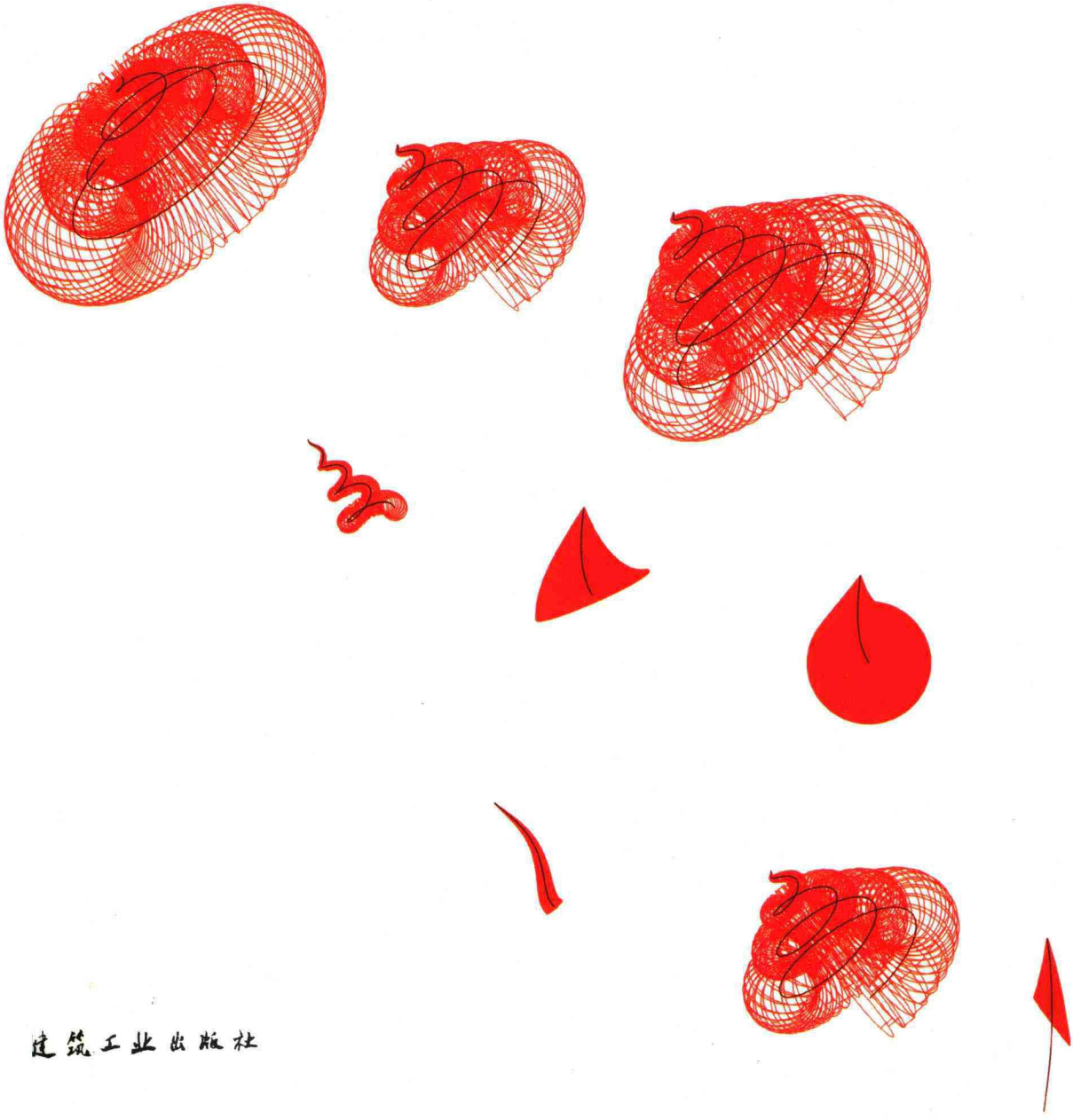
PROGRAMMING.ARCHITECTURE

建筑院校计算机辅助设计译丛

编程·建筑

[英] 保罗·科茨 著
孙澄 姜宏国 刘莹 译



建筑工业出版社

教育部新世纪优秀人才支持计划资助项目（项目编号：NCET-10-0063）
黑龙江省科技攻关计划资助项目（项目编号：GZ10A509）资助

建筑院校计算机辅助设计译丛

编程·建筑

PROGRAMMING. ARCHITECTURE

[英] 保罗·科茨 著
孙澄 姜宏国 刘莹 译

中国建筑工业出版社

教育部新世纪优秀人才支持计划资助项目（项目编号：NCET-10-0063）
黑龙江省科技攻关计划资助项目（项目编号：GZ10A509）资助

著作权合同登记图字：01-2011-7327号

图书在版编目（CIP）数据

编程·建筑/（英）科茨著；孙澄等译．—北京：中国建筑工业出版社，2012.8
（建筑院校计算机辅助设计译丛）
ISBN 978-7-112-14537-9

Ⅰ.①编… Ⅱ.①科…②孙… Ⅲ.①计算机应用-建筑设计 Ⅳ.①TU2-39

中国版本图书馆CIP数据核字（2012）第172399号

责任编辑：戚琳琳　董苏华
责任设计：董建平
责任校对：姜小莲　赵　颖

建筑院校计算机辅助设计译丛

编程·建筑

[英] 保罗·科茨　著
孙澄　姜宏国　刘莹　译

*

中国建筑工业出版社出版、发行（北京西郊百万庄）
各地新华书店、建筑书店经销
北京嘉泰利德公司制版
北京云浩印刷有限责任公司印刷

*

开本：787×1092毫米　1/16　印张：12¼　字数：306千字
2012年9月第一版　2012年9月第一次印刷
定价：45.00元
ISBN 978-7-112-14537-9
（22598）

目　录

怎样阅读这本书

有关本书的网络参考资料和一些其他信息可到下面下划线上的网站查询。

关于本书的主题有一个词汇索引，包含了对于教材内容作进一步讲解的内容或者网络链接。读者可登录下面网站进行查看：

http://uelceca.net/index_and_glossary.htm

如果你的浏览器是最新版本，输入一个字并且右击，浏览器会显示“search Google for...”，通常能够由此连接到 Wikipedia 或者有效的搜索。

本书的印刷包含以下四种字体：

1. 书中的正文部分为宋体。

2. 书中的程序代码为复制的外文书信体，表示一段在语法结构上正确的程序语言。

