M3D 宏包手册

Anthony Phan

2006 年 12 月 18 日

引论

在九十年代（刚过去的世纪）后半段时候，我发现 TEX, MetaFont，然后是 MetaPost。我确实着迷于最后的那个工具，因为它用一种类 MetaFont 的语言生成图形，且允许非常简单图形包含到 TEX文档中去。我在 3D 图形中的用 MetaPost 的最初尝试很简单：投影系统是纯粹的刚体，图形是由一些线和标签以及仅仅 4 个平坦的面填上固定的颜色。于是我觉得必须要一个参数化的标架，因为我们并不总是知道一个图形是否已经从一个合适的角度观察，这或许富有意义。从而最首要的步骤完成：有了一个参数化的标架，处理空间坐标，如果它们在屏幕的投影是定向良好的，就画出轮廓。

然后我听说 Denis Roegel 的 “m3D” 宏包。它的语法不适合我在 MetaPost 中已有的 3D-程序的想法，但它的特性便利于从 `eps` 转换到 `gif` 和动画。我偷学了这个动画设备，自娱以及为我的图形寻找好的角度。为了我的网页插图，我开始了更为复杂的设计 ……

我的中心思想是尽可能保持与通常的 MetaFont 和 MetaPost 程序靠拢，但也考虑到所有的 3-维对象如此的复杂性，应当用上一些非常有技巧的代码。因此我转而思考如何尽善尽美地得到一个稳定的，功能强大的基本程序，以及在其图库中加入的一些通常对象。于是一些相对复杂的对象现在可以通过这些相对基本的对象通过移动，旋转以及放大而得到。当然，我关于对象的想法与面向对象-编程无关，它只是关于刚体朴实的观念而已。

如此的计划所能达到的目标是有限的，这个 “3D” 宏包仍在发展中。当某个朋友问我是否能画出一个比他画得更好的图形时，我很高兴的是能用它去完成。

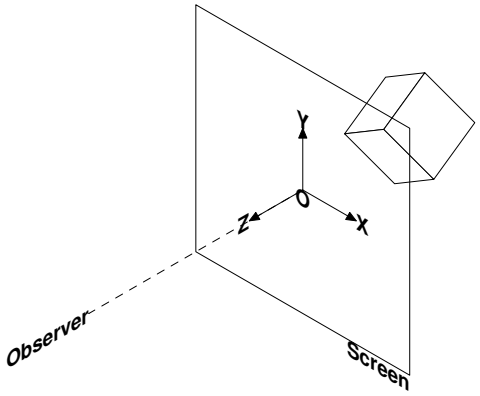
我的其中一个信念是那些 MetaPost 编程着宁愿建立他（她）的宏包系统而不去依赖某人的程序— 特别是当这些程序被声明是不稳定的时候。这些编程者只是简单

地回顾一些语法，一些特别的改进，就公布他（她）自己的程序到网上，用这种方式对其他所有感兴趣于这个程序人给出某些反馈。

1 基本概念

众所周知，MetaPost 提供了标准的设备操控称为颜色的 3 维向量的变量类型：数据类型的检查，加法，减法，数乘。由此看来偏偏缺少了 3 维向量和图形的仿射变换，可是没有人会去抱怨这个，因为 MetaPost 是一个 2-维定向程序语言，因此我们会接受用一些高层次的控制序列来完成这些任务。

2 坐标系统



3 欧拉角

给出正交的标架 $(\mathbf{Ox}, \mathbf{Oy}, \mathbf{Oz})$ 和 $(\mathbf{Ox}', \mathbf{Oy}', \mathbf{Oz}')$, 欧拉角 (θ, ϕ, ψ) 是满足

$$\begin{cases} \mathbf{Ox}' &= \cos\phi\cos\theta \times \mathbf{Ox} + \cos\phi\sin\theta \times \mathbf{Oy} + \sin\phi \times \mathbf{Oz} \\ \mathbf{Oy}' &= -(\cos\psi\sin\theta + \sin\psi\sin\phi\cos\theta) \times \mathbf{Ox} \\ &\quad +(\cos\psi\cos\theta - \sin\psi\sin\phi\sin\theta) \times \mathbf{Oy} + \sin\psi\cos\phi \times \mathbf{Oz} \\ \mathbf{Oz}' &= (\sin\psi\sin\theta - \cos\psi\sin\phi\cos\theta) \times \mathbf{Ox} \\ &\quad -(\sin\psi\cos\theta + \cos\psi\sin\phi\sin\theta) \times \mathbf{Oy} + \cos\psi\cos\phi \times \mathbf{Oz} \end{cases}$$

