# 2016 数据结构与算法课程

# “递归视觉艺术”实习报告集

(北京大学地空学院)

SESSDSA'16 班 著

编辑：陈春含

指导教师：陈斌

2016 年 5 月

# 目录

# 1 A 组

# 数据结构与算法课程

## 递归视觉艺术实习作业报告

## （作品名称：西门吹雪东篱散人）

（魏麟懿*，邵锐成，杨江南，朱金顺，胡胜懿，孙新然）

摘要：创意以一张照片为开始，以想象为结合，利用诗意的意象组合，以静衬动，表现出最具有创意的画面。

关键字：树，花，草，雪，亭子，篱笆，月夜

## 1.1 创意过程与递归思路

### 1.1.1 作品总体介绍

那么这幅画是什么样的呢？“此乃花间一壶酒，独酌无相亲。举杯邀明月，对影成三人”，有“南窗背灯坐，风霰暗纷纷。寂寞深村夜，残雁雪中闻”。是寂静，是悲凉，又带有生命的气息。

利用动态图和最传统的递归来展现。

### 1.1.2 创意来源

其实我们的创意的形成过程十分复杂，先是打算画北大西门的春夏秋冬变化，但由于其中涉及的递归较少，而且超出了我们的正常能力范围，所以我们打算花桃花林的春夏秋冬的变化，这时恰巧看到了一副《桃花源记》的水墨画插图，觉得十分中意，于是乎就打算以此为模板，呈现出我们想象的画面。但是不幸的是，我们在尝试了一下午后，发现树枝的水墨画的自然写意的形态实在难以展现，所以我们转向了正常的桃花林，以一张照片为主。但不幸的是，当我们把那个照片的元素呈现出来并组合起来的时候，我们一致决定利用进行改善。于是乎现在的画面就是我们的想象加设想的结果。

### 1.1.3 递归思路

1，递归算法必须有一个基本结束条件（最小规模问题的直接解决）
2，递归算法必须能改变状态向基本结束条件演进（减小问题规模）
3，递归算法必须调用自身（解决减小了规模的相同问题）

## 1.2 程序代码说明

### 1.2.1 函数说明

1、函数的相互连接：

我们的总程序采用线性结构，即按照作图顺序使各画图函数分步进行。最开始时定义一个 Turtle 类型的变量 pen,作为主要的画笔，并构建 move 函数，其作用是使 pen 在 up 的状态下运行到指定位置。以下是作画顺序：

1. pen 移动到目标位置，运行 drawGreenTree 与 drawGreenTree1 函数,画出两棵青树。
2. pen 移动到目标位置，三次运行 drawPeachTree 函数，画出三棵桃树。
3. pen 移动到目标位置，三次运行 drawPeachTree 函数，画出三棵桃树。
4. 运行 drawFence 函数与 drawGrass 函数，分别画出篱笆和草地，它们都内置有自动移动 pen 至目标位置的功能。
5. pen 移动到目标位置，运行 drawChairandTable 函数与 putPavilion 函数，分别画出石桌石凳和凉亭。
6. 。。。

2、实现思路

函数模块 drawpitch，drawgreentree 的实现思路

我们 drawpitch，drawgreentree 的函数目的是为了画树，先是画树干，后是画叶子和花之类的，叶子利用两个 60 度的圆弧而制成的涂上绿色，颜色的程度随枝干的分叉而不同而呈现出立体感与层次感，而花是以两个 1/4 圆和 1/2 圆组成的，随着树干分叉到一定程度而开始出现。树干的实现是利用传统的递归形成的，不过为了呈现出自然的状态，而改成了利用随机导致其不对称，但这种不对称是一定的，有一部分固定了角度。为了致使 turtle 能在一次递归后能回到原位置，所以采用了 pos 函数去记录当时的位置。

函数模块 create a pavilion 的实现思路

首先将亭子拆解为几部分分块实现，这几部分分别是屋顶、柱子，柱台、天花板，底部栏杆，轮廓空白填充四部分。屋顶部分先借助 turtle 实现一条适当角度弧线（可不断调试获得满意效果），然后沿着设定的轨迹通过位置的移动实现迭代形成屋顶，这里有一个细节，就是屋顶的瓦片效果，这里采用颜色随在弧的不同位置呈周期变化实现这个效果。柱子的实现为了不留下锯齿，采用线条重复勾勒，柱子和柱台相对位置不变，因此将柱子和柱台集成到一个函数方便调用。柱台通过 circle 的位置变化实现。底部栏杆很容易，确定坐标就可实现。然后就是轮廓空白填充，同样确定轮廓，填充即可。

函数模块 grass 的实现思路

关于 grass 的函数，为了表现出更加真实的草的感觉，我们选择让每颗草都有一定程度的弯曲。而这种弯曲并不能简单地用圆弧来刻画，否则会显得太不真实，所以选用化直为曲的思想来实现。每前进一点就旋转一点的角度，多次重复后就可以展现出曲线的效果。但为了模拟真实的弧度，在微元旋转的角度中需要略做调整，使之非线性变化，这样由底部到尖部就画出了一笔优美的弧线，再转头，经过相似的过程，将小草的第二笔画上去，这个过程经过许多次调整，使得这两笔构成的草，尽量自然。再将其填充草绿色，一棵草的函数就结束了。再经过类似的过程画一个相反朝向的草，使得一个向左，一个向右。最后再定义第三个函数，重复调用上面两个函数，但为了使得看起来不是千篇一律，有些参差感，我们选择使用 random 模块调整草的高度和偏离的角度以及草的左右偏向在一定的范围内随机变动，这样就实现了一片草地函数的构建。

## 1.3 实验结果

### 1.3.1 作品描述

“此乃花间一壶酒，独酌无相亲。举杯邀明月，对影成三人”，有“南窗背灯坐，风霰暗纷纷。寂寞深村夜，残雁雪中闻”。是寂静，是悲凉，又带有生命的气息。

## 1.4 实习过程总结

### 1.4.1 分工与合作

魏麟懿负责画树，朱金顺负责画亭子，杨江南负责画草，胡圣懿负责画篱笆和我们的讲解担当，孙新然负责写报告，邵锐成负责我们总的技术指导，几乎各个函数的实现都有他的参与，厉害至极。

### 1.4.2 经验与教训

我们获得的经验就是得在自己力所能及和独家创意之间的平衡，而且在人员时间安排上的确花了很大一部分精力，而且人一多面对困难时很容易产生困顿的情绪，以及不想再干下去，因为有责任分担，但幸运的是，我们坚持了过来，中间还要受其他组大神的图像的冲击，守住自己画的东西显得弥足珍贵。

### 1.4.3 建议与设想

本次实习作业给了我们一个用自己课堂上所学的知识来解决一个具体问题的机会。尽管

我们今后的学习可能会与编程的联系不是那么的紧密，但是在这个大数据、信息化的时代，我们应该在这门课上学会的不仅仅是编程的技能，更是一种用编程来解决问题的思想。所以整体而言，我们对这次作业还是很有兴趣的，在完成它的时候也充满了斗志。
但这次作业也有可以提升的地方。比如在创意方面，我觉得我们可以不仅仅局限于使用 turtle 模块来画出具有艺术美感的图案。在我们的生活中还有许多地方会用到 python 的作图功能，比如数学中的解析几何问题，它繁杂的计算让曾经学习过它的我们感到头疼。我们可以用 turtle 来帮助我们进行计算，这样既帮助我们巩固了编程的知识，在以后的学习中也算是掌握了一个比较实用的技巧。还有一点就是在组队方面，感觉单支队伍的人数可以再多一些。我们在讨论的过程中虽然产生了很多想法，但有很多由于人数和时间的限制，很难去实现，也许扩大队伍的规模可以解决这个问题。

至于实习作业后的设想，我觉得可以在原有小组的基础上在给我们一些任务。这些任务可以没有很明确的 Deadline，但是这个问题有足够的吸引力让我们每天去进行一些思考。我们觉得，编程绝对不只是作为一种技能而存在，而是作为学习、生活中的一种习惯，融入到我们的思维当中。

## 1.5 致谢

感谢技术总监邵锐成对所有函数的修改，花费了巨大的心思，不愧是邵院士。
感谢胡圣懿对我们作品的介绍和作为我们组的颜值担当。
感谢朱金顺画亭子时不辞辛苦的一笔一笔的慢慢画。

## 1.6 参考文献

https://docs.python.org/3.5/library/turtle.html#turtle.ScrolledCanvas　turtle 所有的方法

# 2 C 组

# 数据结构与算法课程

## 递归视觉艺术实习作业报告

## (四季)

(黄鑫* 张思源 夏运 庞骁 熊轶伟 欧宇航)

摘要：本创意的主要来源是沙画，沙画是一种动态作画的方法，而海龟作图的过程亦是动态的，于是我们很自然地期待使用 python 海龟作图呈现出图案变化的动态过程。在画雪花以及树时使用了递归，使用迭代器作太阳的日冕线以避免系统栈溢出，主要原理是函数的递归思想。我们使用了循环，递归，迭代器和 turtle 模块的多种方法，力图展示出四季的的变化大致图景。

关键字：树，雪花，四季变换，递归，循环，迭代器，shape 类型

## 2.1 创意过程和递归思路

### 2.1.1 作品总体介绍

以四季为主题，以树为载体，描绘四季的变换，春天万物苏生，夏天枝叶茂盛，秋天变黄凋零，冬天雪花飞舞。在树以及背景的风中采用递归的程序，主要使用了循环和递归的想法，背景使用渐变背景，动态元素一律使用海龟的自定义 shape。总的来说，是使用自定义的海龟形状实现 python 的 turtle 所不具有的图层功能。

### 2.1.2 创意来源

受到上一届的学长学姐们的作品的启发，我们根据沙画的想法，进行模拟，最初决定利用白色笔进行涂改，将前一季的图案抹去，接着画上新的，以此在一张图上实现动态变化。张思源同志十分积极，强力要求承担主体树的工作，组长紧随其后，决定作为背景担当，剩下的后期工作交给庞骁，夏运同学由于害怕写报告，协助组长做背景。剩下两个人就写报告了。

### 2.1.3 递归思路

树的递归：主要是树干的递归，在画完主干后分叉，再次执行做树干的函数，通过引入

随机量，使得每一次作图的树干都与之前不同。

雪花的递归：本程序中的雪花由希尔伯特曲线构成，在绘制希尔伯特曲线的过程中使用了递归的方法。

迭代器：本程序中的日冕实际上是内摆线，作内摆线的方法是利用函数方程分段描出函数的图像。为了得到任意内摆线各小段两端点的坐标信息，本来我们准备使用递归的方法。但由于 python 对于递归深度有限制，以及系统栈长度有限，我们改用迭代器生成点对的坐标。

## 2.2 程序代码说明

### 2.2.1 数据结构说明

我们在 poly 方法内绘制所需的，较为复杂的图形，加入自己命名的 shape 并导入海龟 shape 的列表中，以将海龟的形状设置为所需。同时，由于图形的位置、方向都可以通过改变海龟的位置和指向加以改变，因而可以实现较为细腻逼真动态效果。在主程序的海龟运行到所需位置时，使用 t.clone()方法在该处复制一只海龟，并且将该海龟加入到一个列表中，以便等待后续操作。

### 2.2.2 函数说明

引用背景:

gc 模块用于减少程序的内存占用

```
#coding:utf-8
import gc
import math
import random
import turtle
from PIL import Image
```

渐变背景绘制函数：

```
def background(x, y, size, t):  # size 控制长度
    def Goto(x, y, t):
        t.up()
        t.goto(x, y)
        t.down()
    t.pencolor(79, 151, 199)
    t.pensize(1)
    t.setheading(0)
    Goto(x, y, t)
    i = 1
    step = (190 - 22) * 1.0
    while True:
```

```
        r = 79 + i * (229.0 - 79.0) / step / 4
        g = 151 + i * (222.0 - 151.0) / step / 4
        b = 199 + i * (238.0 - 199.0) / step / 4
        if r > 255 or g > 255 or b > 255:
            break
        t.pencolor(int(r), int(g), int(b))
        t.forward(size)
        Goto(x, y - i, t)
        t.setheading(0)
        i = i + 1
    t.pencolor(56, 42, 31)
    t.setheading(0)
    Goto(x, -y, t)
    i = 1
    while (t.position()[1] <= -250):
        r = 56 + i * (74.0 - 56.0) / step
        g = 42 + i * (55.0 - 42.0) / step
        b = 31 + i * (41.0 - 31.0) / step
        t.pencolor(int(r), int(g), int(b))
        t.forward(size)
        Goto(x, -y + i, t)
        t.setheading(0)
        i = i + 1
```

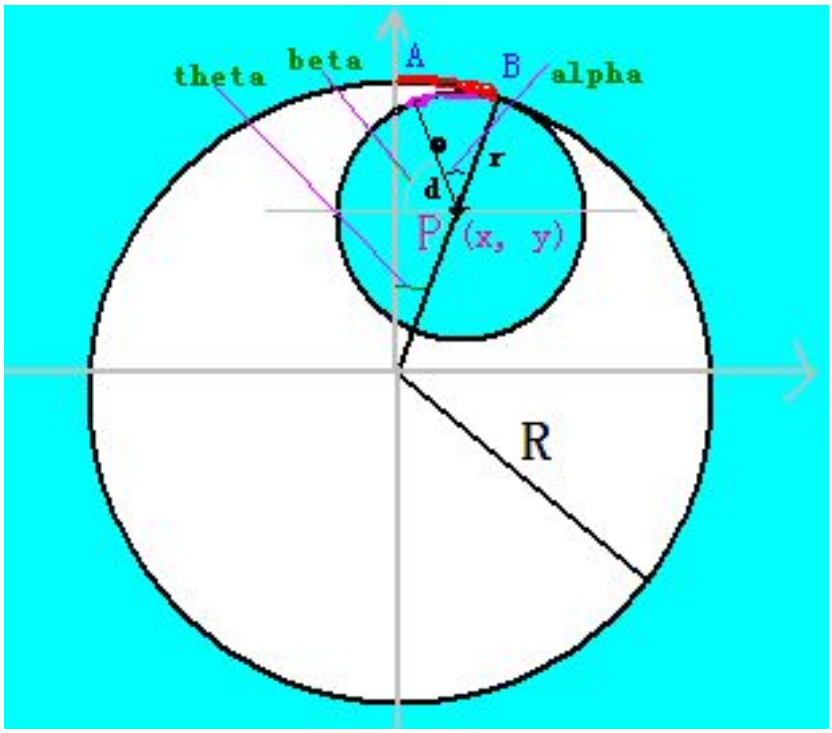
用于绘制太阳的函数：

```
def drawSun(x, y, t, win, r):
    def getADot(A, R, r, d, x0, y0):
        # B、C 分别代表图中的 theta 和 beta
        B = A * r / R
        C = math.pi / 2 - A + B
        x = (R - r) * math.sin(B) - d * math.cos(C) + x0
        y = (R - r) * math.cos(B) + d * math.sin(C) + y0
        return (x, y)
    # def drawPattern(t ,lower ,upper ,R ,r ,d ,step):
    #     if lower == upper :
    #         return 0
    #     else:
    #         i = lower
    #         t.up()
    #         dot1 = getADot(i ,R ,r ,d)
    #         t.goto(dot1)
    #         t.down()
    #         dot2 = getADot(i+ step ,R ,r ,d)
    #         t.goto(dot2)
```

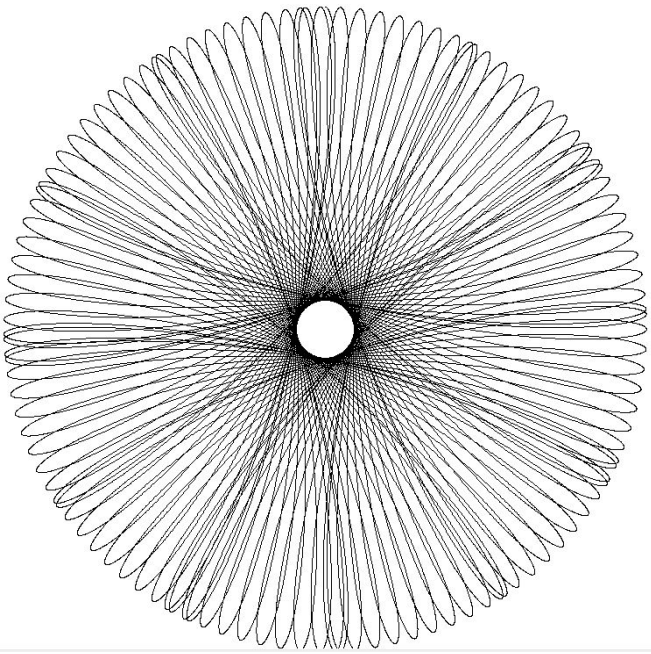
```
#         lower = lower + step
#         drawPattern(t ,lower ,upper ,R ,r ,d ,step)
# # 改变为递归程序，那么在运行到中途一个特定点时会出错，原因是系统栈溢出。
# lower、upper 确定了 alpha 的取值范围
# step 决定了作图的精度

def DotPairs(lower, upper, R, r, d, step, x0, y0):
    if lower == upper:
        yield 0
    else:
        while lower < upper:
            i = lower
            dot1 = getADot(i, R, r, d, x0, y0)
            dot2 = getADot(i + step, R, r, d, x0, y0)
            yield (dot1, dot2)
            lower = lower + step
def drawPattern(t, lower, upper, R, r, d, step):
    x0, y0 = t.pos()
    for data in DotPairs(lower, upper, R, r, d, step, x0, y0):
        if data == 0:
            return 0
        else:
            (dot1, dot2) = data
            t.up()
            t.goto(dot1)
            t.down()
            t.goto(dot2)
def patternPainter(R, r, d, t, win):
    t.hideturtle()
    t.radians()
    win.tracer(10)  # 在修改 step 的同时，建议也修改作画速度
    drawPattern(t, 0, 102, R, r, d, 0.1)

t.up()
t.goto(x, y)
t.down()
t.color("yellow")
t.begin_fill()
patternPainter(9 * r / 5, 9 * r / 5 * 3 / 16, r / 5, t, win)
t.end_fill()
t.up()
t.goto(x, y - r)
t.down()
t.color("red")
```

```
t.begin_fill()
t.circle(r)
t.end_fill()
```

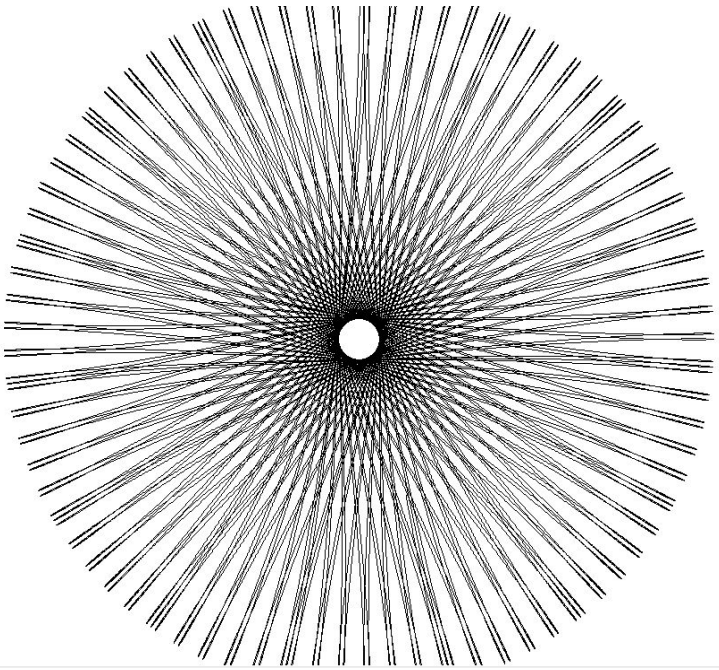
其中，函数内前三个定义的函数实际上是一个更为独立的函数，其作用是在给出任意 R，r，d 的条件下作出其内摆线。



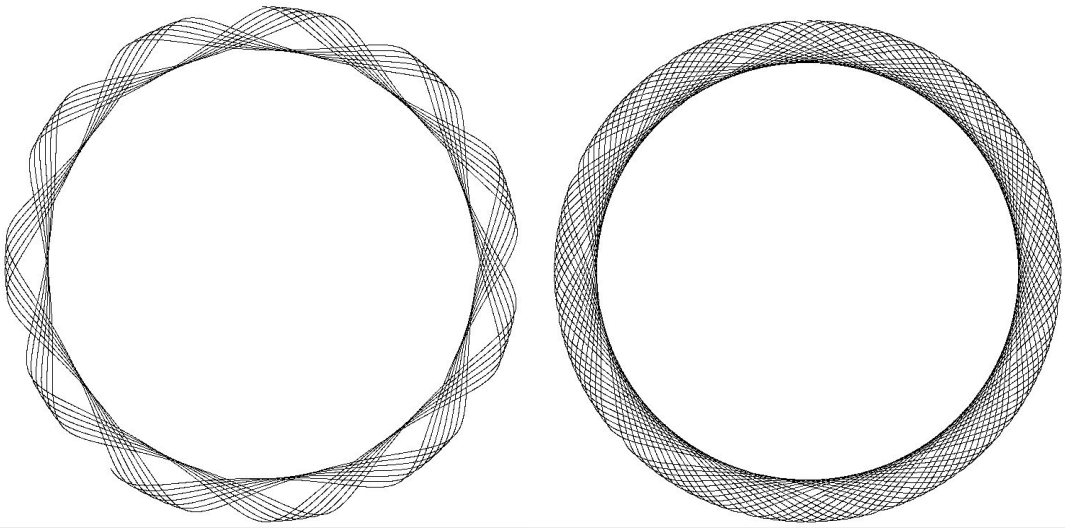

内摆线函数分析



patternPainter(400,233,200)

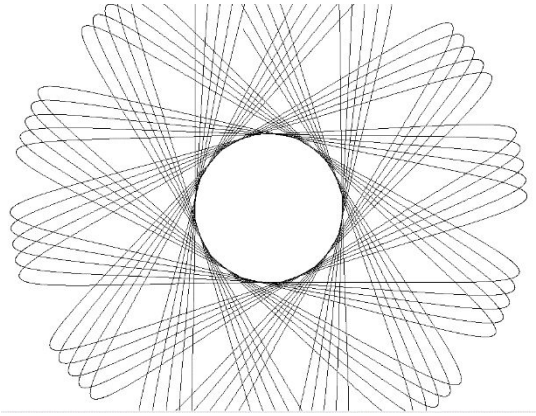


patternPainter(400,233,200)

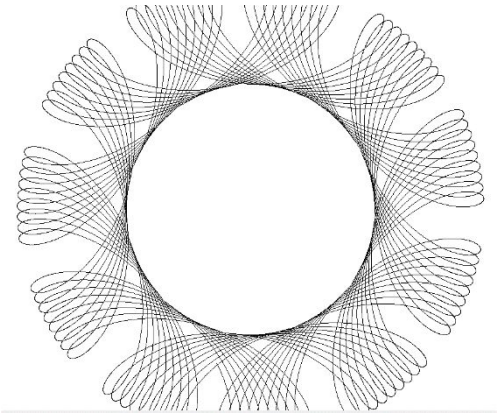


patternPainter(400,73,30)

图形的整体规模由 R 决定，图形中线的重复次数由 R/r，决定，R 相同当 d 大于或小于 r 时，会得到差异很大的图形。



patternPainter(400,233,300)



patternPainter(400,73,100)

绘制云朵的函数：

通过改变 size 改变云朵的大小，形成的是相似的图形。

实际上首先是将云朵的形状加入到海龟的 shape 列表中。

```
def setcloud(size, win):
    n = size
    t = turtle.Turtle()
    t.hideturtle()
    t.speed("fastest")
    t.setheading(-90)
    t.up()
    t.begin_poly()
    t.fillcolor("white")
    t.begin_fill()
    t.fillcolor(255, 255, 255)
    t.begin_fill()
    t.circle(12 * n, -190)
    t.left(90)
    t.circle(20 * n, -150)
    t.left(80)
    t.circle(26 * n, -70)
    t.left(50)
    t.circle(10 * n, -180)
    t.left(60)
    t.circle(26 * n, -90)
    t.left(90)
    t.circle(16 * n, -140)
    t.end_fill()
    t.end_poly()
    cloud = turtle.Shape("compound")
    cloud.addcomponent(t.get_poly(), 'white')
    win.register_shape('cloud', cloud)
```

返回树叶多边形的函数：

返回树叶的所需形状，并加入到海龟的 shape 列表，以便将海龟的形状设置为此。为了方便的改变树叶的颜色，设置了两种颜色不同，形状相同的树叶。

```
def setshapeLeaf(s):
    def leafPoly(size):
        t = turtle.Turtle()
        t.hideturtle()
        t.speed("fastest")
        t.up()
        t.begin_poly()
        t.fd(size)
        temploc = t.pos()
```

```
        t.fillcolor("white")
        t.begin_fill()
        t.circle(size // 2, -180)
        t.circle(3.62 * (size // 2), -63.44)
        t.up()
        t.goto(temploc)
        t.seth(-180)
        t.circle(size // 2, 180)
        t.circle(3.62 * (size // 2), 63.44)
        t.end_fill()
        t.end_poly()
        return t.get_poly()

    aLeafLightGreen = turtle.Shape("compound")
    aLeafGreen = turtle.Shape("compound")
    aLeafYellow = turtle.Shape("compound")
    Poly = leafPoly(10)
    aLeafLightGreen.addcomponent(Poly, (0, 255, 0))
    aLeafGreen.addcomponent(Poly, 'green')
    aLeafYellow.addcomponent(Poly, (255, 122, 0))
    s.register_shape("leafLG", aLeafLightGreen)
    s.register_shape("leafG", aLeafGreen)
    s.register_shape("leafY", aLeafYellow)
```

返回花朵多边形的函数：

返回树叶的所需形状，并加入到海龟的 shape 列表，以便将海龟的形状设置为此。

```
def setshapeFlower(s):
    def huaban(deg, size, t):
        t.fillcolor("white")
        t.begin_fill()
        t.fd(size)
        rad = deg / 180 * math.pi
        r = size * math.tan(0.5 * rad)
        t.circle(r, 180 + deg)
        t.fd(size)
        t.end_fill()

    def flowerPoly(size, parts):
        t = turtle.Turtle()
        t.speed("fastest")
        t.hideturtle()
        t.up()
        t.begin_poly()
        deg = 360 / parts
```

```
        for part in range(parts):
            t.seth((part + 0.5) * deg)
            huaban(deg, size, t)
        t.end_poly()
        return t.get_poly()

    aFlower = turtle.Shape("compound")
    Poly = flowerPoly(5, 5)
    aFlower.addcomponent(Poly, (255, 192, 203))
    s.register_shape("flower", aFlower)
```

用于绘制树的主体程序：
将设置叶、花形状的函数置于了该函数外。
采用函数递归的方法绘制树干，在每个树枝最后的位置克隆一个海龟并且设置为树叶的形状，并将该海龟存入一个列表中，以便对其进行后续处理。在某些树枝的位置克隆海龟并且设置为花朵的形状，并将该海龟存入一个列表中，以便对其进行后续处理。

```
def tree(branchLen, t, lst):
    if branchLen > 5:
        t.color(67, 37, 10)
        t.pensize(branchLen // 6)
        realLen = random.randrange(branchLen - 8, branchLen + 8)
        t.fd(realLen)
        t.right(25)
        tree(branchLen - 15, t, lst)
        t.left(55)
        tree(branchLen - 15, t, lst)
        t.right(30)
        t.up()
        t.backward(realLen)
        t.down()
    else:
        angle = t.heading()
        location = t.position()
        long = len(lst)
        if long == 0 or location != lst[long - 1]['loc']:
            lst.append({'ang': angle, 'loc': location})
```

用于绘制草的函数：

```
def grass(x, y, color, myTurtle, n = 1):
    myTurtle.up()
    myTurtle.home()
    myTurtle.speed(2)
    myTurtle.goto(x, y)  # 初始位置
```

```
    myTurtle.fillcolor(color)
    myTurtle.begin_fill()
    myTurtle.left(90)
    myTurtle.circle(20 * n, 170)
    myTurtle.left(10)
    myTurtle.circle(40 * n, -120)
    myTurtle.right(30)
    myTurtle.forward(30 * n)
    myTurtle.circle(50 * n, 60)
    myTurtle.right(340)
    myTurtle.circle(60 * n, -90)
    myTurtle.backward(40 * n)
    myTurtle.right(25)
    myTurtle.backward(10 * n)
    myTurtle.forward(80 * n)
    myTurtle.right(180)
    myTurtle.circle(40 * n, -90)
    myTurtle.circle(20 * n, 90)
    myTurtle.forward(80 * n)
    myTurtle.right(30)
    myTurtle.circle(40 * n, -110)
    myTurtle.circle(20 * n, 180)
    myTurtle.right(125)
    myTurtle.forward(107 * n)
    myTurtle.end_fill()
```

用于绘制雨滴的函数：
同样的，是返回雨滴多边形的函数，并且加入海龟的 shape 列表。

```
def setWaterDrops(size, win):
    def drawDrop(t, size):
        t.right(60)
        t.circle(2 * size, 60)
        t.circle(size, 180)
        t.circle(2 * size, 60)

    def dropPoly(size):
        t = turtle.Turtle()
        t.hideturtle()
        t.speed("fastest")
        t.up()
        t.begin_poly()
        drawDrop(t, size)
        t.end_poly()
```

```
        return t.get_poly()

    t = turtle.Turtle()
    t.hideturtle()
    t.up()
    aDrop = turtle.Shape("compound")
    poly = dropPoly(size)
    aDrop.addcomponent(poly, (142, 177, 186))
    win.register_shape("drop", aDrop)
```

用于使树叶和花朵落下的函数：

```
def leafs_down(t, y):
    t.up()
    t.setheading(270)
    t.speed(3)
    lenth0 = 0
    while t.position()[1] >= y:
        for i in range(2):
            t.left(2)
            t.forward(15)
        t.color('yellow')
        for i in range(2):
            t.right(1)
            t.forward(15)
        lenth0 += 30
```

用于绘制风的函数：

```
def drawWind(x, y, towards, n, size, t):
    t.up()
    t.goto(x, y)
    t.color("black")
    t.down()
    t.pensize(5)
    t.setheading(towards)
    t.forward(300 * n)
    r = 80
    for i in range(3):
        t.circle(r * n, 140)
        t.pensize(size)
        r -= 20
        size -= 1
```

用于绘制雪花的函数：
该雪花图形是有三段柯奇曲线组成的。在绘制柯奇曲线的过程中使用了递归的方法以使程序思路简明。

```
def setFlakes(size,win):
    def koch(n, len, t):
        if (n == 0):
            t.forward(len)
        elif (n == 1):
            t.forward(len / 3)
            t.left(60)
            t.forward(len / 3)
            t.right(120)
            t.forward(len / 3)
            t.left(60)
            t.forward(len / 3)
        else:
            koch(n - 1, len / 3, t)
            t.left(60)
            koch(n - 1, len / 3, t)
            t.right(120)
            koch(n - 1, len / 3, t)
            t.left(60)
            koch(n - 1, len / 3, t)
    def flakePoly(size):
        t = turtle.Turtle()
        t.hideturtle()
        t.speed(100)
        t.up()
        t.left(30)
        t.begin_poly()
        for i in range(3):
            koch(3, size, t)  # 迭代次数，直线长度
            t.right(120)
        t.end_poly()
        t.down()
        return t.get_poly()
    aFlake = turtle.Shape("compound")
    poly = flakePoly(size)
    aFlake.addcomponent(poly, (142,177,186))
    win.register_shape("Snow", aFlake)
```

在主函数最后调用以绘制人物的函数：
使用 Pil 模块方法以实现。

```
def drawpic(x, y, n, scale, t, w, picture):
    pic = Image.open(picture)
    w.colormode(255)
    t.hideturtle()
    lenth = pic.size[0] / scale
    width = pic.size[1] / scale
    t.pensize(n)
    w.tracer(2000)
    for i in range(int(width / n)):
        t.up()
        t.goto(x, y - n * i)
        t.down()
        for j in range(int(lenth / n)):
            color = pic.getpixel((n * j * scale, n * i * scale))
            if len(color) == 3:
                t.pencolor(pic.getpixel((n * j * scale, n * i * scale)))
                t.fd(n)
            else:
                if pic.getpixel((n * j * scale, n * i * scale))[3] == 0:
                    t.up()
                    t.fd(n)
                    t.down()
                else:
                    t.pencolor(color[0:3])
                    t.fd(n)
            del j
        gc.collect()
        del i
    t.up()
    t.goto(lenth, width)
    t.down()
    if len(pic.getpixel((pic.size[0] - 1, pic.size[1] - 1))) == 3:
        t.pencolor(pic.getpixel((pic.size[0] - 1, pic.size[1] - 1)))
        t.backward(n)
    else:
        t.up()
        t.backward(n)
        t.down()
    if len(pic.getpixel((pic.size[0] - 2, pic.size[1] - 2))) == 3:
        t.pencolor(pic.getpixel((pic.size[0] - 1, pic.size[1] - 1)))
        t.backward(n)
    else:
        t.up()
        t.backward(n)
```

```
        t.down()
    w.update()
```

主函数：

大致流程为

绘制背景，绘制太阳，绘制云朵并飘入界面，绘制草、树、树叶，在云朵的相应绘制雨滴，云朵的颜色变为深绿，绘制花朵，除掉雨滴，绘制风并让云朵飘出屏幕，花朵飘落，树叶变黄，树叶飘落，擦去风，绘制雪花，绘制人物。

```
def main():
    detailsleaves = []
    leafs = []
    flowers = []
    mt = turtle.Turtle()
    t = turtle.Turtle()
    wind=turtle.Turtle()
    gss=turtle.Turtle()
    gss.hideturtle()
    wind.hideturtle()
    t.hideturtle()
    win = turtle.Screen()
    win.colormode(255)
    mt.hideturtle()
    mt.speed('fastest')
    setshapeLeaf(win)
    setshapeFlower(win)
    setcloud(3.5, win)
    setWaterDrops(10, win)
    setFlakes(30,win)
    win.tracer(8)
    background(-900, 500, 2000, mt)
    win.tracer(1)
    # 太阳
    win.tracer(3)
    drawSun(-670, 270, t, win, 50)
    # 云朵
    win.tracer(1)
    mt.up()
    mt.goto(1200, 200)
    mt.shape('cloud')
    cloud = mt.clone()
    cloud.showturtle()
    cloud.speed(4)
    cloud.setheading(180)
    cloud.fd(1550)
```

```
    # 画树
    win.tracer(8)
    mt.up()
    mt.goto(350, -300)
    mt.down()
    mt.setheading(90)
    mt.color(67, 37, 10)
    mt.pensize(124 // 6)
    mt.fd(100)
    tree(124, mt, detailsleaves)

    # 画草
    win.tracer(1)
    grass(-200, -251, 'green', gss, 0.6)
    # 画叶子
    mt.shape('leafLG')
    for detail in detailsleaves:
        mt.up()
        mt.goto(detail['loc'])
        mt.down()
        mt.seth(detail['ang'] + 120)
        tempT = mt.clone()
        leafs.append(tempT)
    win.tracer(1)
    for i in leafs:
        i.showturtle()
        win.delay(50)

    # 雨滴
    mt.setheading(0)
    mt.up()
    mt.shape('drop')
    drops = {i: mt.clone() for i in range(20)}
    win.tracer(1)
    for i in range(4):
        for j in range(5):
            drops[i * 5 + j].goto(-400 + j * 60 + random.randrange(40), -100 + i * 63 +
random.randrange(20))
    for i in range(20):
        drops[19 - i].showturtle()
        win.delay(50)

    # 叶子变绿
    win.tracer(1)
```

```
for i in leafs:
    i.shape('leafG')
    win.delay(50)

# 画花
mt.shape('flower')
mt.color(255, 192, 203)
for detail in detailsleaves:
    have = random.randint(1, 6)
    if have == 1:
        mt.up()
        mt.goto(detail['loc'])
        mt.down()
        mt.seth(detail['ang'] + 120)
        tempT = mt.clone()
        flowers.append(tempT)
win.tracer(1)
for i in flowers:
    i.showturtle()
    win.delay(50)

# 擦掉雨滴
for i in range(20):
    drops[i].hideturtle()
    win.delay(50)
# 画风
win.tracer(1)
cloud.backward(1550)
win.tracer(3)
drawWind(-400, 100, -15, 0.7, 4,wind)
drawWind(-470, 300, -15, 0.7, 4, wind)

# 花瓣飘落
win.tracer(3)
for i in flowers:
    leafs_down(i, -300)
for i in flowers:
    i.hideturtle()

# 树叶变黄
win.tracer(1)
grass(-200, -251, (255, 122, 0), gss, 0.6)
for i in leafs:
    i.shape('leafY')
```

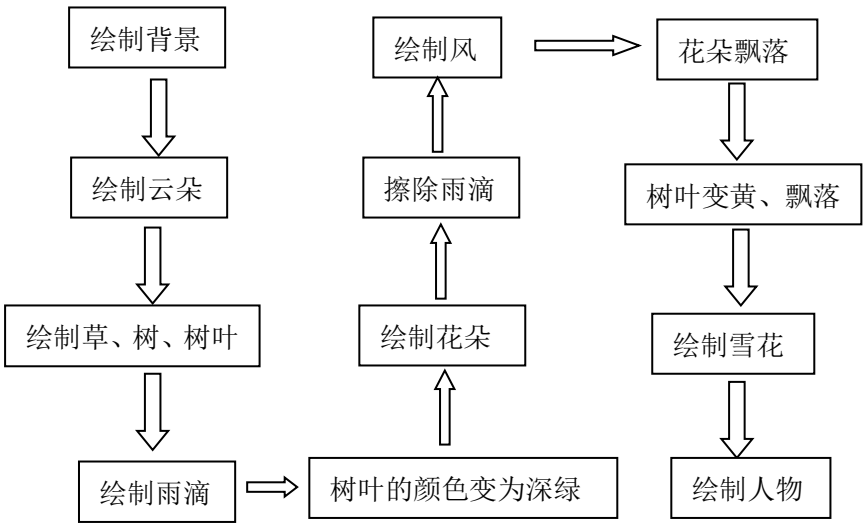
```
        win.delay(50)


    # 树叶飘落
    win.tracer(3)
    for i in leafs:
        leafs_down(i, -300)
        i.seth(random.randint(0, 360))
    gss.clear()
    #擦风
    wind.clear()
    win.tracer(2)
    cloud.fd(1550)


    #画雪
    mt.setheading(0)
    mt.up()
    mt.shape('Snow')
    snows = {i: mt.clone() for i in range(20)}
    win.tracer(1)
    for i in range(4):
        for j in range(5):
            snows[i * 5 + j].goto(-400 + j * 60 + random.randrange(40), -100 + i * 63 +
random.randrange(20))
    for i in range(20):
        snows[19 - i].showturtle()
        win.delay(50)
    for i in leafs:
        i.hideturtle()
        win.delay(50)
    # 画小人
    mt.down()
    mt.setheading(0)
    mt.hideturtle()
    drawpic(30, -100, 1, 1.2, mt, win, 'white.png')
    win.exitonclick()
```

### 2.2.3 程序限制

本程序较为稳定，且不需自行输入参数，因而较少发生错误。目前发现的限制是在不同尺寸的显示屏上，图像的部分区域可能出现无法显示的状况。由于我们是使用 turtle 的自定义

shape 实现图层功能，受 turtle 本身的限制，大幅增加了 cpu 占用，程序运行速度较慢，因而图片可能会在较长时间内保持不变，但这都是正常的！

### 2.2.4 算法流程图




## 2.3 实验结果

### 2.3.1 实验数据

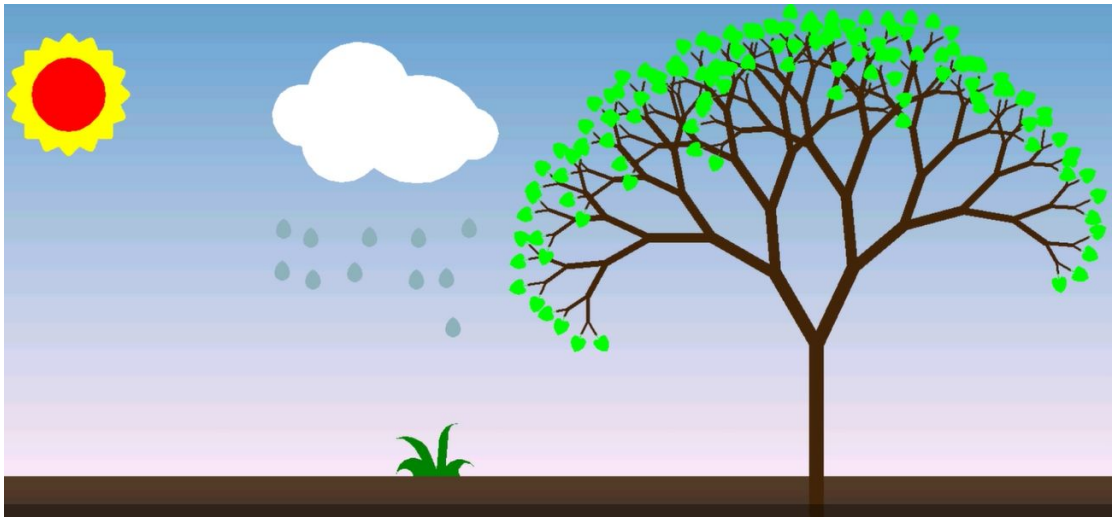
实验环境说明：

- 硬件配置：Intel 酷睿 i7 4710HQ/8g 内存
- 操作系统：Windows10 专业版
- Python 版本：3.53

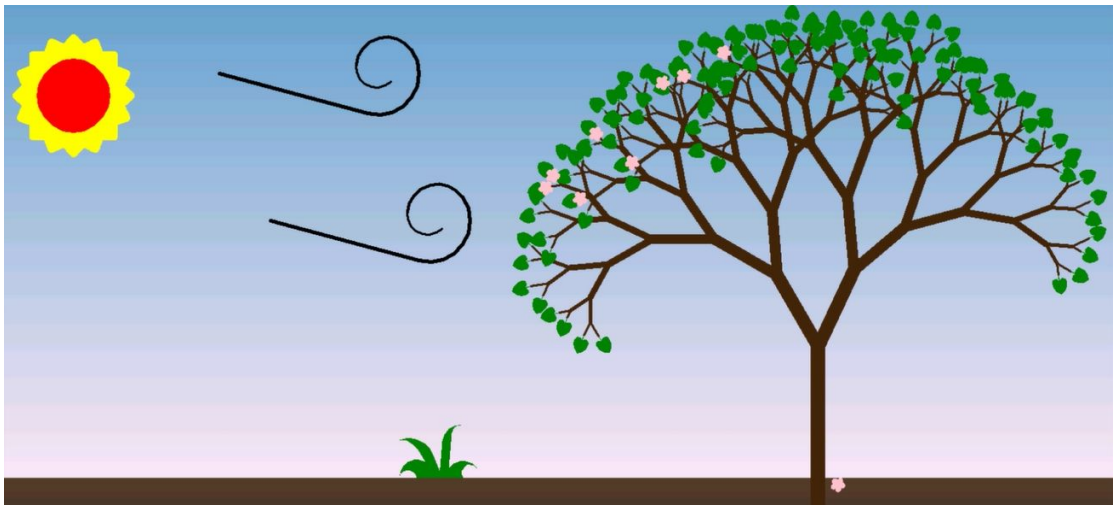
### 2.3.2 作品描述，配以图片

老子云：“天下万物生于有，有生于无。”盘古开天地，阳清为天，阴浊为地。其双目为日月，然后有朝暮之更替，其吐息为风云，然后有四时之变换，其发肤为草木，然后有荣枯之轮回。

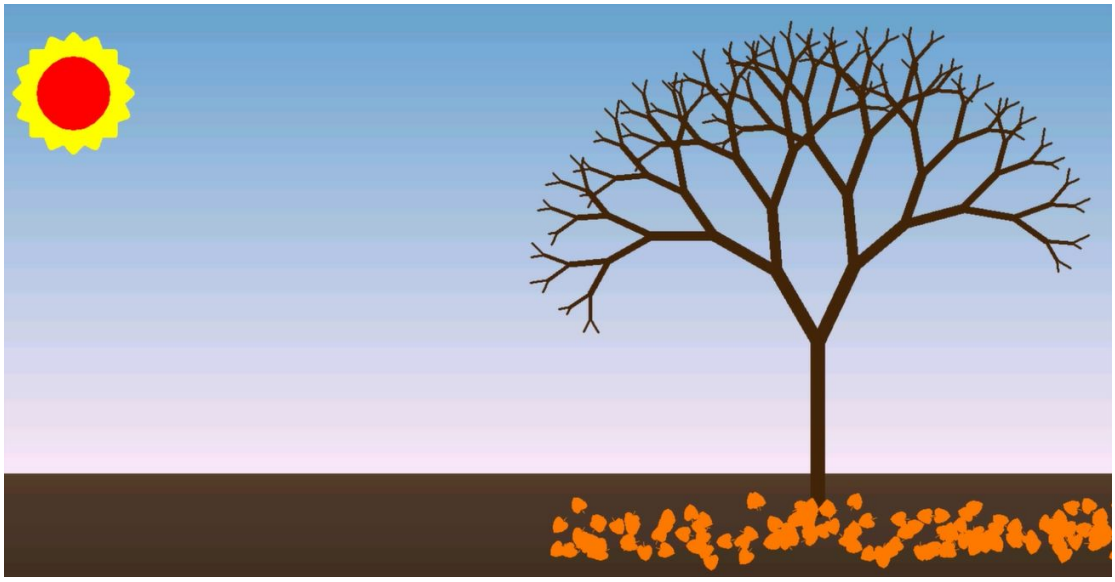
方春始归，青青之色着于枝上，融融之意起于梢头。此一载生机勃发之时，诗人之所怜者，韩文公曰：“最是一年春好处，绝胜烟柳满皇都”，此之谓也。



春老夏还，草木凄凄，一夜风雨之间，便是绿肥红瘦之时。此花落之日，诗人所伤，稼轩“惜春长怕花开早，何况落红无数”之语，古今皆然，独定庵“落红不是无情物”之语，落落有豁然之气。



诗云：“七月流火。”秋末凉风起，草木无情，尽皆飘零。欧阳文忠公曰：“丰草绿缛而争茂，佳木葱笼而可悦；草拂之而色变，木遭之而叶脱。”此秋风之厉也。



秋尽冬至，雨雪菲菲。彼有二女，相与嬉戏，冰雪天地之间，复有冰雪之媛女。

### 2.3.3 实现技巧

在绘制太阳日冕图案的时候，为了减少程序对于系统资源的占用，使用了迭代器。

```
def DotPairs(lower, upper, R, r, d, step, x0, y0):
    if lower == upper:
        yield 0
    else:
        while lower < upper:
            i = lower
            dot1 = getADot(i, R, r, d, x0, y0)
            dot2 = getADot(i + step, R, r, d, x0, y0)
```

```
        yield (dot1, dot2)
        lower = lower + step
```