3. 书中的伪代码采用复制的外文书信斜体，表示一种描述算法但不能在计算机中运行的语言。

4. 书中的引文及图形标注采用楷体。

致谢

本书得以成功编著，要特别感谢以下朋友的鼓励与支持：

为本书的编写工作提供帮助或者提供书中图片的朋友：

- Simon Kim
- Pablo Miranda
- Tim Ireland
- Ben Doherty
- Christian Derix
- Robert Thum（整体结构部分）

为书中实验与仿真提供帮助的朋友：

- Christian Derix——吸引 / 排斥模型
- Pablo Miranda——寻光机器人和群体智能
- Jennifer Coates——Chomsky 的哲学背景和相关建议
- Helen Jackson and Terry Broughton——Isystem/gp evolution
- Bill Hillier——阿尔法语法和语言
- 建筑学硕士课程 Computing and Design/Diploma Unit 6 的所有学生
- Melissa Woolford——组织相关会议来探究发展和促进现实建筑学程序设计方法

另外，还要感谢国家卫生局的医生护士以及他们提供的设备；来自 BIRU Whittington 的 Holly，Bibi，Izzy，Heather 和 Mounia；感谢 Edgeware 社区医院和伊斯灵顿治疗中心。

引言

着眼于中间

简述建筑学研究的本质

在科学研究中往往存在两个对立面，就像在学术界已被认知的 C.P.Snow 的二元分法——科学和艺术。自 Bertallanffy 以来，就出现了第三种途径，他希望将两个对立面融为一体，并把这种思想称为系统理论。

本书的思想是建筑学研究应该定位在任何一组可界定的对立面之间。建造科学已经发展为应用物理学的一个分支，艺术史学则通过法国哲学家转变为批判性理论，这两者都不会对实际建筑设计和设计者产生即刻的影响。

本书主要讨论建筑设计过程中涉及几何学、拓扑学和生成结构的研究。我将会探究最后一个方面，因为在生成结构中有一些合乎主题的内容，这就是所谓“新认识论”。其实，这个新认识论就是一个关于怎样着落在中间点的完美实例。

如果我们着眼于空间和形体的“空间观”，就能够以一些简单数学法则为基础将塑造空间与形体的过程作为设计的一部分。这个过程是近代经由数学与计算机科学方面的实验长期发展形成的理论思想的一部分，例如细胞自动机、集群现象、反应扩散系统以及进化算法。这些视觉形式和空间组织的新途径都表现出“自组织形态”的现象，经常被冠以“涌现”的标题。

