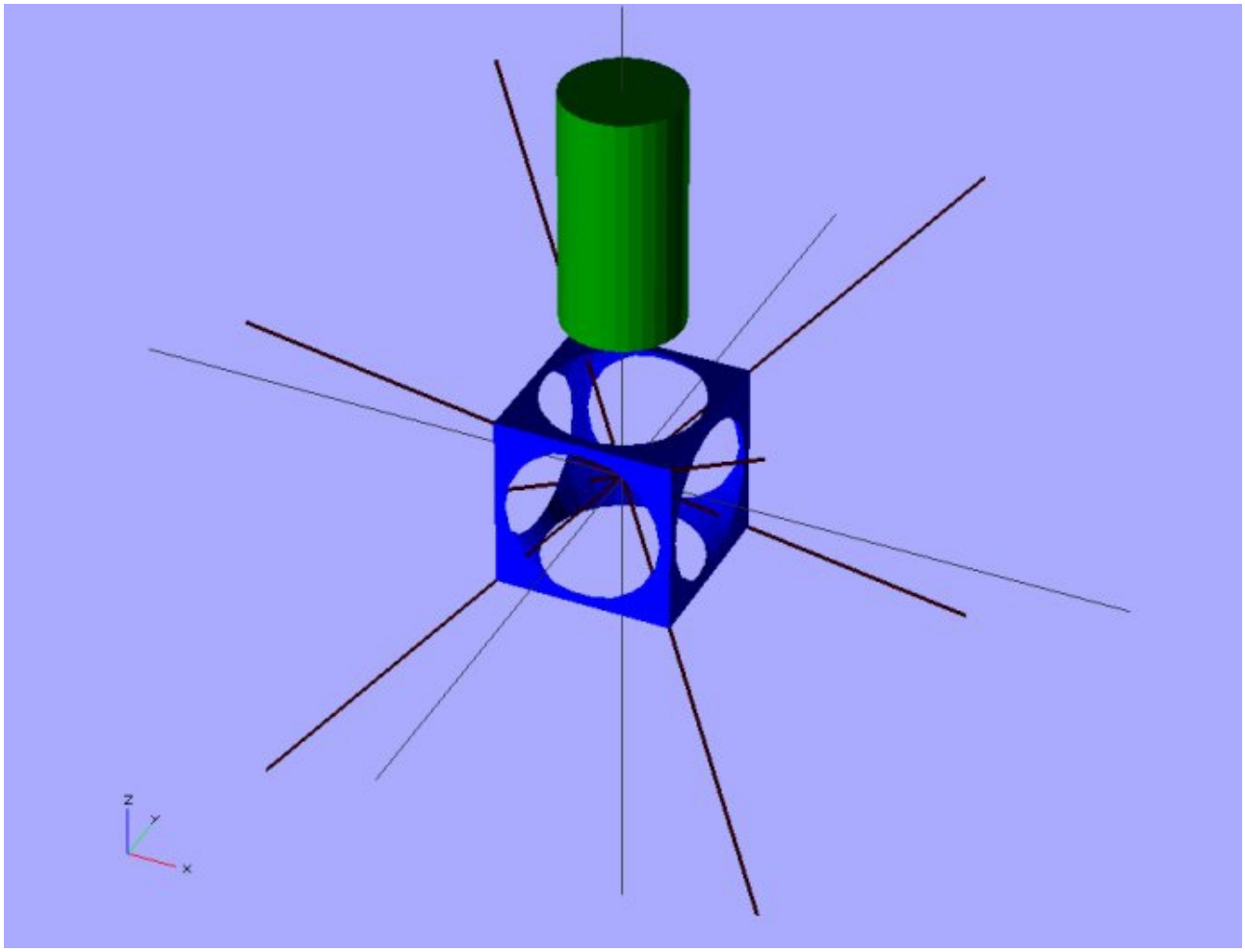
显示坐标轴（按 Ctrl+2）
如果显示轴启用，想表示全局坐标系的原点正交轴的指标。此外，较小的指标轴与轴的名称将显示在可视区域的左下角。较小坐标轴的指示器标记为 x，y，z 分别用红，绿，蓝彩色区分。

Example, Showing the coordinate axes with Ctrl+2:



显示十字线（按 Ctrl+3）
如果显示的十字线启用，该中心将显示四行指向视口房间对角线方向的全局坐标系统，这是非常有用的，当对准模型中的一个特定的点，以保持它在屏幕居中在旋转过程中可视面积。
如果说坐标轴是 90 度平分整个 3D 环境的话，那么十字线就是 45 度的对象模型的指示线。

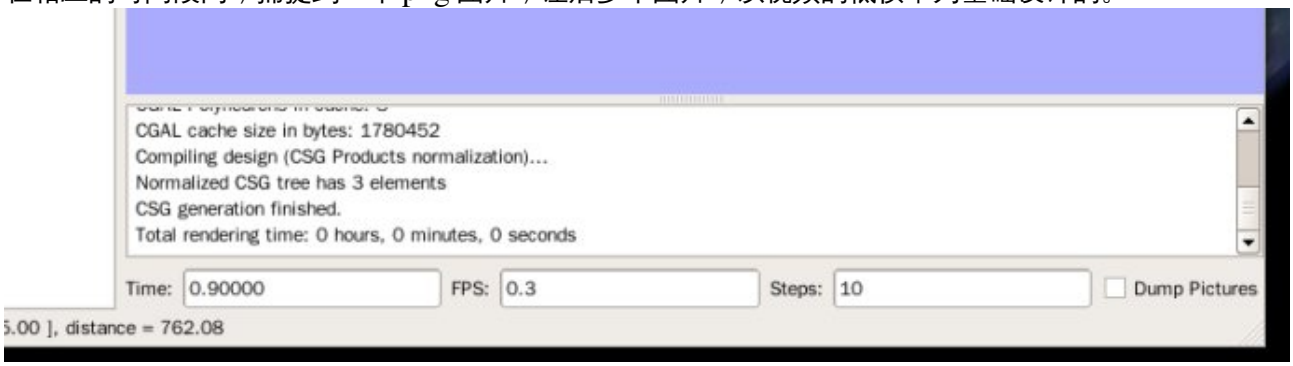
Example showing cross with Ctrl+3:



动画
“动画”选项添加到屏幕下缘的动画栏。只要 FPS 和 Step 设置（合理的值，首先，分别为 10 和 100），前时间是加 1/Steps，FPS 次每秒，直到它达到 1，当它每一个被改变的时间回到 0 时，该程序被重新评估的用变量$t 设置为当前时间轴。了解更多有关如何使用$ t，另见 Other_Language_Features.

这里的动画我也做了测试，$t,这个函数我还是没有太明白，测试了一些，也没有效果，估计后边随着学习的深入，会有一些领悟吧！

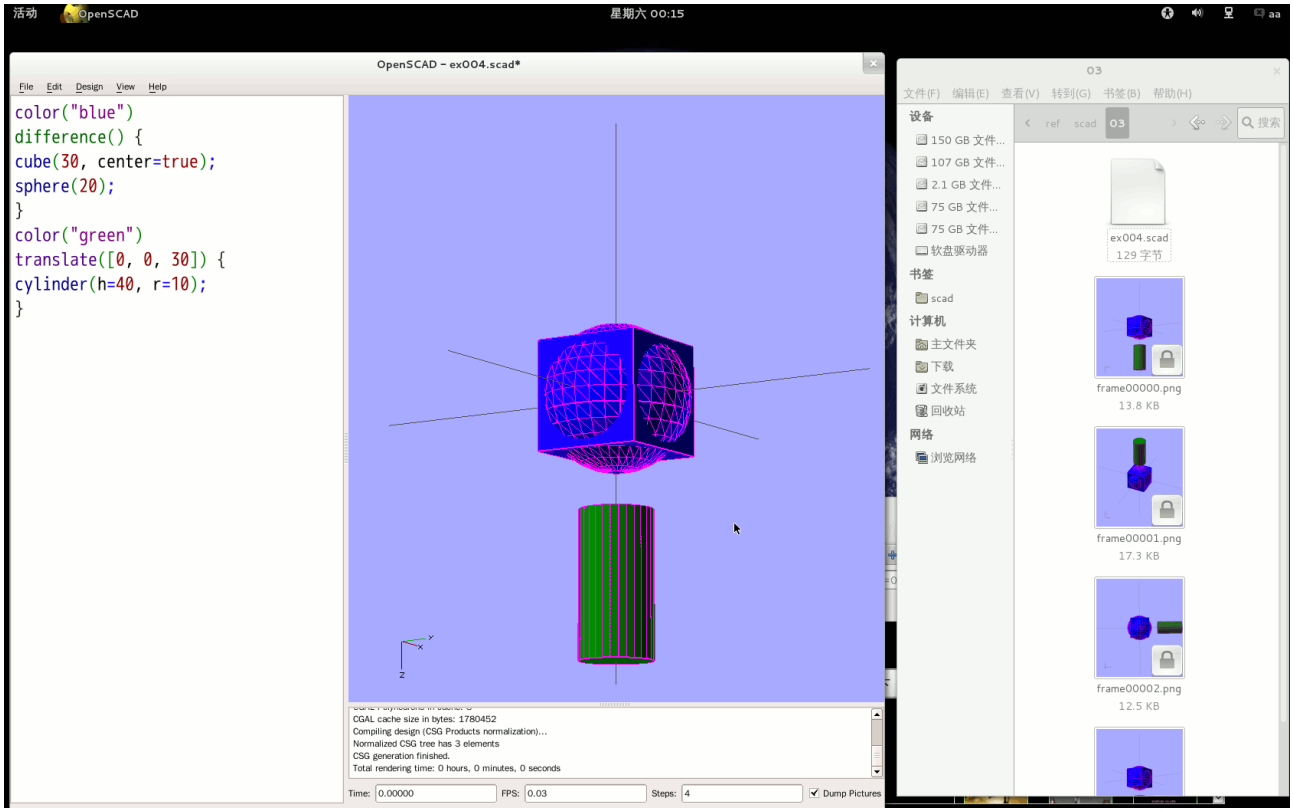
我个人感觉这个 Animation,动画,菜单是 View->Animate，仅仅是将模型的操作，或者多个模型的组合，在相应的时间段内，捕捉到一个 png 图片，让后多个图片，以视频的低帧率为基础设计的。



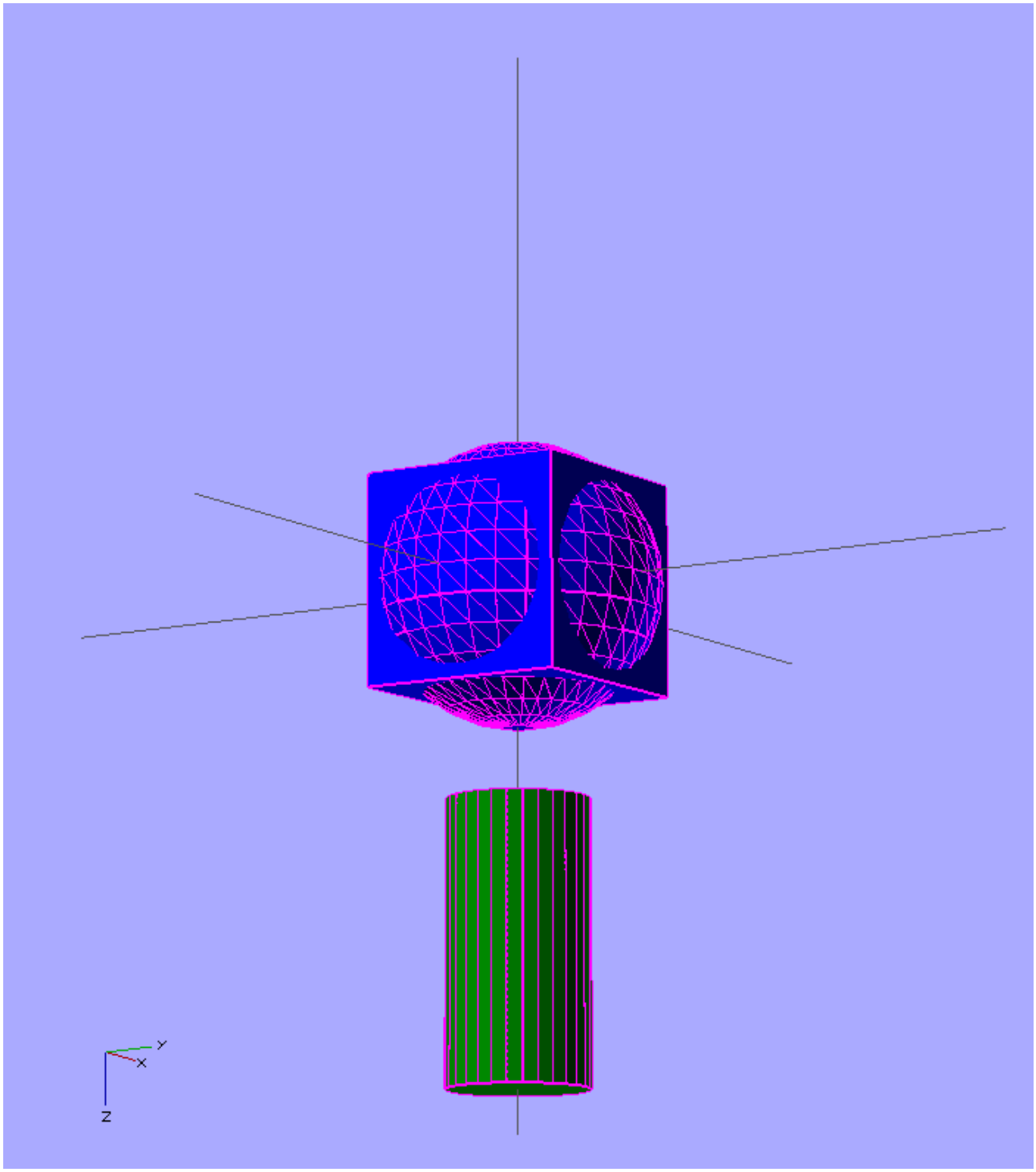
Time: 时间 ，FPS:帧数， Steps:步数，Dump Picture 抓图（选择或不选）
Time=FPS/Steps.
实际间隔时间=1/FPS.
这个时间是相对时间，而非我们通用的时间。这里，根据 FPS（帧数）和 Steps(步数）的参数而变换。比如说，如果 FPS 是 1， Steps 是 2， 3， 5， N， 那么时间就是把 1 分成几分，
举例:
FPS:1, Steps: 2， Time:就是 1/2，第一步就会显示 0.00000，第二步然后显示 0.50000，然后无限次循环,因为 FPS 是 1， 所以实际的时间间隔是 1 秒，而这里现实的 Time 是相对的把 1 秒分成多少步;
如果用公式的话 ，Time=FPS/Steps,而实际的时间间隔，就是实际间隔时间=1/FPS，FPS 的数字越大，实际时间间隔就越快，FPS 的数值越小，实际时间间隔就越慢。
举例:
FPS=2， steps=3，就分成 1/3，第一步 0.33333，第二部 0.66666，第三部 0.99999，实际时间间隔 0.5 秒。
举例:
FPS=0.1, Steps=5, Time 就会依次显示 0.2000, 0.40000, 0.60000, 0.80000, 0.00000，实际时间间隔是 10 秒，就是每十秒然后系统自动抓图一次,无限循环。

这里有一个 gif 图片，我设置的 FPS=0.04, Steps=4，就是每次间隔 1/0.04 秒，OpenSCAD 抓图一次，一共抓图 4 次，抓图完毕后继续循环，我在之间进行了模型的方向调整，从这里还可以看出来系统自动生成 frame0001.png, frame0002.png， 根据步数的多少而生成，选择 FPS 速度的时候要根据你的实际需要的效果去做，这个是用 gimp 图像处理软件，把多个图片组合成一个动态的 gif 图像，每次间隔 3 秒，图片换一次，gif 的，而不是 openscad 的每个 25 秒抓图一次，哈哈。如果你要求的是视频的话，FPS 应该选择快一点，步数量就要根据视频长短而定了，很大的数字，视频编辑软件也有很多啊，ogg 开源格式的比较流行啊，可以尝试一下。

Example gif full deskscreen:



Example gif openSCAD automatic sreenshot:



查看对齐
菜单项的顶部，底部，对角线中心（按 Ctrl+4，按 Ctrl+5，...，按 Ctrl+0，按下 Ctrl + P）对齐查看全球坐标系统.上，下，左，右，正面和背面对齐与轴平行，对角线对齐当它被斜向对齐作为 OpenSCAD 开始。

在操作之前，最好用鼠标单击一下模型，表示操作命令在当前窗口程序下生效。

Ctrl+0

对角戏查看，斜面查看，

一般是向左旋转 45 度，向下旋转 45 度的角度查看，不管你之前的操作如何的乱，只要点击 View->Diagonal，或者 Ctrl+0，就会启动对角线查看。

Ctrl+1 Ctrl+2, Ctrl+3,
我们在上一章就说过了，不在多少了，简单提示:
Ctrl+1，显示网格
Ctrl+2，显示坐标
Ctrl+3，显示交叉线

Ctrl+4 俯视图，从顶部向下看模型
Ctrl+5 底视图，从底部向上看模型
Ctrl+6 左视图，从左侧向右侧看模型
Ctrl+7 右视图，从右侧向左侧看模型
Ctrl+8 正视图，从正前方观看模型
Ctrl+9 后视图，从正后方观看模型

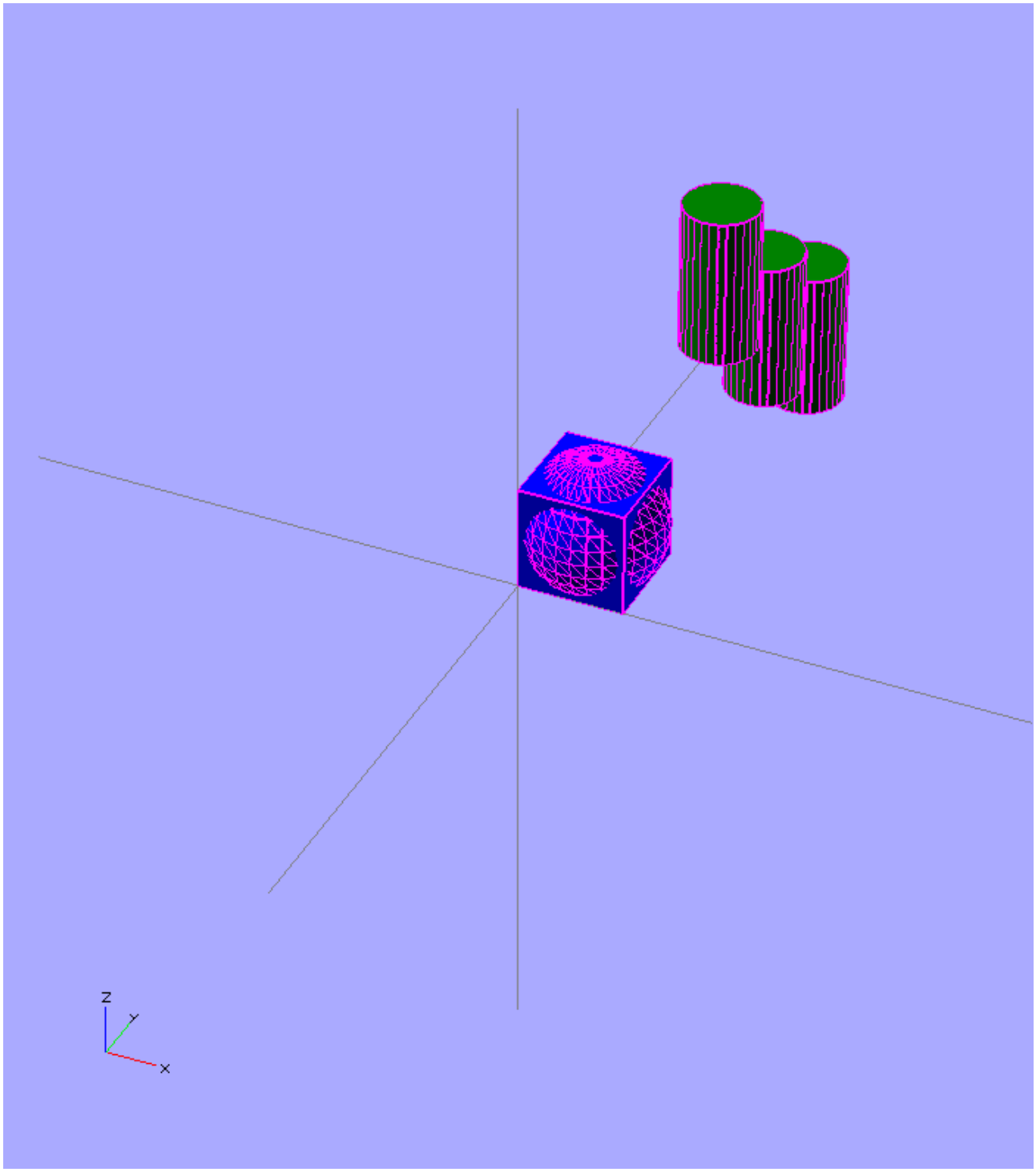
Ctrl+p
位置复位中心，不管你把模型拖动到那个角落位置，View-> Center，模型就会以坐标 XYZ 的 0 点为中心，视图。

举例，我特意吧位置安排成 2 个距离比较远的模型，然后用不同的角度命令去查看，看看效果了！。

Example:

```
translate([15,15,15])
color("blue")
difference() {
cube(30, center=true);
sphere(20);
}
translate ([55,60,10])
color("green")
translate([0, 0, 30]) {
cylinder(h=40, r=10);
}
translate ([55,30,30])
color("green")
translate([0, 0, 30]) {
cylinder(h=40, r=10);
}

translate ([55,0, 60])
color("green")
translate([0, 0, 30]) {
cylinder(h=40, r=10);
}
```

Picture: gif, 2sec per picture, from Diagonal, Top, down, left, right, front, back view:



中心选项将在屏幕中间的把坐标中心（但不是旋转观看）。
默认情况下，视图在透视模式下，意味着 DID 距离远离观众希望看起来更短，因为它是常见的眼睛或相机。当视图模式改变为正交可见距离将不依赖于相机间的距离（视图将模拟一个摄像头无限距离无限焦距）。尤其是，这是很有用的结合顶，等上面介绍的选项，因为这将导致什么人会看到一个类似的 2D 图像工程图纸。

第一部分：8~12

8 基础的立体模型
- 8.1 cube 方形
- 8.2 sphere 球形
- 8.3 cylinder 圆柱形
- 8.4 polyhedron 多面体

9 转变（类似librecad的编辑功能）
- 9.1 scale 比例
- 9.2 resize 改变大小
- 9.3 rotate 旋转
- 9.4 translate 调动（改变位置）
- 9.5 mirror 镜像
- 9.6 multmatrix 多点矩阵分布
- 9.7 color 颜色
- 9.8 minkowski 闵可夫斯基，
- 9.9 hull 去壳

10 CSG Modeling CSG模型
- 10.1 union 联合
- 10.2 difference 剪切
- 10.3 intersection 求交
- 10.4 render 穿透(渲染)

11 Modifier Characters 特征修整
- 11.1 Background Modifier 背景修整
- 11.2 Debug Modifier 调试修整
- 11.3 Root Modifier 根 修整
- 11.5 Disable Modifier 无效修整

12 Modules 模块

基础的立体模型

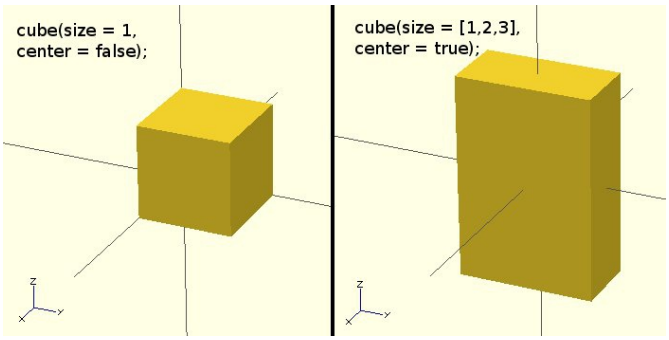
方体
创建一个方体在一个起点的坐标系统。当中心是真，方体将会在起始位置，其他的，将会建立在第一个8段空间。模型的参数名字是可选的如果模型是在相同的排序就像是配置参数一样。

参数
size:尺寸
十进制或者3 位数组值排列。如果一个单独的数字给出，结果将会是一个方体长度方向。如果一个3列数值给出，那么数值将相当于X，Y，Z方向的长度。默认的数值是1.
Center:中心

布尔代数。这个决定了对象模型的位置。如果是(true)是，模型是位于（0，0，0）。否则，模型将会放置到正数的4段空间以一个角度（0，0，0）的位置。默认的是（false)否。

使用实例：

```
cube(size = 1, center = false);
cube(size = [1,2,3], center = true);
```



球体

创建一个球体在一个坐标系统的原点。参数名称是可选择。

参数

r

十进制。这个是球体的半径。球体的分辨率基于尺寸，和$fa, $fs,$fn 变量。更多的信息关于特殊的变量查看：其他语言特征

$fa

角度表示角

$fs

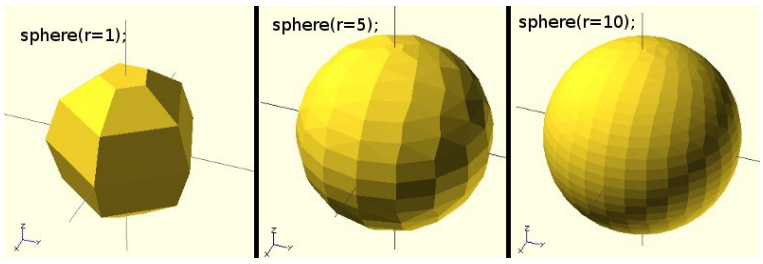
毫米表示角

$fn

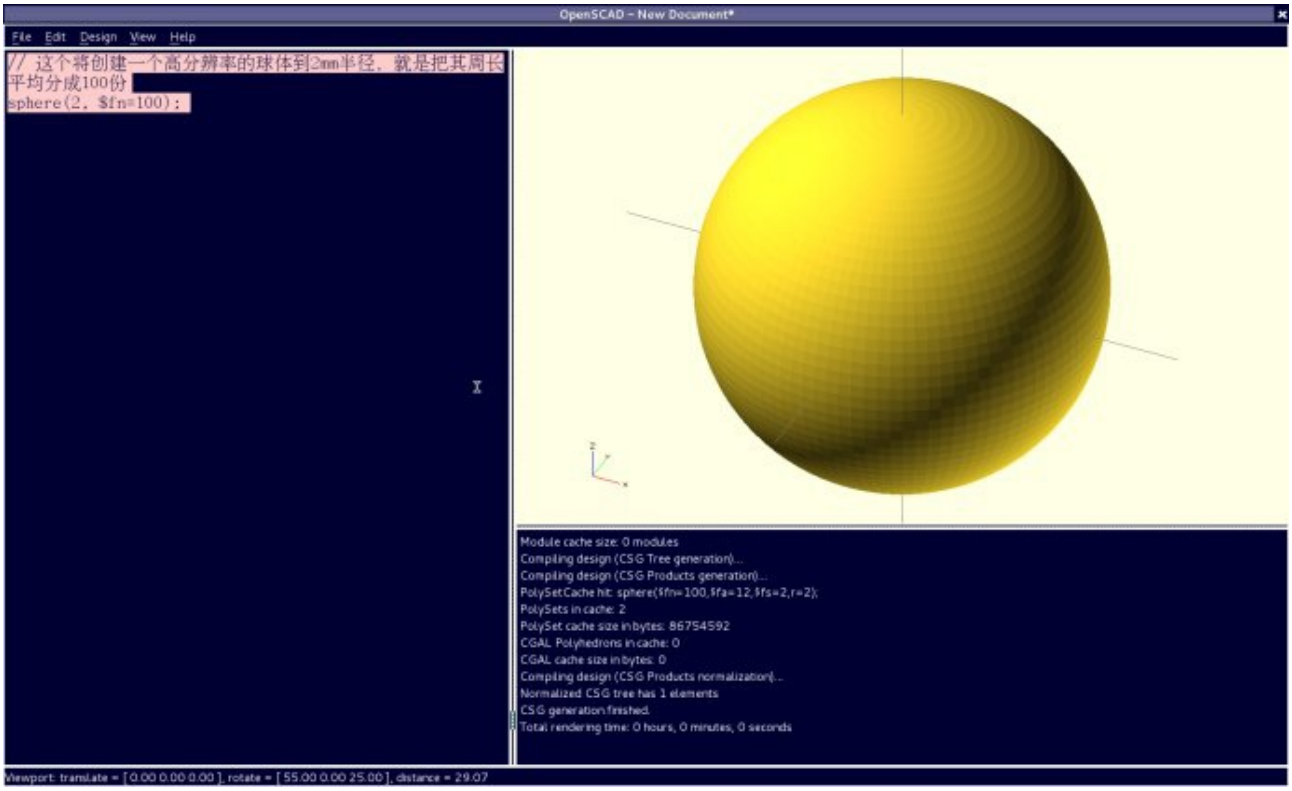
分辨率

使用实例：

```
sphere(r = 1);
sphere(r = 5);
sphere(r = 10);
```



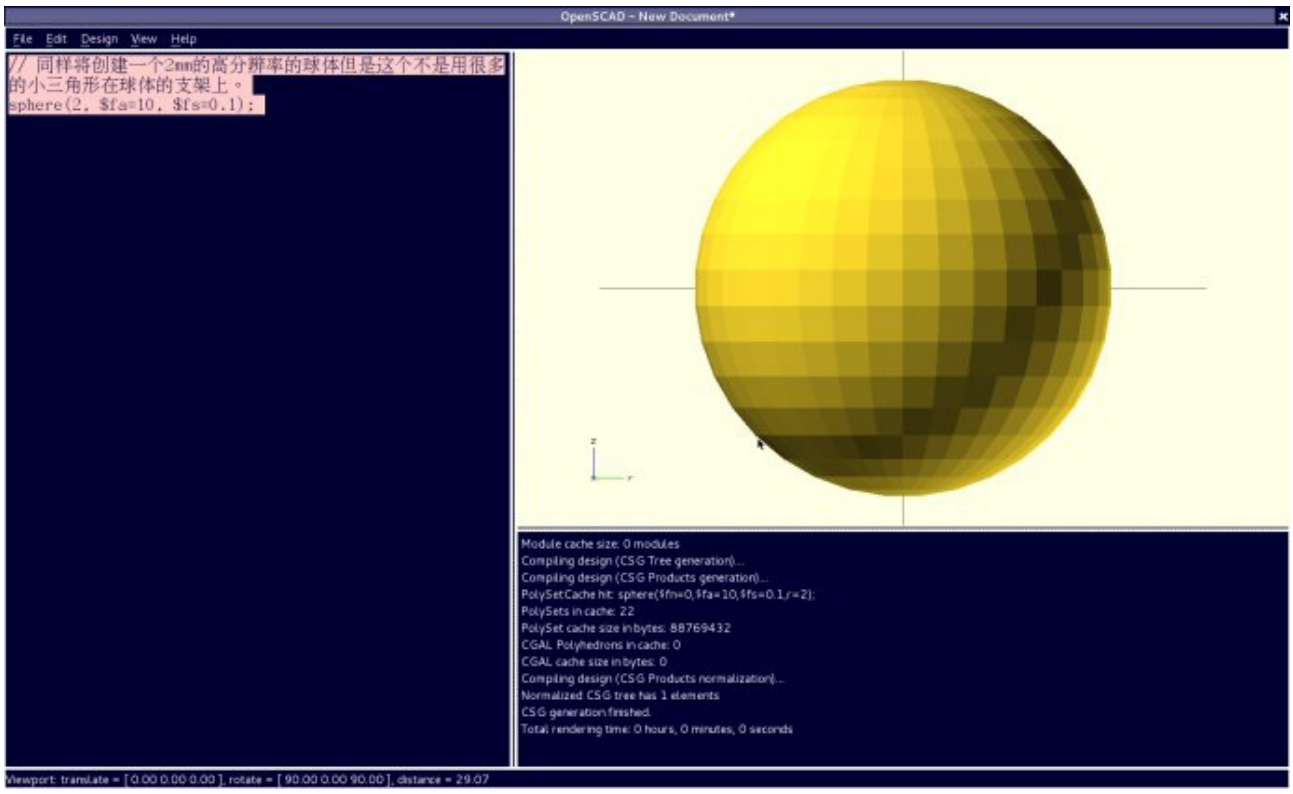
实例：
// 这个将创建一个高分辨率的球体到 2mm 半径，就是把其周长平均分成 100 份
sphere(2, $fn=100);



从图上的方格来看，好像是分了 50 份，不过也可以了，我试了把$fn1000，这时候计算机需要计算一会儿才能够得出来结果，这个 cad 软件和数学的程序好像关联很大阿，如果你用 c 语言计算过圆弧和角度的话，应该知道 openscad 的$fn 是怎么回事了。

特别注意：$fn 后边的数值越大，表示分辨率越高，计算机生成的速度越慢，反之亦然。$fn 是一个可以单独使用的，调整球体分辨率的函数。$fn 数值多少，就是把其周长平均分成多少份。

实例：
// 同样将创建一个 2mm 的高分辨率的球体但是这个不是用很多的小三角形在球体的支架上。
sphere(2, $fa=10, $fs=0.1);



特 别注意：$fa 和$fs 这两个函数是同时使用的。$fa 单独使用没有什么效果，而$fs 可以单独使用，表示周长除以数值，单位是毫米。如果两个函数结 合，$fa 可以表示圆周 360 度，多少度一段为分辨率，如果$fa 大于$fs 的数值，那么就显示大的，如果$fa 小于它，最低的效果就是$fs 的数值，就 是周长/$fs 的数值。

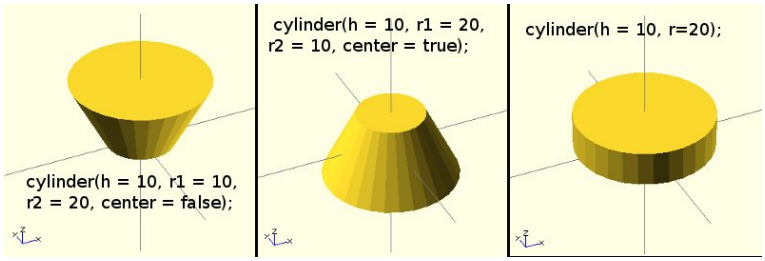
$fa 和$fs 的数值越小，圆周等分的段数越多，分辨率越高。

圆柱体
建立一个圆柱体或者锥体在坐标的原点。一个单半径(r)生成一个圆柱体，2 个不同的半径 (r1, r2)生成一个锥形。
参数
h
十进制。这个是圆柱体的高度。默认数值是 1.
r1
十进制。这个是锥体的底部的半径。默认的数值是 1.
r2
十进制。这个是锥体的顶部的半径。默认的数值是 1.
r
十进制。是圆柱体的底部和顶部一样的半径。使用这个参数如果你想建立一个圆柱体。默认数值是 1.

center :中心
布尔代数。如果 true (是) 圆柱体/锥体的中心高将围绕原点。默认的是 false (否) ,把圆锥体/圆柱体的(r1)底部放置在原点的位置.
$fa
度数表示角
$fs
毫米表示角

$fn
分辨率
使用实例：

```
cylinder(h = 10, r1 = 10, r2 = 20, center = false);
cylinder(h = 10, r1 = 20, r2 = 10, center = true);
cylinder(h = 10, r=20);
cylinder(h = 10, r=20, $fs=6);
```



polyhedron 多面体
创建一个多面体基于一系列的点和一系列的角度。点列表是所有的形状的顶点，角度列表是如何让点关联到多面体的表面。

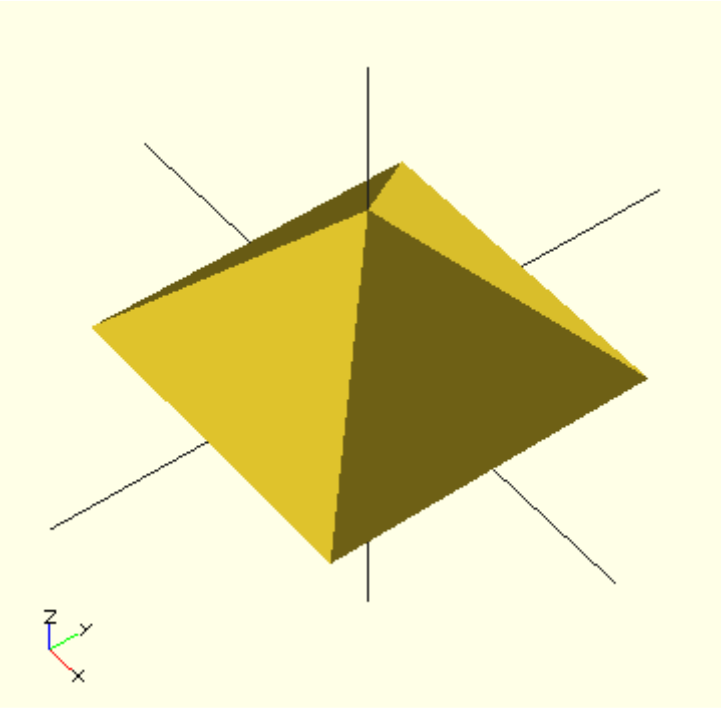
Parameters 参数
points 点
数组的点或者顶点（3 位数组）。
triangles 角度
数组三坐标点（3 位数组，XYZ）。每一个数字是 0 为基数的点源自数组点。
convexity 凸面
正整数。凸面的参数配置最大数值在前端（后端）一个射线交叉的物体可能穿透。这个参数是仅仅需要正确的显示对象模型在 openCSG 模式，而多面体没有渲染效果。
Syntax example 句法实例：

```
polyhedron(points = [ [x, y, z], ... ], triangles = [ [p1, p2, p3..], ... ], convexity = N);
```

三 角点排序：当看到外边向内的面，这个点必须是顺时针。你可以从新安排次序关于点或者次序参照每一个三角的其他三位数坐标组合。三角的排序是无形的。注意你 的多面体不是左右方向都是一样的，OpenSCAD 将会同时显示一个错误或者彻底崩溃，所以特别注意顶点的排序。还有，记住那个'pN' 组件冠以三角数组 0 基数参考的点数组组件。

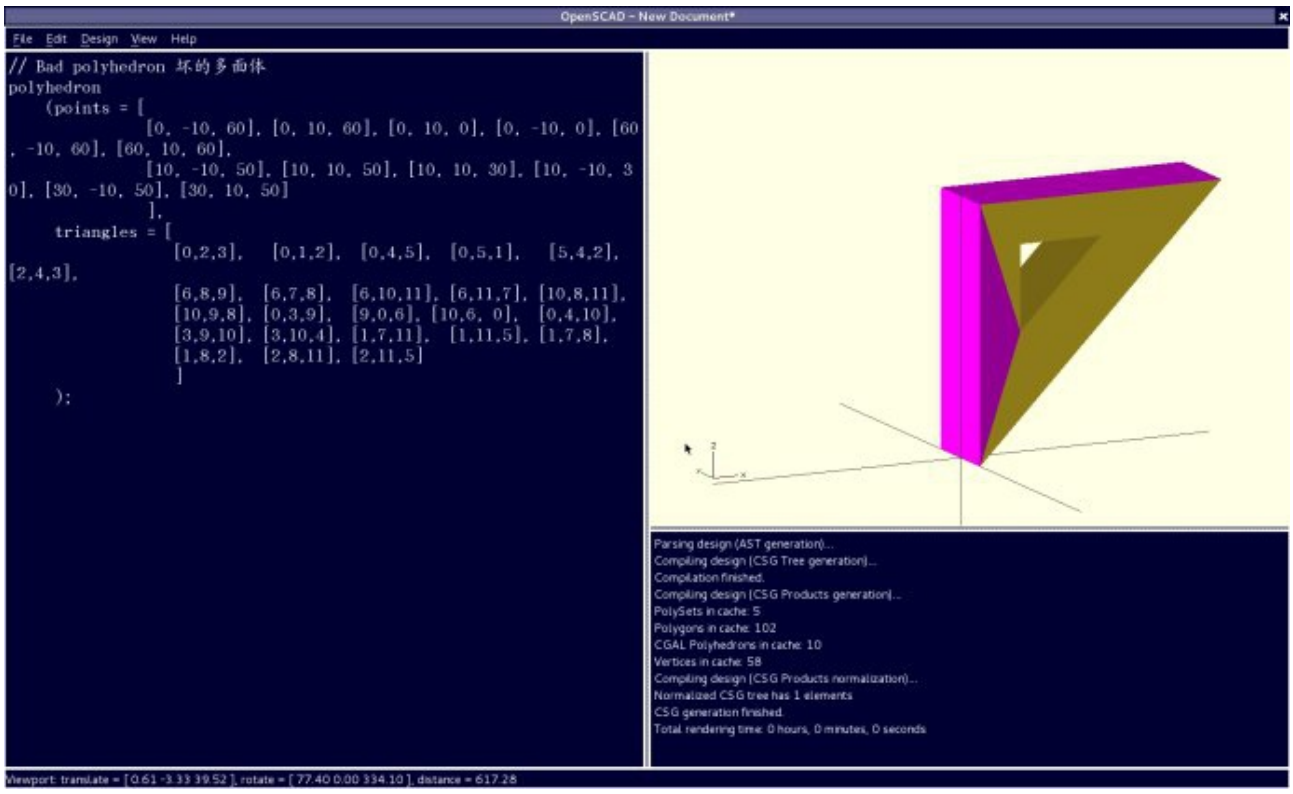
Example, a square base pyramid
实例，一个正方体基的金字塔：

```
polyhedron(
  points=[ [10,10,0],[10,-10,0],[-10,-10,0],[-10,10,0], // the four points at base
           [0,0,10]  ],                                  // the apex point
  triangles=[ [0,1,4],[1,2,4],[2,3,4],[3,0,4],          // each triangle side
              [1,0,3],[2,1,3] ]                         // two triangles for square base
 );
```



图形：一个简单的多面体，正方体基的金字塔。
三角的点的排序： 一个实例是一个非常复杂的多面体，展示如何修复多面体的不好的方位多边形。
当 你选择：Throw together'模式观看源自 view 菜单，compile 设计 (不是编译和渲染 compile and render)你将会看到一个正面的，没有方向的多边形突出显示。不幸的是这个突出显示是不可以在 OpenCSG 预览模式，因为他干涉 OpenCSG 预览 模式的应用。
下方你可以看见代码和图片关于就像一个有问题的多面体，一个问题的多边形（三角形或者组合的三角形）会呈现粉色。
（首先，只有 F5 键可以编译，仅仅编译，compile， 其他的 F6,F7,F8,F9,F10,F11,F12 都不可以显示；只有在 Throw together 模式才可以看到，或者在菜单栏选择视图模式，View->Throw together,)

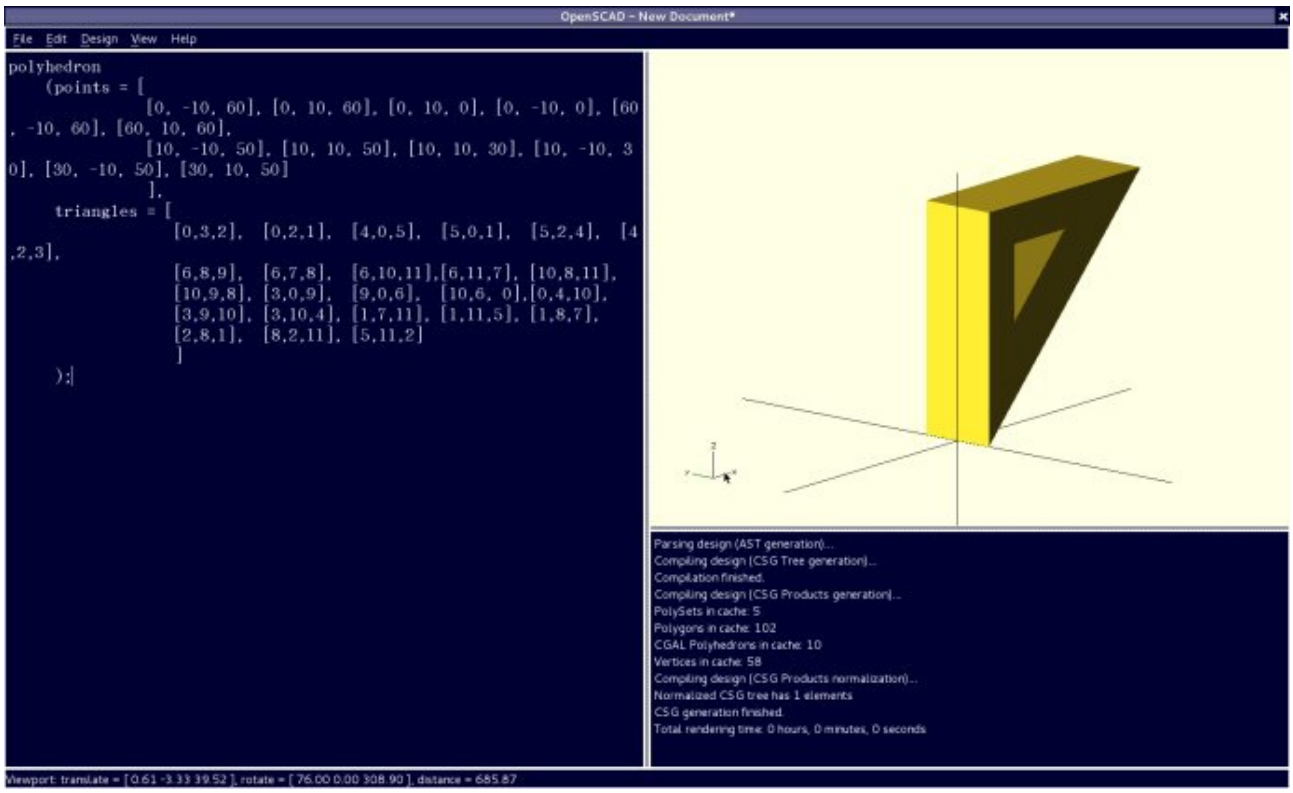
```
// Bad polyhedron 坏的多面体
polyhedron
    (points = [
                [0, -10, 60], [0, 10, 60], [0, 10, 0], [0, -10, 0], [60, -10,
60], [60, 10, 60],
                [10, -10, 50], [10, 10, 50], [10, 10, 30], [10, -10, 30], [30,
-10, 50], [30, 10, 50]
                ],
     triangles = [
                  [0,2,3],   [0,1,2],  [0,4,5],  [0,5,1],   [5,4,2],  [2,4,3],
                  [6,8,9],  [6,7,8],  [6,10,11], [6,11,7], [10,8,11],
                  [10,9,8], [0,3,9],  [9,0,6], [10,6, 0],  [0,4,10],
                  [3,9,10], [3,10,4], [1,7,11],  [1,11,5], [1,7,8],
                  [1,8,2],  [2,8,11], [2,11,5]
                  ]
     );
```