使用 poly 相关方法获得具有自定义形状的海龟 shape，进而实现多层图案。

## 2.4 实验过程总结

### 2.4.1 分工与合作

黄鑫（组长）：设计背景
张思源：树的主体设计及变换
庞骁：后期
夏运：设计背景
熊轶伟：报告撰写
欧宇航：报告撰写、组织聚餐

第一次组会（4.2）：由于是清明节假期，组员不全在学校，我们采用了线上组会的形式，该次组会大致将我们合作作业的主题定位“四季”，决定通过白线覆盖的方法实现树状态的动态变化，并且基本明确各自的分工。

第二次组会（4.6）：在树的主体及叶子、花的编程任务完成后，我们发现若是采取以白线覆盖修改的方式，在秋天擦去花和叶子时，将会导致树枝的某些部分也被擦去，讨论的结果是通过再画一次树枝来解决枝被擦去的问题。

第三次组会（4.10）：在背景完成后，我们理所当然般得发现，我们发现若是采取以白线覆盖修改的方式，在秋天擦去花和叶子时，将会导致天空的某些部分也被擦去！而且由于天空使用的是渐变色彩，难以“修补”，第三次谈论决定由组长和庞晓两人解决补天问题。

### 2.4.2 经验和教训

多与其他组同学交流，多问其他人问题，参考他人的想法，这一点很重要。

合理的函数设计能够使合作事半功倍，优美的函数应该是封闭的，只能通过变量来与主函数互动。

### 2.4.3 经验与感想

实习作业建议：本次作业的课题难度并不大，六人的小组稍显臃肿。
寄语：同学们，你们要相信，python 比任何游戏都好玩，好好享受提出问题、解决问题的快乐吧。

实习作业后续工作：

比如说我们的作业，下一步可以加入交互性如在可以通过鼠标的移动自己把风“画出来”，比如在秋天花朵和树叶掉下时可以只掉鼠标点击的树叶或花。另外，通过斜二测画法等方法，可以在平面上展示空间关系，也许我们还可以把树做成“三维的”。

另外，可以把我们的 PatternPaintor 函数做成一个游戏，用于模拟玩具“繁花曲线规”，因而需要的是使得程序所画的长度可变，以及能够连续的画出具有不同参数的内摆线，并且实现在一张图上画出多层图案、更复杂的图形，之后再做一个简单的 GUI。

## 2.5 致谢

感谢学长学姐们的作品啦，实在太棒了
《flower》 张沙洲
《未名湖》 陈春含

## 2.6 参考文献

Python 官方文档

# 3 D 组

# 数据结构与算法课程

## 递归视觉艺术实习作业报告

## （四时风物）

（刘丽萍 叶诗婷 李雪岩* 田祯 邓迪）

摘要：用包含花、果、枝、叶的幻想的树将抽象的四季与时间变化展示出来，构成一幅四季的剪影。利用递归算法实现树的分支过程，在分支的末端通过 if 条件语句绘制具体花果枝叶中的哪一个。最终作品展示为一颗融合了四季元素的树与背景。
（简要介绍创意来源，涉及到的递归过程，实现原理，涉及的数据结构与算法，作品呈现效果概述）

关键字：四季剪影的树，递归算法

## 3.1 创意过程与递归思路

### 3.1.1 作品总体介绍

利用递归算法实现树的绘制，并将树枝分支的终点加入四季的元素（背景中也实现），形成一幅四季剪影

### 3.1.2 创意来源

创意来自摄影中的延时曝光技术，我们想借鉴这种方法绘制出树的春夏秋冬。但是由于我们的代码水平做不到让树的风景在一个方向连续的变化，我们只好将各个季节的元素融合在一起，形成一幅大杂烩样式的抽象的树。

### 3.1.3 递归思路

我们的递归算法主要应用在树的分支过程（二叉分枝），不断调用自身产生枝干长度更小的树。

## 3.2 程序代码说明

### 3.2.1 数据结构说明

由于水平有限，我们组的程序中并没有涉及到课程讲授的数据结构，只是编写了许多子函数供 main 函数调用。

### 3.2.2 函数说明

Main 函数：移动海龟的位置，执行特定的函数
Tree 函数：采用递归算法绘制树
Fruit 函数：绘制大小随机的红色果子（用圆圈代替）
Leaf 函数：根据颜色需要，共有三个。分别绘制绿色、黄色树叶（半圆+等边三角形）和橘黄色落叶（两个半圆平移扫出的图形）
Grass 函数：在给定区域内的随机位置绘制绿色 V 型草
Snowman 函数：绘制雪人
Kite 函数：绘制风筝

### 3.2.3 程序限制

Tree 函数的初始长度如果过大的话，递归层数太多造成画图时间过长，cpu 及内存的过多占用也会造成卡顿。参数必须为正整数，否则程序报错。

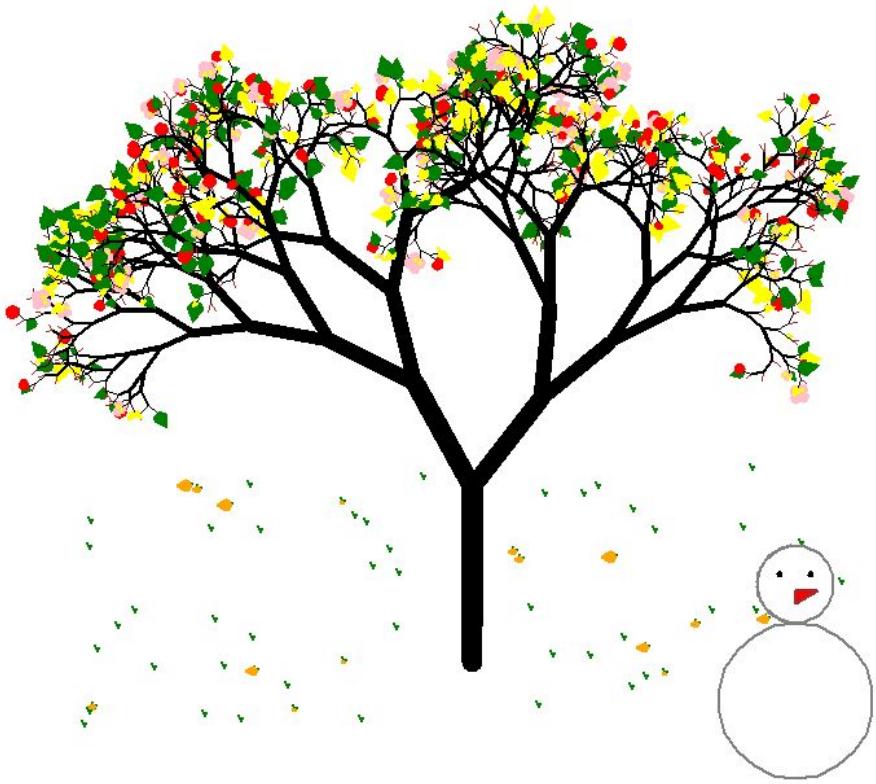
## 3.3 实验结果

### 3.3.1 实验数据

实验环境说明：

- 硬件配置：CPU: I7-3520m　RAM: 8g
- 操作系统：Windows 8.1
- Python 版本：Python 3.5.1

### 3.3.2 作品描述



盛夏的绿荫，秋日的金黄，成熟的果实，枯萎的枝干，它们都生长在这棵四季树上，还有那不合时宜的青草，落叶和雪人。如果说四季变化是大自然赐予温带气候区的礼物，那把这些景物捕捉在一张照片里，会远比这美得多，却又大概是这个样子吧。

### 3.3.3 实现技巧

并没有发现什么黑科技_(:з)<)_ 能不出 bug 的跑出来已经很是欣慰。

## 3.4 实习过程总结

### 3.4.1 分工与合作

- 创意： 刘丽萍、李雪岩
- 编程：李雪岩、刘丽萍
- 写报告小能手：田祯、邓迪
- PPT 制作：刘丽萍、叶诗婷

- 中期报告主讲人：叶诗婷

讨论方式有当面讨论与微信群内讨论，由于大四狗们以浪浪浪为主，不能总是凑在一起，所以微信群内讨论更多。

第一次会后合照

检查代码的邓迪同学


微信记录一则

### 3.4.2 经验与教训

经验是大家要广开脑洞，不要束缚想象力。教训是不要拖延，要尽早动手，并且组里一定要有个大腿可以抱，不要都是大四狗 23333.

### 3.4.3 建议与设想

或许来年的作业可以换成递归的其他形式，比如走迷宫。或者利用其他数据完成其他类型的作业。

## 3.5 致谢

感谢刘丽萍同学提供了设计思路与灵感，促成了本次作品的诞生；感谢参与了中期 ppt 制作与展示的刘丽萍同学与叶诗婷同学，帮助中期因故缺席的组长完成了中期报告。感谢邓迪同学与田祯同学早期在朋友圈、互联网对燕园春景等摄影图片的搜集，提供了早期创作的实物支持。感谢组长李雪岩同学不厌其烦的将大家的意见尽力用代码实现。最后感谢陈老师以及各位助教，提供给我们一个发挥自己创造力的机会，并且是开开心心地写代码:)

## 3.6 参考文献

1.《数据结构与算法》课程讲义
2.脚本之家网站 http://www.jb51.net/

# 4 E 组

# 数据结构与算法课程

## 递归视觉艺术实习作业报告

## 朱家角古镇

E 组：刘晨 薛阶祺 朴承主 崔炫娥 詹煌

摘要 我们的创意来自于课堂上老师呈现过的三维谢尔宾斯基三角形，我们希望用不同颜色线条将其具体实现，主要的递归过程和二维的谢尔宾斯基类似，包括绘制指定边长的正四面体，寻找等分点，继续绘制正四面体，为了使画面更加丰富，我们增加了绘制倒影的方法，绘制长方体的方法，以及为线段着色的方法。总体实现了绘制带有长方体倒影的谢尔宾斯基五面体。

关键字 三维谢尔宾斯基三角形、三维图、五面体

## 4.1 创意过程与递归思路

### 4.1.1 作品总体介绍

总体来说我们实现了利用递归绘制三维谢尔宾斯基五面体，并为其线段添加了颜色，递归的思想就是将三维的大谢尔宾斯基五面体化为三维的小谢尔宾斯基五面体，最简单的情况是只画一个五面体。通过 matplotlib 库中自带的函数实现了绘制二维三角形，三维五面体。

在作品中我们包含了几个主要的函数，分别为绘制五面体，绘制谢尔宾斯基五面体，绘制谢尔宾斯基分层次、级数的五面体集合以及一些应用类的函数。

### 4.1.2 创意来源

我们的创意来自于老师上课呈现的立体谢尔宾斯基三角形，希望自己动手实现该图形并为其添加色彩，组内朴承主和崔炫娥同学负责先期确定视觉指导，并与我们讨论了可行性之后确定采用线条着色的方案，刘晨和薛阶祺同学负责完成具体的代码，詹煌负责代码的复查和完成报告。

### 4.1.3 递归思路

正如日前李克强总理到访北大时所叮嘱的，“凡大事必做于细”，目前我们的代码思路就是，为了画出一个整体的谢尔宾斯基五面体，先着眼于最简单的问题，如何绘制单个三维五面体。之后，在绘制谢尔宾斯基五面体的函数中加入一个参数 degree 来描述我们想要绘制的层数，这里为了视觉效果，我们默认 degree＝3，之后每绘制一层五面体该参数减一，直至绘制单个五面体这一最简单的状况。

## 4.2 程序代码说明

### 4.2.1 数据结构说明

在这部分我们没有创新，主要采用的数据结构是列表。

### 4.2.2 函数说明

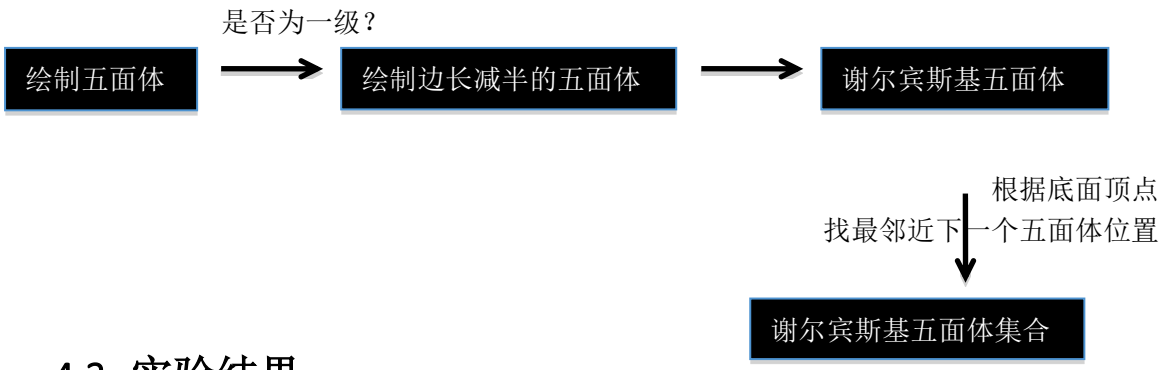
引入了 matplotlib 库，实现了：
1. 在三维空间进行绘制
2. 利用函数构成曲线
3. 界面化显示 pli.show()

主要函数包括：
1. draw_triangle，为绘制单个五面体的函数。
2. sierpinski：利用递归绘制谢尔宾斯基五面体
3. sub_sier：绘制上述五面体的对称五面体
4. sierpinskigalaxy：绘制从中间向外层数减小的谢尔宾斯基五面体集合
5. getnextpoints：从五面体一个顶点出发，利用比例计算得到以此顶点为一个顶点的另一五　　面体。

### 4.2.3 程序限制

目前没有发现这方面的风险，不过确实存在层数大于 7 时电脑有可能跑不出来的情况。

### 4.2.4 算法流程图



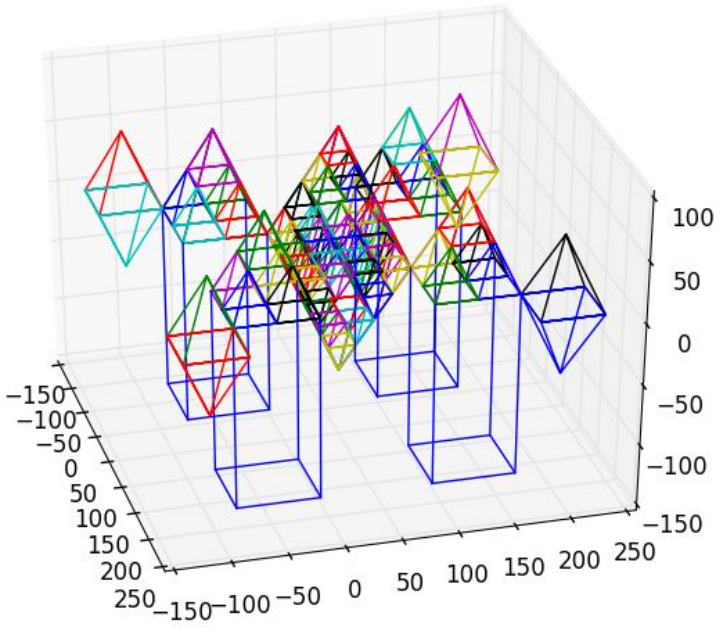

## 4.3 实验结果

### 4.3.1 实验数据

实验环境说明：
硬件配置：处理器：2.7GHz Intel Corei5
内存：8GB 1867 MHz DDR3
图形卡：Intel Iris Graphics 6100 1536MB
操作系统：OS X Yosemite 10.10.5
Python 版本：python 2.1.7

### 4.3.2 作品描述

作品实现了三维空间中的谢尔宾斯基五面体，默认绘制了一大八小共 9 个立体谢尔宾斯基五面体，其中一大四小均有不同颜色的对称投影，剩余四个的底部有长方体，所有线段均着有颜色。之所以命名为朱家角古镇是因为朴承主与崔炫娥同学提到五面体的投让她们想起来朱家角古镇上的房屋与小河。




图：递归视觉艺术作品

图：朱家角古镇

### 4.3.3 实现技巧

将顶点坐标作为数组里面的一个元素，可以构成二维列表，这样可以通过相应位置编号直接对某一顶点进行操作。

## 4.4 实习过程总结

### 4.4.1 分工与合作

小组成员各取所长，刘晨与薛阶祺同学代码能力强，有丰富编程经验，因此专注于创意的实现，朴承主与崔炫娥同学有较高审美水平，对造型以及配色有较好的感觉，因此负责提供创意，詹煌同学主要负责代码的复查以及报告的撰写

### 4.4.2 经验与教训

实习过程中常遇到的问题是脑洞过大难以具体实现，因此以后在进行类似的实习时应注意创意与实现之间的平衡，本组的优点在于分工明确各自发挥所长，因此工作效率很高，缺点就是对每个人的代码水平没有很大的锻炼，在以后的实习中可以考虑更加并行化的写代码方式，争取让每个人都能参与到写代码中。

另外有几处想要利用递归实现的未能成功实现，最后才用了暴力方法，还可以进一步完善。

### 4.4.3 建议与设想

在创意方面除了自选题目之外可以考虑给出一些可选的课题，以便大家参考和选择。

## 4.5 致谢

感谢陈斌老师上课时展示的三维谢尔宾斯基三角形，为我们直接提供了灵感。

感谢易超助教和陈春含助教的帮助，让我们顺利完成了代码的编写。

## 4.6 参考文献

Sierpinski triangle：https://en.wikipedia.org/wiki/Sierpinski_triangle

Matplotlib：http://matplotlib.org/

# 5 H 组

# 数据结构与算法课程

## 递归视觉艺术实习作业报告

## 西子湖畔雷锋塔

冀锐、李银、陈雨豪、高红涛、陈喆、吉梁

摘要：本作品的根本创意来源是雷峰塔的传说，使用了 python 自带的海龟作图系统(Turtle Module)，实现了递归可视化。本作品将基本的分形树递归、圆形递归、螺旋形递归结合在一起，最终将呈现出一个完整的图像，并能够与预设的故事相互契合。

关键字：湖，雷锋塔，分形树，turtle，sky，tree，leaf，flower，tower

### 5.1 创意过程与递归思路。

本作品最终呈现的内容将是一幅完整的图画，分别由背景的天空与湖水、塔、荷花、荷叶、生长的树这五个单元构成，完全是由 turtle 系统所画。所以本作品采用了引用 5 个函数的方法，这五个单元分别进行设计，最终合成到一张图像里。为了使最终的图像能够展现出想要的效果，在编写 5 个函数的时候要先计算其位置，将坐标储存在元组里，从而确保引用时不会出现偏差。

在考虑应该如何设计这次作品，也就在进行最初的创意构想的时候，

上一届学长学姐留下的作品无疑提供了一些最初的启发。但为了防止在创意上的重叠从而导致创意的流失而使最终作品索然无味，在吸收学长学姐的作品的精华时也要注入团队的自己的想法。尽管本组的作品依旧是围绕湖与塔的经典图像来展开，但本组从分形树的设计开始，想追求完全由 turtle 的模组来把画面中的所有图形画出来。也就是说，如果以往的“湖与塔”还要搭配一些背景的照片来实现，那么本组的作品将完全通过 python 来画出，从而充分实现了函数递归的图示化。除此以外，本组还考虑到图像与想要表达的意境或者内涵相呼应，所以试图将其故事化，增加趣味性。

故事概要：西湖之上接天莲叶与映日荷花相互掩映，远处的雷锋塔黝黑而厚重，一株新绿引入眼帘，展开一段传说……

由于本组的作品是由五个函数最终构成的，所以分配任务相对明晰。由李银同学负责荷叶的函数，陈喆同学负责塔的函数，高红涛同学负责荷花的函数。虽然本组看似分而治之，但正如前面所说，因为要确保图像之间的坐标关系，组内也经常要交流信息，最终还要由组长冀锐同学进行统筹。

本作品中的树函数是有简单的分形树递归改编而来；而背景天空的函数则是由一条条线的函数来递归，并实现了从上往下的渐变色；而荷花与荷叶则是由一条一条极小的线来构成弧形，从而绘画出不规则的图形；而塔以及路则是由很多的菱形递归，并有随机展现出不同的大小，从而实现参差不齐的形状。

## 5.2 程序代码说明

主程序应用了 5 个函数，分别是 sky、tower、flower、leaf 以及 tree。而正如名字所示，sky 函数是用来描绘背景的天空、湖水以及小路的；tower 函数是用来描绘的雷锋塔；flower 函数用来描绘荷花；而 leaf 函数则用来描绘远处较小以及近处较大的荷叶；最后，tree 函数则是来描绘分形树，并有一定的图式的时间。

在 sky 函数中，colorline 函数是用来描绘最基本的色线的，以供递归调用。而 sky、water 以及 road 则是天空、水以及路。

而在 leaf 函数中，则有 leaf_1-5 的函数，分别描绘了 5 片荷叶。

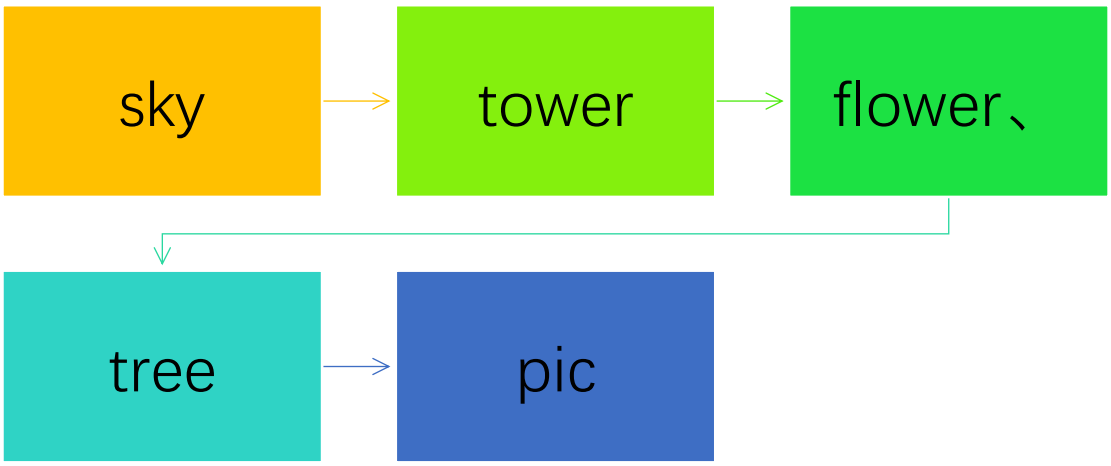
而 tower 函数中，ta 函数是用来描绘菱形图像来供递归的。

最后在 flower 函数中，petal、petalF 以及 petalT 函数是描绘花瓣的，而 root 函数则是描绘茎部，最后 flower 函数则描绘荷花本身。

程序调式结果：

主程序经过多次调试，本组认为在内存较小的时候程序会出现崩溃的情况。而在调试的情况下(windows8 以及 windows10 系统下)会运行正常。

算法流程图：




## 5.3 实验结果

实验环境说明：

CPU：Intel Core i7-6600U CPU @2.60Hz 2.81Hz

内存 16.0GB

操作系统 Windows 10 专业版(64 位)

Python 版本 3.5.1150

作品描述：

本作品展示的是湖畔塔的景象，最终呈现的图像由荷叶、荷花、树枝以及路和塔构成，其中塔与小路是黑色粗线条，荷叶是绿色，荷花是粉色构成。而背景的天空与湖水则具有从天蓝色向白色渐变的特点。而构图上，塔位于图像中央偏右，荷叶分布在前后，荷花在图像下侧，树枝从图像左上角生长出来。整个图像具有立体感。

## 5.4 实习过程总结

本小组在聚齐成员后，采用微信上交流构想，在宿舍内交流代码的方式。其中几次大的组会安排在宿舍内展开。组会主要为了确定大致的构想，以及安排好任务。而组员也时常会提出较好的意见，是本组最终的作品更加完善。

尽管本组对于最终做出的程序大体上是满意的，但在过程中仍然感到有许多值得改进的地方:首先还是拖延症的问题，再确定了主题之后，由于与考试的冲突，本组的很多任务都是在 ddl 之前赶时间完成的，从而是质量没有了保障；其次，再一开始确定主题的时候，组员之间的讨论往往过于思维跳跃，没有一定的根基，以至于浪费了大量的时间；最后，尽管组员之间有充分的交流，但大部分时候组员还是依旧各干各的，没有进行合理的团队合作。

对于往后实习作业，本组希望能够充分实现团队合作的方式，希望能够在个人努力与团队协作之间找到平衡。

## 5.5 致谢

陈斌老师

# 6 J 组

# 数据结构与算法课程

## 递归视觉艺术实习作业报告

## TYPHON 小组

## （POLY LOWER STATION）

（陈一潇，江欣余*，刘旭林，卢亚敏，李姿蓉，张书菀）

**摘要**：“低多边形”是当下十分热门的设计风格，除了网上的图形设计和动效设计，现实中的杂志、电视中这种风格也多有体现。这种设计风格的特点是低细节，面又多又小，高度渲染，经常配以柔光效果。我们运用递归思想，不断分割三角形，通过矩阵查找最终将原图以三角形来拟合，实现低多边形效果。

**关键字**：低多边形 三角形 矩阵 递归

## 6.1 创意过程与递归思路

### 6.1.1 作品总体介绍

我们小组的作品以低多边形概念为灵感来源，通过递归思想和数据结构预算法，实现了

将输入的原画作自动转化为低多边形概念作品的程序。在程序中，我们使用了递归思想，将原图用三角形按递归算法从大到小分割，最终呈现出低多边形的拟合效果，呈现原图的色彩与构图。

### 6.1.2 创意来源

在小组讨论之前，大家也各自搜索了一些资料，进行了自己的思考，但我们的初步思考主要还是受上课展示的分形树作品影响，局限在递归画图的范畴，如对水墨画、风景照以及复杂分形图案等的实现。之后，我们开始试图发掘更多的可能性，考虑使我们的作品更动态，更具交互性。比如设计一个小程序，并配以分形画作的进入画面。还有李姿蓉同学提出通过连续作画，创造更复杂的画面，如实现多幅世界名画。不过这样的作品难度和之前基本持平，没有多大改变和创新。

之后，陈一潇同学介绍的一些他所了解的新颖的概念拓宽了思路。首先是元胞自动机的概念。元胞自动机(Cellular Automaton，简称 CA)，也有人译为细胞自动机、点格自动机、分子自动机或单元自动机。由冯诺依曼在 20 世纪 50 年代发明。它是一时间和空间都离散的动力系统。散布在规则格网 (Lattice Grid)中的每一元胞(Cell)取有限的离散状态，遵循同样的作用规则，依据确定的局部规则作同步更新。大量元胞通过简单的相互作用而构成动态系统的演化。具体讲,构成元胞自动机的部件被称为"元胞"，每个元胞具有一个状态。这个状态只取某个有限状态集中的一个，例如或"0"或"1"，或者是 RGB 颜色中的一种等等。这些元胞规则地排列在被称为"元胞空间"的空间格网上，它们各自的状态随着时间变化。而根据一个局部规则来进行更新，也就是说，一个元胞在某时刻的状态取决于、

而且仅仅取决于上一时刻该元胞的状态以及该元胞的所有邻居元胞的状态。元胞

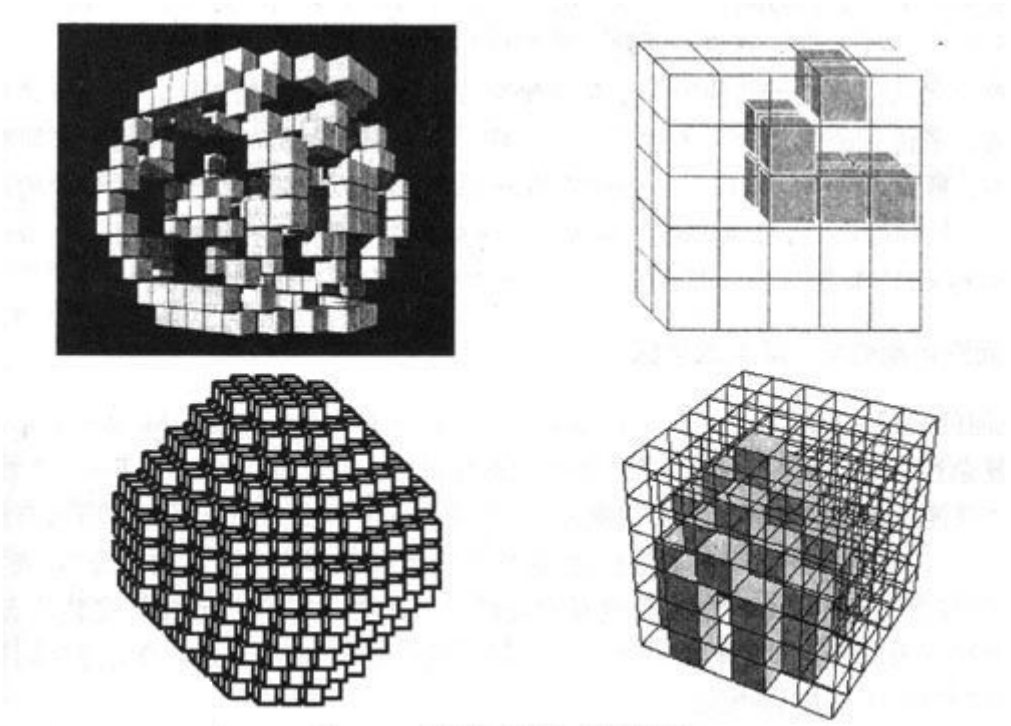


图 2-8　三维的元胞自动机模型

依照这样的局部规则进行同步的状态更新，整个元胞空间则表现为在离散的时间维上的变化。我们运用递归算法便可以实现元胞自动机。经典游戏如贪吃蛇、俄罗斯方块，可以看成元胞自动机。元胞自动机模型用简单的规则和计算进行多次迭代，可以产生类似人工生命的复杂系统，而这些系统中利用分形分析会发现有明显的自相似性，即分形特征。

另一个概念叫做低多边形。继拟物化、扁平化（Flat Design）、长阴影（Long Shadow）之后，低多边形（Low Poly）掀起了全新的设计风潮。低多边形实际是把多色元素，用三角形（或其他基本图形）分割，每个小三角形的颜色，取自原多色元素的相应位置。低多边形用的是纯色填充，形状的识别靠颜色深浅，所以相邻的三角形就不能一个颜色，需要间隔上色。目前，在平面设计，3D 网页设计，图标设计等方面有广泛应用，并有相应软件可以实现低多边形效果。

举例来说，用计算机绘制山峦异常艰难，地表起伏的山峦，要形成细节足够丰富的效果，必须以数以百万计的小三角形构成，即便是在每一帧画面中，小三角形的数目都会达到成百上千。

但如果计算机处理图形的计算能力和内存很受限制，就只有使用尽可能低的算法复杂度，才有可能来绘制出山峦的细节。如果将山峦的形状概括成为数量比较少、面积比较大的一些三角形，计算机很容易实现快速处理；但如果需要精细描绘出细节的起伏，就需要增加三角形的数量，缩小每个三角形的面积。利用分形理论便可以解决这个问题。

早在 1967 年，数学家 Mandelbrot 就在美国权威的《科学》杂志上发表了题为《英国的海岸线有多长？统计自相似和分数维度》(How Long Is the Coast of Britain? Statistical Self-Similarity and Fractional Dimension）的著名论文。

英国的海岸线有多长？不同的测量尺度会带来不同的结果。Mandelbrot 在研究海岸线长度时发现，地理中的曲线长度与其曲线细节有着很大的关联，并且这些曲线通常表现出具有无限长或者是无法定义的特性。幸运的是这些曲线同时表现出了一个很好的特性——「自相似性」。取曲线的一小部分等比例放大后，你会惊奇地发现放大后的部分曲线的形状与原来的整体具有很大的相似性。Mandelbrot 指出自相似性是一个十分强大的工具，在各个领域均有很大的作用。1975 年，他创立了分形几何学 (Fractal Geometry)。在此基础上，形成了研究分形性质及其应用的科学，称为分形理论。

采用迭代的方式不断通过嵌套递归，计算机在有限的硬件条件下，便得以快速生成精细、细节更真实的山峦表面。

显然，对于 3D 模型而言，越多的多边形精细，但也越难于处理。当多边

形” 颗粒” 较大，“分辨率” 较低时，处理器和内存的压力更小。与之相对应，在同等渲染引擎的算力之下，每一帧的场景中多面体数目更少，就意味着体验更流畅。因此无论是早期还是当下的电脑游戏，还是今天移动互联网的带宽限制，都使得低多边形因为 3D 实时渲染更容易处理，而大行其道。

通过分形迭代绘制画作的算法至今仍然被 Geek 们所钟爱，在计算机算力和


内存足以应付 3D 贴图，甚至还能加入各种绚烂特效的今天，分形艺术显得更加多元和绚烂。每年的国际分形艺术大赛的作品，都被人所津津乐道。

国际分形艺术大赛作品《浮潜》，画作是通过分形迭代算法自动生成的。

设计师们主动选择了这种风格，作为实现可能性的一种方式，他们仅仅凭借风格选择的考量，来推崇 Low-Poly，正如复古风潮、蒸汽朋克和像素风一样，成为了一种架空感强烈、包含向过往致敬意味的选择。

新世纪进入第二个十年时，Low-Poly 风格也渐渐进化为一种设计语言，越来越广泛地应用在从平面设计到 3D 场景、从工业设计到建筑的构成之中。从 QQ 6.0 的登录界面，三星手机的默认屏保，再到 2014 年上映的 Disney 影片《灰姑娘》中的水晶鞋，三宅一生设计师 Issey Miyake 设计的手袋，甚至艺术家 David Mesguich 的雕塑……无数出色的 Low-Poly 设计，宣告着它的复兴与重生，成为继扁平化、长阴影之后的流行趋势。

然而在风格兴起的背后，Low-Poly 依然保留着数字化的特征。首先，它可以通过分形迭代算法获得，其次，它与数据衔接紧密。和数据、运算之间天然的纽带，也使得它成为大数据时代，为基于数据的产品设计提供了更多的方式和灵感。

低多边形这个概念既与递归的数据结构紧密联系，又反映了当下的视觉艺术潮流，极好的贴合了本次实习的主题，一经提出便受到大家的青睐。之后的“开脑洞”过程也主要围绕这一概念展开。如张书菀同学提到的类似幻灯片过渡动画的图形消解效果。江欣余同学受到自己把窗帘图案想象成迷宫消遣时间的经历启发，提出既然我们可以创作出简单的几何图形组合成的画面，我们也许还可以通过对画面的分析，将其转化为一幅迷宫。甚至可以在编写一段走迷宫的程序，既检验了所成迷宫的可玩性，又可再次运用递归。小组成员们认为，这个想法很新颖，但有一些较难解决的问题。一是决定迷宫的“墙”条件，要考虑明暗，颜色的多种因素，也受画作本身的风格制约；另一个问题是考虑到画作分析后得出曲线的不可预知，要画出两道平行延伸的墙也许会有一定难度。墙的交叉等问题也需要考虑。我们希望在完成低多边形创作程序之后再考虑迷宫的实现，但最终由于时间有限，没有能够实现。

最终我们的构思是，对平常的低多边形创作反其道而行之，变为随意选取一幅已有的普通的画作或照片，经过边界分析，采用递归方法分割三角形并插值上色，将图像转化为低多边形风格作品。这样，我们做出的程序类似一个软件，理论上可以源源不断的生产低多边形风格艺术作品。这种思路也类似 GIS 中的一些遥感图像解译应用以及 GIS 数据管理的操作，如确定边界，分隔地物，颜色重采样和栅格矢量数据转换等操作。

### 6.1.3 递归思路

本代码将递归思想体现在了三角形分割中。采用矩阵查找符合条件的点，依次分割为从大到小的三角形，这些三角形内的色彩较一致，面积较小。以三角形为基本元，拟合了原图的构图，实现的低多边形的效果。

## 6.2 程序代码说明

### 6.2.1 数据结构说明

初步思路是，对于任意选择的一张图，先对这张图进行初步的识别，提取其像素信息，然后计算各像素点与周围像素点信息的差值，且分别比较。判断出符合条件的点以后，划分三角形，记录像素点的平均差值和像素平均颜色后，再进行判断，当三角形足够小或色差足够小时，就对三角形上色。

程序分步思想解读为：

1、调用函数，创建类并插入图片，设置好初始值

2、获取像素信息，并计算差值

这一步起到了奠基作用。对于所选图形，对其实现低多边形效果转换，需要在这一步判断出操作点。由于原始图画的色彩缤纷、构造多样，计算机实现低多边形效果转换的设计思路关键在于，逐点分析原图的像素信息（包

括色彩等），且对这些信息进行分析，即判断像素信息是否“相近”，色彩区分度是否明显。对于相近的像素信息，原图中表现出的效果为平滑连续，色彩过渡自然顺畅，对于不相近的像素信息，原图中表现出的效果为突跃破碎，色彩反差大，变化大，没有流畅过渡的效果。

3、划分三角形

这一步是低多边形的核心处理。在前一步确定划分方法以后，这一步将对选定的图画进行三角形分割处理，循序渐进，最终实现低多边形的惊艳效果。当三角形的面积足够小或者色差足够小时，就对三角形进行上色。使用该函数，可以较为精准的分析出原图各处的色彩指数，予以适当区分并进行数字再处理。需要指出的是，相邻三角形的涂上的颜色是不一样的，这样才能实现三角形分割效果，用小三角形微元近似模拟出原图的效果，实现低多边形效果转换。

初版代码如下：

```
from PIL import Image
import math

class Node():
    def __init__(self,item,x,y):
        self.data=item
        self.x=x
        self.y=y

path="3.jpg"
img=Image.open(path)
length=img.size[0]
width=img.size[1]
rec=list()
min_sq=400
min_dpix=10
move=[0,1,0,-1,1,0,-1,0]

def GetPixel(im,x,y):#获取像素信息
    p=[im.getpixel((x,y))[0],im.getpixel((x,y))[1],im.getpixel((x,y))[2]]
    return p
```

```
def Get_Pixel_Dis():
    for x in range(length):
        for y in range(width):
            # 计算该像素点与周围四个像素点的差值,分别比较
            stander = GetPixel(img, x, y)
            distant = 0
            for i in range(4):
                tx = x + move[i * 2]
                ty = y + move[i * 2 + 1]
                if tx >= 0 and tx < length and ty >= 0 and ty < width:
                    tmp = GetPixel(img, tx, ty)
                    distant = distant + abs(tmp[0] - stander[0]) + abs(tmp[1] - stander[1]) +
abs(
                        tmp[2] - stander[2])
            rec.append(distant)

    return rec

def Same(x1,y1,x2,y2,x3,y3,x,y):
    vector1=[x1-x3,y1-y3]
    vector2=[x2-x3,y2-y3]
    tv1=vector1[0]*vector2[1]-vector2[0]*vector1[1]

    vector1 = [x1 - x, y1 - y]
    vector2 = [x2 - x, y2 - y]
    tv2 = vector1[0] * vector2[1] - vector2[0] * vector1[1]

    if tv1*tv2 >= 0:
        return True
    else:
        return False

def inside(x1,y1,x2,y2,x3,y3,x,y):
    if Same(x1,y1,x2,y2,x3,y3,x,y) and Same(x1,y1,x3,y3,x2,y2,x,y) and
Same(x2,y2,x3,y3,x1,y1,x,y):
        return True
    else:
        return False

def CountS(x1,y1,x2,y2,x3,y3):
    victor1=[x2-x1,y2-y1]
    victor2=[x3-x1,y3-y1]
```

```
    return 0.5*abs(victor1[1]*victor2[0]-victor1[0]*victor2[1])

def Color(x1,y1,x2,y2,x3,y3,aver,t):
    print(x1,y1,x2,y2,x3,y3)
    lmax = max(x1, x2, x3)
    rmax = max(y1, y2, y3)
    lmin = min(x1, x2, x3)
    rmin = min(y1, y2, y3)

    aver[0]=int(aver[0]/t)
    aver[1]=int(aver[1]/t)
    aver[2]=int(aver[2]/t)

 for x in range(lmin, lmax + 1):
        for y in range(rmin, rmax + 1):
            if inside(x1, y1, x2, y2, x3, y3,x,y) == False:  # 判断点是否在内部
                continue
            img.putpixel((x,y),tuple(aver))

def Build_Trangle(x1,y1,x2,y2,x3,y3,rec):#划分三角形
    lmax=max(x1,x2,x3)
    rmax=max(y1,y2,y3)
    lmin=min(x1,x2,x3)
    rmin=min(y1,y2,y3)

    average=0#记录像素点平均差值
    pmax=0
    pmax_x=pmax_y=0
    aver=[0,0,0]#记录像素平均颜色
    t=0
    for x in range(lmin,lmax+1):
        for y in range(rmin,rmax+1):
            if inside(x1,y1,x2,y2,x3,y3,x,y) == False:#判断点是否在内部
                continue
            t=t+1
            pos=x*(width-1)+y
            #print(x,y,pos,rec[pos].data)
            average=average+rec[pos]
            tmp=GetPixel(img,x,y)
            aver[0]=aver[0]+tmp[0]
            aver[1]=aver[1]+tmp[1]
            aver[2]=aver[2]+tmp[2]
            if rec[pos] > pmax:
                pmax=rec[pos]
```

```
                pmax_x=x
                pmax_y=y
    if t == 0:
        print(CountS(x1, y1, x2, y2, x3, y3))
        print(x1,y1)
        print(x2,y2)
        print(x3,y3)
        raise ZeroDivisionError("t is 0")


    if CountS(x1, y1, x2, y2, x3, y3) < min_sq:#面积足够小上色
        #print(CountS(x1, y1, x2, y2, x3, y3))
        Color(x1, y1, x2, y2, x3, y3, aver, t)
    elif average/t < min_dpix:#色差足够小上色
        Color(x1, y1, x2, y2, x3, y3, aver, t)
    else:
        #print("1")
        Build_Trangle(x1,y1,x2,y2,pmax_x,pmax_y,rec)
        Build_Trangle(x1,y1,pmax_x,pmax_y,x3,y3,rec)
        Build_Trangle(pmax_x,pmax_y,x2,y2,x3,y3,rec)

rec=Get_Pixel_Dis()
Build_Trangle(0,0,length-1,0,length-1,width-1,rec)
Build_Trangle(0,0,0,width-1,length-1,width-1,rec)
b=img
b.save('2.jpg')
```

后来，我们试运行了该代码。Rec 该 orderlist 储存扫描出来的前 max_p 个锐化点，然后每次递归把区域内的最大锐化点摘取出来，进行下一次递归，直至找完 rec。但是我们发现运行时间颇长，时间复杂度亟需优化，且极易达到最大递归深度。后来对此程序进行了微调，把预处理的过程简化了一些，但在 orderlist 上遇到了瓶颈。但是仔细分析后我们发现，速度瓶颈不是在查找锐度上，程序中已预先找好了所以锐度最高的点，设置了有序表，把图里所有的锐度最高的点保存起来，然后逐个判断是否在区域内进行递归，但是找锐化点的依据还需要更严谨一些，色彩赋平均值的机制也需要更完善一些。

因此我们需要更换思路，鉴于 build_trangle 第一层边界的选取对整体效果的影响极大，因此我们需要适当的处理三角形的边界，且查找边缘后需要分隔三

角形的长边。后来我们采用直接用三角形划分的方法，更换终止条件并反复调参，每个区域都单独计算，在最开始取一个点，分成四个，然后在每个三角形里面取一个点，再连顶点分 3 份。这样一来，运行速度大大提高，效果也优化了不少。

这是我们改进后的代码

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 10 10:37:45 2016

@author: Xiao
"""

from PIL import Image, ImageDraw
import numpy as np
import scipy as sp
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.animation as anim

def thumbnail(path, width):
    """make a thumbnail of the image loaded from the given path, with given width"""
    img = Image.open(path)
    x,y = img.size
    y = int(y / x * width)
    x = width
    img.thumbnail((x,y))
    img.save("tb_" + path)

def pre_proceed(path):
    """preproceeding the given image from the path, return the gradient of every point."""

    raw_img=Image.open(path)
    raw_w, raw_h = raw_img.size
    img = plt.imread(path)
    height, width = img.shape[0], img.shape[1]

    r = img[:, :, 0]
    g = img[:, :, 1]
    b = img[:, :, 2]

    #using numpy to calculate the gradient toward four direction of every point
```

```
    d_up = np.zeros(r.shape)
    d_down = np.zeros(r.shape)
    d_left = np.zeros(r.shape)
    d_right = np.zeros(r.shape)
    d_up[1:] = (r[1:] - r[:-1])**2 + (g[1:] - g[:-1])**2 + (b[1:] - b[:-1])**2
    d_down[:-1] = d_up[1:]
    d_left[:, 1:] = (r[:, 1:] - r[:, :-1])**2 + (g[:, 1:] - g[:, :-1])**2 + (b[:, 1:] - b[:, :-1])**2
    d_right[:, :-1] = d_left[:, 1:]

    # average mode to calculate the "distance"
    distance = (d_up + d_down + d_left + d_right) / 4

    return img, height, width, r, g, b, distance

def tri_shape(height, width, x1, y1, x2, y2, x3, y3):
    """return a matrix of ones and zeros,
    the point in the given triangle which is build by the three x and ys is one."""
    shape = np.zeros((height, width), int)
    inside = np.zeros((height, width), int)

    x = np.arange(width) / width
    y = np.arange(height) / height
    x.shape = (1, width)
    y.shape = (height, 1)
    x1 /= width
    x2 /= width
    x3 /= width
    y1 /= height
    y2 /= height
    y3 /= height

    shape[(((x - x1) * (y - y2) - (x - x2) * (y - y1)) * \
    ((x3 - x1) * (y3 - y2) - (x3 - x2) * (y3 - y1)) >= 0) &  \
    (((x - x2) * (y - y3) - (x - x3) * (y - y2)) * \
    ((x1 - x2) * (y1 - y3) - (x1 - x3) * (y1 - y2)) >= 0) & \
    (((x - x3) * (y - y1) - (x - x1) * (y - y3)) * \
    ((x2 - x3) * (y2 - y1) - (x2 - x1) * (y2 - y3)) >= 0)] = 1
    inside[(((x - x1) * (y - y2) - (x - x2) * (y - y1)) * \
    ((x3 - x1) * (y3 - y2) - (x3 - x2) * (y3 - y1)) > 1e-3) &  \
    (((x - x2) * (y - y3) - (x - x3) * (y - y2)) * \
    ((x1 - x2) * (y1 - y3) - (x1 - x3) * (y1 - y2)) > 1e-3) & \
    (((x - x3) * (y - y1) - (x - x1) * (y - y3)) * \
    ((x2 - x3) * (y2 - y1) - (x2 - x1) * (y2 - y3)) > 1e-3)] = 1
```