的实数组，它们可以由方向 $\mathbf{Ox}'$ 按照 $(\mathbf{Ox}, \mathbf{Oy}, \mathbf{Oz})$-坐标系表示的相关知识确定— 我们得到 θ 和 ϕ，而由 $(\mathbf{Ox}', \mathbf{Oy}')$ 生成的平面表示出一个通过 $(\mathbf{Ox}, \mathbf{Oy}, \mathbf{Oz})$ -坐标系表示的落在 $(\mathbf{Ox}', \mathbf{Oy}')$-平面中的一个向量，它与 $\mathbf{Ox}'$ 不共线— 我们得到最后的 ψ.[1]

[1]此段关于 θ, ϕ 的叙述与球面坐标相混淆了，与通常书上关于欧拉角的叙述不符，请查看理论力学课本中相关的内容— 译注

因此，一个称为 Angles 的控制序列在 m3Dplain.mp 中定义了。它的参数是两个向量（或颜色），称为 p 和 q, 它们按下面的途径返回相应于欧拉角的三元组（或颜色），称为 (θ, ϕ, ψ): 如果 p 为零或者是一个非常小的向量，它返回 $(0, 0, 0)$; 否则，θ 和 ϕ 按使得 $\mathbf{Ox}'$ 与 p 具有相同的方向计算；接着，如果 q 看起来与 p 非常接近于共线，ψ 就设为 0, 否则，ψ 设置为帮助定义整个 $(\mathbf{Ox}', \mathbf{Oy}', \mathbf{Oz}')$-正交标架合适的数值，以使得 (p, q)-向量平面等价于 $(\mathbf{Ox}', \mathbf{Oy}')$-向量平面且 $\mathbf{Ox}'$ 与 p 有同一方向的。

4 投影系统

5 演示参数

有很多参数被 m3D 采用了，描述如下。对不建议改动的参数我们打上一个剑号(†), — 很大程度上是为了审美这个理由— 在一幅图中。

- ObsZ:=内部数值 †. 这是屏幕或纸张与观察者之间的距离。它可以带上长度单位。
- Resolution:=内部数值 †. 一些预定义的对象用这个参数来决定画图的步骤数。它可以带上长度单位。
- LightSource:=颜色 †. 它是光源的位置。
- LightAtInfinity:=布尔量 †. 它指示光源是空间中真正的点还是方向。
- Luminosity:=内部数值 †. 这是入射光的强度，它在 0 到 1 之间。
- Contrast:=内部数值 †. 这是通常的对比度参数，它在 0 到 1 之间。
- Specularity:=内部数值。它表示由物体反射的入射光线的比例。它在 0 到 1 之间且在同一图中可以改变，以便粉刷出不同的质感。
- Phong:=内部数值。它表示反射光线在物体展布的量（光泽），它是一个小整数，而且可以在同一图中改变，以便粉刷出各类不同的质感。
- Fog:=内部数值 †. “雾”可以用来粉刷物体。值为 0 意味着没有雾，1 为以指衰减的线性雾，2 为以指数衰减的球状雾。
- FogHalf:=内部数值 †. 它是雾的指数衰减率。它是一个带上度量单位的正数。
- FogZ:=内部数值 †. 它是当雾还可见时相对屏幕往下的 z-坐标。它可以带上度量单位。
- FinePlotFlag:=布尔量。这个布尔量用于某些控制序列，比如 Plot3D。
- mthreeDfont:=字符串 †. 用于 3 维空间中的文本的字体名称。
- ShadedTextFlag:=布尔量。这个布尔量指示了在 3 维空间演示一个文本是否已经用上了特殊效果。