图片：多面体有坏的方向多边形

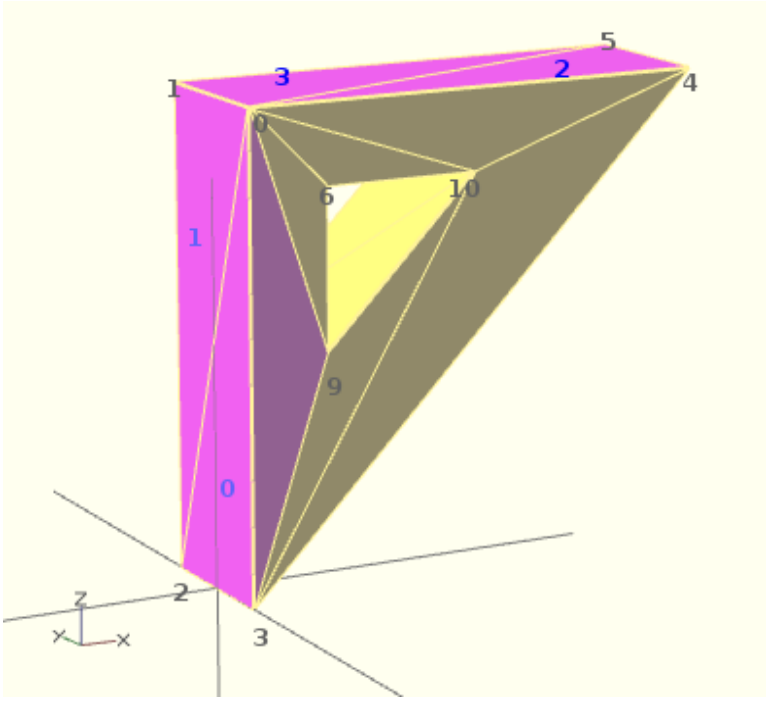
一个正确的多面体将会是以下实例代码：

```
polyhedron
    (points = [
                [0, -10, 60], [0, 10, 60], [0, 10, 0], [0, -10, 0], [60, -10, 60], [60, 10, 60],
                [10, -10, 50], [10, 10, 50], [10, 10, 30], [10, -10, 30], [30, -10, 50], [30, 10, 50]
                ],
     triangles = [
                  [0,3,2],  [0,2,1],  [4,0,5],  [5,0,1],  [5,2,4],  [4,2,3],
                  [6,8,9],  [6,7,8],  [6,10,11],[6,11,7], [10,8,11],
                  [10,9,8], [3,0,9],  [9,0,6],  [10,6, 0],[0,4,10],
                  [3,9,10], [3,10,4], [1,7,11], [1,11,5], [1,8,7],
                  [2,8,1],  [8,2,11], [5,11,2]
                  ]
     );
```



图片：一个正确的多面体。
初学者提示：
如 果你的确不明白“方向”，努力识别错误的方向粉色的三角形，然后改变书则的参考点一直到你获得正确的。举例说明，三角形（[0,4,5])是一个错误的， 我们修整为[4,0,5]。此外，你可能选择“Show Edge”（意思是显示边线条）从”View Menu”（视图菜单），显示一个界面抓取和数字都有点和三角形。
在我们的一个实例中，点是黑色的注释，三角形是蓝色的（我这里是黄色的）。转动对象模型，复制一个备份如果需要。这样你可以找到轨迹。

顺时针技术：
方 向是定义顺时针索引。这个的意思是如果你观看三角形（就像[4,0,5])源自外边，你会看到顺时针的路径围绕着面的中心。弯曲次序[4,0,5]是顺时 针的所以是好的。弯曲次序[0,4,5]是逆时针的所以是坏的。同样，所有其他的顺时针[4,0,5]工作：[5,4,0]&[0,5,4]同样 是好的。如果你使用顺时针基数，你将会总是得到你的外表面（OpenSCAD 的外表面，其他程序始终使用逆时针就像是外表面）。
想象它就像是左手定义：
如果你拿着一个三角形并且弯曲你的手指是同样的次序就像点一样，然后你拇指点是向外的。



多面体有一个坏的方向的多边形
成功的描述“多面体”
* 点的定义所有的点和数组在形状上。
* 三角形是一系列的三角形连接起来用点和数组。
每一个点，在点列表中，是定义了三位数组，x,y,z 位置配置。点在点列表中是自动给出定义符从 0 开始使用三角形列表（0，1，2，3，...etc）.
每一个三角形，在三角形列表中，是定义选择的 3 个点（使用点定义符）画出点列表。
例如 ，triangles=[[0,1,2]]定义一个三角形从第一个点（点是 0 参考点）到第二个点然后到第三个点。
当看到许多三角形的轮廓是，三角形必须列出 3 个点以顺时针的次序。

转换（此章节在 qcad 中是'编辑'菜单）
转换影响的是子节点（最基础的点位置坐标）就像是名字应用到转换他们用不同的方法，比如 moving/rotating,或者 scaling 节点。转换的级联效应是使用一系列的转换命令完成最终的子节点位置。级联效应是获得嵌套法的描述，例如：

```
transform()              e.g.  rotate([45,45,45])
  transform()                      translate([10,20,30])
    child()                          cube(10);
```

转换可以是应用一个组的子节点使用'{' & '}'符号来围住子树（一个组的子组件），例如：

```
translate(0,0,-5)      or the more compact      translate(0,0,-5) {
{                                                 cube(10);
  cube(10);                                       cylinder(r=5,h=10);
  cylinder(r=5,h=10);                           }
}
```

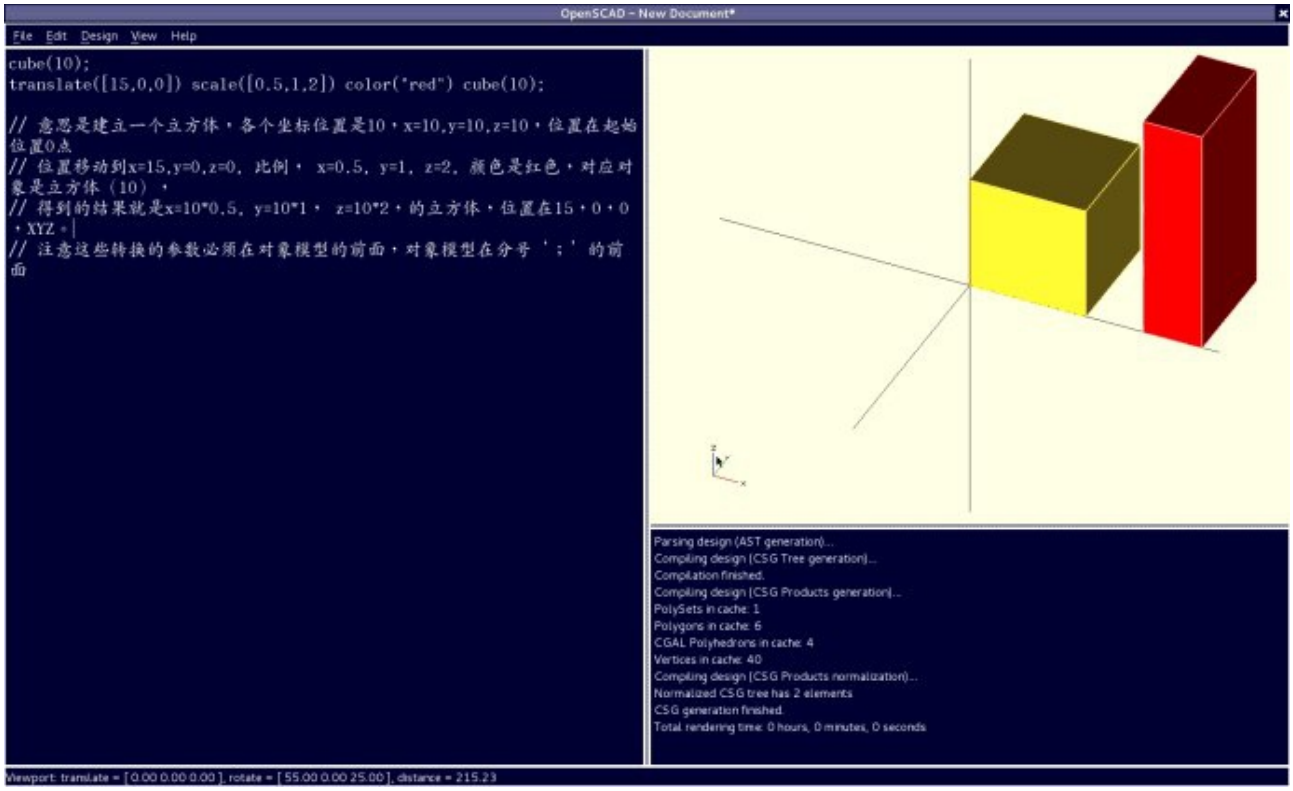
高级概念：
就 像是 OpenSCAD 使用不同的库用于兼容，这个可以介绍一些和转换不一致的 F5 预览的行为。传统的转换符（translate,rotate, scale, mirror & multimatrix）是履行使用 OpenGL 的预览，而其他更多的高级转换符，比如 resize,履行的是 CGAL 操作，CSG 操作的表现就像影响在 下面的对象模型，

而不是仅仅转换它。尤其是可以影响显示识别符的字符，具体的是'#' 和'%' ，那里的顶点也许不是直接显示的，就像影响的预先-改变尺寸 (pre-sizeed)的对象模型，但是影响的后-比例(post-scale)对象模型。

注意：文字的现行版本模式是不完善的。

scale (比例)
比例它的子组件使用配置的数组。这个参数名称是可选的。
Usage Example:
使用实例：

```
cube(10);
translate([15,0,0]) scale([0.5,1,2]) color("red") cube(10);

// 意思是建立一个立方体，各个坐标位置是 10，x=10,y=10,z=10，位置在起始位置 0 点
// 位置移动到 x=15,y=0,z=0， 比例， x=0.5， y=1， z=2， 颜色是红色，对应对象是立方体（10），
// 得到的结果就是 x=10*0.5， y=10*1， z=10*2，的立方体，位置在 15，0，0，XYZ。
// 注意这些转换的参数必须在对象模型的前面，对象模型在分号';'的前面
```

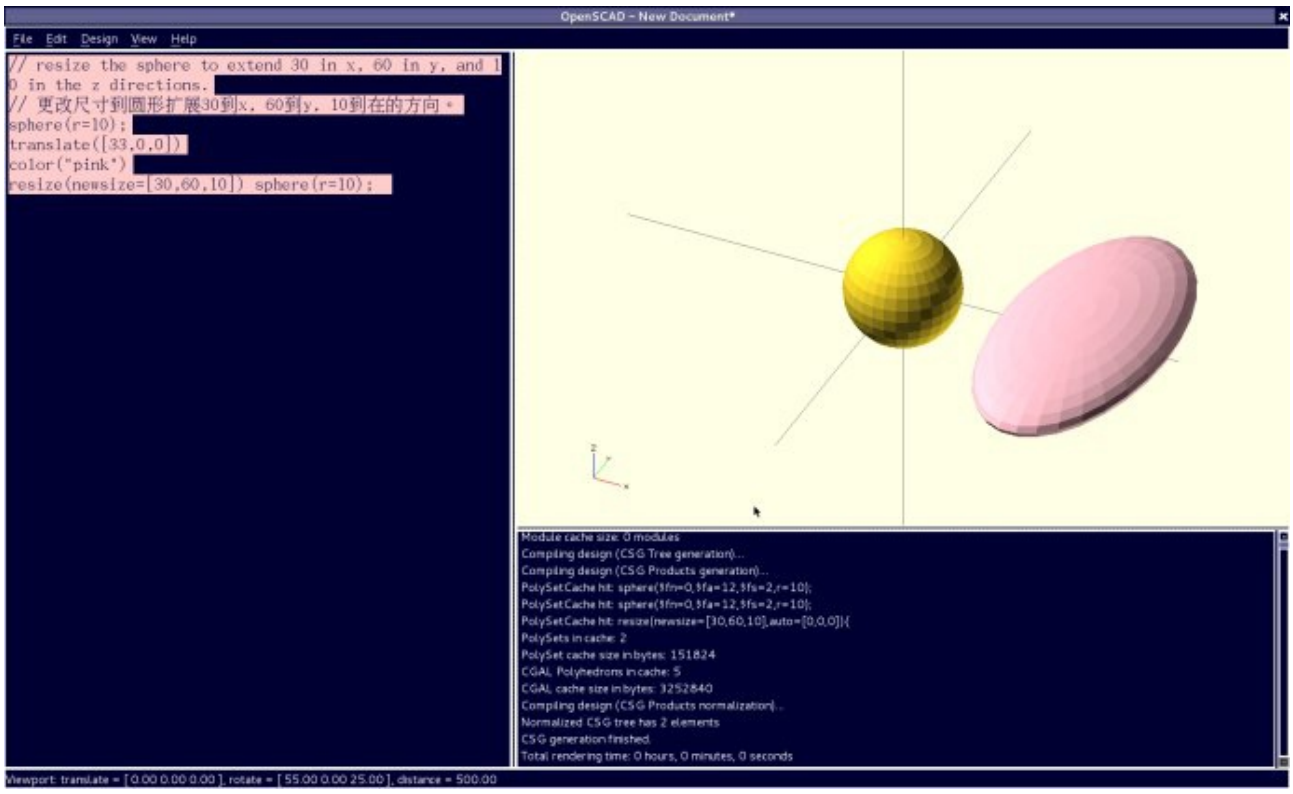


resize (尺寸) (更改尺寸)
在 2013.05 之后的版本才能够支持 resize () 这个命令。它是设置子对象模型的尺寸到匹配的 x,y,z 坐标。

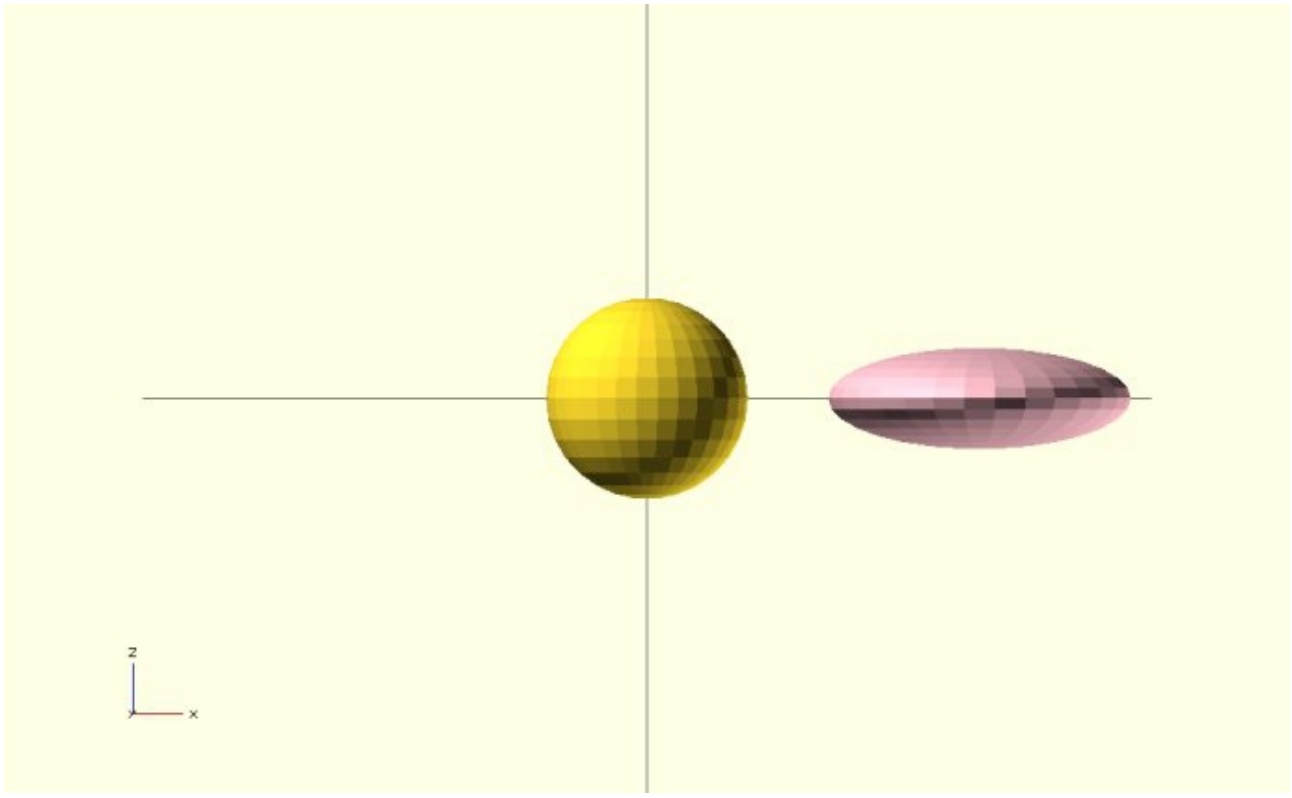
只有高于这个版本的软件，才能够支持文字颜色否则只能用数字来表示。

Usage Example:
使用实例：

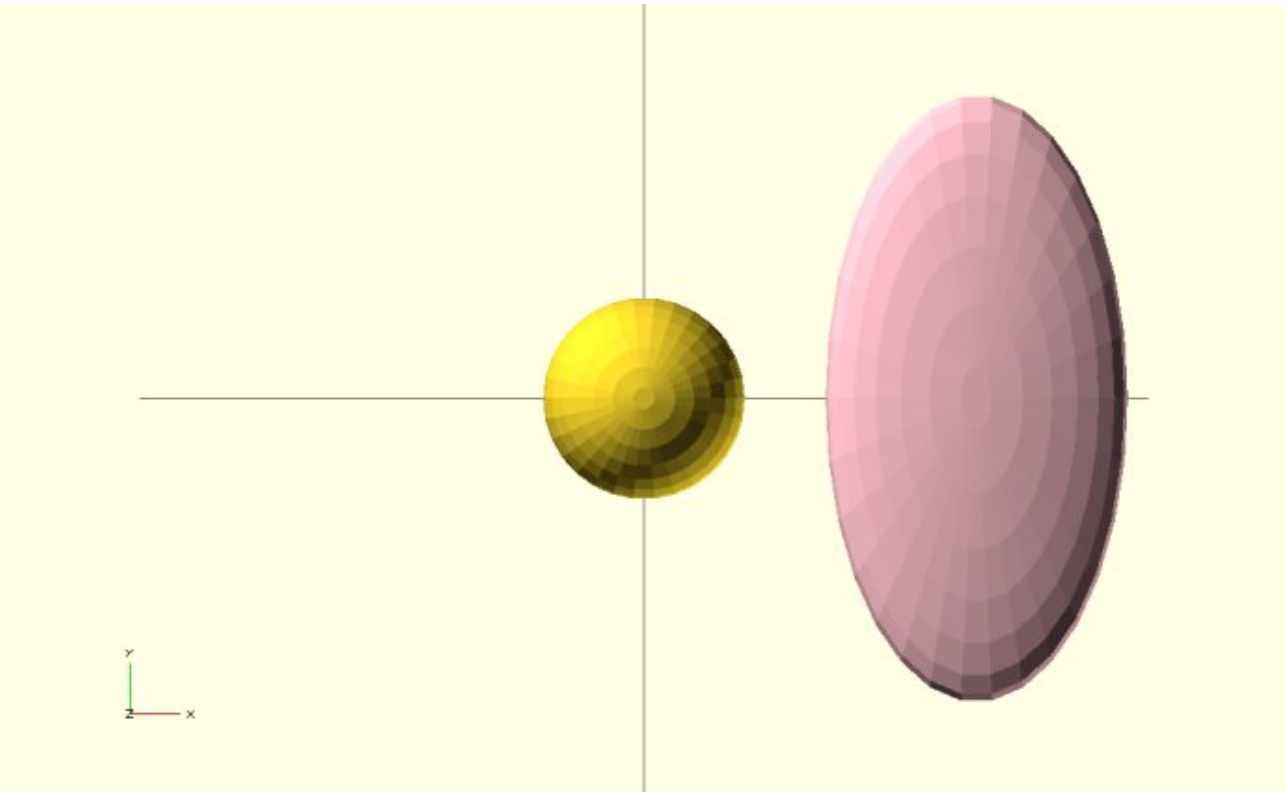
```
// resize the sphere to extend 30 in x, 60 in y, and 10 in the z directions.
// 球体(r=10),半径为10的球体，位置起始0点。
// 位置，33，0，0，颜色，粉色，更改尺寸到圆形扩展30到x，60到y，10到z(总高度）在的方向。

sphere(r=10);
translate([33,0,0])
color("pink")
resize(newsize=[30,60,10]) sphere(r=10);
```


OpenSCAD - New Document*
File Edit Design View Help
// resize the sphere to extend 30 in x, 60 in y, and 1
0 in the z directions.
// 更改尺寸到圆形扩展30到x，60到y，10到在的方向 *
sphere(r=10);
translate([33,0,0])
color("pink")
resize(newsize=[30,60,10]) sphere(r=10);
Module cache size: 0 modules
Compiling design (CSG Tree generation)...
Compiling design (CSG Products generation)...
PolySetCache hit: sphere($fn=0,$fa=12,$fs=2,r=10);
PolySetCache hit: sphere($fn=0,$fa=12,$fs=2,r=10);
PolySetCache hit: resize(newsize=[30,60,10],auto=[0,0,0]){
PolySets in cache: 2
PolySet cache size in bytes: 151824
CGAL Polyhedrons in cache: 5
CGAL cache size in bytes: 3252840
Compiling design (CSG Products normalization)...
Normalized CSG tree has 2 elements
CSG generation finished.
Total rendering time: 0 hours, 0 minutes, 0 seconds
Viewport: translate = [0.00 0.00 0.00], rotate = [55.00 0.00 25.00], distance = 500.00

version 1


z
x

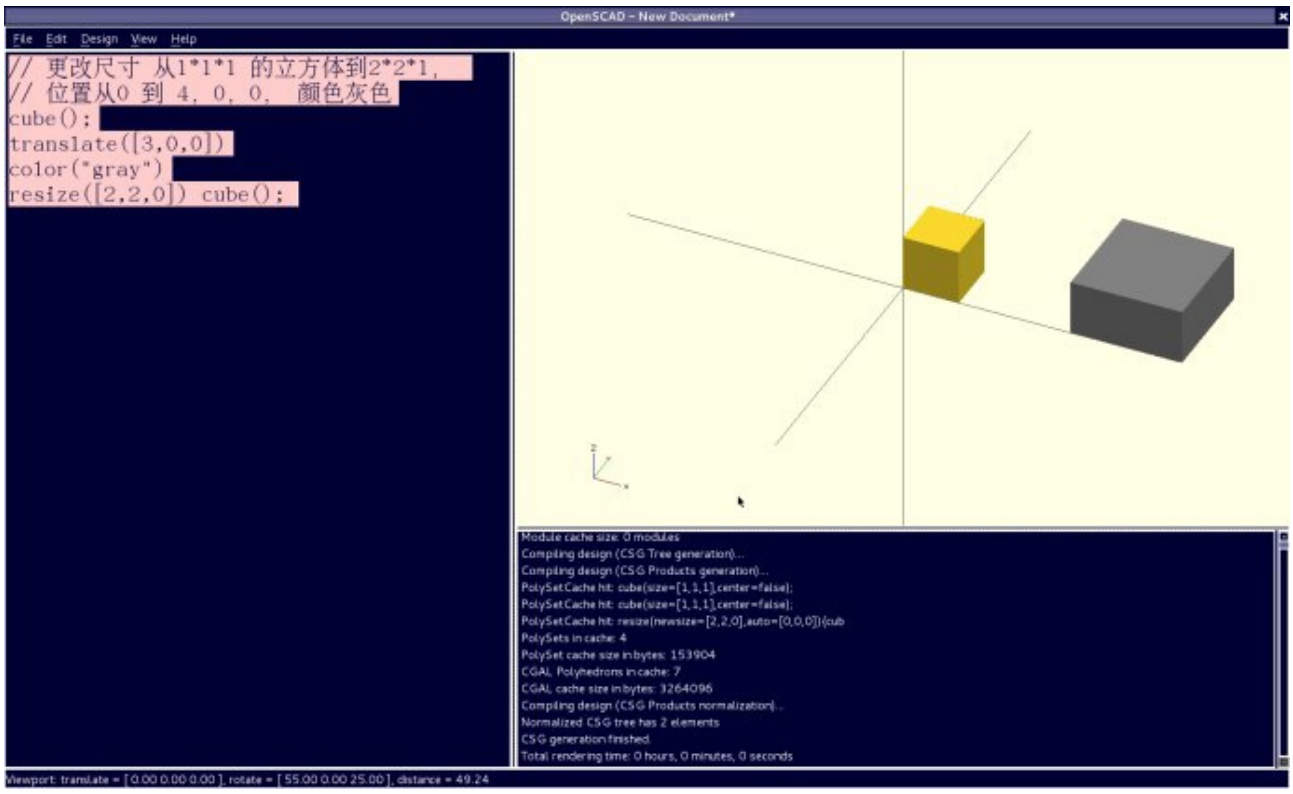
version 2



version 3

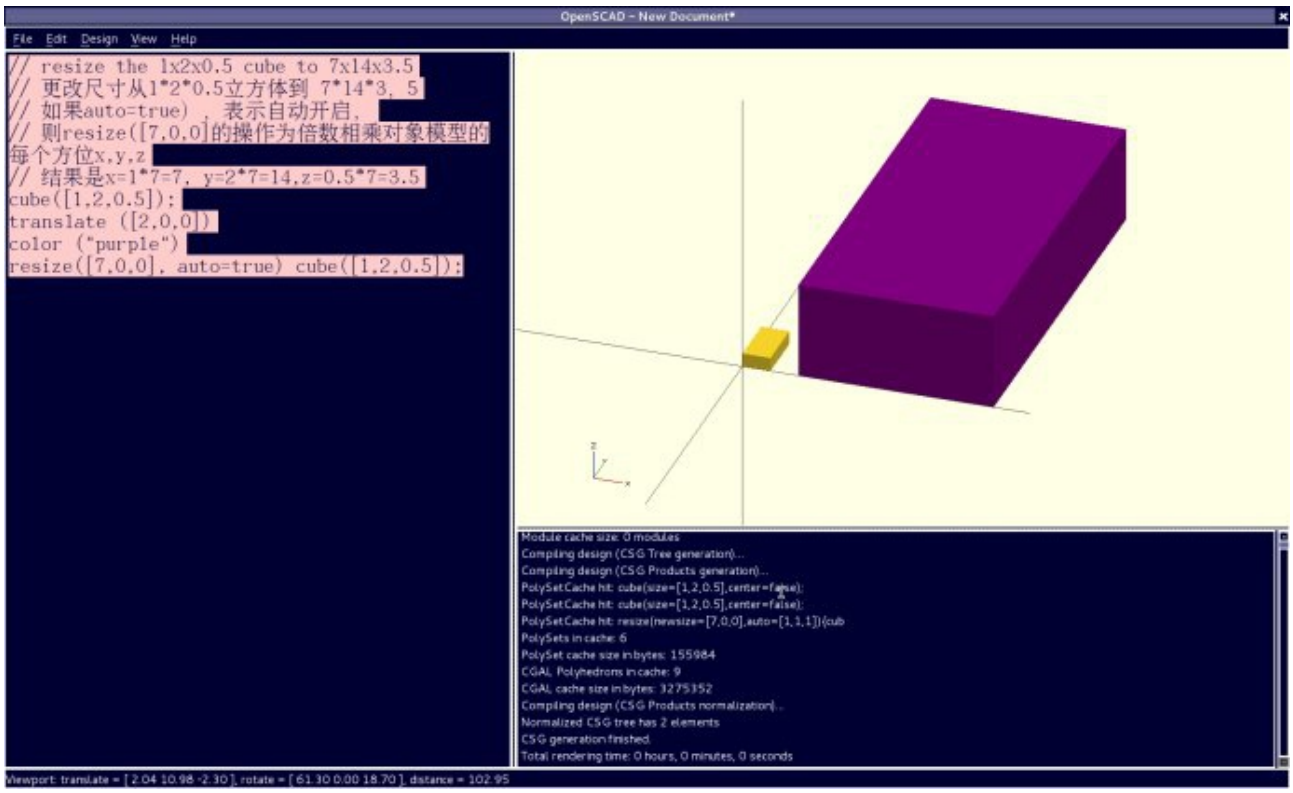
如果x,y,z是0，然后测量是左为准。

```
// 更改尺寸 从1*1*1 的立方体到2*2*1，
// 位置从0 到4，0，0， 颜色灰色
cube();
translate([3,0,0])
color("gray")
resize([2,2,0]) cube();
```



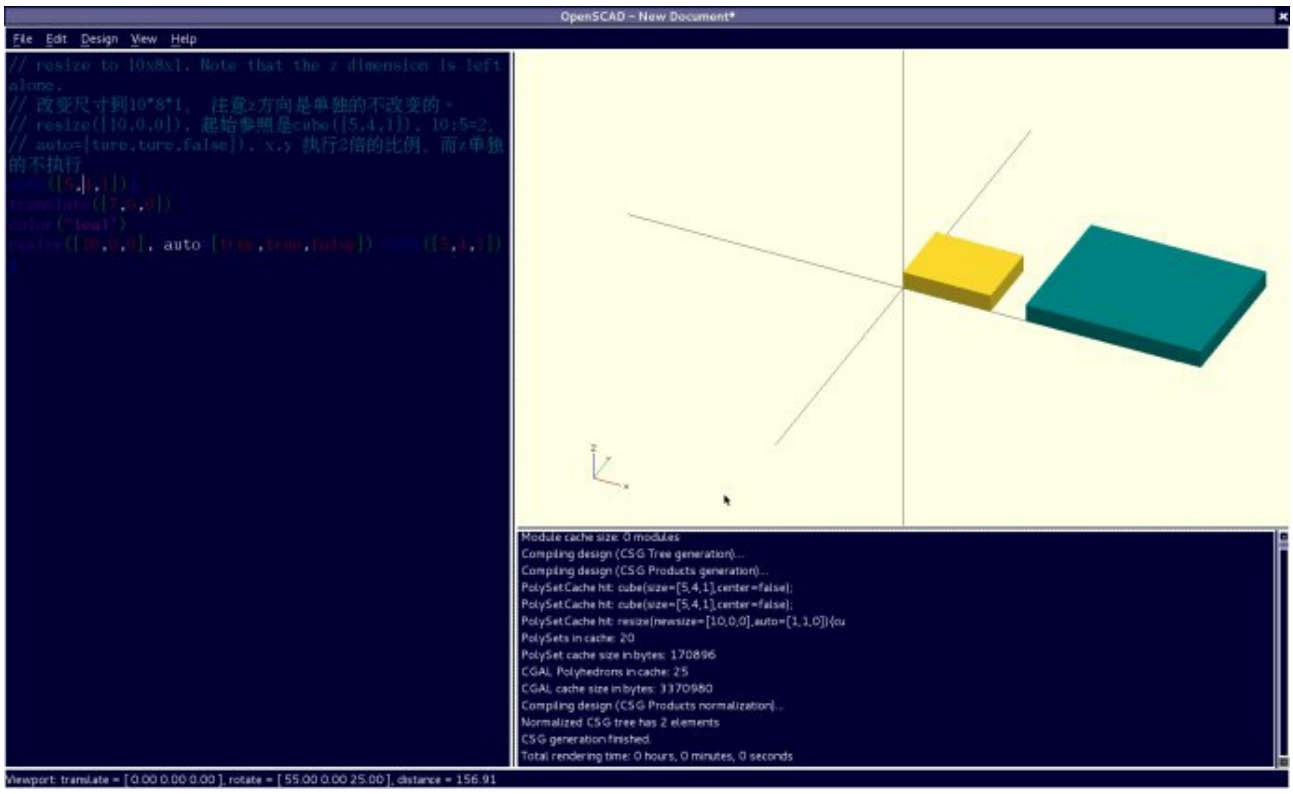
如果是'auto'自动的参数设置为“ture”是，它将会自动比例所有的0 坐标的匹配，比如：

```
// resize the 1x2x0.5 cube to 7x14x3.5
// 更改尺寸从1*2*0.5立方体到7*14*3，5
// 如果auto=true），表示自动开启，
// 则resize([7,0,0],7:1是7，x=1*7=7,y,z都参照x的比例，7倍。
// 结果是x=1*7=7, y=2*7=14,z=0.5*7=3.5
cube([1,2,0.5]);
translate ([2,0,0])
color ("purple")
resize([7,0,0], auto=true) cube([1,2,0.5]);
```



如果'auto”自动的参数可以同样是使用如果你仅仅希望自动比例(auto-scale)一个单个的尺寸测量，和离开其他的就像。例如：

```
// resize to 10x8x1. Note that the z dimension is left alone.
// 改变尺寸到10*8*1，注意z方向是单独的不改变的。
// resize([10,0,0]), 起始参照是cube([5,4,1]), 10:5=2，
// auto=[ture,ture,false]), x,y 执行2倍的比例，而z单独的不执行
cube([5,4,1]);
translate([7,0,0])
color("teal")
resize([10,0,0], auto=[true,true,false]) cube([5,4,1]);
```

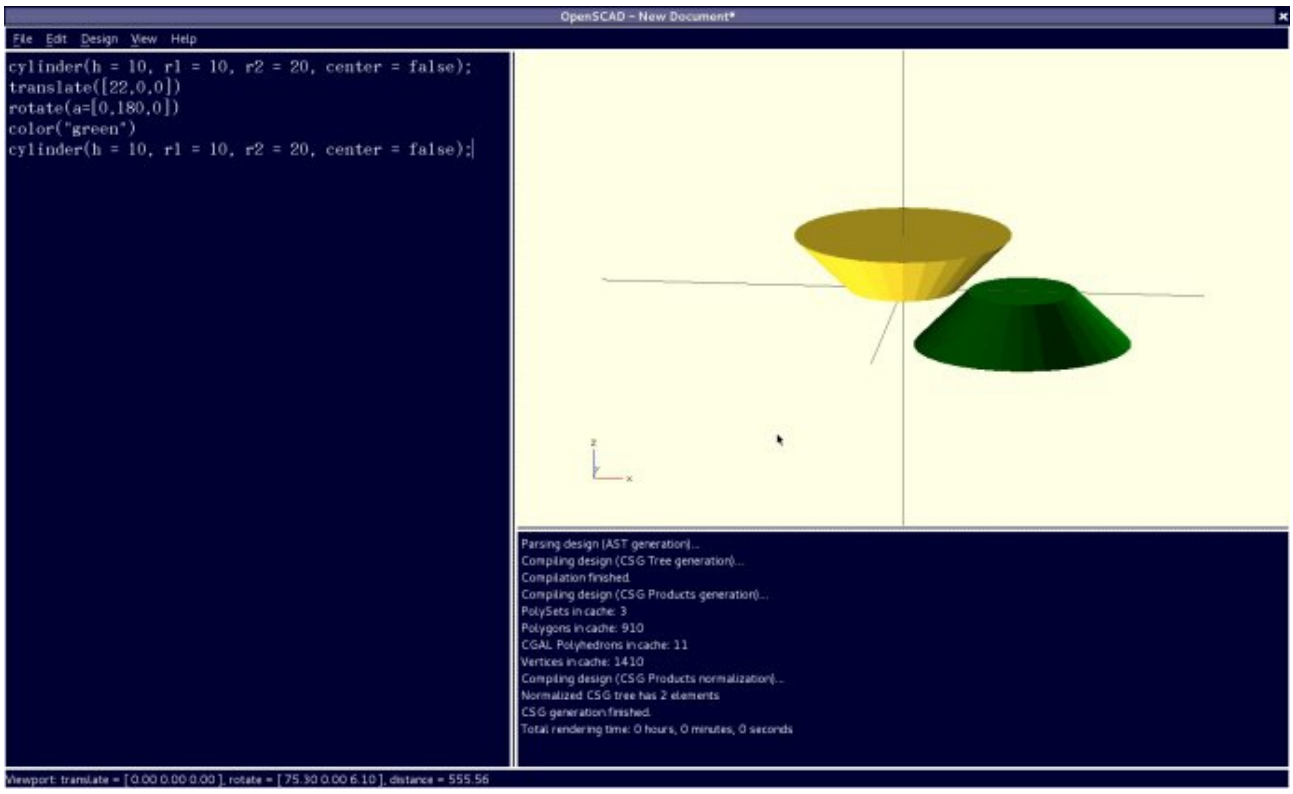


rotate (旋转)
旋转它的子对象模型a 角度从起始的坐标系统或者围绕着任意的轴线 (x,或者 y,或者 z)。这个参数的名字是可选择如果参数是一样的次序配置的。
当一个旋转是配置多个轴线的旋转应用到以下的次序:x,y,z.
Usage 使用句法:

```
rotate(a = deg, v = [x, y, z]) { ... }
For example, to flip an object upside-down, you might do this:
```

实例：反转一个对象模型使其顶端变成底部，你可以这样：

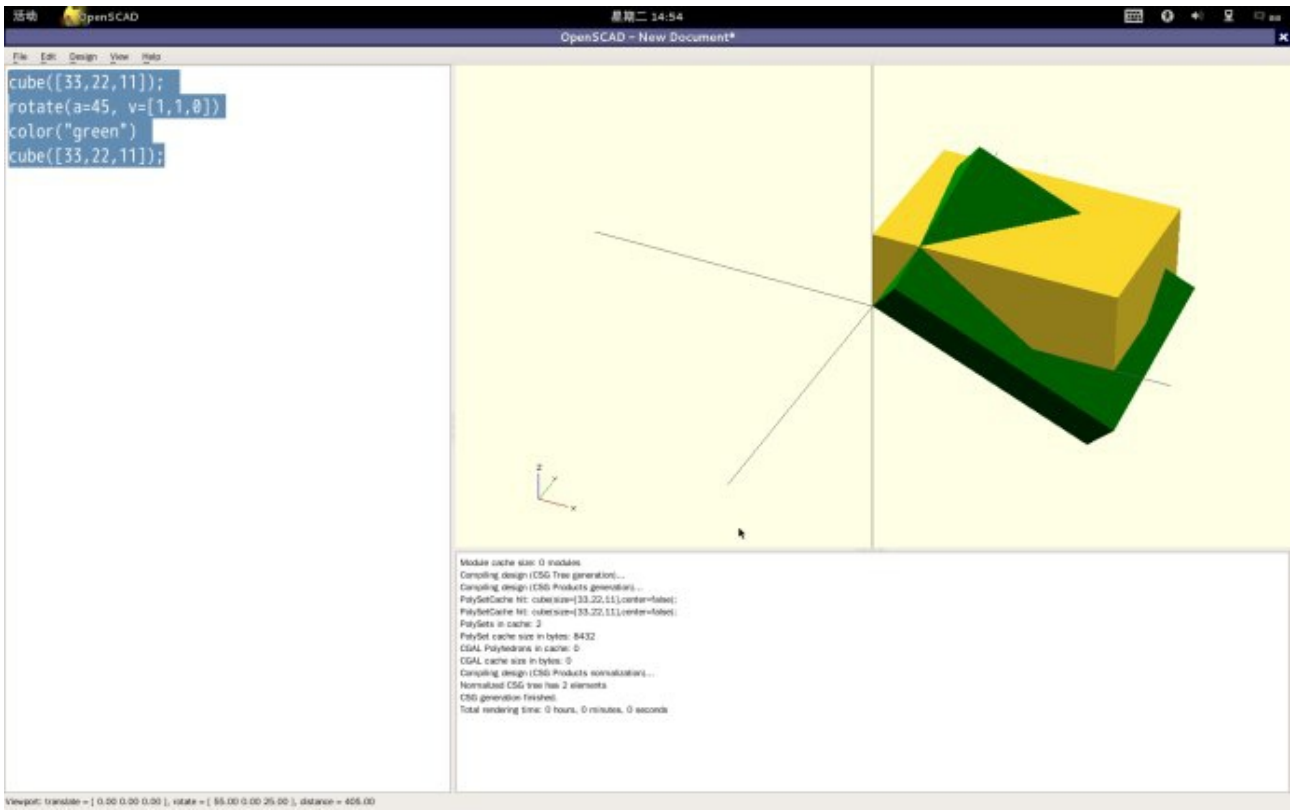
```
cylinder(h = 10, r1 = 10, r2 = 20, center = false);
translate([22,0,0])
rotate(a=[0,180,0])
color("green")
cylinder(h = 10, r1 = 10, r2 = 20, center = false);
```



实例关于将会旋转你的对象模型 180 度围绕着'y'轴线。

可选择的参数'v'允许你设置一个任意的轴到对象模型将会旋转。
实例用于任意轴线源旋转：

```
cube([33,22,11]);
rotate(a=45, v=[1,1,0])
color("green")
cube([33,22,11]);
```

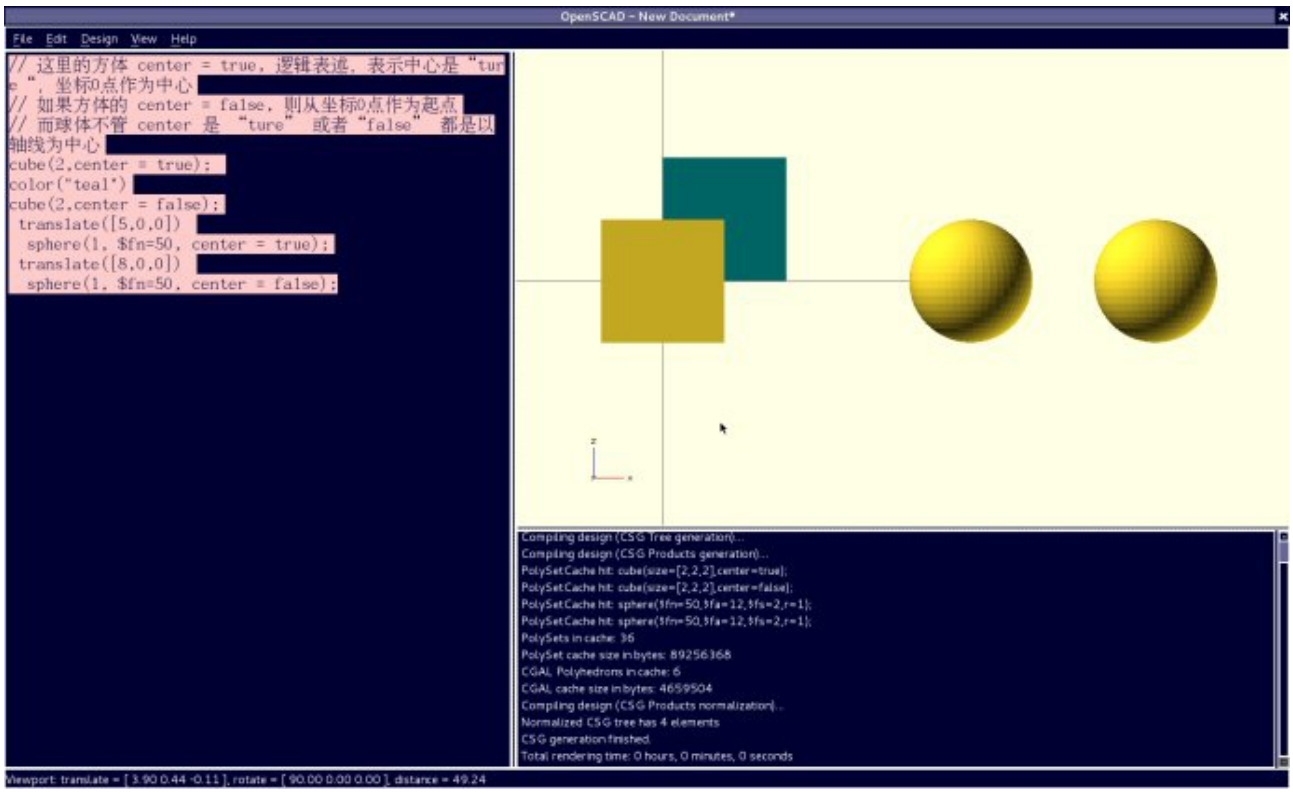


这个实例将会旋转你的对象模型 45 度围绕着自定义的轴线数组[1,1,0]，比如 45 度围绕着 X 和 45 度围绕着 Y 轴线。

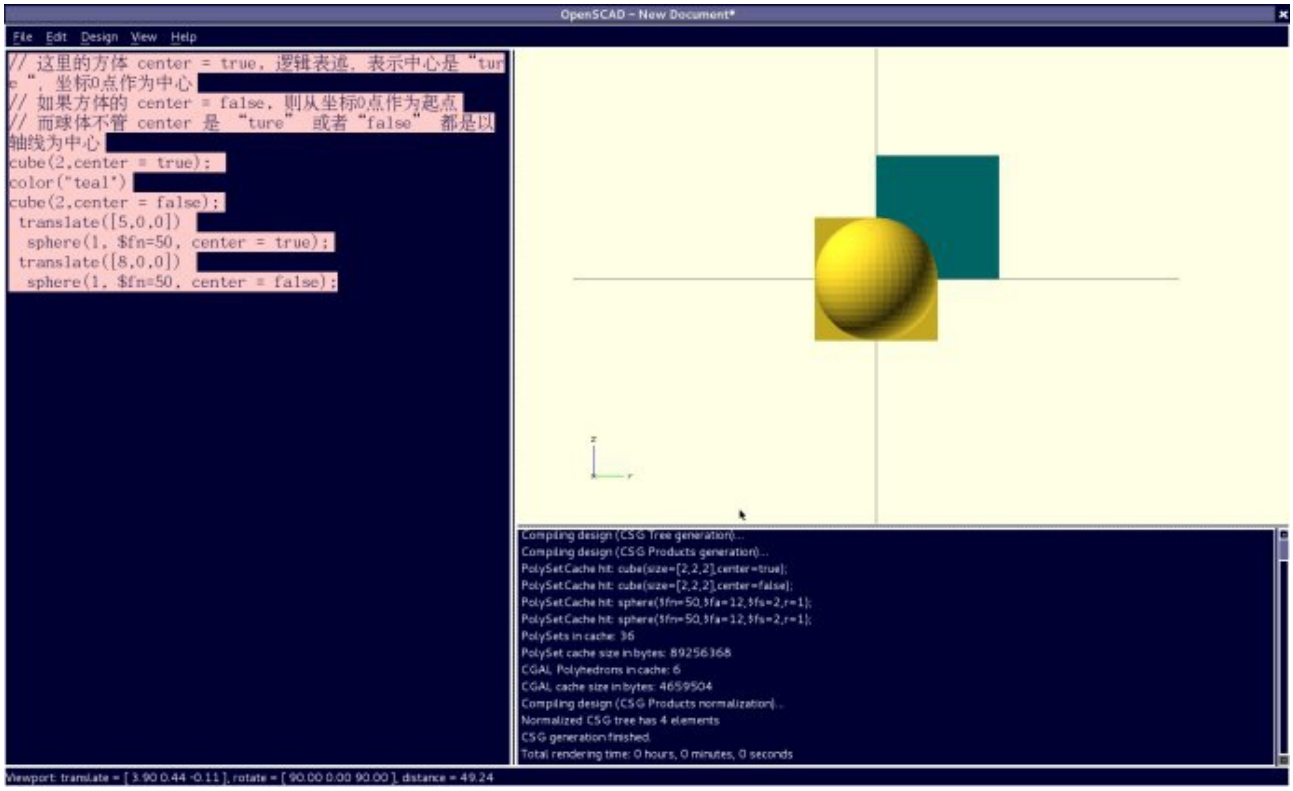
Translate 调动（这里指的是坐标位置的移动）
调动（移动）其子组件到配置的数组数值。参数名字是可选的。