```
    return shape, inside

def color(shape, r, g, b, nr, ng, nb, frames):
    """color the shape with average color of the origin image, and save the result into the frames list"""

    # color the triangle
    nr[shape > 0] = (r*shape).sum() / shape.sum()
    ng[shape > 0] = (g*shape).sum() / shape.sum()
    nb[shape > 0] = (b*shape).sum() / shape.sum()

    # create a frame of the animation and save it
    h,w = shape.shape
    new_img = np.zeros((h,w,3), dtype='uint8')
    new_img[:, :, 0] = nr
    new_img[:, :, 1] = ng
    new_img[:, :, 2] = nb
    frames.append([plt.imshow(new_img)])

def build_triangle(x1, y1, x2, y2, x3, y3, distance, height, width, r, g, b, nr, ng, nb, frames):
    """key function"""
    shape, inside = tri_shape(height, width, x1, y1, x2, y2, x3, y3)  # calculate the triangle's shape
    color(shape, r, g, b, nr, ng, nb, frames)  # color it

    # min_dis = distance.max() * 0.1
    min_dis = np.average(distance) * 0.5  # the stop condition

    # the recursion part
    if not(inside.sum() < height * width * 0.01 and np.max(distance*inside) < min_dis):  # change max to average to get different result

        # find next point to build triangle
        argmax = (distance*inside).argmax()
        xm = argmax % width
        ym = argmax // width

        build_triangle(xm, ym, x2, y2, x3, y3, distance, height, width, r, g, b, nr, ng, nb, frames)
        build_triangle(x1, y1, xm, ym, x3, y3, distance, height, width, r, g, b, nr, ng, nb, frames)
        build_triangle(x1, y1, x2, y2, xm, ym, distance, height, width, r, g, b, nr, ng, nb, frames)
        return None

def main(path, thumb=False):
    """Main function, require the image's path. If thumb is True, will create a thumbnail of the
```

```
image in the origin directory."""
    if thumb:
        thumbnail(path, 800)
        img, height, width, r, g, b, distance = pre_proceed("tb_"+path)
    else:
        img, height, width, r, g, b, distance = pre_proceed(path)

    # initialize the new rgb array.
    nr = np.zeros_like(r, int) + np.average(r)
    ng = np.zeros_like(g, int) + np.average(g)
    nb = np.zeros_like(b, int) + np.average(b)

    # find the point with the great gradient, which will be used to start the recursion
    argmax = distance.argmax()
    x0 = argmax % width
    y0 = argmax // width

    # initialize the plot part. frames is used to save each frame created during the process.
    fig = plt.figure()
    plt.axis("off")
    frames = []

    # main part
    build_triangle(x0, y0, 0, 0, width-1, 0, distance, height, width, r, g, b, nr, ng, nb, frames)
    build_triangle(x0, y0, 0, 0, 0, height-1, distance, height, width, r, g, b, nr, ng, nb, frames)
    build_triangle(x0, y0, width-1, height-1, width-1, 0, distance, height, width, r, g, b, nr,
ng, nb, frames)
    build_triangle(x0, y0, width-1, height-1, 0, height-1, distance, height, width, r, g, b, nr,
ng, nb, frames)



    new_img = np.zeros_like(img)
    new_img[:, :, 0] = nr
    new_img[:, :, 1] = ng
    new_img[:, :, 2] = nb
    #plt.imshow(new_img)  # if cancel the comment, will show the final result as a static picture.
    Image.fromarray(new_img).save("result1_" + path)  # save the static picture

    # create the animation object, with the interval between frames 50ms.
    ani = anim.ArtistAnimation(fig, frames, interval=50, blit=True)

    # for saving, FFMpeg is needed, please set the path to your own "ffmpeg.exe".
    # plt.rcParams['animation.ffmpeg_path'] =
"D:\\Software\\FormatFactory\\FFModules\\Encoder\\ffmpeg.exe"
```

```
    # FFwriter = anim.FFMpegWriter()
    # ani.save("ani.mp4", fps=30, writer=FFwriter)

    plt.show()  # show the animation.

if __name__ == "__main__":
    main("30.jpg")
```

基本原理是相似的，但该代码较先前的代码有所提高，运行速度显著加快，且低多边形切分效果更优。

主要需要解决的问题是运行速度太慢。而考虑到只取了前 N 个点，对三角形的划分也不够细腻，需要一个大幅提高运算速度的算法。此时的时间瓶颈不在于复杂度，而在于 python 本身作为解释型语言的慢速，尤其是大量运用循环时尤甚。注意到原程序在上色时，需要用嵌套的循环遍历 Image 对象中的每一个点，这一步极可能为时间瓶颈所在，但是在不引入新的库的情况下，这一瓶颈无法突破。所以考虑引入 numpy 的数组库，利用底层为 C 语言且经过了深度优化的数组来快速实现这一过程，避免了 python 的循环。同时这样图片的像素数据就储存在了 numpy 的数组里，避免了反复调用 pillow 的函数一个一个读取。

并且为了解决只取了前 N 个点（N 通常难以超过 1000，不然算不过来）的问题，改变了程序的逻辑，点数不再确定，回归到之前的终止递归条件，即色差或面积足够小就终止。

### 6.2.2 函数说明

thumbnail：由于图片太大可能导致运算缓慢，而实际上绘图并不依赖于图片大小，只依赖于相对数值，我们写了这个函数，调用 pillow 库的 thumbnail 函数，把图片先压缩再处理。另外这个函数也会顺便把压缩过的图片保存在当前目录下以便对比。

pre_proceed：这个函数是封装了读取图片和预处理计算出每个点的色差的功能。通过向量化，这一过程得以很快完成。实际上我们是将(r,g,b)看作一个欧氏空间

内的向量，计算不同点的色差就是计算两个色彩向量之间的欧几里得距离。

tri_shape：这个函数接受三个点的坐标，返回一个对应的矩阵。这个矩阵里，处在三个点坐标围成的三角形内的点的值为 1，其余为 0，这就可以被之后很方便的用来考察一个点是否在三角形内，以及为这个三角形上色。计算的方法实际上与之前相似，都是矢量叉乘，但是为了避免用 python 的循环，使用了 numpy 的布尔索引功能。

color：上色。实际上有了前面的 shape 矩阵这一步就变得很容易，shape 矩阵可以直接用来作为 numpy 的索引，这样就可以筛选出在三角形内部的点。另外，在这个函数中我们也实际上绘制出了当前上色的结果，并保存在了 frames 数组中，这为之后绘制动画提供了方便。

build_triangle：这是关键函数，也是递归所用的函数，基本思想如同之前所说，首先给当前的三角形上色（为什么不是满足结束条件再给小三角形上色，这是为了动画展现的时候比较好看），然后判断是否满足结束条件，如果是就结束，否则找出当前三角形内色差最大的点，并继续以这个点和三角形的三个顶点为新的三个三角形的顶点，递归调用本函数。

main：为了模块化，把程序主题以及一些初始化，预处理也都封装成了一个函数，这样便于以后在其他地方调用。接受两个参数，图片的路径和是否要先生成缩略图（默认为否），然后做了一大堆初始化的工作（比如创建新图片的 rgb 三个通道的数组），并挑出整张图中色差最大的点，这个点和图片的四个顶点把图片分成四个三角形，对这四个三角形调用 build_triangle。另外最后为了生成动画，还调用了 matplotlib 库的动画函数，但是由于递归的要求，只能把每帧保存下来来做动画，这样的缺点是比较占据内存。

### 6.2.3 程序限制

该程序设计思路较为缜密，暂未找到限制条件

## 6.3 实验结果

### 6.3.1 实验数据

实验环境说明：

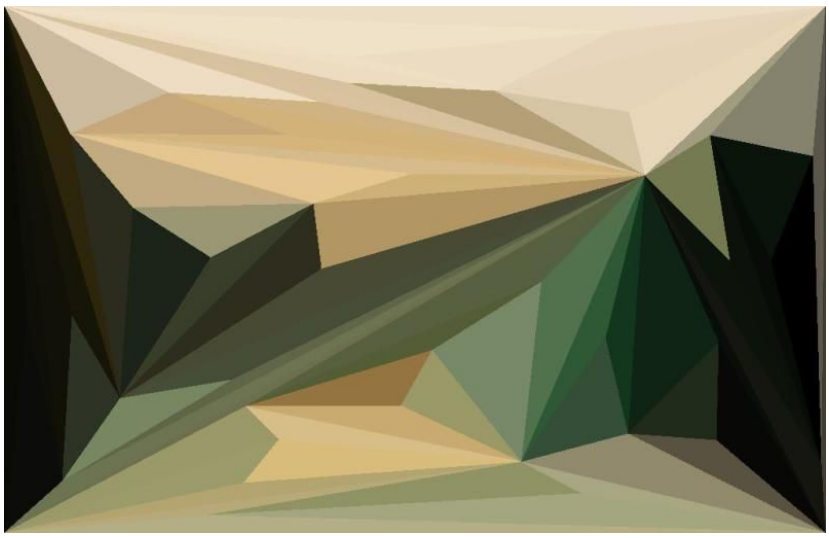
- 硬件配置：CPU core i5,内存 8g
- 操作系统：windows 10
- Python 版本：Python3.5

### 6.3.2 作品描述

该作品采用低多边形的风格，将原图用色差较小、面积较小的三角形来拟合，实现了别样的艺术效果，用抽象来表示具体，用虚拟来仿写真实，在抽象中营造着别致的美丽。对于任意的一张图，该代码都可以实现上述转换。

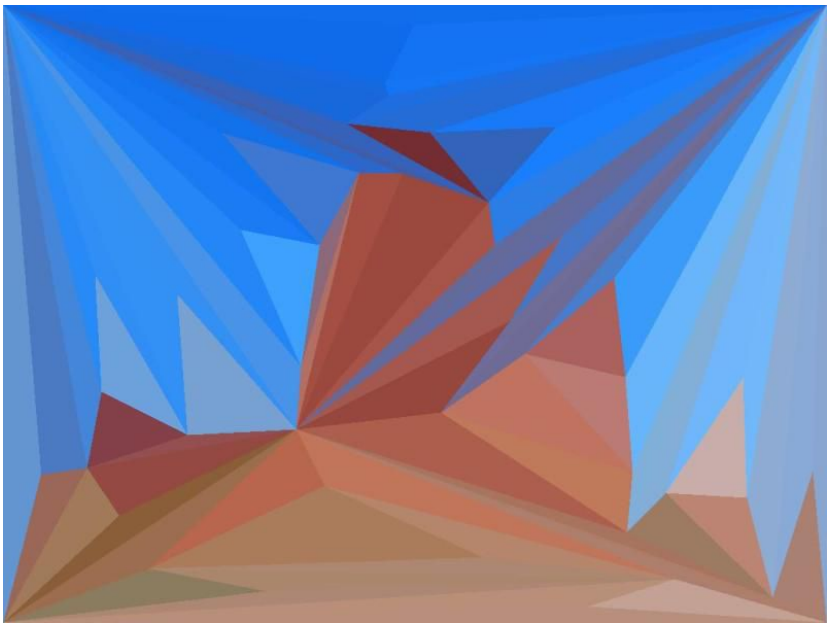
成果举例如下：

（原图一）



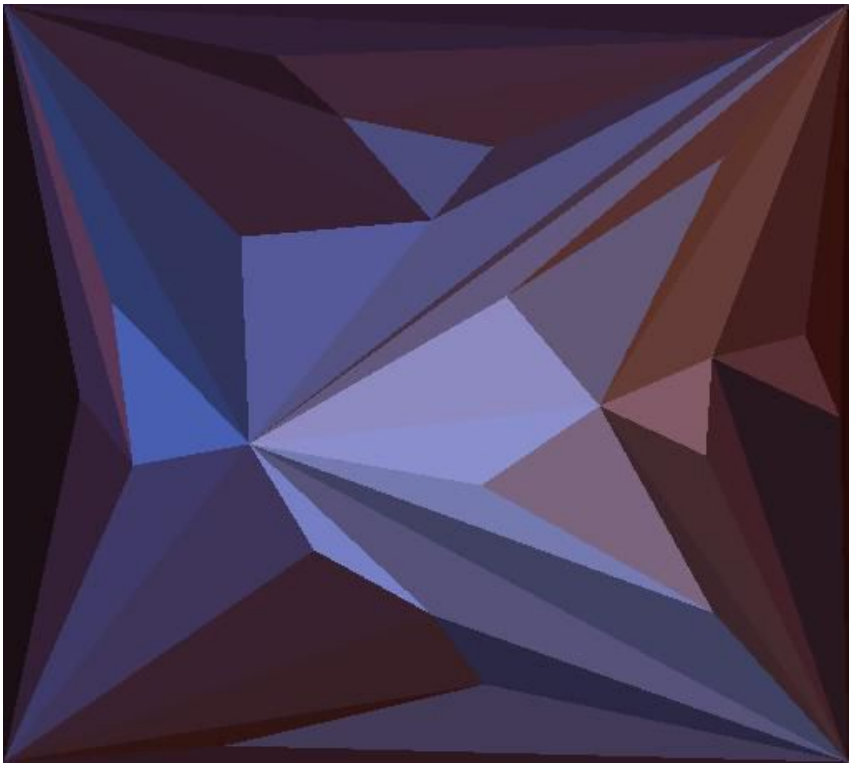
（新图一）


（原图二）



（新图二）

当然，本作品也可以实现极佳的艺术效果，做出精美绝伦的风格图片，如下图展示：



### 6.3.3 实现技巧

- 图像用矢量表示，用矩阵来储存信息
- 判断点是否在三角形内部时，用了三角形叉乘的算法
- 找极值点与处理像素锐化程度时，采用矩阵算法，提高了时间效率

## 6.4 实习过程总结

### 6.4.1 分工与合作

本小组成员：陈一潇，江欣余*，刘旭林，卢亚敏，李姿蓉，张书菀

合作交流方式

1. 主要在微信群中进行沟通讨论，上传程序进行协同操作。
2. 小组会议集中讨论。


热闹的微信群讨论

分工：

1.开发算法和编程任务：陈一潇和李姿蓉同学主要负责，其他同学辅助程序试运行及调参工作。

2.小组报告：张书菀同学负责与编程同学沟通，编写有关算法部分；其余由江欣余同学负责。

3.展示材料制作：刘旭林和卢亚敏同学，江欣余和张书菀同学提供部分材料。

第一次小组会议合影

TYPHON 小组

第一次小组会议记录

记录人：江欣余

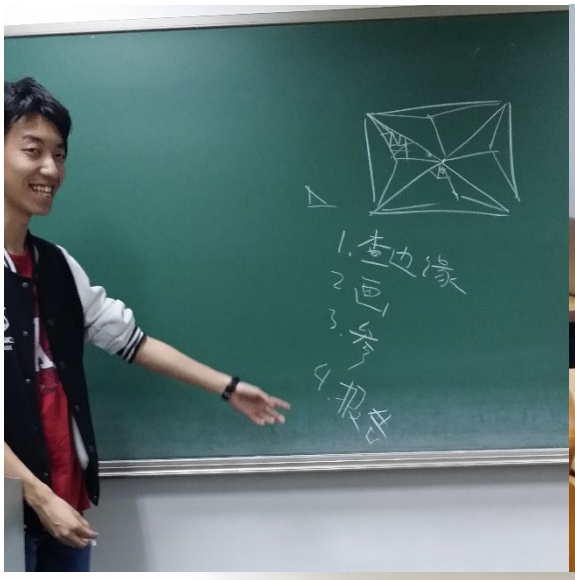
地点：四教 404

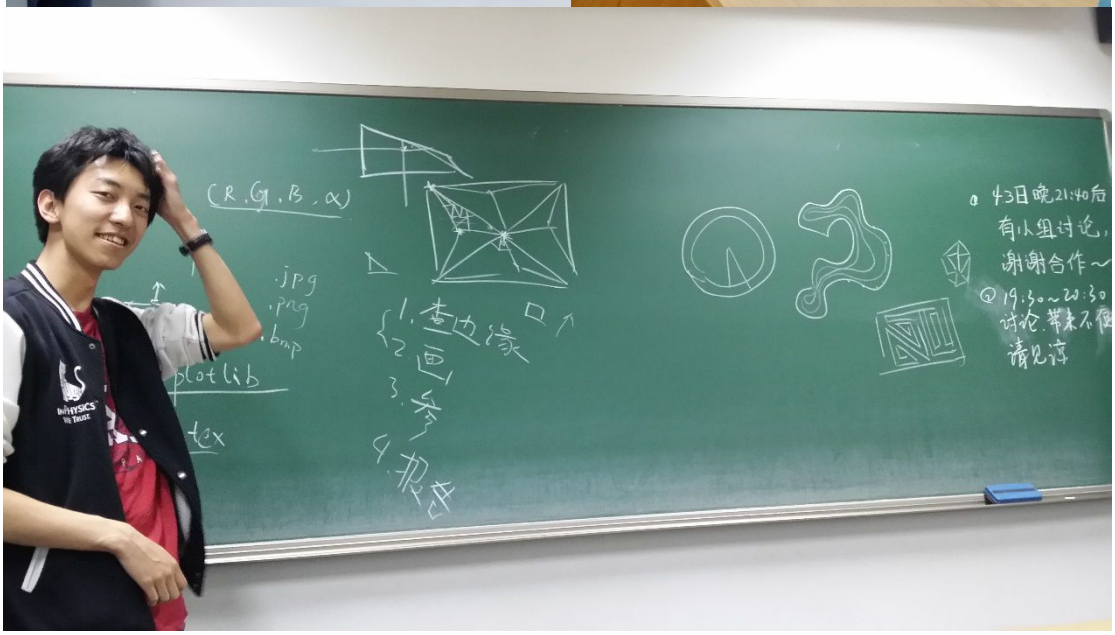
参与人员：全体组员

时间：2016 年 4 月 3 日 19:30-20:30

会议内容

- 进行头脑风暴，探讨作品构思和创意；
- 决定作品内容，建设程序基本框架，讨论需要应用的数据结构与算法；
- 决定小组分工；




- 确定作业进度安排。

小组讨论剪影

TYPHON 小组

第二次小组会议记录

记录人：江欣余

地点：四教 406

参与人员：全体组员

时间：2016 年 4 月 15 日 21:00-22:00

会议内容

1. 负责编程同学和负责完成报告同学介绍工作完成情况；
2. 就一些问题进行沟通，交换意见；
3. 讨论第二次展示内容和负责人员；
4. 探讨继续完善的方向。

第二次小组会议合影

## 6.4.2 经验与教训

### 6.4.2.1 组长感想：

我们组的同学参与热情都很高，相处也很融洽，方便了我作为组长的工作。但有一些小细节我没有做好。一个小问题是我们的微信群的群主不是我，我也一直没有要求更换过来，每次要@所有人就要一个一个点过来，造成了麻烦，当然这要怪自己的拖延症。还有就是分工不是很明确，造成了组员的一些困扰。

好的部分比如经常在群中发布消息或是协调组员解决问题，提高了效率，带动了组员的参与。我们也没有刻意限定截止时间，而是根据问题的解决情况和课程的安排不断调整，辅以组员的积极参与，较好的平衡了效率与完成质量的关系。

#### 6.4.2.2 组员感想：

这次完成数算期中大作业，我最大的感触就是团队的重要性。我们组一个有六个人，每个人都有自己的课程，自己的时间安排。所以，大家每次组会都能按时到真的是很不容易。另外，组长根据我们每个人的能力进行了恰当的分工，也使得效率得到了很大的提高。大家各司其职，尤其是主要负责代码部分的同学，一次次地调参真的是很辛苦。不过终于我们还是做到了。团队，合作，我收获很多。

### 6.4.3 建议与设想

后续工作我们一方面可以完善寻找边界、颜色重插值以及分割形状的方面的问题，并且使我们的作品更接近一个封装好的程序。另一方面，我们可以完成之前的迷宫创作的构想，受课上 turtle 走迷宫的启发，我们可以把本来较为复杂的想法更抽象化，降低实现难度。对学弟学妹的寄语：这是一个比较有趣的大作业，可以自由的“开脑洞”，也可以学习一些编程技巧。

## 6.5 致谢

本次大作业在陈老师的指导下，各位助教的讲解下，本组组员共同参与，合作完成。感谢陈老师为我们创造了本次大作业的机会，使得我们小组的成员多次聚在一起，在热烈的讨论中各抒己见，取长补短，迸发出许多颇有价值的思想火花，最终得以完成低多边形的程序设计。也要感谢各位助教在整个大作业期间的耐心指导，从选题到代码到报告，均给予详尽的说明与建议，这极大的推进了我

们的作业完成进程，让我们明确了方向。最后，衷心感谢本组所有成员的倾心付出。本次大作业中，在组长江欣余的带领下，各位组员均身担重责，对于最终成果的贡献均功不可没。其中，陈一潇、李姿蓉主攻代码设计，其他组员侧重于调参与细节完善，此外，卢亚敏、刘旭林负责展示的 PPT 与讲稿制作，江欣余、张书莞负责报告书写与其他组织统筹工作。

本次大作业让我们获益良多，在完成了自主设计研发的低多边形程序之外，我们还更加深刻的领会了 Python 思想，希望在今后的学习中习得更多 Python 的精华。在合作中，组员间也加深了友谊，在互补中完成了团队成果。雄关漫道真如铁，而今迈步从头越，期待今后的再次团队合作，勇攀高峰，突破自我，创造奇迹。

## 6.6 参考文献

1.http://baike.baidu.com/link?url=DGnRdSzwMZ8ydD7jGKHQU7mCeCq1BiQS3ir41ZyZN2sYe3uE1O0m_UcIvwUp0nUOpEvsQK48onIsTxsyjfyCoK

2.http://www.swarma.org/complex/models/ca/ca3.htm?jdfwkey=flqcw

3.http://baike.baidu.com/link?url=XmjiI9C9-sPzgZQqhAg1EWAVZclE_MoJM4bUaHcb7pq6yTLHfK9DE-6v70uEyM8BcmLPNn_LHmeCpJrV24KfUq

4.http://www.tuicool.com/articles/bqYfMri

# 7 K 组

# 递归视觉艺术实习作业报告

# 之 Topological Printer

（K 组：齐厚基*，王琦，王召平，李宣孛，杨帆，陈超强）

**摘要**：本作品利用了分形几何中吸引子的概念，通过对交互式输入的基本图形进行多次线性变换，即递归操作，实现了由任意形状向吸引子过渡的视觉效果。运行者将一个任意形状的基本图形输入程序，程序进行递归操作，初始输入端输入的是使用者输入的坐标。由于动力系统的特殊性，无论初始输入的图形是什么形状的，最终的递归图案都是确定的。作品采用的主要数据结构是 list。并用时间换取空间和用空间换取时间的两种方式。目前采取的方式是第二种，即采用一个列表记录上一次递归的数据，这个列表在下一次递归时直接被调用。本作品定义了 topo 类，类的初始化是初始图形的坐标列表。类中的方法有 draw（绘图），trans（变换）。由于要将每次递归的点分组绘制，所以采用了列表的列表。

**关键字 ：吸引子 分形 动力系统 交互**

## 7.1 创意过程与递归思路

### 7.1.1 作品总体介绍

本作品借鉴了分形几何中吸引子的概念，运行者将一个任意形状的基本图形输入程序（通过在屏幕上点按来实现），并决定是否进行填充。随后，程序进行递归操作，相当于一个动力系统，初始输入端输入的是使用者输入的坐标。随后，按照迭代规则（事先输入的范例）进行一次坐标变换，并返回迭代后的坐标组。迭代后的坐标组又被输入作为初始坐标组，清屏后进行下一次递归绘图。由于这一动力系统的特殊性，无论初始输入的图形是什么形状的，最终的递归图案都是确定的，好像有一种力量在吸引图形一样。

空间换取时间的两种方式。第二种方式是目前采用的方式。就是采用一个列表记录上一次递归的数据，这个列表在下一次递归时直接被调用。

本作品定义了 topo 类，类的初始化是初始图形的坐标列表。类中的方法有 draw（绘图），trans（变换）由于要将每次递归的点分组绘制，所以采用了列表的列表。

### 7.1.2 创意来源

在初次小组会议后，我们组暂定画彼岸花和蒲公英，并做出动态效果。在这个实现的过程中，组长渐渐发现，递归思路在这些图案上运用并不多。

于是，在第二次和第三次组会之间，组长翻阅了大量的资料。终于在与递归相关的书籍内发现了分形几何学一书，其中介绍的吸引子吸引了我们的注意。

自然界的大多数图形都是十分复杂而且不规则的。然而，人们发现，依据一些简单的规则进行无限重复，简单的图形也可以呈现现实世界中存在的复杂状态。这些图案都存在着一定的自相似性。分形就是这样一种不规则，但有这很强自相似性的几何集合。分形中比较常用的概念是分维。

一个正方形，将它的边长扩大 3 倍后，和原来相似，而且相当于 32=9 个小正方形。同样的，一个正方体的边长扩大 3 倍后，相当于 33=27 个小正方体。推广一下 ，就得到：ld=N。l 是放大倍数。d 就是拓扑维度。d=loglN=lnN/lnl 如果 d 不仅仅局限于整数，而是实数域的一部分，那么这个 d 便称为分形几何中的分维。分形几何与动力系统密切相关。

在这个过程中，通过大量数学分形的例子，我们看到：简单的生成规则在无限次迭代过程之后，可以生成极为复杂的分形图形。所以，我们决定采用动力学的语言，重新讨论这一主题。

最简单的动力模型具有三个要素：输入，输出和动力学的作用规律。

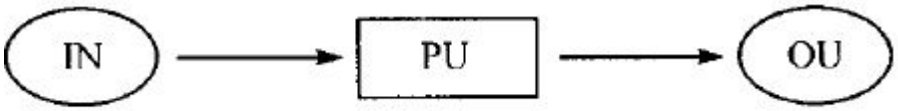



图 2.1　最简单的动力学模型

IN＝输入；OU＝输出；PU＝作用规律

我们已经接触过的动力学理论莫过于牛顿力学，对于牛顿力学来说，F 是作用物体上的力，通常作为动力学的输入;而作用规律：F=ma 代表了系统作用规律的线性关系。在经过数次讨论和学习之后，由此，我们得出了我们程序的初步设想。

### 7.1.3 递归思路

本作品利用了分形几何中吸引子的概念。

在这里我们借用多功能复印机的概念。多功能复印机的运转是由反馈通过对交互式输入的基本图形进行多次线性变换，即递归操作，实现了由任意形状向吸引子过渡的视觉效果。作品采用的主要数据结构是 list。并有用时间换取空间和用过程来控制的。上一次的复印输出，即作为下一步的复印输入，不断地进行下去。只要将一次复印的输入与该次复印的输出输出加以仔细比较，就可以知道该台复印机有几个镜头，每个镜头是放大、缩小还是旋转变换，以及不同镜头或像结果在输出时是如何排列的等等。

这里有一个十分特殊的情况，假定多功能复印机有三个镜头，每个镜头都缩小 1/2。每次复印后，输入和输出的图形都没有变化。总结以上的结果，用多功能复印机不断进行复印，不断输入图件如何，都会得到一幅一样的图形。这个最终图形就叫做这台多功能复印机的吸引子，也叫作这个迭代过程的吸引子。

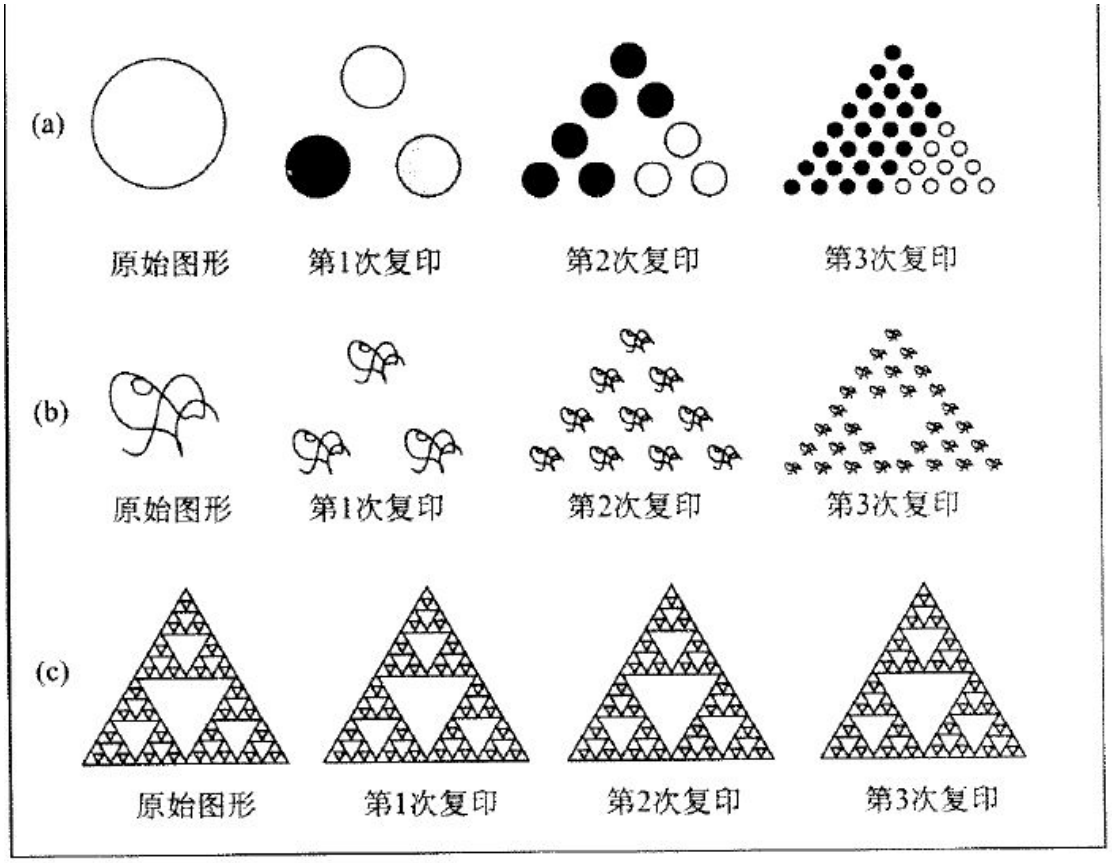



图 2.8　用 3 种不同的图形作为原始输入图件，
多功能复印机前 3 次复印的结果

为了完成这一操作，我们借用动力系统的概念，控制输入和动力系统的规律，力图达成结果。

一开始，我们的初步设想是想使用纵向递归的方式，类似于二叉树的数据结构。设置一个计数单元，当计数单元小于递归深度时，就记录当前数据位置，同时进行下一次递归。即进行下一次坐标变换并绘图。但是这种方法主要有两个缺点：第一是作图的可视效果不强，因为乌龟的每次移动都是最终一次递归的图像，无法展现图形逐渐由初始图形到吸引子的过程；第二，由于在每一个节点都需要进行重复计算和计数，算法的时间复杂度很高。所以，我们决定改变递归思路，也就是设置一个“函数缓存”用来直接存储上一次递归的信息，在下一次递归调用时直接以上一次的整体数据作为初始数据，这样在计算变换的过程，时间消耗降低了 x^n+x^n-1+x^n-2…+x （ x 为复制次数，n 为递归深度） 倍。而且可以在一次调用后清屏，展现逐步逼近的效果。这样，整体的效果就是：清屏——绘图——坐标变换——清屏——绘图……最后我们得出成果。

## 7.2 程序代码说明

```
import turtle
import time
t=turtle.Turtle()
t.speed(0)
w=turtle.Screen()
class topo:    #定义 topo 类，其 init 方法为设置初始图形坐标列表
    def __init__(self,totalPointList):
        self.totalpointlist=totalPointList
    def draw(self,fill):  #定义每层递归的作图方法。当 fill 值为 true
时进行填充
        global t,w
        for pointlist in self.totalpointlist:
            t.penup()
            t.goto(pointlist[0])
            t.pendown()
            if fill:
                t.begin_fill()
            for location in pointlist:
                t.goto(location)
            if fill:
                t.end_fill()
    def trans(self,totaltranslist):#坐标变换方法，由线性变换公式
u=ax+by+c,v=dx+ey+f 在 totaltranslist 中的每一个列表存储的数据为变
换公式中的 a,b,c,d,e,f.由于是嵌套循环，所以有几个列表就会循环几次。每循
环一次就进行一次变换，将变换的结果输入新的 totallist
最终用 totallist 替代原来的 totalpointlist
        newtotallist=[]
        for translist in totaltranslist:
            for pointlist in self.totalpointlist:
                newlist=[]
                for x,y in pointlist:

newlist.append((translist[0]*x+translist[1]*y+translist[2],t
ranslist[3]*x+translist[4]*y+translist[5]))
                newtotallist.append(newlist)
        self.totalpointlist=newtotallist
def NHtrans(self,totaltranslist):#非线性变换的尝试，但最终由于没有
十分漂亮的图形被画出，因而没有展示其构图
    newtotallist=[]
    for translist in totaltranslist:
        for pointlist in self.totalpointlist:
```

```
            newlist=[]
            for x,y in pointlist:

newlist.append((x*(translist[0]**translist[1])+y*(translist[
2]**translist[3])+translist[4],x*(translist[5]**translist[6])
+y*(translist[7]**translist[8])+translist[9]))
            newtotallist.append(newlist)
    self.totalpointlist=newtotallist
```

```
def read():    #由于交互模块是直接调用的改进版 demo，所以为了传参，设置了一个 cache 文件，在 paint 模块中获得的点的坐标会被传入 cache 文件。Read 函数按行读取 cache 文件中的内容并进行转换，转换成 topo 类接受的初始化形式
注：在另一种用时间换取空间的方法中，cache 文件是直接存储所有点的坐标的。
    f=open("cathe","r")
    p=[]
    for line in f:
        s=line.split(' ')
        for i in range(len(s)):
            s[i]=int(s[i])
        p.append((s[0],s[1]))
    f.close()
    return p
for i in range(8):#在循环中实现递归，当最后一次递归时，不进行清屏。range 为递归次数
    a.draw(True)
    time.sleep(1)
    if i<7:
        t.reset()
    a.trans(haha)
```

交互模块：

```
from turtle import #*调用 turtle 所有内容
def newgoto(x,y):#这个函数将接受的 x,y 参数写入 cache 文件
    global f
    f.write("%d %d\n"%(x,y))
def main():#主函数，将 turtle 作为中心的参考点，接受鼠标在屏幕上的 x，y 坐标。并传给 newgoto 函数
    shape("circle")
    resizemode("user")
    shapesize(.5)
    width(3)
    up()
    onscreenclick(newgoto,1)
```

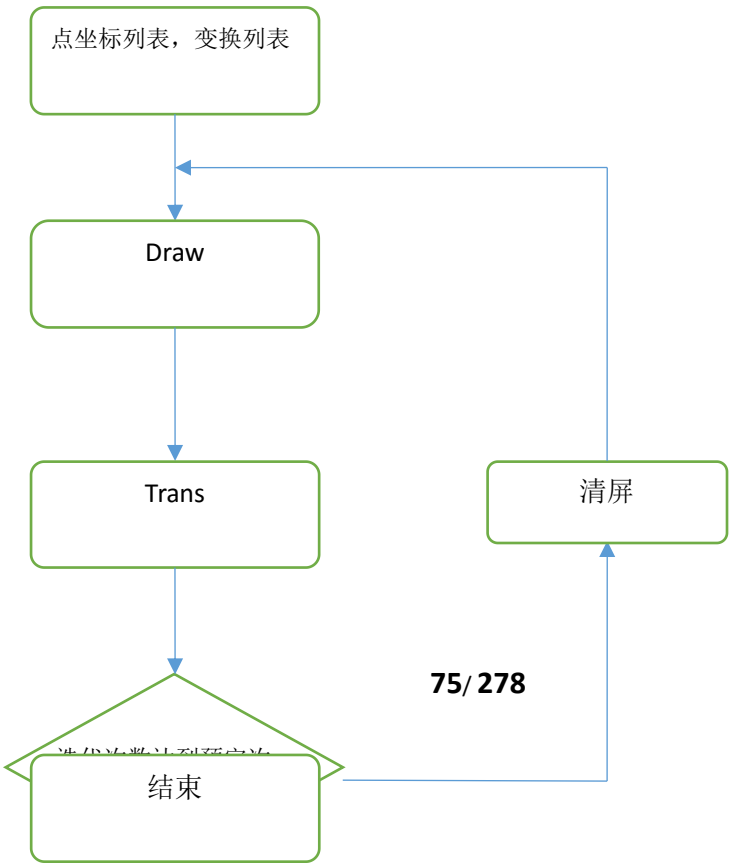
```
    return "EVENTLOOP"

if __name__ == "__main__":#使用 mainloop 以实现对不定数量的点的绘制，但由于其无法在结束后再执行其他内容（这一技术尚未实现），所以分成了两个模块
    f=open("cache","w+")
    main()
    mainloop()
```

数据结构说明与函数说明见程序注释。

### 7.2.1 程序限制

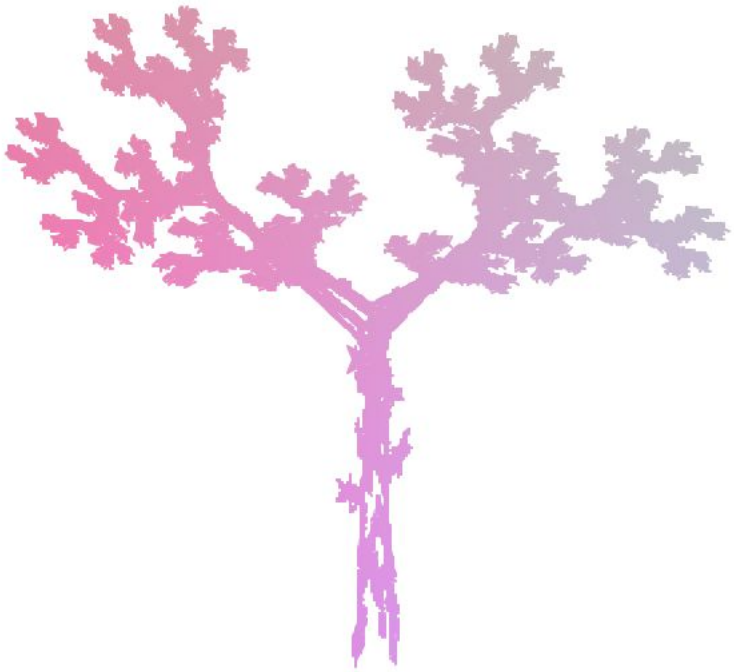
在用空间换时间的情况下，由于内存的限制，在用空间换时间的算法下，该程序无法实现无限次递归，只能实现大约 8 到 10 次递归。在用时间换空间的情况下，在第九次及以后的递归用时较长。程序出错的可能情况是：一、内存不足；二、系统瘫痪，电脑中病毒等系统错误。

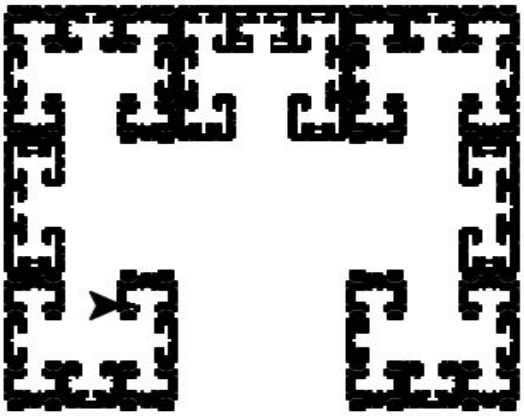
### 7.2.2 算法流程图

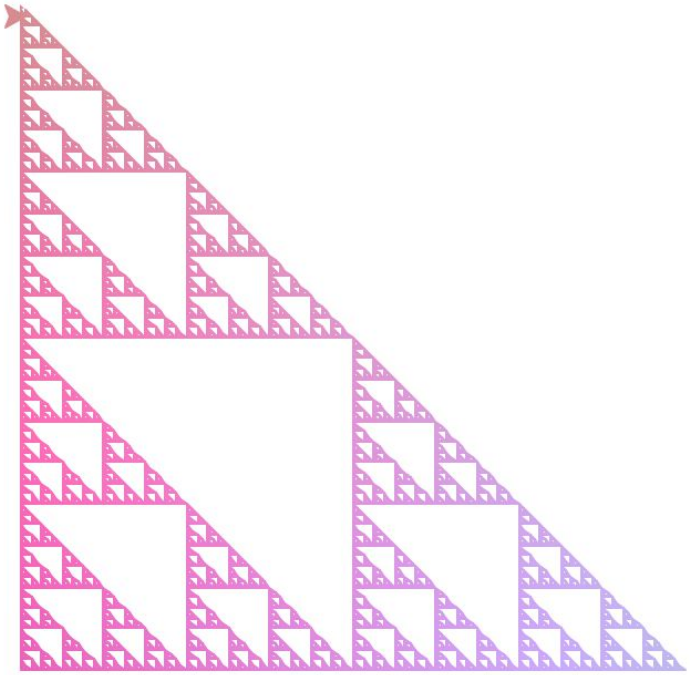


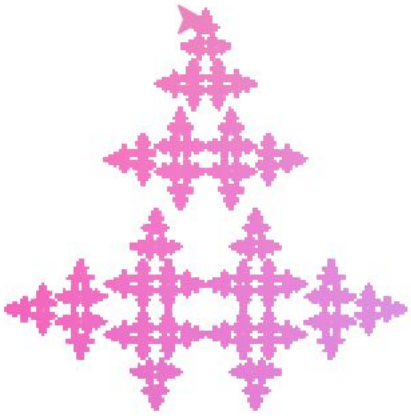
### 7.2.3 实验结果

此处附上第九次迭代的图像









### 7.2.4 实验数据

**实验环境说明：**

硬件配置： MacBook Pro MF840CH/A

i5-5257U/8G

操作系统： Mac OS X El Capitan 10.11.4

Python 版本： 3.5.1

### 7.2.5 作品描述

本作品采用了交互方法输入基本图形，随后根据预先设定的变换方式进行变换，可以看到图形随迭代深度的增加整体形状越来越趋向于吸引子的形状。而且还增加了色彩变换。
此外，还附上小组同学的初期作品，虽然递归元素不是很多，但视觉效果很棒。

### 7.2.6 实现技巧

一、采用列表的列表绘制点组；二、设置线性变换参数，实现动力系统的坐标变换；三、采用读写文件的形式调用交互模块。

## 7.3 实习过程总结

### 7.3.1 分工与合作

本小组一共六名成员，每位成员均在这次大作业中发挥了重要的作用。其中，组长齐厚基负责统筹大家的意见，确定最后的主题，协调刷代码的时间，做第一次以及最后一次的报告。同时，组长和王琦，王召平，李宣孛，陈超强负责写代码和调试程序。PPT，报告以及视觉效果的负责者是杨帆。

由于各种原因，小组的讨论方式多在聚餐时进行。除此之外，广泛的运用微信，也是我们组的命令能尽快执行的重要原因之一。组长多次组织程序员们进行头脑风暴开发，并进行模板化开发，及把程序分成三个模块，同时进行开发，以提高效率，保证完成速度。同时，同学们广泛利用课下课后的时间，积极展开合作，针对一个问题反复尝试。终于达成了最后的结果。

（第一次组会忘了拍大家的照片，放一张吃的东西意思一下。）

（第二次组会，图中其实有五个人。）

（第三次组会）

### 7.3.2 经验与教训

本小组属于开工时间比较早的小组之一，在小组成立的第二天就开展了第一次组会。第一次组会我们决定做一个花和蒲公英动态的效果，但是在一个星期之

后，我们做出了彼岸花和蒲公英的动态，可惜的是，我们发现这其中运用的递归非常少。所以组长决定大刀阔斧地进行第二次尝试，终于在第二次和第三次组会之间，决定做吸引子树。在研究了大量资料之后，我们组完成了第二次尝试，最终不仅实现了吸引子树，还添加了交互的部分。

希望下次大作业的时候，我们可以更明确我们的目标。

### 7.3.3 建议与设想

由于交互模块的引入，总共需要运行两个程序，因而可以尝试采用 tkinter 模块将其整合。

## 7.4 致谢

首先，感谢我们六位组员的协作：齐厚基，王琦，王召平，李宣孛，杨帆，陈超强。其中由衷感谢组长的操劳与努力。

另外，感谢助教和老师的指导。尤其是陈春含学姐的指导。

## 7.5 参考文献

https://docs.python.org/3.5/library/turtle.html
分形几何学（第二版）陈颙

# 8 M 组

# 数算 turtle 及递归大作业报告

小组成员：刘立超，赵宸兴宇，李京寰，童刘奕

## 8.1 选题介绍

超级玛丽中的马里奥是日本任天堂的招牌人物，超级玛丽这款游戏可以说是红白机上的经典，决定以此为选题，也是为了回忆一下童年里那些永不褪色的游戏角色。

对于本次递归作业，小组在经过商议后，决定做一个超级玛丽的可操控动态图像（其实这一提议也是在后面编程的过程中陆续形成的）。尽管本次作业的主题是递归（这也就意味着某些可以通过递归较快得出的图案比较适合，如树、云彩、花朵等自身具有相似性的图案），但我们仍然希望可以做出一些与众不同的东西。

通过头脑风暴，小组中的一名成员提出了超级玛丽的想法。本着可以循环就可以递归的思想，我们发现由一个个像素块组成的马里奥以及游戏中的背景完全可以用递归来画，同时我们也可以增加一些新的创意使之更加符合本次作业的主题。因此最终大家一致决定，本次大作业就完成超级玛丽的一个场景，并尽量在作图的基础上使这一场景中的超级玛丽能走能跳，如果能撞砖头当然更好不过。

## 8.2 设计方案

本次设计在整体上还是考虑递归算法，但是由于超级玛丽这一角色在整体作图上很难使用递归方法，所以我们仍旧选择使用像素块画出了人物，这也引出了准备过程中最单调的一项工作——数像素块——笨但是简单粗暴。背景图由于某些部分相似性较高，所以我们使用递归合成了地下的砖头（超级玛丽脚下的砖头具有连续的相似性）、天空的蓝色背景和超级玛丽要撞的砖头。为了符合此次递归主体，我们还将超级玛丽中原本应该撞出的蘑菇改为树，超级玛丽在撞到砖头之后，砖头上会长出我们使用简单的递归算法画出的树，当然，我们并没有把这棵树当做作品的亮点。

在成功画出了马里奥之后，我们希望它可以左右移动。因此，在作图过程中我们使用像

素点同时画了超级玛丽左右移动时的对称的图像，并使用键盘监视器控制其左右移动。既然能够移动，那也要能跳跃。由于跳跃时的人物形象与行走时的也不相同，我们又使用像素点画出了超级玛丽跳跃时双脚叉开的图像。

不同图像间的转换，或者说不同运动状态的转变更为困难一些。运动时的动态效果是我们在后期的主要思考问题。最后，我们使用多张图像的快速转变来表达运动状态。如向左移动时，我们就使用四张图像之间的连续变化来表现其动态的运动过程，最终效果还是达到了我们的预期。

## 8.3 算法简介

算法主要分为四个部分，第一部分是超级玛丽的图像。我们使用像素点画出了超级玛丽的十种状态下的图像，分别是：向左移动，三种；向右移动，三种；站定，朝向左，一种；站定，朝向右，一种；跳起，朝向左，一种；跳起，朝向右，一种。这一部分算法没有使用递归，而是使用像素点直接合成图像，并在之后的程序中将其作为一个 turtle 对象进行处理。

第二部分是绘制背景。背景中主要包括蓝色背景、云、空中待撞的砖头、绿色的下水管、地下的砖头和左侧底角的山。其中，地上的砖头使用了递归算法，而其他背景图案则直接调用 turtle 中的函数画出。

第三部分是定义对超级玛丽的操控。主要包括控制其向左移动一步，向右移动一步，垂直跳起、向右上方跳起、向左上方跳起。此外，当超级玛丽跳起撞到空中砖头的时候，砖头上会生长出一棵使用递归方法画出的树。

第四部分是主函数，调用前面三个部分的函数生成超级玛丽，并且使用键盘监视器，读取键盘上的操作，使图像听从用户的操作，根据用户的按键进行运动。

## 8.4 算法实现

1. 超级玛丽的图像绘制（我们一共画出了十个不同的马里奥，每一个都对应着一个 16 乘 16 的列表来储存色块，这是其中一个）