- ObjectColor:=颜色。这是在空间中填充一个面用的颜色。它可以在任何时候改变，以便使得图形有丰富的颜色。

6 关于光源

在 m3Dplain.mp 只定义了一个光源。它可以设置在无穷远（ LightAtInfinity:=true）或场景中的一些点（ LightAtInfinity:=false）。在所有的情形中它的坐标 LightSource 用于整个屏幕标架且当对象平移或旋转时不会改变，也就是对观察者来说光是固定的。

如果希望光源和特定的对象关联，我们得在对象的中定义写上诸如

LightSource:=GDir(x,y,z)

如果光源是座落在无穷远；或者

LightSource:=GCoord(x,y,z)

如果光源是座落在空间中的某一点（ (x,y,z) 是该处的局部坐标）。我们注意到这是控制序列 GDir 和 GCoord 的使用中非常仅有的方向。

如果需要多个光源，我们得重新定义 m3Dplain.mp 的控制序列 Light （由于它已经是沉重的机构，所幸能行得通）

7 从外面，里面看物体

通常，观察者从物体的外面看物体。因此，如果他们的面的定向轮廓的投影呈现出正的定向，则该面要画出。这就是为什么控制序列 Orientation 默认值是 Outside. 相反的情形我们可以用

Inside;

然后可以用

Outside;

回到默认情形。还可以尝试下面的

OutsideIn; 或 InsideOut;

在这种情形下，面总是画出。两者之间的区别是后者的光线效果恢复原状。如果希望达到更加奇异的效果，他(她)可以摆弄在 m3Dplain.mp 中作为基础而写进的 Orientation 定义和 Orienation_ 数值。

8 顺序及消隐

画单独一个凸体是一件容易的事情：只要画出所有轮廓可见的面，投影，当画它的外部（或内部）边缘时作为一个正的（或负的）定向路径。因此，在这种情况下消隐面不成问题。内和外的转换可以由 Inside 和 Outside 控制序列。它们简单地使得定向的条件反过来。

当处理非凸体的时候，我们不得不分解这个物体为凸的几部份，然后以合适的顺序画出。因此，我们一个相当有技巧性控制序列称为 QuickSort 已经为这个目的而设计了。它是一些必须包括至少两项的文字论述且它的输出是一个称为 SortedList 的控制序列，它的内容是先前的列表并被整理为依照 SortCriterion. SortCriterion 是一个包括两个论断（将要被分类的对象列表），它们替代文字是一个布尔量。默认情况下，这个论断是三元组且这个条件是关于它们实际上的相对当前观察者的深度。因此 QuickSort(三元组的列表) 将输出 SortedList ，它的替代文字正好是期望的三元组列表顺序。我们要想改变的话，只须改写 SortCriterion ，以便对数值，二元组，字符串分类。

如果我们依照空间中点的列表的深度执行一系列动作，那么按照这个方法去分类将会是优雅的。一个自然的做法是采用下面的过程：

OnDepth;

Refpoint 三元组;

Action (界定的控制序列)

这个工作程序如下：在 OnDepth 阶段保存和重设一些东西；使 Action_counter 递增，以及保存那些在 Refpoint 援引的当前参考点（一个三元组）；保存界定的控制序列到一个可变的控制序列以当前 Action_counter 计数。(此处有一个小小的技巧我寻觅了很长很长一段时间)；在 endOnDepth 依照参考点的的深度安排 1,..., Action_counter 这些列表，然后依照存储的列表执行动作。在这个进程中最为有趣的是动作可以依赖一些参数，就像循环或宏包参数。注意到 SortCriterion 当执行 endOnDepth 有一个特殊和临时的涵义: 它的两个论断便成为一些在 1,...,Action_counter 中的指示在两个对应参考点深度比较。

9 整合文本

很清楚，为了得到好看和有趣的图，有时需要集成一些文句在三维空间，正如我曾经做过一两次让文本环绕着图形。这是比较特别的。为此定义一些一般的方案对我来说是无意义的：它非常复杂，很难设想人们愿意去做。无论如何，一些控制序列允