句法描述：translate(v = [x, y, z]) { ... }
例如：

```
// 这里的方体 center = true，逻辑表述，表示中心是“ture “，坐标 0 点作为中心
// 如果方体的 center = false，则从坐标 0 点作为起点
// 而球体不管 center 是 “ture” 或者“false” 都是以轴线为中心
cube(2,center = true);
color("teal")
cube(2,center = false);
 translate([5,0,0])
  sphere(1, $fn=50, center = true);
 translate([8,0,0])
  sphere(1, $fn=50, center = false);
```



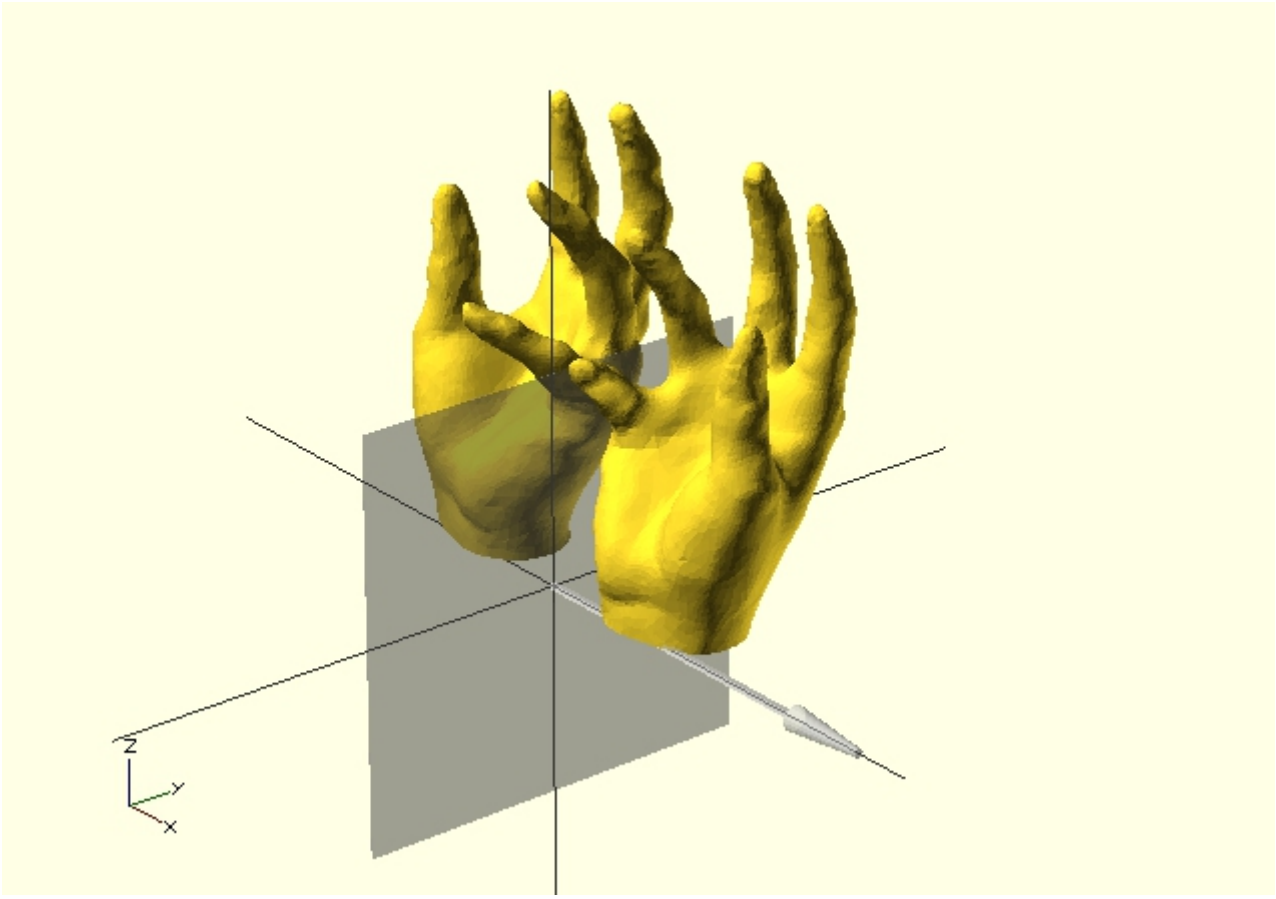
version 1



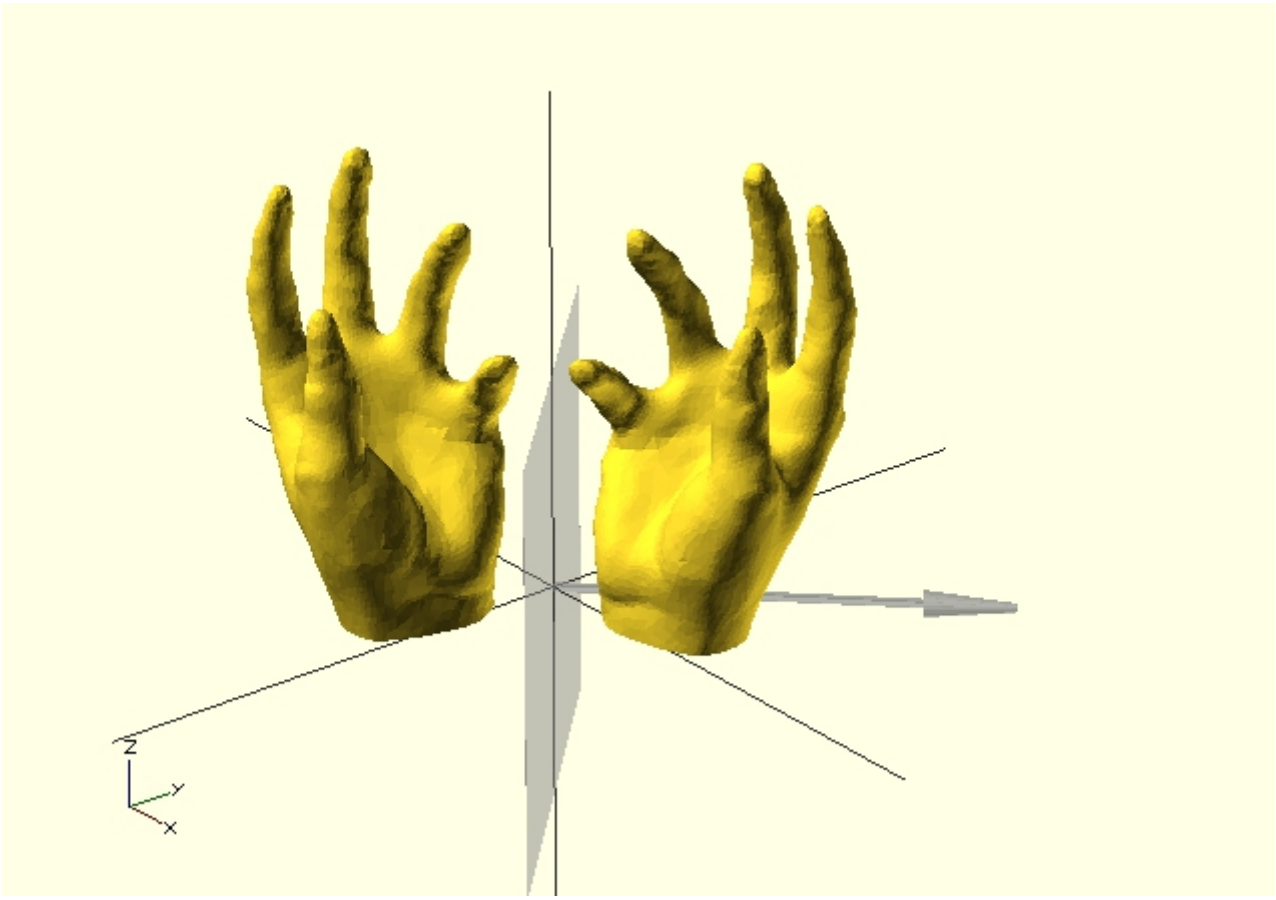
version 2

mirror 镜像
镜像子组件在一个平面上围绕起点。参数关于 mirror()是一个普通的数组对应一个平面交叉围绕着起点对对象模型进行镜像效果。
三
句法实例:

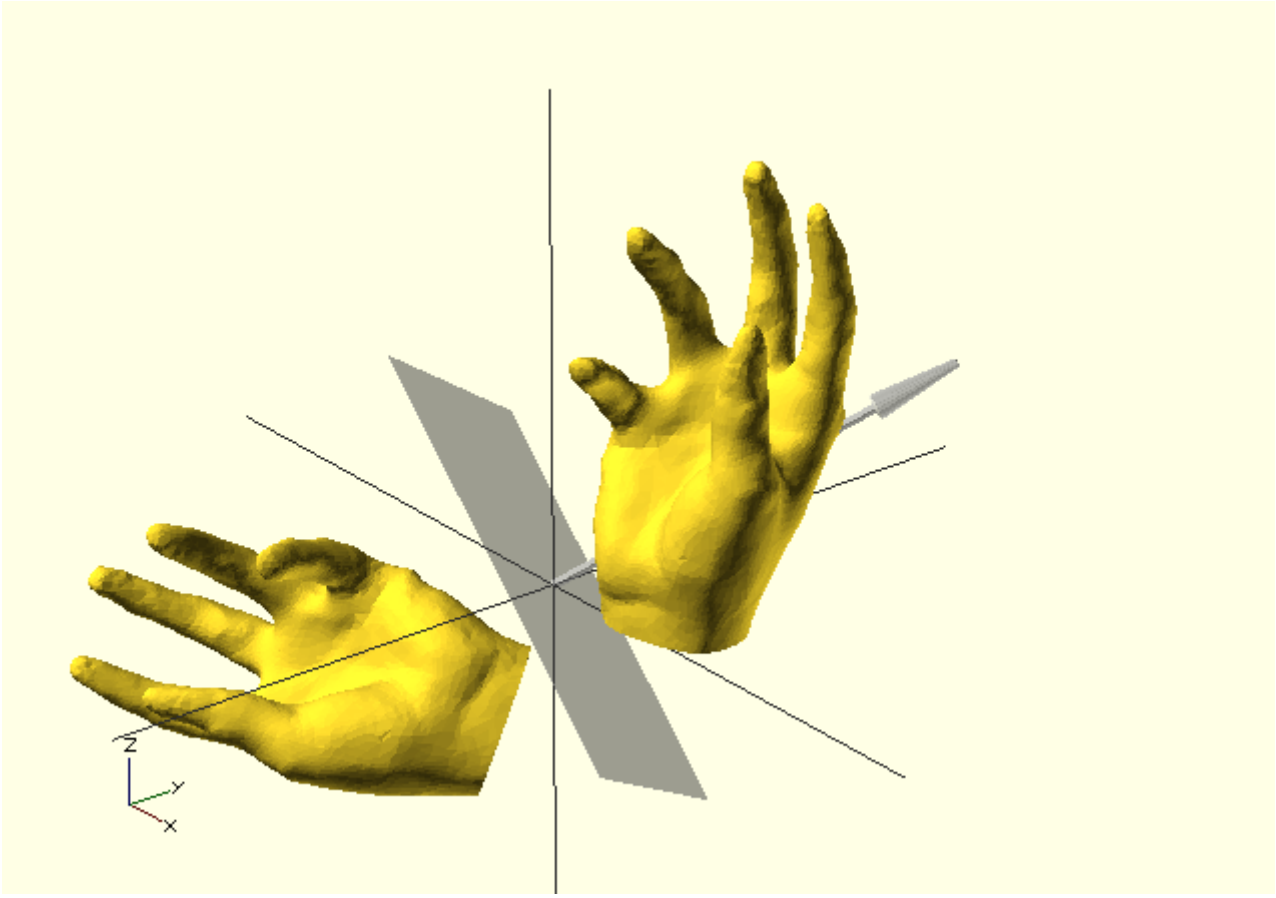
```
mirror([ 0, 1, 0 ]) { ... }
```




```
mirror([1,0,0]);
```




```
mirror([1,1,0]);
```

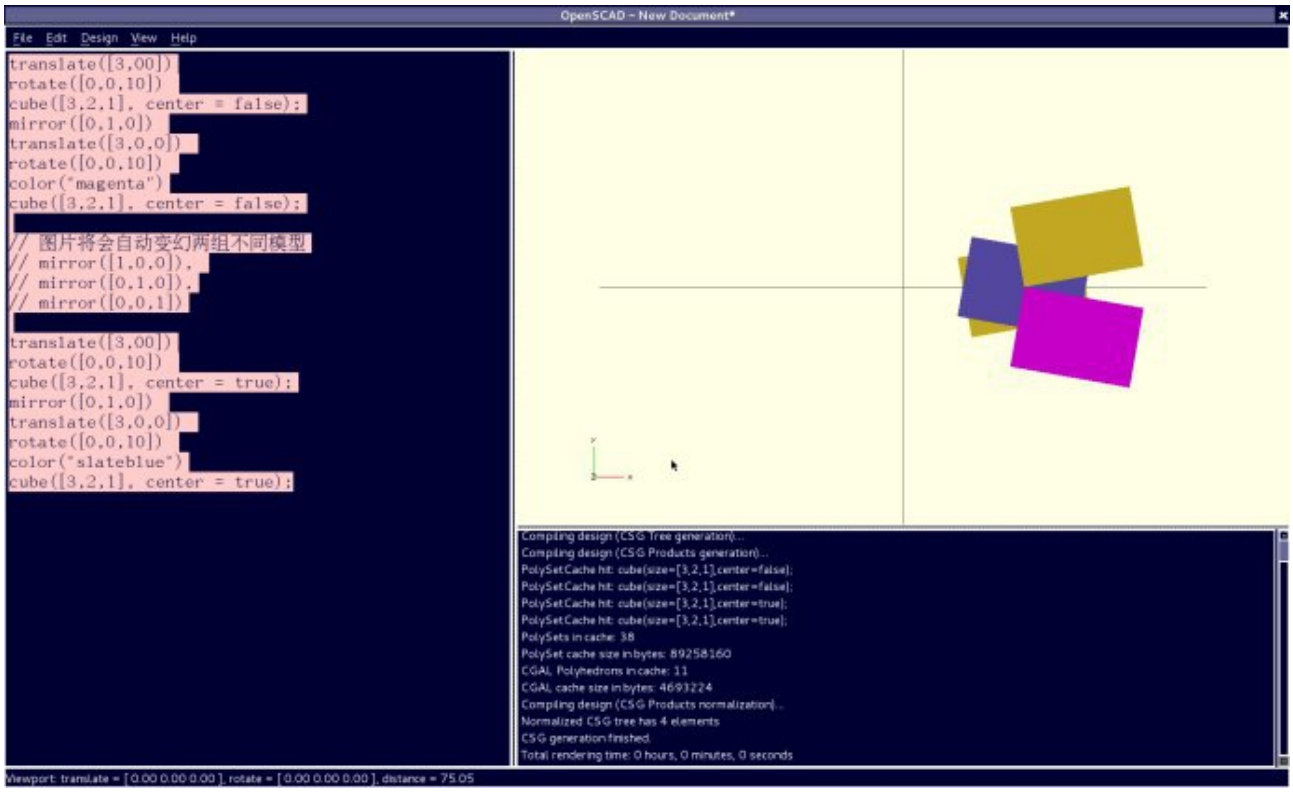



```
mirror([1,1,1]);
```

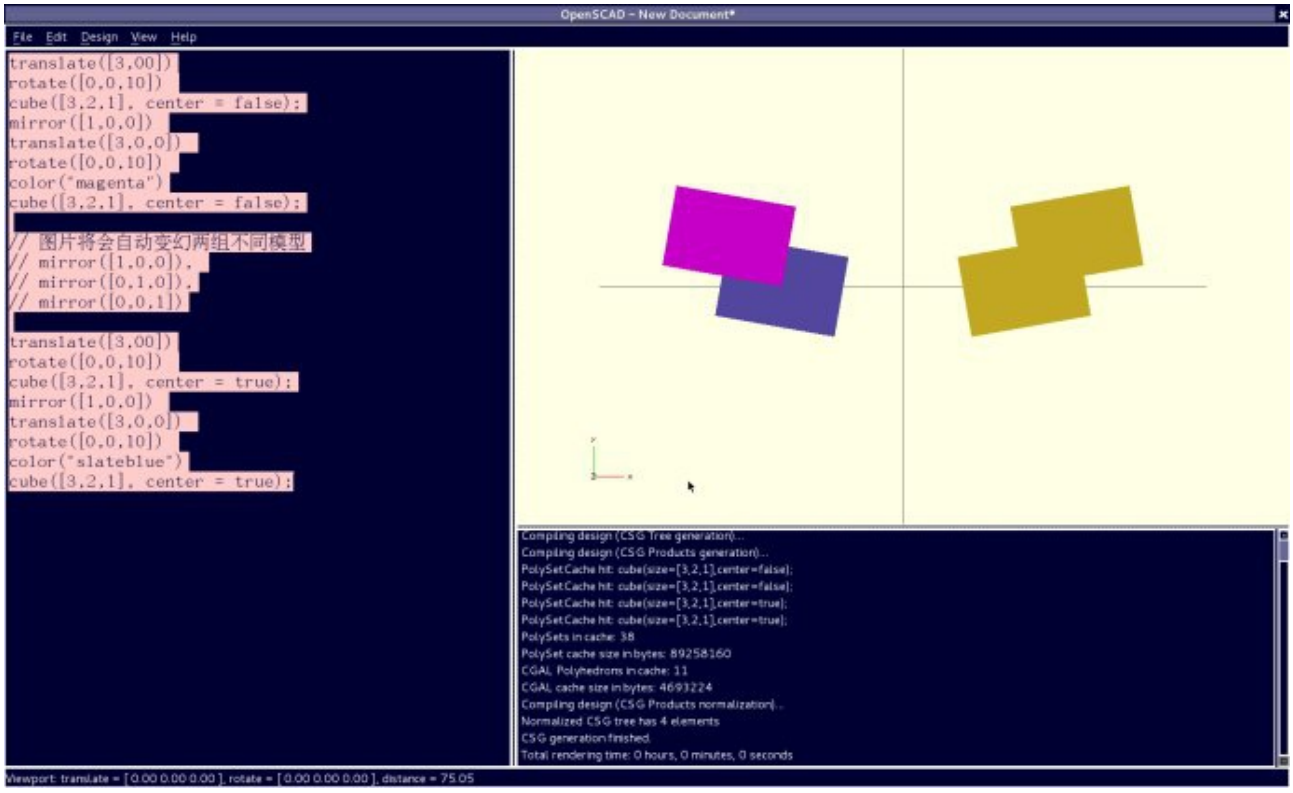
应用实例：

```
translate([3,00])
rotate([0,0,10])
cube([3,2,1], center = false);
mirror([0,0,1])
translate([3,0,0])
rotate([0,0,10])
color("magenta")
cube([3,2,1], center = false);

// 图片将会自动变幻两组不同模型
// mirror([1,0,0]),
// mirror([0,1,0]),
// mirror([0,0,1])

translate([3,00])
rotate([0,0,10])
cube([3,2,1], center = true);
mirror([0,0,1])
translate([3,0,0])
rotate([0,0,10])
```

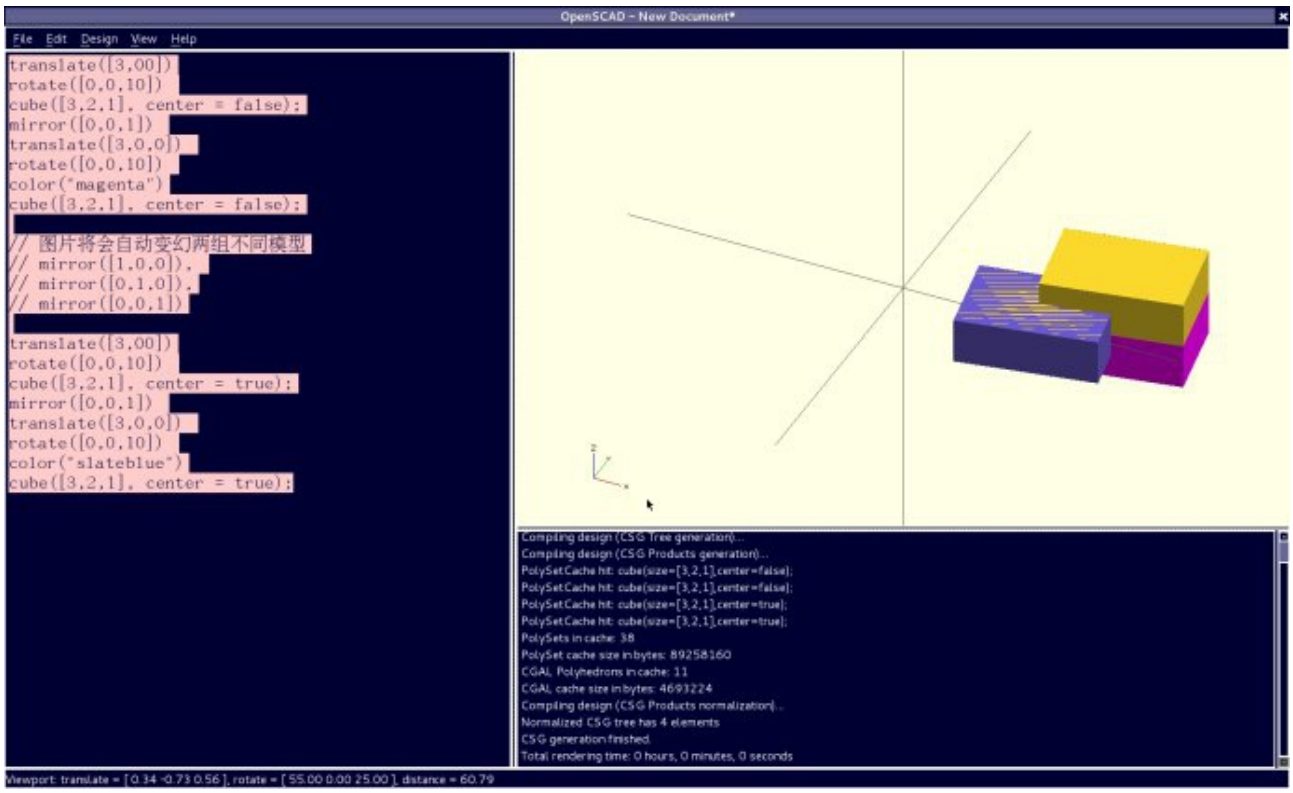
```
color("slateblue")
cube([3,2,1], center = true);
```



version 1



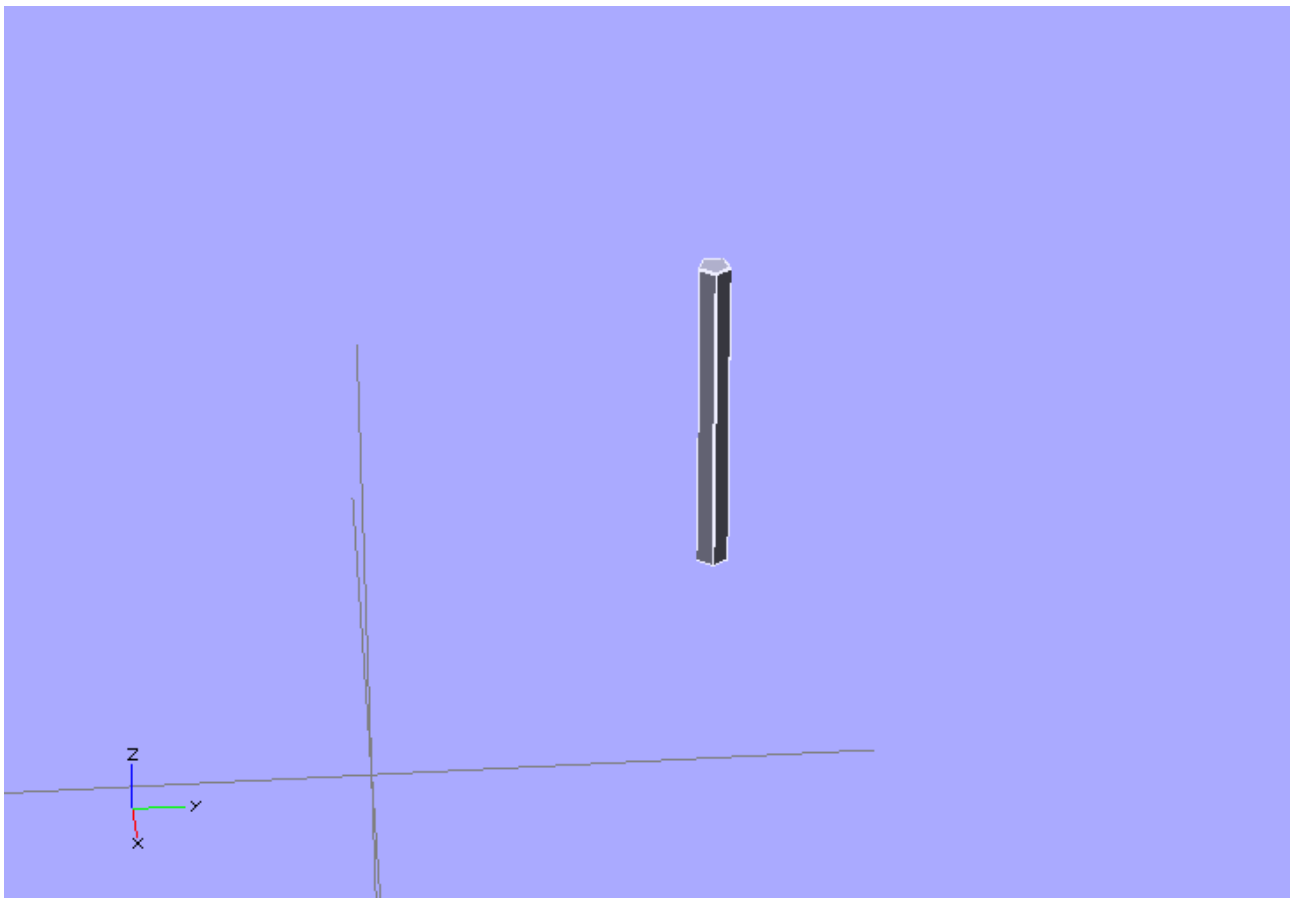
version 2



version 3

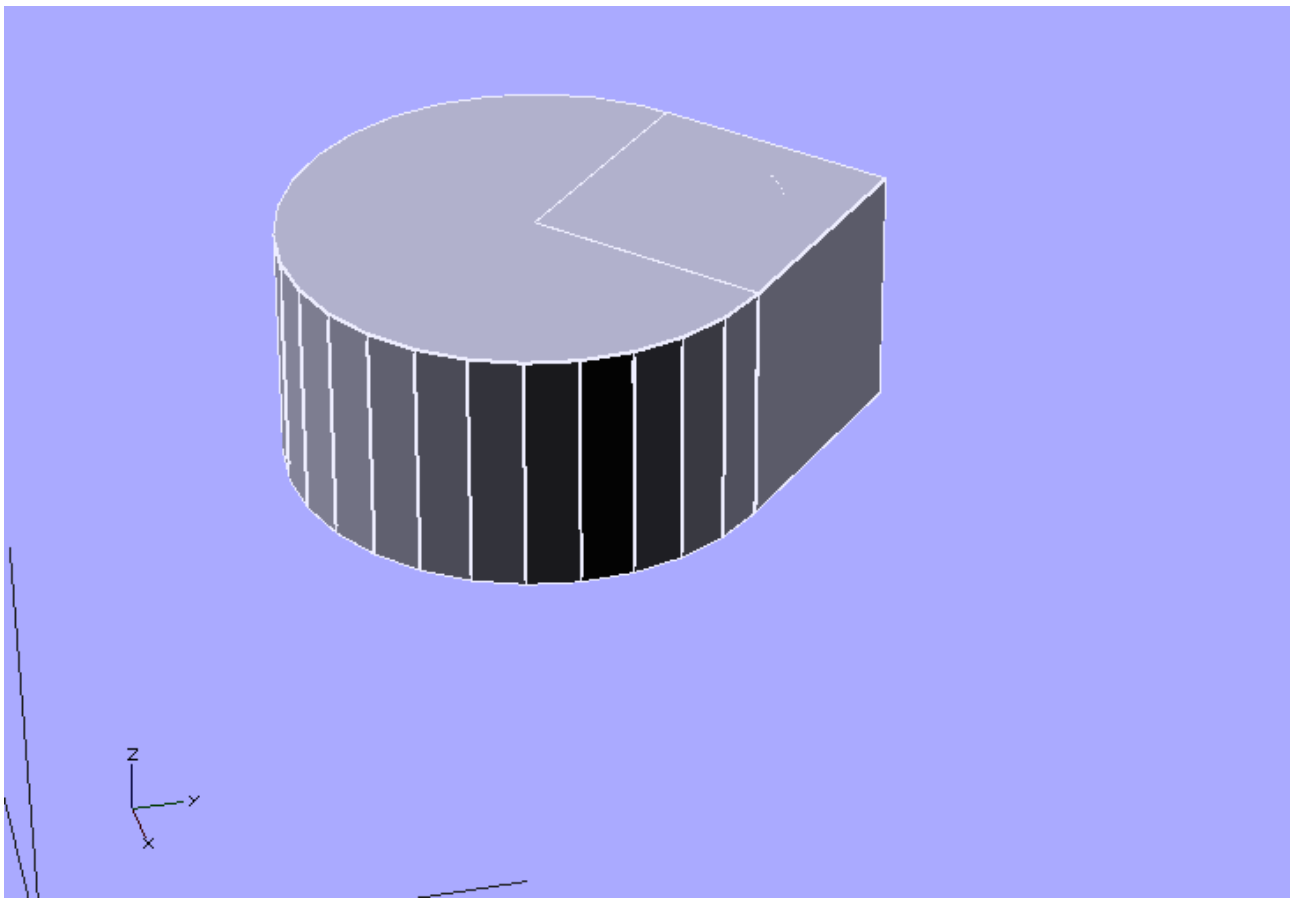
Multmatrix 多矩阵
倍增的几何是所有的子组件用给出的 4*4 转换数组矩阵。
（这一章我暂时还没有应用成功，正在测试）
句法描述：multmatrix(m = [...]) { ... }
实例（转换[10,20,30]）：

```
multmatrix(m = [ [1, 0, 0, 21],
[0, 1, 0, 20],
[0, 0, 1, 30],
[0, 0, 0, 1]
]) cylinder(20);
```



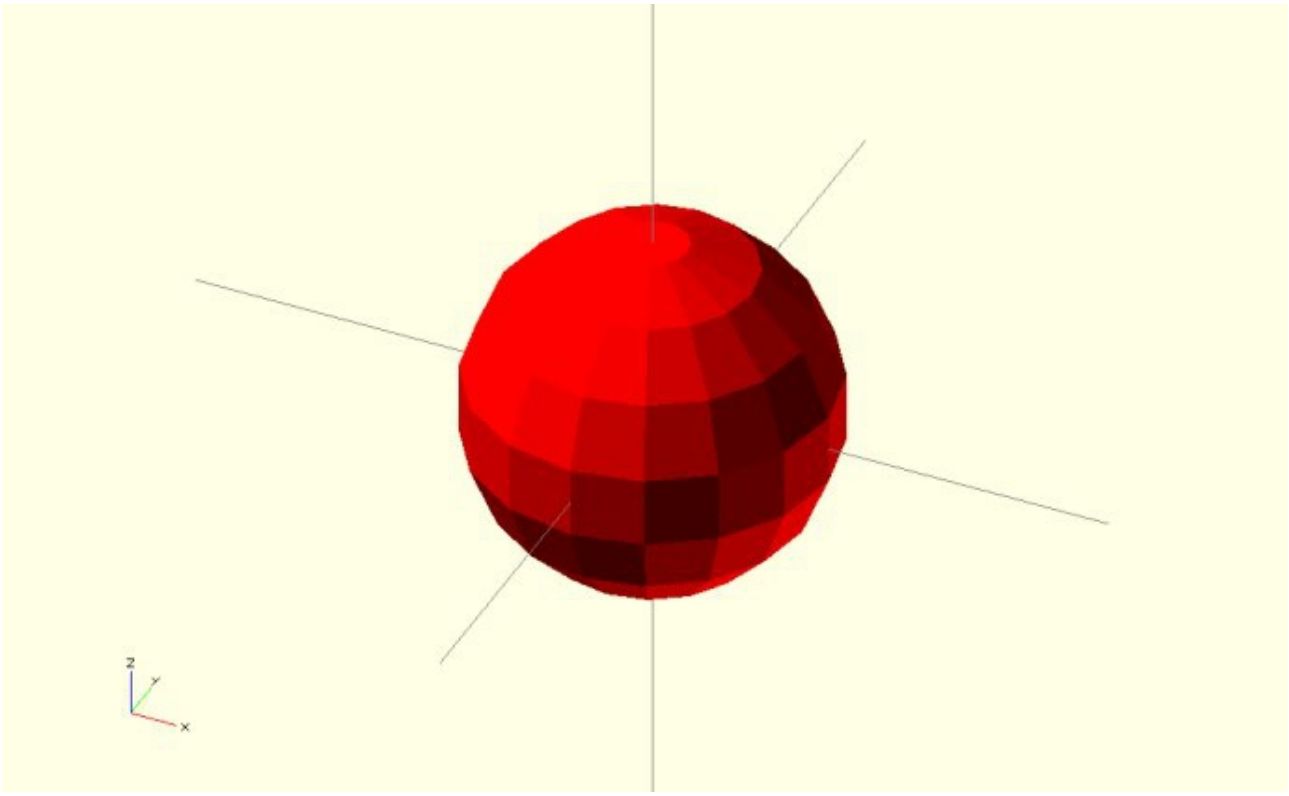
Example (rotates by 45 degrees in XY plane and translates by [10,20,30]):

```
angle=45;
multmatrix(m = [ [cos(angle), -sin(angle), 0, 10],
                 [sin(angle), cos(angle), 0, 20],
                 [0, 0, 1, 30],
                 [0, 0, 0,  1]
               ]) union() {
   cylinder(r=10.0,h=10,center=false);
   cube(size=[10,10,10],center=false);
}
```



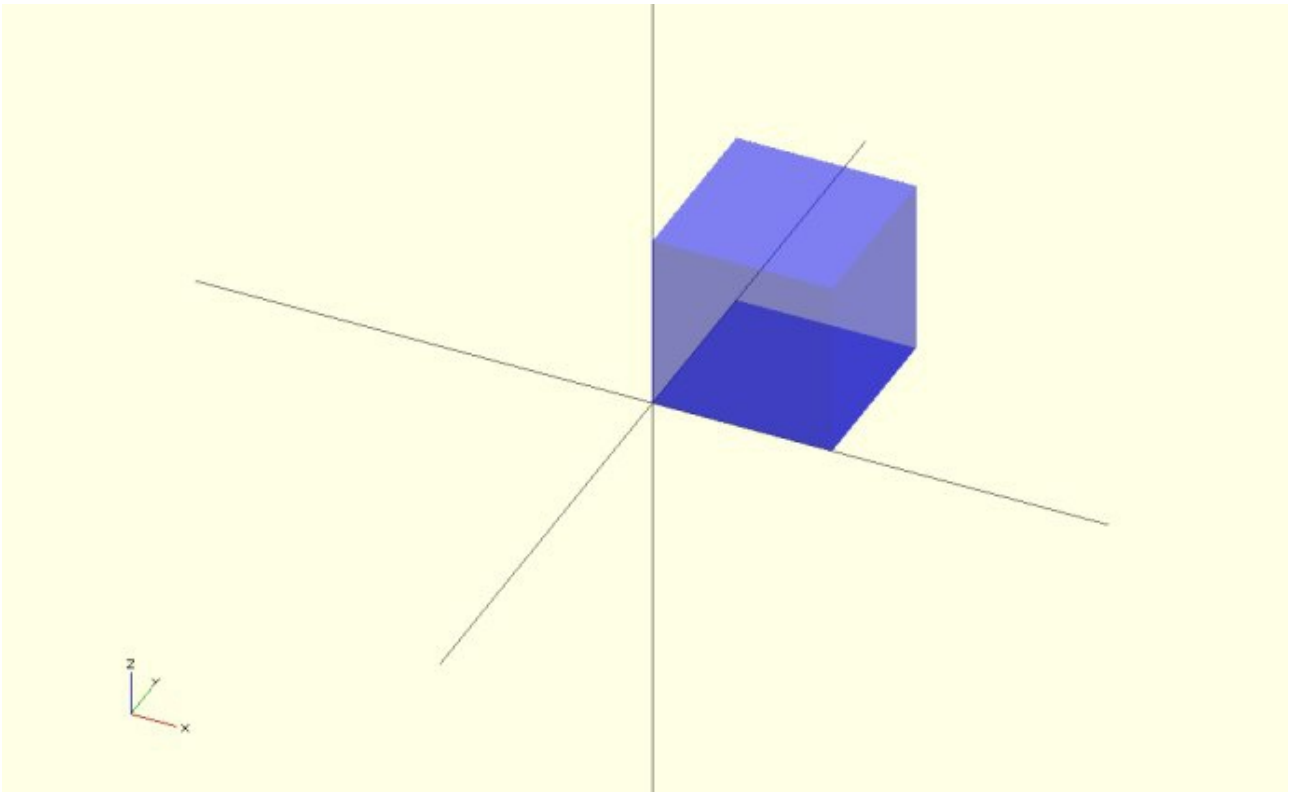
color 颜色

显示子组件使用设置的 RGB 颜色 + alpha 数值。 这个仅用于 F5 CGAL 预览，而 STL 的 F6 目前不支持颜色。alpha 数值将会默认到 1.0(不透明的) 如果你没有设置。
句法实例:
color([r, g, b, a]) { ... }
注意的是 r, g, b 一个数值会局限于浮点数的范围在{0.0 ~1.0},更优于其他传统的正整数{0~255}.
同样你可以配置数值就像是函数一样， 比如， R，G，B 正整数在{0~255}可以使用：
句法: color([R/255, G/255, B/255]) { ... }
就像 2011.12 版本，颜色同业可以选择名字；名字不是敏感的情况，比如，创建一个红色的球体，你可以使用下面的代码：实例：
color("red") sphere(5);



Alpha 同样可以和名字的颜色一起使用,实例： (alpha 这里是表示透明度的，仅仅输入数字 0.0～1.0)

```
color("Blue",0.5) cube(5);
```



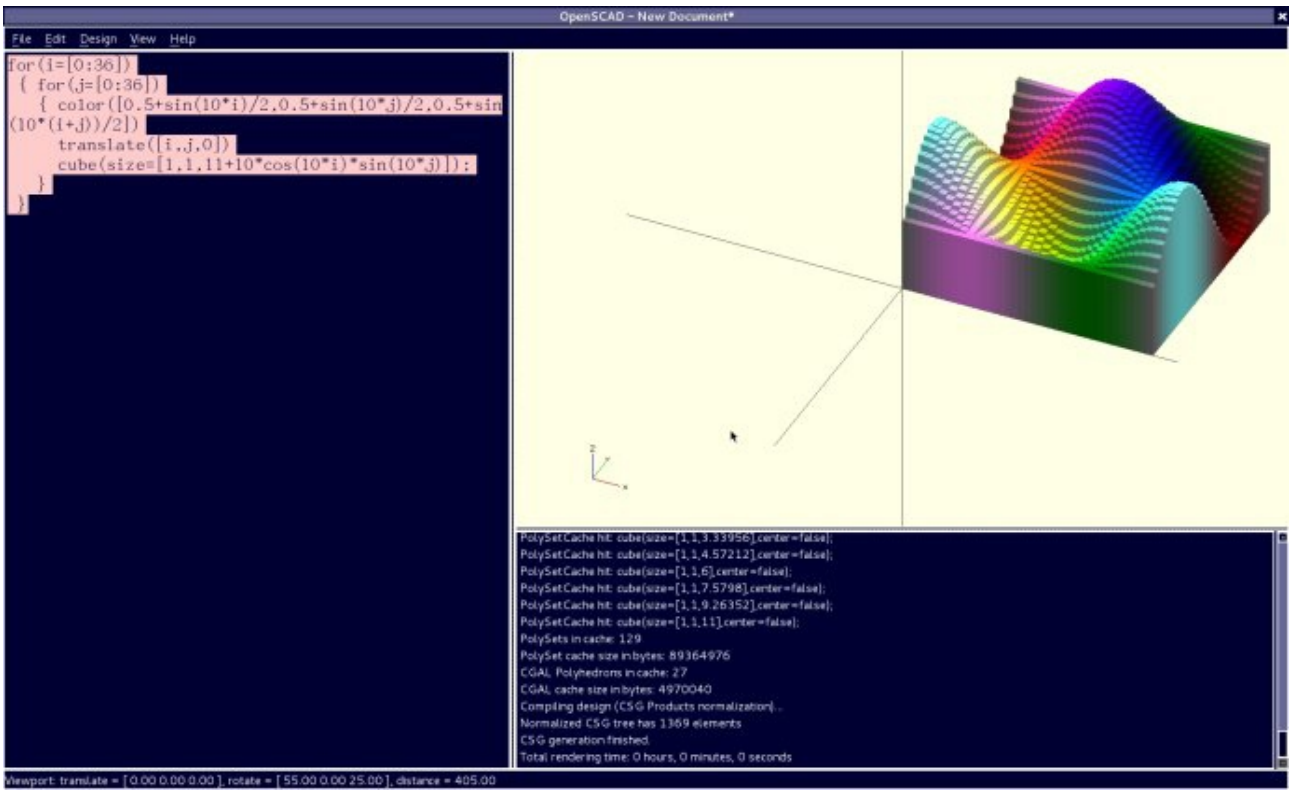
The available color names are taken from the World Wide Web consortium's SVG color list. A chart of the color names is as follows, (note that both spelling of grey/gray including lategrey/slategray etc are valid):
可以使用的颜色的名称是通过 World Wide Web 万维网协会的颜色列表。
一个图表用于颜色的名字如下，(注意不可以同时拼写 grey/gray 包括 slategray/slategray 等等类似的错误。

一个三维的多颜色的正弦波图

Here's a code fragment that draws a wavy multicolor object

这里是代码数据包片段用于生成一个多颜色的波形曲线对象模型，实例代码：

```
 for(i=[0:36])
 { for(j=[0:36])
   { color([0.5+sin(10*i)/2,0.5+sin(10*j)/2,0.5+sin(10*(i+j))/2])
     translate([i,j,0])
     cube(size=[1,1,11+10*cos(10*i)*sin(10*j)]);
   }
 }
```



就样 -1<=sin(x)<=1 ,然后 0<=(1/2 + sin(x)/2)<=1, 允许 for 循环 RGB 组件指定到颜色逗留在{0,1}区间。关于“网页颜色“的图表源自于维基百科。

Minkowski 闵可夫斯基

A box and a cylinder
一个盒子和一个圆柱体

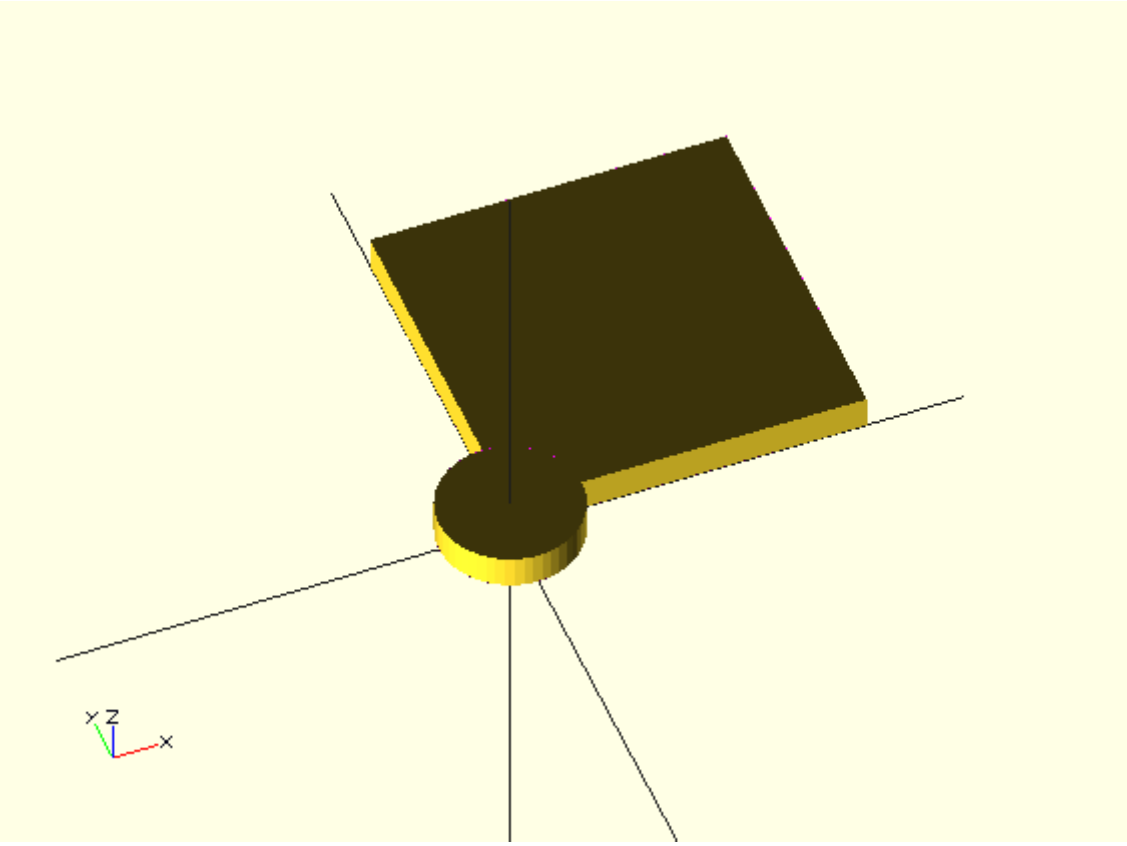
Minkowski sum of the box and cylinder
闵可夫斯基累加用于方体和圆柱体的效果

显示闵可夫斯基累加的子节点。
使用实例:
Say you have a flat box, and you want a rounded edge. There are many ways to do this, but minkowski is very elegant. Take your box, and a cylinder:
说你有一个平坦的盒子，你想要一个圆滑的边。那么许多的方法可以做，但是闵可夫斯基是非常优美的。拿一个方体盒子，和一个圆柱体。

```
$fn=50;
cube([10,10,1]);
cylinder(r=2,h=1);
```