```
colorstandright=[['white','white','white','red','red','red','red','red','white','wh
ite','white','white','white','white','white','white'],\

['white','white','red','red','red','red','red','red','red','red','red','white','whi
te','white','white','white'],\
```

```
['white','white','green','green','green','yellow','yellow','green','yellow','white',
'white','white','white','white','white'],\

['white','green','yellow','green','yellow','yellow','yellow','green','yellow','yell
ow','yellow','white','white','white','white','white'],\

['white','green','yellow','green','green','yellow','yellow','yellow','green','yello
w','yellow','yellow','white','white','white','white'],\

['white','green','green','yellow','yellow','yellow','yellow','green','green','green
','green','white','white','white','white','white'],\

['white','white','white','yellow','yellow','yellow','yellow','yellow','yellow','yel
low','white','white','white','white','white','white'],\

['white','white','green','green','red','green','green','green','white','white','whi
te','white','white','white','white','white'],\

['white','green','green','green','red','green','green','red','green','green','green
','white','white','white','white','white'],\

['green','green','green','green','red','red','red','red','green','green','green','g
reen','white','white','white','white'],\

['yellow','yellow','green','red','yellow','red','red','yellow','red','green','yello
w','yellow','white','white','white','white'],\

['yellow','yellow','yellow','red','red','red','red','red','red','yellow','yellow','
yellow','white','white','white','white'],\

['yellow','yellow','red','red','red','red','red','red','red','red','yellow','yellow
','white','white','white','white'],\

['white','white','red','red','red','white','white','red','red','red','white','white
','white','white','white','white'],\

['white','green','green','green','white','white','white','white','green','green','g
reen','white','white','white','white','white'],\

['green','green','green','green','white','white','white','white','green','green','g
reen','green','white','white','white','white']]
```

2. 背景图（背景主要包括山、云、地面和空中的砖）

为了便于调整，在构造画各部分的函数时，特意把函数的参数变成了坐标位置以及尺寸，实现可以在画布上的任意位置画出所需大小的图形，使得后续对布局的调整变得方便

```
Draw(-760,-240,20,2)  画地面的砖，起点（-760，-240），20 列，两行

draw_bottletop(220,-240)

draw_bottlebottom(200,-140) 和上一个合起来画出水管，坐标（200，-140）

drawzhuantou(-100,120)

drawzhuantou(-20,120)这两个是画空中的砖头，坐标（-100,120）和（-20,120）

drawyun(-200,250,30)

drawyun(500,200,30)这两个是画云，坐标（-200,250）和（500,200），30 控制尺寸

drawshan(-300,-240,250)画山，坐标（-300，-240），250 控制尺寸
```

2. 马里奥变 turtle

主体是一个函数，这个函数又调用了另一个函数

```
def setturtle(astr,alist):    astr 是你给这个形状的名字，alist 是对应的列表
    s=Shape("compound")
    register_shape(astr,s)
    for i in range(16):
        addshape(s,alist,i,0)   调用 addshape 函数
def addshape(s,alist,i,j):          把一行的色块添加进 turtle 中
    if alist[i][j]!='white':
            p=((10*i,10*j),(10*i,10*j+10),(10*i+10,10*j+10),(10*i+10,10*j))
            if alist[i][j]=='red':
                s.addcomponent(p,(219,43,0))
            if alist[i][j]=='green':
                s.addcomponent(p,(139,115,0))
            if alist[i][j]=='yellow':
                s.addcomponent(p,(255,155,59))
    if j <15:
        addshape(s,alist,i,j+1)          递归
```

3. 马里奥的操控（只需要左移和撞砖头）

运用 onkey（）以及 listen（），实现对马里奥的控制，定义马里奥不同动作（走，跳，跳上水管）的函数

```
onkey(UP,'w')

    onkey(RIGHT,'d')
```

```
onkey(LEFT,'a')
onkey(rightjump,'e')
onkey(liftjump,'q')
listen()
```

其中 UP，RIGHT,LEFT,rightjump，liftjump 都是定义的函数

4. 主函数

调用创立 turtle 形状的函数以及控制函数

## 8.5 用户体验

在美观方面，本次成果基本与原游戏的图像类似，但是由于像素点较大，会在一定程度上影响美观程度，影响用户体验。

在与用户交互方面，由于 turtle 整体运行效果的原因和程序优化程度有限，在控制上还存在一定的时延，并不能准确的还原游戏里的所有动作，毕竟不能期望只用一个 turtle 就能编出一款游戏，在交互的及时性上还有待提高。

但与此同时，超级玛丽左右移动和跳跃时的动态效果与原游戏类似，用户对图像的移动效果能够得到一定满足；移动控制较符合人们的日常认知，通过 W、A、D 三个键来控制超级玛丽的左右移动和向上跳跃，同时 E、R 键还用于左右上方向的跳跃，便于操作，但是只能在特定位置使用。

## 8.6 不足与期望

像素点过大，图像不清晰。这在之前已经提到过，导致图像不够美观，在未来的改进中，我们将使用更小的像素点进行优化，或者直接将整个游戏界面缩放，使图像看上去更美观。

用户按下按键之后，图像运动的反应速度较慢，用户控制方面的及时反馈性还有待增强。在这方面，主要是因为整个程序的复杂度较高，在很多需要进行优化的地方并没有进行优化，如在图像运动时，不断生成图像使其产生动态效果的方法较为复杂，我们会在后续工作中试图找出更为简单的方法。

向左前方跳起和向右前方跳起存在问题。目前，只支持在水管的两边可以向左前方或右前方跳起，即直接跳到水管上，但是还不支持任何时候向左前方、右前方跳起。同时，不支持游戏中原有的，按下跳起键后可以左右移动的操作。但由于过于复杂的操作与本次的主题

无关，所以我们目前尚未考虑。在后续进展中（如果还有必要的话），我们希望能够对进一步添加更加复杂的操作方式。

## 8.7 小组分工

我们并没有精确的规定每个人的分工（因为没有大腿，只能抱团攻坚），而是选择把作业分为几步来做，每一步都共同努力。
我们将整个作品的工作分成了以下几个阶段：
1.第零期：主题的选择。
这一阶段，我们提出了两种方向：风景和游戏，最终选择了构图更加简单的游戏，同时并没有把过多的注意力放在背景的漂亮与否上。
2.第一期：马里奥的绘制
这一阶段任务是准备各种需要的色块的列表，但是从后面的需求来看，这一阶段的准备并不充分，造成我们在后期码代码的过程中又搜集了一次列表
3.第二期：怎样让马里奥动起来
这一阶段任务是想出一种让马里奥动起来的方法，首先我们想到的是在不同的位置画图，一帧一帧的绘制，然而由于画一个马里奥的步骤太多，时间过长，所以效果很渣，这一问题的解决使得后面的一切步骤水到渠成，对 turtle 功能的发掘就是在解决问题的过程中开展的。后来，我们在 IDLE 自带的例子中发现了 tracer（）这个函数，果断付诸实践，然而仍旧不完美，就在我们纠结要不要从头开始的时候，我们发现了 turtle 中最灵活的东西——海龟，我们想怎样才能把一个图变成海龟呢？
4.第三期：怎么样把马里奥变成一只海龟
这一阶段我们开始使用两种把一个多边形变成海龟的方法，并且在最后的版本中选择了代码最短的那种，到这一阶段结束时，我们已经可以把想要的任何一只马里奥变成灵活的海龟，整个代码中技术含量较高的一块成型
5.第四期：交互的实现
把马里奥变成海龟之后，我们由最初的想要做一幅动图变成了想要像游戏里那样控制它，交互的要求诞生了，然后又是 IDLE 中的一个函数，可以说是为我们提供了现成的素材，模仿之后熟练运用，我们做出了一个听党指挥的马里奥
6.第五期：对第一期的准备进行了补充
7.第六期：背景的描绘
这里的要求在于要控制背景物的坐标位置，以便于能够对马里奥的动作、位置进行精确的指挥，
8.第七期：把背景和马里奥的函数组装起来
这个过程想起来很简单，但是由于模块调用的不同，造成了海龟的混乱，所以并不是一气呵成的，最后还是选择了暴力结合——强行把两部分代码堆到一块，经过一些调整之后，成功出炉
9.第八期：bug 的寻找与修复
这一过程我们已经基本完工，就是进行各种操作，看看程序哪里有漏洞，修改参数，然后把能够改成递归的地方修改成递归，还有就是写报告。
这一阶段我们开了最后一次网络组会：（报告哥因断网缺席）


因为小组成员基本上在一个宿舍，所以这算得上是一次很正式的组会，平时每天都相当于在开会

# 9 N 组

# 数据结构与算法课程

# 递归视觉艺术实习作业报告

（卢国军*，韦庆朗，尹泽藩，贾彩霖，徐贝贝，王宁）

摘要：

此次作品最初的创意来源来自于去年分形树中一幅仅有的抽象派画作。受到此幅作品的影响，我们最终选定了中国的一幅山水画作为描绘对象。无论从其制作的难度还是其观赏性来说，都是比较合适的一幅作品。另外此幅作品中用到递归的有四处，一是小楫轻舟，二是夜空明月，三是月下竹林，四是空中浮云。主要实现原理还是通过景物中的自相似来调用函数完成绘制，如竹林中存在竹节的自相似、明月中存在圆环的自相似、轻舟舟体存在无数往返折线的自相似。作品最终的整体效果会给大家呈现出一幅比较有意境的水墨山水画，此幅画涉及到的景象有明月、竹林、轻舟、山水、浮云，其中大部分用到了颜色的渐变。另外此幅作品还会化静为动，有竹叶飘落，轻舟摇摆等动态效果。

关键字：

水墨山水画，竹，月，船只，山水背景，递归

## 9.1 创意过程与递归思路

### 9.1.1 作品总体介绍：

我们希望利用山水画中重复性的，规则性的意象，通过递归再现山水画的意境。主要再现的意象有竹，月，船只以及山水背景。

如通过竹节的相似性，依次减少竹节的长度，最终实现整个竹子的再现。再如水墨画中墨色的深浅，我们把意象分成规律性重复的部分：将船只分为一条曲线的重复摆动，将月亮分为若干环形区域的叠加，将每一部分的颜色进行递变，

最终实现整体颜色的连续变化。另外作品还会引入随机数来实现一个竹叶飘落的动态效果。

### 9.1.2 创意来源：

我们小组起始的思路是从可以用递归实现的意象、图片出发，从难度较低的图像做起，时间允许的条件下，向高阶作品演化。递归可以实现的意象常常具有规律性，重复性和对称性，依据此种特点，我们提出了如下几种较为常见的意象及图片：

1、莫奈的作品《日出印象》,《野罂粟》:

这两幅图片都具有大片重复背景，而过于关注细节的部分较少，可以留给我们较大的发挥空间，利用递归的思想完成图片的再现。而且其中的构图基本元素主要是花朵、草地、树木与天空，并无特别棘手的图像，可操作性较强。

2、梵高的作品《向日葵》:

梵高的画作中，以向日葵为对象的多达 11 幅，花朵之间的相似性，花瓣之间的重复性都为利用递归思想创造了有利的条件。花朵之间开放状态，颜色差异也避免作品沦于单调。

3、 水墨山水画：

从水墨画中最常见的竹子考虑，竹节的重复排列，竹叶的相似性都可以加以利用，而且对于竹子之间的不同姿态，竹叶的不同指向、形状，我们都希望可以通过修改函数的参数加以实现。再从竹子出发，参考上述作品，再加以课堂上所学的树状结构，我们推广到山水画中的梅花、菊花等主要由枝节和花朵构成的植物，石林、水流等由相似部分构成的景物。

4、 花开过程：

动态图像的实现对于我们目前的水平来说可能略有困难，所以我们希望截取一个过程中最具有代表性的几个图像，借助海龟的绘画过程完成一个“伪动态图像”。经过前面的铺垫，我们还是选取花朵这个代表物，因其具有花苞、花朵和残花三个最具有代表性的阶段，而且以梵高的《向日葵》为例，其作品中正好同时具有含苞待放，盛开和残败的向日葵花朵，可以为我们提供构图的模板。

5、 星空：

星星基本相似，但其排列组合则具有不规则性，做出基本的形状，即可构建我们自己的星空。

经过讨论，我们小组成员认为，从时间和工作量的角度考虑，作品最好具有现成的模板，避免在艺术设计上浪费过多的时间，因此我们将星空的选项先排除在考虑范围之外，时间足够充裕的或者有课外兴趣的同学可以在基础作品上去挑战此类作品。

而“伪动态图像”则是以静态图像为基础，进一步形成的，所以我们最终将目光锁定在静态作品《野罂粟》和《向日葵》上。

《向日葵》中细节部分较多，我们刻意去模仿很可能做出来的图片很生硬，缺乏感染力，甚至可能会画虎不成反类犬。而相比之下，《野罂粟》的视野范围较为宽广，不关注细枝末节，在我们看来，属于门槛稍微低一些的作品。而且其艺术性也很强，符合我们的选题原则。所以我们小组首先敲定了莫奈的作品《野罂粟》。

然而结果并不令人满意，《野罂粟》中背景颜色的变换过于复杂，虽然勉强可以用递归实现，但最终呈现出来的结果与原作相去甚远，我们决定更换题目，选择用少量颜色即可创造意境的山水画。

### 9.1.3 递归思路

1、轻舟船体描绘：为达到水墨画的艺术效果，将船身看做是无数细线描绘出来的。首先让 turtle 倾斜一定的角度走一条直线，随后折回但有一定的偏移，此为一次函数调用，随后减小走线长度，重复调用此函数，直到走线长度达到预定值，完成整个船体的绘制。

2、月亮颜色渐变：将月亮分为若干个环形区域，从月亮的最外围开始，从圆周 C 上的某一点出发，绘制一个比圆周 C 稍微小一些的圆周 C1，在 C 与 C1 所围成的不规则环形区域中填充颜色，从而将月亮分为一个个的环形，环形的半径大小的依次减少。

3、倒影的描绘：描绘倒影包含与船体同样的递归思路，通过一系列折线来营造水墨画的意境。

4、竹节重现：将竹子分为若干相似的竹节，对于给定的基础竹节的长宽参

数，设定长宽梯度差，逐渐改变竹节的大小，从而实现竹子的大小。

5、浮云的实现：用许多小角度圆弧去勾勒形状不规则的浮云外廓，进而缩小圆弧半径，从浮云外廓向内部逐渐递进，同时填充渐变色。

## 9.2 程序代码说明

### 9.2.1 函数说明

一、对于一条完整的小船，我们定义了六个函数，其中每个函数均对应船上一个部位。其中：

drawline(len,angle)用于实现船体的绘制，我们在这一函数中运用了递归的算法，绘制船体的方式是通过多条折线的来回走动去填充船体，以避免因单一填充导致船体色彩单调继而影响整幅画的水墨画风格。算法可抽象为以素描的方式去实现船体的绘制。注意到此函数有两个参数，len 表示折线的长度，angle 表示折线偏转的角度（此角度会尽量接近 180 度，从而实现整个船体的填充）。另外，由于递归中函数的调用，为避免陷入死循环，我们规定一个线的长度，且每次调用函数线都会减少一定的值，当线的长度值小于规定值时，结束调用。

Shadow(len,angle)函数用于实现船体在水中的倒影，其基本思想与 drawline 函数相一致，也采用了递归去描出船的倒影，不过其参数会做一些修改，以达到较好的效果。

Quare(len,t,w)用于实现船体顶部一块方形区域的绘制，为了尽量达到原图的效果，我们通过计算得到一定的长度和角度的参数，其内部的主要操作是控制 Turtle 的走线长度和转向角度，由于原图中转向角度已确定，在传参的时候只给定一个走线长度 len，用于控制方形区域的大小，函数中的 t 和 w 分别表示海龟和页面窗口。

Stick(p,t,w)函数用于实现船体顶部的几根桅杆的绘制，在这里需要说明的是，程序最后是将 Quare 函数放在 stick 函数中调用的，基本思路是固定船头一点，turtle 从此处出发，水平方向向船尾移动，在移动的过程中，在恰当的位置的竖直方向上画出桅杆和方形区域。画完船顶桅杆和方形区域后，turtle 会回到船头的位置。在此说明 p 代表船头的位置坐标，t 和 w 分别代表海龟和页面窗口。

Flat(p,t,w)函数用于实现船上甲板的绘制，其基本思想和 Quare 函数一致，通过控制 turtle 的走线与偏转角度，在船顶勾画出一个甲板区域并填充与甲板不一向的颜色以作区分。其中 p 仍代表船头的位置，t 和 w 分别代表海龟和页面窗口。

String(x,y,t,w)　函数有三个，此系列函数用于实现船头弯曲的绳子和木桩的绘制。现就其中一个函数做一些说明：x,y 分别代表船头的横坐标与纵坐标，首先将 turtle 移动到船头与水面之间的某个位置，并在此处调用内置函数 circle(radius,angle)做向上一小段圆弧连接到船头，然后再向下做一小段圆弧到达某一位置，再从该位置再做一段圆弧并在该圆弧末端绘制出一根竖直的木桩，整个过程结束。这里所用到的思想就是用几段弯曲的弧线去逼近一根弯曲的绳子，以达到较为自然与柔和的效果。

由于该幅作品中有多艘小船，其它小船的基本实现方法与此类似，在这里就不做赘述。

二、对于一个颜色渐变的月亮，利用一个主要函数 moon(turtle,radius,a,b,c)来绘制，辅助函数 line(turtle,sem)辅助绘图。其中，radius 是月亮的半径，a,b,c 为 RGB 模式下的颜色参数，sem 为半径。

moon(turtle,radius,a,b,c)中，基本思路为将月亮分为若干个环形区域，具体操作为：从月亮的最外围开始，从圆周 C 上的某一点出发，绘制一个比圆周 C 稍微小一些的圆周 C1，在 C 与 C1 所围成的不规则环形区域中（形状类似于“c”）填充颜色 Color1，之后利用递归重复此类操作，直到月亮中的所有区域都被填充满。

具体实现过程中，从最大圆周的 C 的最低点出发，画一个比 C 的半径（radius）小 5 的圆周 C1，将其填充上 RGB 模式下的颜色。那么问题则变为给一个 C1 圆周填充上渐变色，问题规模缩小，并且调用了自身。结束条件为圆周的半径(radius)小于等于 0 时。

而对于颜色的渐变，在每次递归调用时，将颜色参数 a,b,c 适当地增加一些，最终令颜色变为灰白色。

辅助函数 line(turtle,sem)是用来实现 moon()中画圆周的部分，从圆周上某一

点出发，每次转过一定的角度，前进一定的距离，当转过 360°时，即可回归起始点。

具体实现过程中，我们让海龟每次转过（360° / 288 = ）1.25°，这个圆周的半径为 sem，则该圆的周长为 （ 2 x π x sem = ） 6.28sem。则每次转过 1.25°应走过的距离 l = （ 6.28 sem / 288 = ）0.0218sem，从而形成一个闭合的圆周。

三、对于动态落叶与船摆的实现，我们并未自己定义函数，而是引用了 Python 内置函数 time 和 random，前者用来调节每秒放映的帧数，后者用来随机确定落叶的位置。

四、对于长宽逐渐减少的竹子，我们定义了三个函数，分别是：

用辅助函数 Bamboo1( a , b )实现最基本的竹节，先绘出竹节的最外围，然后填充颜色即可。a,b 用来控制竹节的长与宽。

主函数 Bamboo2 则利用递归画出整个竹子，对于给定的起始竹节（ a , b ），当 a 不小于 4 时，则利用 bamboo1 绘制一个（a , b）的竹节，然后再调用 Bamboo2 自身，即 Bamboo2( a – 2 , b – 5 ),直到 a < 4，绘制一个（a , b）的竹节，结束程序。

辅助函数 Bamleaves(turtle,p)则用来在位置 p 处用 turtle 绘制一片叶子，将一片叶子分为不同半径，弧度对应的若干圆弧，整合起来则可以实现叶子的重现。

五、对于一个颜色渐变的山峦以及山峦里的零落小树的绘制，我们定义了 paint($r_1$,$g_1$,$b_1$,$r_2$,$g_2$,$b_2$,$x_1$,$y_1$,$x_2$,$y_2$),tree(),kumol()三个函数。注意到 paint 函数里参数众多，其本质上是实现颜色在（$r_1$,$g_1$,$b_1$)和($r_2$,$g_2$,$b_2$)之间的渐变，渐变范围限定在 $x_1$-$x_2$ 之间，主要走线方向为上下走线。在 kumol 函数里，嵌入了多个 paint 函数，以此来弥补单个 paint 函数渐变距离的不足，Tree 函数旨在模仿原图中的山峦中较小的树木，注重于写意，因此引用了随机数来决定枝条的数目。

六、对于颜色渐变更为复杂的浮云的描绘，此段代码共设置了四个函数。Firt() 函数用来设定 turtle 的初始位置与方向，其参数即为 turtle；arc（t，l，a，n）函数用来实现许多圆弧即浮云部分外廓的绘制，其中 t 为 turtle,l 为圆弧的半径，用

于辅助 turtle 转向的渐变，n 代表循环画圆弧的次数。Shape1（t,sem,a,b,c）函数则是用来实现浮云内部颜色的渐变，t 代表 turtle,sem 用来辅助调节各个圆弧的半径，a,b,c 则为一次调用函数所填充的颜色。在这里需要说明的是，sem 是随着调用逐渐加大的，当 sem 大于 20 时，结束整个递归过程。另外当 a>30 时，（a,b,c）在 shape1 函数中是逐渐减小的，当 a<255 时，（a,b,c)在 shape2 函数中是逐渐增大的。通过 shape1 和 shape2 的调用，实现一个颜色从外到里先由深变浅，再由浅变深的效果。

### 9.2.2 程序限制

一、在用递归画船体的过程即 drawline 函数中，我们事先会设定一个线的长度以避免无限递归的出现。另外在每次递归调用的同时，我们都会用与之匹配的线性方程去调整线的长度。为达到较好的绘制效果，我们对参数 Len 做出以下要求：

1、必须大于事先设定的线的长度

2、在不改变函数内部线性方程的前提下，不得随意更改 Len 的长度，否则会出现因递归不够而导致船体变短的极端情况

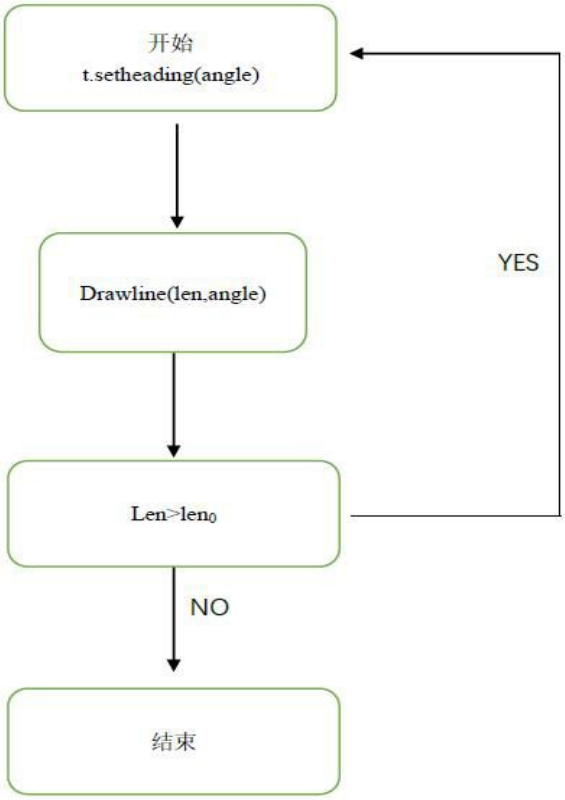
二、在画月亮的过程中，要实现计算好递归的次数，以确定颜色的初始参数，确保颜色参数的变化的最终效果是颜色变为灰白色。如果递归次数过多，或者初始颜色参数过小，会导致颜色参数溢出，即参数大于 255 的情况，使绘图失败。因此，moon()函数对于参数有以下要求：

1、可以大致确定递归的调用次数

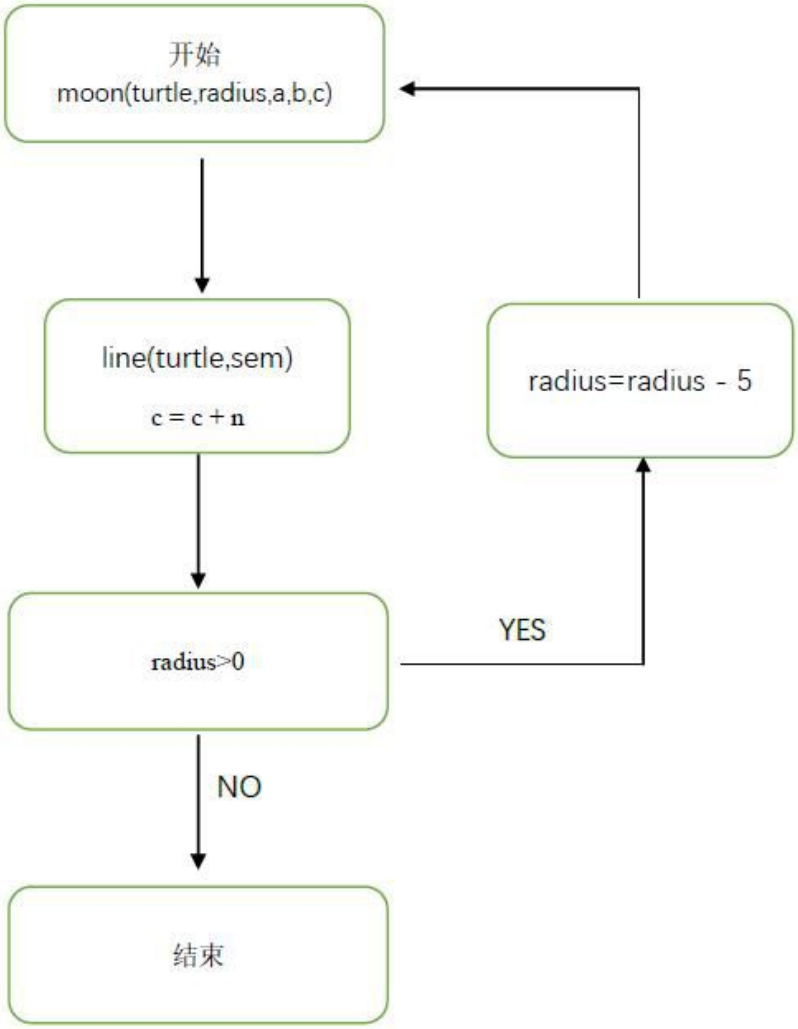
2、颜色参数要根据调用次数事前调整

## 9.2.3 算法流程图

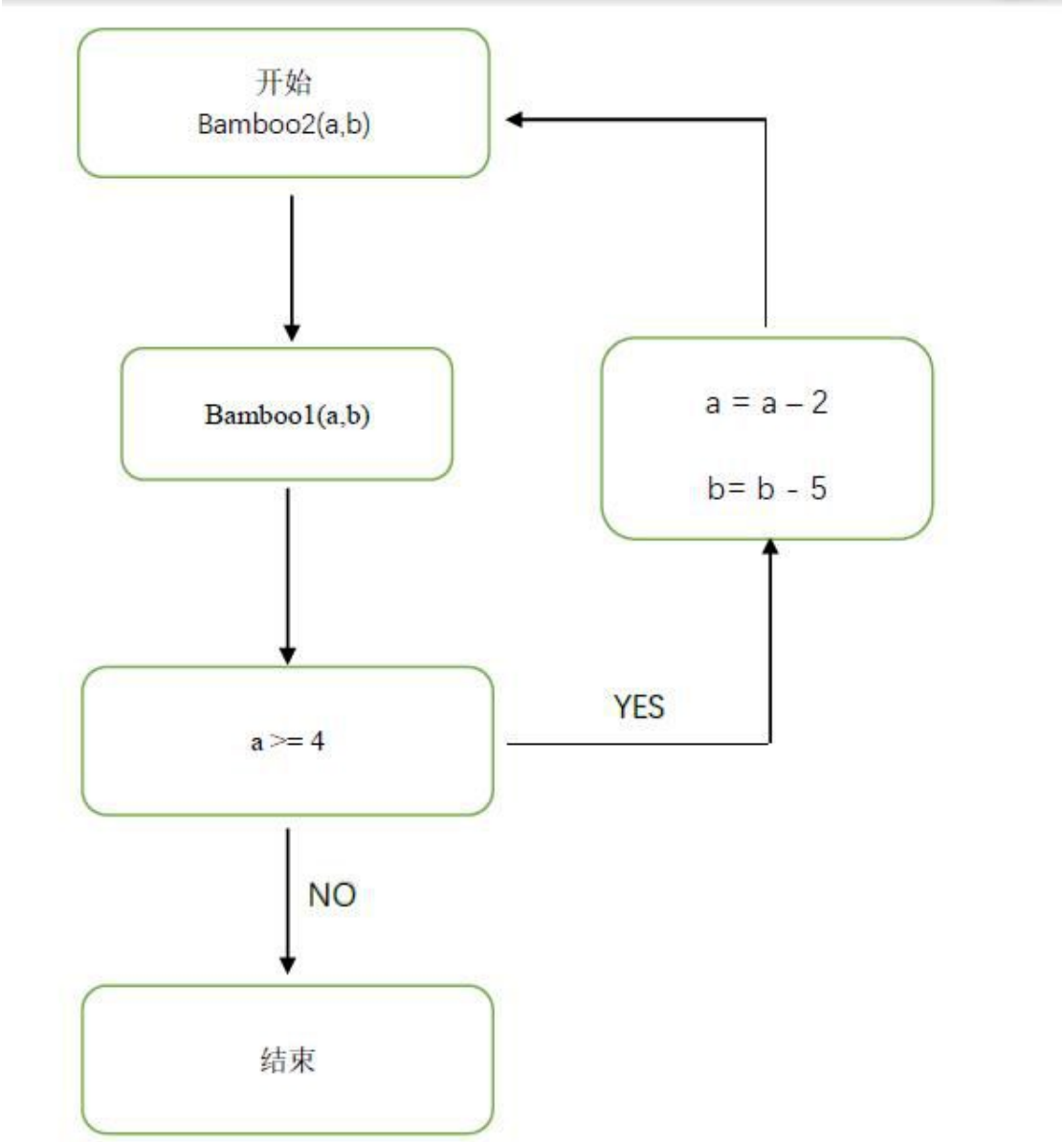
一、小船中递归



## 二、月亮中递归流程图（浮云中的递归思路同月亮）



三、竹子中递归流程图




## 9.3 实验结果

### 9.3.1 实验数据

实验环境说明：

硬件配置：Intel(R) Celeron(R) CPU N2940 @1.83GHz 4.00GB 内存

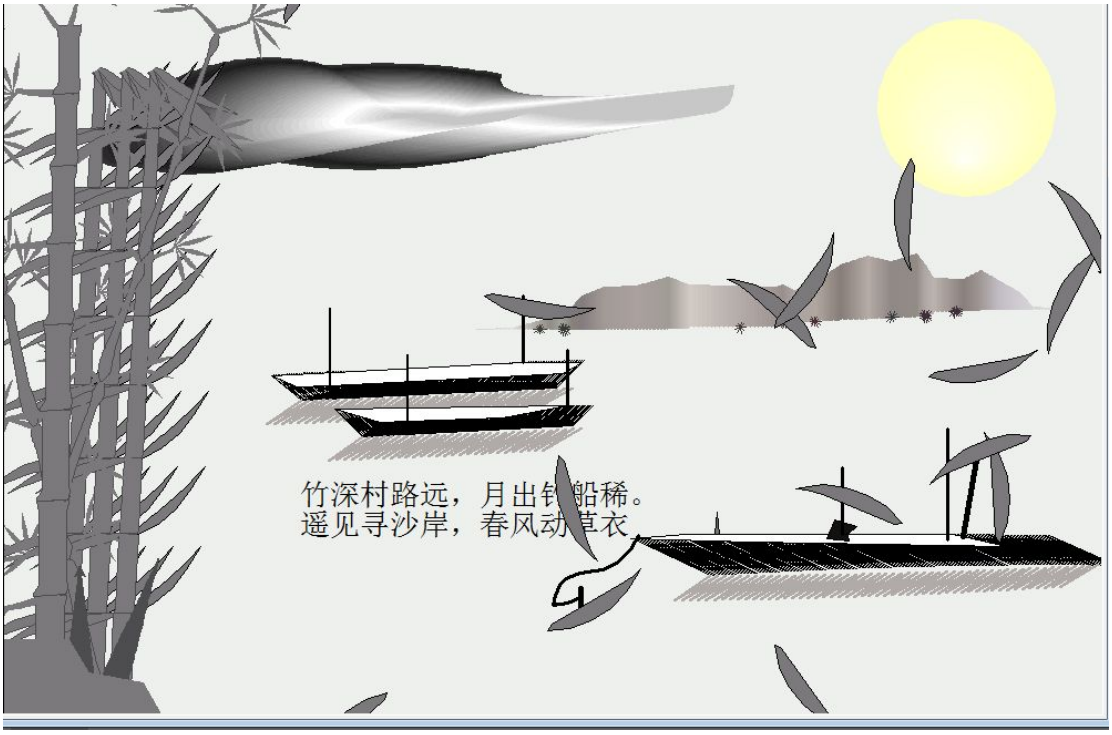
操作系统：Windows 8.1 专业版 64 位操作系统，基于 x64 的处理器

Python 版本：Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:54:25) [MSC

v.1900 64 bit (AMD64)] on win32

## 9.3.2 作品描述

此次作品总体上是一幅比较有意境的水墨山水画，涉及到的主要意象有明月、浮云、小楫轻舟、竹林、山峦，此为静景；另外还会有一个竹叶飘落的动态实现。关于整幅作品的看点，从整体上来看，各种充满诗意的意象巧妙组合，点缀于整块画布，营造出的一种水墨风格其实就是一个看点；其二，整幅画中多处用到色彩的渐变，在绘制中观看色彩的渐变过程也是充满了乐趣与享受；其三，小组成员严格参照原图的坐标及比例，在绘制的过程中也是细致入微，尤其是在船的绘制中，细致到每一根桅杆，每一条细绳，在竹叶的绘制中也力求竹叶形状的相似；其四，除静景以外，整个画布上会有竹叶飘落，为整幅作品再添一些韵味。




## 9.3.3 实现技巧

1、此次作品的一个亮点在于整幅画颜色渐变实现的方面，为尽量使作品有山水画的风格，我们严格规避了单调的颜色填充以及笔直的直线绘制。我们不惜用大量的精力与时间去对画中的每一个意象做细致的描绘。此幅画中的月亮用到简单

的递归，将月亮细分为多个环形区域，再填充连续的颜色，最终呈现出了一个较好的渐变效果。同样用到渐变的还有浮云、山峦的描绘，其基本实现技巧与月亮的描绘大同小异，均用到了化整为零，逐个填充的思想。

2、此次作品的另一个亮点在于落叶与船摆动态的实现。而实现这一化静为动效果的方法，则是通过几个十分简单的内置函数：time 函数和 random 函数，通过原作品的 undo 操作及快速转换竹叶和轻舟位置的操作，以达到一个动态的效果。

## 9.4 实现过程总结

### 9.4.1 分工与合作：

在第一次小组会议上，由卢国军同学引导，我们合理划分了各个成员的任务，确定了合作的方式。

艺术设计需要集思广益，我们一致认为需要大家一起思考、讨论，这样大家的目标更为明确，效率会高一些。

编程的工作量较大，需要的人手多一些。经过交流讨论，我们推选编程基础较好，能力较强的韦庆朗同学主导程序的编写，与尹泽藩、徐贝贝、贾彩霖同学合作，共同完成程序。

卢国军和王宁则负责整个过程中记录组会内容、拍摄照片、编写报告以及做主题展示等，实时跟进任务的进程，确保将创意、思路、困难和解决方法记录下来。

为了保证记录的同学可以随时了解到进程以及程序的思路，在不能保证大家的计划表可以同步留出时间的情况下，我们确立若干次了小组会议并商议将程序的代码、新的进展以及组会内容总结和图片等，即时同步到微信群中，确保每一位成员都能参与任务，了解任务的进度。

图 1（第一次组会合影）

记录人：王 宁

2016 年 4 月 1 日

会议内容：

1、组员提出各自的想法，交流意见，最终选择莫奈的《野罂粟》作为小组将要实现的编程作品。

2、根据组员各方面的能力进行初步分工，分为编程小组和报告小组。

3、确定小组工作的日程安排，小组工作的进度需根据日程安排来逐步推进。

图 2（第二次组会合影）

记录人：王宁

2016 年 4 月 5 日

会议内容：

1、将莫奈的《野罂粟》中的各种意象分解，由编程小组承担对不同意象的重现任务。其中主要意象有云朵、罂粟花、树林、草丛、人物、房子等。

2、确定了云朵、花、树林、草丛可以用递归的方法去实现。同时确定了一张统一的坐标图，以防在最终意象的整合过程中出现偏差。

3、组员尹泽藩展示了他用于实现罂粟花重现的代码，大家了解其算法的基本思想，并经过讨论与研究，决定在其原代码的基础上，添加其他颜色的罂粟花，并将原花型做了修改，增加了花瓣。

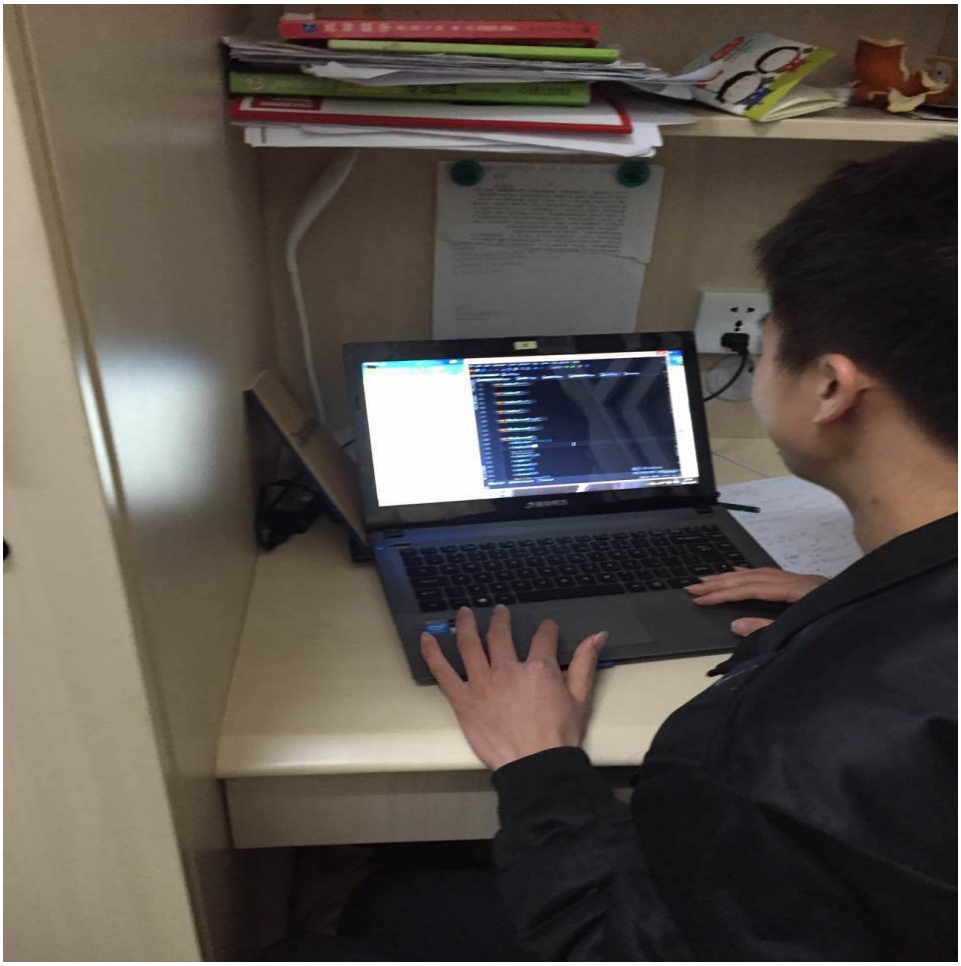


图 3（第三次组会现场）

记录人：王　宁

2016 年 4 月 6 日

会议内容：

1、主题由莫奈的《野罂粟》更改为山水画，仍然对意象进行拆分，主要有竹子、月亮、背景中的山水等

2、寻找可以作为模板的成品图片，确定一张统一的坐标图

图 4（第四次组会现场）

记录人：王　宁

2016 年 4 月 10 日

会议内容：

1、组员交流编程进展情况，进行效果展示，共享某些效果的实现技巧，对于各成员提出自己的看法，对于成果中不合适的部分提出了修改建议。

2、大致确定后续计划，14 日由卢国军同学进行课堂展示，向大家阐述我们创意来源及构成要素。编程组的成员继续完成和修缮程序。

图 5（第五次组会合影）

记录人：王宁

2016 年 4 月 14 日

会议内容：

1、组员展示各自程序效果，提交代码

2、记录组成员阅读理解代码，编写有关报告

3、由卢国军同学收集，韦庆朗同学整合最终程序

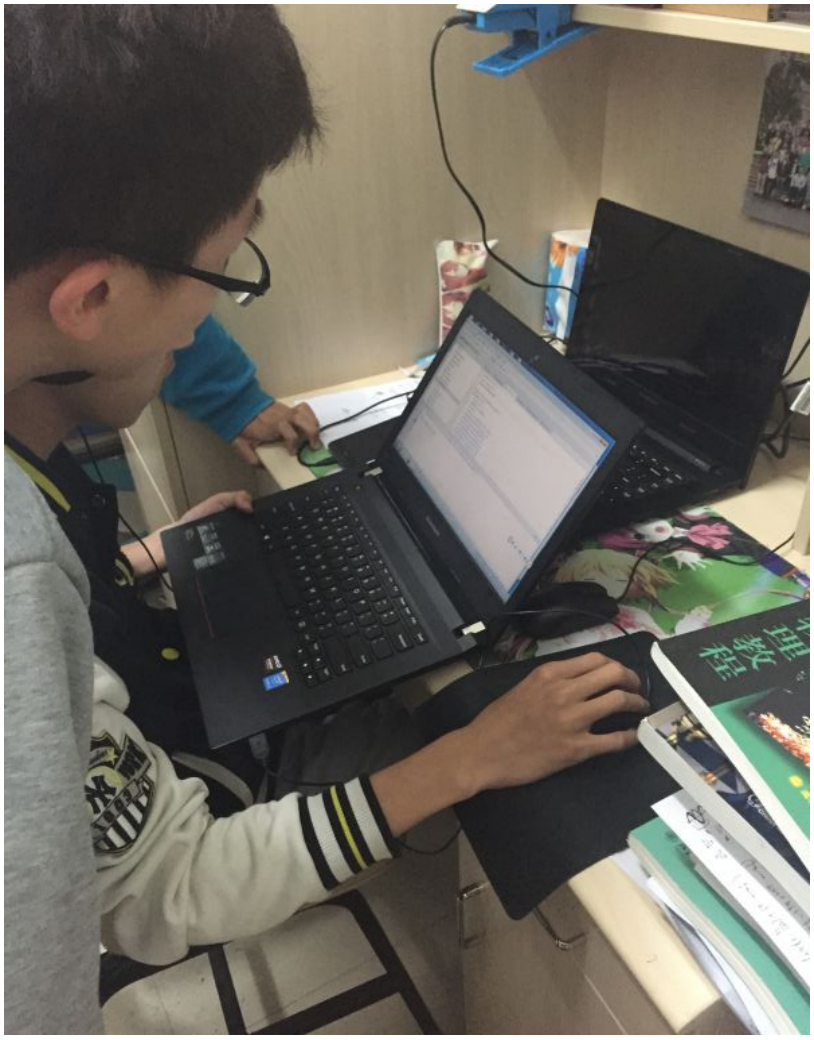


图 6（第六次组会现场）

记录人：王宁

2016 年 4 月 16 日

会议内容：

1、组员观看初步整合效果，并提出一些建议继续完成景物的整合

2、小组决定在原有的基础上加一些与作品相符合的诗词

3、决定于 17 日正式完成整个代码的整合以及报告的收尾工作

## 9.4.2 经验与教训

本次实习过程中，我们从递归的特点出发，提出了多种创意，对每一种创意进行了分析和拆解，依次进行排除筛选，进行难度梯度划分，选出最初方案和备

用方案，我们认为这种做法是比较有效率的。在发现问题时能及时纠正错误，甚至改变方向，以减少时间损失。将一幅图像进行拆分，在分派给指定的组员，可以实现任务合理分配，保证完成的效率及质量。

最为重要的是，我们小组组员办事效率较高，没有出现任务延期的现象，配合度较高，可以保质保量地按时完成任务。

不足之处在于，我们对于某些任务（野罂粟）的难度估计过于乐观，只关注了大片重复的背景，但是对于背景中细节的重复频率，重复位置没有做出预测，导致虽然可以画出图像，但拟合度较低，与原作品的差距较大，而且关于颜色渐变的实现难度，也并没有进行预测，这些失误让我们不得不放弃最初的计划，转而将目标放在山水画上。

### 9.4.3 建议与设想

我们希望在这个程序的基础上，可以实现更改参数从而长出不同形态的竹子，画出不同形态的山水。

## 9.5 致谢

在此感谢所有小组成员为此次作业所付出的一切。

编程组的成员韦庆朗、尹泽藩，徐贝贝和贾彩霖在期中考试的压力之下，仍然抽出大把的时间用于编程。在组员完成野罂粟部分程序的情况下，小组临时又改变了方向，各组员临危不乱，脚踏实地，最终圆满的完成了任务。

记录组的成员卢国军和王宁需要记录每次组会内容，进行梳理总结，反映在报告上。每天需要实时跟进程序进度，理清程序思路，用文字进行描述。

最后，感谢每天为我们答疑的老师和助教，为我们解决编程中遇到的问题。

# 10 P 组

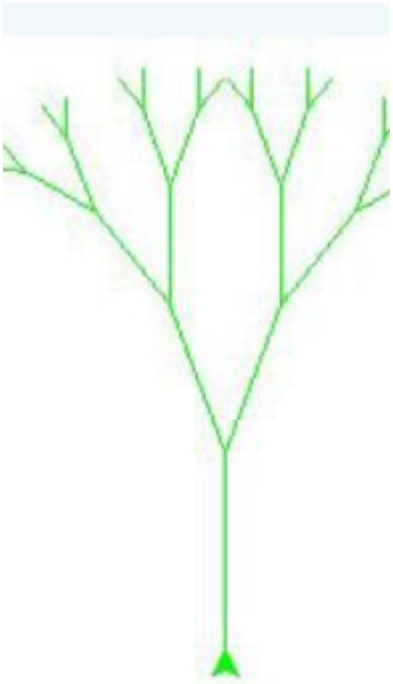
# 数据结构与算法课程

## 递归视觉艺术实习作业报告

名称：英雄 P

小组名单：周强*，许严，耿晓状，陈相，徐运铎，薛莅治

**摘要**：树的递归课程，我们联想到和树密切相关的一个无人不知的游戏——dota。我们尝试从游戏中截取高清截图给予我们素材，然后以递归树为背景，把游戏截图模型化，即分成一块一块的较简单的几何图形，然后用强大的海龟作图和列表 字典 元组 字符串 随机数这些数据类型依然分片完成背景树、炮台、路、天空。最后，仿造游戏里的设置，完成一个伟大的人物英雄的设计。

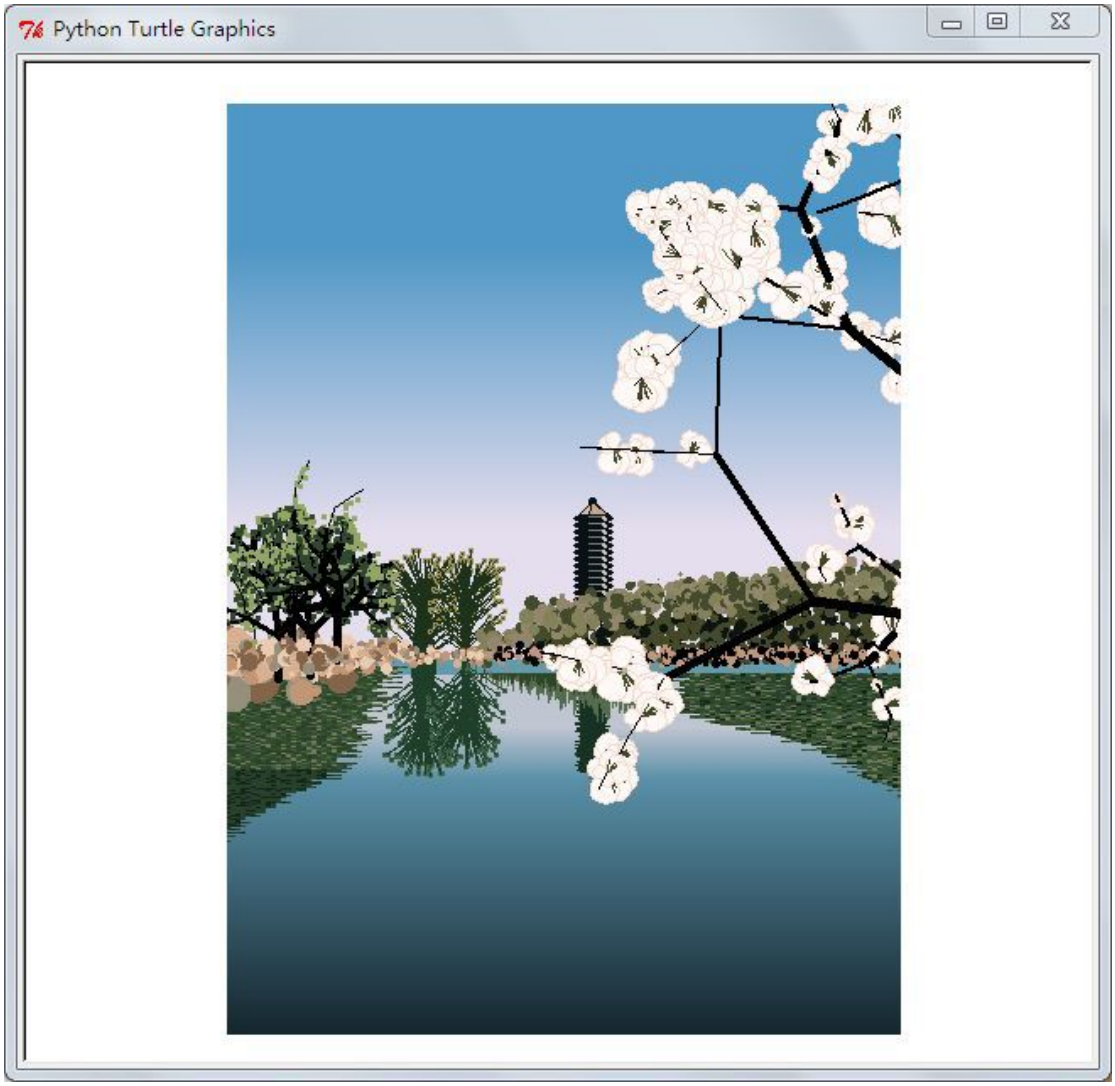


**关键词　英雄　树　游戏**

# 10.1 创意过程与递归思路

## 10.1.1 作品总体介绍

作品的呈现为类似 dota 游戏英雄的姿态，一个英雄以充满奇幻色彩的天空为背景，由许多茂密的树木和几座炮台包围，英雄脚踏石板路，露出威武之态。考虑到递归的作品虽然有很美的对称性，但是作品比较千篇一律，我们从陈春含学姐助教的未名湖作品中得到了一些启示（该作品中的递归成分很少，几乎只有从画面右端伸出的枝条和塔）：一个好的作品不一定要以递归为中心，相反，可以巧妙地以递归为辅衬，这样即能满足“作品中包含递归”的要求，也能在作品的创作上更自由。于是，我们便以递归树作为作品的背景，而中心则以标题“英雄”为主。运用的数据结构“四大容器”都有，但更复杂的数据结构暂时没有用到。实现作品的技巧，即以“透视”画法依次呈现背景，最后呈现英雄。如此做，小组之间的分工也就更明确。





（希望陈斌老师不要生气，画得很丑，但是我们是出于对伟大的您的敬仰和喜爱才画您的，绝对是这样！）

### 10.1.2创意来源

我们的创意来源除了游戏背景和主题，就是英雄。我们打算自己创造一个游戏里原本没有的“东方英雄”，谁能胜任这个角色呢。经过商讨，我们觉得，幽默风趣又不乏智慧光辉的伟大的作为引导我们进入 python 大家庭的陈斌老师再合适不过。大家一拍即合，就这么定了。

### 10.1.3递归思路

我们的递归思路主要以课堂所讲的递归树为主，通过引入随机数，把由若干圆弧构成的树叶加入到树枝末尾，并且利用递归画出很对称的松树，分配合理的草地，空中的云朵和闪电，先画出模型（闪电草地等），然后递归实现多个模型，递归的作用一是增加数量，可以是作品部分的数量（如树枝），亦可以是整体的数量，二是调节整体作品中多个重复对称部分的大小（如远近的树）以此作为英雄背景。

## 10.2 程序代码说明

### 10.2.1数据结构说明

由于本小组的 python 水平有限，我们很难想大神一样把各种线性非线性数据结构运用得行云流水，但是我们运用基本数据结构“四大容器”，随机数，也在一定程度上达到了我们的预期效果，也一样完成了作品，我们还是很拼的，但奈何基础不尽如人意。

### 10.2.2函数说明