这种方法为建筑学提供了一个很好的范式，用很多相互联系的反馈回路、动态关系和底层模型产生的不确定突发结果，替代了静态几何和传统还原。同时，这将产生一些新的认识——群体的共识——来替换设计者的探索。

这本书旨在解释这些系统作为利用这些模型探究空间怎样形成群和群怎样产生空间；广场和市场，哪一个先产生？我们可以将变化的环境看做信息的携带者，正如建筑改变环境那样，变化的环境改变了建筑的职能；把建筑看做空间占用的一个过程。

算法

现如今，计算机学科中算法的应用越来越普遍。很多词汇，例如“过程”衍生出的词汇“设计过程”或者“建筑过程”，都可以被用来描述一些软件的应用。这也就意味着，着眼于中间的思想可以看做一系列设计过程的中间工作。传统的建筑设计过程是随着时间逐渐进行的，而我们要探究的就是摆脱时间的限制。对于一个动态的系统，我们可以通过运行算法来预知设计的结果。

因此，我们可以利用图、动作以及各种绘图来设计算法，但需要将操作、问题描述以及问题的目标紧密地联系在一起。当把算法表示成语言时，算法的描述相对于问题目标的距离进一步加大，这种抽象成语言的做法就会涵盖许多种的语法规则。而在语法中普通意义上的语法和结构语法哪个重要一直存在争议，但前者作为程序文本的基础要比后者更加开放。

因此算法通常以某种语言的程序文本的形式表达。这个程序文本是要用某种语言来书写。任何语言都包含两个要素：

1. 由一些规定可用的字母或符号构成的程序语言词库；
2. 将一些单独的程序词汇组合成指令语句的规则方法，也就是语法规则。

语言可以分为两种：自然语言和人工语言。

自然语言

自然语言已经在人与人之间的生活中发展了 10 万年左右了。语法学家和语言学家花了数千年来研究语言是怎样作用的，由于我们还不是很了解大脑是如何工作的（假设大脑的作用不可或缺），他们最初将语言的正式结构定义为基于拉丁文的一系列人为创造的规则（拉丁语是一种人造语言，但在人类社会中已经有 1500 年左右不被使用了）。语法的一个重要方面就是语法机制或者文本按照语法的分解。在 20 世纪，Onions 和之后的 Chomsky 定义了语法系统，也就是语法规则。即人们怎样运用该规则把一篇文章分解成基本的语句，再应用同样的方法将所得到的语句继续分解，直到得到基本的语句成分。这种简单的方法同样可以反向地运用，从语言最基本的组成部分进行合理的组合形成一个正确的句子。对此，Chomsky 称为生成语法。Chomsky 说过一句著名的话："语法应该能够生成一种语言中所有具有良好结构的句子"。这里的良好结构指的是语法正确。我们按语法生成句子，但语法正确不能保证在语义上有用处，但语法上不正确一定会妨碍语义上的意义——你不可能明白没有语法规则的句子的含义，因为你无法确定句子的各个组成部分表达的是哪个意向。

Chomsky 的观点是，对于这个生成方案，可以用数学的方式表示：

- 有限的词库（对于受过一般教育讲英语的人 40—120000 个词汇量）；
- 有限的语法（有一套固定的词汇组合规则）。

然而，用词语生成无数个正确句子却是可能的。这不仅是将 49 个音标按照一定方式进行组合（英语中共有 49 个音标），而且可以产生无数结构上正确的句子。

实际上，即使是只含有很少一些词汇和语法规则的语言（如第三章阐述的简单的 Lindenmayer 系统），也能够产生相当数量的语句形式。

对于一些设计或者项目，生成语法思想（在 Chomsky 的术语学中）的运用会比单纯摆弄参数提供更多的可能性。这将有助于产生尽可能多的设计思路，对于设计思想的深层次重构有重要意义。不仅如此，如果你要利用以"图灵机"为模型的计算机，就可以用文本程序语言来提供更灵活和抽象的表示。

语言的优势在于：

1．能够生成无数个语法正确的句子；

2．它能够被以递归方式解析，允许嵌入、多重分支以及具有很多的组合可能性。

人工语言

自然语言含有不确定的语法并且其词库会随着自然演变和发展发生主观变化。而人工语言则具有明确的语法和词库。

我们可以利用这些人工语言来描述算法，这是一类可以用计算机程序代码描述的算法。程序代码是一种很特殊的文本。如果经过适当的训练，人们就可以读懂程序代码；这点和自然语言中一样，如果你没学过法语，也就看不懂普鲁斯特的法语拼写。正如德国媒体理论家 Friedrich Kittler 所说，程序代码是唯一能够阅读自身的文本，其中一个著名的例子就是 Pascal 编译程序（Pascal 语言是一种结构性良好的语言，在 20 世纪 70 年代由 ETH 的 Niklaus Wirth 创造，以 17 世纪法国的数学家、哲学家 Pascal 命名）。