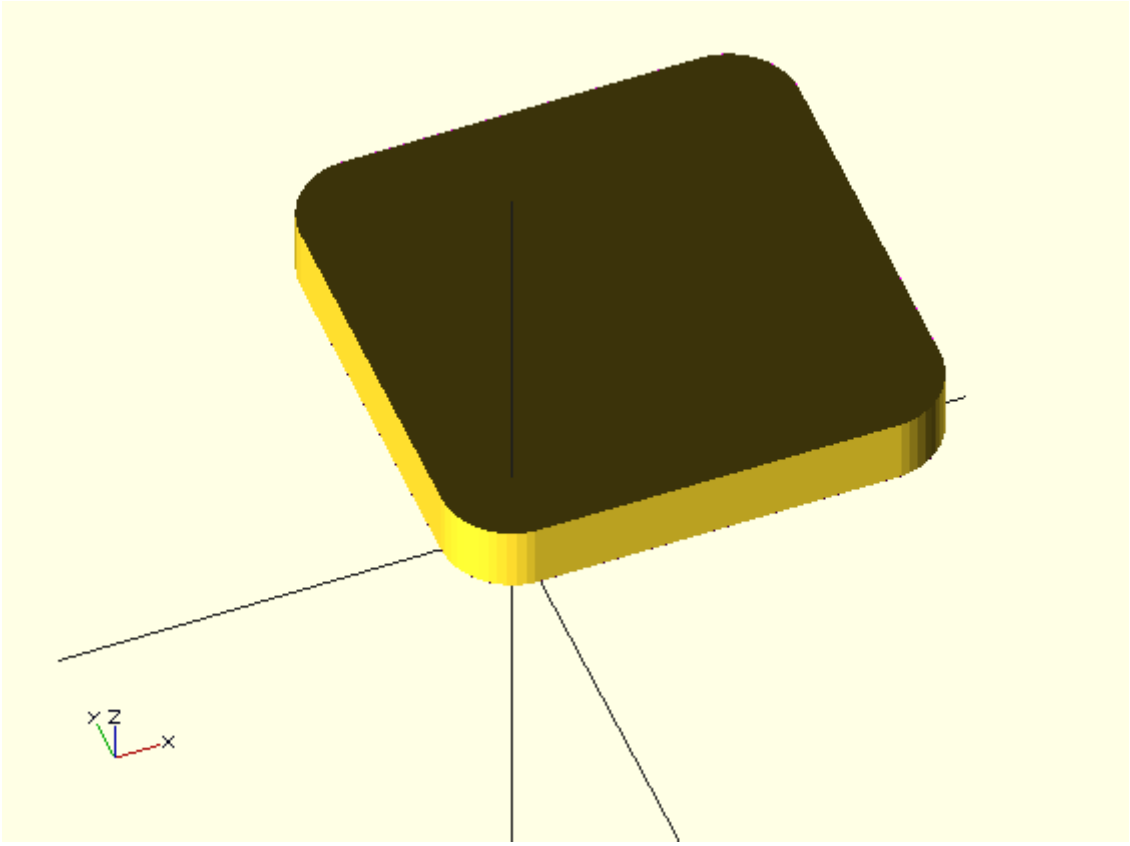
Then, do a minkowski sum of them:

```
$fn=50;
minkowski()
{
 cube([10,10,1]);
 cylinder(r=2,h=1);
}
```

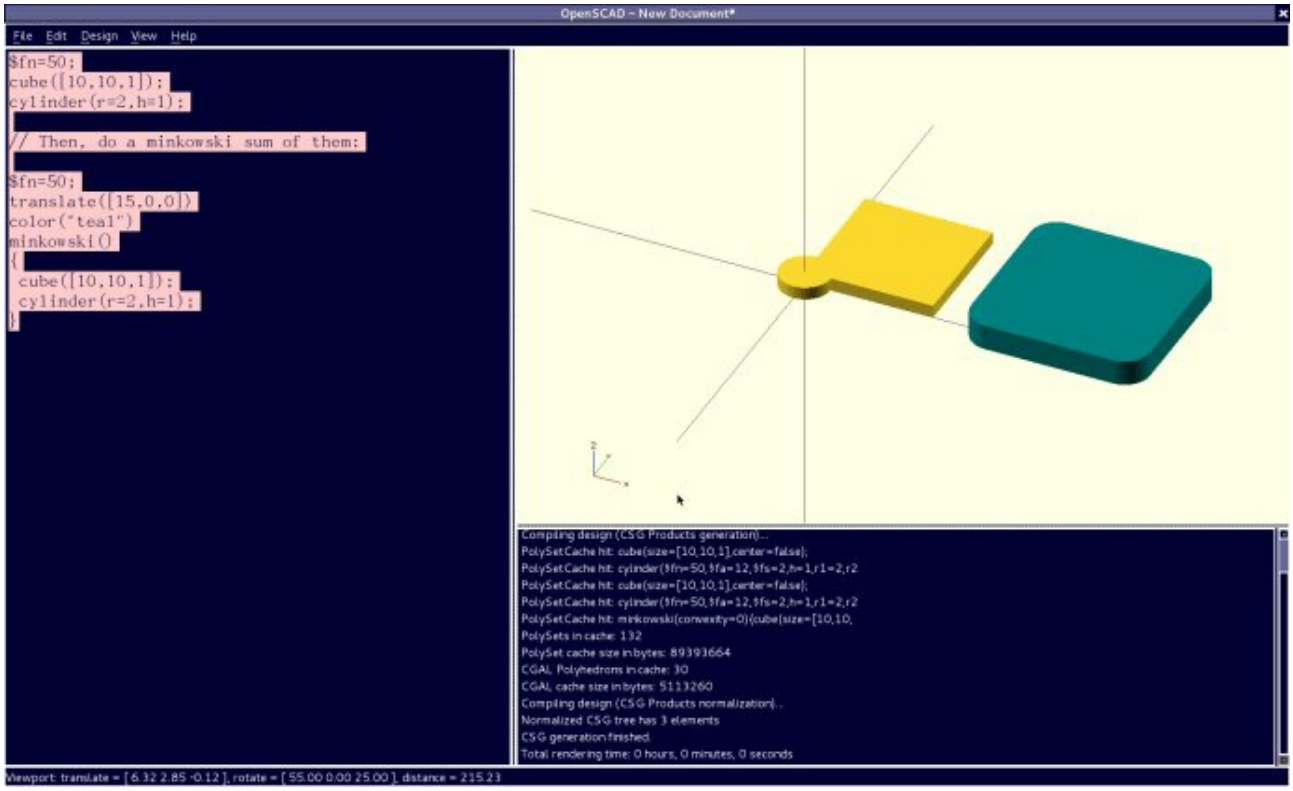


组合实例：

```
//first
$fn=50;
cube([10,10,1]);
cylinder(r=2,h=1);
```




```
// Then, do a minkowski sum of them:

$fn=50;
translate([15,0,0])
color("teal")
minkowski()
{
 cube([10,10,1]);
 cylinder(r=2,h=1);
}
```

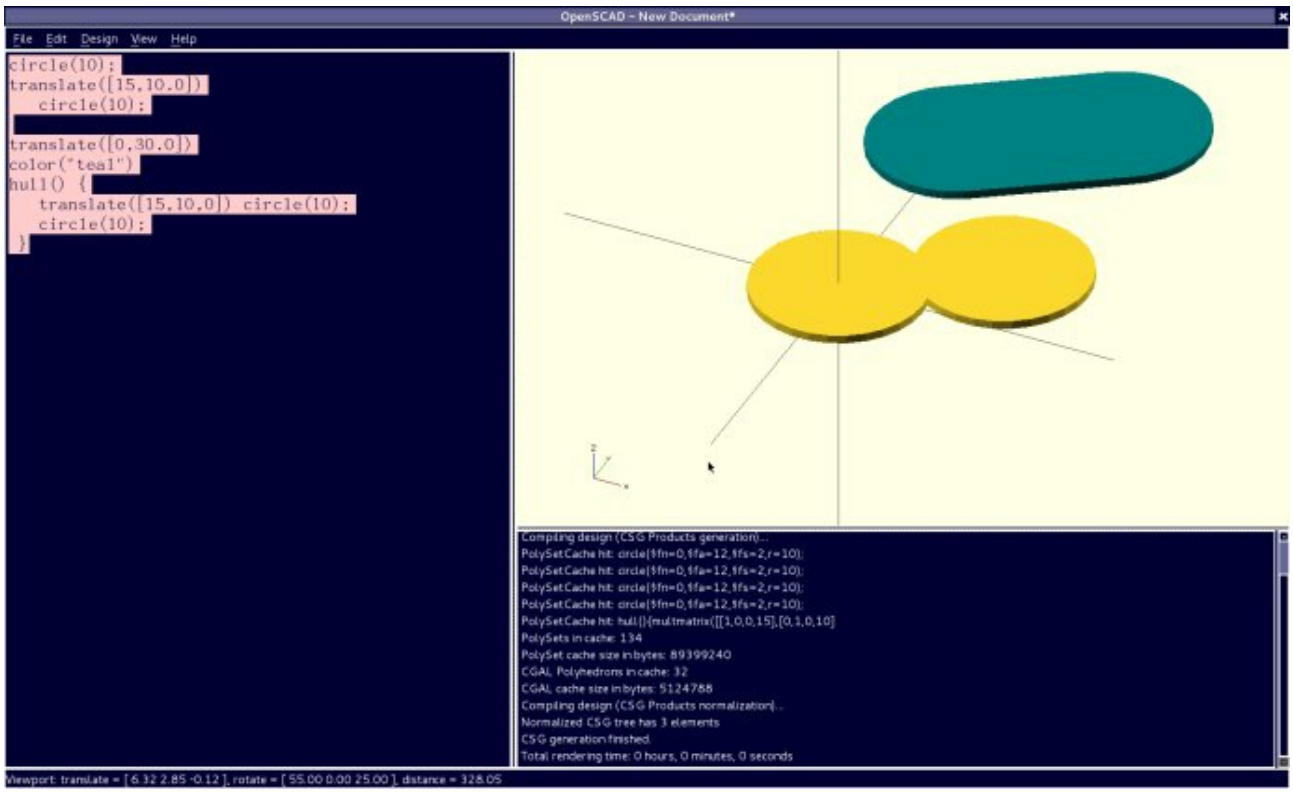


hull 船体

两个圆柱体

凸圆船体的两个圆柱体
显示凸圆船体的子节点
使用实例:

```
hull() {
   translate([15,10,0]) circle(10);
   circle(10);
 }
```


OpenSCAD - New Document*
File Edit Design View Help
circle(10);
translate([15,10,0])
circle(10);
translate([0,30,0])
color("teal")
hull() {
translate([15,10,0]) circle(10);
circle(10);
}
Compiling design (CSG Products generation)...
PolySetCache hit: circle($fn=0,$fa=12,$fs=2,r=10);
PolySetCache hit: circle($fn=0,$fa=12,$fs=2,r=10);
PolySetCache hit: circle($fn=0,$fa=12,$fs=2,r=10);
PolySetCache hit: circle($fn=0,$fa=12,$fs=2,r=10);
PolySetCache hit: hull(){multmatrix([[1,0,0,15],[0,1,0,10]
PolySets in cache: 134
PolySet cache size in bytes: 89399240
CGAL Polyhedrons in cache: 32
CGAL cache size in bytes: 5124788
Compiling design (CSG Products normalization)...
Normalized CSG tree has 3 elements
CSG generation finished.
Total rendering time: 0 hours, 0 minutes, 0 seconds
Viewport: translate = [6.32 2.85 -0.12], rotate = [55.00 0.00 25.00], distance = 328.05

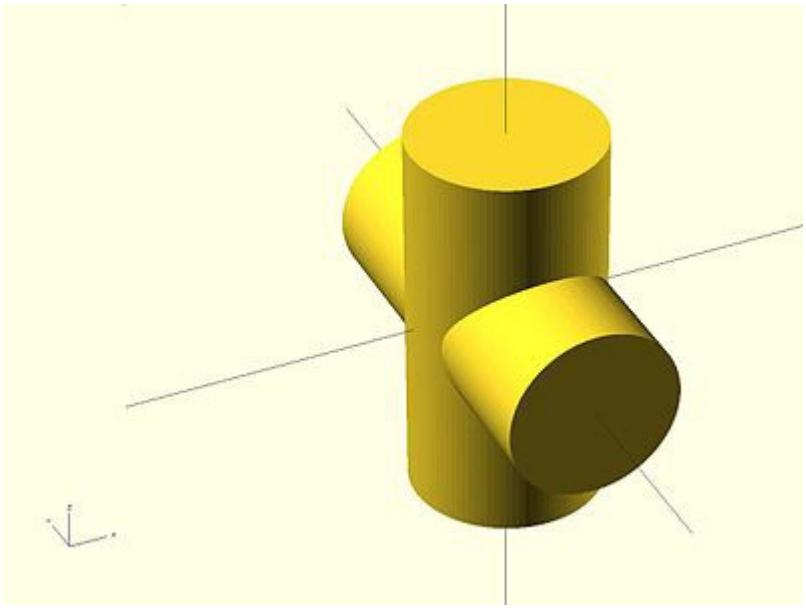
10，CSG Modeling CSG模型
1.10.1 union 联合
1.10.2 difference 分割
1.10.3 intersection 求交
1.10.4 render 穿透(渲染)
11，Modifier Characters 特征修整

1.11.1 Background Modifier 背景修整
1.11.2 Debug Modifier 调试修整
1.11.3，Root Modifier 根 修整
1.11.4，Disable Modifier 无效修整
12Modules 模块

CSG模型

union联合
穿件一个联合到所有的子节点。这个是求和sum到所有的子节点。
Usage example:
应用实例：

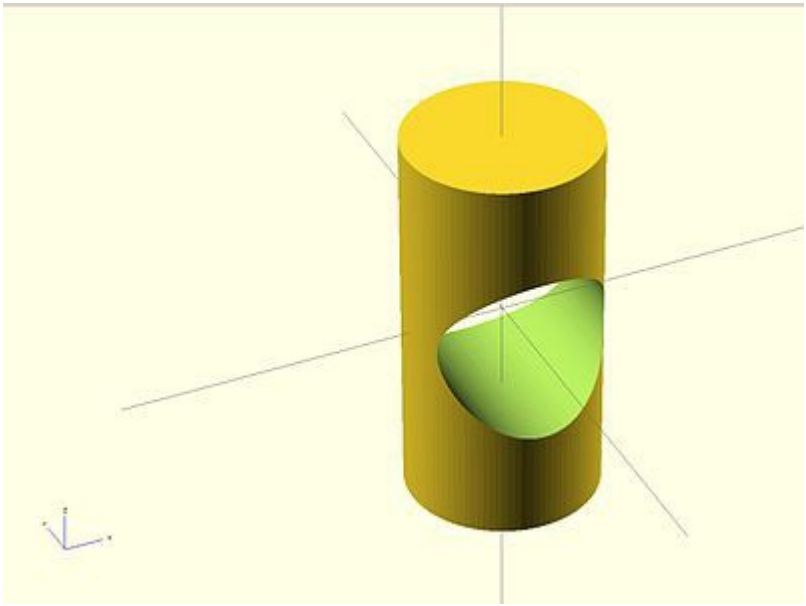
```
union() {
        cylinder (h = 4, r=1, center = true, $fn=100);
        rotate ([90,0,0]) cylinder (h = 4, r=0.9, center = true, $fn=100);
}
```



声明：联合是一个暗示的时候是不可以用的。不过这个是托管的，比如，在分割到一个群组首先子节点到一个里面。

difference 分割
减去第二个（所有的更远的）子节点从第一个。
Usage example:
应用实例：

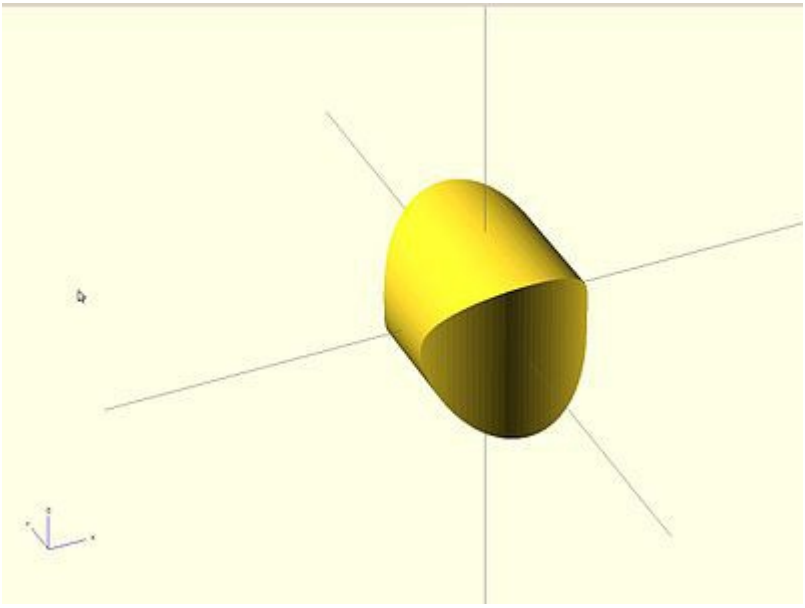
```
difference() {
        cylinder (h = 4, r=1, center = true, $fn=100);
        rotate ([90,0,0]) cylinder (h = 4, r=0.9, center = true, $fn=100);
}
```



intersection 求交
建立一个求交到所有的子节点。这个仅保持其重叠部分。

Usage example:
实用案例：

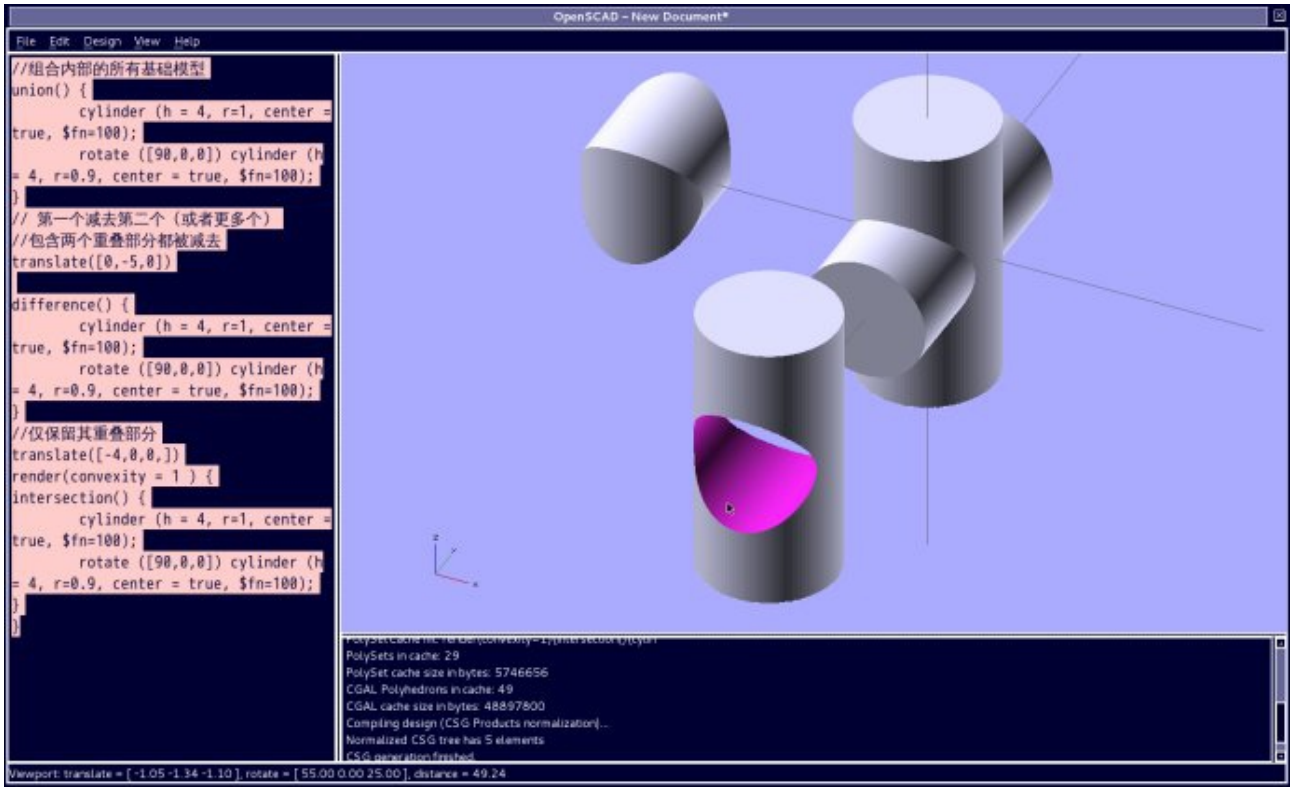
```
intersection() {
        cylinder (h = 4, r=1, center = true, $fn=100);
        rotate ([90,0,0]) cylinder (h = 4, r=0.9, center = true, $fn=100);
}
```



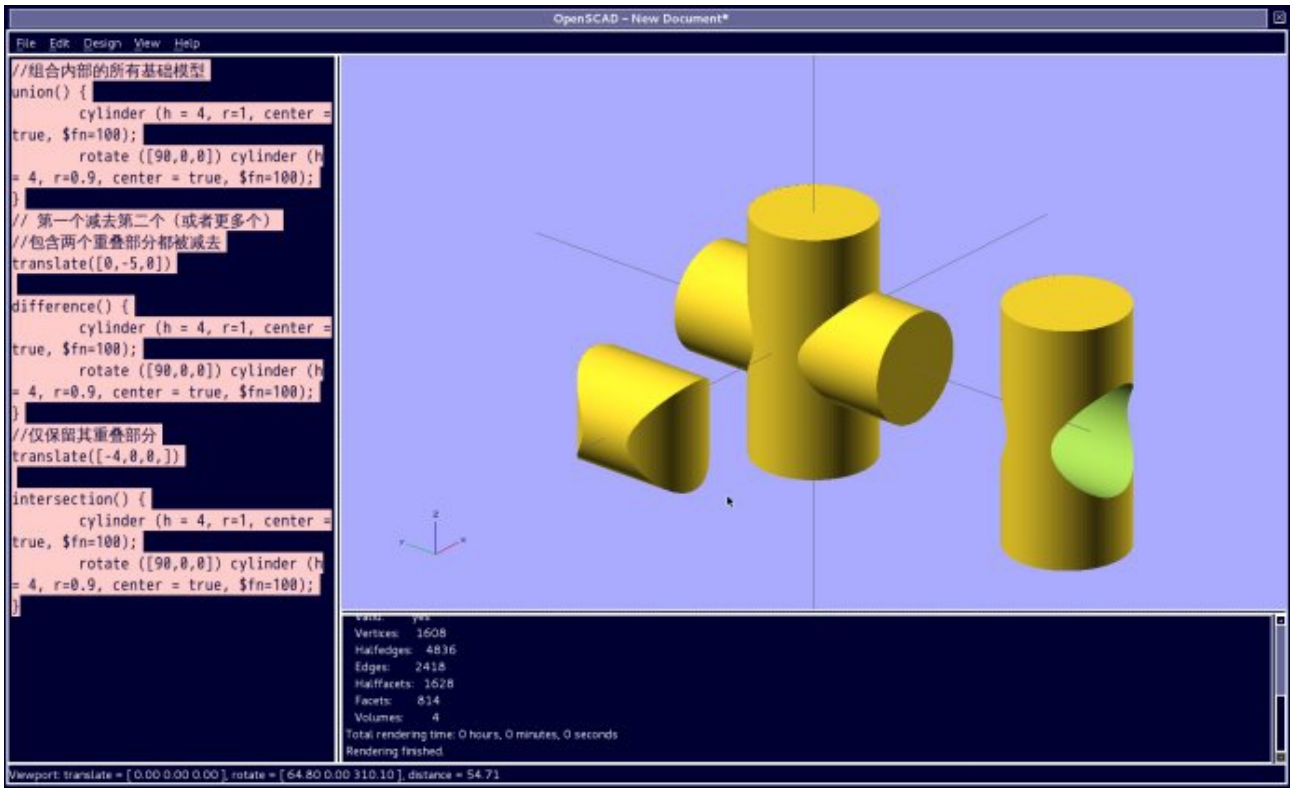
render 穿透渲染
总是计算 CSG 模型用于这个树（尽管是 OpenCSG 预览模式）。凸面参数配置最大的数值是前侧（后侧）一个射线交叉到目标对象应该穿透。这个参数是仅仅需要正确的显示对象在 OpenCSG 模式，对于多变体穿透渲染无效。
Usage example:
render(convexity = 1) { ... }
我反正测试了半天，没有发现如何使用这个函数。

所有 CSG 模型函数应用实例：

```
//组合内部的所有基础模型
union() {
        cylinder (h = 4, r=1, center = true, $fn=100);
        rotate ([90,0,0]) cylinder (h = 4, r=0.9, center = true, $fn=100);
}
// 第一个减去第二个（或者更多个）
//包含两个重叠部分都被减去
translate([0,-5,0])

difference() {
        cylinder (h = 4, r=1, center = true, $fn=100);
        rotate ([90,0,0]) cylinder (h = 4, r=0.9, center = true, $fn=100);
}
//仅保留其重叠部分
translate([-4,0,0,])
render(convexity = 1 ) {
intersection() {
        cylinder (h = 4, r=1, center = true, $fn=100);
        rotate ([90,0,0]) cylinder (h = 4, r=0.9, center = true, $fn=100);
}
}
```



Version 1



version 2

修整字符

修整字符是用于更改其子节点的出现或者行为的某个特殊符号。他们特别是应用于调试那些可以用于突出的特殊模型，从渲染中或者包容，或者排斥他们。

深入理解：

就像是 OpenSCAD 用的不同的软件库到应用的容量，这个可以介绍一些不一致的效果到 F5 预览行为的转换。传统的转换（位置移动，旋转，比例，镜像和多个矩阵）是应用 OpenGL 演化出来预览的，其他的更多先进的转换，比如，尺寸大小， 演示一个 CGAL 操作，行为就像 CSG 操作的影响下面的模型，不是仅仅转换它。求其是这个修整字符可以影响显示效果。，具体比如"#"， "%"， 哪里的亮度也许不能够显示直观的，就像是加亮预从定义尺寸的模型，不过，加亮的是后比例的模型。

注意：颜色的改变触发到字符修整将会仅仅显示在"Compile"" 编译"模式，不会在"compile and Render(CGAL)" 模式。（参见颜色章节）

Background Modifier 背景修整

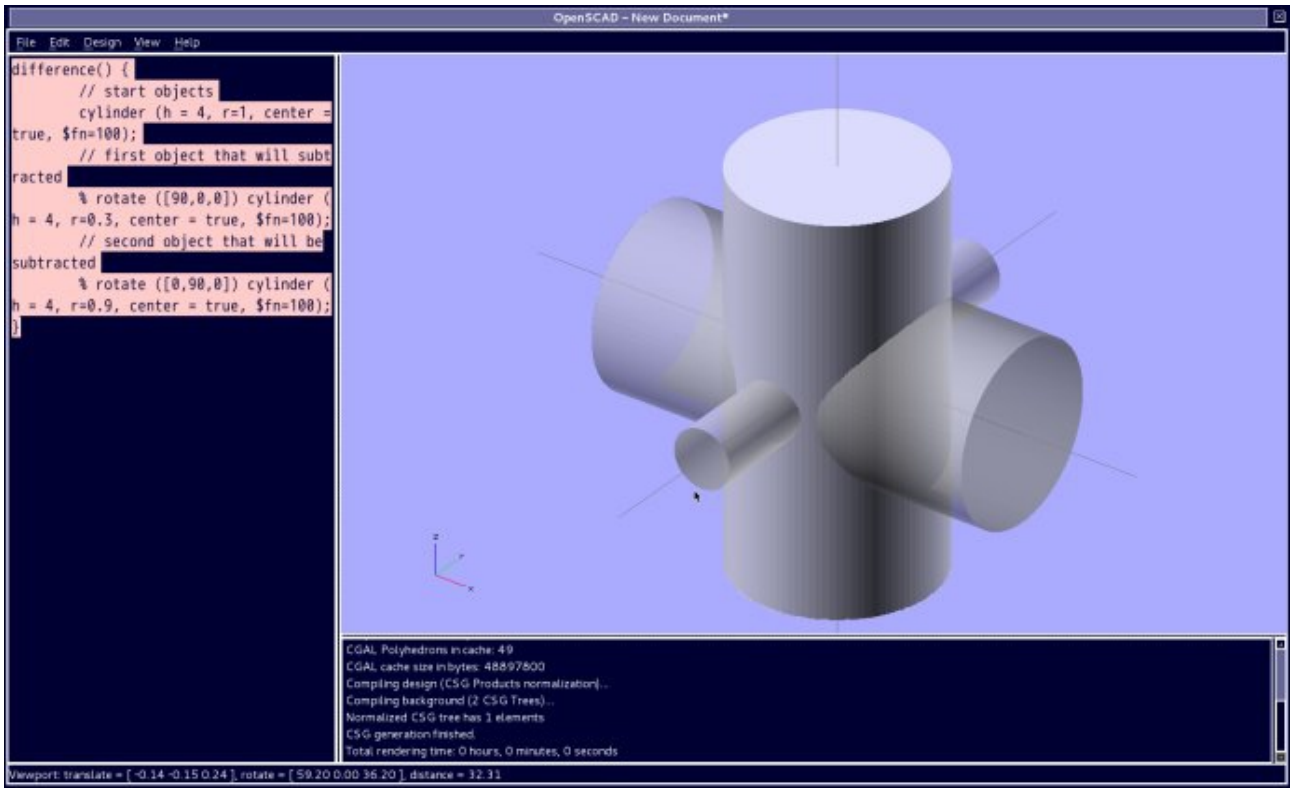
忽视这个子树叉到正常的渲染过程，画一个有透明灰色的模型（所以的转换是仍然应用到节点的在这个树叉上）

应用句法：

```
% { ... }
```

案例代码：

```
difference() {
        // start objects
        cylinder (h = 4, r=1, center = true, $fn=100);
        // first object that will subtracted
        % rotate ([90,0,0]) cylinder (h = 4, r=0.3, center = true, $fn=100);
        // second object that will be subtracted
        % rotate ([0,90,0]) cylinder (h = 4, r=0.9, center = true, $fn=100);
}
```

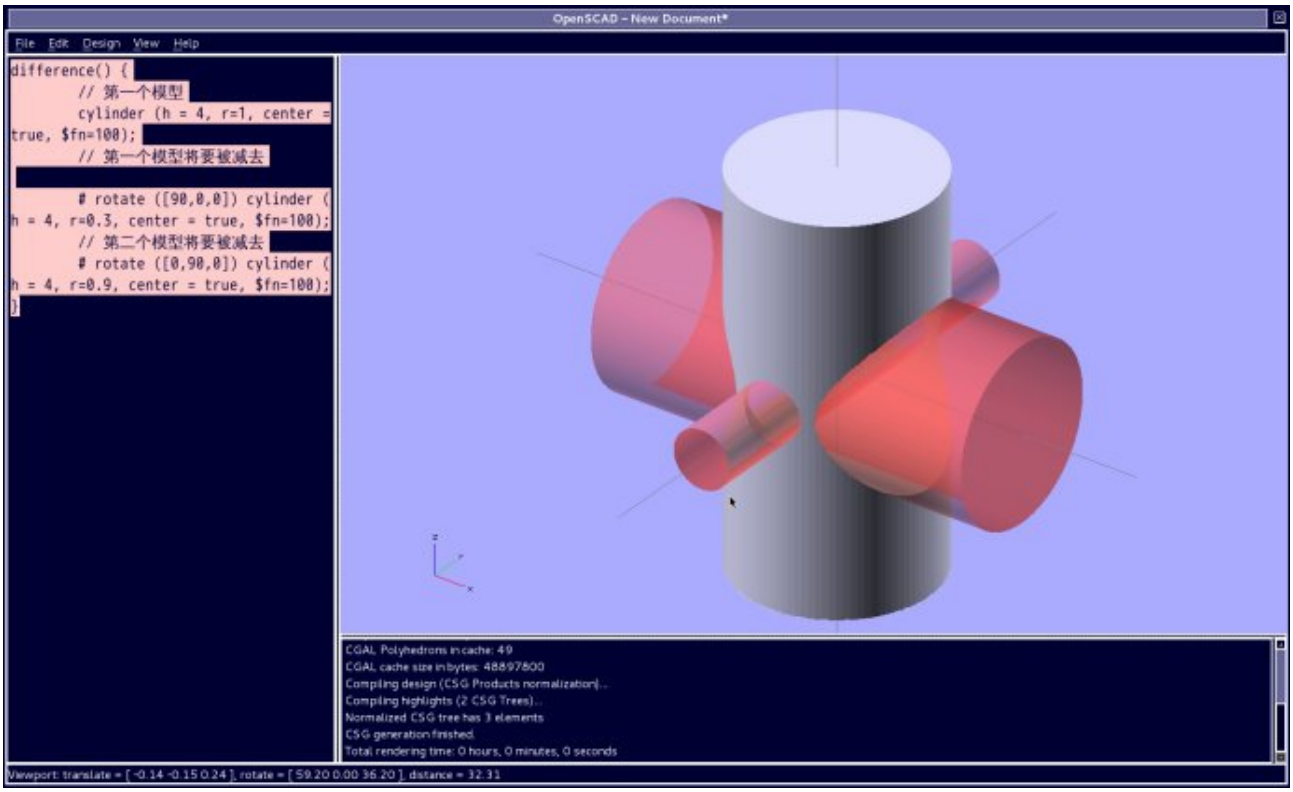


背景修整实例

Debug Modifier 调试修整

用于这个子树（大括号内的模型）就像普通的一样渲染过程，但是同样画一个没有修整的转换粉色。

```
 Usage example:
 # { ... }
Example:
difference() {
        // start objects
        cylinder (h = 4, r=1, center = true, $fn=100);
        // first object that will subtracted
        # rotate ([90,0,0]) cylinder (h = 4, r=0.3, center = true, $fn=100);
        // second object that will be subtracted
        # rotate ([0,90,0]) cylinder (h = 4, r=0.9, center = true, $fn=100);
}
```



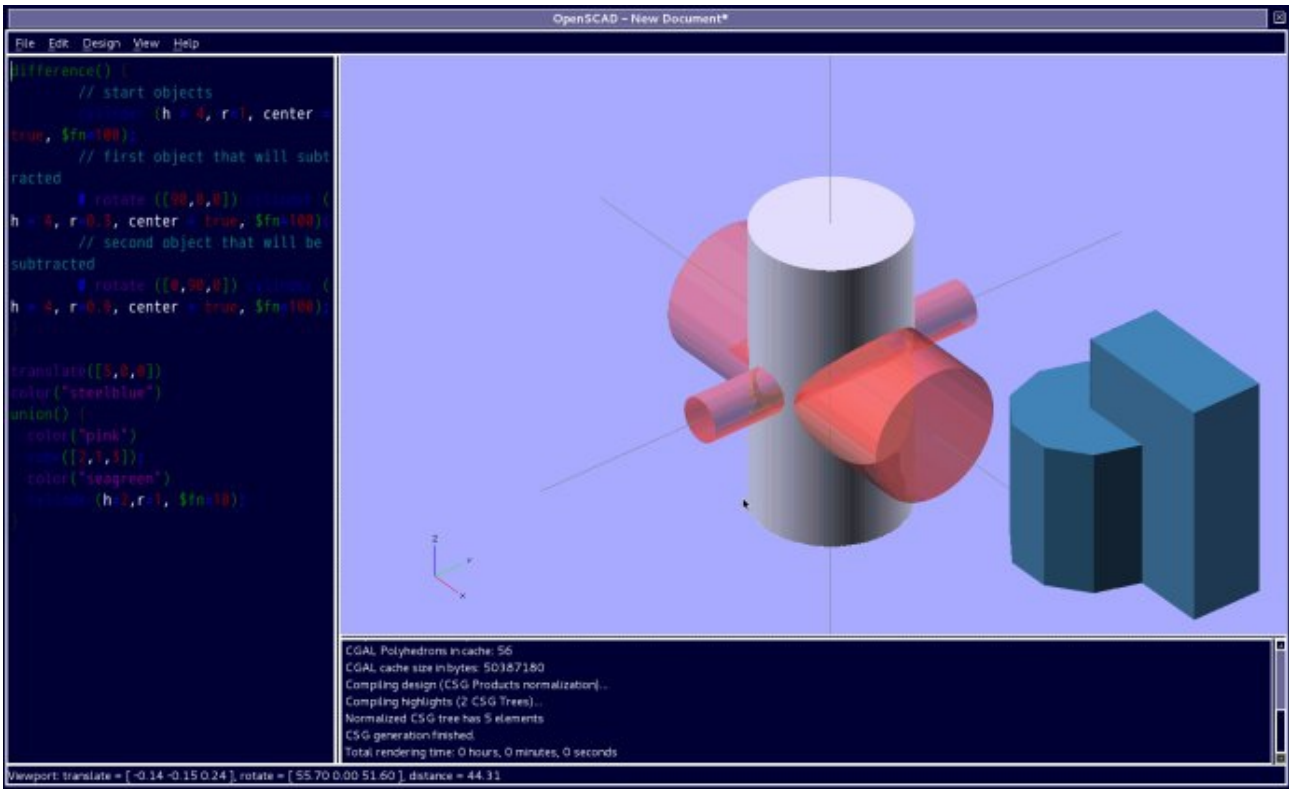
OpenScad 调试修整实例

Root Modifier 根修整
忽视设计的剩余部分，用这个子树就像是设计的根。

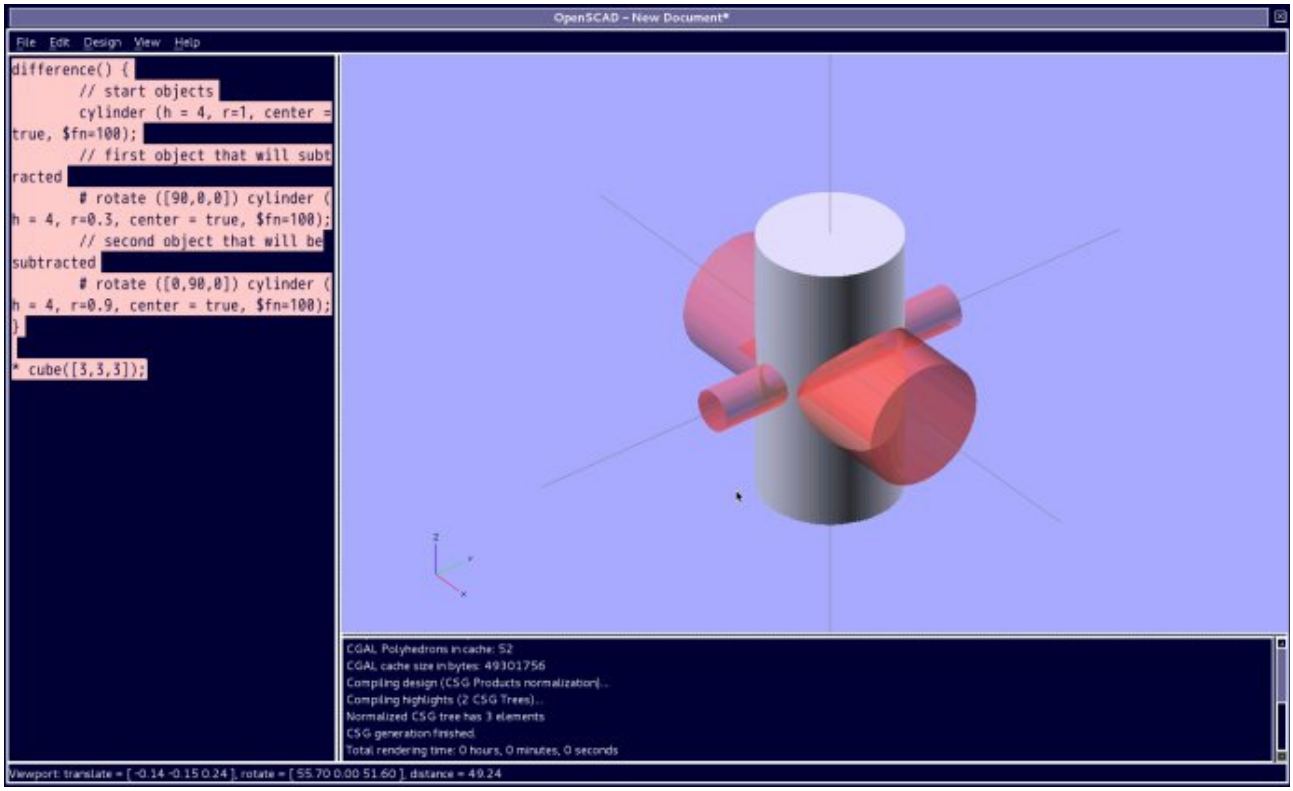
句法:
```
! { ... }
```

应用实例：（我对每一个组合和单个模型进行了根命令“！”的操作，也可以理解为仅显示当前，忽视剩余的部分）

```
difference() {
        // start objects
        cylinder (h = 4, r=1, center = true, $fn=100);
        // first object that will subtracted
        # rotate ([90,0,0]) cylinder (h = 4, r=0.3, center = true, $fn=100);
       // second object that will be subtracted
        # rotate ([0,90,0]) cylinder (h = 4, r=0.9, center = true, $fn=100);
}

translate([5,0,0])
color("steelblue")
union() {
  color("pink")
  cube([2,1,3]);
  color("seagreen")
  cylinder(h=2,r=1, $fn=10);
}
```



Disable Modifier 无效修整
简单的忽视这个相关的子树。（可以理解为忽视当前对象模型）
句法：
* { ... }
使用案例：



P-4.9

模块

定义你自己的模块（粗略的比较一个宏描述或者是一个函数在其他语言中的）是一个强大的方法用于重新使用的手续。
例如：

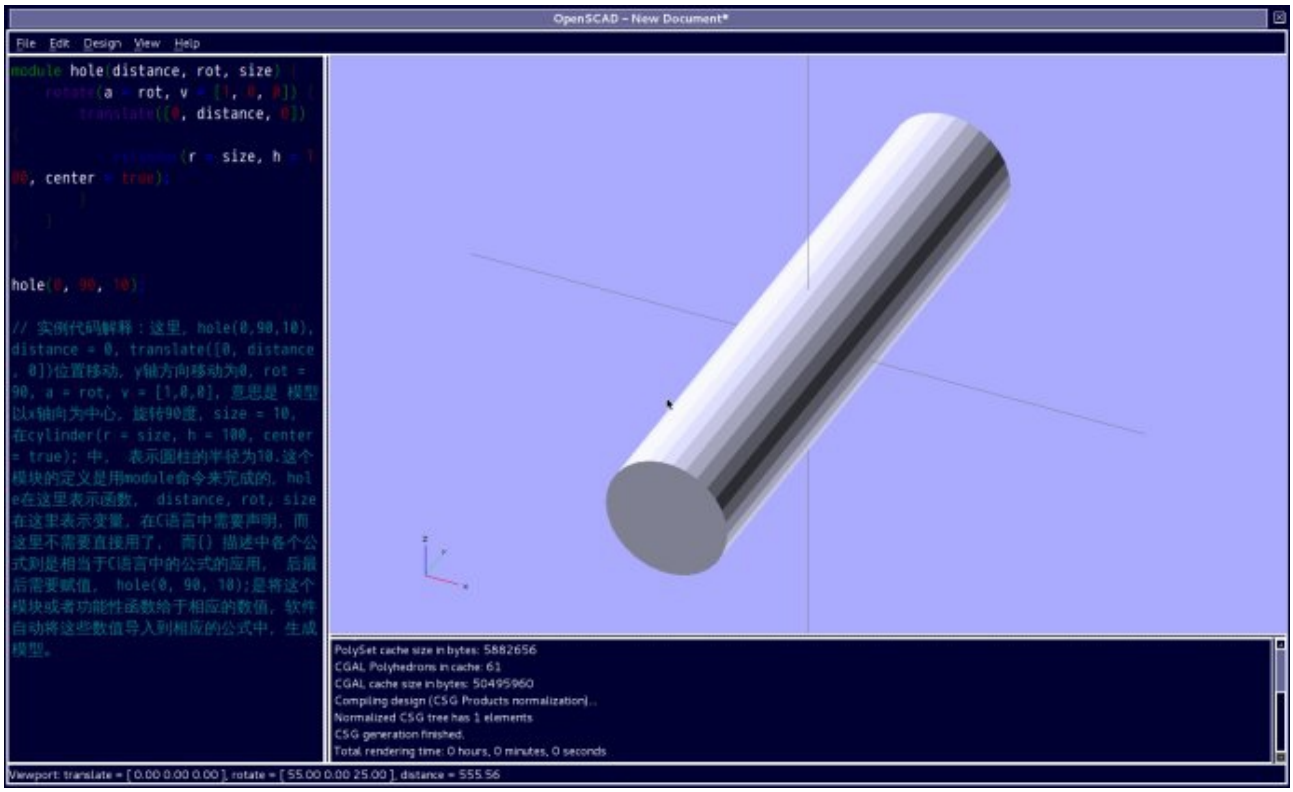
```
module hole(distance, rot, size) {
    rotate(a = rot, v = [1, 0, 0]) {
        translate([0, distance, 0]) {
            cylinder(r = size, h = 100, center = true);
        }
    }
}
```

在这个实例中，通过其中的 distance(距离)rot(旋转) size(尺寸)，允许你重新使用这个功能性的乘积幂，保存很多行代码并且渲染你的程序更加的简单。类似于 C 语言中的 struct，或者 scanf 编译后的函数。

你可以例证模型通过数值（或者公式）到参数就像是C语言的函数call:
```
hole(0, 90, 10);
```

完整的实例代码：
```
module hole(distance, rot, size) {
    rotate(a = rot, v = [1, 0, 0]) {
        translate([0, distance, 0]) {
            cylinder(r = size, h = 100, center = true);
        }
    }
}
hole(0, 90, 10);
```