```
def zeus():
```
画英雄

```
def shandian(x,y,h,t,angle,hh,hhh):
```
画闪电，分别为长度，角度，坐标

```
def yun():
```
画随机云

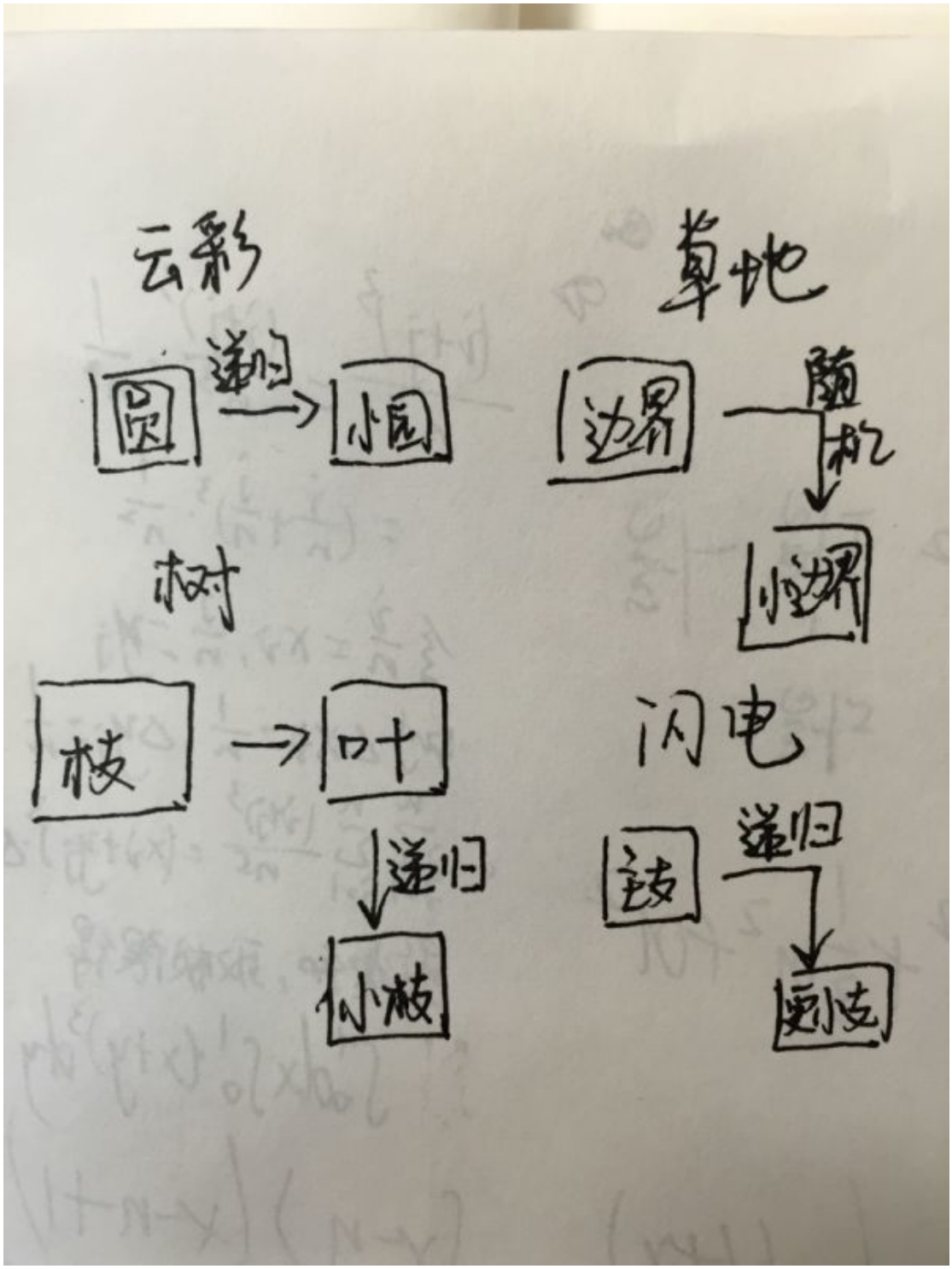
```
def stone(x1,y1,x2,y2,x3,y3,x4,y4,t,color1,color2,color3):
```
十三石头

```
def tree(t,width,a,se1,se2):
```
树的颜色，叶子的颜色，高度

## 10.2.3程序限制

暂未可知

## 10.2.4算法流程图



# 10.3实验结果

## 10.3.1实验数据

实验环境说明：
硬件配置（Inter Core i7）
操作系统（windows10）
Python 版本（3.4）

## 10.3.2作品描述


递归闪电


帅气英雄


随机云朵

 递归树

### 10.3.3实现技巧

暂无

## 10.4 实习过程总结

### 10.4.1分工与合作

创意总监：全组 报告：许严 英雄创作：耿晓状 薛莅治 整体背景：徐运铎 陈相 职业种树人：周强 视频后期制作：耿晓状 薛莅治

### 10.4.2经验与教训

经验：本次大作业无疑让我们队 python 这个神奇而方便的语言有了更深刻的了解，在完成的过程中，我们结合《画图》找到了合适颜色的像素点 RGB 值，我们知道怎样调节树木的位置让它们更好地呈现透视效果，在完成自己的部分之后我们无一不很开心，虽然我们画的并不是很棒，但是的确体会到了运用 python，运用程序呈现作品而带来的优越感和成就感，这次作业也让我们更喜欢上了 python。
教训：由于接触 python 的时间有限，我们画的树，路，天空等等的细节不丰满。这是一个很严重的问题。

### 10.4.3建议与设想

建议：希望有更多的时间来适应 python，总觉得老师给的时间有些仓促，毕竟来到了期中考试周，精力不集中，如果提前一两周公布大作业，那该多好。

## 10.5致谢

感谢全体小组成员，感谢陈斌老师孜孜不倦的教诲，感谢易超学长，孙鹰学长，春含学姐的帮助和指导，大家做得都非常棒！

## 10.6参考文献

递归课件
《大神教你 matlab》

# 11 Q 组

# 数据结构与算法课程

## 递归视觉艺术实习作业报告

## （Qoo）

凌坤 苏克凡 姚媛媛 周思阳* 赵旭炜

摘要：随着现如今知识总量的大幅增长，学科之间的交叉融合也已经成为了当前研究领域的一种重要方法。在数据结构与算法课程中学到的递归方法，亦可以与其他学科进行交叉，产生意想不到的效果。本组便在递归算法和植物学、抽象艺术等领域进行交叉，创作了此系列作品。本文主要讲述本组创意的产生过程，算法的实现和最终效果。

关键字：植物花序 蒙德里安 安迪·沃霍尔 递归方法

## 11.1创意过程与递归思路

### 11.1.1作品总体介绍

该作品分为两个部分。

第一部分：植物学家的实验田——一层一层剥开你的枝

该部分为植物花序图的绘制。花序是植物学上的一个概念，是指花序轴及其着生在上面的花的通称，也可特指花在花轴上不同形式的序列。花序可分为有限花序和无限花序，有限花序主要指聚伞花序，而无限花序包括总状花序、圆锥花序、穗状花序、复穗状花序、肉穗状花序、柔荑花序、伞形花序、伞房花序、复伞形花序、复伞房花序、头状花序、隐头花序。花序常被作为被子植物分类鉴定的一种依据。

我们在数十种植物花序中选择了总状花序、二歧聚伞花序、螺状聚伞花序、复伞形花序等四种花序。每个花序的绘制分为模式图和实例图两个部分，模式图主要依据《植物学（第二版）》上的标准模式进行绘制，实例图则选用这四种花序的代表植物：紫藤、胡萝卜、聚合草和卷耳来绘制。该部分除总状花序·紫藤

仅使用循环结构外，其他部分均采用了递归的方式进行枝叶的绘制。

第二部分：蒙德里安的辣椒田——过分的现代抽象艺术

该部分是现代抽象画的创造性表达。主要借鉴了蒙德里安和安迪·沃霍尔两位画家的绘画形式，来表现我们自己内心的情感。

彼埃·蒙德里安（Piet Cornelies Mondrian），荷兰画家，风格派运动幕后艺术家和非具象绘画的创始者之一。出生在荷兰的阿姆尔弗特，曾在海牙、鹿特丹、罗伦、巴黎和伦敦等十多处地方居住作画，晚年移居纽约，大幅度扩展创作领域，对建筑、工艺和设计产生很大影响。蒙德里安是几何抽象画派的先驱，与德士堡等组织“风格派”，提倡自己的艺术“新造型主义”。认为艺术应根本脱离自然的外在形式，以表现抽象精神为目的，追求人与神统一的绝对境界，亦即今日我们熟知的“纯粹抽象”。

安迪·沃霍尔（Andy Warhol）被誉为 20 世纪艺术界最有名的人物之一，是波普艺术的倡导者和领袖，也是对波普艺术影响最大的艺术家。他大胆尝试凸版印刷、橡皮或木料拓印、金箔技术、照片投影等各种复制技法。沃霍尔除了是波普艺术的领袖人物，他还是电影制片人、作家、摇滚乐作曲者、出版商，是纽约社交界、艺术界大红大紫的明星式艺术家。

该系列作品以蒙德里安《灰色的树》与《百老汇的爵士乐》两幅画为蓝本，借鉴安迪·沃霍尔的艺术风格，创造出一种由随机产生的不确定性与相似性，反映出本组对人生的一些终极思考。

### 11.1.2 创意来源

在陈斌老师展示出分形树第一版的那一刻，作为曾学习生物竞赛的几位便想到了植物学中叶序和花序的知识，觉得可以以此为主题进行创作。于是便马上召集组员，迅速开始了前期的准备工作。

而在创作花絮图的过程中，大家偶然发现了我们创作的分形树与蒙德里安伟大作品《灰色的树》的某些一致性，于是决定在花序的基础上进一步进行抽象化的创作，经过系列资料的查询，我们选择了以蒙德里安画作为蓝本，以安迪·沃霍尔的构图作为构架，创造出属于自己的抽象艺术，于是便有了 Qoo 组现在的作品。

### 11.1.3递归思路

#### 11.1.3.1 胡萝卜中的递归：

```
def branch(t,linelen):  #枝干#
```

```
……
    if linelen>30:
        t.fd(linelen)   #往上画出长为 linelen 的一条线段#
        t.rt(30)   #海龟向右转 30°#
        branch(t,linelen*0.7)   #递归，画出一条长为 linelen*0.7 的线段#
        t.lt(20)   #海龟向左转 20°#
        branch(t,linelen*0.7)    #递归，画出一条长为 linelen*0.7 的线段#
        t.lt(20)    #海龟向左转 20°#
        branch(t,linelen*0.7)    #递归，画出一条长为 linelen*0.7 的线段#
        t.lt(20)   #海龟向左转 20°#
        branch(t,linelen*0.7)   #递归，画出一条长为 linelen*0.7 的线段#
        ……
```

### 11.1.3.2 聚合草中的递归：

```
def branch(t,linelen):
    t.lt(30)
    if linelen>20:
        ……
        fl(t,linelen*0.5)   #执行 fl(t,r)函数，画出倒着的淡紫色小花#
        branch(t,linelen*0.618)   #递归，直到 linelen<=20 为止#
        ……
```

### 11.1.3.3 卷耳中的递归：

```
def branch(t,linelen):   #画“没有尾巴”的一株卷耳#
    if linelen>20:
        ……
        branch(t,linelen*0.618)   #递归，画叶子，画笔粗 linelen*0.618*0.0618，海龟左转
20°，画长为 linelen*0.618 的线段#
        ……
        branch(t,linelen*0.618)   #递归，画叶子，画笔粗 linelen*0.618*0.0618，海龟左转
20°，画长为 linelen*0.618 的线段#
        ……
```

### 11.1.3.4 复伞形花序

```
def compound_umbel(branchLen,t): #主干结构
    if branchLen <= 100: #判断枝干长度
        t.width(2) #设置宽度
        stalk(branchLen,t) #画出细节部分
        for i in range(4): #循环 4 次
            t.right(50) #右转 50 度
            wide(branchLen,t) #设置宽度
            t.forward(branchLen) #前进枝干的长度
            t.right(90) #右转 90 度
            t.width(2) #设置宽度
            t.begin_fill()
            t.circle(5) #画一个圆点
            t.end_fill()
            t.left(90) #左转 90 度
            t.backward(branchLen) #后退枝干的长度
        t.left(75) #左转 90 度
    else:
        t.width(3) #设置宽度
        stalk(branchLen,t) #画出细节部分
        for i in range(4): #循环 4 次
            wide(branchLen,t) #设置宽度
            t.right(50) #右转 50 度
```

```
            t.forward(branchLen) #前进枝干的长度
            compound_umbel(branchLen/3,t)#递归
            t.backward(branchLen) #后退枝干的长度
        t.left(75) #左转 75 度
```

这部分代码是关于画出花序的主干部分的，branchLen 表示枝干的长度，函数 stalk（）表示画出枝干旁边的两片小叶子，函数主体部分是一个判断，通过判断完 branchLen 的长度，如果小于 100，就画出上面带原点的部分；如果大于 100，就会分叉，分叉就通过再次递归 compound_umbel（）这个函数本身来实现。

### 11.1.3.5 螺状聚伞花序

```
def bostryx(branchLen,t,angel): #主干形状
    if branchLen < 40: #判断枝干的长度
        t.left(90) #左转 90 度
        t.begin_fill()
        t.circle(branchLen/20) #画一个圆点
        t.end_fill()
        t.right(90) #右转 90 度
    else:
        t.width(branchLen/20) #设置宽度
        stalk(branchLen,t,angel) #画出细节部分
        bostryx(branchLen-10,t,angel-4) #递归
```

这部分代码是关于螺状聚伞花序的主体的螺旋，其中，函数 stalk（）表示画出

每次螺旋部位所附带的自身的细节，branchLen 为枝干的长度，函数主体通过判断 branchLen 的长度，如果 branchLen 的长度小于 40，函数结束，并画出最后那部分的形状，如果 branchLen 的长度大于 40，就缩小 bostryx()函数的大小，并通过 stalk（）函数画出细节。

### 11.1.3.6 总状花序

```
def raceme(branchLen,t): #主干的形状
    if branchLen<35: #判断枝干的长度
        stalk(branchLen,t) #画出节点处的细节
        t.forward(branchLen-2) #前进枝干的长度减 2
        spot(t) #画出圆点
    else:
        stalk(branchLen,t) #画出细节部分
        raceme(branchLen-5,t) #递归
```

这部分函数是关于总状花序主体的形状，其中函数 stalk()表示每一次递归所画出的细节部分，branchLen 为枝干的长度，函数主体通过判断 branchLen 的长度，如果 branchLen 的长度小于 35，函数结束，并画出最顶处的形状，如果 branchLen 的长度大于 35，那么就缩小 raceme（）函数的大小，并通过 stalk（）函数画出细节。

### 11.1.3.7 二歧聚伞花序

```
def pleiochasium(branchLen,t): #主干的形状
    if branchLen <=60: #判断枝干的长度
        width(branchLen,t) #设置宽度
        stalk(branchLen,t) #画出节点处的细节
        for i in range(3): #循环 3 次
            t.right(30) #右转 30 度
            t.forward(branchLen) #前进枝干的长度
            t.left(90) #左转 90 度
            t.begin_fill()
            t.circle(3) #画一个圆点
            t.end_fill()
            t.right(90) #右转 90 度
            t.backward(branchLen) #后退枝干的长度
        t.left(30) #左转 30 度
    else:
        width(branchLen,t) #设置宽度
        stalk(branchLen,t) #画出节点处的细节
        t.right(30) #右转 30 度
        t.forward(branchLen) #前进枝干的长度
        pleiochasium(branchLen-40,t) #递归
        t.backward(branchLen) #后退枝干的长度
```

```
    width(branchLen,t) #设置宽度
    t.right(30) #右转 30 度
    t.forward(branchLen-5) #前进枝干的长度减 5
    t.left(90) #左转 90 度
    t.begin_fill()
    t.circle(3) #画一个圆点
    t.end_fill()
    t.right(90) #右转 90 度
    t.backward(branchLen-5) #后退枝干的长度减 5
    width(branchLen,t) #设置宽度
    t.right(30) #右转 30 度
    t.forward(branchLen) #前进枝干的长度
    pleiochasium(branchLen-40,t) #递归
    t.backward(branchLen) #后退枝干的长度
    t.left(30) #左转 30 度
```

这部分函数是关于二歧聚伞花序主体的形状，其中函数 stalk()表示在枝干的分叉处的小叶子，branchLen 为枝干的长度，函数主体通过判断 branchLen 的长度，如果 branchLen 的长度小于 60，函数结束，并画出结束时的形状，如果 branchLen 的长度大于 60，那么就缩小 raceme（）函数的大小，递归三次，得到最终的图像。

# 11.2程序代码说明

## 11.2.1函数说明

### 11.2.1.1 胡萝卜

```
import turtle,random
t=turtle.Turtle()  #生成一只海龟#
t.lt(90)  #左转 90°#
t.speed('fastest')
def cf(t,r):  #颜色为粉红色的花#
    t.color('#EEE8AA','#FFFFE0')
    for i in range(5):  #循环 5 次，共有五朵花瓣#
        t.begin_fill()
        t.rt(30)  #海龟右转 30°#
        t.circle(r,60)  #画一个圆弧，半径为 r，角度为 60°#
        t.lt(120)  #左转 120°#
        t.circle(r,60)  #画一个圆弧，半径为 r，角度为 60°，此时画完一个花瓣#
        t.lt(78)  #左转 78°#
        t.end_fill()  #填充颜色#
        i+=1

def branch(t,linelen):  #枝干#
    t.color("#CDC673")
    t.pensize(linelen//20)
    if linelen>30:
        t.fd(linelen)  #往上画出长为 linelen 的一条线段#
        t.rt(30)  #海龟向右转 30°#
        branch(t,linelen*0.7)  #递归，画出一条长为 linelen*0.7 的线段#
        t.lt(20)  #海龟向左转 20°#
        branch(t,linelen*0.7)  #递归，画出一条长为 linelen*0.7 的线段#
        t.lt(20)  #海龟向左转 20°#
        branch(t,linelen*0.7)  #递归，画出一条长为 linelen*0.7 的线段#
        t.lt(20)  #海龟向左转 20°#
        branch(t,linelen*0.7)  #递归，画出一条长为 linelen*0.7 的线段#
        t.rt(30)  #海龟右转 30°#
        t.penup()
        t.bk(linelen)  #回到长为 linelen 的一条线段的末尾#
        t.pendown()  #落笔，继续循环，直到 linelen 小于 30#
    elif linelen<=30:
        cf(t,linelen)  #最后的每一个分支上执行 cf(t,linelen)函数，令其长出花朵#
```

```
        t.color("#CDC673")

def main(t,linelen):
    for i in range(2):   #画两个同样的图#
        t.penup()
        t.goto(-100-60*i,i*10)
        t.setheading(90)
        t.pendown()
        branch(t,linelen)
        t.rt(180)
        t.pensize(linelen//20)
        t.circle(linelen,60)   #在主干的下端加一条半径为 linelen，角度为 60° 的圆弧#
    t.ht()
```

### 11.2.1.2 聚合草

```
import turtle
t=turtle.Turtle()
t.speed('fastest')

def lf(t,r):   #画绿色的叶子#
    t.rt(90)    #海龟向右转 90°，此时指向屏幕的南方#
    a=t.pensize()
    t.pensize(1)
    t.color('#6E8B3D','#9ACD32')
    t.begin_fill()
    t.circle(r/446*265,90)   #画半径为 r/446*265，角度为 90° 的圆弧，此后指向屏幕的东方#
    t.rt(180)   #海龟向右转 180°，此后指向屏幕的西方#
    t.circle(r/446*183,-90)    #画半径为 r/446*183，角度为-90° 的圆弧，此后方向指向屏幕的北方#
    t.circle(r,90)   #画半径为 r/446*183，角度为-90° 的圆弧，此后方向指向屏幕的西方#
    t.end_fill()
    t.pensize(a)
    t.rt(180)

def petal(t,r):   #画淡紫色的小花#
    t.pensize(1)
    t.color("#9F79EE",'#FFE1FF')
    t.begin_fill()
    t.circle(r,60)   #初始时，海龟指向屏幕的东方，画半径为 r，角度为 60° 的圆弧，此后海龟的方向为东偏北 60° #
```

```
    t.circle(r/3,90)  #画半径为 r/3，角度为 90° 的圆弧，此后海龟的方向为西偏北 30°#
    t.right(150)  #海龟向右转 150°，此后海龟的方向指向屏幕的东方#
    t.circle(r/3,-60)  #画半径为 r/3，角度为-60° 的圆弧，此后海龟的方向为东偏南 60°#
    t.right(150)  #海龟向右转 150°，此后海龟的方向为西偏北 30°#
    t.circle(r/3,90)  #画半径为 r/3，角度为 90° 的圆弧，此后海龟的方向为西偏南 60°#
    t.circle(r,60)  #画半径为 r，角度为 60° 的圆弧#
    t.end_fill()  #填充颜色#

def fl(t,linelen):  #画倒着的有花萼的淡紫色小花#
    t.fd(linelen*0.1)
    d=t.heading()
    c=t.pensize()
    t.setheading(240)
    t.rt(30)  #此时海龟指向屏幕的南方#
    petal(t,linelen)  #执行 petal(t,linelen)函数，画出淡紫色的小花#
    t.lt(60)
    t.color('#6E8B3D','#9ACD32')
    t.begin_fill()
    t.circle(linelen,10)  #画半径为 linelen，角度为 10° 的圆弧#
    t.lt(90)
    t.circle(linelen/6,120)  #画半径为 linelen/6，角度为 120° 的圆弧#
    t.lt(90)
    t.circle(linelen,10)  #画半径为 linelen，角度为 10° 的圆弧#
    t.end_fill()
    t.color('#6E8B3D')
    t.pensize(c)
    t.setheading(d)
    t.bk(linelen*0.1)

def branch(t,linelen):
    t.lt(30)
    if linelen>20:
        t.pensize(linelen*0.0618)
        t.color('#6E8B3D')
        lf(t,linelen*0.8)  #执行 lf(t,r)函数，画出叶子#
        t.fd(linelen*0.309)
        t.circle(linelen*0.618,45)  #画半径为 linelen*0.618，角度为 45° 的绿色圆弧，作为植物的茎#
        fl(t,linelen*0.5)  #执行 fl(t,r)函数，画出倒着的淡紫色小花#
        branch(t,linelen*0.618)  #递归，直到 linelen<=20 为止#
        t.rt(30)
        t.circle(linelen*0.618,-45)
    else:
        t.penup()
```

```
        t.ht()

def main(t,linelen):    #画出一株完整的聚合草#
    t.color('#6E8B3D')
    t.pensize(linelen*0.0618)
    t.lt(90)
    t.fd(100)
    t.circle(linelen,45)
    branch(t,linelen)

def m(r,linelen):    #画出两株聚合草#
    t.penup()
    t.goto(-100,-100)
    t.pendown()
    main(t,linelen)
    t.goto(-50,-50)
    t.setheading(10)
    t.pendown()
    main(t,linelen*0.618)
```

### 11.2.1.3 卷耳

```
import turtle
t=turtle.Turtle()
t.speed('fastest')
def leaf(t):    #画绿色的叶子#
    t.pensize(1)
    t.rt(90)    #海龟向右转 90°，此后指向屏幕的南方#
    t.color('#A2CD5A','#CAFF70')
    t.begin_fill()
    t.fd(5)    #画长为 5 的线段#
    t.circle(10,90)    #画半径为 10，角度为 90°的圆弧，此后海龟的方向指向屏幕的东方#
    t.fd(5)    #画长为 5 的线段#
    t.circle(10,-90)    #画半径为 10，角度为-90°的圆弧，此后海龟的方向指向屏幕的南方#
    t.rt(180)    #海龟向右转 180°，此后指向屏幕的北方#
    t.circle(5,90)    #画半径为 5，角度为 90°的圆弧，此后海龟的方向指向屏幕的西方#
    t.lt(90)    #海龟向左转 90°，此后海龟的方向指向屏幕的南方#
    t.bk(5)    #向北画长度为 5 的线段，此后海龟的方向指向屏幕的南方#
    t.circle(10,-90)    #画半径为 10，角度为-90°的圆弧，此后海龟的方向指向屏幕的西方#
    t.bk(5)    #向东画长度为 5 的线段，此后海龟的方向指向屏幕的西方#
    t.circle(10,90)    #画半径为 10，角度为 90°的圆弧，此后海龟的方向指向屏幕的南方#
    t.rt(180)    #海龟向右转 180°，此后海龟的方向指向屏幕的北方#
```

```
    t.circle(5,-90)  #画半径为 5，角度为-90°的圆弧，此后海龟的方向指向屏幕的西方#
    t.end_fill()

def petal(t,r):  #画淡紫色的小花瓣#
    t.pensize(1)
    t.color('#7D9EC0','#F7F7F7')
    t.begin_fill()
    t.circle(r,60)  #初始时，海龟指向屏幕的东方，画半径为 r，角度为 60°的圆弧，此后海龟的方向为东偏北 60°#
    t.circle(r/3,90)  #画半径为 r/3，角度为 90°的圆弧，此后海龟的方向为西偏北 30°#
    t.right(150)  #海龟向右转 150°，此后海龟的方向指向屏幕的东方#
    t.circle(r/3,-60)  #画半径为 r/3，角度为-60°的圆弧，此后海龟的方向为东偏南 60°#
    t.right(150)  #海龟向右转 150°，此后海龟的方向为西偏北 30°#
    t.circle(r/3,90)  #画半径为 r/3，角度为 90°的圆弧，此后海龟的方向为西偏南 60°#
    t.circle(r,60)  #画半径为 r，角度为 60°的圆弧#
    t.end_fill()  #填充颜色#

def fl(t,r):
    for i in range(5):  #画一朵五个花瓣的花#
        petal(t,r)
        t.rt(12)
        i=i+1

def branch(t,linelen):  #画“没有尾巴”的一株卷耳#
    if linelen>20:
        leaf(t)  #画一个叶子#
        t.pensize(linelen*0.0618)
        t.color('#A2CD5A')
        t.lt(20)  #海龟向左转 20°#
        t.fd(linelen)  #画长为 linelen 的线段#
        branch(t,linelen*0.618)  #递归，画叶子，画笔粗 linelen*0.618*0.0618，海龟左转 20°，画长为 linelen*0.618 的线段#
        t.bk(linelen)
        t.pensize(linelen*0.0618)
        t.rt(40)  #海龟向右转 40°#
        t.fd(linelen)  #画长为 linelen 的线段#
        branch(t,linelen*0.618)  #递归，画叶子，画笔粗 linelen*0.618*0.0618，海龟左转 20°，画长为 linelen*0.618 的线段#
        t.bk(linelen)
        t.lt(20)
    else:
        fl(t,10)  #当 linelen<=20 时，在最短的茎上添上五瓣的花朵#
        t.color('#A2CD5A')
```

```
def jr(t,linelen):    #画一株“有尾巴的”卷耳#
    t.lt(80)
    branch(t,linelen)
    t.pensize(linelen*0.0618)
    t.rt(180)
    t.circle(200,40)

def main(t,linelen):    #画两株“有尾巴的”卷耳#
    jr(t,linelen)
    t.penup()
    t.goto(20,-20)
    t.pendown()
    t.setheading(45)
    jr(t,linelen-10)
    t.ht()
```

### 11.2.1.4 紫藤

```
import turtle,random

def lwsf(t,r):    #画左边部分的叶子#
    t.setheading(180)    #让海龟指向屏幕的西方#
    t.color('#EED2EE','#EED2EE')
    x=t.xcor()
    y=t.ycor()
    t.begin_fill()
    t.circle(210*r,61)    #画半径为 210*r，角度为 61°的圆弧，此后海龟指向西偏南 61°#
    t.rt(180)    #海龟向右转 180°，此后海龟指向东偏北 61°#
    t.circle(473*r,-31)    #画半径为 473*r，角度为-31°的圆弧，此后海龟指向北偏东 2°#
    t.rt(180)    #海龟向右转 180°，此后海龟指向南偏东 2°#
    t.circle(34*r,148)    #画半径为 34*r，角度为 148°的圆弧，此后海龟指向西偏北 30°#
    t.circle(792*r,22)    #画半径为 792*r，角度为 22°的圆弧，此后海龟指向西偏北 8°#
    t.lt(24)    #海龟向左转 24°，此后海龟指向西偏南 16°#
    t.circle(312*r,49)    #画半径为 312*r，角度为 49°的圆弧，此后海龟指向西偏南 65°#
    t.goto(x,y)
    t.end_fill()
    t.setheading(0)

def rwsf(t,r):    #画右边部分的叶子#
    t.setheading(180)    #海龟指向屏幕的西方#
    t.color('#CDB5CD','#CDB5CD')
    x=t.xcor()
```

```
    y=t.ycor()
    t.begin_fill()
    t.circle(210*r,-61)  #画半径为 210*r，角度为-61°的圆弧，此后海龟指向西偏北 61°#
    t.rt(180)  #海龟向右转 180°，此后海龟指向东偏南 61°#
    t.circle(473*r,31)  #画半径为 473*r，角度为 31°的圆弧，此后海龟指向南偏西 2°#
    t.rt(180)  #海龟向右转 180°，此后海龟指向北偏东 2°#
    t.circle(34*r,-148)  #画半径为 34*r，角度为-148°的圆弧，此后海龟指向南偏东 30°#
    t.circle(792*r,-22)  #画半径为 792*r，角度为-22°的圆弧，此后海龟指向南偏东 8°#
    t.rt(24)  #海龟向右转 24°，此后海龟指向南偏东 32°#
    t.circle(312*r,-49)  #画半径为 312*r，角度为-49°的圆弧，此后海龟指向南偏西 17°#
    t.goto(x,y)
    t.end_fill()
    t.setheading(0)

def drawbranch(t,linelen):
    while linelen>15:
        t.pensize(linelen//7)
        t.color("#CAFF70")
        t.fd(linelen*0.25)  #画茎的主干，长为 linelen*0.25#
        t.lt(135)  #海龟向左转 135°#
        t.circle(0.7*linelen,-45)  #画半径为 0.7*linelen，角度为-45°的圆弧，这是茎的分
支，在左边#
        t.pensize(1)
        lwsf(t,linelen*0.002)  #画左边的花瓣#
        t.pensize(linelen//7)
        t.color("#CAFF70")
        t.circle(0.7*linelen,45)  #画半径为 0.7*linelen，角度为 45°的圆弧，返回到主干茎
的末端#
        t.rt(135)  #海龟向右转 135°#
        t.fd(linelen*0.5)  #继续画茎主干，长为 linelen*0.5#
        t.lt(45)  #海龟向左转 45°#
        t.circle(0.7*linelen,45)  #画半径为 0.7*linelen，角度为 45°的圆弧，这是茎的分支，
在右边#
        t.pensize(1)
        rwsf(t,linelen*0.002)  #画右边的花瓣#
        t.pensize(linelen//7)
        t.color("#CAFF70")
        t.circle(0.7*linelen,-45)  #画半径为 0.7*linelen，角度为-45°的圆弧，返回到主干
茎的末端#
        t.rt(45)
        t.fd(linelen*0.25)
        linelen=linelen*0.9