Pascal 编译程序使用 Pascal 语言编写，Pascal 语言是一种类似英语语法的高级语言。要使计算机能够将编写的程序变成可执行的一系列二进制代码（0 和 1），我们需要一种叫做编译器的软件，来将 Pascal 语言编写的程序翻译成机器代码。但编译器并不是原本就存在的，它其实也是一个由 Pascal 语言编写的程序，所以，在编译要运行的程序代码之前，Pascal 编译程序需要首先将自身的代码编译成为一个编译器。这就是所谓的自我开发，这件看起来不可能的工作就是由一系列软件完成的，首先有一个开启工作的软件，其次是解读软件，最后就是解读后的更深层次的软件，依此逐渐往下工作。

计算机语言是什么？

很多人有时会把任何有关技术的语言文本都看做程序代码，这样是不合适的。例如 HTML（超文本标记语言，用于互联网页描述）就不是本书所要探讨的那种语言，它其实是一段数据。它虽然有明确的语法和词库，但是它必须经过用程序设计语言编写的一套能够根据这些数据在电脑上显示字或图片的算法才能够生成网页。

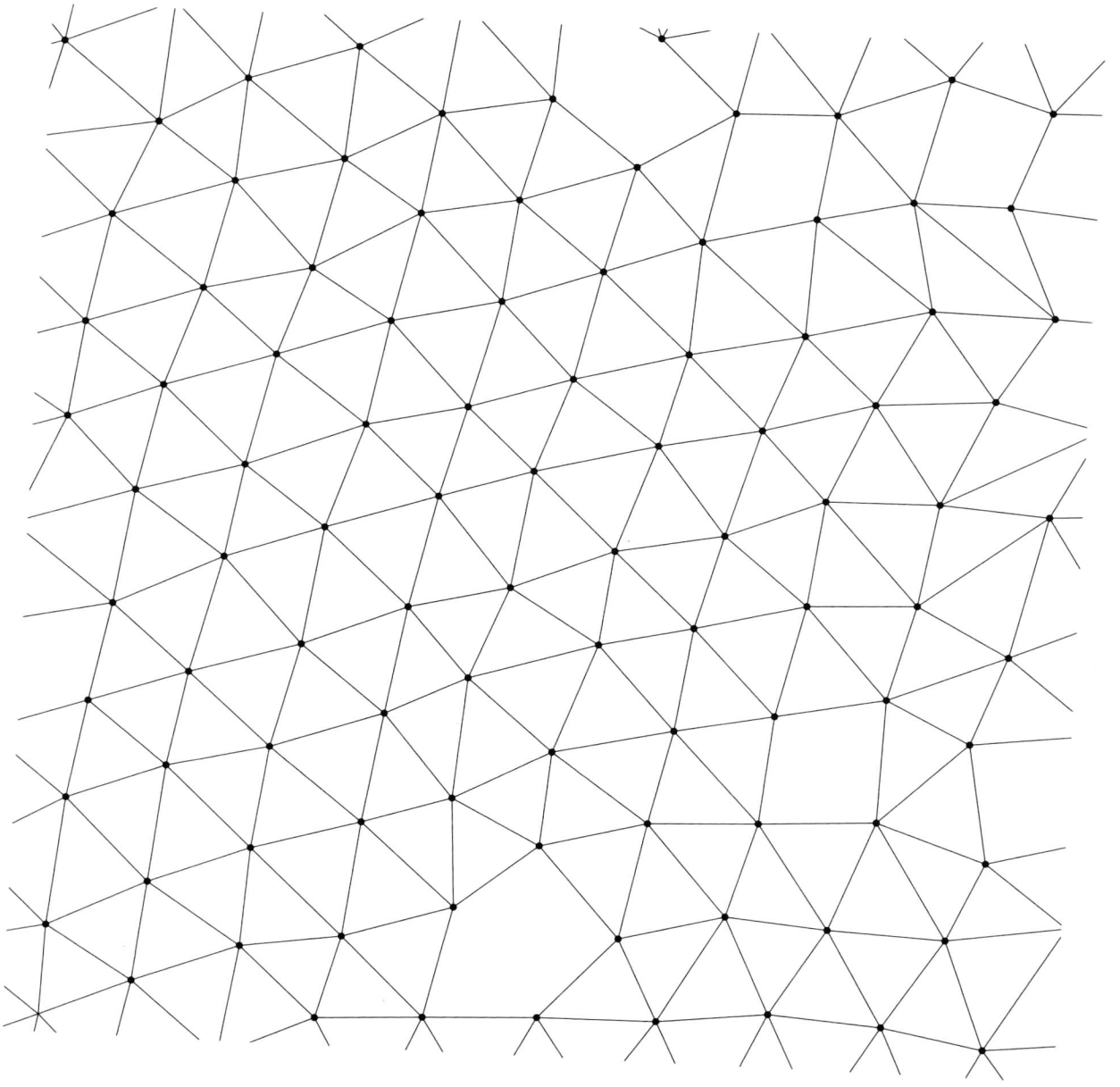
还有一个相似的例子就是纯数据。作为一个能够阅读自身的文本，需要具有特定的表示、控制结构以及算术运算。HTML 或者纯数据只是一些名词的集合，而一个程序文本需要另外包含动词、形容词和副词。