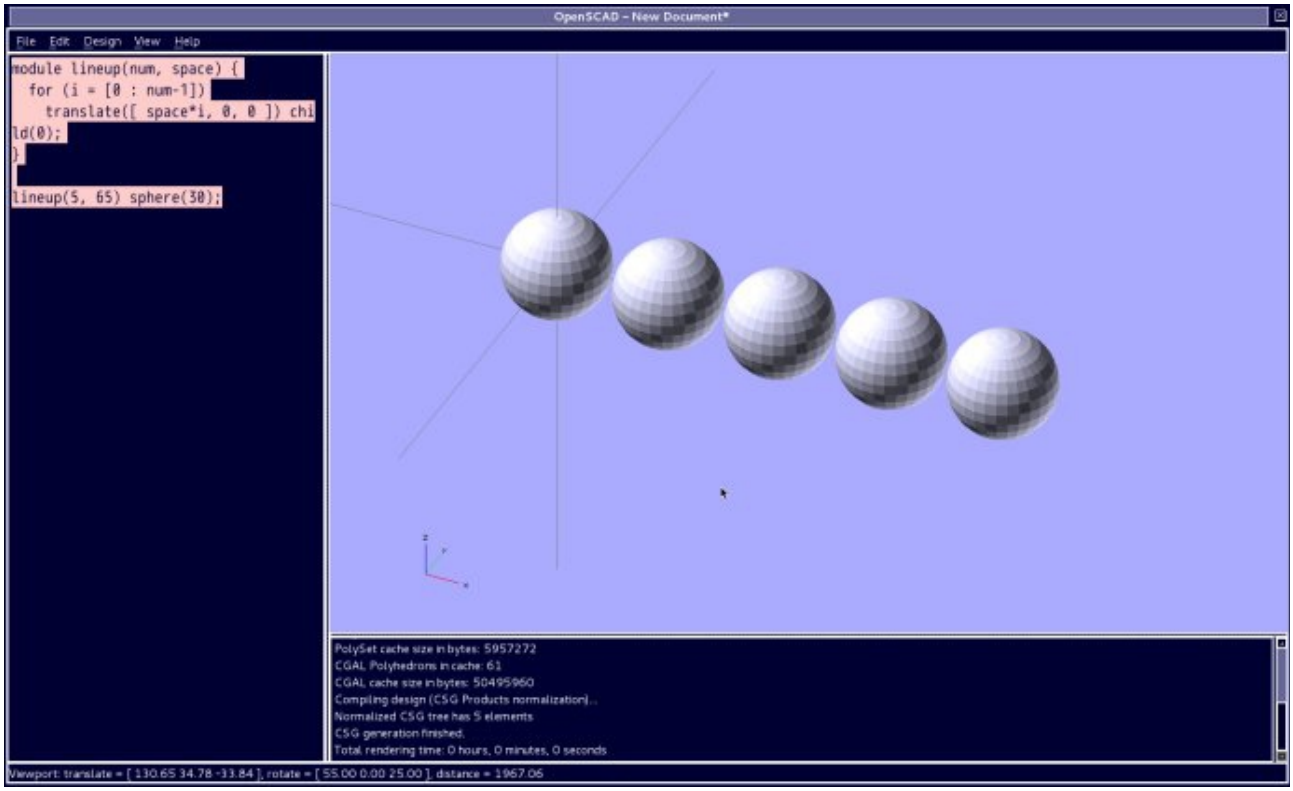
模型图片如下：



P-4.10
实例代码解释：这里，hole(0,90,10)，
distance = 0, translate([0, distance, 0])位置移动，y轴方向移动为0，
rot = 90, a = rot, v = [1,0,0]，意思是 模型以x轴向为中心，旋转90度，
size = 10, 在cylinder(r = size, h = 100, center = true); 中，表示圆柱的半径为10.
这个模块的定义是用module命令来完成的，hole在这里表示函数，distance, rot, size 在这里表示变量，在C语言中需要声明，而这里不需要直接用了，而{} 描述中各个公式则是相当于C语言中的公式的应用，后最后需要赋值，hole(0, 90, 10);是将这个模块或者功能性函数给于相应的数值，软件自动将这些数值导入到相应的公式中，生成模型。

The child nodes of the module instantiation can be accessed using the child() statement within the module:
子节点的模块例证可以访问使用 child() 声明在模块中：

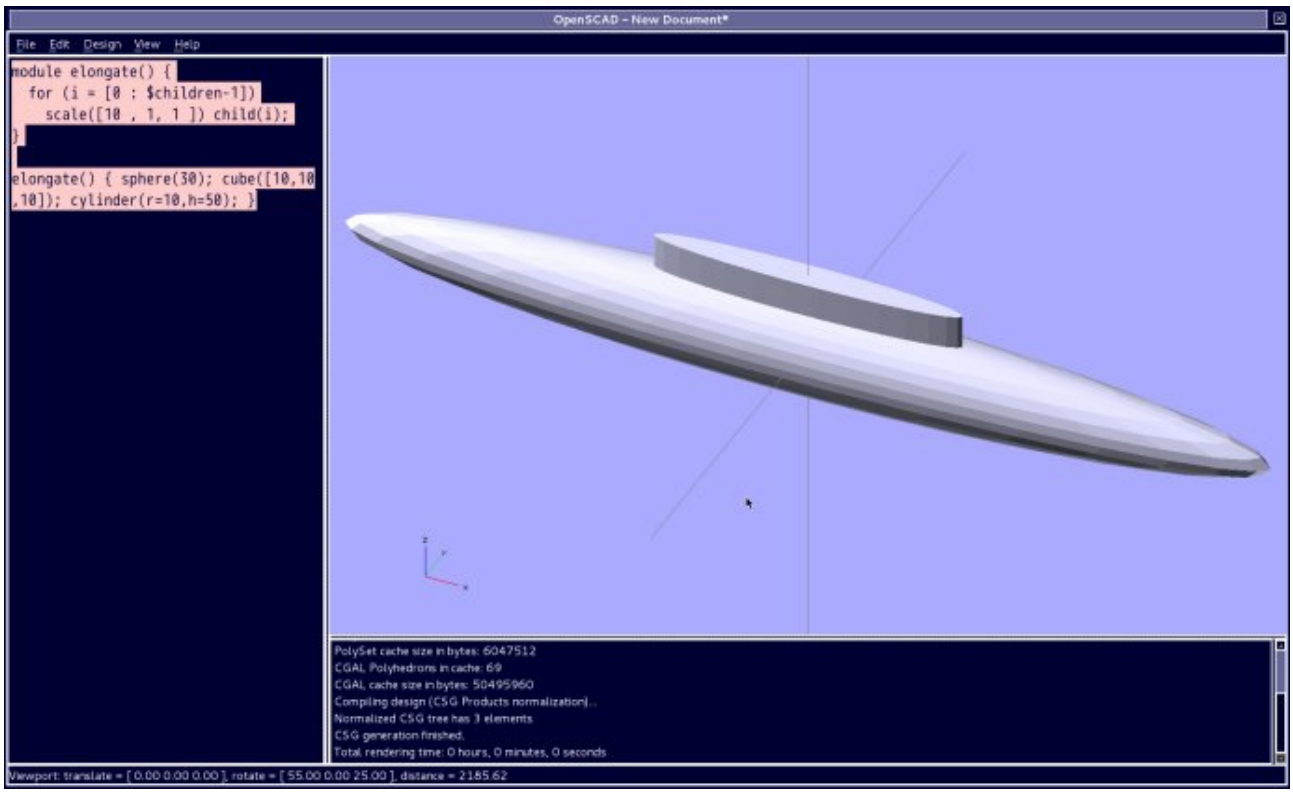
```
module lineup(num, space) {
  for (i = [0 : num-1])
    translate([ space*i, 0, 0 ]) child(0);
}

lineup(5, 65) sphere(30);
```



P-4.11

If you need to make your module iterate over all children you will need to make use of the $children variable, e.g.:
如果你需要制作你的模块重复所有的子对象模型你将会需要使用$符号子对象模型变量，例如：

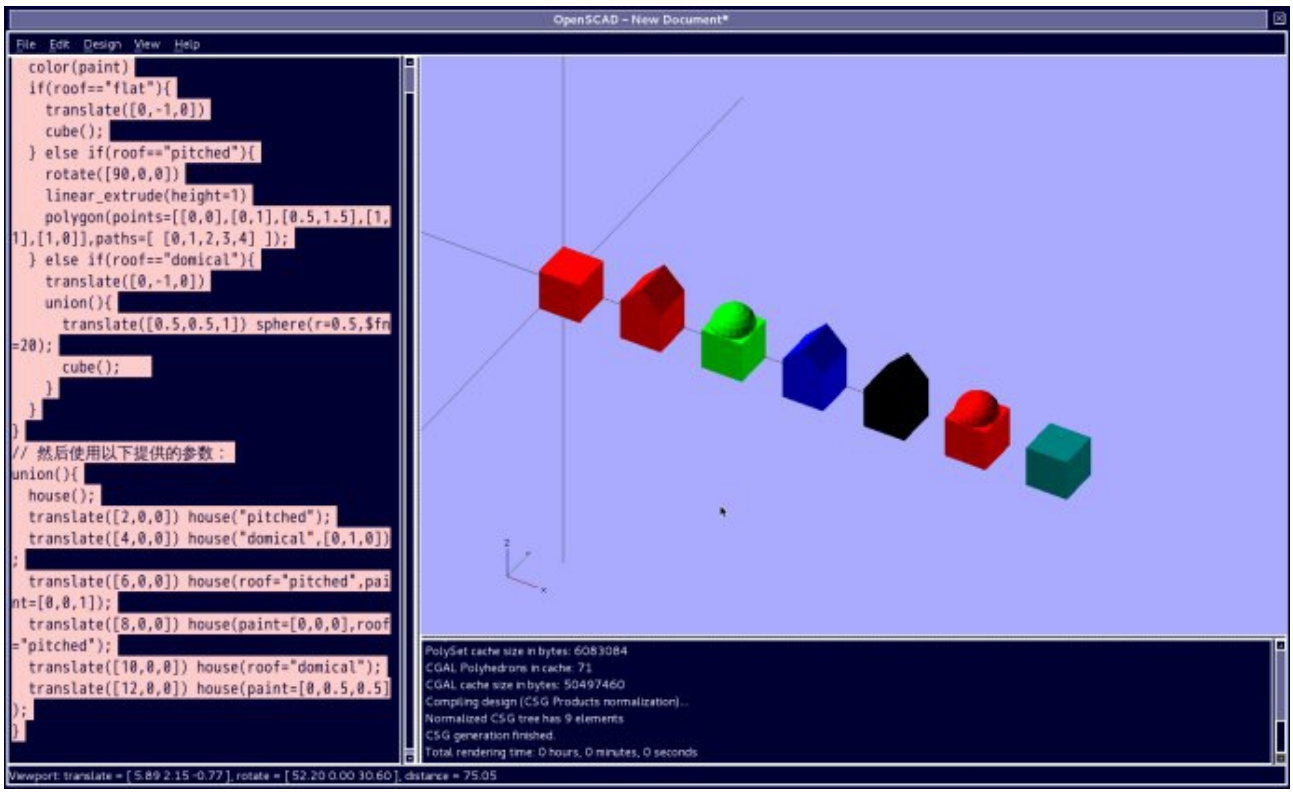
```
module elongate() {
  for (i = [0 : $children-1])
    scale([10 , 1, 1 ]) child(i);
}

elongate() { sphere(30); cube([10,10,10]); cylinder(r=10,h=50); }
```



P-4.12

一个可以配置默认数值到参数：

```
module house(roof="flat",paint=[1,0,0]){
  color(paint)
  if(roof=="flat"){
    translate([0,-1,0])
    cube();
  } else if(roof=="pitched"){
    rotate([90,0,0])
    linear_extrude(height=1)
    polygon(points=[[0,0],[0,1],[0.5,1.5],[1,1],[1,0]],paths=[ [0,1,2,3,4] ]);
  } else if(roof=="domical"){
    translate([0,-1,0])
    union(){
      translate([0.5,0.5,1]) sphere(r=0.5,$fn=20);
      cube();
    }
  }
}
// 然后使用以下提供的参数：
union(){
  house();
  translate([2,0,0]) house("pitched");
  translate([4,0,0]) house("domical",[0,1,0]);
  translate([6,0,0]) house(roof="pitched",paint=[0,0,1]);
  translate([8,0,0]) house(paint=[0,0,0],roof="pitched");
  translate([10,0,0]) house(roof="domical");
  translate([12,0,0]) house(paint=[0,0.5,0.5]);
}
```


OpenSCAD - New Document*
File Edit Design View Help
color(paint)
if(roof=="flat"){
translate([0,-1,0])
cube();
} else if(roof=="pitched"){
rotate([90,0,0])
linear_extrude(height=1)
polygon(points=[[0,0],[0,1],[0.5,1.5],[1,
1],[1,0]],paths=[[0,1,2,3,4]]);
} else if(roof=="domical"){
translate([0,-1,0])
union(){
translate([0.5,0.5,1]) sphere(r=0.5,$fn
=20);
cube();
}
}
}
// 然后使用以下提供的参数：
union(){
house();
translate([2,0,0]) house("pitched");
translate([4,0,0]) house("domical",[0,1,0])
;
translate([6,0,0]) house(roof="pitched",pai
nt=[0,0,1]);
translate([8,0,0]) house(paint=[0,0,0],roof
="pitched");
translate([10,0,0]) house(roof="domical");
translate([12,0,0]) house(paint=[0,0.5,0.5]
);
}
PolySet cache size in bytes: 6083084
CGAL Polyhedrons in cache: 71
CGAL cache size in bytes: 50497460
Compiling design (CSG Products normalization)...
Normalized CSG tree has 9 elements
CSG generation finished.
Total rendering time: 0 hours, 0 minutes, 0 seconds
Viewport: translate = [5.89 2.15 -0.77], rotate = [52.20 0.00 30.60], distance = 75.05

P-4.13

第三部分(1~7，13~15) 概述,语法句法表达式，数学，运动仿真

1.1 概述
1.1.1 标注
1.1.2 变量
1.1.2.1 矢量
1.1.2.1.1 选项
1.1.2.2 字符串
1.1.2.3 变量的编译时设置，不是运行时
1.1.2.3.1 例外1
1.1.2.3.2 例外2
1.1.3 获得输入
2 条件和循环函数
2.1 循环
2.2 交叉For循环
2.3 If 阐述
2.4 Assign阐述

3 数学运算符
3.1 数量的算术运算符
3.2 关系运算符
3.3逻辑运算符
3.4 条件运算符
3.5 矢量数字运算符
3.6 矢量运算符
3.7 矢量 点-积 运算符
3.8 矩阵乘法

4 数学函数
5 三角函数

6 其他的数学函数

7 字符串函数
7.1 str
7.2 Also See search()

1,General 概述
1.1.1, Comments 标注 (注释说明)

OpenSCAD使用一个编程语言建立模型然后显示在屏幕上。标注是一种途径使其离开注释在代码内（不管是你自己或者未来的程序员）描述如何让这些代码工作，或者它做什么。标注是不被编译器评估的，所以不需要使用自己明白的代码。

OpenSCAD 使用 C++ 样式的标注：

```
// This is a comment
myvar = 10; // The rest of the line is a comment
```

双反斜杠是表示标注本行，而反斜杠加型号表示标注一段在范围内的内容。

1.1.2, Variables 变量

Variables in OpenSCAD are simply a name followed by an assignment via an expression
变量在 OpenSCAD 是一个简单的名字跟随一个任务通过一个表达式。

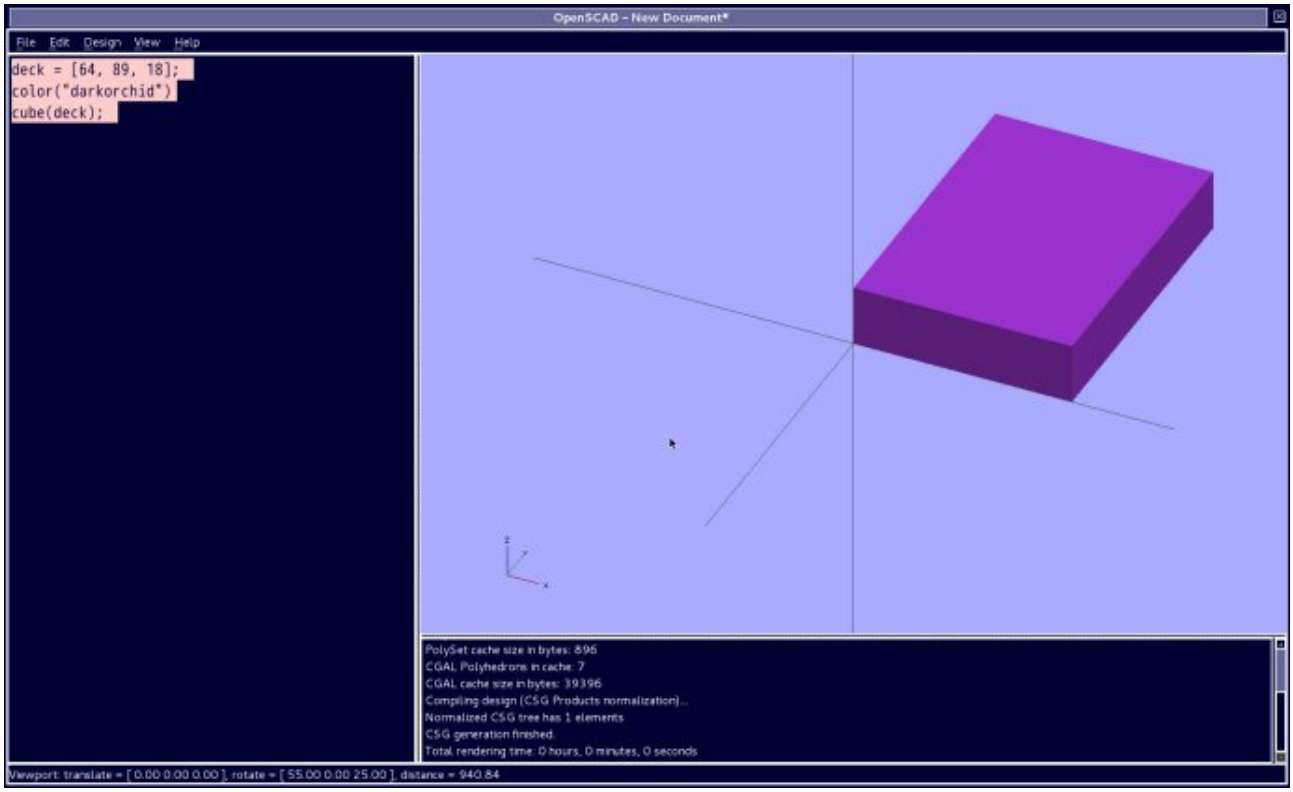
Example:
实例：

```
 myvar = 5 + 4;
```

1.1.2.1, Vectors 数组（矢量）

变量 可以组合一起到一个 数组中使用 中括号。数组是非常有用的当处理 X，Y，Z 坐标和尺寸的时候。
实例：

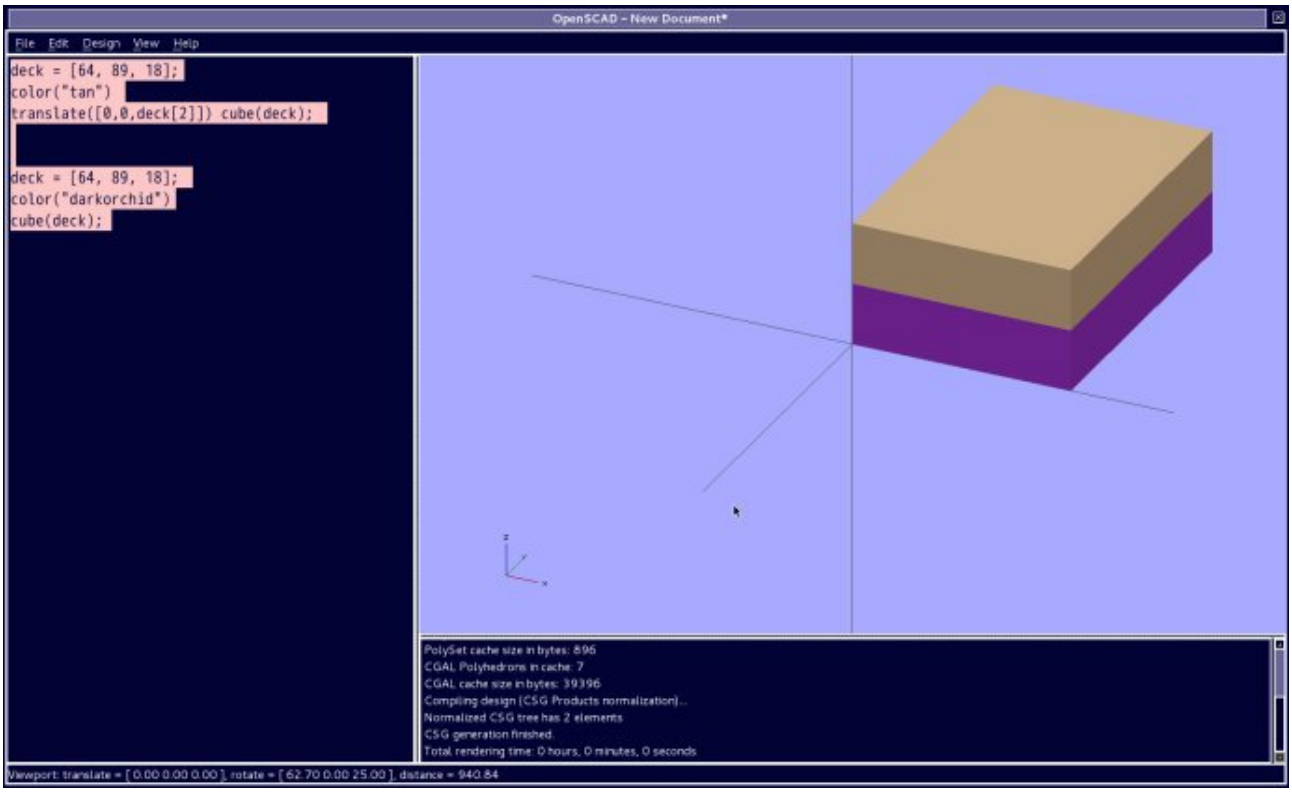
```
deck = [64, 89, 18];
color("darkorchid")
cube(deck);
```



Output A cube with the sizes: X = 64, Y = 89, Z = 18.
输出一个立方体的尺寸是：X = 64, Y = 89, Z = 18.

Selection 选择
You can also refer to individual values in a vector with vector[number].
你也可以选择单独的的数值在一个数组用使用数组中的某一个分组，vector[number].

实例：
```
deck = [64, 89, 18];
translate([0,0,deck[2]]) cube(deck);
```



图片效果是分别显示0,1,2 deck中的第一个分组数值，第二个，第三个在相应的轴线上。

输出是同样的立方体和上一个例子一样，但是在Z轴向上升了18mm.
（说明：为什么用deck[2]，目的是为了Z方向移动deck函数的第三个分组的数字。如果是deck[0]，那么Z轴向就会移动64，如果是deck[1],那么Z轴向就会移动89）。

1.1.2.2, Strings 字符串

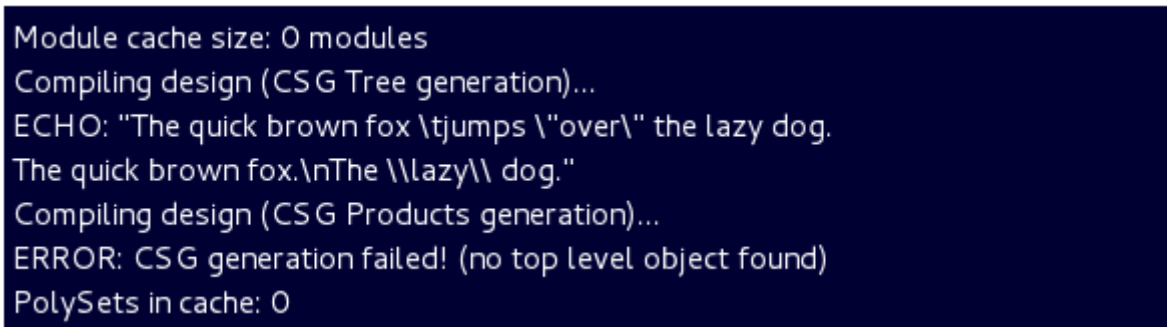
外在的双引号或者是反斜杠需要忽视掉（\" 和 \\ 分别的）。其他的忽视特殊符号是新行（\n），tab(\t) 和返回（\r).

注意：这个功能是从新版的OpenSCAD-2011.04. 你可以升级旧版本的文件使用sed命令.在linux系统的终端中运行命令：sed 's/\\/\\\\/' non-escaped.scad > escaped.scad
实例：

```
echo("The quick brown fox \tjumps \"over\" the lazy dog.\rThe quick brown
fox.\nThe \\lazy\\ dog.");
```

输出显示:

```
ECHO: "The quick brown fox jumps "over" the lazy dog.
The quick brown fox.
The \lazy\ dog."
```



（我这个系统不认\t，\n,不知怎么回事）。

1.1.2.3，变量是在编译时间设置的，不是运行时。

注解： 因为 OpenSCAD 计算它的变量数值在编译时，而不是运行时，所以最后的变量任务将会应用到任何地方的变量是使用的。它也许会帮助想象他们就像是奔腾的常量更胜于变量。

实例：

```
// The value of 'a' reflects only the last set value
a = 0;
echo(a);
a = 5;
echo(a);
```

输出显示：

```
ECHO: 5
ECHO: 5
```

```
Module cache size: 0 modules
Compiling design (CSG Tree generation)...
ECHO: 5
ECHO: 5
Compiling design (CSG Products generation)...
ERROR: CSG generation failed! (no top level object found)
PolySets in cache: 0
```

这个意思也是你不可以再分配一个变量在一个”if”内的板块。
实例：

```
a=0;
if (a==0)
{
a=1; // <- this line will generate an error.
}
```

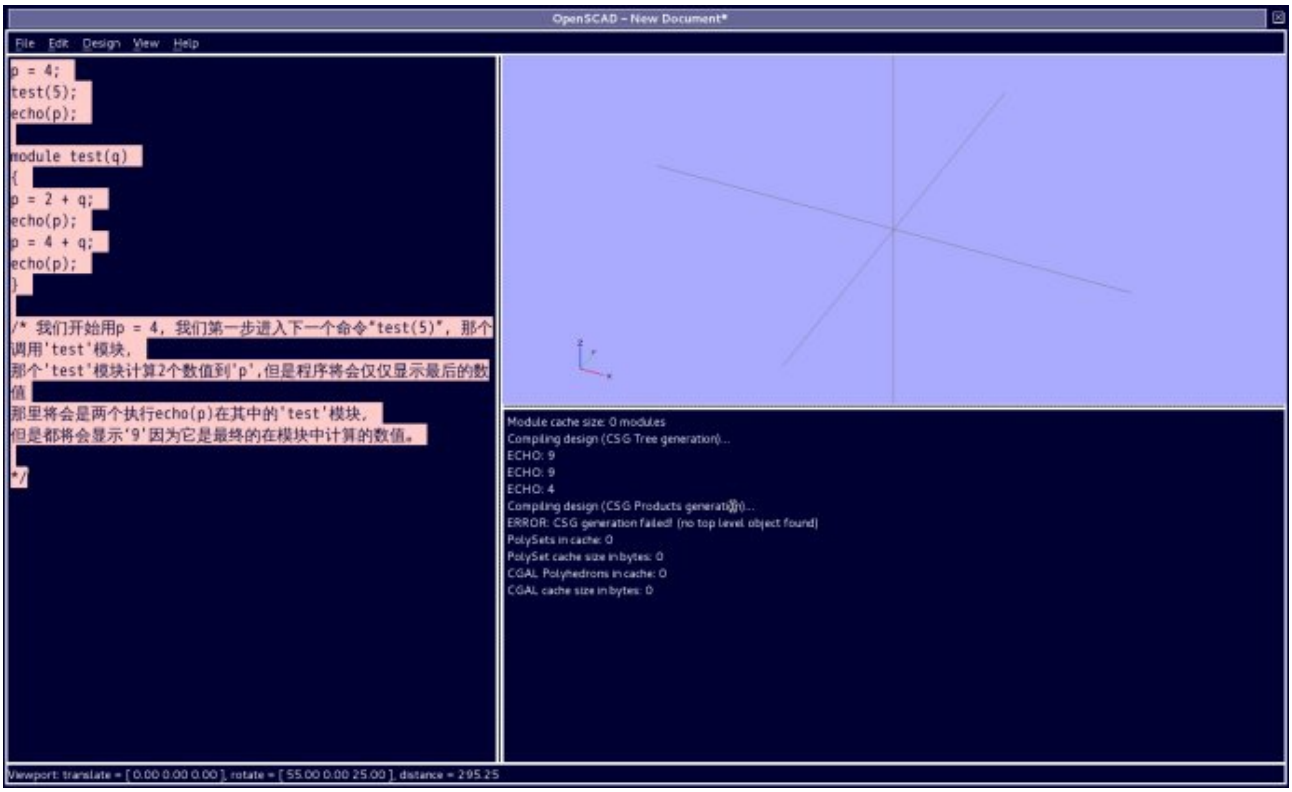
```
Parser error in line 4: syntax error

ERROR: Compilation failed!
```

输出编译错误

这个行为无论如何是涉及到不管是根或者特殊的召唤一个模块，意思你可以重新定义一个变量使用一个模块，不影响它的外部的数值。尽管，所有的例子包含召唤将会对待描述关于随后的设置的数值成为普遍的使用。

实例：

```
p = 4;
test(5);
echo(p);
p = 6;
test(8);
echo(p);
module test(q)
{
p = 2 + q;
echo(p);
p = 4 + q;
echo(p);
}
```



这个现象始终是一种顺时针感觉，它允许你做一些有趣的事情；例如，如果你设置你的共享库文件得到一个默认的数值定义一个变量在根级别，当你包含文件（引用文件）到你自己的代码，你可以重新定义或者放弃这些常量，使用简单的赋值到一个新的数值给他们。

参见赋值到更多的醒目的范围更改数值。

1.1.3，Getting input 获得输入

现在我们有变量，它可以是完美的能力得到一个输入将他们从代码代替设置的数值。那里有一些函数到读取数据从 DXF 文件，或者你可以设置一个变量使用 -D 切换到命令行。

1.1.3.1，获取一个点从一个图纸中。

获得一个点是比较使用的从读取一个原始的点在一个二维预览在一个技术图纸中。函数 dxf_cross 将会读取交叉的两个直线在一个涂层上你可以配置和返回交叉点。这个意识是点必须给出两条直线在 DXF 文件，而不是一个点存在。
实例：

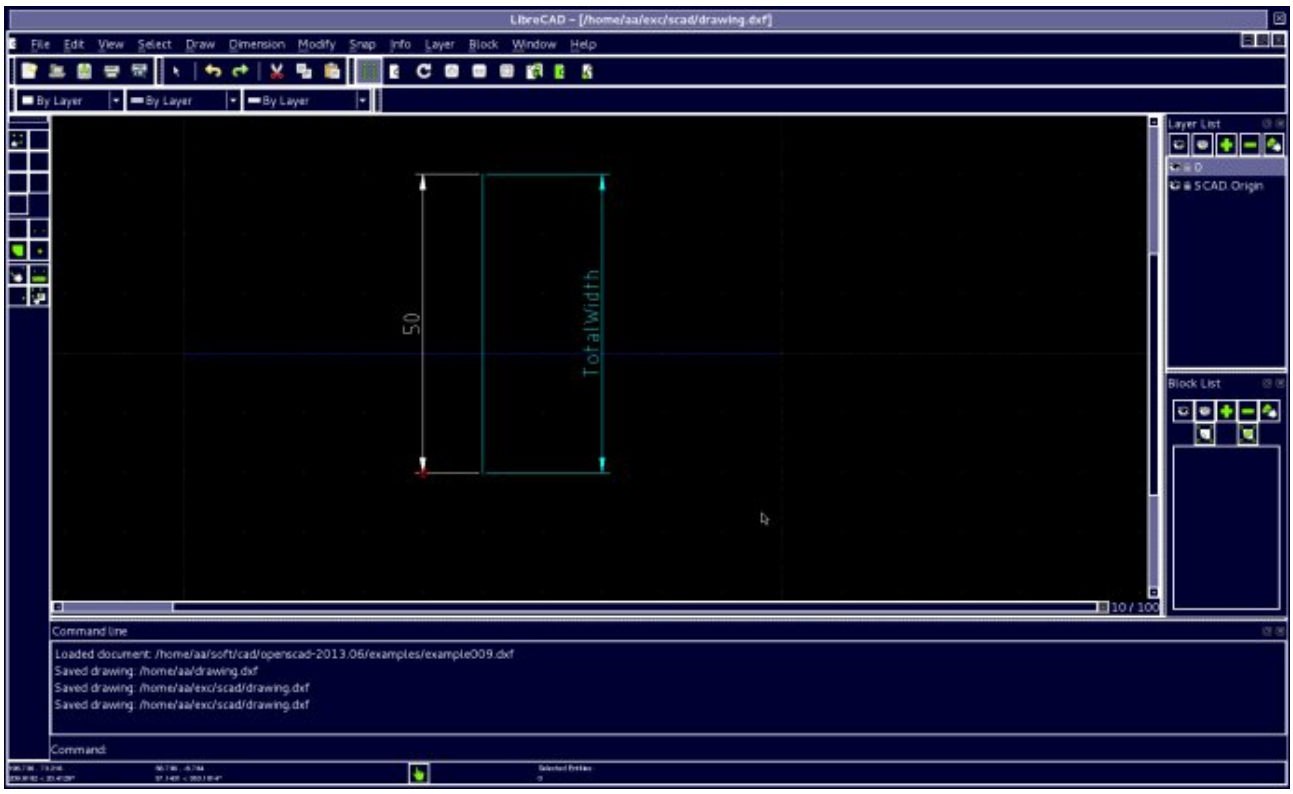
```
OriginPoint = dxf_cross(file="drawing.dxf", layer="SCAD.Origin",
origin=[0, 0], scale=1);
```

1.1.3.2，获取一个测量数值

你可以读取一个测量从一个技术图纸上。这是非常有用的从读取一个旋转角度，一个挤出的高度，或者介于零件之间的空间距离。在图纸中，创建一个测量但是不显示测量的数值，但要有一个标识符。 从而读取其数值，你配置这个标识符从你的描述中：
实例：

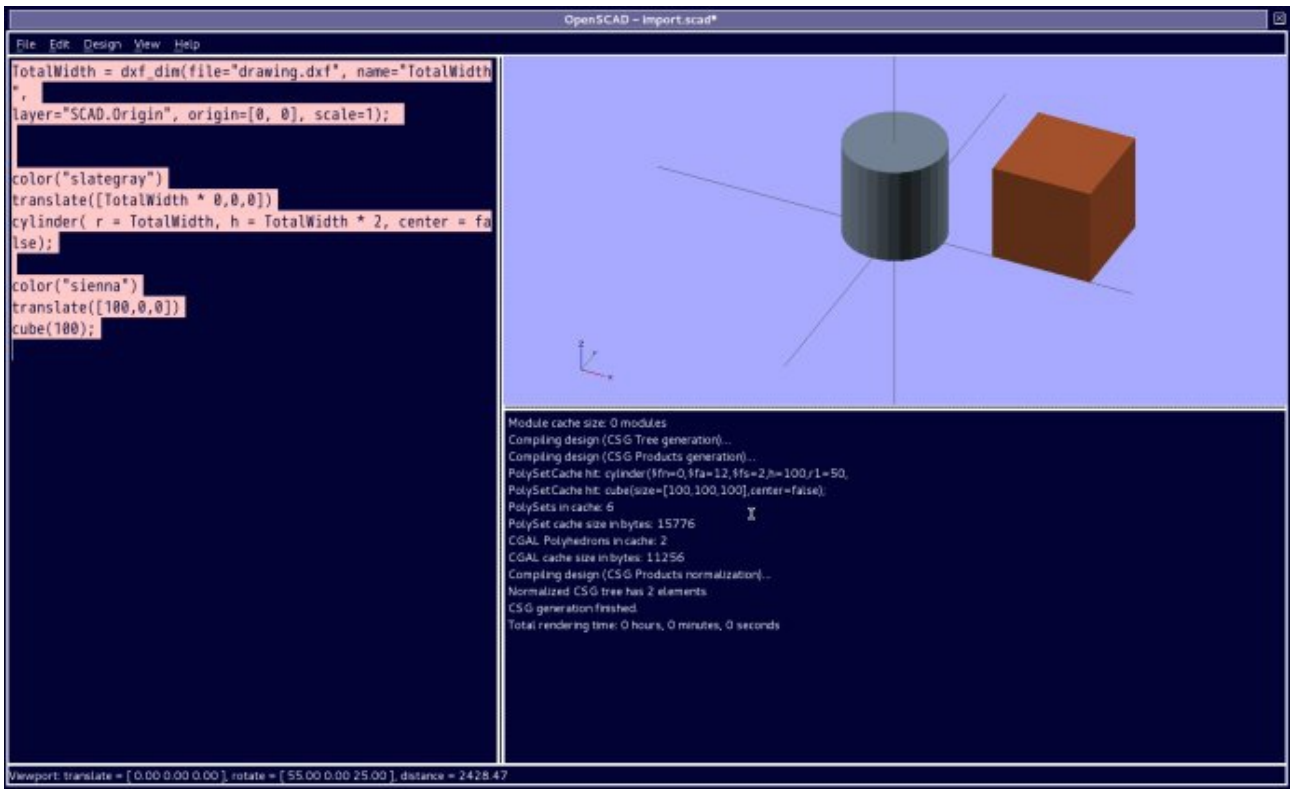
```
TotalWidth = dxf_dim(file="drawing.dxf", name="TotalWidth",
layer="SCAD.Origin", origin=[0, 0], scale=1);
color("slategray")
translate([TotalWidth * 0,0,0])
cylinder( r = TotalWidth, h = TotalWidth * 2, center = false);
color("sienna")
translate([100,0,0])
cube(100);
```

这里，我先在 librecad 中创建一个文件，命名为 drawing.dxf,必须放在和 scad 文件同一个文件夹，首先你必须把你的 scad 文件保存到 一个指定的文件夹，默认的是/home/user（用户名）/,然后利用 librecad 软件创建一个新的涂层，涂层的名字是 SCAD.Origin， 然后，在这个涂层下建立测量，模型可以在其他涂层中画，但是测量必须在这个涂层下，并且把测量的数值更改成'TotalWidth'的标识 符，librecad 命令从 modify->text_edit 选项，然后选择测量，然后把数值更改成标识符，用于 OpenSCAD 识别。然后你就 可以在 OpenSCAD 中继续创建模型，并且将标识符引入模型，标识符代表 dxf 中的测量的数值。我这模型中的数值是 50，我分别用圆柱体使用标识符和立 方体使用数字对比模型。



这个图形是 librecad 的 dxf 图纸模型截图。

图形是 openscad 使用 drawing.dxf 中的'TotalWidth'标识符后，建立的和数字对比的模型。
动画效果，改变 X 轴向的位置和中心的逻辑是否。



作为一个漂亮的实例使用所有的测量，详细参见 Example009 (file-> example->example009)和在 OpenSCAD 网站主页上的那个 PC 风扇模型。

1.2 条件和循环函数

1.2.1 循环
1.2.2 交叉 For 循环
1.2.3 If 阐述
1.2.4 Assign 阐述

1.2.1, For 循环

循环根据某个数值在一个数组中或者在一个范围。

数组版本（句法描述）：for(变量=<数组>) <运行做一些事情>-<变量> 是指定每一个成功的数值在数组中。

范围版本（句法描述）：for(变量=<范围>) < 运行做一些事情>

范围：[<开始>:<结束>]-循环包含从开始到结束。同样的工作如果 if<结束>是小于<开始>.

范围：[<开始>:<增量><结束>]-循环从开始到结束并使用给出的增量累计。增量可以是一个分数，几分之几。
注解：增量是给出的就像是一个绝对数值并且不能够是负数。如果<结束>是小于<开始>增量必须保持不变。

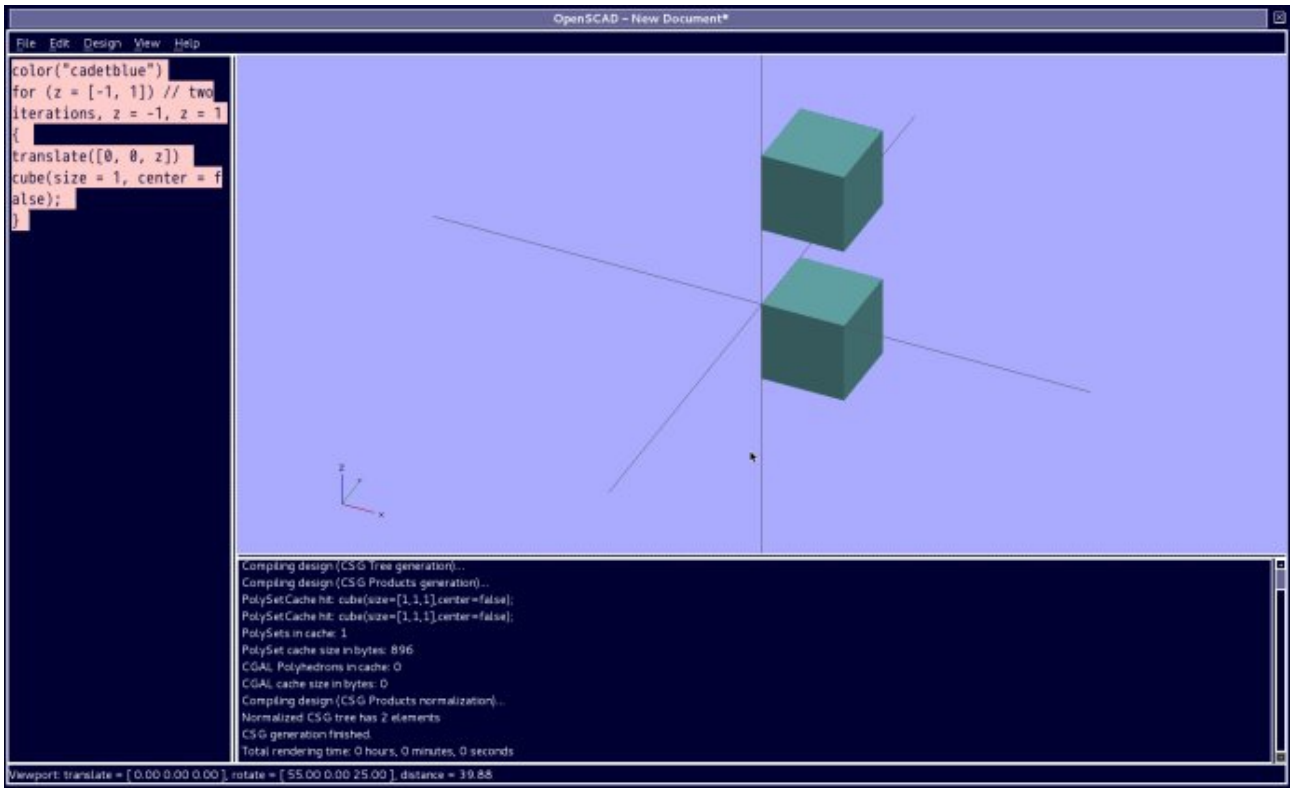
注意：如果增量不是一个偶数的除数从<开始>到<结束>，循环数值在最后的循环中将会是<结束>-（<结束>-<开始> mod<增量>）.