许移动平坦的文本绕圈会是一个要素。而且，象这样基本的程序可以有助于设计特殊的控制序列用于复杂的任务。

9.1 简单文本

一个被称为 simpletext 的对象在 m3Dplain.mp 定义了。它的特殊参数的组成，首先是一些字符串描述定位("left", "justify", "center", "right"), 然后是一些告知文本的参考点在何处（特别是 "right", "urt", "top", "ulft", "left", "llft", "bot", "lrt" 或[say!] "center", 那么，一列字符串就会一个叠一个地呈现出来。

比如，在顶部许多层将会展示（更进一步看关于放大参数）

```
UseObject(simpletext, Origin, (90, 0, 90), 10pt, "justify", "ulft",
                "Come let me sing into your ear;",
                "Those dancing days are gone,",
                "All that silk and satin gear;",
                "Crouch upon a stone,",
                "Wrapping that foul body up",
                "In as foul a rag:",
                "I carry the sun in a golden cup;",
                "The moon in a silver bag.");
```

J.B. Yeats 的诗 “Those dancing days are gone” 的第一部分，在基点 Origin 上，依照标架旋转了 $(90, 0, 90)$, 放大了 10 pt. 如果可以的话，每一行将会被调整，基点将参照文本的左上角。

这个 simpletext 对象采用由字串 mthreeDfont命名的字体（默认值: "rphvb"）。它的设计尺寸由被称为 mthreeDfontsize（默认值是: 10pt）数值变量定义。数值参数 baselineskip 按通常角色（默认值是: 12pt）。这些参数仅仅与与字体相关，与 CurrentScale无关。因此，当用 Simpletext 到一个对象时，注意尺寸的叙述要像

```
UseObject(simpletext,原点,欧拉角,尺寸, 列出字串,字串方位,字串列表)
```

因为尺寸必须是一个局部尺寸。

调整是通过伸展字体的常规空间的宽度到 TextStretchFactor 得到的。(默认值 2 已经相当大)。逐次地进入句子的空间：它们不会缩小为一个单独的空间。

每一字母分别画出只是为了保证当投影不是线性时，作用在字母上的仿射变换近似正确。 由于每一个字母有一个特殊的规模，我们必须设 prologues 为 1 或 2 以便不会超出 MetaPost 的能力（全体字体只会简单地在 eps 图形的开始声明且不是每个字母采用它自己的变换）。 当然，我们可以为此采用 PostScript™ 字体。这就是为什么我们已经引进 mthreeDfont 缘故。

9.2 弯曲文本

改进中，还没有公布。

评论 — 记住在图形注释仍然可以采用通常的控制序列就像

```
label.方位(标签, proj(x, y, z))
```

诸如 simpletext 或 curvedtext 是为了相当特殊的效果。

10 动画

10.1 引论

Denis Roegel 论证了用通常的 Unix 工具合并一系列 MetaPost 输出成一个 GIF 动画（3D 宏包）。我从他的 "metapost to shell" 脚本中学了不少。想法如下是：首先，跟进所有的输出的 eps 的最大边框界限；然后转换这些 eps 输出为以这个最大边框为界限的 PostScript™ 或 eps 文件；分别转换这些最后得到的文件为单一的 GIF 图像；然后再合并所有这些图像为一个动画。

10.2 在 m3D 如何做

在文件的末尾将会生成一个额外的脚本，它的名字默认是 animate-script 。一旦 MetaPost 的工作完成，在类 Unix 系统面板的终端（ xterm）的当前目录中执行

```
bash animate-script
```

最后的输出是 jobname.gif ，此处 jobname 是 MetaPost 程序确实的名字。

10.3 额外程序

这个脚本需要下列程序: sed 和 convert。我们选用 sed 来改变所有 MetaPost 输出的边界参数— 所得到的临时文件命名为 jobname.xxx.eps，此处 jobname.xxx 为 MetaPost 输出文件名之一。这个 PostScript™ 到 GIF 的转换由在 Roegel 的宏包中的 Netpbm Library 执行且它们的合并到一个动画中是由 gifmerge（一个非标准但非常漂亮的 Unix 程序，可以在网上自由获得）。最近几年，ImageMagick（开始版权归 Dupont de Nemours，后来是 ImageMagick Studio，但实际上是非常自由）已经散布到几乎所有的 Linux 发行版本中。它是一个把任意东西转成所有东西甚至动画的高质量工具。由前面脚本引用的其中一个基本的控制序列是 convert。