计算机语言是基于图灵机的基本操作的，计算机应具备对某种数据结构进行存储数据、读取数据或数据地址的能力。Pascal 编译器之所以能够编译自身的代码，就是因为图灵机不加区分地把程序和数据存储在一起（即所谓的“存储程序计算机”）。在图灵机中，一段数据可以作为一条指令，一条指令也可以生成一段数据。这种固有的自反性就是计算机能够利用程序代码进行自我开发的原因。

因此，我们可以通过写程序代码完成一些复杂的算法。这就是本书的要义，也是在书中对生成设计进行一系列描述的原因。通过专注于算法我们可以用通用的词汇和语法在建筑设计的两个对立面之间架起对话理解的桥梁，在设计过程中不去限定所要设计的一些模糊的部分，但是能够得出一系列的结果。

第一章

反思表现



前言部分对设计表达做了简要的描述。对Chomsky的思想做了基本的陈述：有限的语法和词库可以生成无数有效用的结构。可以将这种思想运用到程序设计语言中，通过编写程序来模拟生成的建筑，用来表示空间和形态的结构特征。这就是生成算法，应包含对研究对象的完整描述，包括对于对象的测量、分析，对象的图解以及对象的具体实现等。

我们所编写的程序作为一种人工语言，通常还要依赖于一些硬件加以实现。这里所说的硬件一般是指图灵机的一些开发产品，这些硬件可以提供多种多样的功能表现，从简单的制表到硬件编程、3D打印以及沉浸式虚拟环境。这些方面都只是一些底层算法的呈现，也是最原始的表现。同时令300人去遵循指令并将算法推演出来是有可能的（就像同步的游泳者一样），对此，下面讲的内容会有所体现。

一些简单程序文本实例

首先要探究的问题是怎样用标识语言（Logo language由Seymour Papert编写的一种值得尊敬的人工智能（AI）语言，有关Papert的经历将会在下一部分详细叙述）来描述一些简单的几何图形或实体，例如圆、球体以及其他的一些多面立体图形。

三角形和圆形

对于二维的情况，可以通过下面一个实验来验证通过程序描述图形的方法。在二维平面中，有许多点随机地散置在平面上。

对每个点规定以下规则：

- 搜索所有其他的点，找到离自己最近的那个点；
- 然后向远离最近的点的方向进行移动。

所有的点都同时按照上述规则进行移动。

这样就会出现一个问题：对于某个点，在远离离自己最近的点的过程中，该点可能会离其他的点相对更近一些。对于这个问题，我们无须担心，因为如果一个点碰到这种情况，它就会改变方向，然后远离他们。要注意，所有的点都是同时按照这个规则运动的。

我们可以用 NetLogo 语言向电脑输入上述规则来展示这些点的运动情况，这样就可以很简单的建立一个并行计算的机制。众多点可以用“海龟”来描述。“海龟”是一种自主的计算机运算单元，所有的“海龟”都遵循下面的程序：

```
to repel
ask turtles
[
  set closest-turtle min-one-of other
      turtles [distance myself]
  set heading towards closest-turtle
  back 1
]
end
```