嵌套式循环： for(变量1 = <范围或者数组>， 变量2 = <范围和数组>) <运行做一些事情，使用全部变量>

for 循环可以被嵌套，就像是一个常规的程序。一个速记是每个循环可以给出同样的阐述。

实例1-循环通过一个数组：

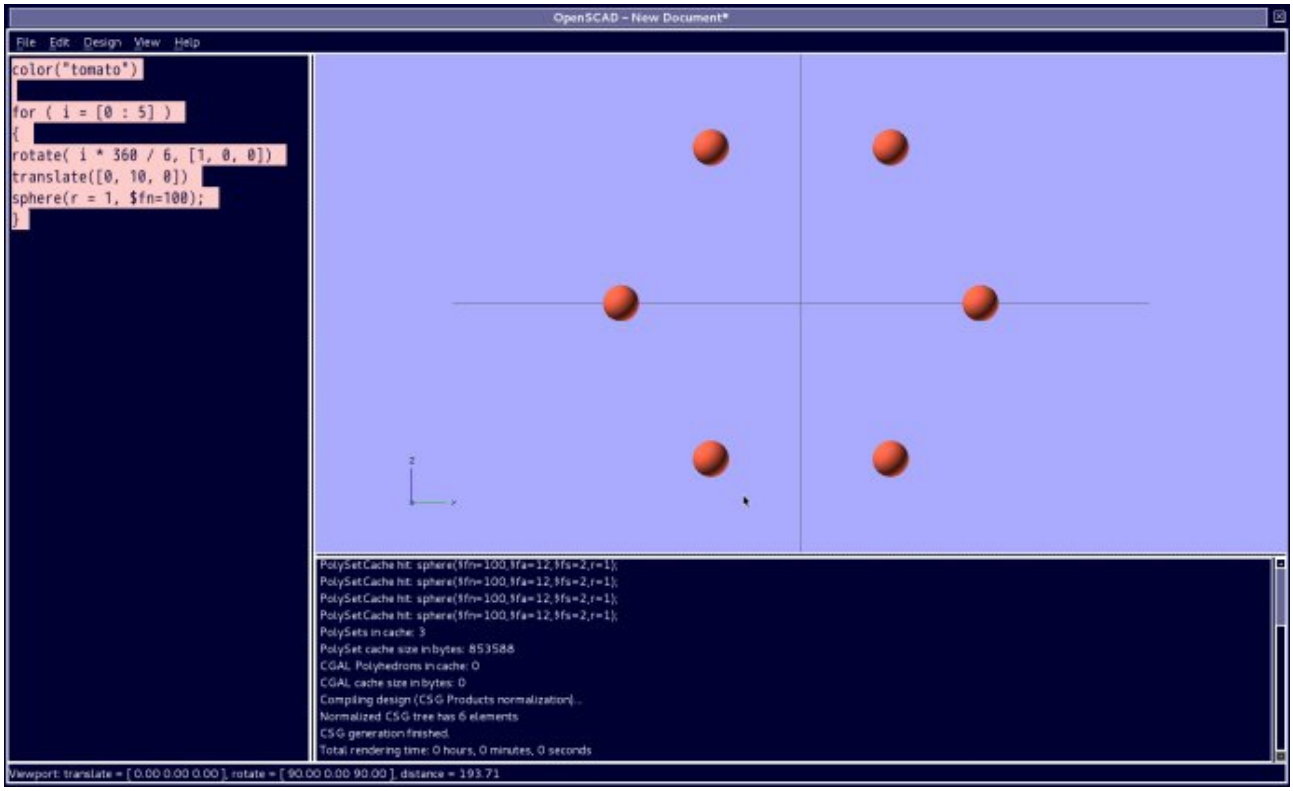
```
color("cadetblue")
for (z = [-1, 1]) // two iterations, z = -1, z = 1
{
translate([0, 0, z])
cube(size = 1, center = false);
}
```



OpenSCAD 循环通过一个数组

实例 2a-循环通过一个范围：

```
color("tomato")
for ( i = [0 : 5] )
{
rotate( i * 360 / 6, [1, 0, 0])
translate([0, 10, 0])
sphere(r = 1, $fn=100);
}
```



解释：旋转的角度，因为默认的增量是正整数，从 1 到 5，然后，如果选择 i*360/6，结果是 60 度，而 360/5 是 72 度，和增量一样，所以等分，如果是 360/8，那么就会是 45 度，会每间隔 45 度有一个模型，而有一个多余的，在负方向，所以可能是系统默认的，0，或者是起始吧！

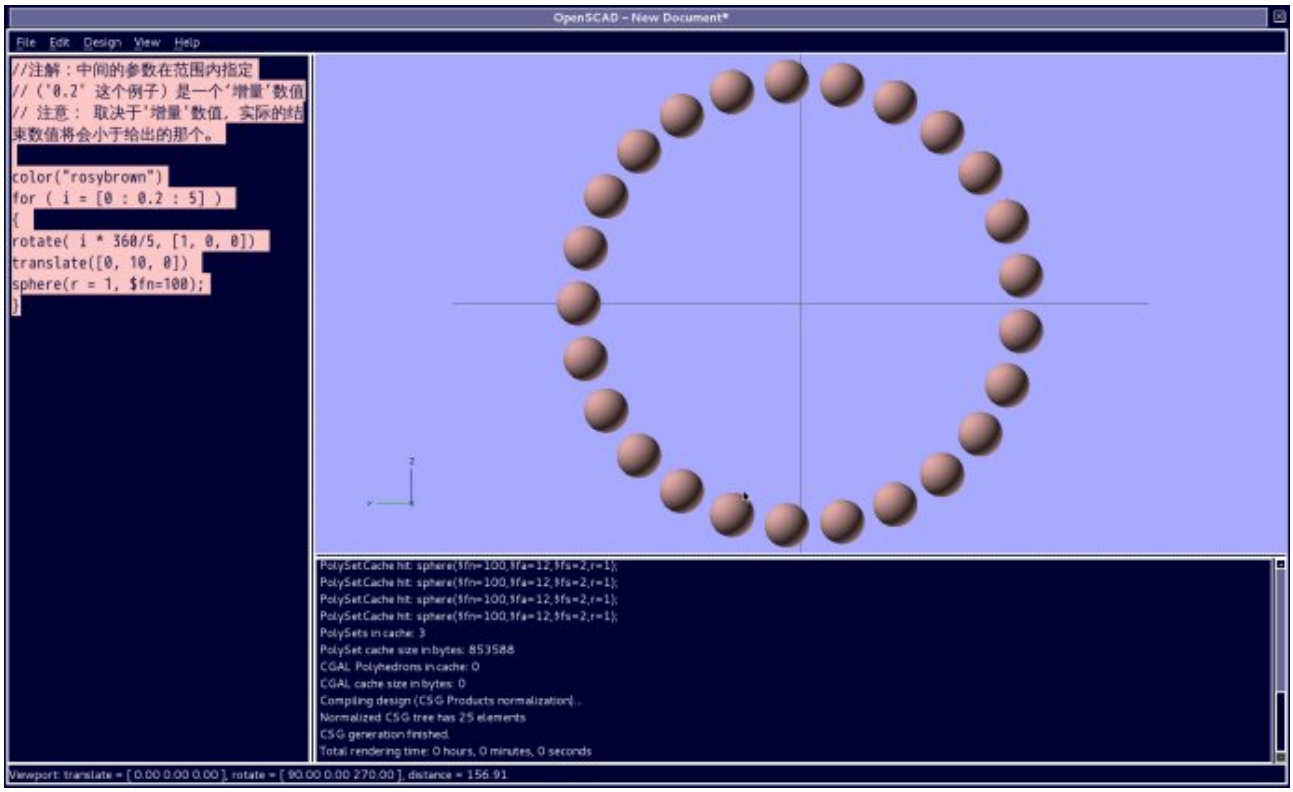
OpenSCAD 循环通过一个范围

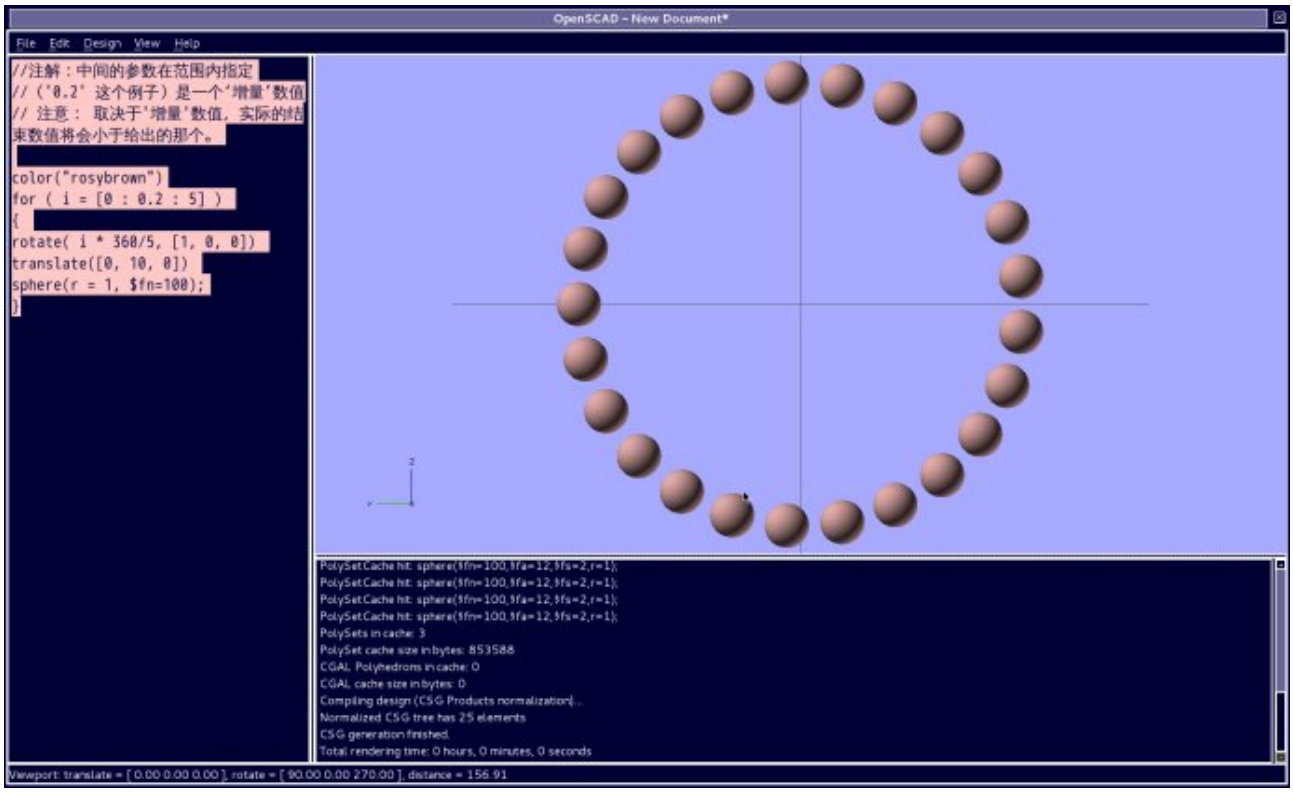
实例 2b-循环通过一个范围配置一个增量：

```
//注解：中间的参数在范围内指定
// ('0.2' 这个例子）是一个'增量'数值
// 注意： 取决于'增量'数值，实际的结束数值将会小于给出的那个。

color("rosybrown")
```

```
for ( i = [0 : 0.2 : 5] )
{
rotate( i * 360 / 6, [1, 0, 0])
translate([0, 10, 0])
sphere(r = 1, $fn=100);
}
```

解释：为什么没有成一个完整的环形，需要计算，因为增量是 0.2，从 0 到 5 一共增量 25 次，而系统模型给出的是 i*360/6,角度就是 60 度，而系统却 是呈现出紧密的排序，如果是 360/8，那么排序就会更紧密，而完全平均排开是（360/25）*5，得出是 72 度，那么 360/5 正好得到 72 度，或者 更简单的方法就是增量的正数是多少，角度旋转就是 360/多少，这样就可以等分了。

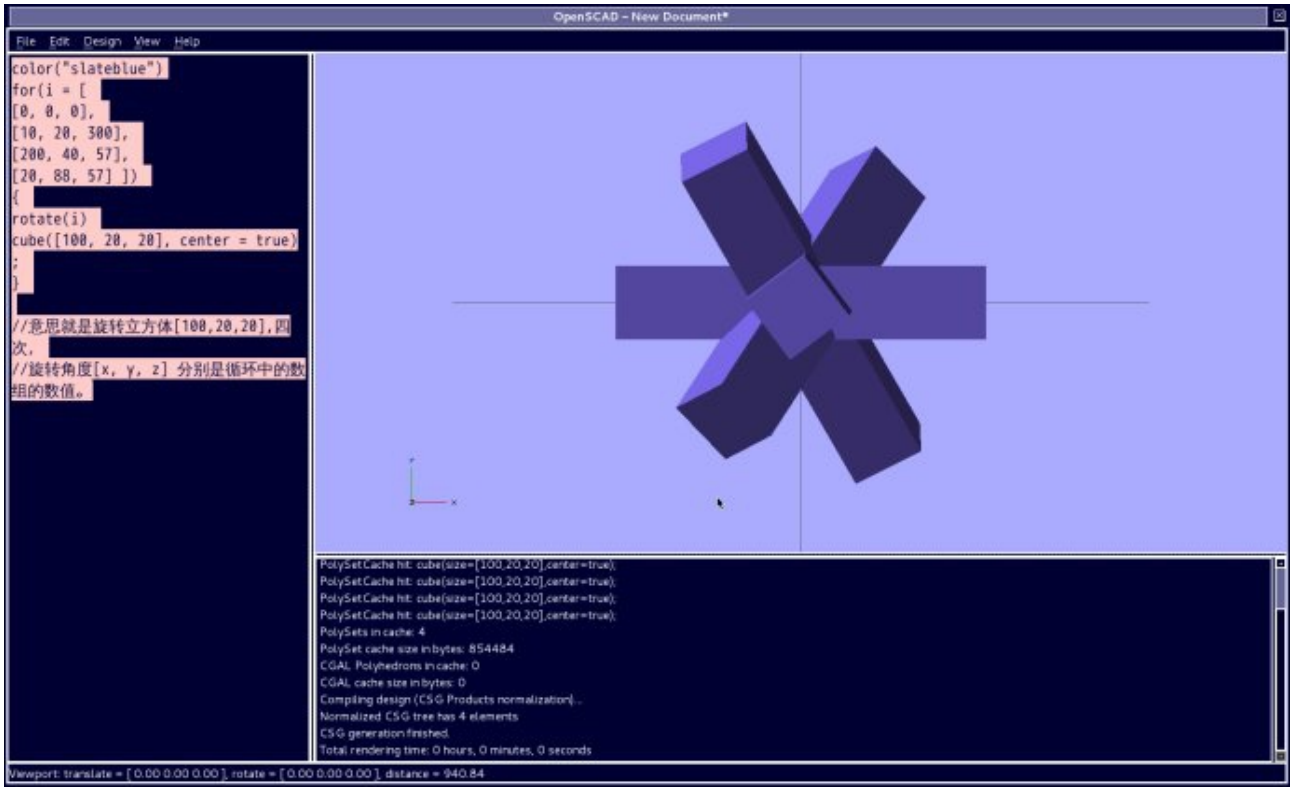




实例 3-循环通过一个数组的一个数组（方旋转）：

```
color("slateblue")
for(i = [
[0, 0, 0],
[10, 20, 300],
[200, 40, 57],
[20, 88, 57] ])
{
rotate(i)
cube([100, 20, 20], center = true);
}
```

//意思就是旋转立方体[100,20,20],四次，
//旋转角度[x, y, z] 分别是循环中的数组的数值。

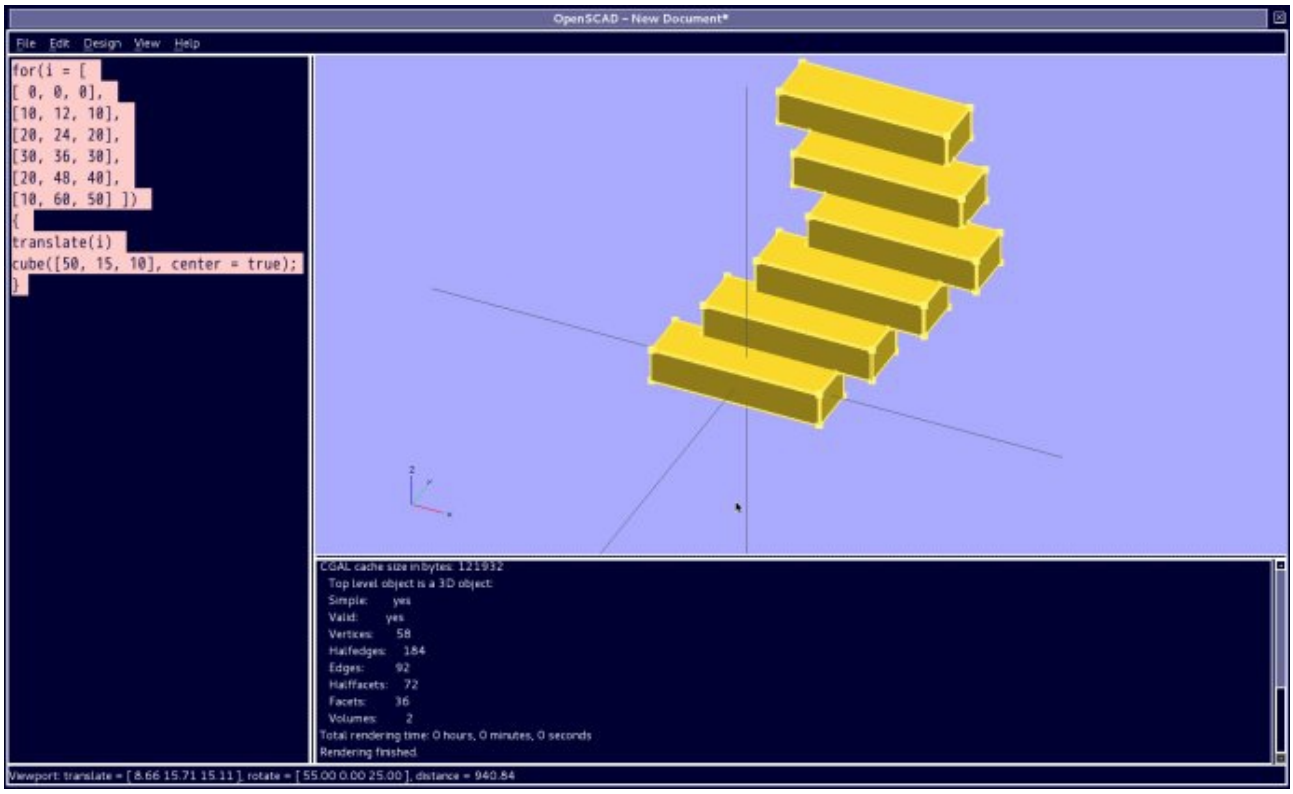


OpenSCAD for 循环（旋转）

Usage example 4 - iteration over a vector of vectors (translation):

实例 4-循环通过一个数组的数组（位移或者说置位）：

```
for(i = [
[ 0, 0, 0],
[10, 12, 10],
[20, 24, 20],
[30, 36, 30],
[20, 48, 40],
[10, 60, 50] ])
{
translate(i)
cube([50, 15, 10], center = true);
}
```



这个就容易理解多了，循环每一个不同坐标位置的数组，立方体模型尺寸[50,15,10].

OpenSCAD for 循环（位移）

Nested loop example
嵌套式循环实例：

```
color("palevioletred")
for (xpos=[0:3], ypos = [2,4,6])

// do twelve iterations, using each xpos with each ypos
translate([xpos*ypos, ypos, 0])
cube([0.5, 0.5, 0.5]);

color("blueviolet")
translate([0,1.5,0])
cube([2,0.5,0.5]);
color("fuchsia")
translate([0,3.5,0])
cube([4,0.5,0.5]);
color("cyan")
translate([0,5.5,0])
cube([6,0.5,0.5]);
```

注解：这里，循环数组，xpos[0:3]， 这个因为有了：分号，表示是循环的次数，从0开始，到末尾的数值结束循环，而ypos[2,4,6]，仅仅是代表一组数值的数组，引用到循环内而已。置位translate函数定义了坐标的位置，x，y,z，x = xpos * ypos，这里就是一个循环数组乘以一组数组，是一个变量，随着循环的结尾的数值增加其次数，而中间的距离确实2,4，6分别分开的，数值的坐标结果就像是

```
y 6 6 6 6
y 4 4 4 4
y 2 2 2 2
   x x x x
```

交叉的For 循环

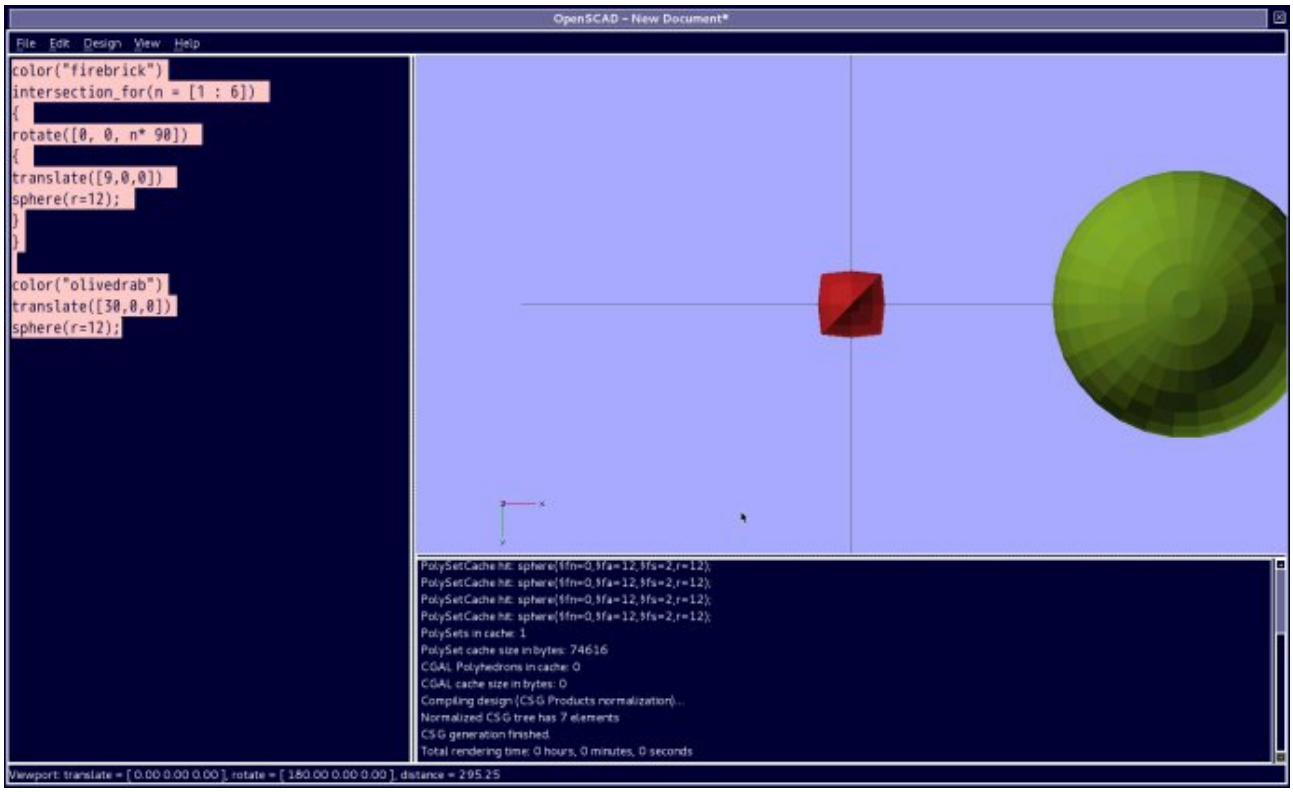
循环通过数值在一个数组中或者范围内并且使用一个交叉的内容。
注解： intersection_for()函数是一个工作围绕的因为一个问题（因素）你不可以获得预期的结果使用一个组合的标准的for()函数或者intersection()阐述。

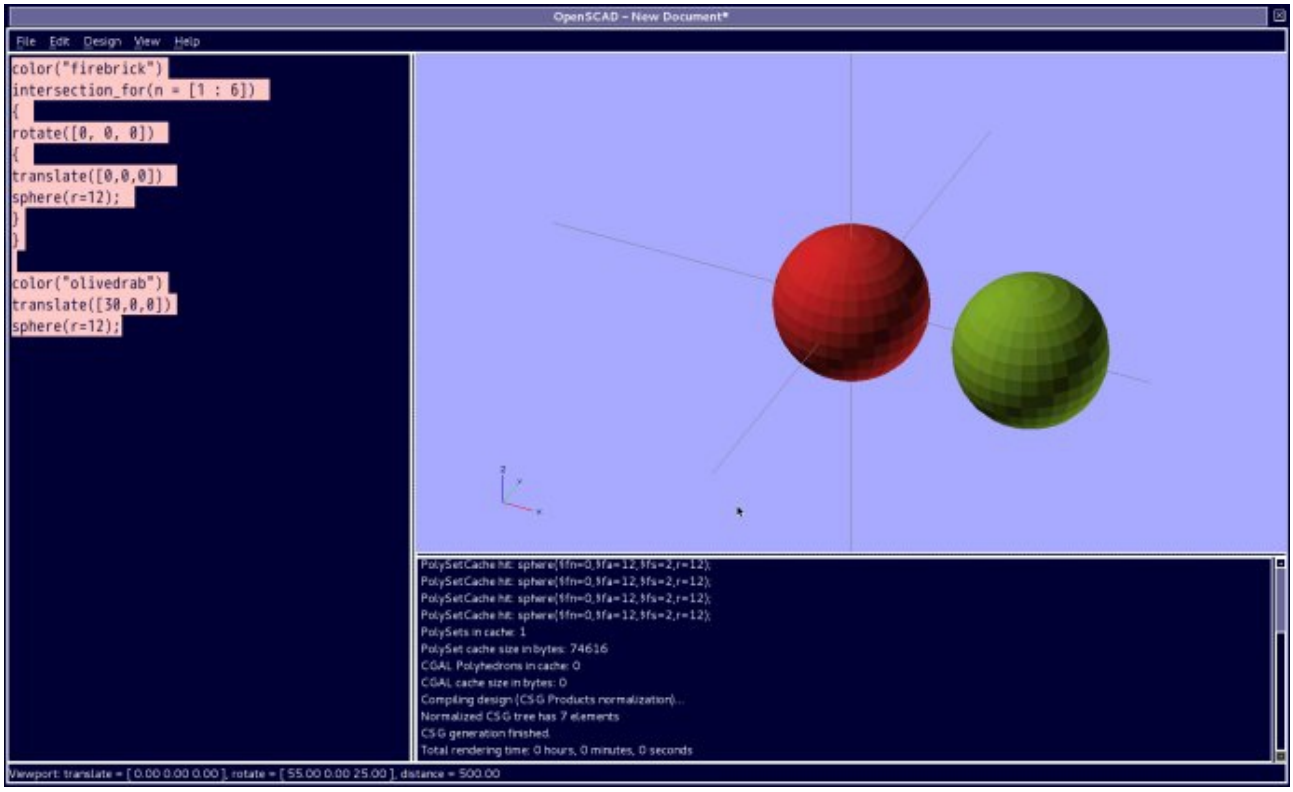
参数
<循环变量的名称>

变量的名称在 for 循环的内部使用。

实例 1 — 循环通过一个范围：

```
color("firebrick")
intersection_for(n = [1 : 6])
{
rotate([0, 0, n * 60])
{
translate([6,0,0])
sphere(r=12);
}
}

color("olivedrab")
translate([30,0,0])
sphere(r=12);
```





这里，我做了一个对比，以方便更好的理解， translate([6,0,0]),意思是将模型想 x 方向移动 6mm，然后围绕者坐标的中心，循环和旋转，（如果移动的数值大于半径 12,那么全部没有交叉，就不会产生模型了）rotate([0,0,n*60]),表示以 z 轴线为中线旋转 60 度，循环从 1 到 6，得出 6 个模型，（如果旋转中是 60，而不是 n*60,那么输出就是一个根据 z 轴向旋转而没有交叉的圆形）然后，将这 6 个模型的交叉部分保存，其余删除，所以这里的位置移动和旋转的角度都是影响其变化的变量。

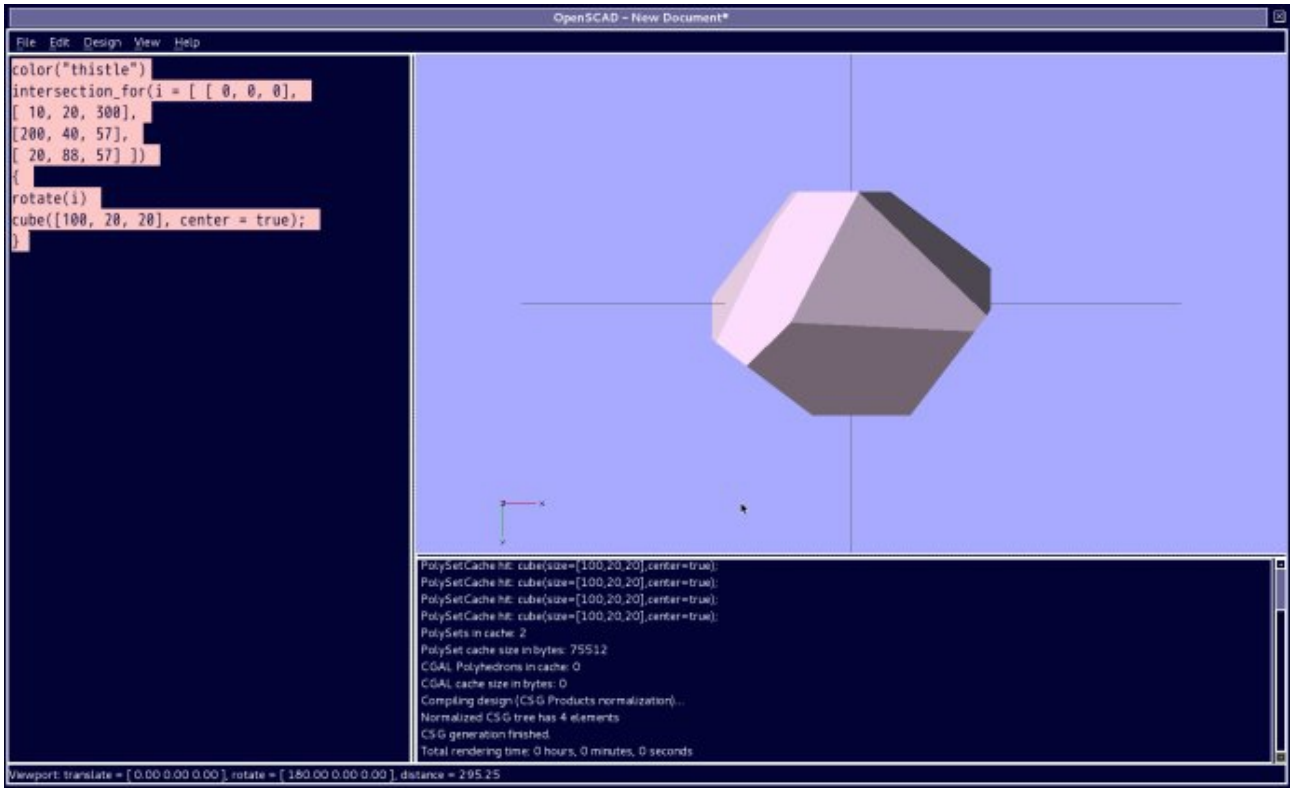
图片：OpenSCAD Intersection for

OpenSCAD 交集 for 循环
实例 2 — 循环通过一个数组-旋转：

```
color("thistle")
intersection_for(i = [
 [ 0, 0, 0],
[ 10, 20, 300],
[200, 40, 57],
[ 20, 88, 57] ])
{
rotate(i)
cube([100, 20, 20], center = true);
}
```

如果你理解了以上的实例的描述，那么这里实例就比较容易理解了，这里数组有 4 组，给出了的数值相当于 x,y,z 的旋转角度，表示将立方体（[100,20,20]），center=true 表示以坐标中心为基准参考，循环对 4 组不同的数值分别进行 xyz 角度的旋转，具体旋转的数是数组中填 充的用逗号间隔的数值。然后将所有的模型交叉（交集）的部分留下，其他的全部删除，就是一个新的模型。

参见图：



OpenSCAD 交集 for （旋转）

If 阐述
条件评估的一个分支。

参数
布尔代数表达式必须被应用到这个条件。

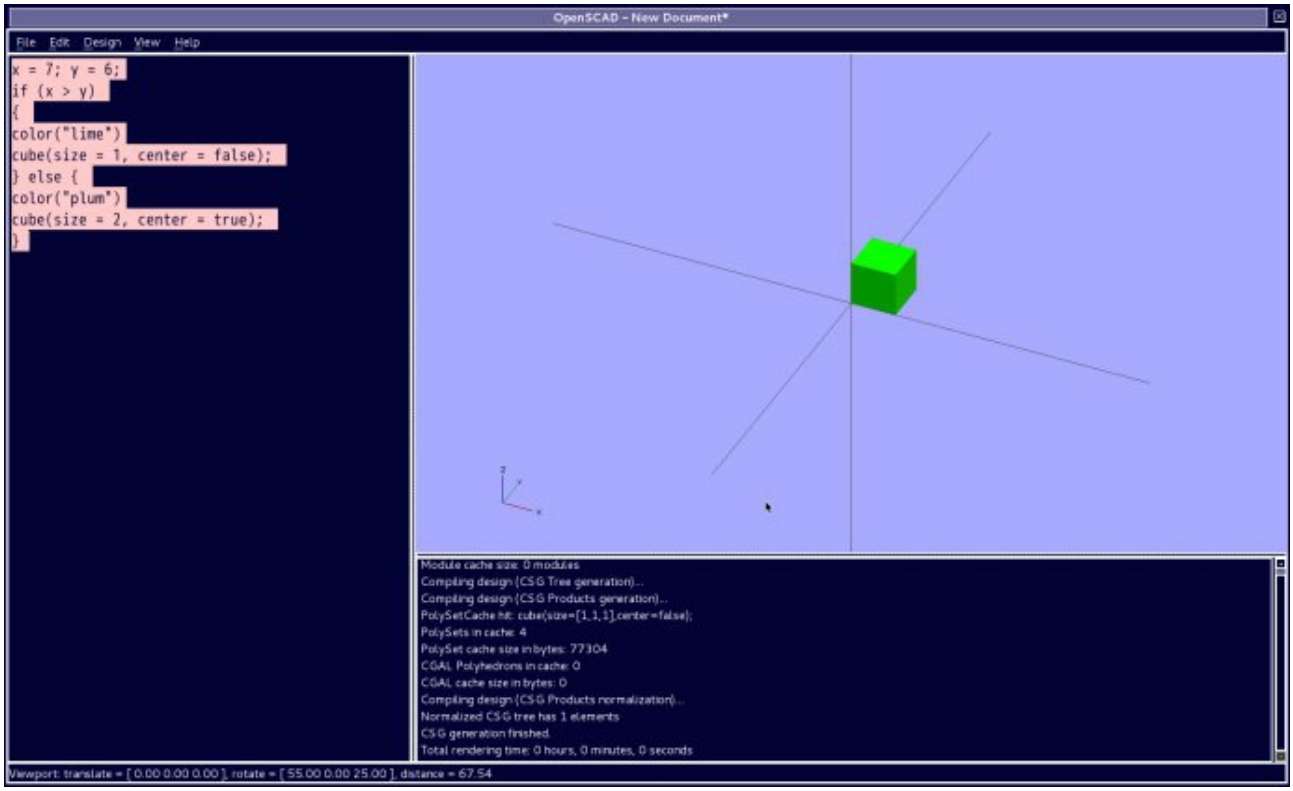
注解：
不要混淆表示符号'=' 和等于符号'==', 句法描述是 if(a=b) 运行一些程序函数 () ；
// 将会导致失败并且没有任何错误提示

实例：

```
x = 5; y = 6;
if (x > y)
{
color("lime")
cube(size = 1, center = false);
```

```
} else {
color("plum")
cube(size = 2, center = true);
}
```

这里我分别定义了 x=5， y=6， 和另一组程序 x=7,y=6,其他参数不变，我们可以看到系统的选择。



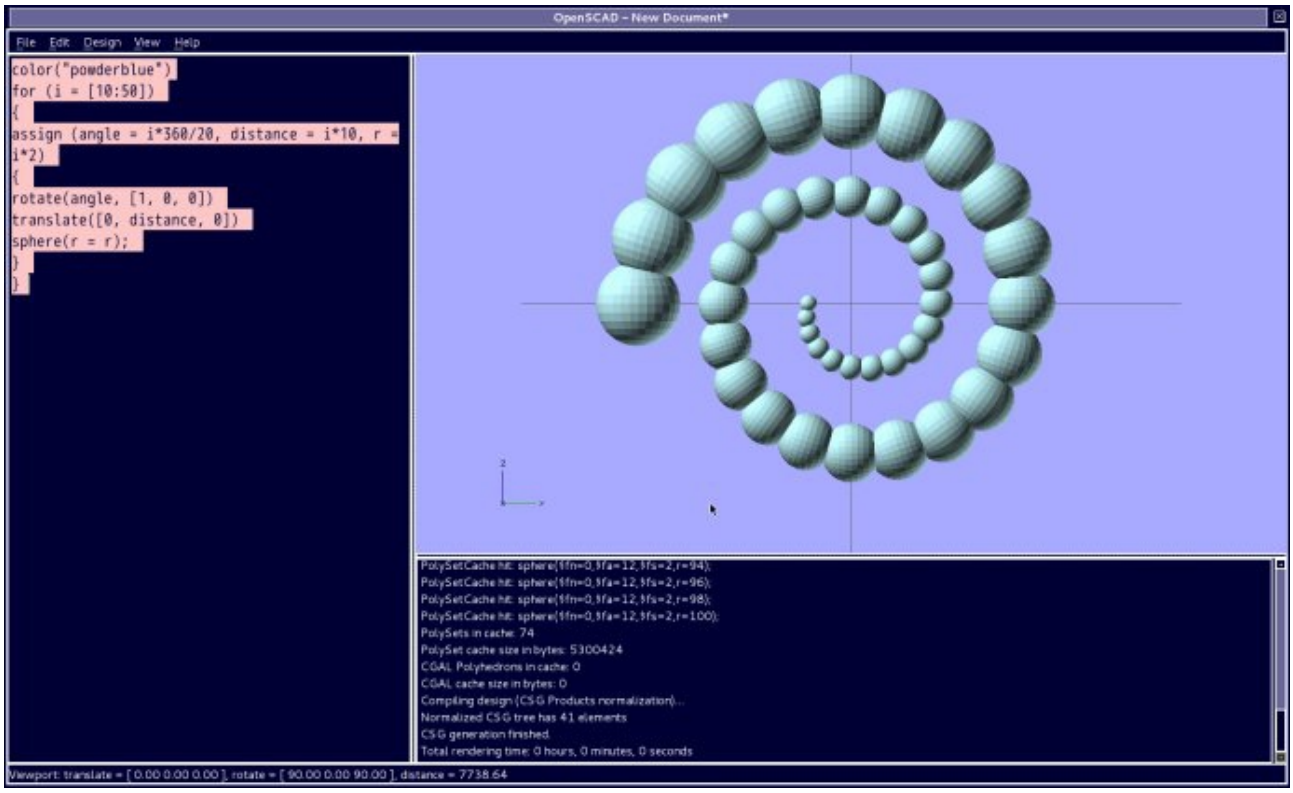
赋值的阐述

设置变量到一个新的数值用于一个分支（子树）。

参数
变量必须是赋值或者从新赋值的(re-)assigned.

实例：

```
color("powderblue")
for (i = [10:50])
{
assign (angle = i*360/20, distance = i*10, r = i*2)
{
rotate(angle, [1, 0, 0])
translate([0, distance, 0])
sphere(r = r);
}
}
```



这里的赋值有一点想 C 语言中的赋值，不过 C 中是直接赋值的，而这里多了一个 assign 函数和一个括弧，^_^，不管如何，非常好用阿！

这里，循环 for(i=[10:50],就是循环从 10 开始到 50 结束，共 40 个，包含 0 就是 41 次循环，angle=i*360/20，就是每个球形旋转 18 度，rotate(angle,[1,0,0]),旋转是围绕着 x 轴线进行的，而置位 translate([0,distance,0]),把置位也成了一个增量循环，distance=i*10,就是沿着 y 轴线每个球形移动

循环的相应参数*10 的位置，sphere(r=r)等于符号后边的 r 是赋值的相当于代数的字符，也是一个增量数值，r=i*2，就是球形的半径从增量的数值中每一次每一个增量都乘以 2 之后得到的结果是球体的半径。

我有个图形分别对不同的参数做了调整得出不同的结果。

OpenSCAD 中文教程——6.1

数学运算符

很多的内容是我个人的创建的模型，意在学习数学的函数应用到 CAD 模型中，利用 C 语言的编程模式和思维来计算一个复杂的机械模型，本想一章就可以带过数 学，没想到写着写着就多了，估计 2 章节应该够了，引用了大量的 C 语言的 math.h 的应用，希望可以对如何学习和应用 openSCAD 的数学计算模型方面 有所帮助，有一点像 CAS（计算机代数系统）或者 gnuplot，octave 之类数学软件了。

但是的确有这个功能，无论 openCSG（结构立体机械软件库），还是 CGAL（几乎是一个最强大的开源的数学和综合理科的软件库，史无前例的庞大），其 数学功底是相当深入的，所以学习 openSCAD 就像是用数学的算法来构建一个机械模型，虽然有时候计算机的速度的确跟不上，但是只有有好的方法，就可以 非常简便，简短，简洁的描述语言完成一个复杂的机械模型。

能够把这个数学章节学习透彻，那么你在计算机辅助制图(CAD)，计算机辅助加工(CAM)，和计算机辅助工程（CAE）的学习和应用，甚至到开发，都是终身受用的。

数学运算符——第一部分

纯量的运算符号

标量运算符使用数字就像是运作或者产生一个新的数字。

+ add 加法

- subtract 减法

* multiply 乘法

/ divide 除法

% modulo 余数

这里'-'可以使用就像是前缀运算符应用到一个负数中。

关系运算符

所有的关系预算服把一个数字就像是运算或者产生一个布尔代数数值。等于和不等于符号可以同样比较布尔代数数值。

< 小于

<= 小于等于

== 等于

!= 不等于

>= 大于等于

> 大于

逻辑运算符

所有的逻辑运算符把布尔代数就像是运算或者产生一个新的布尔代数数值。

&& Logical AND 逻辑'与'

|| Logical OR 逻辑'或'

! Logical NOT 逻辑'非'

条件运算符

这里？： 运算符可以是使用到条件的评估这个或者下一个表达式。工作就像?:运算符源于C语言家庭。

？： 条件运算符

案例给出：x>0?”pos”:”neg”,输出的结果是”pos”如果x大于0，否则”neg”.

向量-数字运算符 (应该有线性代数的基础最好)

向量-数字运算符把一个向量数组和一个数字就像是运算和产生一个新的向量数组。

*， 乘以所有的向量数组中的所有成员用数字

/，除以所有向量数组中的所有成员用数字

向量数组运算符

向量数组运算符把数组就像是运算和产生一个新的数组

+ add element-wise 各个成员相加

- subtract element-wise 各个成员相减

这里”-”可以是前缀运算符应用到成员的负数向量数组。

向量数组 点-积 运算符

向量数组 点-积 运算符把两个向量数组就像是运算和产生一个纯量。

*， 累加向量数组的成员的和

矩阵乘积

乘以一个矩阵使用一个向量数组，矩阵的向量和矩阵的向量

*，矩阵/向量数组 乘积

（关于以上的运算，单独的计算并应用到模型中，是比较耗费内存的所以，我们讲到如何引入数据模型，dat格式的文本数据模型，然后计算出一个模型后，再来完善如何使用这些数学运算符）。

数学函数

这里，希望多了解一些关于三角函数，对数指数等数学理论，最好是在C语言中编程过或者使用过一些

简单的科学计算器等，gnuplot，octave，sagemath 等一些数学软件，maple，mathematica，matlab（付费非开源数学软件），中有一些公式的应用，gsl 是一个不错的数学库，gnu 的项目，支持 C 和 C++，如果你学过 PASCAL 或者 FORTRAN，那么以上的话都可以略过。

abs

学绝对值函数。返回正的数值将一个标记的十进制数字或者小数点。

实例：

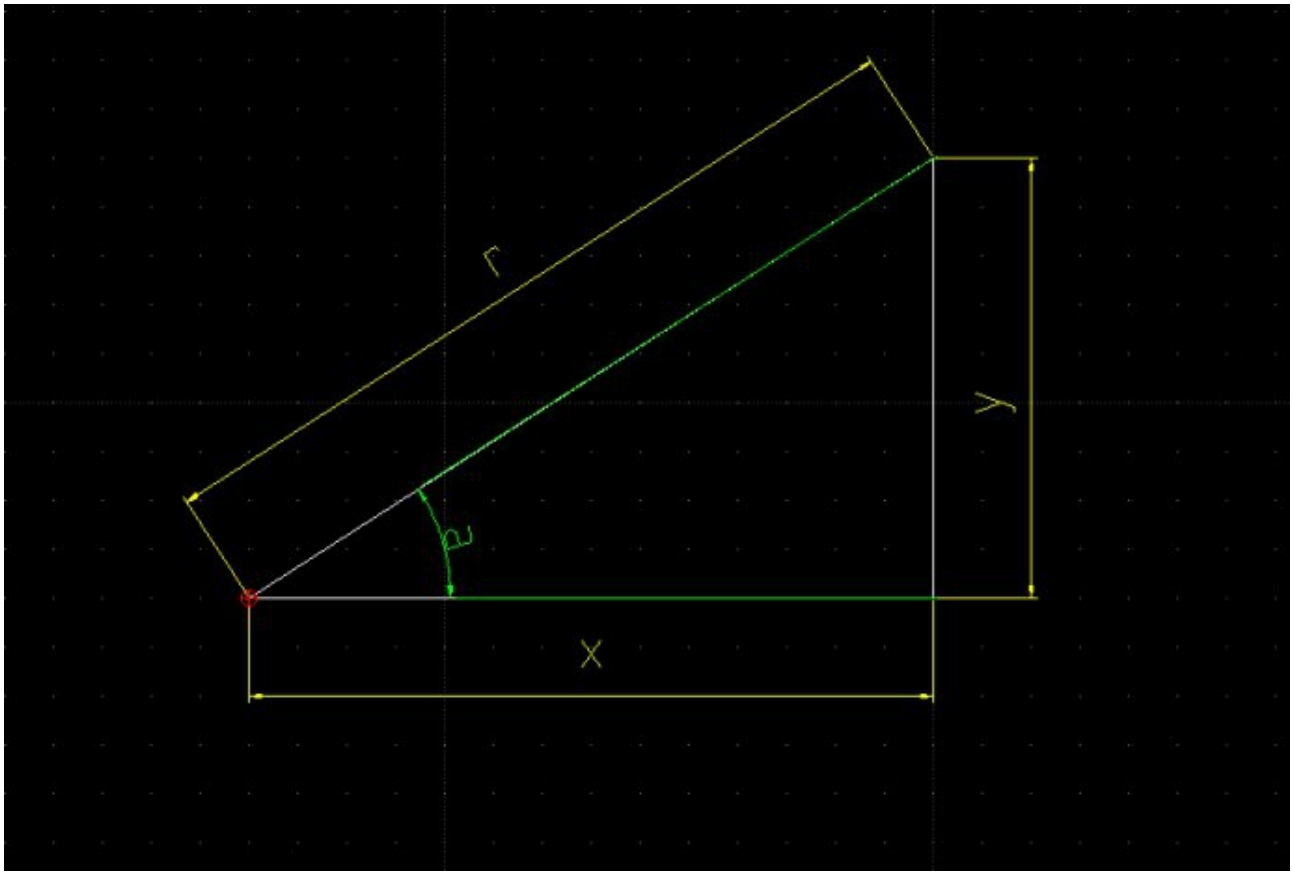
```
abs(-5.0);

abs(0);

abs(8.0);
```

Results:

输出结果：

```
5.0

0.0

8.0
```

三角函数基础知识：



这里，x 表示 x 坐标距离，y 表示 y 坐标距离，r 表示斜边，或者直接到达坐标的距离。

如果原点 o 为一个圆心，r 代表一个圆的半径，a 代表半径的旋转角度，旋转 r，形成一个圆形，坐标跟随着旋转的角度 a 的变化而随之变化。如果是一个机械装置，数控机床的直线进给，运动的方向是 x 和 y 方向，如果使其对圆弧补偿完成一个不同半径 r 的数值的圆形，就需要计算，给出 r 的数值，循环角度从 0 度到 90 度，然后分别对不同的域进行配置 x 和 y 的正负，完成最后的圆形。

Sin，cos，tan，cot，sec，csc 后边一个角度，运算后，这些数值都是在-1 和 1 之间。用字母 g 代表这个数值。

如果是循环 x 坐标，或者 y 坐标的距离，系统会生成不同的角度。从而根据这些角度计算出不同的坐标位置。而反三角函数的应用是某个-1 到 1 之间的数值（三角函数的比例数值），计算出其角度，应用也比较广泛。

求三角函数

```
Sin (a) = y / r ;
Cos (a )= x / r
Tan (a) = y / x
Cot (a) = x / y
Sec (a) = r / x
Cot (a) = x / y
Csc (a) = r / y
```

求角度(反三角函数)

```
asin(y/r) = a
acos(x/r)=a
atan(y/x)=a
```

Pytheras 定理

```
x**2+y**2=r**2
R=√x**2+y**2
x=√r**2-y**2
y=√r**2- x**2
```

求半径

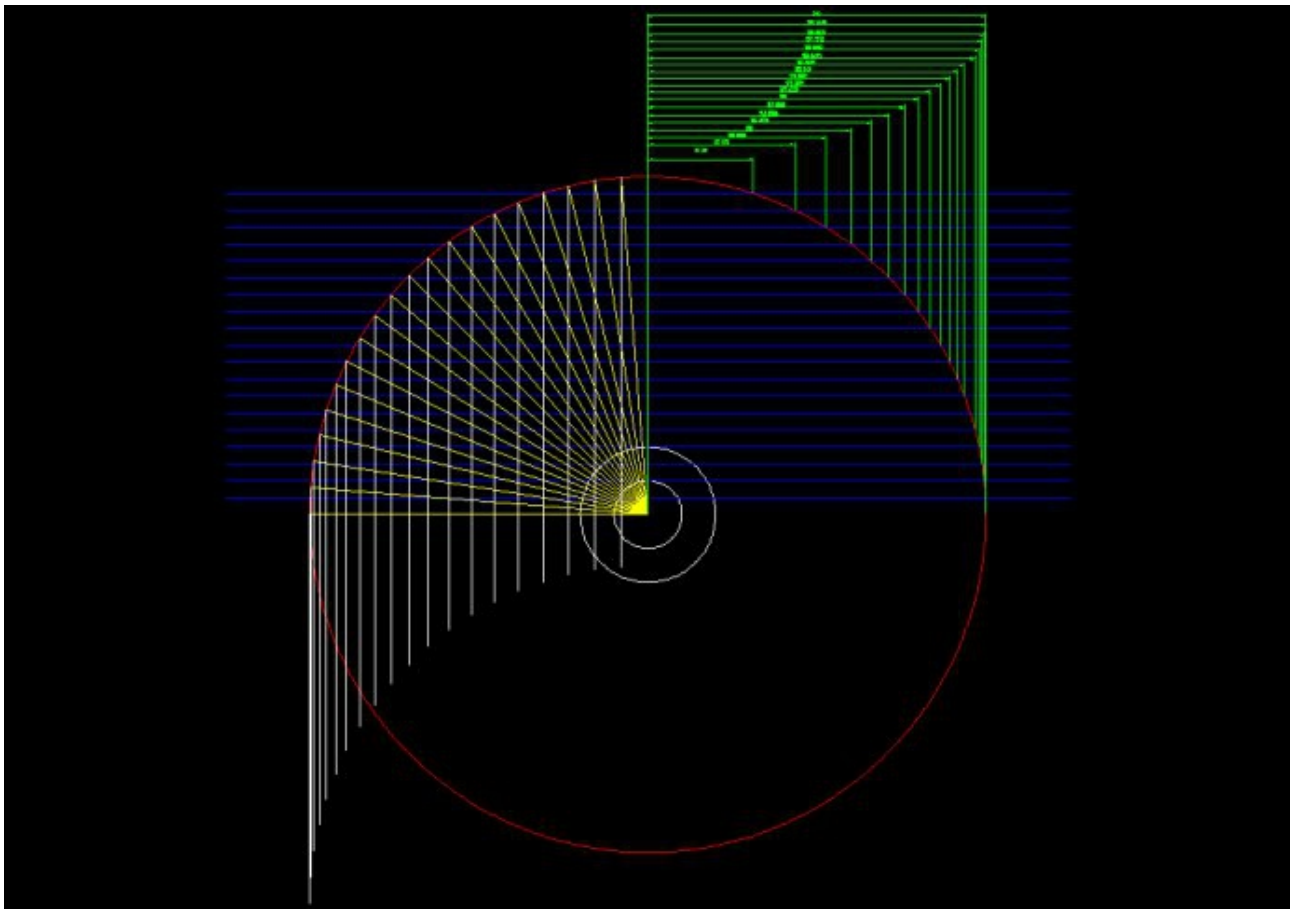
```
r=y/sin(a)
r=x/cos(a)
```

求 y 坐标

```
y=sin(a)*r
y=tan(a)*x
```

求 x 坐标

```
x=cos(a)*r
x=y/tan(x)
```



这个图片是一个圆形用角度和距离分别等分求其坐标位置，或者根据坐标位置求圆弧。

蓝色的横线把 y 坐标等分，而得出的 x 坐标的距离则是绿色部分的不等的结果。

黄色的直线把圆形通过角度等分，得出的白色是圆形的半径，长度一样，垂直，末端的参差不齐表示 y 坐标的数值，而垂直的白色半径直线的间距则表示 x 坐标的距离，明显是不一样的详细的参见三角函数数值表。

acos 反余弦
数学的 arccosine，或者 反余弦函数。

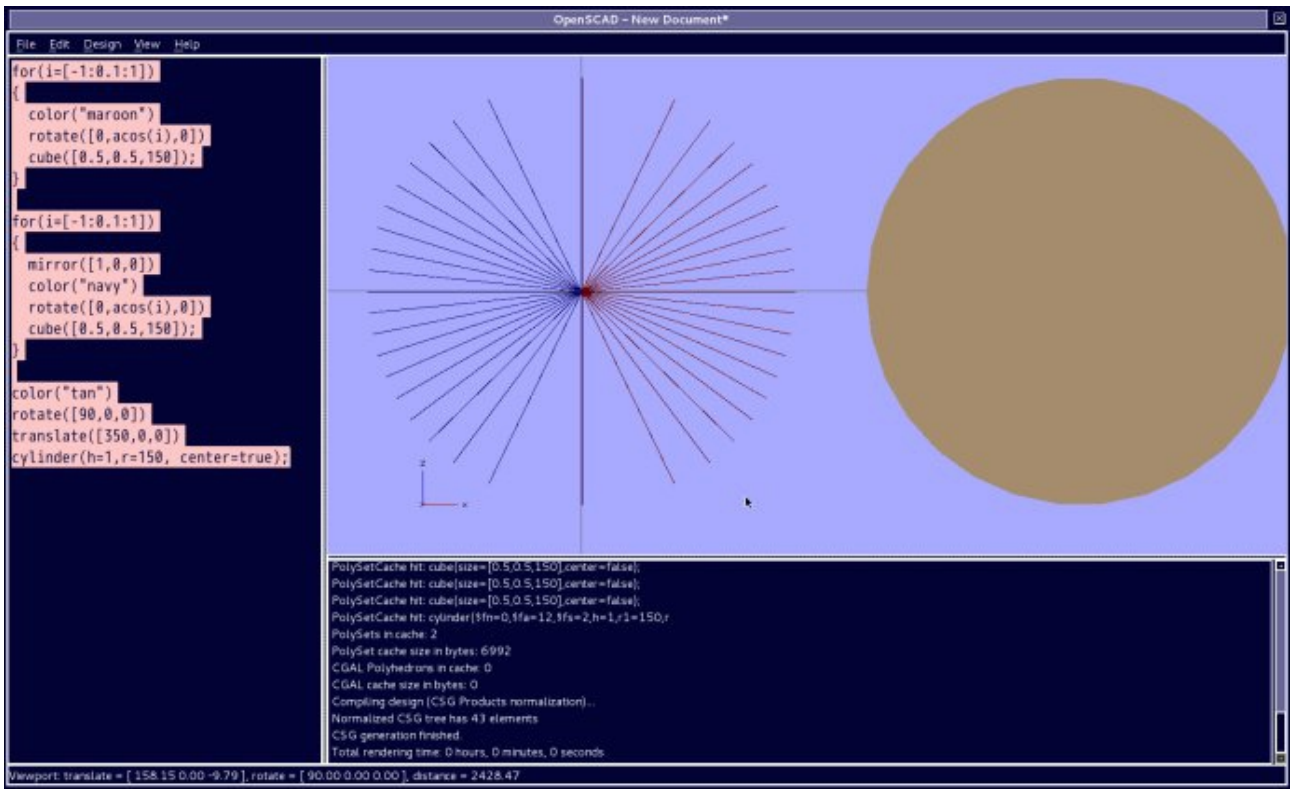
这里，我们用立方体[0.5,0.5,150](相当于一个直线)， 循环-1 到 1，增量为 0.1 (之后分别用不同的密度 0.01，0.001) ， 使用 acos 函数计算出不同角度，做一个 (-1 到 1 的角度只有 180 度，半个圆形,使用 mirror 命令才能完成整个圆形；也可循环从 0 到 1，增量 0.1，利用 rotate 完成整个圆形)等分圆形，根据循环的密度，确定这个圆形的精度。

acos(x/r)=a，通常是已知 x 坐标和 r 半径，求角度和 y 坐标.