def main():
```

```
t=turtle.Turtle()
t.speed('fastest')
t.penup()
t.goto(-300,300)
t.pendown()
t.rt(90)
for i in range(16):
    linelen=random.randrange(20,60,10)
    a=t.xcor()
    drawbranch(t,linelen)
    t.setheading(120)
    t.color('#EEE0E5','#EEE0E5')
    t.pensize(5)
    t.circle(20,-60)
    t.rt(120)
    t.circle(20,-60)
    t.penup()
    t.setheading(270)
    i=i+1
    t.goto(a+1.6*linelen,300)
    t.pendown()
t.ht()
```

#### 11.2.1.5 复伞形花序

wide（），设置枝干长度对应的宽度的关系

stalk（），画出节点两边的叶子

compound_umbel（），图像的主体，通过递归画出花序的结构，调用 wide（）和 stalk（），实现节点的细节和递归后宽度的变化

main（），调用所有的函数，调用 turtle 画图，并作出图像的边框

#### 11.2.1.6 螺状聚伞花序

stalk（）画出每个节点的细节

boystyx（）递归的主要体现，通过递归画出螺状的结构，调用 stalk（）实现节点的细节

main（）调用所有的函数，调用 turtle 画图，并作出图像的边框

#### 11.2.1.7 总状花序

spot（）画出枝干顶点的原点

stalk（）调用 spot（）函数，画出节点处的细节

raceme（）递归的主要体现，通过递归画出直线上升的结构，调用 stalk（）和 spot（）实现节点的细节

main（）调用所有的函数，调用 turtle 画图，并作出图像的边框

#### 11.2.1.8 二歧聚伞花序

stalk（）画出每个节点底部的两个分支叶子

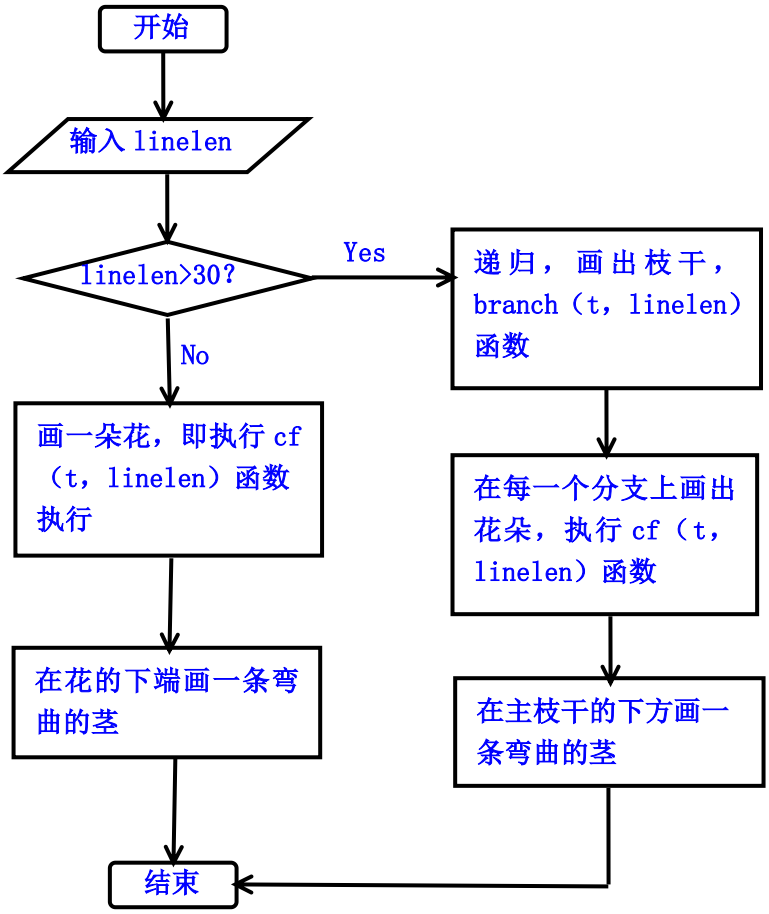
width（）设置枝干的长度和宽度的关系

pleiochasium（）递归的主要体现，通过递归画出，调用 stalk（）和 width（）

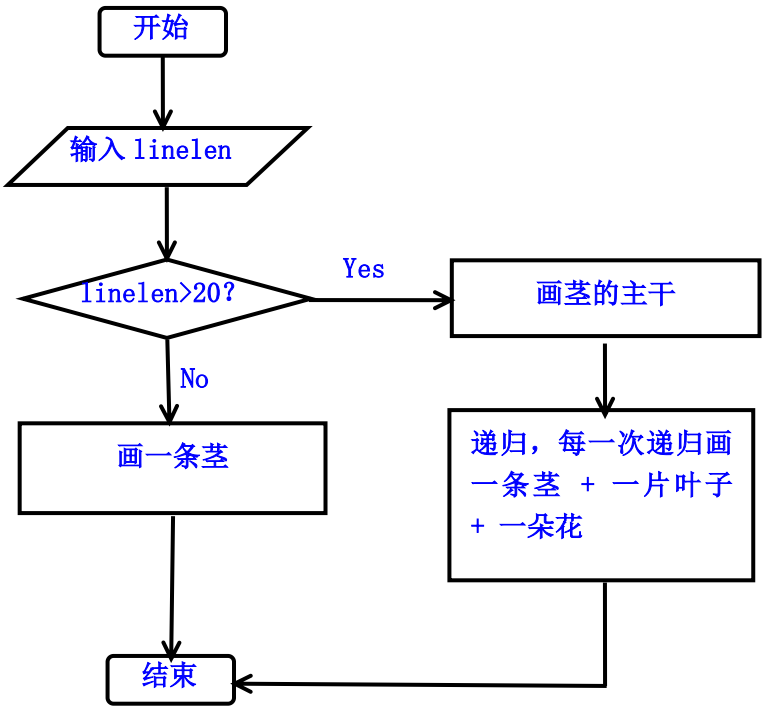
main（）调用所有的函数，调用 turtle 画图，并作出图像的边框

## 11.2.2算法流程图

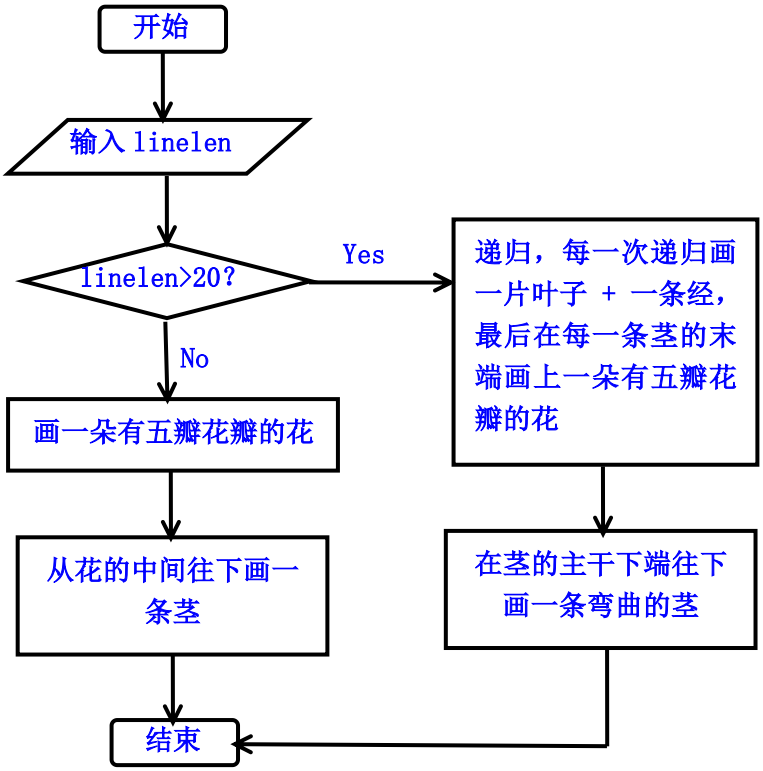
### 11.2.2.1 胡萝卜

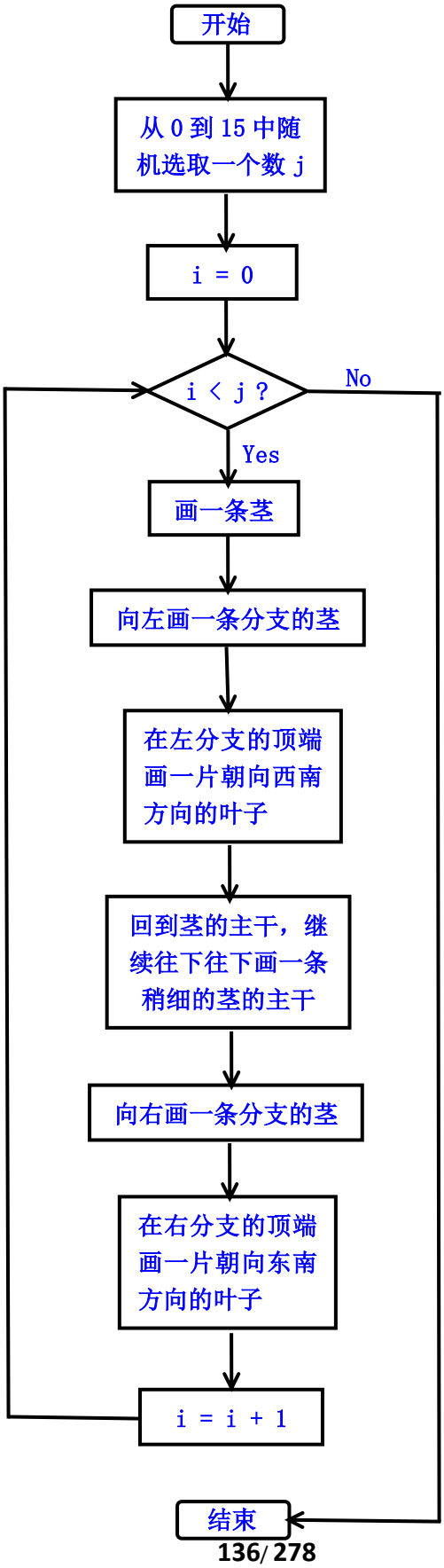


### 11.2.2.2 聚合草


开始
输入 linelen
linelen>20?
Yes
No
画茎的主干
画一条茎
递归，每一次递归画一条茎 + 一片叶子 + 一朵花
结束

### 11.2.2.3 卷耳

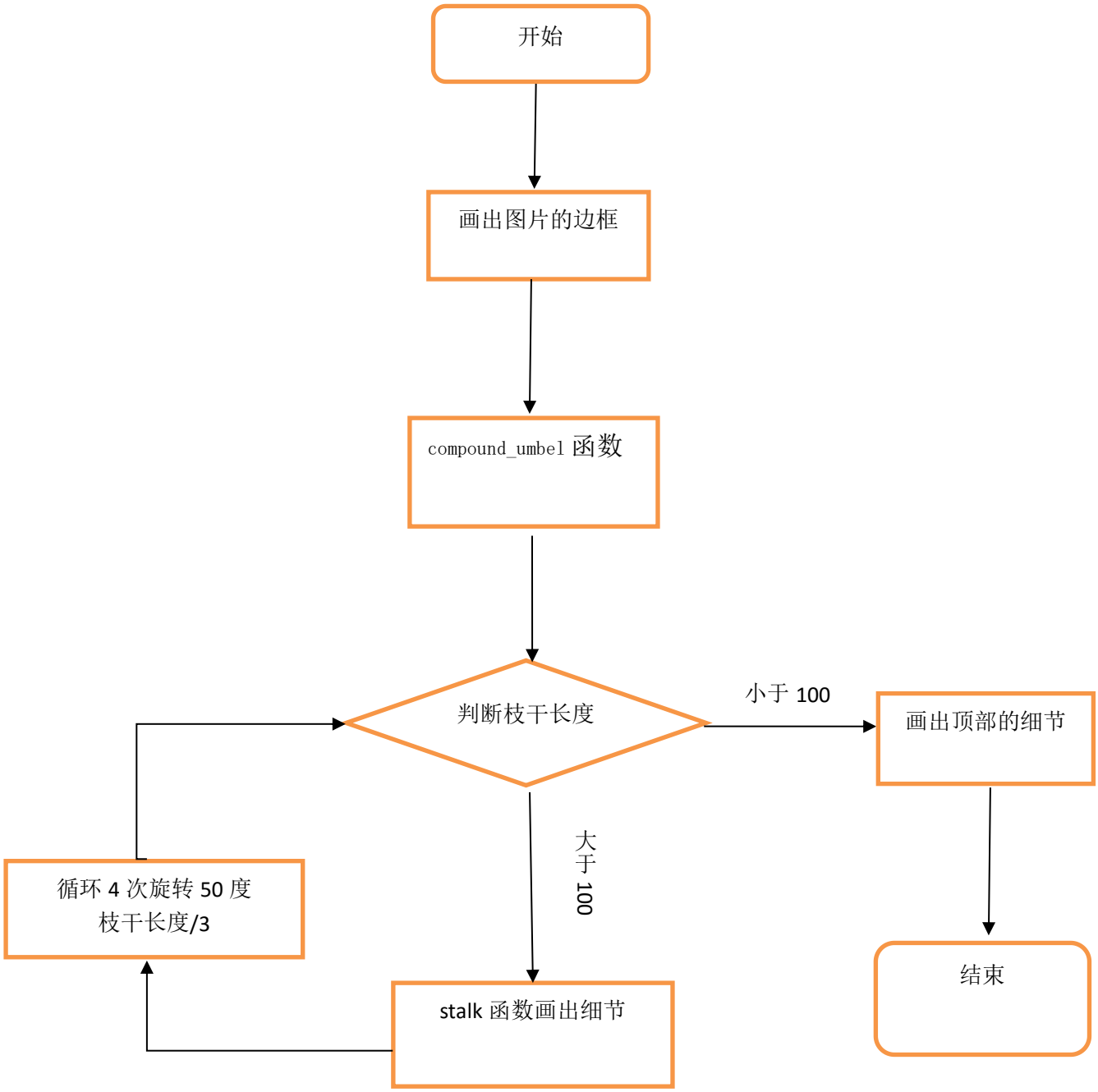

开始
输入 linelen
linelen>20?
Yes
No
递归，每一次递归画一片叶子 + 一条经，最后在每一条茎的末端画上一朵有五瓣花瓣的花
画一朵有五瓣花瓣的花
从花的中间往下画一条茎
在茎的主干下端往下画一条弯曲的茎
结束


开始
从 0 到 15 中随机选取一个数 j
i = 0
i < j ?
No
Yes
画一条茎
向左画一条分支的茎
在左分支的顶端画一片朝向西南方向的叶子
回到茎的主干，继续往下往下画一条稍细的茎的主干
向右画一条分支的茎
在右分支的顶端画一片朝向东南方向的叶子
i = i + 1
结束

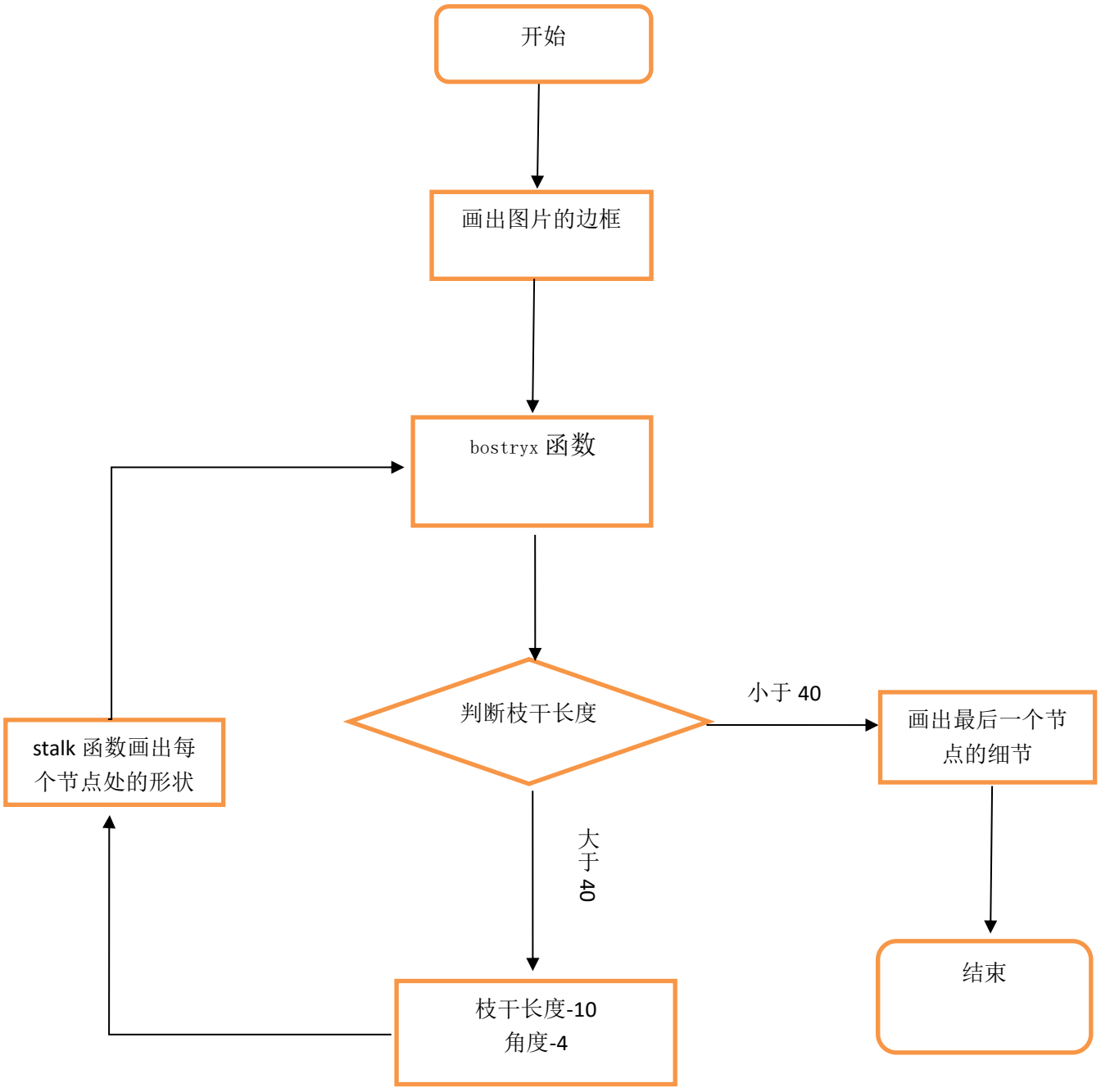


### 11.2.2.4 紫藤

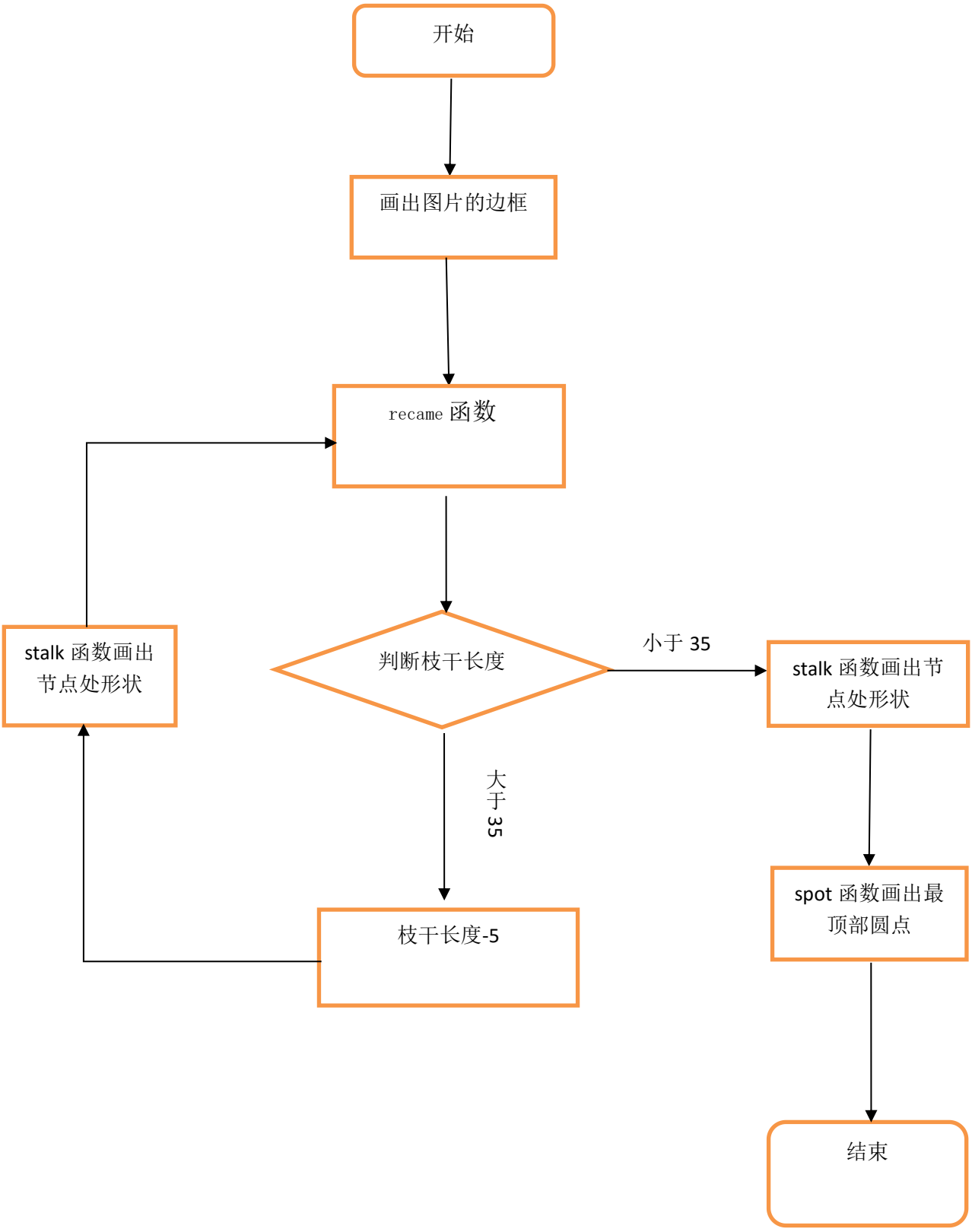
### 11.2.2.5 复伞形花序


开始
画出图片的边框
compound_umbel 函数
判断枝干长度
小于 100
画出顶部的细节
结束
大于 100
stalk 函数画出细节
循环 4 次旋转 50 度
枝干长度/3

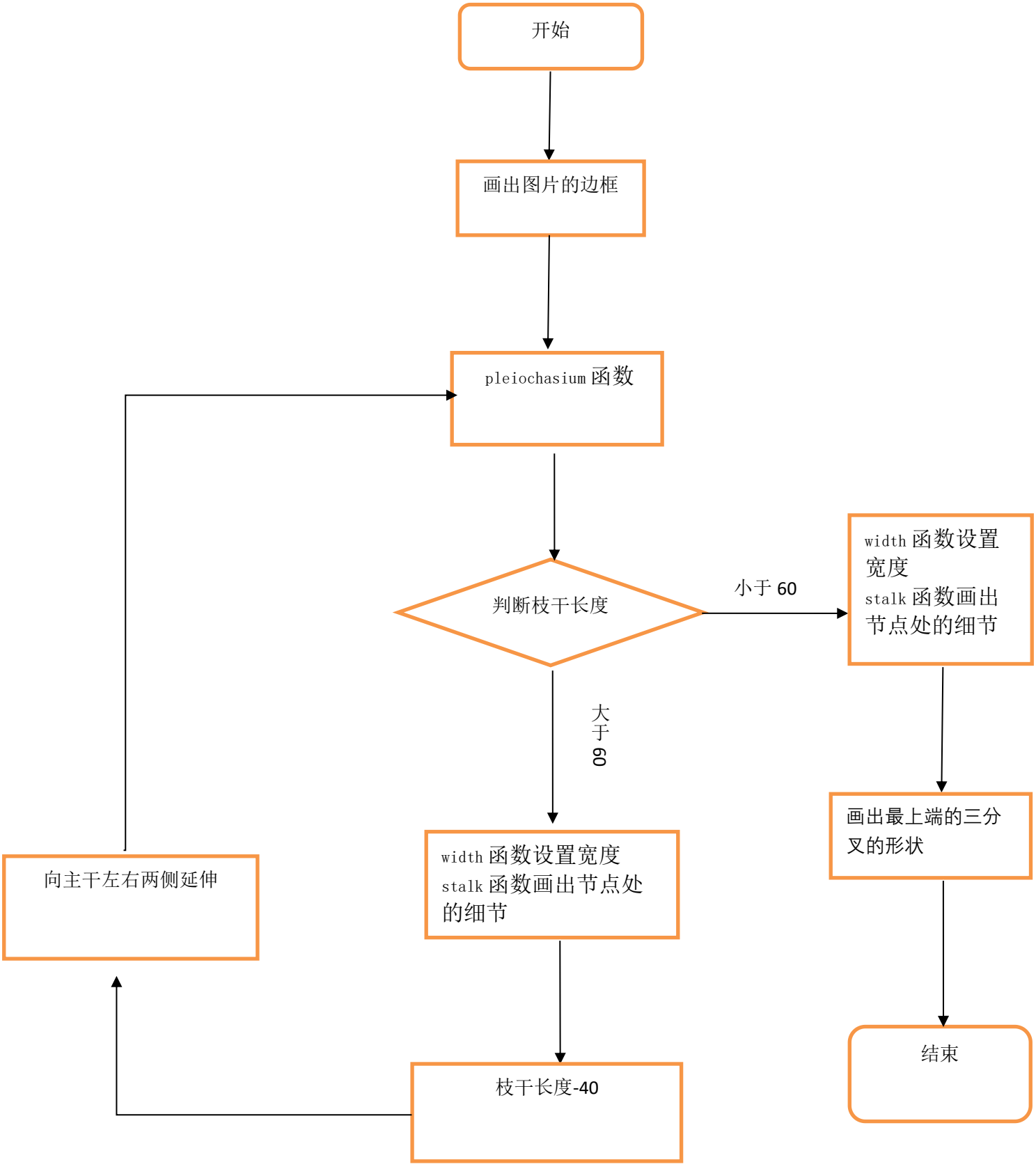
### 11.2.2.6 螺状聚伞花序



### 11.2.2.7 总状花序


开始
画出图片的边框
recame 函数
判断枝干长度
小于 35
大于 35
stalk 函数画出节点处形状
stalk 函数画出节点处形状
枝干长度-5
spot 函数画出最顶部圆点
结束

### 11.2.2.8 二歧聚伞花序



# 11.3实验结果

## 11.3.1实验数据

实验环境说明：

- 硬件配置：（1.6 GHz Inter Core i5/4 GB 1600MHz DDR3）
- 操作系统：（OS X Yosemite/10.10.5）
- Python 版本：（Python 3.5）

## 11.3.2作品描述

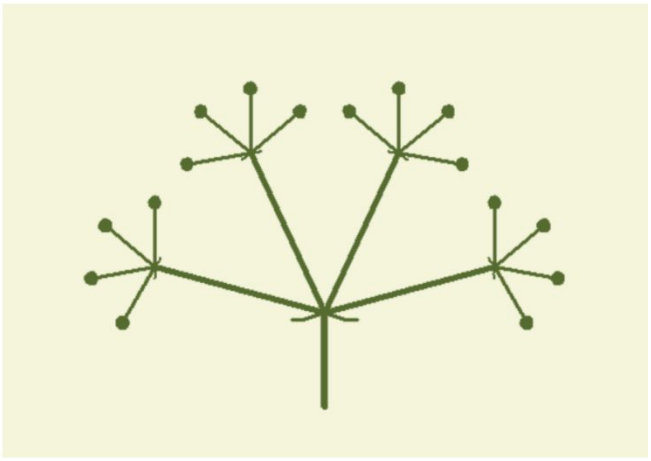
第一部分：植物学家的试验田——一层一层剥开你的枝

（1）总状花序·紫藤

总状花序是无限花序的一种。其特点是花轴不分枝，较长，自下而上依次着生许多有柄小花，各小花花柄等长，开花顺序由下而上，如白菜、紫藤等。从总状花序演化出穗状花序、头状花序，伞形花序、伞房花序；再进一步演变成小穗、肉穗花序、柔荑花序等。

紫藤，别名藤萝、朱藤、黄环。属豆科、紫藤属，一种落叶攀援缠绕性大藤本植物。干皮深灰色，不裂；春季开花，青紫色蝶形花冠，花紫色或深紫色，十分美丽。紫藤为暖带及温带植物，对生长环境的适应性强。产河北以南黄河长江流域及陕西、河南、广西、贵州、云南。

（2）复伞形花序·胡萝卜



复伞形花序是指多回伞形花序，是在每两个伞梗的基础上再生长出一个伞形花序。而伞形花序则是无限花序的一种。其特点是花轴缩短，大多数花着生在花轴顶端，每朵小花的花柄基本等长，因而小花在花轴顶编排列成圆顶形，开花的顺序是由外向内。

胡萝卜，又称红萝卜或甘荀，为野胡萝卜的变种，本变种与原变种区别在于根肉质，长圆锥形，粗肥，呈红色或黄色。二年生草本，高 15-120 厘米。茎单生，全体有白色粗硬毛。基生叶薄膜质，长圆形；叶柄长 3-12 厘米；茎生叶近无柄，有叶鞘。复伞形花序，花序梗长 10-55 厘米，有糙硬毛；总苞有多数苞片，呈叶状，羽状分裂；伞辐多数，结果时外缘的伞辐向内弯曲；小总苞片 5-7，线形；花通常白色，有时带淡红色；花柄不等长，长 3-10 毫米。果实圆卵形，长 3-4 毫米，宽 2 毫米，棱上有白色刺毛。花期 5-7 月。产中国四川、贵州、湖北、江西、安徽、江苏、浙江等省。全国各地广泛栽培。分布于欧洲及东南亚地区。

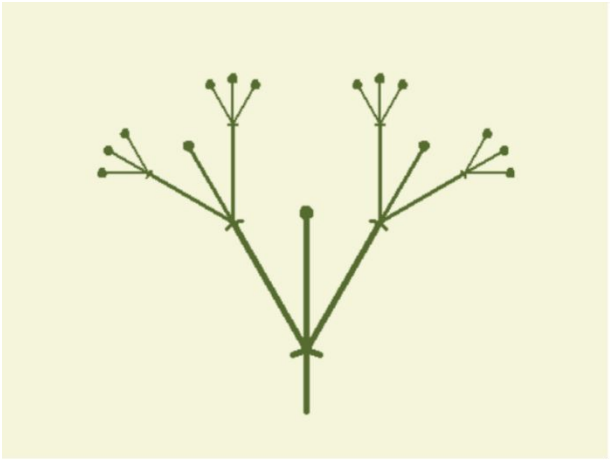
（3）螺状聚伞花序 • 聚合草

聚伞花序即有限花序，其花序轴为合轴分枝，因此花序顶端或中间的花先开，渐渐外面或下面的花开放，或逐级向上开放。聚伞花序如果各次分出的侧枝，都向着一个方向生长，则称螺状聚伞花序。

聚合草，别名爱国草、肥羊草、友益草、友谊草、紫根草、康复力、外来聚合草、西门肺草、紫草根。丛生型多年生草本，高 30-90 厘米，全株被向下稍弧

曲的硬毛和短伏毛。生于山林地带，为典型的中生植物。原产苏联欧洲部分和高加索，中国于 1973 年朝鲜将其作为珍贵礼物送给中国，从此引种栽培。聚合草适应性广，产量高，利用期长，适口性好，是优质高产的畜禽饲料作物，其也有有较高的营养价值，可作药用，也有一定的观赏价值，

（4）二歧聚伞花序 • 卷耳

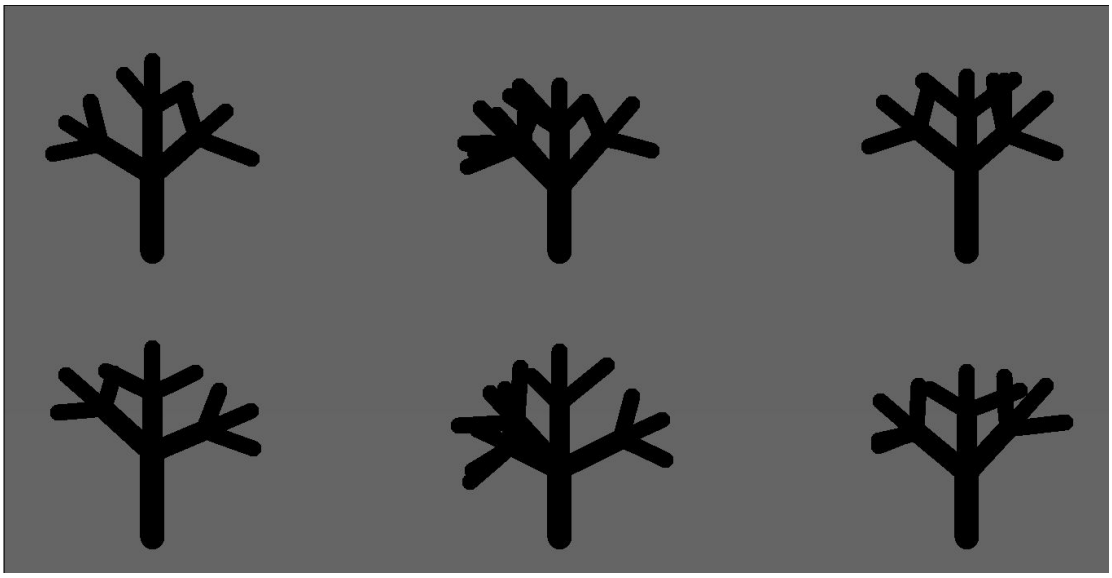


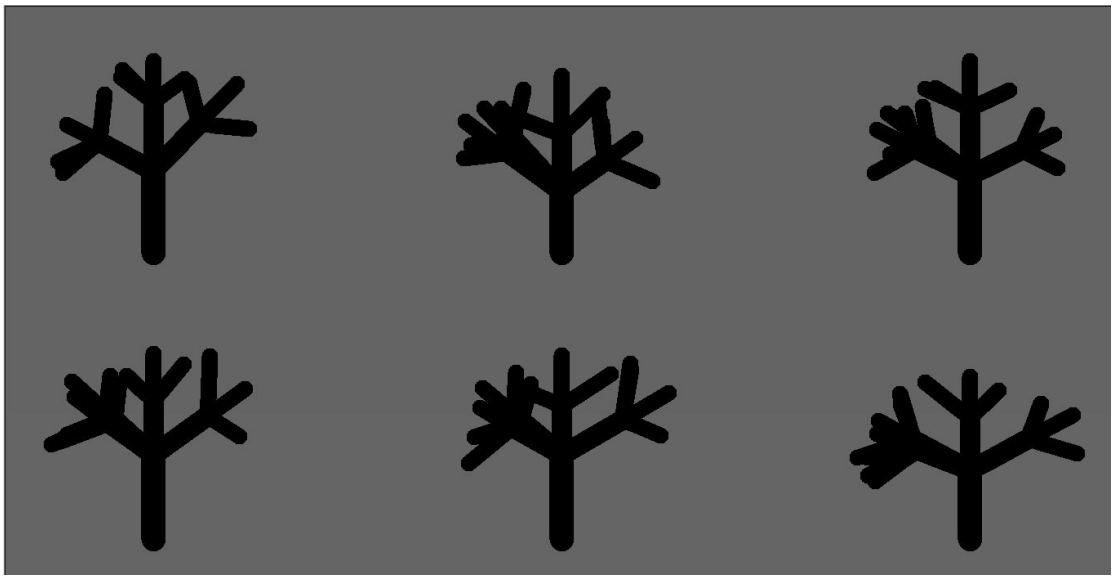
二歧聚伞花序同样是聚伞花序（有限花序）的一种，指的是顶芽成花后，其下左右两侧的侧芽发育成侧枝和花朵，再依次发育成花序。

卷耳，别名：狭叶卷耳、细叶卷耳、无毛卷耳，石竹科、卷耳属多年生疏丛草本，高 10-35 厘米。茎基部匍匐，上部直立，绿色并带淡紫红色，下部被下向的毛，上部混生腺毛。叶片线状披针形或长圆状披针形，顶端急尖，基部楔形，抱茎，被疏长柔毛，叶腋具不育短枝。聚伞花序顶生，花瓣 5，白色，倒卵形，蒴果长圆形，长于宿存萼 1/3，顶端倾斜，10 齿裂；种子肾形，褐色，略扁，具瘤状凸起。花期 5-8 月，果期 7-9 月。

第二部分：蒙德里安的辣椒田——过分的现代抽象艺术

（1）六倍灰色的树

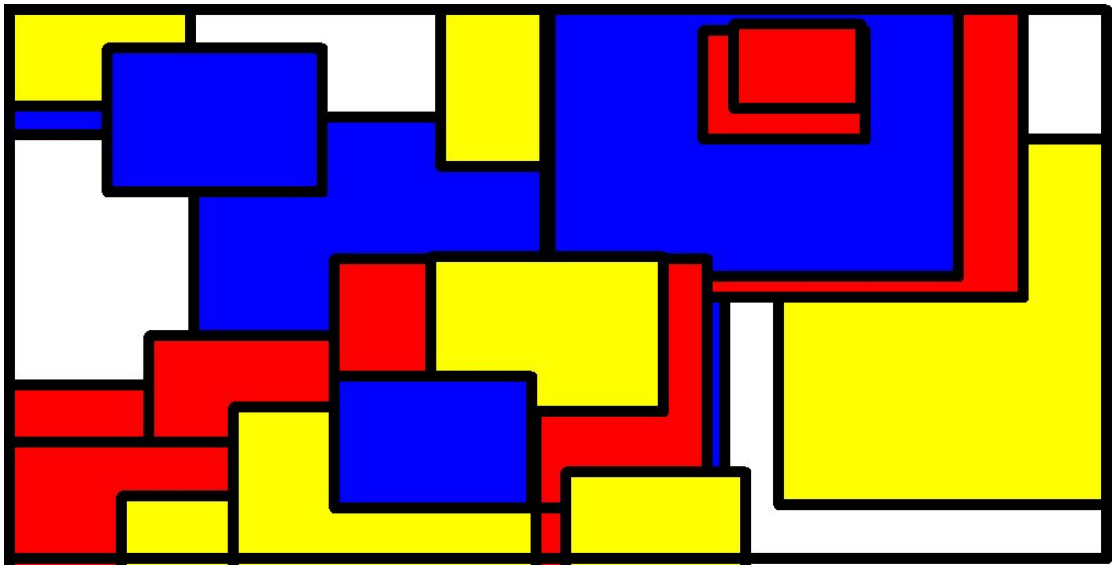


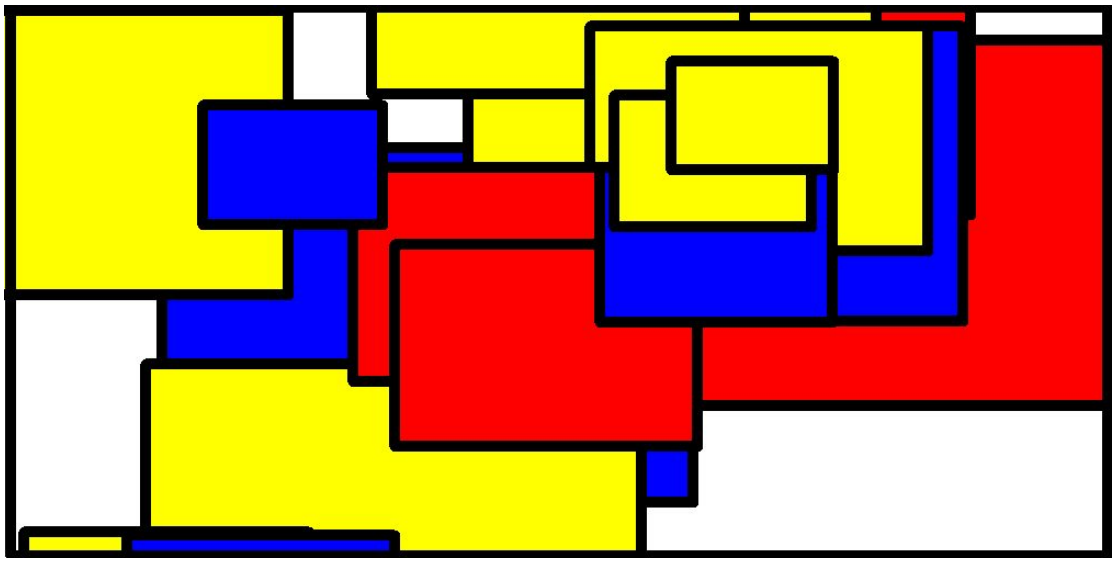


利用随机算法和分形树的重复所创造出的六棵感觉有共同之处但基本又不相似的树。该作品本质上与蒙德里安《灰色的树》作品差距甚远，所希望表达的想法也有很大不同。这六棵树的差别是因为算法中的随机性，然而我们所取的随机范围是很有限的，出现这般大不相同的结果，说明了一种“失之毫厘，谬以千里”的意味。说明很多东西至少稍有偏差，便会产生完全不同的后果。

这种启示不仅仅需要应用在平日里的学术研究中，更重要的是对现在大家专业选择的一种戒告。看似平淡无奇的专业选择，好像差不太多的专业，对你以后的影响是完全不同的。有人认为自己以后又不会从事专业相关职业，这种选择应该影响不大，不过有的时候很多事情就是从你看似没什么的意义的选择中开始的。

（2）百老汇之死





同样这一组图也运用了随机算法和递归思路，在白色的平面上随意的添加大小颜色不同的色块。

《百老汇的爵士乐》这幅作品创作于蒙德里安的后期阶段，此时的他已经迁往纽约。在北美的大地上，他再也看不到第二次世界大战的战火，只有百老汇的歌舞升平，他当时创作的这幅画目的是为了用红黄蓝来表达自己内心的愉悦之情。

然而我们这幅作品在色块上用的更加纷繁复杂，由于随机作用和递归的加入，各种色块相互交叠，整张画面中原来蒙德里安绘画中的规整基本荡然无存。之所以取名《百老汇之死》，目的也是想用这些过度放置的色块提醒大家，类似于蒙德里安画面中规整的有节制的色块相当于适当的消遣，我们这幅画展现的就是过度娱乐后造成的混乱局面。

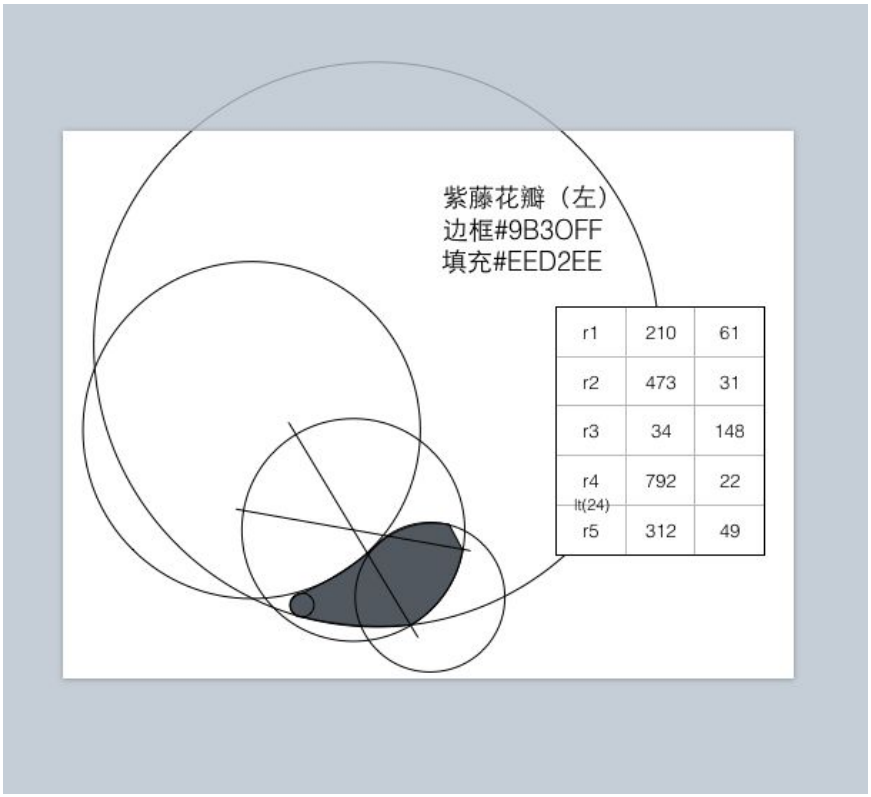
### 11.3.3实现技巧

使用 Keynote 进行图形测量的技巧

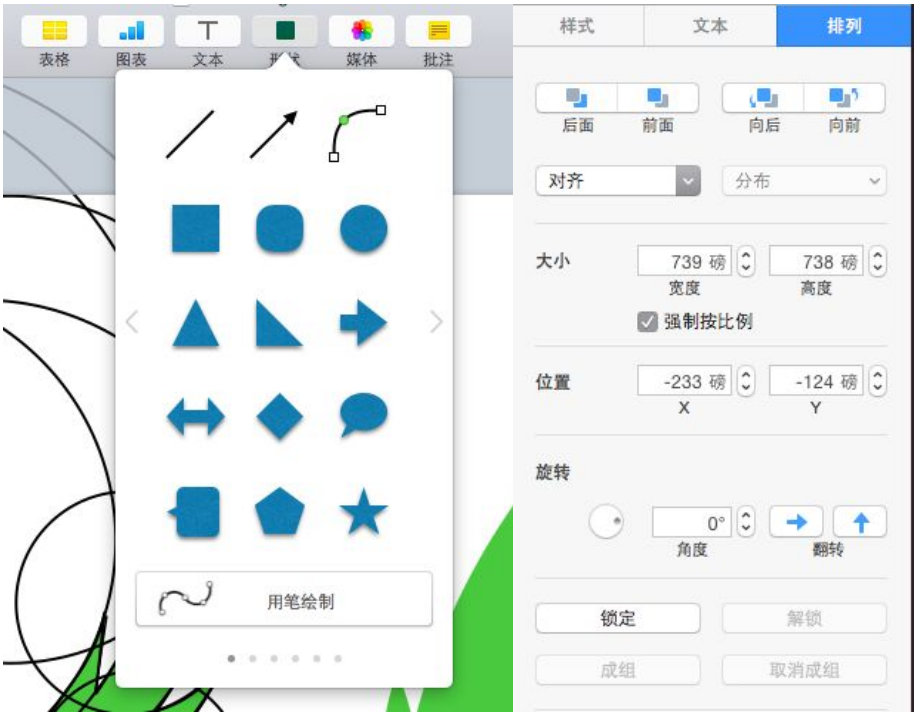
在 turtle 的使用中，想要画出“规整”的图形，步骤中各种命令的执行都依赖于明确的参数。而在刻画诸如植物的叶子等并不一定呈现出明确的几何构成的天然形成的图形时，依据人眼的“估计”或是“凑数”的方法是很困难的，在需要进行对自然图形的刻画时，我们小组利用了 OS 系统中 Keynote 自带的“用笔绘制”功能与对几何图形各种参数的显示功能，尽量用圆与直线去拟合自然的图形。

在测量过程中，首先收集能明显表现出花瓣或树叶形状的真实图片，其后在真实图片之上利用“用笔绘制”，用尽可能少而切合的弧线“复制”出花瓣或树叶，移去图片避免干扰后再利用基础图形尤其是圆重现图形的边界；最后利用两条代表直径的直线的移动，以及 Keynote 自带的图形位置关系与图形自身参数显示功能，得到弧的度数、半径、角度变换等参数。而得到的这些参数，按比例放

大或缩小后便可以应用与 turtle 的命令中。




（对紫藤花瓣的测量）




（左：用笔绘制功能；右：对图形参数的显示）

# 11.4实习过程总结

## 11.4.1分工与合作

### 11.4.1.1 小组分工

周思阳：组长，抽象艺术部分创作
苏克凡：花序实例图创作，汇报展示
凌　坤：花序模式图创作，作品整合
姚媛媛：花序实例图部分报告写作
赵旭炜：花序模式图部分、抽象艺术部分报告写作

### 11.4.1.2 小组交流

组会交流
微信交流

### 11.4.1.3 小组组会

第一次（时间：4 月 4 日 19：00，地点：逸夫二楼 3522 教室）

组会内容：

（1）植物花序部分花序及代表植物选择
（2）抽象艺术部分创意提出
（3）小组分工及作业时间安排表制定
（4）实现算法初步讨论

第二次（时间：4 月 10 日 18:30，地点：第四教学楼 411 教室）

组会内容：
（1）植物花序部分、抽象艺术部分成果初步展示
（2）植物花序部分、抽象艺术部分成果讨论与改进
（3）报告写作事项讨论与布置
（4）中期展示、终期展示事项讨论

## 11.4.2经验与教训

经验：选题从自己所熟悉的领域出发
　　小组讨论时要有明确的分工和时间安排
　　根据小组组员的具体情况合理安排工作
教训：应该多与其他组的同学交流，相互学习一些算法
　　也有助于开拓自己的思路，做出更多更好的作品

### 11.4.3建议与设想

#### 11.4.3.1 作业建议

（1）建议在分组之前由全班同学将有实力的同学票选出来，将其分散到不同的组中，大家先根据这些同学分好组，再在组内自行选出组长，这样既能保证公平，又可以满足一些技术较好的同学不愿意承担管理任务的想法。

（2）视情况取消中期展示，直接用两次课的时间进行终期展示，在终期展示之前小组将成果及报告全部提交，这样先报告的组和后报告的组不会存在不公平的现象，还可以给予每个组更多的机会进行自己作品的展示。

#### 11.4.3.2 后续设想

（1）在已完成的四组植物花序模式图和实例图的基础上，将其他花序的模式图和实例图补充完整，可用于大学生命科学学院和中学生物学奥林匹克竞赛的教学。

（2）对现代抽象艺术作进一步的了解，在画法、算法的综合了解基础上创造出更多的有自身思想的抽象艺术作品。

## 11.5致谢

（1）感谢 Qoo 组的所有成员，在全组同学编程技术不占优势的情况下完成了多幅作品，每位同学都各司其职，做到了最好。

（2）感谢所有参与这一次作业的同学们，大家的中期报告都很精彩，也让我们受益匪浅，学习到了很多东西。

（3）感谢陈斌老师以及各位助教的大力支持和耐心的帮助。

## 11.6参考文献

《植物学（第二版）》 陆时万，徐祥生，沈敏健编著，高等教育出版社

# 12 R 组

# 数据结构与算法课程

## 递归视觉艺术实习作业报告

## （作品名称）

王天宇*、侯传鹏、朱桥宇、龚汐洋、覃港、邓若林

摘要：我们组递归作业创意来源于不断地实验递归与绘画的结合所产生的抽象想法，将递归用于提高作图效率。在这一过程中引用了 turtle 模块和 image 模块，使得作品能够与原图基本一致。

关键字：复原、turtle 模块、image 模块、递归函数

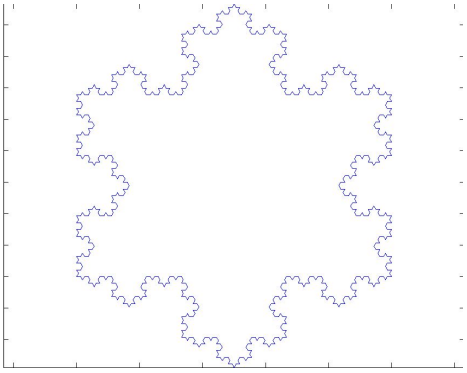
## 12.1创意过程与递归思路

### 12.1.1作品总体介绍

作品是三幅不同的图片，第一幅是卢浮宫的入口，第二幅是一张油画，第三幅则是一个彩蛋。我们的递归思想体现在复原图片的过程中，通过递归的方式使得整张图片被高效而完整地复原出来。在这个作品中我们用到了定义函数、定义宏变量和使用递归迭代自身的方式，并引入了一个新的模块来帮助我们完成这一作品。

### 12.1.2创意来源

递归是一种创作的方式，turtle 加上它可以画出栩栩如生的树木、雪花、鸟巢。它将自然界具有分形性的物体简化为最简单的几何图形的叠加：直线、曲线、圆、正多边形等等，再通过调用自身把不同大小的最简叠加重复输出，从而构成了精美的图案。



我们从树开始，先试着表现具体的事物，把树干粗细变化展现出来，再试着加上叶子、果子、花，从而让这棵树看上去更加完整。接下来，我们并没有去继续给这棵树加上现实中它身上长着的部件，而是试图用更抽象的简单图形去表现树乃至树林。我们期望达到的效果是当你一眼看去第一感觉就是：这是树。然而当你再仔细看时，发现只是简单的几何图形的组合所带给你的树的想象。这颇似毕加索的绘画生涯，前期他力求写实，晚年则总是试着用抽象的方式表现对象的特点。

但是我们又想到，在递归上做文章一定是其他组的重点。如何用递归的方式进行艺术创作，并结合我们自己的想象力来展现自己的作品应当是其他组作品的主要想法。那么，我们有没有可能把递归融入到代码中，并不用它来进行创作，而用它来在 turtle 中复原现实世界呢？我们的作业创意，即从此处来。

### 12.1.3递归思路

在输出图片时，由原来的循环逐点输出改进为递归输出。每次调用自身时输出的长方形区域尺寸长与宽各缩小两个单位，从而达到提高输出效率的结果。

## 12.2程序代码说明

### 12.2.1数据结构及函数说明

**我们主要是将一张图片看成是一个不断缩小的长方形组成，基于这个想法写出如下代码。**