为了理解这段代码的含义，首先来观察下面所示的代码框架：

```
to repel

  do something

end
```

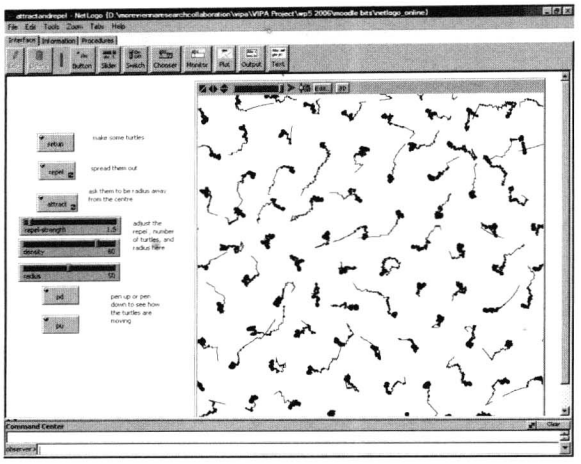
因为我们要为计算机定义怎样去做，所以要用“to repel”作为开头（因为各个点是靠潜在的排斥力达到静止）。To 和 end 之间的内容是真正的代码。然后是短语“ask turtles”，你可能会有疑问，是谁发出的请求？海龟是指空间中的点，它们相当于很多小的计算机单元，程序运行时会给所有的海龟发送信息，让它们执行中括号里边的程序代码。

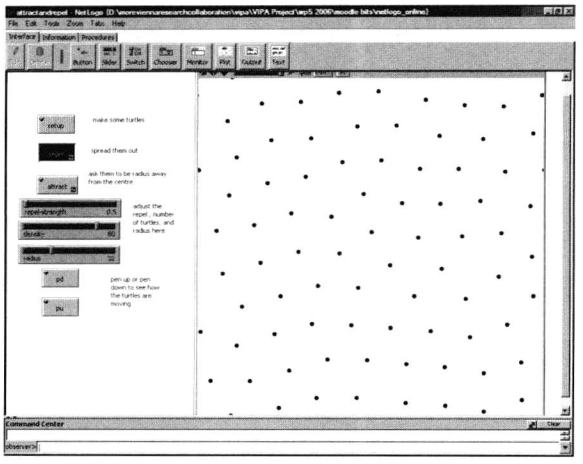
也就是下面三条语句：

```
1)  set closest-turtle min-one-of other
    turtles [distance myself]
2)  set heading towards closest-turtle
3)  back 1
```

程序运行时，海龟会被告知去寻找离它最近的那个点。它们必须要记住离自己最近的是哪个海龟，这一点可以通过将最近的海龟的标识存储到名为“closest-turtle”的变量中。然后，海龟被告知向远离那个点的方向移动一步。

有意思的是，我们还必须要计算机记住其他海龟所在的地址。如果一只海龟向所有的海龟（包括自身）发出 ask 的信息，那么就会得到最近距离为零，就意味着那只小海龟向远离自己的方向移动，显然这不是我们想要的。这是一个很好的例子来说明我们为什么不得不为这些呆呆的机器们编写最傻瓜的程序语句。





左侧上面的那张图展示的是运行上述程序后海龟从最初散落的点到相互间构成三角形网格的过程中的移动痕迹。系统一共运行了 500 步，我们可以看到海龟能够较快地找到合适的位置并待在那里（海龟的移动轨迹都不是很长而且移动轨迹相互之间无交叉）。

上述的程序还有本书中的很多其他的程序实例都是用 NetLogo 语言编写的。NetLogo 语言的产生要追溯到 20 世纪 60 年代的 LISP 语言（见第三章），在 LISP 语言的基础上出现了 Logo 语言（在下一章会有描述），之后又进一步产生了 StarLogo 语言，而 NetLogo 语言就是由 StarLogo 语言发展而来的。更多的知识可以去了解关于 Resnick（1994）的一些资料。

这些海龟静止之后构成了三角形的形态，下面是这些点连成的三角形网格。

左边是程序运行后连接成的两幅图形。每幅图的持续时间都不长，像所有的动态系统那样，任意时刻的形态都可以捕捉，但是会随着海龟的不停移动而消失。

生成网格

在一些排斥力的作用下，平面上所有的点都保持静止，形成三角形网格的图，因为当它们偏离网格交点时就会处在一个不稳定的状态，然后会重新退回到原来的位置。要注意的是，这些点的摆动并不包含在算法内（算法只包含前面所述的两条规则）。我们期待上述算法能够得出什么？看上去好像是点的漫无目的的移动；然而，实际上这些点在某种潜在的同那两条规则没有关系的力的作用下达到静止。这是一个“涌现”的例子，其思想是：通过不断的并行运算，产生一个高度有序的图形；它的生成原理是所形成的三角形网络是耗能最少的平衡点组成的二维网格，每个点与其相邻的6个点距离相等。这也是我们第一个算法示例，它具有模型的认知独立性（在这里的模型是指 repel 程序的代码），独立于算法的结构输出。也就是说，生成的稳定的三角形网格（上述算法的结构输出）并没有明确地写入算法规则中，是一个有关分布式表达的例子。