实例：

for(i=[-1:0.1:1])

```
{
 color("maroon")
 rotate([0,acos(i),0])
 cube([0.5,0.5,150]);
}
for(i=[-1:0.1:1])
{
 mirror([1,0,0])
 color("navy")
 rotate([0,acos(i),0])
 cube([0.5,0.5,150]);
}
color("tan")
rotate([90,0,0])
translate([350,0,0])
cylinder(h=1,r=150, center=true);
```



asin 反正弦

数学 arcsine，或者反正弦，函数。

asin(y/r) = a，已知 y 坐标数值和 r 半径，求其三角形角度和 x 坐标。和 acos，数值应用的范围在-1 和 1 之间。详细的参见三角函数对照表。

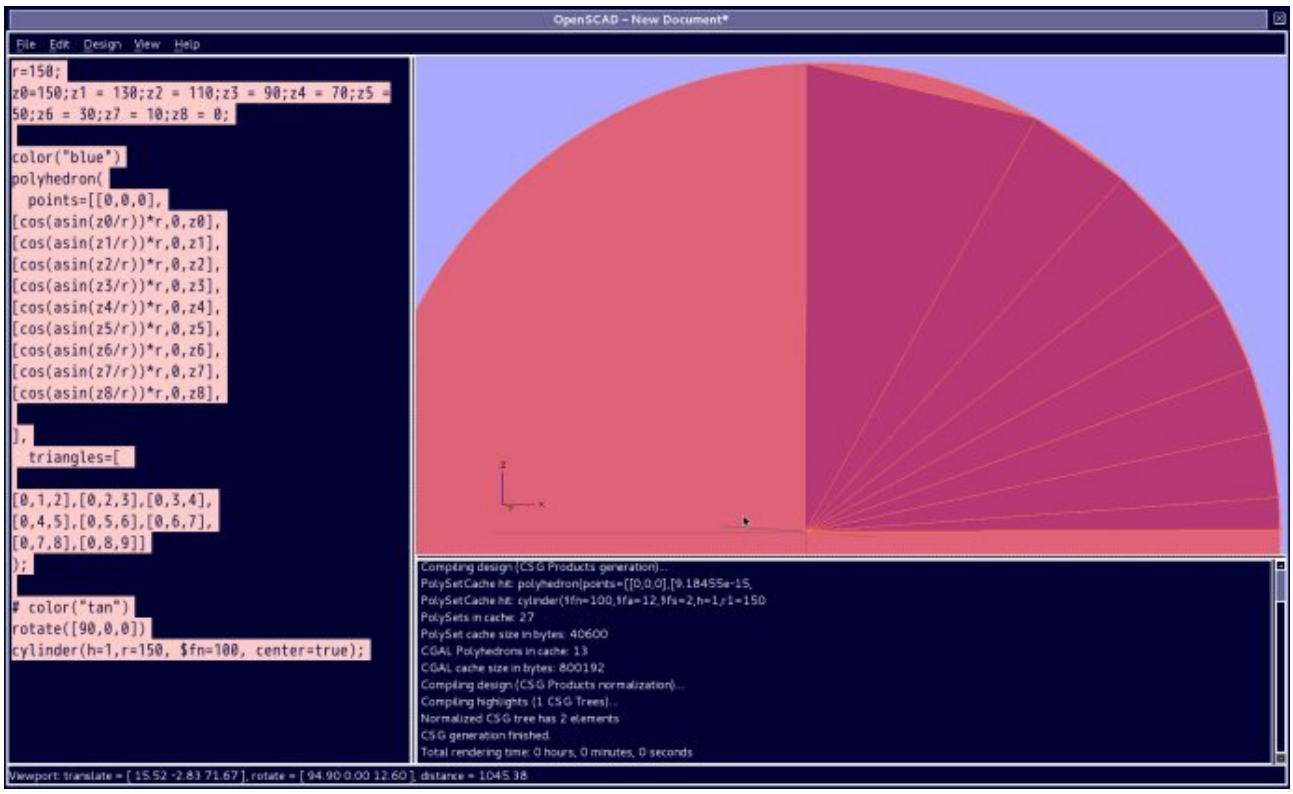
实例是根据角度求出多边形，得到一个多边形组合的圆形，组合规律根据 asin 的反正弦函数的规律组合。

实例：（这个效果是 F12 或者 view->'thrown together'，得到的效果，Ctrl+1 显示边缘，或者 view->show edges 得到的效果，这两个选择都必须选择，然后才能够开到图片的效果）。

代码如下：

```
r=150;

z0=150;z1 = 130;z2 = 110;z3 = 90;z4 = 70;z5 = 50;z6 = 30;z7 = 10;z8 = 0;

color("blue")

polyhedron(

 points=[[0,0,0],

[cos(asin(z0/r))*r,0,z0],

[cos(asin(z1/r))*r,0,z1],
```

```
[cos(asin(z2/r))*r,0,z2],
[cos(asin(z3/r))*r,0,z3],
[cos(asin(z4/r))*r,0,z4],
[cos(asin(z5/r))*r,0,z5],
[cos(asin(z6/r))*r,0,z6],
[cos(asin(z7/r))*r,0,z7],
[cos(asin(z8/r))*r,0,z8],
],
 triangles=[
[0,1,2],[0,2,3],[0,3,4],
[0,4,5],[0,5,6],[0,6,7],
[0,7,8],[0,8,9]]
);
# color("tan")
rotate([90,0,0])
cylinder(h=1,r=150, $fn=100, center=true);
```



atan 反正切

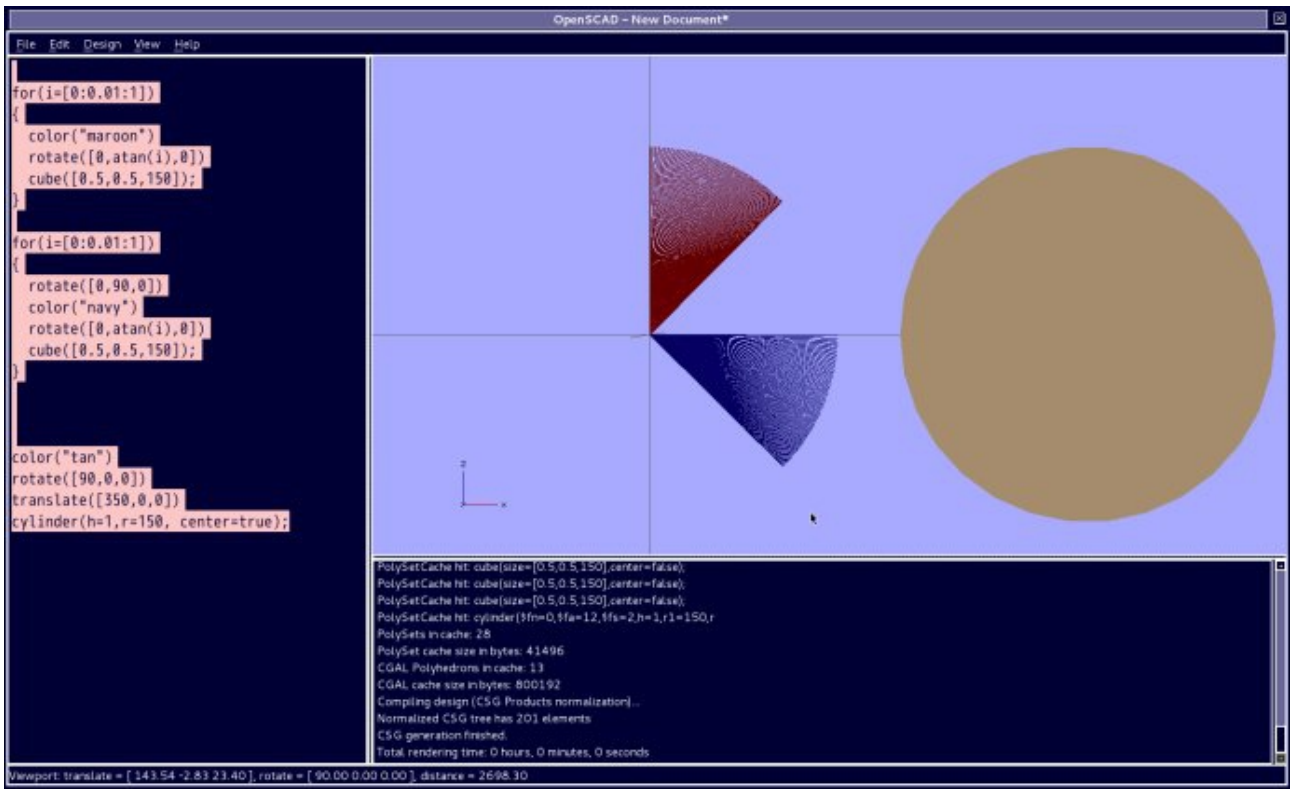
数学的arctangent，或者反正切，函数。是原则数值返回到x的正切弧度，用角度来表示。

atan(y/x)=a，已知x和y坐标数值，求角度，

实例：

```
for(i=[-1:0.01:1])
{
 color("maroon")
 rotate([0,atan(i),0])
 cube([0.5,0.5,150]);
}
for(i=[-1:0.01:1])
{
 mirror([0,0,1])
 color("navy")
```

```
 rotate([0,atan(i),0])
 cube([0.5,0.5,150]);
}
for(i=[-1:0.01:1])
{
 rotate([0,90,0])
 color("tea1")
 rotate([0,atan(i),0])
 cube([0.5,0.5,150]);
}
for(i=[-1:0.01:1])
{
 rotate([0,270,0])
 color("goldenrod")
 rotate([0,atan(i),0])
 cube([0.5,0.5,150]);
}
color("tan")
rotate([90,0,0])
translate([350,0,0])
cylinder(h=1,r=150, center=true);
```



atan2

数学的两个参数的 atan 函数。返回原则数组的弧度正切的 y/x，用角度来表达。参见：atan2(C 语言）。

（随后再做更多的实例和讲述）

cei1 最高限额

数学的 ceiling 函数。

在 C 语言中的描述和解释：

最小的积分数值，但是不小于 x(就像是一个浮点数值）。

```
#include
#include

int main ()
{
  printf ( "cei1 of 2.3 is %.1f\n", cei1(2.3) );
  printf ( "cei1 of 3.8 is %.1f\n", cei1(3.8) );
  printf ( "cei1 of -2.3 is %.1f\n", cei1(-2.3) );
  printf ( "cei1 of -3.8 is %.1f\n", cei1(-3.8) );
  return 0;
}
```

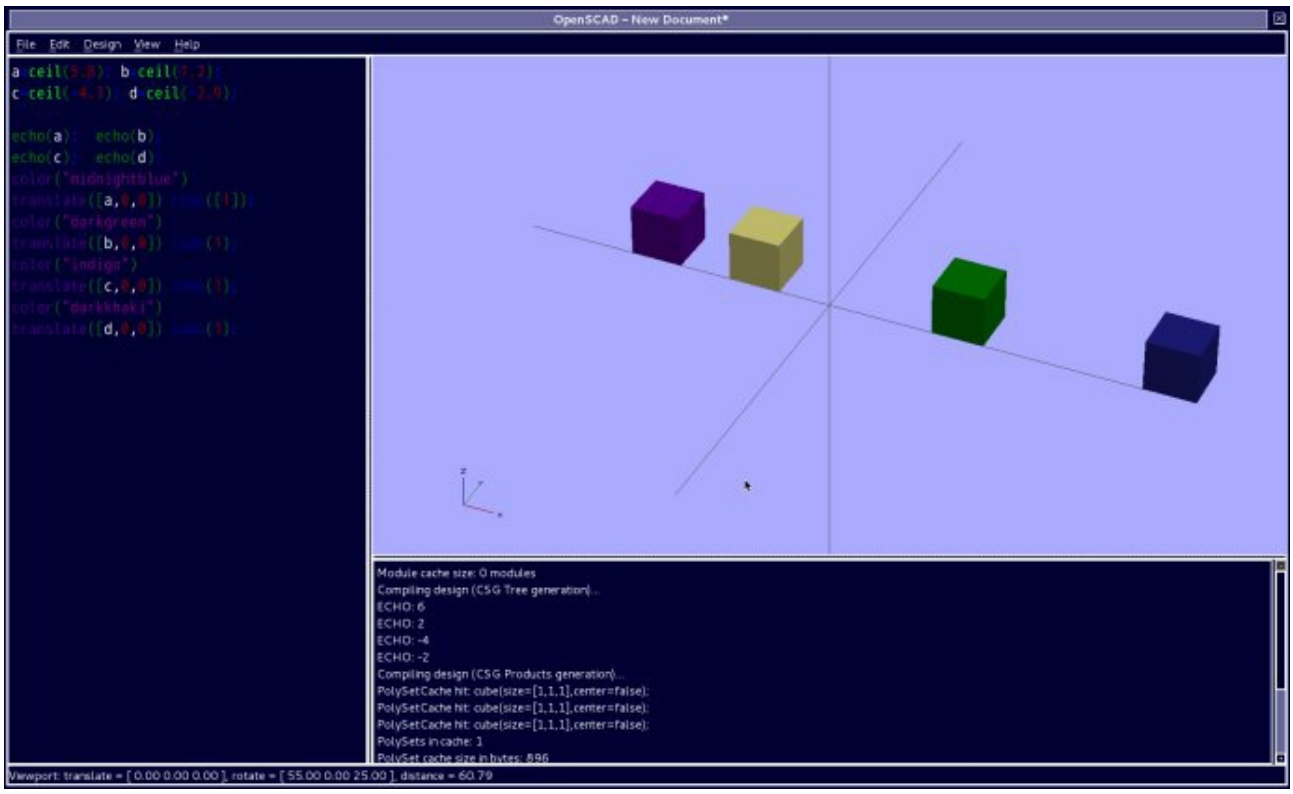
输出结果：

```
cei1 of 2.3 is 3.0
cei1 of 3.8 is 4.0
cei1 of -2.3 is -2.0
cei1 of -3.8 is -3.0
```

在 openSCAD 中，应用是一样的，不过这里的显示是用了 echo，终端 BASH 命令，和在 BASH 编程语言中和 printf 的意思是一样的，而在计算机图形学中，是立即显示计算的数值，openSCAD 应该是这一类的。实例中，a=cei1(5.8); b=cei1(1.2); c=cei1(-4.3); d=cei1(-2.9);得到的实际输出是 a=6,b=2,c=-4,d=-2.和 C 语言中计算的一样。

实例：

```
a=cei1(5.8); b=cei1(1.2);

c=cei1(-4.3); d=cei1(-2.9);

echo(a); echo(b);

echo(c); echo(d);

color("midnightblue")

translate([a,0,0]) cube([1]);

color("darkgreen")

translate([b,0,0]) cube(1);

color("indigo")

translate([c,0,0]) cube(1);

color("darkkhaki")

translate([d,0,0]) cube(1);
```

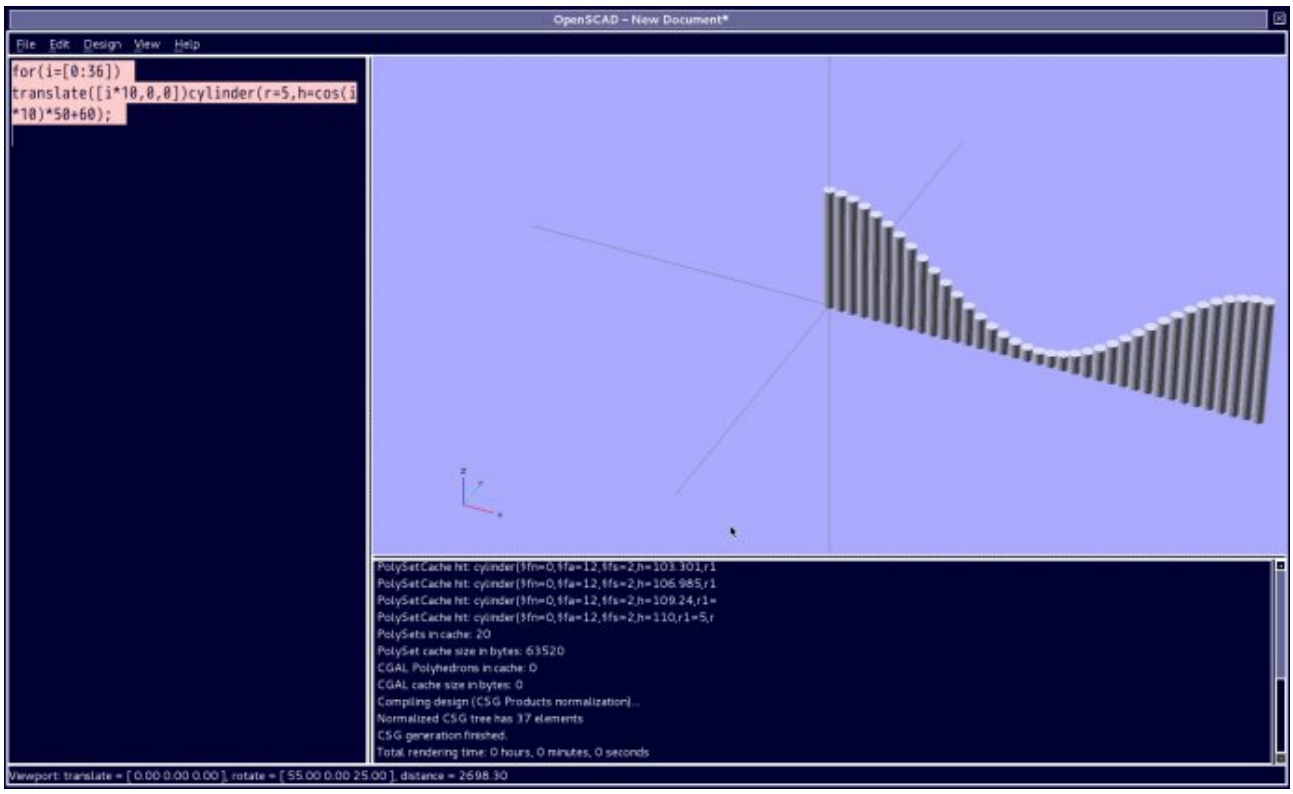


os 余弦

数学的余弦函数，Cos （a ）= x / r， 一般指已知角度，r 半径或者 x 坐标，求 r 半径或者 x 坐标。

实例：

```
for(i=[0:36])
translate([i*10,0,0])cylinder(r=5,h=cos(i*10)*50+60);
```



openSCAD cos 函数

C语言函数的描述：

```
#include
#include

#define PI 3.14159265

int main ()
{
  double param, result;
  param = 60.0;
  result = cos ( param * PI / 180.0 );
  printf ("The cosine of %f degrees is %f.\n", param, result );
  return 0;
}
```

输出：

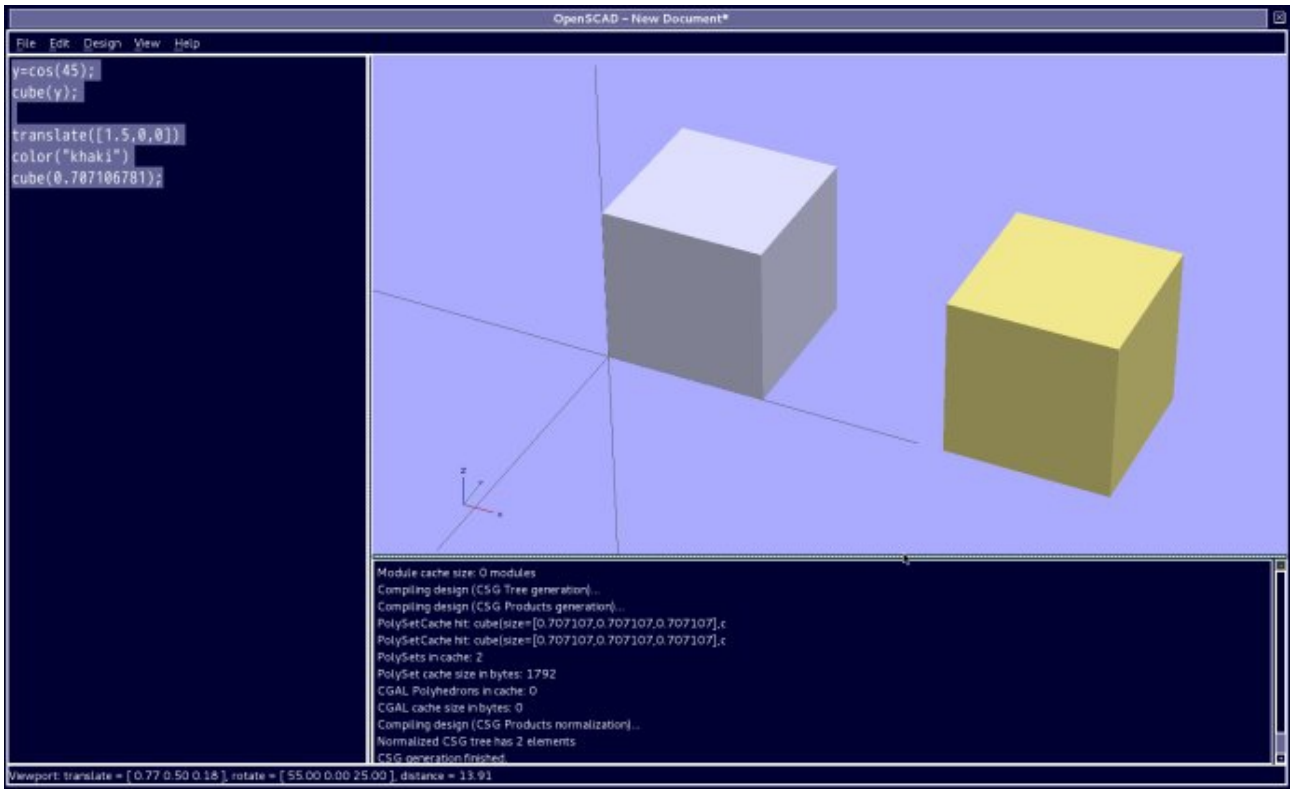
```
The cosine of 60.000000 degrees is 0.500000.
```

而在openSCAD中，可以直接输入cos(数值），而不是C语言中的cos(数值*PI/180）。

实例中立方体尺寸 cube(y)， y=cos(45)的数值是 0.707106781，而另外一个用立方体的尺寸直接用数值描述，输出的结果是一样的。

实例：

```
y=cos(45);

cube(y);

translate([1.5,0,0])

color("khaki")

cube(0.707106781);
```



这里，更多的应用根据公式可以拓展。

公式：Cos (a) = x / r,是根据已知 r 半径，或者循环增量 x 坐标,或者循环增量 a 角度，可以计算出一个圆弧轨迹。

实例：

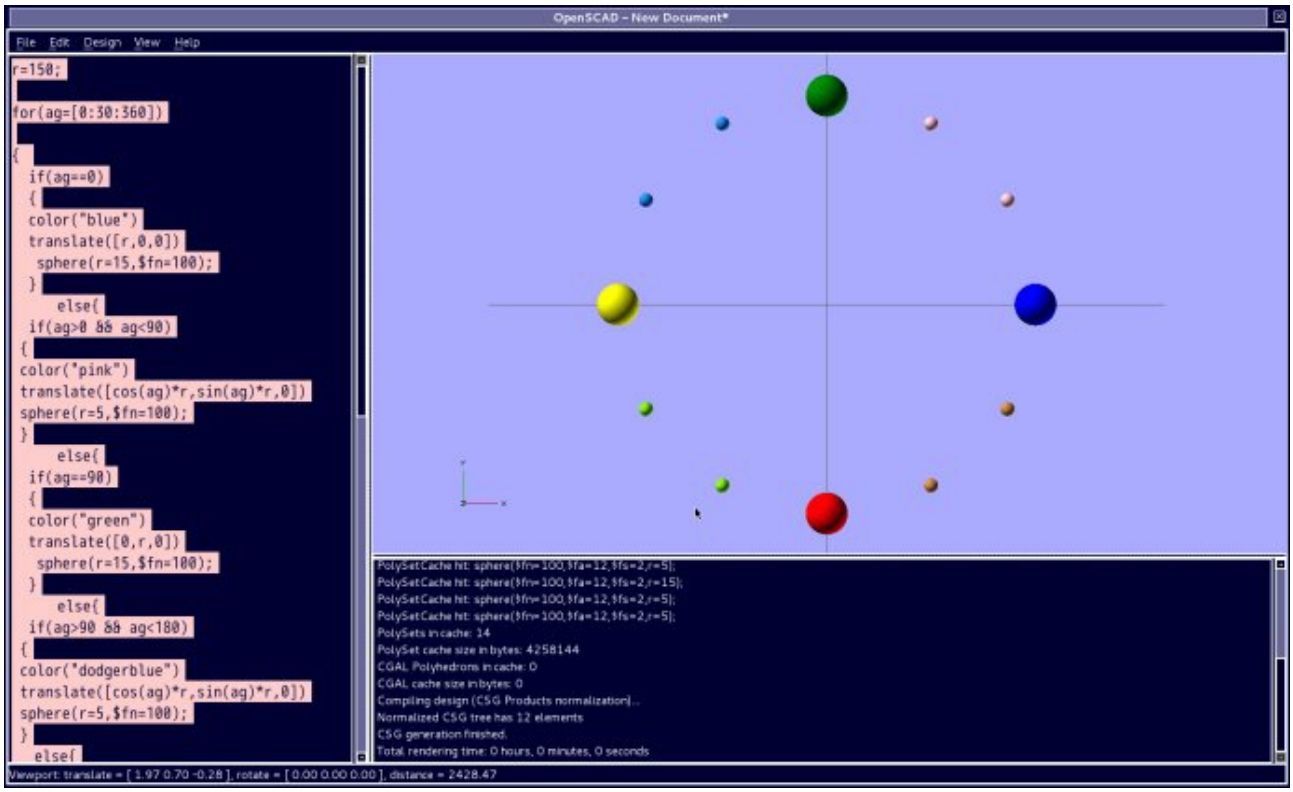
（这个实例是一个颜色分明的 12 小时时钟图形，用数学的的方法计算的.）

```
r=150;
```

```
for(ag=[0:30:360])
{
 if(ag==0)
 {
 color("blue")
 translate([r,0,0])
  sphere(r=15,$fn=100);
 }
else{
 if(ag>0 && ag<90)
{
color("pink")
translate([cos(ag)*r,sin(ag)*r,0])
sphere(r=5,$fn=100);
}
else{
 if(ag==90)
 {
 color("green")
 translate([0,r,0])
  sphere(r=15,$fn=100);
 }
else{
 if(ag>90 && ag<180)
{
color("dodgerblue")
translate([cos(ag)*r,sin(ag)*r,0])
sphere(r=5,$fn=100);
}
```

```
else{
 if(ag==180)
 {
 color("yellow")
 translate([-r,0,0])
  sphere(r=15,$fn=100);
 }
else{
 if(ag>180 && ag<270)
{
color("lawngreen")
translate([cos(ag)*r,sin(ag)*r,0])
sphere(r=5,$fn=100);
}
else{
 if(ag==270)
 {
 color("red")
 translate([0,-r,0])
  sphere(r=15,$fn=100);
 }
else{
 if(ag>270 && ag<360)
{
color("peru")
translate([cos(ag)*r,sin(ag)*r,0])
sphere(r=5,$fn=100);
}
```

```
}}}}}}}
}
```

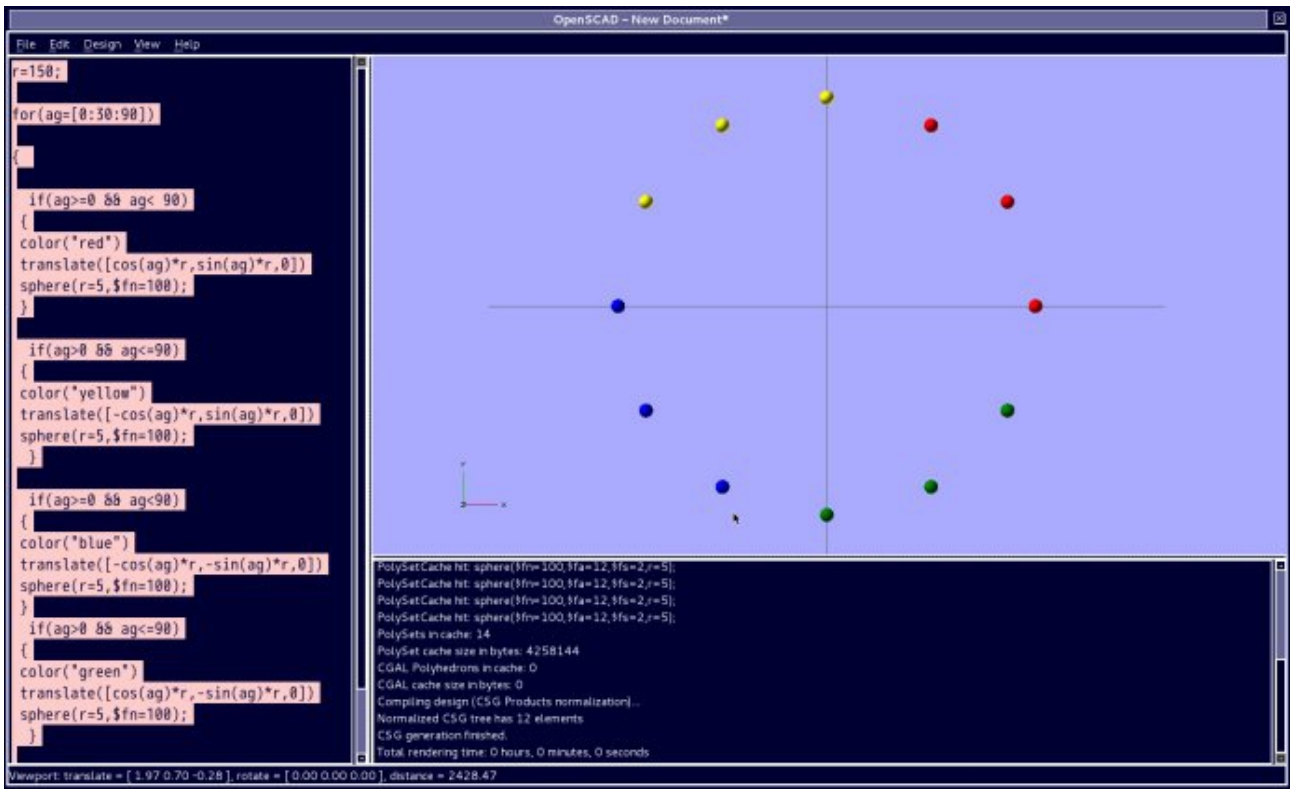


这里，还有一个相对简化的 12 位置时钟图形的画法：

实例代码：

```
r=150;
for(ag=[0:30:90])
{
 if(ag>=0 && ag< 90)
{
color("red")
translate([cos(ag)*r,sin(ag)*r,0])
sphere(r=5,$fn=100);
}
 if(ag>0 && ag<=90)
```

```
{
color("yellow")
translate([-cos(ag)*r,sin(ag)*r,0])
sphere(r=5,$fn=100);
 }
 if(ag>=0 && ag<90)
{
color("blue")
translate([-cos(ag)*r,-sin(ag)*r,0])
sphere(r=5,$fn=100);
}
 if(ag>0 && ag<=90)
{
color("green")
translate([cos(ag)*r,-sin(ag)*r,0])
sphere(r=5,$fn=100);
 }
}
```

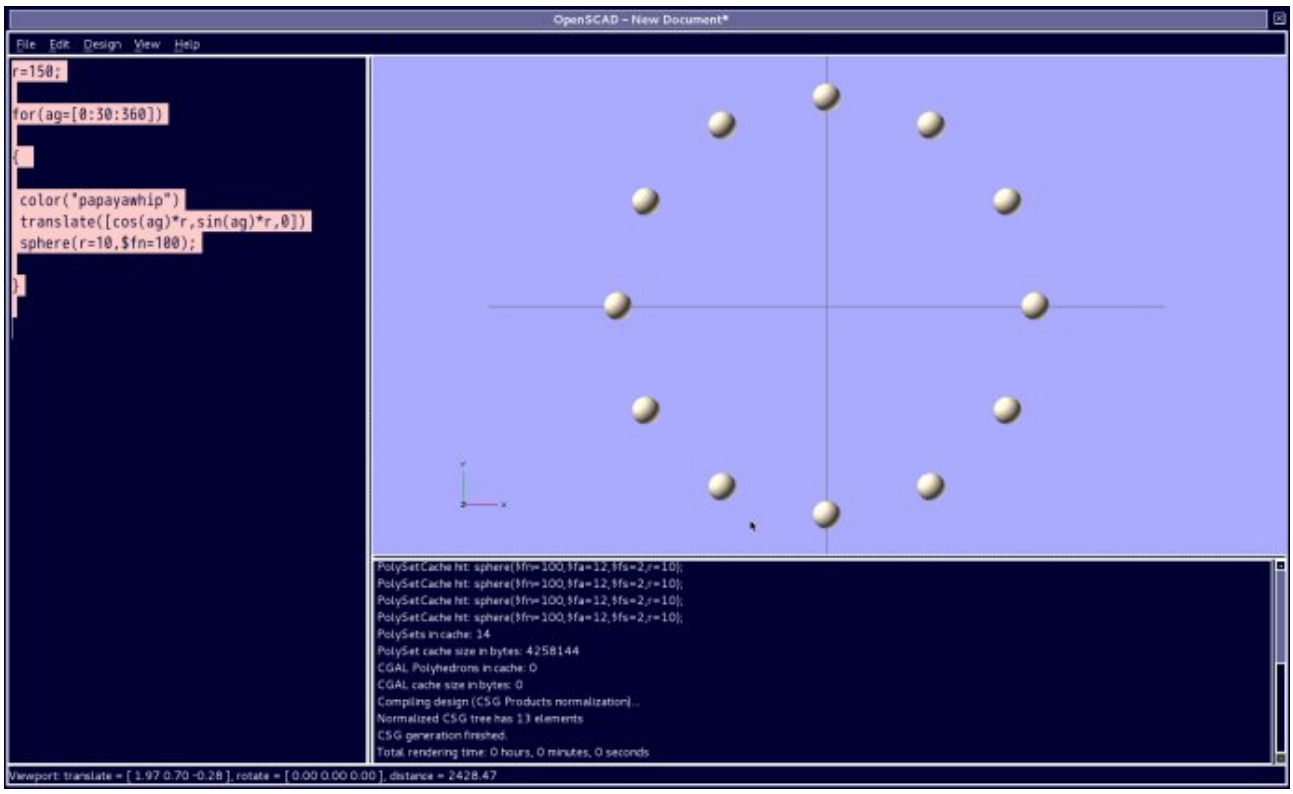


还有一个最简单的方法，上边的两个实例是使用了一些刚刚学习的数学运算符，逻辑运算符， 循环，条件选择，等等，更好的方便了学习 openSCAD 的各个功能。

更简单的 12 位置时钟图形的画法：

实例代码：

```
r=150;

for(ag=[0:30:360])

{

color("papayawhip")

translate([cos(ag)*r,sin(ag)*r,0])

sphere(r=10,$fn=100);

}
```



如果机械加工一个球形，机械是 x,y,z,三个坐标（机械进给系统是三坐标，x，y，z 方向），那么其轨迹就是首先加工一个圆形（x,z 坐标的圆形），然后再以这个圆形的圆心点为中心旋转(以 x,或者 z 轴为中心旋转），从而可以完成整个球形。

（注意：增加一个嵌套的循环增量 bg,用于旋转，bg 的密度，决定着旋转的精度；而循环增量 ag 是用于生成一个圆形，ag 的密度，决定着圆弧本身的精度， 不可以仅仅使用一个增量，否则球体无法完成。精度越高，计算的越复杂，计算机反应的速度就越慢，作为轨迹运算，的确很耗费内存。

还有，当 ag,bg 的增量分别是 5 的时候，计算就超出了范围，引出了一个警告（WARMING）。

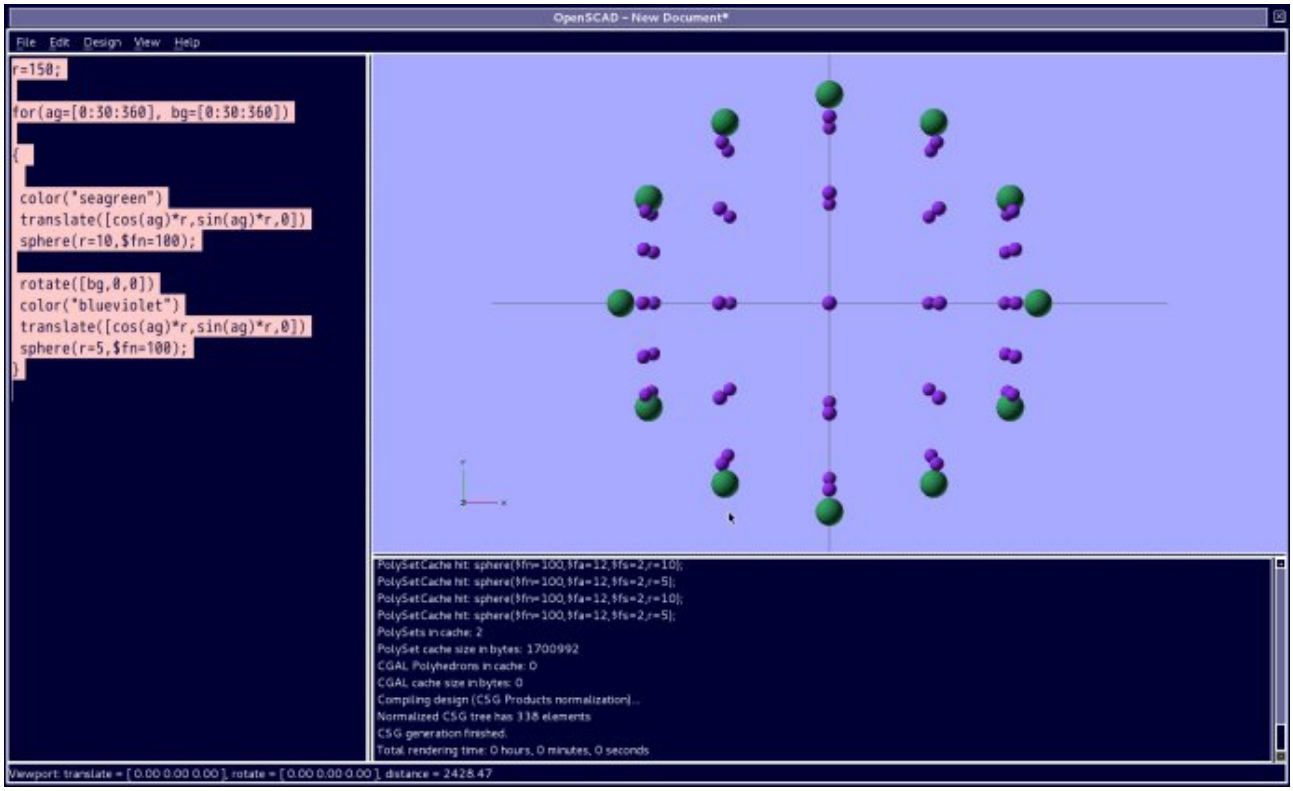
```
WARMING: Normalized tree has growing past 4000 elements. Aborting normalization.

WARMING: Normalized tree has 4000 elements.

WARMING: openCSG  rendering has been disabled!
```

因为 openCSG 最多支持 4000 个模型，正常情况下所有，我们这个球形轨迹超了，但是如果是用引入数据文件（dat,txt 等基础文本格式的），然后生成一个模型，可以比较灵活的应用数学的功能，后边我们详细的在学习它吧！）