```
from PIL import Image  （使用了 Image 模块来取图片中像素点的 RGB 值）
import turtle
import time          （需要的时候用来延迟 turtle 跑的时间增强节奏感，但由于 turtle 跑得太慢，占的内存过多，所以）
turtle.colormode(255)
t=turtle.Turtle()
s=turtle.Screen()
im=Image.open('8.jpg')
w,h=im.size          （导出图片的长和宽）
print(im.size)
count=0
```

```
w1=w
h1=h
k1=min(w1,h1)    (用于下面递归过程中的终止)

def gubc():          (此处使用了全局变量，count 是用来计算迭代次数，从而使下一部分中的，i，j 能够依照我们设想的那样变化，同样 w1，h1，k1 的不变性会在下文介绍其用处)
    global count
    global w1
    global h1
    global k1

def huixing(w,h,width,p):  (w 是每次运行的长，width 是取像素点时，没几个像素点娶一个，显然 width 决定了图片的清晰度，p 是 pensize)
    #time.sleep(0.01)
    global count
    global w1
    global h1
    global k1

    while count<k1/2:（这是迭代的终止条件，count 记录的是迭代的次数）
        turtle.tracer(1500,20000)   （此处是调节 turtle 的速度，由于代码中）
        #zuo bian di yishuhang
        for i in range(count,count+1,width):
            for j in range(count,h+count,width):   （这里是跑出来一个长方形图片左边的一条边）
                t.setheading(90)
                t1=im.getpixel((i,h1-1-j))  （取出我所需点的 RGB）
                t.up()
                t.goto(i-w1/2,j-h1/2)   （根据相应的坐标关系使取出的 RGB 对应到 turtle 的坐标系中）
                t.down()
                t.color(t1)
                t.begin_fill()
                t.pensize(p)
                t.fd(1)
                t.end_fill()

        for j in range(h+count-1,h+count,width):
            for i in range(0+count,w+count,width):（长方形图片上面的那条边）
                t.setheading(90)
                t1=im.getpixel((i,h1-1-j))
                t.up()
                t.goto(i-w1/2,j-h1/2)
                t.down()
```

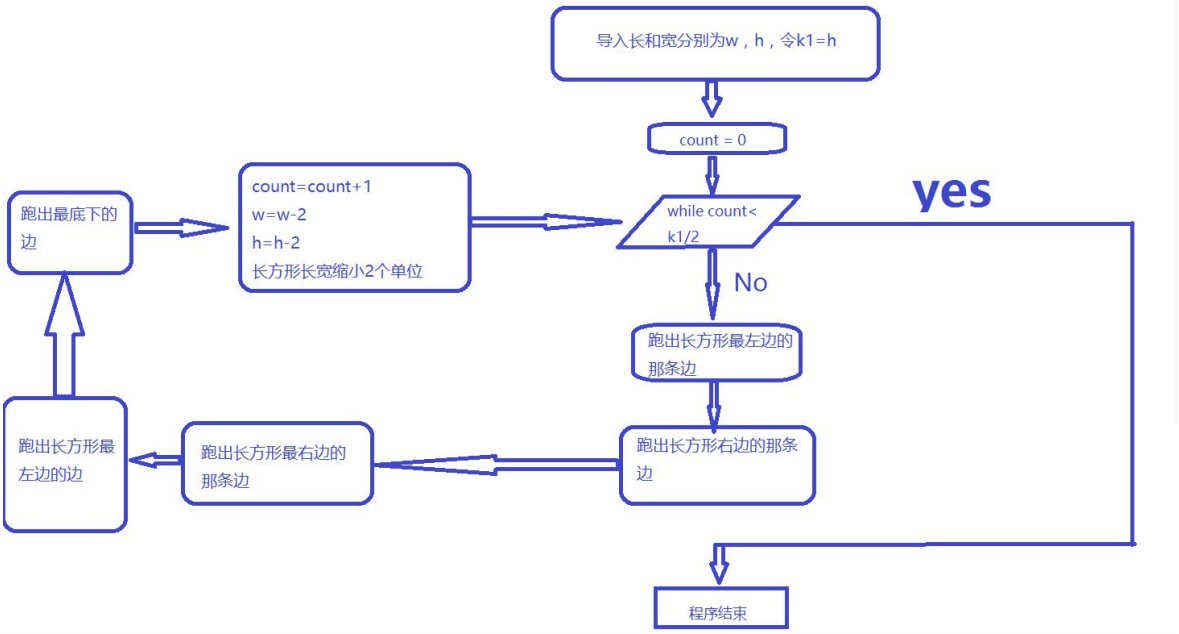
```
                t.color(t1)
                t.begin_fill()
                t.pensize(p)
                t.fd(1)
                t.end_fill()

        for i in range(w-1+count,w+count,width):
            for j in range(h-1+count,-1+count,-width):（右边的那条边）
                t.setheading(90)
                t1=im.getpixel((i,h1-1-j))
                t.up()
                t.goto(i-w1/2,j-h1/2)
                t.down()
                t.color(t1)
                t.begin_fill()
                t.pensize(p)
                t.fd(1)
                t.end_fill()
        #zouhouyihang

        for j in range(count,1+count,width):
            for i in range(w-1+count,-1+count,-width):（底下的那条边）
                t.setheading(90)
                t1=im.getpixel((i,h1-1-j))
                t.up()
                t.goto(i-w1/2,j-h1/2)
                t.down()
                t.color(t1)
                t.begin_fill()
                t.pensize(p)
                t.fd(1)
                t.end_fill()
        turtle.update()
        count=count+1   （记录迭代次数）
        turtle.update()
        huixing(w-2,h-2,1,p)(进行递归)

huixing(w,h,1,2)
s.exitonclick()
```

## 12.2.2程序限制

我们在运行程序的过程中，我们画的点的位置还有此点地颜色数据均存在内存中，影响

程序的运行速度。当随着程序的运行，turtle 所画的图像越来越大的时候，其位置数据就会越庞大，在图像上的点的颜色数也会越来越庞大，占用的内存也会越多，对程序的运行速度的影响就会越剧烈。当对象图像大道一定程度后，画图的点的位置及其颜色的数据会超过一定的限度。在内存占用达到了一定程度后，程序就会停止运行，自然此时就不出想要的结果。因此当我们所绘制的图片尺寸到达一定大小时，程序就会因为内存不够而崩溃。上述就是我们小组所写的程序受限的地方。

### 12.2.3算法流程图




## 12.3实验结果

### 12.3.1实验数据

实验环境说明：

- 硬件配置：Intel(R) Core(TM) i5-3337U CPU @ 1.80GHz / 4GB
- 操作系统：Windows 10
- Python 版本：python 3.4.4

### 12.3.2作品描述

经过运行，该段代码可以完整而高效地复原打开的文件图片，较单纯的引用 image 模块并扫描输出图片，大大提升了输出效率。其应用具有普遍性，对于任意 jpeg 格式的文件均可以用此代码复原。复原度 100%。

我们此次选取了三幅图片，一幅是我们的组号 R，一幅是卢浮宫，最后一幅是紫罗兰油

画。

### 12.3.3实现技巧

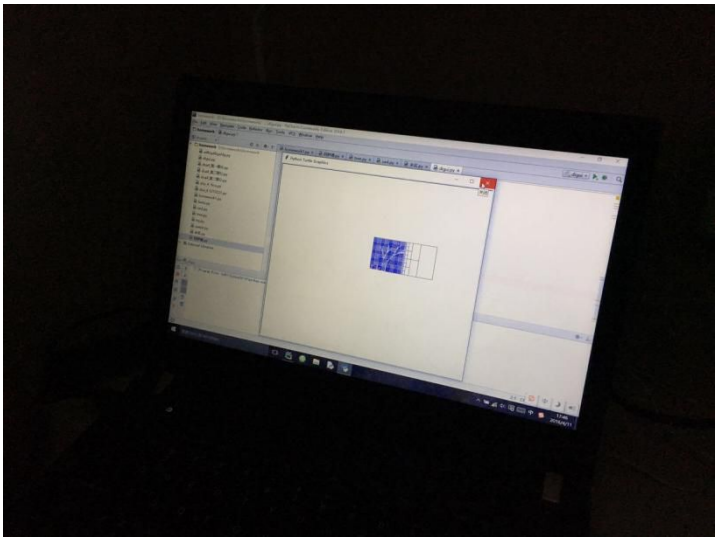
通过引入 image 模块，将一幅图片上的每一个像素的 RGB 信息提取出来，并复原到 turtle 的画布上，通过这种方法，理论上只要有足够的内存和足够长的运行时间，任何图片都能被完整地复原出来。

为了提高输出效率，我们在 image 模块之后加入了递归函数。通过不断地迭代自身实现 trace 的循环，从而能够高效地将图片绘制出来。

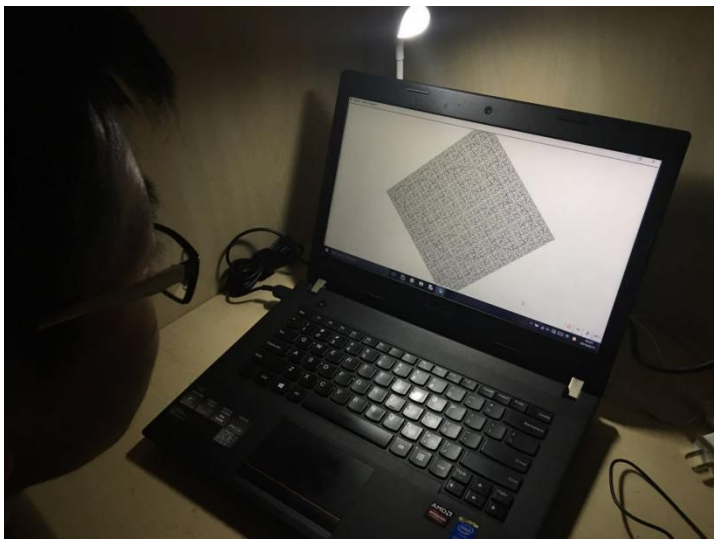
## 12.4实习过程总结

### 12.4.1分工与合作

组长：王天宇
程序员：侯传鹏、朱桥宇、龚汐洋、覃港
报告攥写：邓若林
合作方式：分模块写作代码
交流方式：线下组会交流

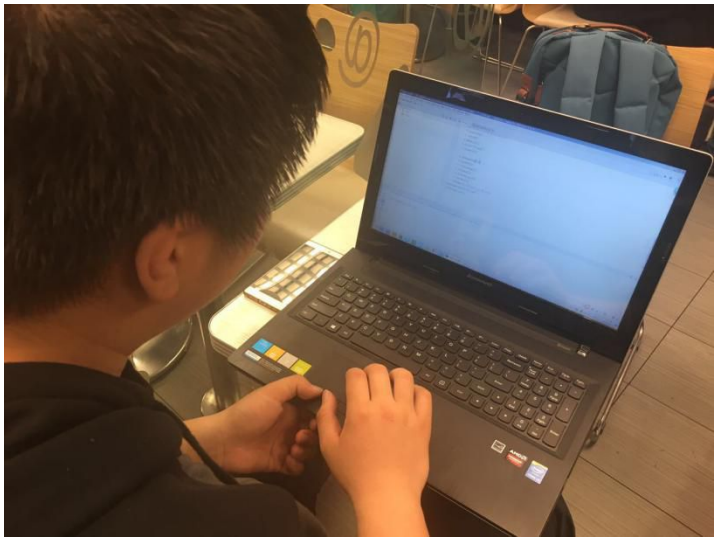


第一次组会时交流递归创意

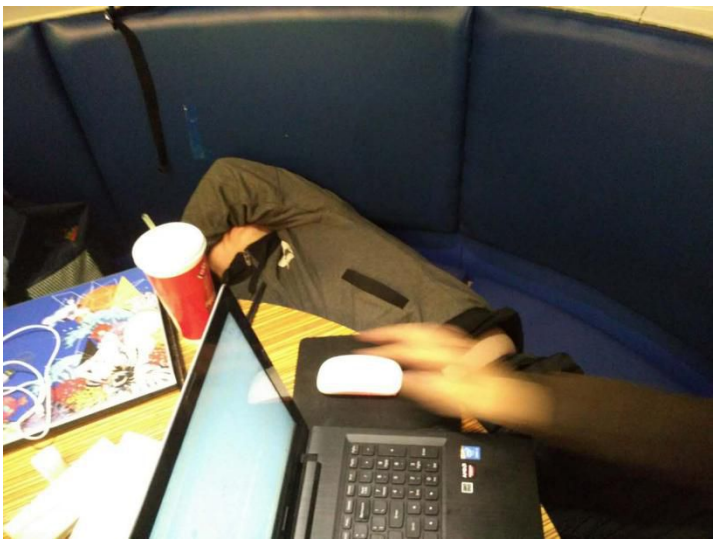


侯传鹏的递归作品

在 KFC 开第二次组会

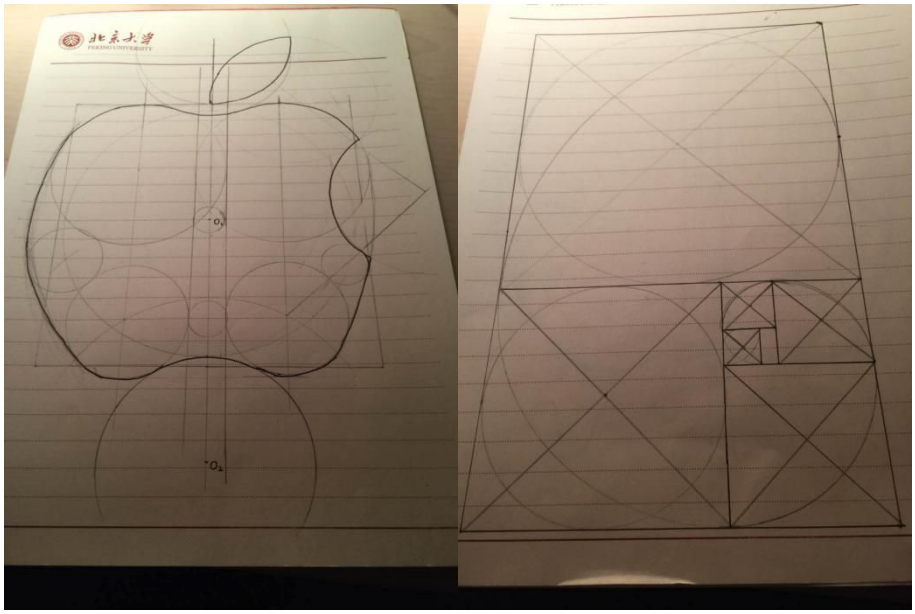


龚汐洋在第二次组会过程中码代码 ing



第三次组会，某人太累了躺下就睡

第四次组会，在 zoo coffee

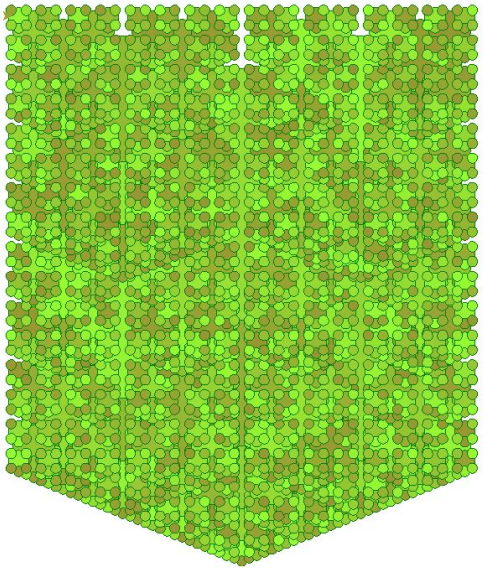


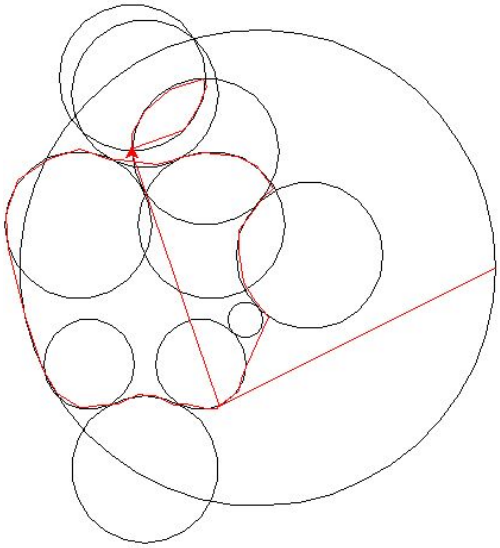
朱桥宇手动绘图

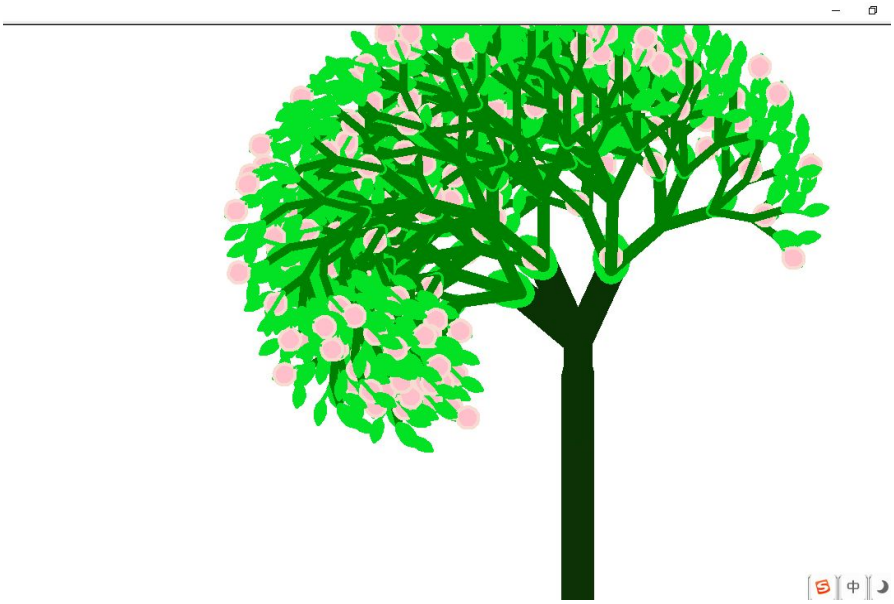
最后一次组会

## 12.4.2经验与教训

在这次的作业过程中，我们曾提出很多的设想，也简单的画了一些递归的图案。在编程的过程中，加深了对递归的理解和运用。比如：在这个递归中，是采用大圆套小圆的方法循环下去。实际的图案很漂亮，我们这为我们的递归的练习打下了基础。而右面的这张苹果的 LOGO 是我们最初的设想,但是画出来后总是不能令人满意，所以就放弃它了。最后，我们使用了 Image 这个模块，通过取像素点结合回型递归，来达到绘图的目的。







在编程的过程中遇到的问题时对 turtle 模块的不了解，只能使用一些最简单的前进后退颜色等。结果使得一些东西写出来后很复杂。还有，在编程的过程中，分工编程再合并不如一起做更有效率，这也是我们日后需要改进的地方。

## 12.4.3建议与设想

建议篇：

1. 希望展示时间能再多给一些，5 分钟的展示时间有点短，一下子就讲完啦。
2. 是否可以考虑把大作业的完成时间跨度再拉长一些？这样可以在我们中期报告的基础上更加完善自己的作品。

设想篇：

1. 这门课真的很有趣也很虐，每节课要消化的东西太多，但是当自己投入到创作中把自己

的想法付诸实际的时候，就会觉得自己能够学以致用，非常满足。尤其是当看到自己的作品时的那种喜悦是无以言表的。所以，希望数算在给我们打基础的同时，也能多给我们一些发挥想象力的作业。

## 12.5致谢

感谢小组每一个成员的付出。
感谢陈斌老师和诸位助教大大。
感谢 zoo coffee、kfc、西部马华。
感谢 pycharm。

## 12.6参考文献

来自该网址的部分模块 https://docs.python.org/2/library/turtle.html#turtle.setx

# 13 S 组

# 数据结构与算法课程

## 递归视觉艺术实习作业报告

## （月夜星空）

S 组：郭孟曦*，孔淑媛，张二禾，郑毅权

摘要：

整幅作品是一副完全由 python 语言来编写的代码实现的原创抽象画作。作品的创意来源于我们对“夜空”的思考，对自然美的追求，对宇宙的无限遐想。在绘制画中的树和波浪线的时候，我们使用了递归算法，通过对树的层数递归以及波浪线的层数递归，实现我们想要的规模。我们的作品使用抽象艺术来表达自然艺术：夜的特点。整幅作品为一副抽象与现实结合，昏暗与光影结合，静与动结合的抽象艺术作品。作品为一副静止的画作，画作本身体现着独特的夜晚静态美，或是对宇宙的崇敬，或是对宁静的追求。波浪线与背景星空形成对比，反应出自然界静中那充满生机的动。

关键字：月夜；星空；树；抽象；python；turtle；递归

## 13.1创意过程与递归思路

### 13.1.1作品总体介绍

我们的作品用抽象为一副表达夜空之美的原创抽象画作品。作品从构图到颜色搭配都为我们自己的创意。整幅作品基于 Python turtle 库以及递归算法实现。其中，画中的主体部分：树以及波浪线的实现均运用了递归。背景和树的部分运用了随机。作品中除了递归所使用的栈以外，没有其他特殊的数据结构。使用算法有：递归算法，随机算法。背景部分运用随机色块渲染方法。作品中的亮点在于构图与颜色的选取。背景部分使用了色块随机渲染的方法，在标准黑（0,0,0）到标准蓝（0,0,255）之间随机生成不同颜色的色块，并随机置于屏幕不同位置。在数量达到一定程度时，可铺满屏幕，并展现出夜色的效果。同样地，我们还随机生成了标准黄（255,255,0）到浅黄（255,153,0）色块，用于表现星光的效果。通过树的部分使用三个随机，使得实现的树更加真实。树的实现在常规实现的基础上增加了许多更体现自然更加随机的因素。比如需要有树干，枝条和叶子的分化以及树干分叉的个数，树干的角度，树干长度的随机性甚至于枝条的三种递变的颜色（以体现层次感），这样就赋予了树的绘制

以极大的丰富性。我们用不规则的曲线来表现层层叠叠的山岳，曲线的起伏和变色又创造出一种波浪的效果，“丘陵的波浪”形成了一种奇幻的效果，曲线之间的小间隙透出了背景的颜色，丰富了画面的视觉效果并带来了立体感。我们运用两个椭圆来表达一种抽象月亮。月亮由两个椭圆构成，外侧椭圆由四阶颜色渐变得到，内侧椭圆由二阶渐变得到。两个椭圆嵌套目的是对月亮抽象，表型一种从地上看天上月亮的典雅感，同时又表现对这原在天边的宇宙中的星体的敬畏。

### 13.1.2创意来源

创意源于我们对“夜”的思考。在创意过程中，郑毅权同学站在美术的角度，考虑到表达树生机勃勃过于俗套，边思考如何运用对自然艺术极致表现的手法：抽象画，来表现树独特的夜晚之美。在创意过程中，我们借鉴了抽象大作，梵高的《星空》。与《星空》相同的是，我们都想运用抽象画的手法来表达夜色独特的美。不同的是《星空》重点体现星星的躁动，而我们的画中，骚动起伏的波浪与平静典雅的树形成对比，内侧月亮和外侧月亮的对比，来体现夜极致的静态美，以及自然这静中暗藏的生机与其迸发的欲望。

### 13.1.3递归思路

1. 作品中树的绘制以递归为主。Tree 函数每层调用只绘制一段 branchlen 长度的某个朝向的树干或枝条，而在顶端又会涉及到两个或三个随机方向的下一层 tree 函数的调用，调用完后 turtle 原路返回，tree 函数执行前后 turtle 位置不变。而控制递归终止的原理在于每一次调用下一层 tree 函数时他的参数 branchlen 长度会逐渐缩小，当 branchlen 缩小到零以后就递归停止，从而画出一棵完整又富含变化的树
2. 画曲线时通过两个参数 t_与 y 分别控制曲线的宽度与位置，每次运行程序 t_和 y 的数值都会发生改变，直到 t_不再符合条件即跳出程序，体现了递归的思想。

## 13.2程序代码说明

### 13.2.1数据结构说明

绘制中并没有采用基础数据结构。树和波浪线的递归实现过程利用到栈的原理。

### 13.2.2函数说明

#### 13.2.2.1 背景相关函数

在实现背景时，我们定义了 background 函数，该函数以三个循环语句为主体。首先我们绘制一个极大的，颜色为“black”的圆来打底。第一个 for 循环作为第一次渲染，颜色由（0,0,0）到（0,0,255）。第二层 for 循环为提高背景亮度，为浅层渲染，颜色由（0,0,128）到（0,0,255）。第三层 for 循环用于制造黄色圆点。在背景实现中，我们引入了位置的随机。具体思路为引

入随机数 count（1,2,3,4），分别在平面内的第一象限，第二象限，第三象限和第四象限来绘制圆点（图一）。

```
if count==1:
    t.setposition(x,y)
if count==2:
    t.setposition(-x,y)
if count==3:
    t.setposition(-x,-y)
if count==4:
    t.setposition(x,-y)
```

**图一.** 背景圆点随机分布的实现。引入随机数 count（1,2,3,4），分别在平面内的第一象限，第二象限，第三象限和第四象限来绘制圆点

## 13.2.2.2 波浪线相关函数：

在形成曲线的程序中，定义了 4 个函数，分布别为两种不同类型的曲线的形成函数与递归调用函数，可以自由地调用函数形成不同颜色不同粗细的曲线。如图，line_type1(a,b,c,d,e,f)是左半部分曲线的生成函数，（a,b,c）（d,e,f）分别表示曲线的两种颜色，在递归中调用 line_type1 可以控制曲线的颜色以及生成位置（图二）。

```
def digui_1(a,b,c,d,e,f,t_,x,y):
    if t_>0:
        t.penup()
        t.goto(x,y)
        line_type1(a,b,c,d,e,f)
        digui_1(a,b,c,d,e,f,t_-1,x,y-4.5)
```

**图二.** 曲线生成递归程序。在递归中调用 line_type1 可以控制曲线的颜色以及生成位置

不规则波浪线的实现函数如图所示（图三），让海龟在每走一步之后进行左转或右转，转动的度数由给定的初始角度 a_0 和转动次数以及两次转动的角度差（图中为 0.01 度）所决定。技术上比较简单，但是为了画出满意的形状需要进行大量调试。需要调整每一次前进的步长、初始角度 a_0、两次转动的角度差以及循环次数。为了保证曲线平滑，在两段曲线衔接的地方要保证 a_0 是连续变化的。

```
a_0=0.8
for i in range(90):
    t.color(c0_1,c0_2,c0_3)
    t.fd(1)
    t.left(a_0)
    a_0=a_0-0.01
```

**图三.** 不规则波浪线的实现。初始角度下的连续变换。

关于曲线的配色，为了搭配画面整体效果，左边的曲线稍微偏暖色系，而右边的曲线偏

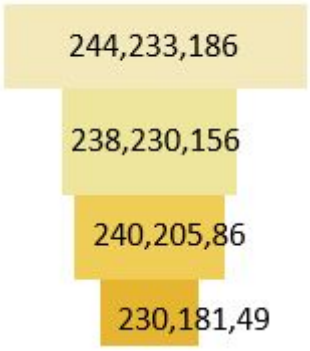
冷色系。为了保证画面的整体感，左边的曲线向右延伸时逐渐过渡到右边曲线的颜色，有了连贯的感觉。

### 13.2.2.3 月亮相关函数：

月亮由两个椭圆组成。两个椭圆的实现方式各不相同。

外侧椭圆的实现：

外侧椭圆为一个中心辐射式渐变椭圆。在实现时，我们先定义了draw_yellow_gradient_line(length)函数。为了提高绘画速度，我们将整个 length 分为 52 份，定义 n=length/52。由于有四阶变色，我们将 52 均匀地分成 4 分，每份有 13n。在 13n 的过程中，通过计算，我们实现了第一阶段：(255,255,255）到（244,233,186）的渐变；第二阶段：(244,233,186）到（238,230,156）的渐变；第三阶段：(238,230,156）到（240,205,86）的渐变；第四阶段：(240,205,86）到（230,181,49）的渐变（图四）。每个渐变过程用 for 循环实现（图五）。有了该函数后，我们定义了 draw_ovalouside(posx,posy,a,b)函数。通过椭圆标准方程：y^2/a^2+x^2/b^2=1，将之改为极坐标形式，得到长度与角度的关系（图六）。再通过 for 循环，环绕 360 度，得到椭圆。




**图四.** 四阶颜色变化图。

```
for j in range(13):
    t.pencolor(color1,color2,color3)
    t.forward(n)
    color3=color3-1.37
    color2=color2-1.69
    color1=color1-0.85
for i in range(13):
    color3=color3-5.46
    color2=color2-0.69
    color1=color1-0.15
    t.pencolor(color1,color2,color3)
    t.forward(n)
for i in range(13):
    color3=color3-5.38
    color2=color2-1.92
    color1=color1+0.15
    t.pencolor(color1,color2,color3)
    t.forward(n)
for i in range(13):
    color3=color3-2.85
    color2=color2+1.07
    color1=color1+0.85
    t.pencolor(color1,color2,color3)
```

**图五.** 四阶渐变的实现。

```
sinangle=math.sin(math.radians(angle))
cosangle=math.cos(math.radians(angle))
length=a*b*math.sqrt(1/((b**2)*(sinangle**2)+(a**2)*(cosangle**2)))
t.left(angle)
t.pendown()
draw_yellowgradient_line(length)
t.right(angle)
```

**图六.** 外侧椭圆的绘制。

内侧椭圆的实现：

定义了 ovalinside(posx,posy,a,b)函数。应用椭圆标准方程：y^2/a^2+x^2/b^2=1，通过 x 的移动来计算得到 y 的值，并用 turtle 进行 forward(y)操作（图七）。过程中，x 每向前移动一个像素点（1），其颜色发生变化，实现从（5,56,185）到（121,223,238）的渐变。

```
y=math.sqrt((1-(x**2)/(b**2))*(a**2))
t.forward(y)
t.backward(y)
```

**图七.** 运用椭圆标准方程绘制内侧椭圆。

### 13.2.2.4 树相关函数

采用 turtle 绘制树的 tree 函数主要参数为 branchlen(枝长)，以及代表是否保证第一次绘制的枝干为二叉的参数 first，其中 branchlen 可以相对控制递归的次数（树的大小），first 值为 True 时，树的基部为二叉，同时其他枝干的分叉为二叉或三叉随机，从而保证树的美感。

```
else:
    a=random.randrange(20,35)
    b=random.randrange(40,55)
    c = random.randrange(1, 3)
    d = random.randrange(20, 30)
    if c==1 or first==True:
        first=False
        t.color('black')
        t.forward(branchlen)
        t.right(a)
        tree(abs(random.randrange(branchlen-max+step,branchlen-min,step)), t,first)
        t.left(b)
        tree(abs(random.randrange(branchlen-max+step,branchlen-min,step)), t,first)
        t.right(b-a)
        t.color('black')
        t.backward(branchlen)
    else:
        first=False
        t.color('black')
        t.forward(branchlen)
        t.right(a)
        tree(abs(random.randrange(branchlen-max,branchlen-min,step)), t,first)
        t.left(b)
        tree(abs(random.randrange(branchlen-max,branchlen-min,step)), t,first)
        t.left(d)
        tree(abs(random.randrange(branchlen-max,branchlen-min,step)),t,first)
        t.right(b - a+d)
        t.color('black')
        t.backward(branchlen)
```

**图八.** 绘制树干。体现随机性和对树形态的控制。

```
if branchlen>0:
    if branchlen <=branch:
        if branchlen<=node1:
            if branchlen < node2:
                t.pensize(3)
                t.color(green1)
                t.forward(20)
                if branchlen < node3:
                    leaf(t, 10, green1)
                t.color(green1)
                t.right(20)
                tree(branchlen - leaflen, t,first)
                t.left(40)
```

**图九.** 通过 branchlen 与 branch，node1，node2，node3 进行比较分类确定树干，枝条，叶子等的绘制。

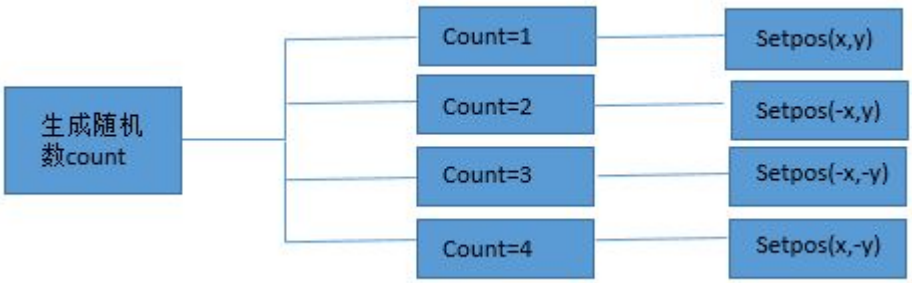
（说明算法中各主要函数的接口、功能、采用的算法等）

## 13.2.3程序限制

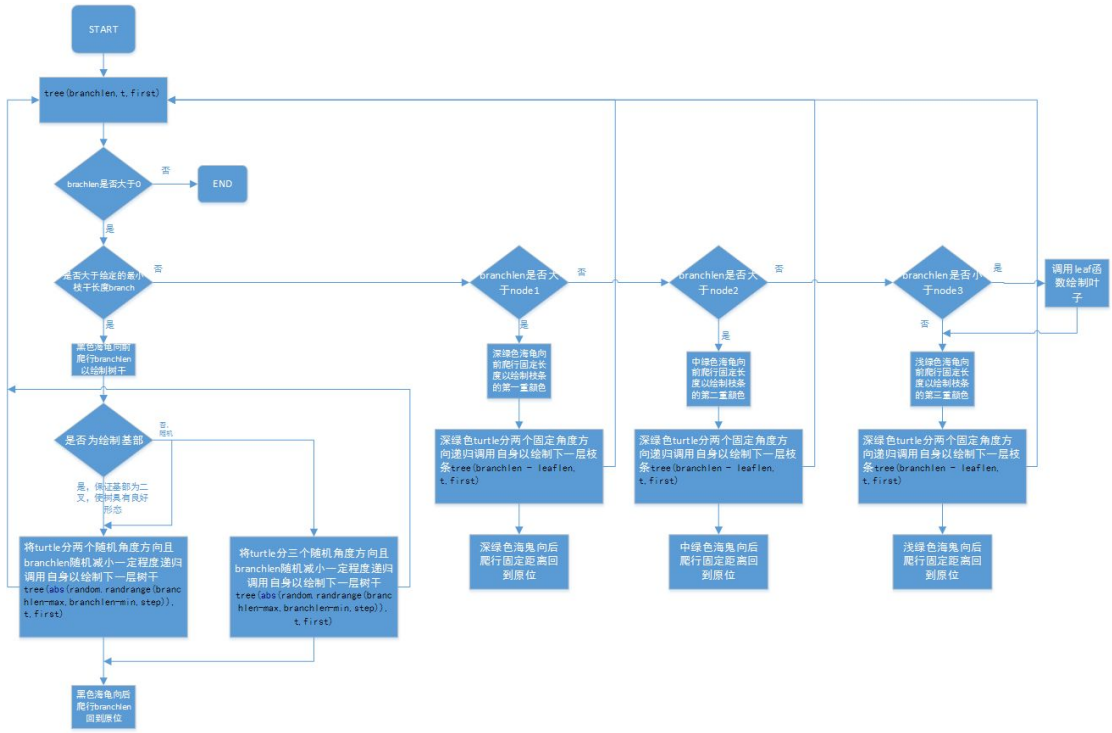
改程序无任何限制条件。（说明在何种极端条件下，程序可能出错）

## 13.2.4算法流程图

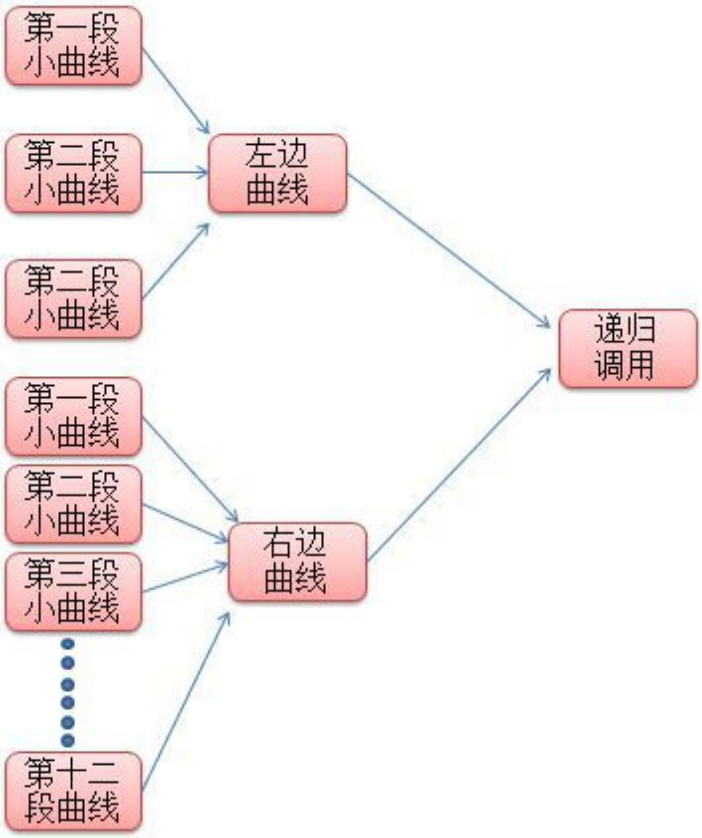
### 13.2.4.1 随机点生成流程图



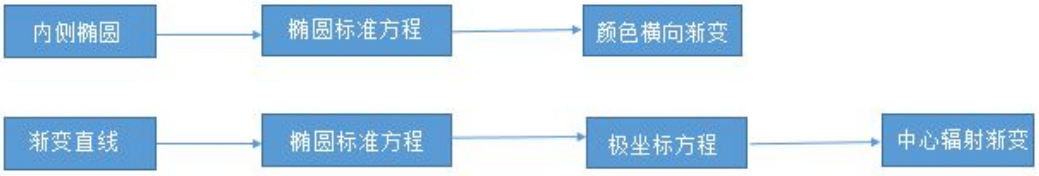

### 13.2.4.2 树的实现流程图



#### 13.2.4.3 波浪线实现流程图




#### 13.2.4.4 月亮实现流程图




## 13.3实验结果

### 13.3.1实验数据

实验环境说明：

- 硬件配置：CPU: Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz　2.50GHz　　RAM: 6.00GB
- 操作系统：Windows 10
- Python 版本：Python 3.4

## 13.3.2作品描述

我们的作品使用抽象艺术来表达自然艺术：夜的特点。整幅作品为一副抽象与现实结合，昏暗与光影结合，静与动结合的抽象艺术作品。作品富有立体感又不过分写实。作品为一副静止的画作，画作本身体现着独特的夜晚静态美，或是对宇宙的崇敬，或是对宁静的追求。波浪线与背景星空形成对比，反应出自然界静中那充满生机的动。本幅作品完全原创。配色和构图是本幅作品最大的亮点。

## 13.3.3实现技巧

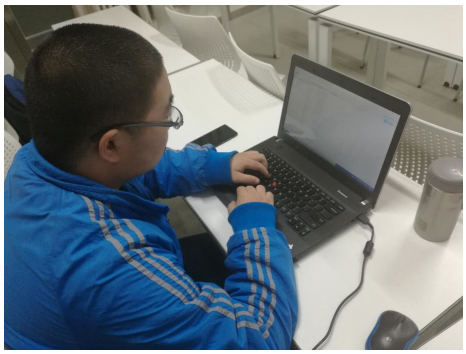
背景部分运用随机色块渲染方法。作品中的亮点在于构图与颜色的选取。背景部分使用了色块随机渲染的方法，在标准黑（0,0,0）到标准蓝（0,0,255）之间随机生成不同颜色的色块，并随机置于屏幕不同位置。在数量达到一定程度时，可铺满屏幕，并展现出夜色的效果。同样地，我们还随机生成了标准黄（255,255,0）到浅黄（255,153,0）色块，用于表现星光的效果。通过树的部分使用三个随机，使得实现的树更加真实。树的实现在常规实现的基础上增加了许多更体现自然更加随机的因素。比如需要有树干，枝条和叶子的分化以及树干分叉的个数，树干的角度，树干长度的随机性甚至于枝条的三种递变的颜色（以体现层次感），这样就赋予了树的绘制以极大的丰富性。我们用不规则的曲线来表现层层叠叠的山岳，曲线的起伏和变色又创造出一种波浪的效果，“丘陵的波浪”形成了一种奇幻的效果，曲线之间的小间隙透出了背景的颜色，丰富了画面的视觉效果并带来了立体感。我们运用两个椭圆来表达一种抽象月亮。月亮由两个椭圆构成，外侧椭圆由四阶颜色渐变得到，内侧椭圆由二阶渐变得到。两个椭圆嵌套目的是对月亮抽象，表型一种从地上看天上月亮的典雅感，同时又表现对这原在天边的宇宙中的星体的敬畏。

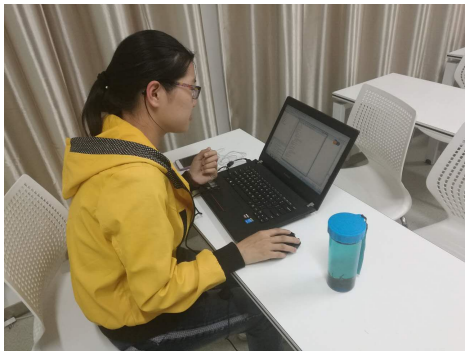
# 13.4实习过程总结

## 13.4.1分工与合作

郭孟曦同学在整个过程中与郑毅权同学一起讨论，确定整幅画的构图和颜色搭配。张二禾同学完成了树的设计和代码编写。郑毅权同学完成了波浪线的设计和代码编写。郭孟曦同学完成了背景的设计和代码编写，并与孔淑媛同学一同设计和编写了月亮部分。

我们小组采用线上（微信）和线下（组会）方式进行交流。前前后后共进行过 10 次组会。每次组会除了讨论画面设计以外，还共同讨论各自在实现过程中遇到的编程问题。






## 13.4.2经验与教训

我们工作由两位精通绘画的同学完成设计，四位同学通力合作，将任务分配，每个人实现画作的不同位置，保证高质高效地完成。由于部分组员编程能力较差，在完成任务过程中遇到许多困难，我们以组会的形式来共同讨论大家在下面实现过程中遇到的困难，对于较难的地方，我们一起解决。在整幅画的配色方面，我们多次实验，多次改进，进过反复对比，敲定最后的颜色搭配。

本次工作中，由于代码是分由不同同学完成，虽然最后都能运行，都能实现预期效果。但是在代码的编写方面，很多同学还存在不规范的情况，导致代码可读性需打一点折扣。这是我们需要改进的地方。

### 13.4.3建议与设想

建议在组队时定下组长后，组员进行随机分配，这样可避免实力很强的同学集体抱团的情况。在本次组队分配中，由于许多同学出现抱团情况，加之一部分同学退课，最后导致我们组的组员只有四人。比绝大部分组都要少两人。这导致了我们在工作过程中，每个人都不得不承担更重的任何，不得不完成更大的工作量。建议今后的分组过程中，能尽量排除期中退课的人数，做到每组人数尽量均匀。

## 13.5致谢

感谢陈斌老师对我们作业的指导！感谢所有助教对我们的帮助！

## 13.6参考文献

1. https://docs.python.org/3.3/library/turtle.html

# 14 T 组

# 数据结构与算法课程

## 递归视觉艺术实习作业报告

## （T 组的“三个代表“）

【小组成员】

李逸飞*、云沿淞、华思博、聂劭质、毋轩琦、廖丝丝

【摘要】

我们小组的作品的创意来源与争吵和进行过程中的不断探索。最初，也就是大家开第一次组会，由于之前就已经收集了一些图片。当然对这些图片的要求是不仅美观，而且要能够运用递归来实行（毕竟这是对我们的主要要求之一）。所以在组会时，我们只是进行选择，经过一番争论，我们最终确定了一幅有星球、云朵、树、草等细节的美丽的图片。当然，后来由于小组的成员的脑洞在创作过程中得到了激发，又产生了两套表情包，即是骑士巡游和生命游戏，当然为了有趣和更加贴近生活，我们还采用了递归算法写了一个画心的算法。在我们的风景画中，基本所有的单个元素都采用了递归的算法，而在两个其他单个的作品中则是有不同的递归想法，比如：骑士巡游的递归想法就是：一条道走到黑，如果此路不通，就往回走，选择其他的路径，最终如果不能走到头，那么返回 0，否则返回 1（实际上并没有返回，只是一个副作用）。按照 Warnsdroff 算法进行排序，以便快速找解。而另外一个作品心则是采用:主要是在心中再画一颗心，就需要不断的在心形区域内调用画心函数，不断地缩小问题的规模，当心充分小的时候，退出递归。具体的关于递归过程和数据结构等的介绍都会在后面的报告中都会有所呈现。由于在我们组具有是同一幅图，都会分成不同的部分让不同成员完成的特点，因此我们的作品中会有很多风格不同的算法结构和递归思想，也会有不同的作品。当然我们的作品在最后的展示中都会是完整的，并会分为几个部分：以风景画作为主体，将生命游戏当作开胃菜，将强烈“膜蛤”的骑士巡游作为杀手锏的一个具有震撼效果的小组思想碰撞的结晶。

【关键字】

风景画；云；山；树；草；蓝色星球；骑士巡游；心；生命游戏；深度优先搜索；Warnsdorff 算法；交互式循环结构；pillow；tkinter 导出。

## 14.1创意过程与递归思路

### 14.1.1作品总体介绍

由于我们小组的作品分为三个不同的部分，并且它们的产生过程也是不同的。

**风景画：**

由云，河流，星球，树，草，星球以及大陆等元素组成，并且各自具有不同的算法和思

路。河流：真实的河流不是笔直的，严格说来处处弯曲，如何做到这一点呢，让它每次只向前走一点，然后随机转一个角度，再走一点，周而复始，宏观上看来河流便是弯弯曲曲的了。然而问题来了，分叉处如何让 turtle 返回起始点呢？每次转的角度都压到栈里，每一段河流存一个栈，整体作为一个列表，这样返回时，只需向相反方向转栈中所存角度即可；树：前人画树，树干的每一段基本都是直的，而且一样粗。我就想，既然一棵树可以分为很多枝干，那么一根枝干也可以细分。反复调用自身从而增长。在分支处调用自身从而分叉。星球：通过画一系列椭圆，调整颜色与短轴的关系，从外向内画，可以画出渐变的效果。并受此启发，在主函数中用类似于这个的办法画一系列同心圆，颜色渐变，产生星球外的光晕。然后介绍改良版的每一小段河流或大陆边界，主要考虑到一定要在星球的内部不能跑到星球圆外去，因此加上了一些判断语句，当它离圆边界过近时将方向变为指向圆心，并前进一小段然后继续原来的弯曲的行进，离下边界过近时会向上行进，并将方向调为上，等等。

**骑士巡游：**

用栈的数据结构来记录和寻找路径，利用深度优先搜索(dfs)以及 Warnsdorff 算法设置权重以达到快速寻找哈密顿圈的目的。

主要的技巧是 Warnsdorff 算法，据测试，对 6*6 的情形，无需回溯即可找到解。在 100*100 的情况下，存在需要回溯的情况。这说明 Warnsdroff 算法是一个相当有效的算法，大大减少了搜索的节点数。

深度优先搜索即用到递归，深度优先搜索本身就是靠自己不断的调用自己，不断地将节点压入栈中，如果需要回溯则不断弹出来实现。可以通过设置权重和 IDA*等算法进行搜索时的优化。是一种图的遍历算法，当然，也非常适合隐式图搜索。

**心：**

如字面意思，就是画了一颗比较有立体感的心。

递归方面就是通过不断的调用自己，而在每次调用的时候改变大小和颜色，以达到立体感的增强。数据结构就是单纯的栈，算法是单纯的拟合，拟合那个心需要一定的技巧与想象，以及对基本图形，基本函数，基本性质的熟悉

**生命游戏：**

生命游戏它完完全全是个递归，每次都调用上一个时刻各方格的生存状态来决定这个时刻的各方格的生存状态，它就是一个不断调用自己的过程

## 14.1.2创意来源

**风景画：**

第一次组会下来，我们就选定了我们所要表现的品。本来大家是各有想法的，有的建体动态，有的希望能做一幅与众不同的图•••••但是最后由于组员说了，我们也可以就做简单的风景画，但是与众不同的是，我们可以把它做得精细，做出视觉冲击，这个管带你在一番争论下，得到了大家的认可，所以风景画产生并没有太多的故事，而是由于我们一颗颗希望把普通做到极致的精细与美丽，希望在无数的“撞衫”的情况下，依旧能够更胜一筹的决心。

**生命游戏：**

高考之前在网上了解过 0 人游戏，知道了生命游戏这个名字。而且生命游戏实质上是一个 R×R->R×R 的一个映射，理应算是递归。生命游戏的可交互性强，乐趣多，作为大作业的一部分很合适。

第一次见是看到组长在网上找到一些关于生命游戏成果的视频，具有极大的震撼之感，兴趣自然不必说。当即向组长申请做这个作为我们风景画的附属品，为其锦上添花。、

**骑士巡游：**

实际上和我个人经历有关。我感觉我的人生就是在使用 Warnsdroff 算法，我要去为自己正名，Warnsdroff 算法和我的选择是有意义的。我对国际象棋有相当浓厚的兴趣，是国家一级运动员，对棋盘上的哈密顿圈有一定的研究:彻底研究清楚了 4*n 的哈密顿圈存在性问题，简单构造了 3*x(x<10)的哈密顿圈存在性或者证明不存在，所以这个程序非常的自然，手到擒来。

**心：**

这是组长一个人的创意，某天想到了在力学课上舒幼生老师讲过的在黑板上画黑板的故事，他的结论是黑板上不能画黑板。其实这个故事本身很有意思，细细想来，集合论的坍塌或是重建都和这个故事背后引出的例子相关：A 是所有满足 X∈ X 的集合，问 A 是不是∈ A。这个例子通过不断的调用自己，自相似，导致了矛盾，发现 A 和 A 的关系是无法确定的，给集合论带来了巨大的冲击。和理发师悖论有异曲同工之妙！

但是通过多次的试验，发现在一张图上画自己非常难以画出来美感，所以最后我选择了一个简单的图形以便于产生美感，而且心贴近生活，在各种场合都有所运用和象征，所以不妨做一颗立体效果的心。

## 14.1.3递归思路

**骑士巡游：**

递归说明：

VIS 数组表示走过的格子。