10.4 细节

下面提供更为详细的解释。

- Animate(数值,布尔量或颜色)
- AnimateScipt 字符串变量，由 Animate 输出的（bash）脚本的名字。
- AnimateFormat 字符串变量，动画像的格式。默认值是 "gif" ，但可以改变，比如， "mpg" 或 "mng".这个格式必须是 convert 命令能辨认的，或其他的将可得到的程序（针对 "mpg" 格式的 mpg2encode）。
- AnimateQuality 数值变量，特别地可以取值为 1，2，4 ...
- AnimateDelay 数值变量，在动画中每帧 1/100 秒时间。
- AnimateLoop 数值变量，动画的参数，它的默认值等于 0（无限次循环播放）。
- compute_bbox 以及 xmin_, xmax_, ymin_, ymax_ 已经在前面解释过。

11 直接输出为 eps

```
DirectEPS 文件名;
......
endDirectEPS;
```

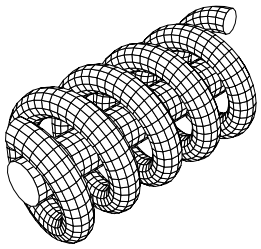
12 一些例子

第一张图由相当于 TechnoFill 的 Fill 生成（我某天得换掉这个名字），接下去的一个由相当于 WireFill 的 Fill生成（这种图形相当的传统）。我已经加入了一个文本（W.B.Yeats）来测试 simpletext 对象。在第二张图中有一个圆柱，但更为重要的是一个称为 tube 的对象的用法：通过 $x(t), y(t), z(t)$, 一个半径 r, 一个 t 的范围给出的空间中的一条路径，我们可想像得出这个对象。然而运算非常易碎（二阶）且可能导出意料之外甚至丑陋的效果。如果对象 cylinder 在 m3Dlib01.mp 定义，那么 tube 在 m3Dplain.mp 就是因为我觉得它是一个基本的工具。

```
let Fill = SolidFill;%TechnoFill;
ShadedTextFlag := true;
TextColor := red;
beginfig(thisfig);
  interim prologues := 1;
  OnDepth;
    Refpoint(1,0,0);
    Action
    (UseObject(etube, Origin, (0, 90, 0), 1cm,
```

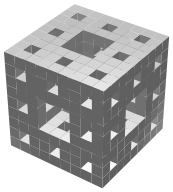
```
  "(cosd(t*90), 0, t)", 0.25, -3, 0, true, false));
    Refpoint(-1,0,0);
    Action
    (UseObject(etube, Origin, (0, 90, 0), 1cm,
  "(cosd(t*90), 0, t)", 0.25, 0, 3, false, true));
    for a = 0 step 45 until 315:
      Refpoint Dir(a+95,0);
      Action
      (UseObject(simpletext, Origin, (a+95, 0, 90), 5pt, "left", "ulft",
    "Come let me sing into your ear;",
    "Those dancing days are gone,",
    "All that silk and satin gear;",
    "Crouch upon a stone,",
    "Wrapping that foul body up",
    "In as foul a rag:",
    "I carry the sun in a golden cup;",
    "The moon in a silver bag."));
    endfor
  endOnDepth;
endfig;
```


```
let Fill = WireFill;
beginfig(thisfig);
  ObjectColor := (1, 215/255, 0);
  for i = -2 upto 2:
    UseObject(tube, Origin, (0, -90, 0), 0.75cm,
      "(cosd(t*360), sind(t*360), t)", 0.25, i-0.5, i,
      if i = -2: true else: false fi, false); endfor
  ObjectColor := 0.5white;
  UseObject(cylinder, (-2.5, 0, 0)*0.75cm, (0, -90, 0), 0.75cm, 0.4, 5);
  ObjectColor := (1, 215/255, 0);
  for i = -2 upto 2:
    UseObject(tube, Origin, (0, -90, 0), 0.75cm,
      "(cosd(t*360), sind(t*360), t)", 0.25, i, i+0.5,
      if i = 2: true else: false fi, false); endfor
endfig;
```