实例代码：

```
r=150;

for(ag=[0:30:360], bg=[0:30:360])

{

color("seagreen")
```

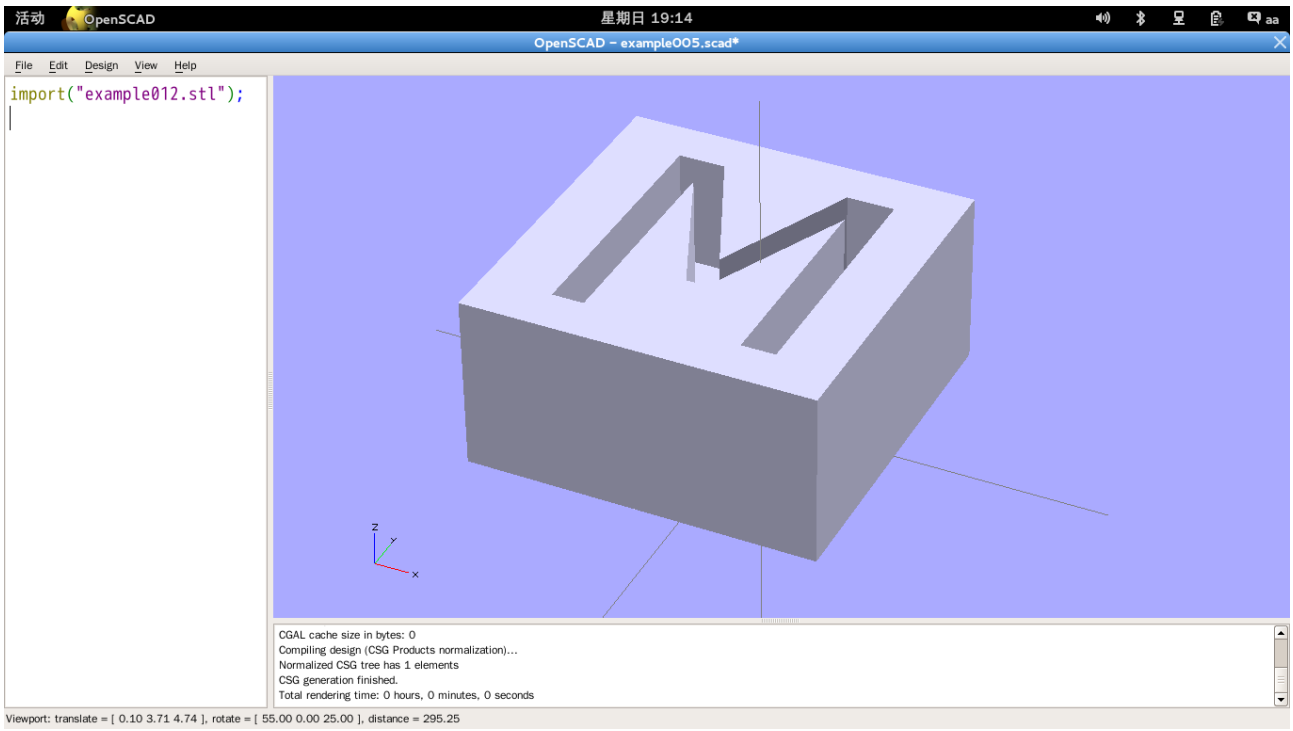
```
translate([cos(ag)*r,sin(ag)*r,0])
sphere(r=10,$fn=100);
rotate([bg,0,0])
color("blueviolet")
translate([cos(ag)*r,sin(ag)*r,0])
sphere(r=5,$fn=100);
}
```



7,
13 Importing Geometry 导入几何
            13.1 import 导入
            13.2 import_st1 导入 st1
14 Include Statement include 描述
15 Other Language Features 其他的语言特点
        15.1 Special variables 特殊变量
        15.1.1 $fa, $fs and $fn
        15.1.2 $t
        15.1.3 $vpr and $vpt
        15.2 User-Defined Functions 用户自定义函数
        15.3 Echo Statements echo 描述
        15.4 Render 渲染
        15.5 Surface 表面
         15.6 Search 搜索
                1.15.6.1 Index values return as list 索引数值返回就像列表
                1.15.6.2 Search on different column; return Index values 搜索在不同的行，返回索引数值
                1.15.6.3 Search on list of values 搜索在数值列表上
                1.15.6.4 Search on list of strings 搜索在字符串列表上
                1.15.6.5 Getting the right results 获取正确的结果

13,导入几何

import 导入
在当前 OpenSCAD 模型导入文件中使用
参数
<文件>
一个字符串，包含路径的 STL 或 DXF 文件。
使用示例：
```
import ("example012.stl", convexity = 5);
```



注：在 OpenSCAD 的最新版本，import () 现在用于导入二维（DXF 用于扩展）和三维（STL）文件。
import_stl
<不推荐使用。使用的命令，而不是 import..>
在当前 OpenSCAD 模型导入 STL 文件的使用
参数
<文件>
含有对 STL 文件的路径字符串包含。
<凸>

Integer。凸参数指定前侧的最大数量（背侧）射线相交的对象可能渗透。此参数时，才需要进行正确地显示在 OpenCSG 预览模式的对象，并且对没有影响
多面体的渲染。
使用示例：

```
import_stl("example012.stl", convexity = 5);
```

include 语句

在目前形式的文字是不完整的。

对于包括 OpenSCAD 从外部文件的代码，有两种可用的命令：
・包括<文件名>作为，如果包含文件的内容写在
包含文件，并 使用<文件名>进口模块和功能，但不执行任何命令 比其他的定义。 库文件中搜索在同一个文件夹中的设计是从开放的，还是在 在 OpenSCAD 安装的库文件夹中。您可以使用相对路径规范要么。如果他们在其他地方，你一定要给的完整路径。新版本有预定义的用户库，请参阅 OpenSCAD_Us

er_Manual/库页面，这也文档 包括在 OpenSCAD 一些库文件。
Windows 和 Linux/ Mac 上使用不同的分隔符的目录。 Windows 使用\，例如目录\ file.ext，而其他人使用/，例如目录/ file.ext。这可能导致跨平台的问题。然而 OpenSCAD 在 Windows 上正确处理使用/，所以使用/在所有包含或使用的语句会在所有的平台上。 使用包括<文件名>允许缺省变量在库中指定。这些 默认值可以在主代码覆盖。一个 openscad 变量只有在一个值该程序的生命。当有多个任务需要的最后一个值，但第一次创建变量时指定。这有效果库中的分配时， 如您以后使用更改默认的变量，必须在该分配 include 语句。请参见下面的第二个例子。

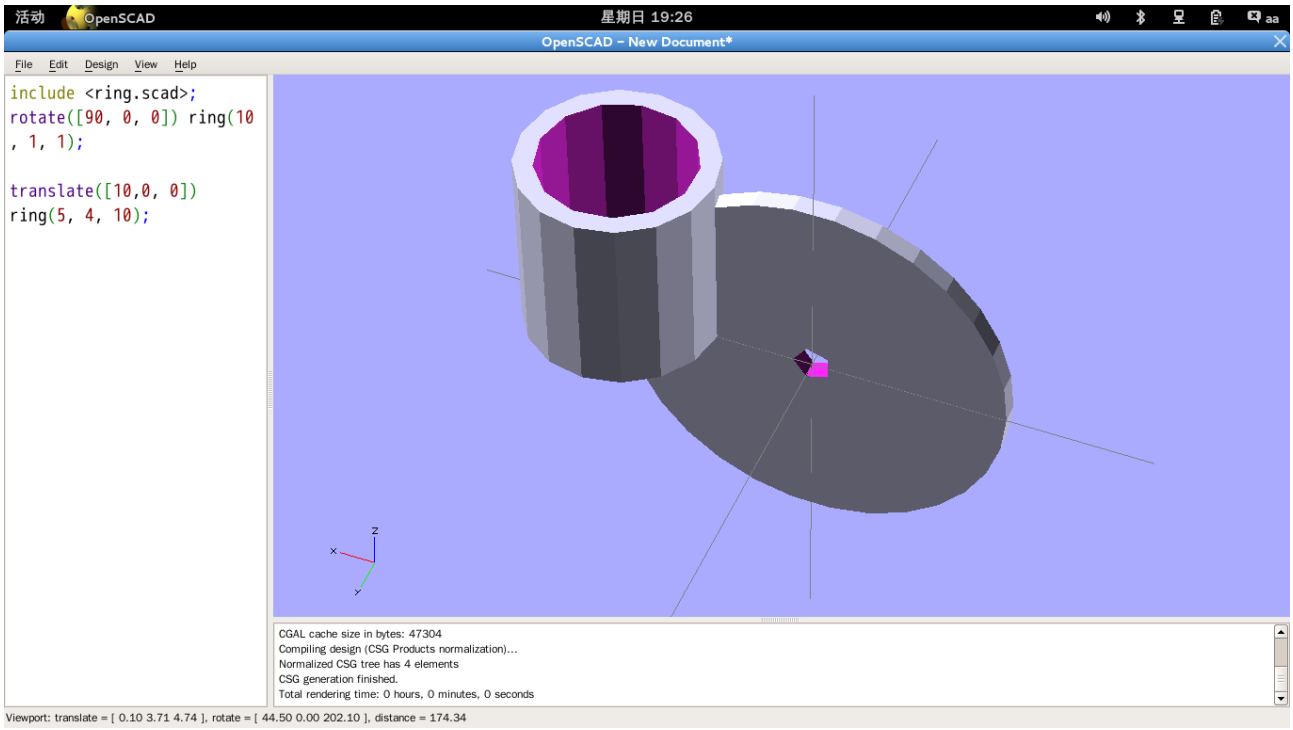
首先保存一个库文件 ring.scad,然后再用打开一个新的 openSCAD 文件，用 include 引用。

注意一定要保存到同一个文件夹内，否则需要输入路径。

```
// ring.scad,
module ring(r1, r2, h) {
difference() {
cylinder(r = r1, h = h);
translate([ 0, 0, -1 ]) cylinder(r = r2, h = h+2);
}
}
```

整个操作就和 C 语言的引用类似。
打开一个新的 openSCAD 文件:

```
include <ring.scad>;
rotate([90, 0, 0]) ring(10, 1, 1);

translate([10,0, 0])
ring(5, 4, 10);
```

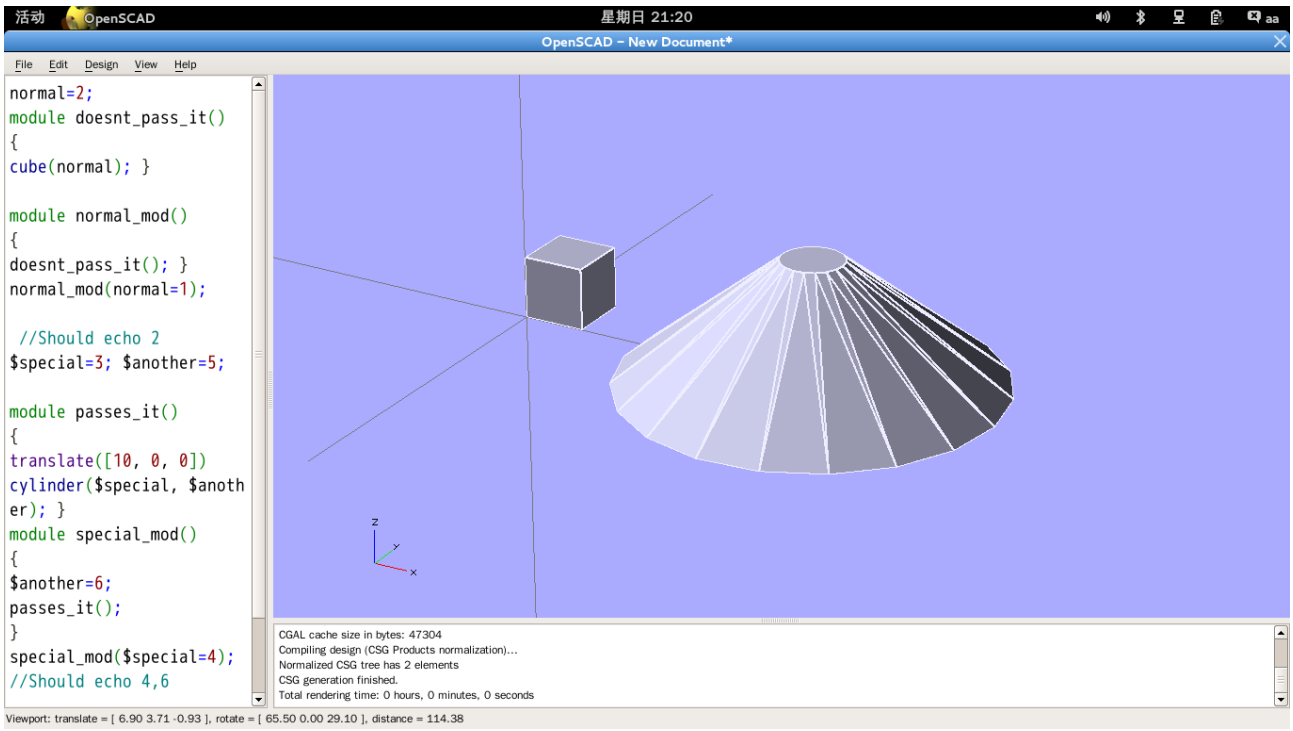


其它语言功能
在目前形式的文字是不完整的。
特殊变量
所有的起始用'$'是特殊变量，变量的语义是类似于
在LISP特殊的变量：他们有活力，而不是词法范围。
这也就意味着，他们正在有效地自动向前作为参数传递。

个人理解： 这个变量$就相当于C语言中的*， pointer。

比较正常的一个特殊的变量：

```
normal=2;
module doesnt_pass_it()
{
cube(normal); }

module normal_mod()
{
doesnt_pass_it(); }
normal_mod(normal=1);

//Should echo 2
$special=3; $another=5;

module passes_it()
{
translate([10, 0, 0])
cylinder($special, $another); }
module special_mod()
{
$another=6;
passes_it();
}
special_mod($special=4); //Should echo 4,6
```



$fa, $fs, $fn,
$fa, $fs, $fn,特殊变量控制，用来产生弧面的数量：
$fa,是片段的最小角度。即使是一个巨大的圆圈不会有更多的碎片比 360 这个数除以。默认值是 12（即 30 片段为完整的
圆）。

最小允许值是 0.01。任何试图设置一个较低的值将导致

警告。
$fs,是一个片段的最小尺寸。这个变量，因为很小的圈,数量较少使用$fa 于指定片段。默认值是 2 的最小 允许的值是 0.01。任何试图设置一个较低的值会产生警告。
$ FN 通常为 0。当这个变量的值大于零，则其他两个变量 被忽略，全部循环利用碎片这个数字呈现。缺省值是 0。 当$fa 和$fs 用于确定碎片的数量为一个圆，然后 OpenSCAD 永远用不少于 5 片段。

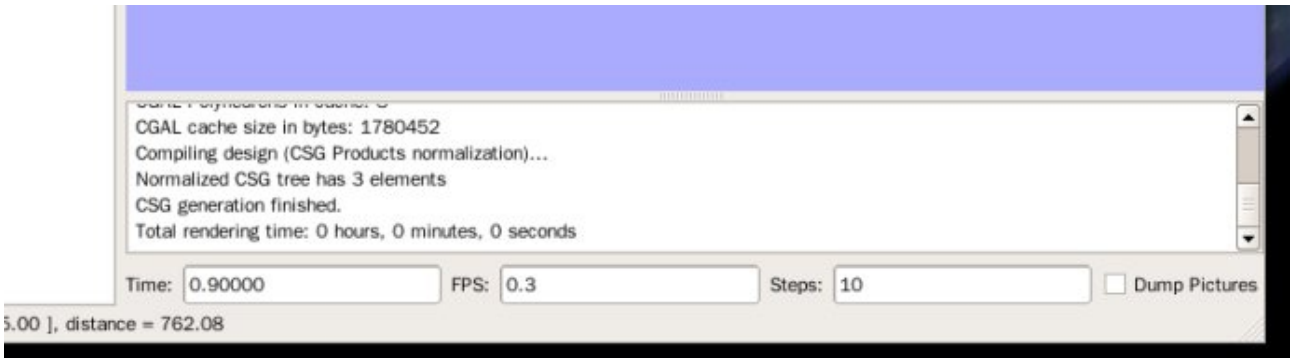
之前详细的介绍过，这里不再说了。

$t

这个函数是整个 openSCAD 的运动仿真核心工具之一。

语法： 是将$t 函数用个乘符号‘*’ 乘以任何固定的数值，当乘以之后，这个函数就变成了一个变量，就会从 0 到这个数值。

同时这个函数必须配合菜单栏的"View"的子菜单“Animate"使用，生成一个带有时间延迟控制，和精度控制函数。



1，Steps，步数，就是多少张图片的意思。
2，FPS，FPS，帧率，每秒钟有多少张图片，这里有两个数值需要输入，
3，Time,则是一个自动生成的函数，就是根据帧率和步数计算出来的。

然后，我们的固定数量*$t 之后的变量，分成多少份，就是 steps，而多少时间显示这些分数，就是 FPS.

这里有一个时钟仿真案例，仅供参考。

首先，我们先用文档编辑器，或者 openSCAD 编辑以下代码，然后将这些代码保存为 hdf1.scad,因为这个时钟仿真程序比较大，所以将一些函数放在头文件内，主体程序引用这些函数即可。

```
// hd1.scad

module clock(){

r=150;

for(ag=[0:30:360])

{

 if(ag==0)

 {

 color("blue")

 translate([r,0,0])

  sphere(r=15,$fn=100);

 }

else{

 if(ag>0 && ag<90)

{
```

```
color("pink")
translate([cos(ag)*r,sin(ag)*r,0])
sphere(r=5,$fn=100);
}
else{
 if(ag==90)
 {
 color("green")
 translate([0,r,0])
  sphere(r=15,$fn=100);
 }
else{
 if(ag>90 && ag<180)
{
color("dodgerblue")
translate([cos(ag)*r,sin(ag)*r,0])
sphere(r=5,$fn=100);
}
else{
 if(ag==180)
 {
 color("yellow")
 translate([-r,0,0])
  sphere(r=15,$fn=100);
 }
else{
 if(ag>180 && ag<270)
{
```

```
color("lawngreen")
translate([cos(ag)*r,sin(ag)*r,0])
sphere(r=5,$fn=100);
}
else{
 if(ag==270)
 {
 color("red")
 translate([0,-r,0])
  sphere(r=15,$fn=100);
 }
else{
 if(ag>270 && ag<360)
{
color("peru")
translate([cos(ag)*r,sin(ag)*r,0])
sphere(r=5,$fn=100);
}
}}}}}}}
}
}
```

保存 hdf1.scad 之后，我们打开一个新的 openSCAD 窗口，保存这个文件到 hdf1.scad 的文件目录中，否则无法引用 hdf1.scad 文件。比如说我们可以保存为 clock.scad 等等。

以下是时钟仿真程序代码：

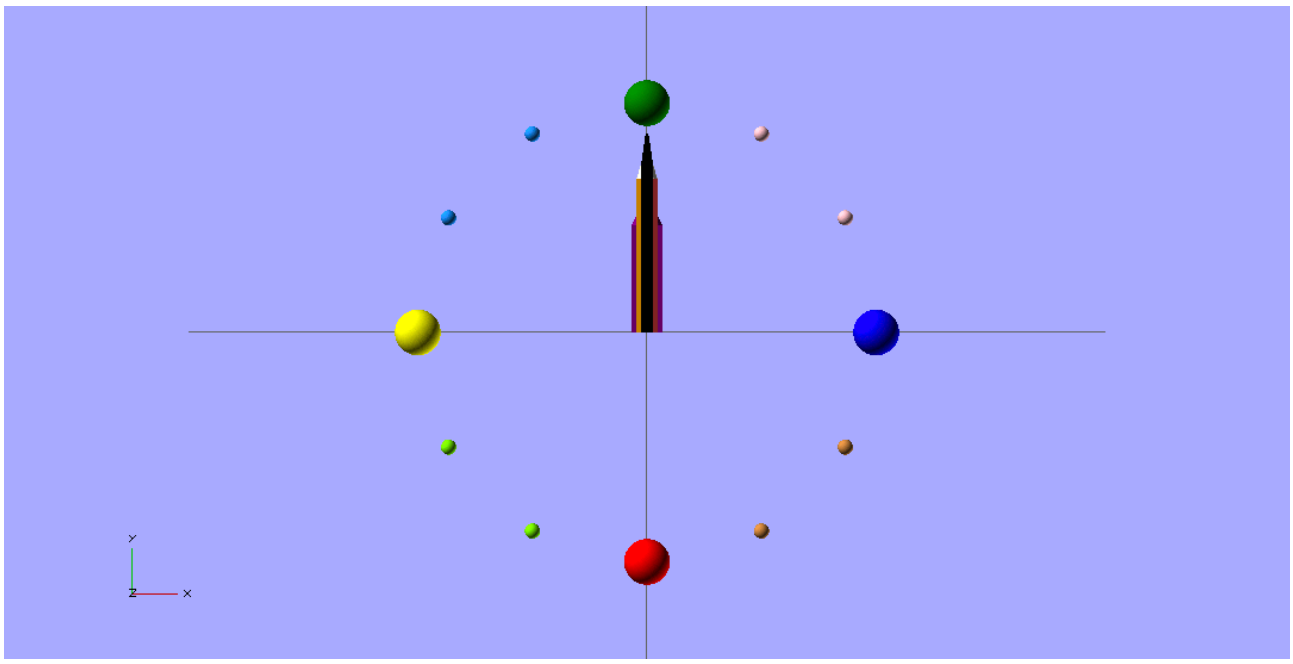
```
include <hdf1.scad>


clock();


module pin(a, b){
color("brown")
cube([b, a, 20]);
```

```
color("orange")
translate([0, a, 0])
rotate([0, 0, 180])
cube([b, a, 20]);


color("beige")
translate([0, a, 0])
rotate([-90, 0, 0])
cylinder(20, b, h=20);

}




// hours pin
rotate([0, 0, 0])


color("purple")
pin(70,10);



// minutes pin
rotate([0, 0, 0])


pin(100, 7);



// seconds pin
rotate([0, 0, 360*$t])


color("black")
pin(110, 4);
```

我们然后点击"View"->"Animate", 当你看到'Animate'打上对勾，且右下角的运动仿真状态栏出现了Time, FPS, Steps,的时候，这时候需要你设置FPS和Steps了。

我们已经把秒针的旋转设为360，因为一个圆弧是以360度来计量一个元周期的，所以秒针的指针设置为rotate([0, 0, 360*$t]);参考已经设定好的程序代码。这里不需要改动，只是强调一下。

然后我们设置FTS为4，Steps=4,这个时候黑色的秒针就开始已0.25秒一次的速度移动了。



这个图片的帧率是 4，张数是 4，旋转角度是 360 度，就是在 0.25 秒旋转 90 度。

FPS=1， 就是一秒显示运动仿真一个 Steps，Times 还是会显示 1/Steps 秒.
FPS=3，就是一秒内显示三个 Steps.
Times = 1 / Steps 秒，
Steps/FPS=实际每秒显示多少张图片。
FPS=Steps，就是一秒内显示所有的 Steps.

我测试了，openSCAD 的最快的帧率是 50 steps 每秒，如果 Steps 太高也是不可以运行的，所以适当。

```
include <hdfl.scad>

clock();

module pin(a, b){
color("brown")
cube([b, a, 20]);

color("orange")
translate([0, a, 0])
rotate([0, 0, 180])
cube([b, a, 20]);

color("beige")
translate([0, a, 0])
rotate([-90, 0, 0])
cylinder(20, b, h=20);
```

```
}
```

```
// hours pin
rotate([0, 0, 360*$t])

color("purple")
pin(70,10);


// minutes pin
rotate([0, 0, 360*60*$t])

pin(100, 7);
```

因为 openSCAD 计算旋转一个圆圈周期是 360 度，所以这里的时钟仿真最大可以做到分针和时针，而秒针的确没有办法计算了。

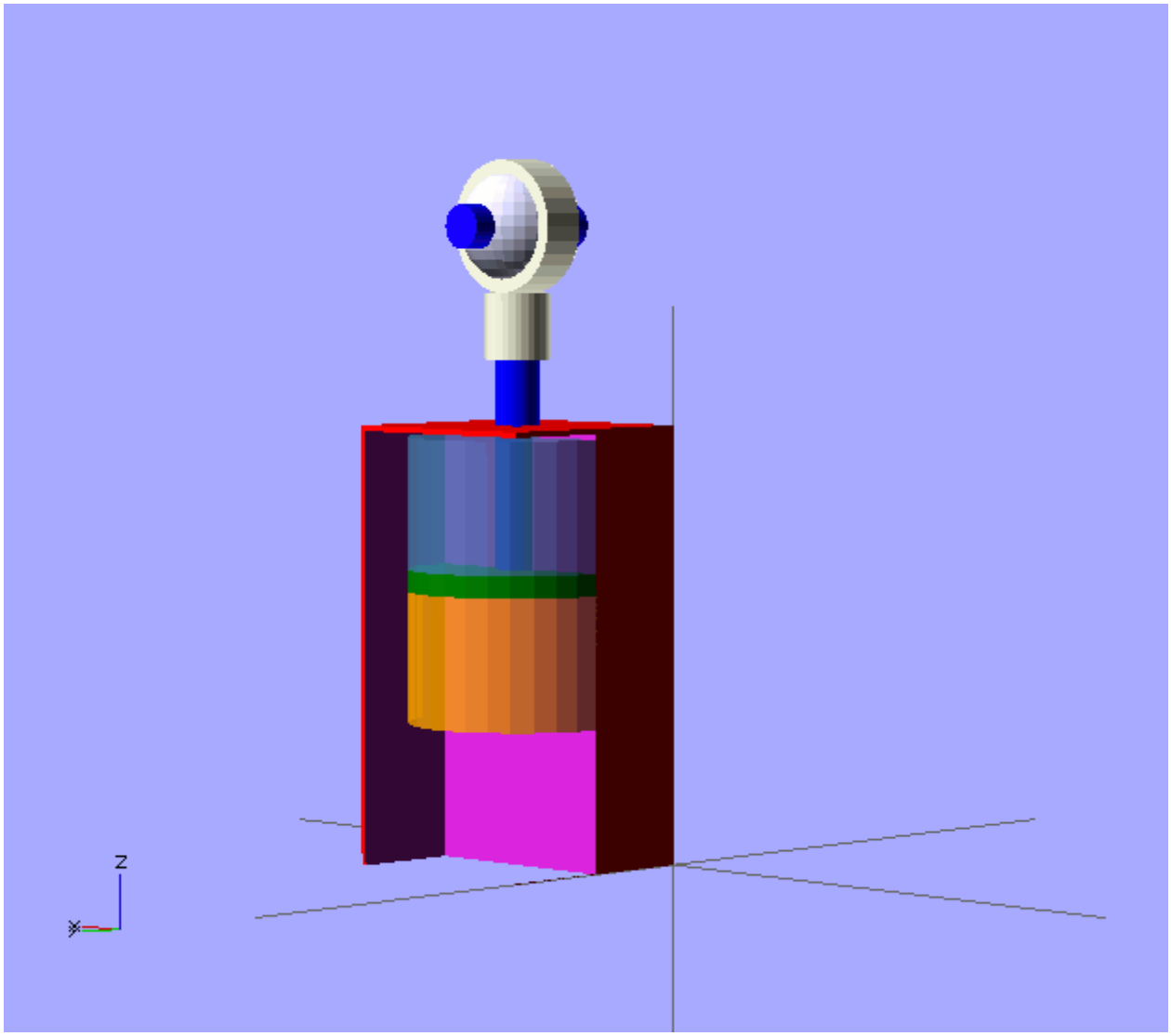
我们把时针旋转设置成 360*$t,表示整个运动仅仅仿真一个圆周，而这个时候分针则需要运动 60 圈，也就是 60 个圆周，表达为 360*60*$t。

这个时候设置 FPS 为 60， Steps 为 360， 否则运算的太快，openSCAD 的运动会呈现不出来的。

这个是一个液压缸运动仿真的案例，仅供参考。

```
module cushion(){
difference(){
color("red")
cube([200, 200, 400]);
translate([-5, 100, -5])
cube([200, 200, 400]);
}


translate([0,0,-120*$t])
union(){
color("blue")
translate([100, 100, 250])
cylinder (r=20, h=260);


color("beige")
union(){
translate([100, 100, 460])
```

```
cylinder (r=30, h=60);

rotate([90, 0, 0])
translate([100, 580, -120])
cylinder(r=60, h=40);

}

translate([100, 100, 580])
sphere (r= 52 , fn=200);

color("blue")
rotate([90, 0, 0])
translate([100, 580, -170])
cylinder (r=20, h=140);

color("green")
translate([100, 100, 250])
cylinder(r =100, h=20);




}



color("steelblue", 0.8)
translate([100, 100, 390])
rotate([0,180,0])
cylinder(r =100, h=120);

color("steelblue", 0.8)
translate([100, 100, 270])
rotate([0,180,0])
cylinder(r =100, h=120*$t);

color("orange", 0.8)
translate([100, 100, 130])
cylinder(r =100, h=120-120*$t);






}



cushion();
```



$vpr and $vpt

当你在一个位置添加$vpr，$vpn 的时候， 光标在编辑栏的那个位置，选择

“Edit “-> “Paste Viewport Rotation” or “Paste Viewport Translation”

当你选择Rotation or Translation 的时候，最底下的状态栏的数据就会复制到你的光标的位置。

就是你当前图像的视图角度和模型的坐标中心偏移的参数。

Example :

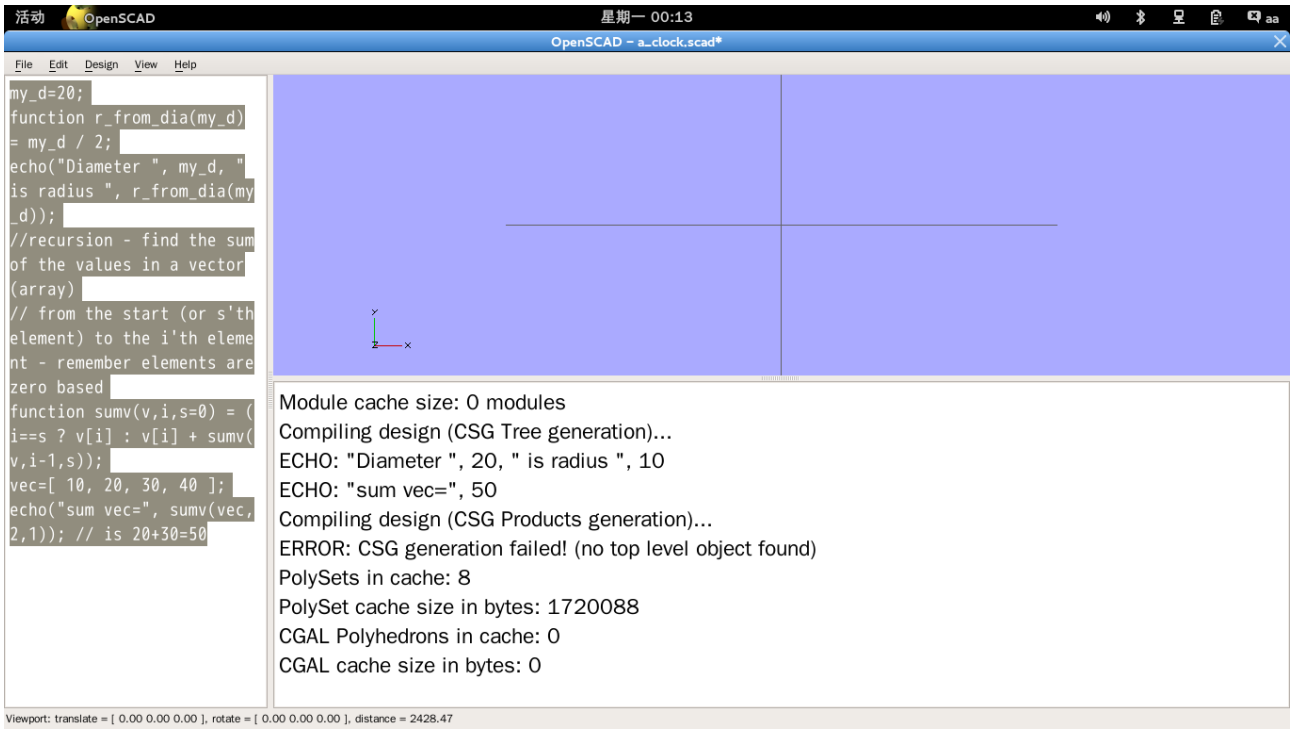
```
cube([10,10,$vpr [0]/10]);
```

```
// Ctr1+p && Ctr1+0
//[ 0.00, 0.00, 0.00 ]

//[ 55.00, 0.00, 25.00 ]

// Ctr1+p && Ctr1+4

//[ 0.00, 0.00, 0.00 ]

//[ 0.00, 0.00, 0.00 ]

// Ctr1+p && Ctr1+7

//[ 0.00, 0.00, 0.00 ]

//[ 90.00, 0.00, 90.00 ]

// if we move by mose

//[ 3.90, 5.65, 5.99 ]

//[ 57.80, 0.00, 22.10 ]
```

用户定义的函数

对于代码的可读性和再定义一个函数。

```
my_d=20;
function r_from_dia(my_d) = my_d / 2;
echo("Diameter ", my_d, " is radius ", r_from_dia(my_d));
//recursion - find the sum of the values in a vector (array)
// from the start (or s'th element) to the i'th element - remember elements are
zero based
function sumv(v,i,s=0) = (i==s ? v[i] : v[i] + sumv(v,i-1,s));
vec=[ 10, 20, 30, 40 ];
echo("sum vec=", sumv(vec,2,1)); // is 20+30=50
```



，

echo

这个函数输出内容的编辑窗口，利用的是 BASH 的 shell 语言，echo，回声的意思，就是你输入什么，他就显示什么。。代码调试用。参见字符串函数 str()。

Render 渲染

即使在预览模式的强制生成网格。用布尔运算变得太慢的轨道。

我这个软件无法运行这一项。

Surface
表面

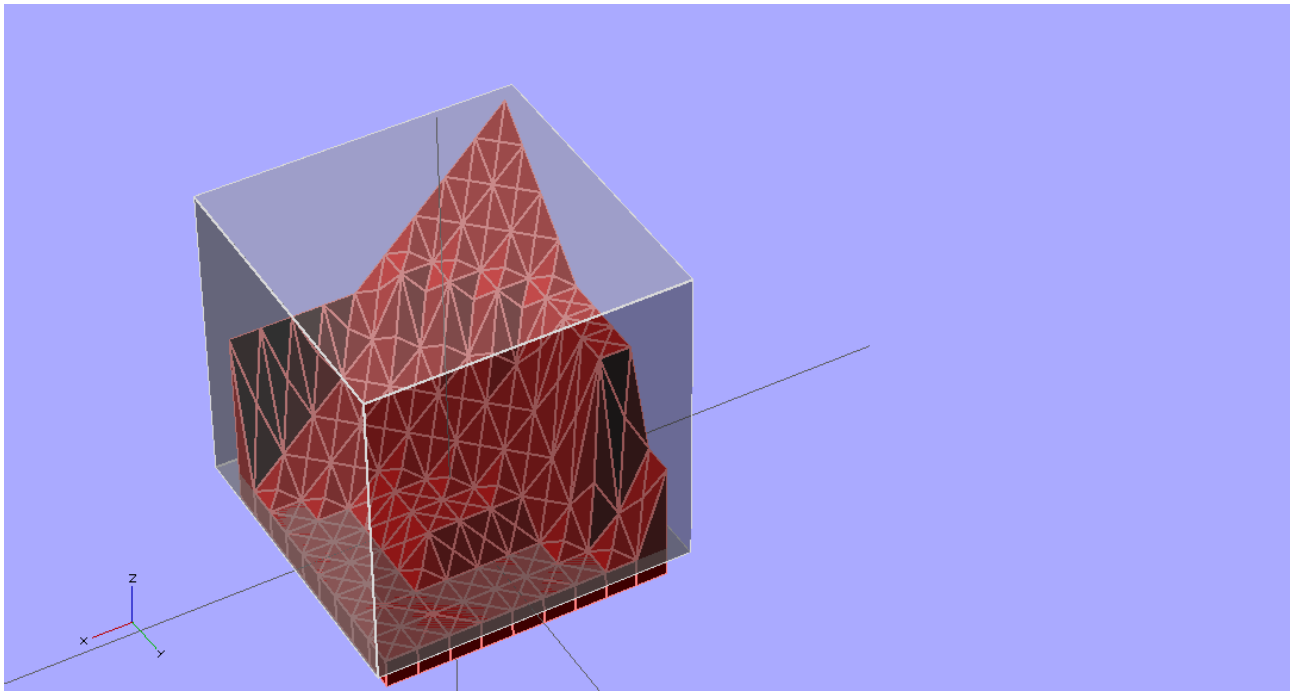
我们首先要用文本编辑器 nano，VI，gedit，geany，emacs，etc...等，把一下数据编辑内，保存为 sf.dat 格式，同时在同一个文件夹新建一个 sf.scad 的文件，
代码内容如下：

```
//sf.scad
color("red")
surface(file = "sf.dat", center = true, convexity = 5);
%translate([0,0,5])cube([10,10,10], center =true);
```

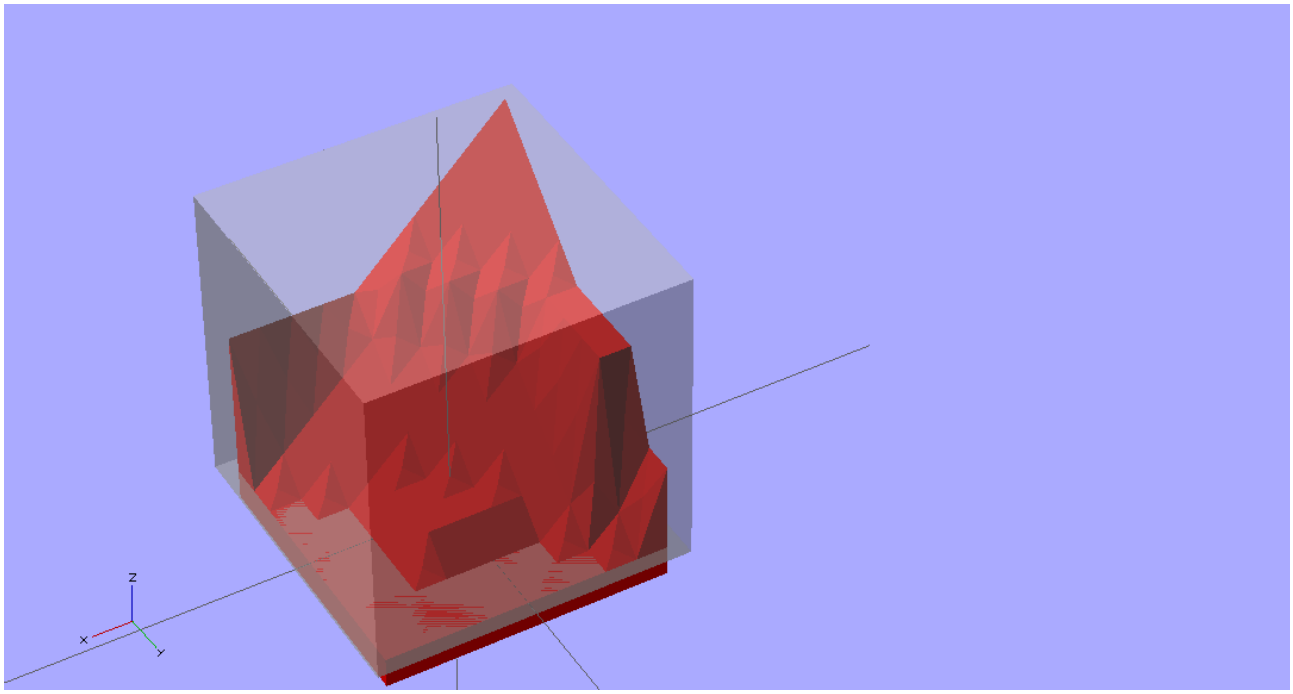
文件 sf.dat 格式代码内容如下：

```
#sf.dat
10 9 8 7 6 5 5 5 5 5
9 8 7 6 6 4 3 2 1 0
8 7 6 6 4 3 2 1 0 0
7 6 6 4 3 2 1 0 0 0
6 6 4 3 2 1 1 0 0 0
6 6 3 2 1 1 1 0 0 0
6 6 2 1 1 1 1 0 0 0
6 6 1 0 0 0 0 0 0 0
3 1 0 0 0 0 0 0 0 0
3 0 0 0 0 0 0 0 0 0
```

因为模型比较小，所以要调整模型的视图比例，直到合适为止。
Ctrl+l,显示边界。

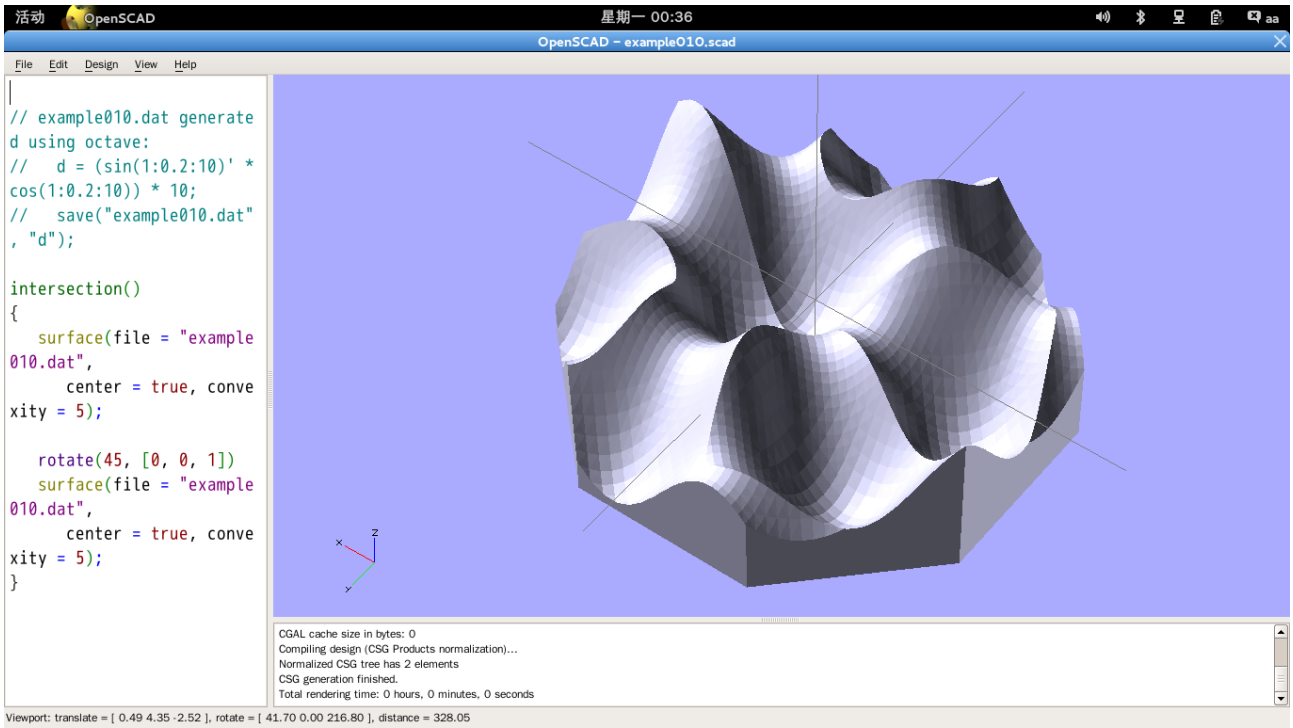


Version 1,



version 2.

打开'File"-> "Examples" -> "example-010.scad"，打开后，F5 编译，直接生成一个数学模型的曲面，如图。



search

该 search()函数是一个通用函数来找到一个或多个（或所有）事件

在向量的值或值的列表，列表中的字符串或更复杂的构造。

搜索使用

搜索 (match_value，string_or_vector [，] index_col_num num_returns_per_match [，]) ；

搜索参数

・match_value

・可以是单一值或值的向量。

・字符串作为字符遍历搜索功能的载体；

不搜索子串。

・注意：如果 match_value 是一个向量的字符串，搜索查找的字符串精确

比赛。

・看下面的例子 9。

・string_or_vector

・的字符串或向量搜索匹配。

・num_returns_per_match (默认值：1)

・默认情况下，搜索寻找返回每个元素的 match_value 比赛

作为一个列表的索引

・如果 num_returns_per_match > 1，搜索返回列表上列出的

为每个元素的索引值 match_value num_returns_per_match。

・看下面的例子 8。

62

・如果 num_returns_per_match = 0，搜索返回一个列表，列出所有匹配

为每个元素的索引值 match_value。

・看下面的例子 6。

・index_col_num (默认值：0)

・当 string_or_vector 是一个向量的向量表，多维或多

列表的列表的构造复杂，match_value 可能不会在第一个发现

(index_col_num = 0) 柱。

・看下面的例子 5 为一个简单的使用例子。

搜索使用的例子

看到 example023.scad 包括一可渲染的例子 OpenSCAD。

指数值返回列表

Example
Code

```
search("a","abcdabcd");
1
Result
[0]
2 search("e","abcdabcd"); []
3 search("a","abcdabcd",0); [[0,4]]
4 search("a",[ ["a",1],["b",2],["c",3],["d",4], [[0,4]] (see also
 ["a",5],["b",6],["c",7],["d",8],["e",9] ], 0); Example 6 below)
```

Search on different column; return Index values

Example 5:

```
search(3,[ ["a",1],["b",2],["c",3],["d",4],["a",5],["b",6],["c",7],["d",8],
["e",3] ], 0,
1);
```

Returns:

```
[2,8]
```

Search on list of values

Example 6: Return all matches per search vector element.

```
search("abc",[ ["a",1],["b",2],["c",3],["d",4],["a",5],["b",6],["c",7],["d",8],
["e",9] ],
0);
```

Returns:

```
[[0,4],[1,5],[2,6]]
```

Example 7: Return first match per search vector element; special case return vector.

```
search("abc",[ ["a",1],["b",2],["c",3],["d",4],["a",5],["b",6],["c",7],["d",8],
["e",9] ],
1);
63
```

Returns:

```
[0,1,2]
```

Example 8: Return first two matches per search vector element; vector of vectors.

```
search("abce",[ ["a",1],["b",2],["c",3],["d",4],["a",5],["b",6],["c",7],["d",8],
["e",9] ],
2);
```

Returns:

```
[[0,4],[1,5],[2,6],[8]]
```

Search on list of strings

Example 9:

```
1Table2=[ ["cat",1],["b",2],["c",3],["dog",4],["a",5],["b",6],["c",7],["d",8],
["e",9],
["apple",10],["a",11] ];
1Search2=["b","zzz","a","c","apple","dog"];
12=search(1Search2,1Table2);
echo(str("Default list string search (",1Search2,"): ",12));
```

Returns

```
ECHO: "Default list string search
([\"b\", \"zzz\", \"a\", \"c\", \"apple\", \"dog\"]):
[1, [], 4, 2, 9, 3]"
```

Getting the right results

```
// workout which vectors get the results
v=[ ["O",2],["p",3],["e",9],["n",4],["S",5],["C",6],["A",7],["D",8] ];
//
echo(v[0]);
// -> ["O",2]
echo(v[1]);
// -> ["p",3]
```

```
echo(v[1][0],v[1][1]);
// -> "p",3
echo(search("p",v));
// find "p" -> [1]
echo(search("p",v)[0]);
// -> 1
echo(search(9,v,0,1));
// find 9 -> [2]
echo(v[search(9,v,0,1)[0]]);
// -> ["e",9]
echo(v[search(9,v,0,1)[0]][0]);
// -> "e"
echo(v[search(9,v,0,1)[0]][1]);
// -> 9
echo(v[search("p",v,1,0)[0]][1]);
// -> 3
echo(v[search("p",v,1,0)[0]][0]);
// -> "p"
echo(v[search("d",v,1,0)[0]][0]);
// "d" not found -> undef
echo(v[search("D",v,1,0)[0]][1]);
// -> 8
```

OpenSCAD Version

version() and version_num() will return OpenSCAD version number.
- The version() function will return the OpenSCAD version as a vector, e.g. [2011, 09, 23]
- The version_num() function will return the OpenSCAD version as a number, e.g. 20110923

OpenSCAD 用户手册/使用 2D 子系统

2D 子系统部分根据以上的内容，基本自己就可以理解了。

不再具体翻译了，就看自己领悟了。

CAD 的图纸是没有国界和语言张障碍的。