分布式表达

上述算法也是阐释分布式表达概念的例子。算法运作的方式就是写入一定的规则让每个海龟都同时去遵循。每一个海龟（也就是一些小的自主运算单元）都运行上述的程序，作出自身的判断——谁离自己最近——独立于其他海龟进行运动——转定方向、后退。上述的 repel 算法是该系统中唯一可用的描述，其他的任何东西只不过是一般的排定事件或者整个模拟活动的启停，并且活动的表达是由所有的海龟展现出来的。海龟能够彼此影响，并且具有一些可视的限制力量，例如他们能够“感应”到离自己最近的海龟并作出适当的反应。但是他们并不知道三角形网格的形成，因为这只能被全局观察者观察到，也就是在操作计算机进行模拟的人员（即你本人）。不同层次的观察者之间的差别是分布式表达的一个主要方面，这在本书后面部分会经常被提到。对于分布式模型，那些小的自主程序之间存在的反馈是至关重要的；如果每个海龟都不注意相邻海龟，那么就什么都不会产生。这在下面要介绍的细胞自动机和本章末尾的 Pondslime 算法中有明显的体现。

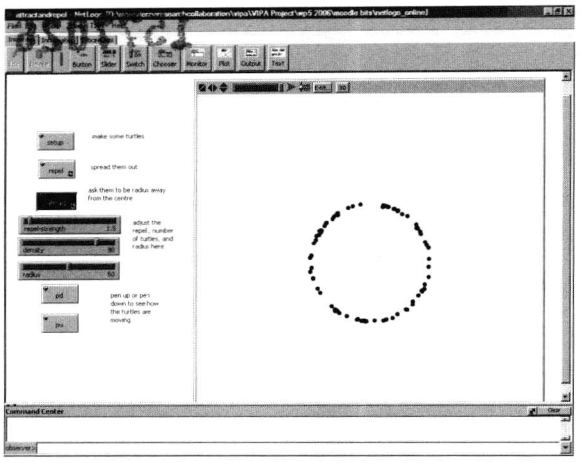
将这些自下而上的小程序与做三角网格的常规方法相比较是很有意义的。当然，仅利用简单的 Wallpaper 算法也可以有很多描述作图的方法。

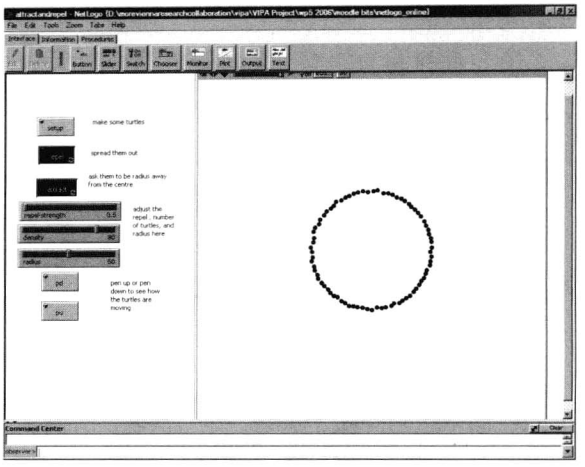
Wallpaper 算法

绘制一条点间隔为 1 的虚线，然后按照下面的原则添加第二条虚线：将上一条虚线上的所有点在 X 方向上同时偏移 0.5 的距离，在 Y 方向上同时偏移$\sqrt{0.75}$的距离。

然后可以按照上述偏移规则不断的添加虚线。

根据勾股定理可知，0.75 的平方根就是一个边长为 1 的等边三角形高的长度，它的值大约是 0.86602540378443864676372317075294。这并不是一个理想的数字，并且这个算法也不能塑造出符合潜在的动力学的描述，仅仅是机械的上下移动之后得到一个尺寸上不算理想的结果。由此，上述作图方法与算法作图的区别就很显然了，在以后的建模仿真过程中要注意这一点。在算法中对生成规则的简单描述要比传统几何作图有效得多。