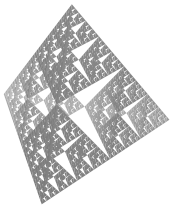


这里有两个 Sierpinski--Menger 对象：海绵（对象命名为 `sierpinskip_sponge`）和垫圈（对象命名为 `sierpinskip_gasket`）。这个海绵由等价于 `SolidFill` 的 `Fill` 画出，而垫圈由等价于 `SolidWireFill` 的 `Fill` 画出。两个对象都在 `m3Dlib01.mp` 中定义。这个垫圈— 由于以 4^n 增长，此处 n 是循环的阶数— 比画海绵容易得多— 后者以 20^n 增长。在我当前的 MetaPost 的实现中对海绵达到 3 阶不是一件容易的事情。

```
ObjectColor:=0.75white;
let Fill = SolidFill;
beginfig(thisfig);
  UseObject(sierpinski_sponge, Origin, (10,0,0), 1.5cm, 2);
endfig;
```

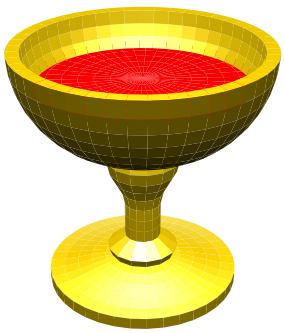


```
beginfig(thisfig);
  UseObject(sierpinski_gasket, Origin, (30,0,0), 1.5cm, 5);
endfig;
```



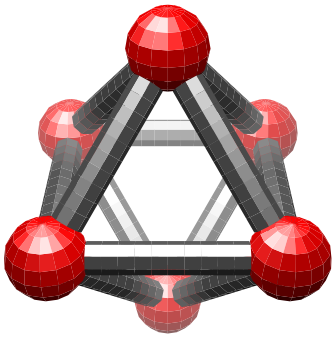
```
let Fill = SolidFill;
beginfig(thisfig);
  save u; u=3cm;
  ObjectColor := (1, 215/255, 0);
  ObjectPath :=
  (eps, 0.65){right}
  ...{up}(0.45, 1)
  --(0.5, 1){down}
  ...{left}(0.1,0.6)% bowl
  {right}...{down}(0.15,0.55)
  ...{down}(0.075,0.35){down}
  ...{down}(0.075, 0.2)
```

```
  ...(0.15,0.15){down}
  ...{left}(0.1,0.1){right}
  ...{right}(0.4,0.05)
  --(0.4,0){left}
  ...{left}(eps,0.05);
  OnDepth;
    % contents
    Refpoint (0,0,0.8u);
    Action (ObjectColor := red;
      UseObject(revolution, (0, 0, 0), Origin, u,
  (eps, ypart point 0.8 of ObjectPath)..point 0.8 of ObjectPath);
      ObjectColor := (1, 215/255, 0););
    % bowl
    Refpoint (0,0,u);
    Action (UseObject(revolution, (0, 0, 0), Origin, u,
  subpath (0.75,3) of ObjectPath););
    % stem
    Refpoint (0,0,0.5u);
    Action (UseObject(revolution, (0, 0, 0), Origin, u,
  subpath (3,8) of ObjectPath));
    % foot
    Refpoint (0,0,0.25u);
    Action (UseObject(revolution, (0, 0, 0), Origin, u,
  subpath (8,11) of ObjectPath));
  endOnDepth;
endfig;
```

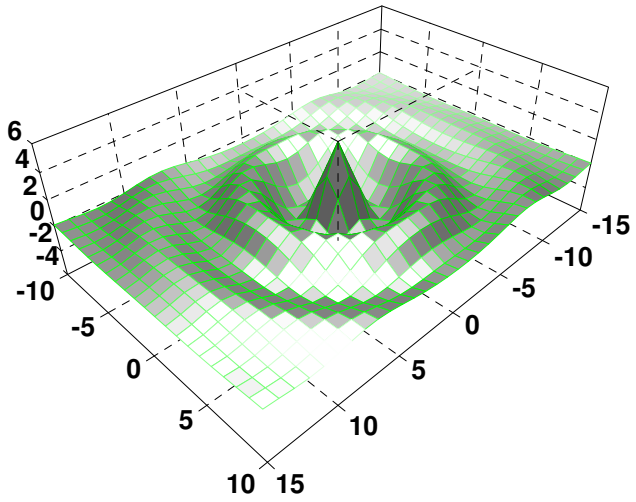


```
Object molecule =
  M1 = (0, 0, 1); M2 = (1, 0, 0); M3 = (0, 1, 0);
  M4 = (-1, 0, 0); M5 = (0, -1, 0); M6 = (0, 0, -1);
  save srad, lrad, j; srad := 0.25; lrad := 0.1;
  OnDepth;
    for i = 1 upto 6:
      Refpoint M[i];
      Action(ObjectColor:=red;
        UseObject(sphere, M[i], Origin, srad));
    endfor
    for i = 1, 6:
      for j = 2, 3, 4, 5:
        Refpoint 0.5[M[i], M[j]];
        Action(ObjectColor := 0.375white;
```

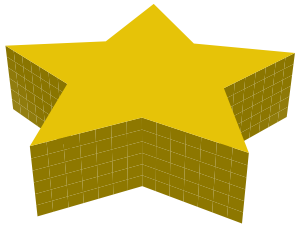
```
          SpheresLink(M[i], M[j], srad, srad, lrad));
      endfor
    endfor
    for i = 2 upto 5:
      Refpoint 0.5[M[i], M[if i = 5: 2 else: i+1 fi]];
      Action(ObjectColor := 0.375white;
        SpheresLink(M[i], M[if i = 5: 2 else: i+1 fi], srad, srad, lrad));
    endfor
  endOnDepth;
endObject;

let Fill = SolidFill;
beginfig(thisfig);
  UseObject(molecule, Origin, Origin, 2cm);
endfig;
```



```
let Fill = SolidWireFill;
ObjectColor := 0.5white;
PenColor := green;
%FinePlotFlag:=true;
beginfig(thisfig);
  interim prologues := 1;
  pickup thin.nib;
  Euler(0,0,0,0.2cm);
  Frame(-15 step 5 until 15)(-10 step 5 until 10)(-4 step 2 until 6);
  Plot3D("4cosd(180/3.14159*(x++y))*mexp(-(x++y)*50)", -15, 15, -10, 10);
  FrameMark (0,0,4);
  endFrame;
endfig;
```



下面例子展示了在 m3Dplain.mp 中定义的被称为 cylinderlike 对象的用法。



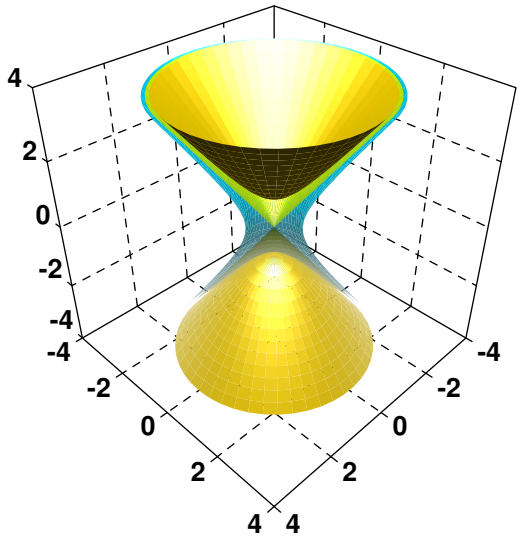
它的特殊参数是一个 xOy-封闭路径以及柱形的高度。因此，前面的图形可以如下得到。

```
let Fill = SolidFill;
ObjectColor := (1, 215/255, 0);
beginfig(thisfig);
  UseObject(cylinderlike,(0,0,0),(-210,0,0),1cm,
    for i= 0 upto 4:
      2dir(i/5*360)--dir((i+0.5)/5*360)--
    endfor cycle, 1);
endfig;
```