IFELSE 是很多程序语言的关键字，它具有让计算机根据问题来进行选择运算的能力。我们知道，条件语句有很多种形式，在这里我们只采用“ifelse”语句。

下面的建构实例需要决定在下面的程序段中运行哪条路线的语句。

实例：假设你站在一个道路的分岔口，你需要决定向左走还是向右走。你会怎么做呢？这时如果你发现自己口袋里有一张姑母给你的便条，上面写着：

｛“当你站在道路分岔口的时候，如果你吃过午饭了就往左走，否则就往右走”｝

然后你发现当时你已经吃过午饭了，所以就往左走。问题于是解决了（向左走显然是茶馆的方向）。

在 attract 脚本中，那张你姑妈给你便条，上面写着“如果你离圆心的距离小于半径，那么就后退一步；否则，就前进一步”。

IFELSE 语句的思想就是：首先提问一个问题，然后根据是或否来选择要执行的运算。

IF <something is true> ***THEN*** DOTHIS
ELSE DOTHAT

这就是称之为 IFELSE 语句的原因。

标准形式

ifelse (conditional expression)
[thing to do if true]
[thing to do if false]

扩展模型——海龟画圆

下面要讲的就是典型的 Papert 式的利用算法生成几何图形的例子。在这个例子中的几何图形是基于圆形，其实算法还可以生成很多的复杂图形，比如维诺多边形（生成网格）。这些都是系统协调的例证，因为你所看到的位元（印在本页对面的两幅图）是系统的所有组成部分（几乎所有海龟）达成某种协议后生成的结果。而至于海龟被要求遵守什么协议，这其中涉及什么建筑思想值得我们思考。总的来说，这个任务为每个海龟分配了两个存在冲突的动作命令：每个海龟的自身运动和整个群体图案的高度有序。

Papert 提出下面的等式：

```
Xcirc = originX + Radius cos (angle)
Ycirc = originY + Radius sin (angle)
```

其中，(originX) 表示海龟当前所在的点，(Xcirc, Ycirc) 点表示海龟下一步要到达的圆上的点，angle 表示需要偏转的角度。

上述等式并不能得到关于圆的任何有用的信息。然而，我们可以用 NetLogo 语言编写一段程序，通过告诉海龟不断的重复向前移动、小角度向左偏转方向来形成一个圆（可参见第二章中对 S. Papert 的介绍）。

程序代码如下：

```
To circle
  Repeat 36
  Forward 1
  Turn Left 10
End repeat
End circle
```

编写这段程序只需要熟悉英文并且构造一个简单的运动规律。

就像 Resnick 在《Turtles, Termites and Traffic Jams》(1994) 一书中曾指出，我们可以利用并行计算的思想提出另一种方案，那就是用很多个海龟画圆，而不是一个。这个算法基于圆的基本特征：圆上所有的点离圆心的距离都相等。

用海龟完成这个方案，我们应该：

创建很多散置的海龟

- 让每个海龟都朝向圆心的方向；
- 让每个一海龟都测试自己与圆心之间的距离；
- 如果海龟与圆心的距离小于半径，就后退一步（因为太近）；
- 如果海龟与圆心的距离大于半径，就前进一步（因为太远）；
- 不停地按上述规律进行移动。

上述过程可以用 NetLogo 语言进行如下描述：

```
to attract
ask turtles
[
  set heading towardsxy 0 0
  ifelse ((distancexy 0 0 ) < radius)
      [bk 1]
      [fd 1]
]
end
```

请注意，在程序中，并没有告诉海龟应该移动到哪个点，他们只是按条件选择前进或后退。实际上，形成的圆只有运行程序的人能观察到（海龟们并不知道它们形成了一个圆）；而且，我们能看到的也只是它们形成圆形后的画面，其形成圆形的过程是观察不到的。上述程序运行的结果就是一圈的海龟构成一个圆形。实际上，海龟达到静止绕成一个圆圈后，还有一项工作要做。由于海龟的出发点是随机分布的并且最终都随机的分布在圆圈上，所以海龟静止后生成的是一个有缺口、并不是连续完整的圆。为使海龟能够沿着圆圈均匀分布开来，我们需要运行之前提到的 Repel 算法。所不同的是，这里海龟移动的并不是一步，而是由界面中“slider”控制的一个不确定值。

此处所用的 Repel 算法的程序如下：

```
to repel
ask turtles
[
  set closest-turtle min-one-of other
      turtles [distance myself]
  set heading towards closest-turtle
  bk repel-strength
]
end
```