它相当简单。然后是 revolution 对象的一个复杂例子。

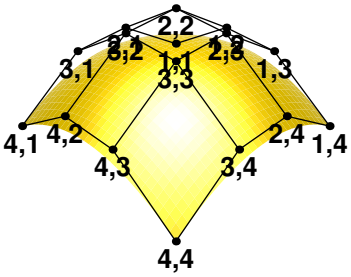
```
beginfig(thisfig);
  interim prologues:=1;
  Euler(0,0,0,0.5cm);
  save h, dh, r; h=4; dh = 1; r=0.75;
  Frame(-4 step 2 until 4)
    (-4 step 2 until 4)
    (-4 step 2 until 4);
  Inside;
  % hyperboloide  une nappe
  ObjectColor := (0, 215/255, 1);
  UseObject(revolution,Origin,Origin,1,
    (r*sqrt(h*h+1), h){-r*h/sqrt(h*h+1),-1}
    for y = h-dh step -dh until -h-eps:
      ...(r*sqrt(y*y+1), y){-r*y/sqrt(y*y+1), -1} endfor);
```

```
  % cone
  ObjectColor := (215/255, 1, 0);
  UseObject(revolution,Origin,Origin,1, (r*h, h)--(eps, 0)--(r*h,-h));
  % hyperboloide  deux nappes
  ObjectColor := (1, 215/255, 0);
  for side = "Inside", "Outside":
    scantokens side;
    UseObject(revolution,Origin,Origin,1,
      reverse((eps, 1){right} for y = 1+dh step dh until h+eps:
    ...(r*sqrt(y*y-1), y){r, sqrt(y*y-1)/y} endfor));
  endfor
  UseObject(revolution,Origin,Origin,1,
    (eps, -1){right} for y = 1+dh step dh until h+eps:
      ...(r*sqrt(y*y-1), -y){r, -sqrt(y*y-1)/y} endfor);
  endFrame;
endfig;
```



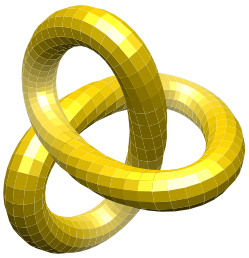
以及一个 Bezier 由控制序列 `BezierPatch` 得到的分片曲面（当且仅当 `tracingchoices` > 0 时才画出控制点。）

```
tracingchoices:=1;
beginfig(thisfig);
  interim prologues := 1;
  interim CurrentScale := 1.5cm;
  BezierPatch(10,
    (-1,-1,-0.5), (-1,-0.3,0), (-1,0.3,0), (-1,1,-0.5),
    (-0.3,-1,0), (-0.3,-0.3,0.5), (-0.3,0.3,0.5), (-0.3,1,0),
    (0.3,-1,0), (0.3,-0.3,0.5), (0.3,0.3,0.5), (0.3,1,0),
    (1,-1,-0.5), (1,-0.3,0), (1,0.3,0), (1,1,-0.5));
endfig;
```



被称为 tube 的对象可以非常简单地处理一些像画扭结的复杂情形：

```
Ox:=(1,0,0);
Oy:=(0,1,0);
Oz:=(0,0,1);

beginfig(thisfig);
  UseObject(tube,Origin,Origin,0.5cm,
    "(cosd(t)+2cosd(2t), sind(t)-2sind(2t), 2sind(3t))",
    0.5, 360, 0, false, false);
endfig;
```



这就是为什么关于这个对象更为复杂的版本: etube(enhanced tube 增强的管道)被定义。

```
beginfig(thisfig);
  UseObject(etube,Origin,Origin,0.5cm,
    "(cosd(t)+2cosd(2t), sind(t)-2sind(2t), 2sind(3t))",
    0.5, 360, 0, false, false);
endfig;
```