```
def availableMove(x, y):
    AvailableMove = list()
    for i in DIR:
        if 1 <= x + i[0] <= ROW and 1 <= y + i[1] <= COL and VIS[x + i[0]][y + i[1]] != 1:
            AvailableMove.append([x + i[0], y + i[1]])
    return AvailableMove
```

这个函数找到了所有可以走的格子，压到栈中，最后返回之

```
def NAvailableMove(x, y):
    return len(availableMove(x, y))
```

返回可以走的格子的个数，方便作为权重来估计优先走哪个格子

```
def orderedAvailableMove(x, y):
    Temp1 = availableMove(x, y)
    for s in Temp1:
        s.append(NAvailableMove(s[0], s[1]))
    ordered = list()
    for i in range(9):
        for s in Temp1:
            if i == s[2]:
                ordered.append(s)
    return ordered
```

返回按照权重排序的可选方案

```
def dfs(depth, x, y):
```

```
global FINDSOLUTION
if depth >= COL * ROW:
    VIS[x][y] = 1
    FINDSOLUTION = True
VIS[x][y] = 1
stackDfs.append((x, y))
for d in orderedAvailableMove(x, y):
    if FINDSOLUTION == True:
        break
    dfs(depth + 1, d[0], d[1])
if FINDSOLUTION == True:
    return
else:
    VIS[x][y] = 0
    stackDfs.pop()
```

dfs 主要递归函数:就是一条道走到黑，如果此路不通，就往回走，选择其他的路径，最终如果不能走到头，那么返回 0，否则返回 1（实际上并没有返回，只是一个副作用）。按照 Warnsdroff 算法进行排序，以便快速找解。

**心：**

主要是在心中再画一颗心，就需要不断的在心形区域内调用画心函数，不断地缩小问题的规模，当心充分小的时候，退出递归。和在黑板上画黑板有异曲同工之妙。

```
def drawHeart(scale,Color,cx,cy):
    SCALE=scale
    x=int(100*SCALE)
    y=int(100*SCALE)
    centerx=cx
    centery=cy
    for i in range(x):
        for j in range(y):
            s=centerx-x//2+i
            t=centery-y//2+j
            if s<centerx :
                if
(s-centerx)**2+(-centery+t)**2-(-centerx+s)*(-centery+t)<=int(1000*SCALE**2) :
                    A.putpixel([s,t],Color)
            else :
                if
(-centerx+s)**2+(-centery+t)**2+(-centerx+s)*(-centery+t)<=int(1000*SCALE**2) :
                    A.  putpixel([s,t],Color)
```

按照某种方式生成递归序列，将递归序列代入 scale 变量，用 scale 变量计算出颜色的 RGB 值，最后即可由特殊函数来作图。

**树：**

前人画树，树干的每一段基本都是直的，而且一样粗。既然一棵树可以分为很多枝干，那么一根枝干也可以细分。反复调用自身从而增长。在分支处调用自身从而分叉。代码：

```
def GrowTree(turtle,size,height,position,heading,color):
    '''树的函数，size 粗细，height 高度，position 底部坐标，heading 起始朝向一般为 90（向上），color 颜色，可以写 rgb 的元组函数名懒得改了'''
    turtle.penup()
    turtle.setpos(position)
    turtle.setheading(heading)
    turtle.pendown()
    turtle.pencolor(color)
    for i in range(20):#增长
        size=size*f3(i)#粗细变化函数可换，f1,f2,f3 差不多
        turtle.pensize(size)
        turtle.forward(height*(random.randrange(60,141)/100)/200)
        if i%7==0 and i>7:#树干开始弯弯曲曲了
            if random.randrange(0,2)==0:
                turtle.left(random.randrange(5,15))
            else:
                turtle.right(random.randrange(5,15))
    position=turtle.position()
    heading=turtle.heading()
    if random.randrange(0,10)<=5 and size>=1:#分支
        if random.randrange(0,1)==0:
            heading+random.randrange(0,30)
        else:
            heading-random.randrange(0,30)
        if 30<heading<150:#稍稍限制一下方向
            GrowTree(turtle,size,height,position,heading,color)
        if random.randrange(0,1)==0:#长树叶啦，Leaf 也有三种可以选
            Leaf1(turtle,1,9,turtle.position(),random.randrange(0,360),'black')
    while size>=random.randrange(1,3):#递归，终止时的粗细范围，可调，1 为下限
        return GrowTree(turtle,size,height,position,heading,color)#调用自己继续长长
```

**河流**

探索过程中产生的河流却是用到了递归，这是组员华思博自己想到的一个算法：

```
def fore(n):#河流一小段的函数
    s.append(Stack())
    for i in range(n):
        t.forward(10)
        an=random.randint(-20,20)
        s[-1].push(an)
        t.left(an)
```

真实的河流不是笔直的，严格说来处处弯曲，如何做到这一点呢，我想到让它每次只向前走一点，然后随机转一个角度，再走一点，周而复始，宏观上看来河流便是弯弯曲曲的了。然而问题来了，分叉处我如何让 turtle 返回起始点呢？我将每次转的角度都压到栈里，每一段河流存一个栈，整体作为一个列表，这样返回时，只需向相反方向转栈中所存角度即可，于是有了返回函数：

```
def backi():
    for i in range(s[-1].size()):#河流返回函数
        t.right(s[-1].pop())
        t.backward(10)
```

于是以这两个函数为基础，河流函数产生了：

```
def drawsn(length,width):#画河流函数
    t.pensize(width)
    fore(length)
    if (length>3)&(width>1):#递归边界
        r=random.randint(1,3)#分叉数量（随机），下一段河流的#长度，宽度由分叉数量及上一段河流决定
        if r==1:
            drawsn(length-length//4,width//1.2)
        elif r==2:
            t.left(30)
            drawsn(length-length//3,width//1.5)#这都是递归
            t.right(60)
            drawsn(length-length//3,width//1.5)
            t.left(30)
        elif r==3:
            t.left(35)
            drawsn(length-length//3,width//2)
            t.right(35)
            drawsn(length-length//3,width//2)
            t.right(35)
            drawsn(length-length//3,width//2)
            t.left(35)
        backi()
        s.pop()
    else:
        backi()
        s.pop()
```

然而，由于星球本身比较小，河流的宽度如果太宽在星球上会极度失真，因此，最终这个河流的递归算法并没有被应用，但是那个我自创的 fore 函数却保留了下来，并进一步改进，为之后我画的大陆做了极大贡献。

**生命游戏**

生命游戏的规则是这样的：

1. 当前细胞（方格）为存活状态时，当周围低于2个（不包含2个）存活细胞时，该细胞变成死亡状态。（模拟生命数量稀少）

2. 当前细胞为存活状态时，当周围有2个或3个存活细胞时，该细胞保持原样。

3. 当前细胞为存活状态时，当周围有3个以上的存活细胞时，该细胞变成死亡状态。（模拟生命数量过多）

4. 当前细胞为死亡状态时，当周围有3个存活细胞时，该细胞变成存活状态。（模拟繁殖）

一旦给定了初始条件，在规则的作用下这一时刻的状态便会决定下一时刻的状态，一帧一帧演化下去。生命游戏它完完全全是个递归，每次都调用上一个时刻各方格的生存状态来决定这个时刻的各方格的生存状态，它就是一个不断调用自己的过程，具体的代码等等详见后面。

# 14.2程序代码说明

## 14.2.1数据结构说明

①递归程序无可避免的利用了栈，其他的高级数据结构并不在本次作业的考虑范围内。
②绘画过程中我用得最多的就是循环结构和判断结构，还有一个可能不算结构了，就是给每个函数都加上缩放系数，以便最后整个风景画的整合。

## 14.2.2函数说明

**草：**

```
def GrassLine(t,position,size,height,r,g,b)
#草里面一根线的函数，size 为笔的粗细，height 是长度，position 是底部坐标，rgb 写 0-255 的
def SingleGrass(t,position,size,head,height,r,g,b)
#一根草的函数，head 是朝向，size 是笔的粗细，height 为高度
def GrassQueue(t,size,height,r,g,b,num,y)
#一行草的函数，y 是纵坐标，t 是海龟,num 是数量
```

上面三个都是画草的，参数都差不多，思想就是把草拆解成多个函数。因为草的形状过于简单，没有用递归。
画树的函数已经在前面展示过了，还有一些画树叶的函数，无非是把树叶的线条分解成简单的几何图形，就不展示了。

**骑士巡游：**

递归函数如上已给出，介绍非递归部分。

```
def Hint(x, y, GRID):
    x+=1;y+=1
    global stackDfs
    global FINDSOLUTION
    cnt=0
    FINDSOLUTION = False
    for i in range(COL):
        for j in range(ROW):
            VIS[i+1][j+1]=GRID[i,j]
            if VIS[i+1][j+1]==1 :
                cnt+=1
    stackDfs = list()
    dfs(cnt, x, y)
```

```
    if len(stackDfs) == 0:
        return (-1, -1)
    elif len(stackDfs) >=2:
        return stackDfs[1]
    else :
        return (-2,-2)
```

Hint 函数，顾名思义，就是给出提示功能。通过 dfs 找解，找到解立即返回，无解则返回无解，游戏结束则返回游戏结束。

```
def drawGrid():
    for x in range(0, WINDOWWIDTH+CELLSIZE, CELLSIZE):    # draw vertical lines
        pygame.draw.line(DISPLAYSURF, DARKGRAY, (x, 0), (x, WINDOWHEIGHT))
    for y in range(0, WINDOWHEIGHT+CELLSIZE, CELLSIZE):    # draw horizontal lines
        pygame.draw.line(DISPLAYSURF, DARKGRAY, (0, y), (WINDOWWIDTH, y))
```

画出黑色的格子

```
def showText(string,x,y,size=60):
    font=pygame.font.SysFont('ActionIsShaded',size)
    text_surface=font.render(string,True,(0,0,255))
    DISPLAYSURF.blit(text_surface,(x,y))
```

在指定位置输出给定大小的字符串

```
def clearTheText(x1,y1,x2,y2):
    pygame.draw.rect(DISPLAYSURF,[255,255,255],[x1,y1,x2,y2],0)
```

直接清除一个矩形区域的所有内容，无返回值

```
def colourGrid(item, GRID, COLOR):
    x = item[0]
    y = item[1]
    y = y * CELLSIZE    # translates array into grid size
    x = x * CELLSIZE    # translates array into grid size
    # p,q,r=random.randint(0,255),random.randint(0,255),random.randint(0,255)
    if GRID[item] == 0:
        pygame.draw.rect(DISPLAYSURF, WHITE, (x, y, CELLSIZE, CELLSIZE))
    if GRID[item] == 1:
        pygame.draw.rect(DISPLAYSURF, COLOR, (x, y, CELLSIZE, CELLSIZE))
    return None
```

一个一个画他们的颜色

```
def blankGrid():
    gridDict = {}
    # creates dictionary for all cells
    for y in range(CELLHEIGHT):
        for x in range(CELLWIDTH):
            gridDict[x, y] = 0    # Sets cells as dead
    return gridDict
```

初始化，全部是未走过的格子

```
def printCount(GRID):
    count=0
```

```
    for i in range(COL):
        for j in range(ROW):
            if GRID[i,j]==1 :
                count+=1
    clearTheText(CELLSIZE,WINDOWHEIGHT,CELLSIZE*4,WINDOWHEIGHT+CELLSIZE)
    showText('count = '+str(count),CELLSIZE,WINDOWHEIGHT)
```

在巧妙的位置打出我们到底走了多少步

```
def startingGrid(GRID):
    for item in GRID:
        GRID[item] = 0
    return GRID
```

定义一个 GRID——数组

主函数先是随机的染一个格子，再不断地根据鼠标的操作改变格子：左键来走到下一个格子，右键返回上一步。如果点到了 Hint，可以获得一个提示~点到了 pygame 的 x，那么会退出工作。

最后从主函数开始执行。

**风景画中的星球和大陆：**

统一说明;e 为缩放系数在主函数中会给定

首先介绍星球的渐变色函数

```
    def T(x,a):     #椭圆函数
        return 500*e*math.sqrt(1-(x/a)**2)
    def drawFnScaled (t, fn, lx, upper,ly, step):        #画球任意函数边界及上色
```

函数（t 为 turtle，fn 为函数，lx 为坐标原点横坐标，ly 为坐标原点纵坐标，upper 为 x 的变动范围从 lx-upper 到 lx+upper，step 为每步的步长，通过计算函数上的不同点纵坐标，连线作图）

```
        upper1=upper*e
        t.goto(lx-upper1,ly)
        t.pendown()
        t.pencolor(upper*0.42,upper*0.42,255)        #笔渐变色
        t.begin_fill()
        t.fillcolor(upper*0.42,upper*0.42,255)    #上色渐变色
        x = -upper1+1
        y = fn (x,upper1)
        t.goto (x+lx, y+ly)
        while x < upper1-1:
            x = x + step*e
            y = fn (x,upper1)
            t.goto (x+lx, y+ly)
        t.goto(t.xcor(),ly)#由于椭圆最后斜率为无穷，故不能画到头，只能最后加一笔
        t.end_fill()
        t.penup()
```

由于函数为椭圆，可通过取不同的短轴的值画不同颜色深浅的椭圆叠加可能产生球形渐变色的效果，于是有了下面这个函数：

```
    def drawboundary(ran):
```

```
        drawFnScaled (t, T,x0,ran,y0, 1)
```

通过画一系列椭圆，调整颜色与短轴的关系，从外向内画，可以画出渐变的效果。并受此启发，在主函数中用类似于这个的办法画一系列同心圆，颜色渐变，产生星球外的光晕。

然后介绍改良版的每一小段河流或大陆边界，主要考虑到一定要在星球的内部不能跑到星球圆外去，因此加上了一些判断语句，当它离圆边界过近时将方向变为指向圆心，并前进一小段然后继续原来的弯曲的行进，离下边界过近时会向上行进，并将方向调为上。

```
def fore(n,xp,yp):#n 为行进次数， xp,yp 为圆心的坐标
    for i in range(n):
        if math.sqrt((t.xcor()-xp)**2+(t.ycor()-yp)**2)>=480*e:#边界限制
            t.penup()
            x=t.xcor()
            y=t.ycor()
            t.goto(xp,yp)
            t.goto(x,y)
            t.right(180)
            t.pendown()
            t.forward(15*e)
        if t.ycor()<=yp+20*e:#边界限制
            t.setheading(90)
            t.forward(15*e)
        t.forward(5*e)
        an=random.randint(-45,45)
        t.left(an)
```

有了上面那个每一小段的大陆函数，得到画大陆的函数，大陆函数采取数学中极坐标的想法，每按照 fore 函数走一小段后便回到极点，再返回之前的点，将方向调为与矢径垂直，再继续按 fore 函数走一段，这样可以保证大陆边界随机的同时在一定程度上实现对它的控制，可以限制它在球体内。但当循环多次时，最终还要返回极点，这个返回的过程如果做成直线必然很难看，因此我将这个直线分成十份，取九个点，每次 fore 一段后依次按顺序每次去这九个点中的一个，最终回到极点，将一条直线分成九段曲线加上九段直线，明显比一条直线好看的多：

```
    def land(xt,yt,re): #画大陆函数（xt， yt 为极点， re 为初始矢径的位置）
        t.penup()
        t.goto(xt,yt)
        t.setheading(180)
        t.forward(re)
        t.right(90)
        x2=t.xcor()
        y2=t.ycor()
        for i in range(60):
            if (math.sqrt((x2-x0)**2+(y2-y0)**2)>=470*e):
                t.goto(xt,yt)
                t.goto(x2,y2)
                t.right(180)
                t.pendown()
```

```
                t.forward(20*e)
            if (y2-y0<30*e):
                t.pendown()
                t.setheading(90)
                t.forward(20*e)
            t.pendown()
            fore(10,x0,y0)
            x2=t.xcor()
            y2=t.ycor()
            t.penup()
            t.goto(xt,yt)
            t.goto(x2,y2)
            t.right(90)
        x3=x2
        y3=y2
        for i in range(1,11):
            x2=t.xcor()
            y2=t.ycor()
            t.goto(xt,yt)
            t.goto(x2,y2)
            t.right(160)
            t.pendown()
            fore(5,x0,y0)
            t.goto(x3+i*(xt-x3)/10,y3+i*(yt-y3)/10)
            t.penup()
```

在 land 函数外套上上色函数，直接边画边上色，事实上 land 函数画出来并不是只有一个闭合的曲线，而很可能出现数个匪夷所思的图像边界，而直接上色歪打正着正好画出了大陆的样子，有些曲线有交叉而恰恰在大陆内形成了河流，因此，大陆就这么凑巧出来了。其实之前花很大心思画过其他形式的大陆，甚至加上了湖泊和沙漠，但组长和别的组员都说丑，让我很伤心，但我也不得不承认瞎搞弄出来这个大陆比之前的漂亮的多……而这一切都是随机带来的，随机大法好！

**生命游戏：**

生命游戏的规则及主体构图参考了网上的数据结构。

简便起见，只展示一个最核心的变换代码，及我们自己创新的东西。从上一个时刻到下一个时刻的变换代码：

```
def tick(lifeDict):#lifeDict 为一个字典，储存方格的生死数据
    newTick = {}
    for item in lifeDict:
        numberNeighbours = getNeighbours(item, lifeDict)
# getNeighbours 为另一个函数，能获得该格子周围存活格子数量
        if lifeDict[item] == 1:#活格子的情况
            if numberNeighbours < 2:#不到两个，死亡
                newTick[item] = 0
            elif numberNeighbours > 3:#比三个多，死亡
```

```
                newTick[item] = 0
            else:
                newTick[item] = 1
        elif lifeDict[item] == 0:
            if numberNeighbours == 3: #复活
                newTick[item] = 1
            else:
                newTick[item] = 0
    return newTick #返回新的数据
```

我们将实验及观察得出的初始数据做成矩阵输入到 lifeDict 中，即可得到美妙的变换图形。方便起见，我们创了一个交互式循环结构，可以直接在格子上点以改变初值：

```
    while h==0:
        for evnt in pygame.event.get():
            if evnt.type==MOUSEBUTTONDOWN:
                mouse_pressed = pygame.mouse.get_pressed()
                pos_x,pos_y = pygame.mouse.get_pos()
                curx=(pos_x-1)//CELLSIZE
                cury=(pos_y-1)//CELLSIZE
                if mouse_pressed[0]==1 :#点左键格子生死状态改变
                    if GRID[curx,cury]==0:
                        GRID[curx,cury]=1
                    else:
                        GRID[curx,cury]=0
                    for item in GRID:
                        colourGrid(item, GRID,(0,255,0))#上色函数
                    drawGrid()#画格子函数
                    pygame.display.update()
                elif mouse_pressed[2]==1 :#右键结束输入，开始运行
                    h=1
                    break
```

生命游戏如果掌握了它的原理编程序并不难，上面程序中的上色函数、画格子函数等详见程序。难的是初值的选取，如何选取初值才能得到绚烂的画面，而我们也是在一次次的偶然与不懈的观察以及浓厚的兴趣中得到了许多宝贵的初值，能演变出绚丽的画面。

**整体：**

主函数：在整合后的主程序中，依次调用了小组成员按照规定格式写的程序文件，即子函数中的尺寸、绘图等模块。

对于部分难以处理的，递归过多的片段，主程序采取先运行后储存为图片，再导入最终的作品中的模式。在这个过程中，主程序调用了小组成员的绘图代码，运行完成后再导入专用的图片转换函数，得到可用于绘图的图片文件，最终再调用图片绘图。

图片转换的函数的工作是将 turtle 作出的图转化成可以再被 turtle 调用的图片文件。由于 turtle 是矢量作图，而且其导出依赖于 tkinter，所以直接导出 png 或 gif 格式文件是不可能的。经过研究，我们发现通过 tkinter 可以导出 eps 矢量图文件，但此文件不能被 PIL 库读取。最终，我们调用了 c 语言编译的 pillow，将 eps 文件转化为 png 文件再用 PIL 库进行后续处理。

**云：**

yuan(a,b,A,l,r)即为以（a,b）为左端点，r 为半径，放缩 l 倍，填充 A 颜色画圆。yun(A,B,P,r,l)即为画颜色渐进从 A 到 B 的云，初始园半径 r，放缩 l 倍，此函数中运用了递归，下一次画圆的左端点做随机加法并半径适度所小，递归调用直至 r<1 时停止。画云的函数虽然并没有用复杂算法，但是效果还是很令人满意的。getMiddle 即为得到中间过程圆的混合色，画山的代码中，draw(A,B,C,N,i,large)画山，A,B,C 为三个坐标，山脚山顶。N 为递归次数，large 为放缩系数。i 为第 i 次，运用了中点位移法画山，递归调用并画三角，最终得到山的形状。

## 14.2.3程序限制

**树和草：**

草地里的草太多，这样画计算机会炸的。可以先画一部分，再将目前的“半成品”输出，再将其作为背景图加入，在上面继续画，会有所缓解。

还有画树的函数，因为大量使用随机数，如果人品极差，画出来的树会莫名其妙。

李逸飞：骑士巡游：无限制。如果棋盘过大，可能会导致爆栈，需要手动设置递归次数上限。深度优先搜索的状态空间数是有限的，必然可以在有限时间内结束。

心：需要保证初始输入值大于等于 0，否则会报错。但是实际上是看不到程序内部函数的，所以这种可能不存在。

**风景画中的星球及大陆：**

程序本身没什么限制，不会因为过于复杂而跑得很慢，但虽然我有两个措施抑制大陆超出星球边界，在极其特殊的随机条件下仍可能出现大陆超出星球边界的情况，但可以保证 95%的情况下不会出去。

**生命游戏：**

如果方格数量过多，可能导致程序跑起来比较慢，无法达到设定的帧数，但不会崩溃

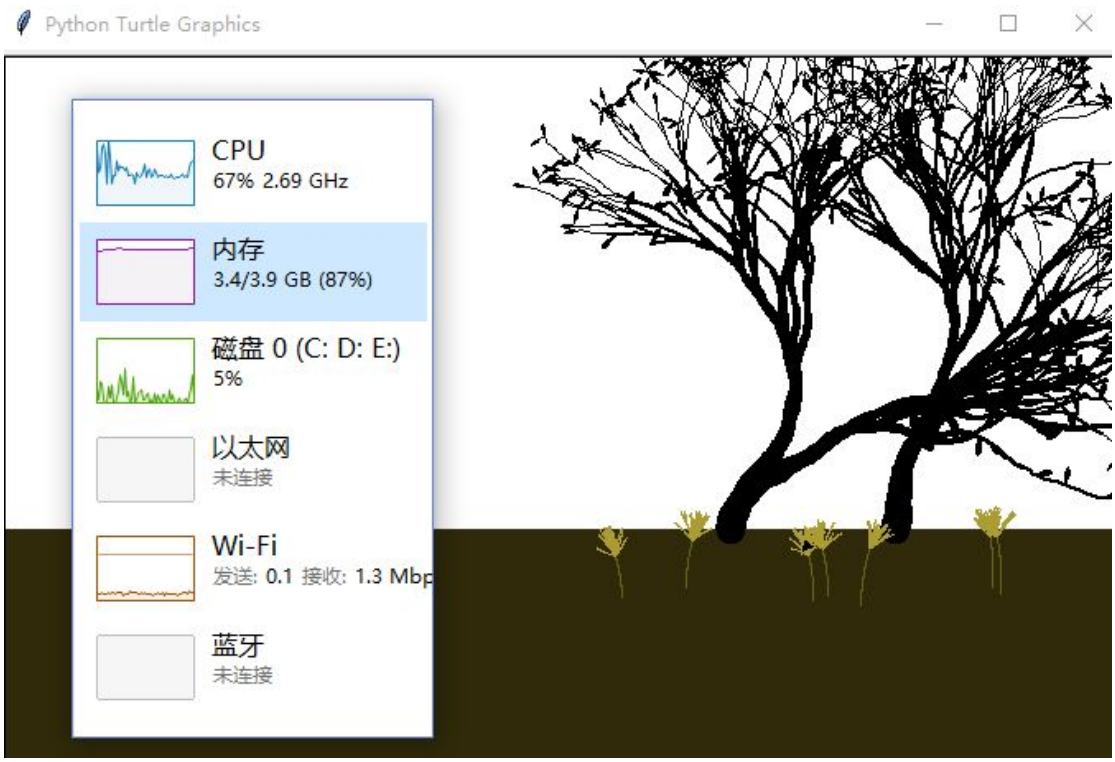
**云和山 ：**

对于程序面对的限制，无论是画云还是画山所用想法都简便易懂，主要代码只有 40 余行，效果也不错，只要所输入数据符合科学范围，比如颜色的（r,g,b）分量都应保持在 0—255 之间，把图像进行放缩时放的太大会超过 turtle 边界，测试得到画山放缩系数不建议大于 1，画云的放缩系数不建议大于 2。但是！！需要注意的是，由于计算能力有限性，递归次数过大时会导致程序未响应，画山不建议递归次数大于 8，画云不建议大于 13。还有可能出错的地方是出示给定坐标不应过于偏，可能会是画图画在 turtle 外。

**整合过程：**

目前最大的困难在于绘制树、草、云时递归过多而导致内存占用太大，计算资源不足，计算机常常在绘图到一半时就崩溃。我们分析得知，由于 turtle 采用的是矢量作图，所以其绘制的每一条线段、每一片区域都需要占用特定内存。矢量绘图互相重叠，递归逐层深入，显然，这样一来，turtle 最终会占用大量内存，更令人绝望的是，turtle 没有整理内存的功能，所以到最后往往会如下图一样，硬件资源利用殆尽，turtle 移动速度极慢。

我们设想的解决方案是将部分已完成的，图案简单而占用大量资源的图保存为位图并清除内存，再通过调用位图完成最终的图案。位图占用资源较少，理论上可行。




## 14.2.4算法流程图

(如果流程图较为复杂，可以选择采用总图和分图的形式)

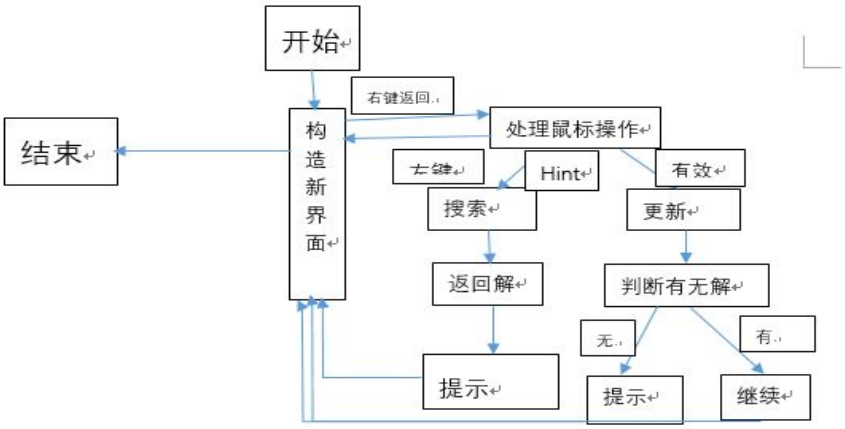
**风景画中的星球及大陆**

画出最外层的圆形光晕——画出球体的椭圆颜色渐变——画出大陆

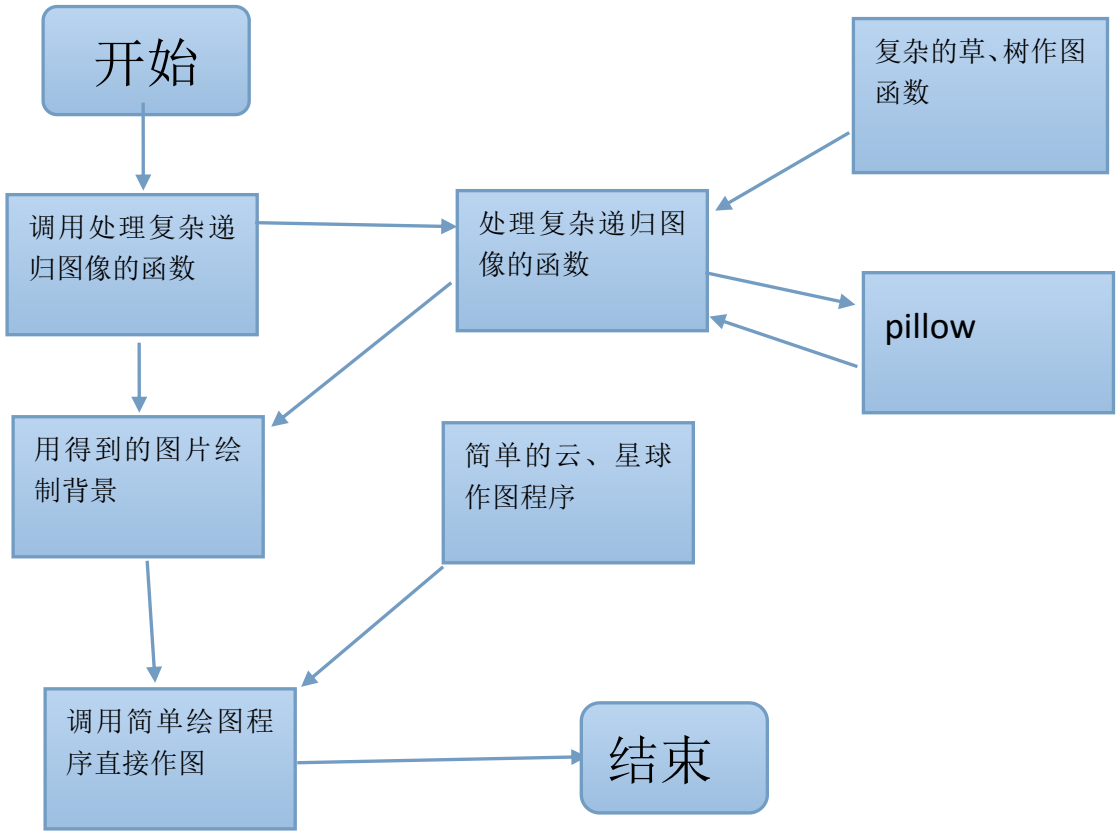
**生命游戏**

设定基本参数（长宽，格子大小，帧数等）——创建存储数据的字典——（画格子）——对应活格子上色——反复刷新进入下一帧

**骑士巡游：**



**整合：**




## 14.3实验结果

### 14.3.1实验数据

**李逸飞：** CPU： Inter® Core™ i7-4712MQ CPU @ 2.30GHz 2.30Ghz
RAM:8.00GB(7.73GB 可用)
系统类型：64 位操作系统
操作系统：windows7
Python 版本：python3.5 64-bit

**华思博：** CPU：Intel(R) Core(TM) i7-4600U CPU@ 2.10GHz 2.70GHz
RAM:8.00GB(7.69GB 可用）
系统类型：64 位操作系统
操作系统：windows7 专业版
Python 版本：IDLE (Python 3.4 GUI - 32 bit)

**聂劭质：** 硬件配置：（CPU/内存）CPU Intel(R) Core(TM) i5-2410M CPU @2.3GHz
内存 4GB DDR3
操作系统：（名称/版本） win10 专业版
Python 版本：（版本号） python3.5

**云沿淞：** 实验环境：python 3.5
CPU:Intel I7 4710MQ

四核八线程
主频 2.5GHZ 最大频率 3.5HZ
内存 8G
操作系统 Windows7 专业版 64 位

**毋轩琦：** CPU：Intel® Core™ i5-4210U CPU @ 1.70GHz 2.40GHz
RAM:4.00GB
OS:Windows10
Python:3.5 32bit

**廖丝丝：** RAM:4.00GB
OS:Windows10
Python:3.5 32bit

## 14.3.2作品描述

（详细说明作品呈现的内容、特色，具有交互式功能的作品可以分步骤展示，建议配以图片）

**心：**

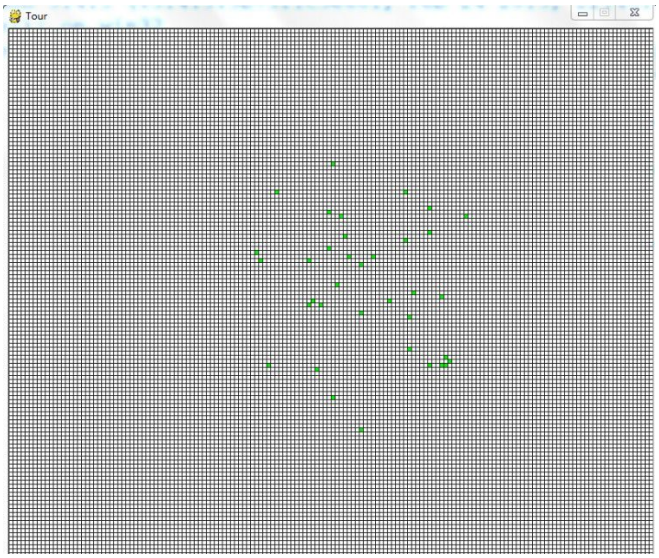
就是利用 PIL 巧妙地画出来一颗心，虽然很简单，但效果异常的好就足够了

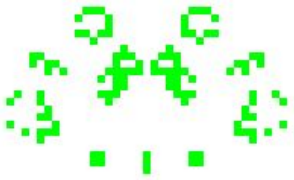
**骑士巡游：**

**风景画中的星球及大陆：**

主要展现的是星球的立体效果，通过颜色的渐变实现。看起来还是有一定立体效果的。而大陆在某种程度上说体现的完全是随机之美，因为它的随机性及其之强，因为我主要限制它不能超出星球范围和大概整体是转圈，因此每一次画出的大陆都不同，也不乏一些酷似美国和亚洲的大陆 233。而河流完全是画多了的大陆边界，在大陆内部便成了河流，可以说是歪打正着……

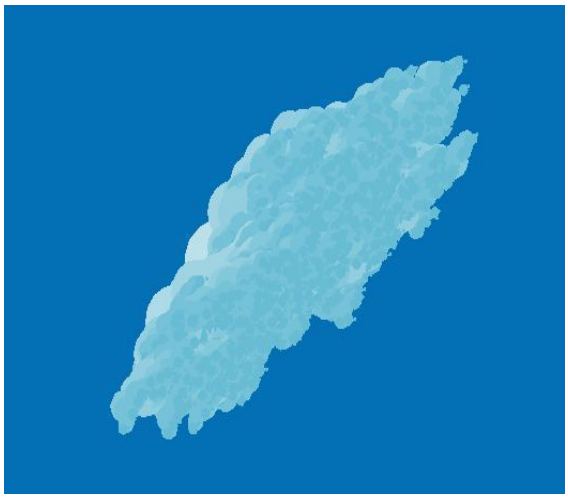
**生命游戏：**



生命游戏有着它自己的规则，我们主要是寻找恰当的初值使它可以自己发展成壮丽的画面。我尝试过修改它的规则，但发现只有在它本来的规则下才能产生最多的变化而以最多的变化次数达到稳定。因此我们主要是寻找恰当的初值使它可以自己发展成壮丽的画面。而事实也非常令人吃惊，一个小小的变化就会导致整个系统的稳定崩溃，几个很小的格子组合到一起就可能变换出整个屏幕的风暴，其中满满的都是表情包 233，它有着无限的可能，变幻出无限绚丽的画面！

**云：**



在画云时确实遇到了一次问题，就是运行不出结果，后来发现是自己的参数使得递归调用了 100 次，运行太慢，后来调整参数以后只调用 13 次得到了上面的结果，画出的云是随机的，方向大小均可调。

**山：**

**最终的作品介绍：**

作品由一颗地平线上的星球、两颗树的阴影与一片草组成，同时还有两朵云作装饰。作品的每一个部分都体现了递归思想，但每一部分所用的方式不同。如云层利用不断画圆来近似，加入随机数后每一片云都不同。而星球整体由特定的数学公式一圈一圈完成，而达到渐变效果。

最细致的是一片草，利用了投影公式来判断每一个位置的草的尺寸，再利用递归的函数生成，由于调用了随机数，所以每一棵草都不相同。

本次大作业我与飞神共同完成的彩蛋中的贴图与对通关的及时反馈是我来实现的。本作品的特色在于其核心的骑士巡游问题的解决是使用递归完成的。同时还具有充分的交互功能，整体风格很有特色。

## 14.3.3实现技巧

**树和草中的秘密：**

RGB 变化真是烦死了，框架画好，但是颜色难看。调了不知道多少变化函数。考虑了麦克斯韦速率分布、速度分布等比较偏的算法思路；

**心中的技巧：**

利用了熟知的椭圆，将其关于 x 轴对称，即可得到一颗心；逐个对像素值进行操作即可得到美观的结果。这个例子说明，基本函数，基本图形，基本性质，基本变换要熟练的掌握！

黑科技：x^2-|x|y+y^2==1　　>_<

**骑士巡游：**

利用了常用算法 dfs，在 dfs 的基础上增添了权重的设计，最终可以几乎不用回溯就可以找到解。实现的时候读入了鼠标的操作，有简单的交互功能，并且为了保证游戏的趣味性，增添了 hint 功能以保证每个人都能够愉快的完成游戏。

黑科技：Warnsdroff 算法，dfs（深度优先搜索）

**风景画中的星球及大陆**

其中最大的成功有两个：一是自创的那个 fore 函数，大陆完全以这个函数为基础；还另一个就是随机的应用，大陆的形状千变万化，河流的形状千变万化，而最好的办法就是在一定范围内任其发展而在大局上加以控制，以达到贴近真实大陆的效果。

**生命游戏**

生命游戏的技巧就是不断学习，仔细观察，甚至可以胡乱设定初值然后观其变化，往往很可能产生一些奇特的结构，及时留下照片，他们的初值就有了，有了初值后将不同的元素

放到一个图里，便可以构成一个个有趣的故事

**总体整合时：**

本程序处理内存占用的技巧在于将部分矢量图转化为位图储存后清除内存再此载入图片，此方法的优越性在于释放了不必要的重叠了很多次的矢量图片，使得绘图速度增加。为了实现此功能，特意调用了外部程序，体现了 python 胶水语言的特点。

# 14.4实习过程总结

## 14.4.1分工与合作

李逸飞：组长，分配任务，组织组会，并且完成了骑士巡游和心；
聂劭质：负责所有内容的整合，尤其是将风景画中的星球大陆，树，云和草灯整合在一起，构成一幅美丽的图片，还有调试其他程序中的 bug；
华思博：负责生命游戏，星球大陆和河流，编程并且开脑洞；
毋轩琦：负责树和草的程序的编写；
云沿淞：负责云和山的程序的编写；
廖丝丝：负责写报告，查找一些素材和资料，打打杂并且学习编程。

最终结果的确认

## 14.4.2经验与教训

对于这样的问题，各人有各人的看法，当然也又害羞的组员不愿意写，但是我们还是看看小伙伴们怎么说吧：

**毋轩奇：**

我觉得分工要看个人兴趣。有点大腿喜欢钻研算法，随他们去。有的人喜欢线条颜色变化，也随他们去。若是让他们都干自己不喜欢的事情，大家都干不好。虽然每个人实力不同，但只要都尽了去做了，都是好样的。而且结果不会差的。

还有我们组的想法很多，虽然有一条主方向，但是一些其他的想法仍旧没有放弃，也取得了不错的结果。所以我认为在最终完成之前，try everything

**李逸飞：**

经验：我觉得我们组一个好就在于没有大腿，这就保证了参与活动的时候不会出现一个人干了 100%的任务，大家的积极性都很高。而且我们没有抛弃任何一个想法，尊重每个人智慧的结晶，充分的调用了所有人的积极性，才使得任务能够快速高效的完成，而且完成度相当之高。算法的难度，结果的精简与华丽……我们组无论是在算法上还是在结果上都有非常突出的闪光点！所以我觉得尊重每个成员，尊重创意是我们组顺利完成作业的核心。尊重是我组的一大闪光点

分工的合理也是一大要素。既有硬性的指标以及硬性指标实现的思路，又给出了拓展的例子分发给了有兴趣的组员，使每个人都能够在大作业的完成中有所收获。合理的分工保证大家积极主动地投入到算法设计程序编写上来，是我组的又一闪光点。

反思教训：作为一个组长，首先我的确不是很称职。我没有很充分的团结组员，组员之间也没有打成一片，凝聚力差，因此当他们各自出现问题时，他们选择自己解决或者不解决，很少有人来问我。而且我作为组长理应做最艰难的任务，但是我为了保持我所谓的“灵活性”，我把比较重要的任务分配给了其他人。虽然不是说其他人做就做的不好，但我觉得作为组长，最重要的任务意味着最大的责任，而这最大的责任需要我来承担。

其次，我不能很好地批评他们的不足，总是乐呵呵的鼓励。实际上从长久的角度看，有问题就应该提出来，不能够藏着掖着，正确性应该高于内部的团结。这就导致我不断的欺骗自己，欺骗他们，最终的结果并不能让人满意。

而且，我没有意识到组员的心理变化。我没有想到有的同学会认为这个期中大作业只有 10 分，所以仅仅是付出了 10 分的努力。可能我从心底里就认为只有 100 分的付出才能有 60 分的回报，如果付出少于 100 分，那么就不配得到回报。我不应该把我这种天真幼稚的想法推广到其它人身上并加以要求，我对组员的要求过高可能会导致作业的失败。

最后，我还是有些软弱。当初分配任务的时候受学长学姐震撼太深，导致我们组的核心工作和其他组有雷同。如果当初能够从前辈的影子里跳出来，坚定地选择一个方向，我相信我们能做的更好。

**华思博：**

我有的时候过于固执，像我先前画的大陆，所有人都说丑，但我却没有推翻重画，力图说服别人那不丑，而徘徊其中，白白浪费了一些时间。此外我感觉组中的交流还不是很够，有的时候自己的想法不被别人理解，但别人也没说出来，导致有的时间被白白浪费。

**廖丝丝：**

每个人都有自己擅长的地方，在组队的时候不要只想着抱大腿，而是要合理地组队，尽量地发挥每个人的长处，在这样的小组里，并不是每个人都要是编程的高手，而是尽量地发挥每个人的长处，那么我相信每个人都会有收获的。

**云沿淞：**

可以试着做一些有趣的小游戏

### 14.4.3建议与设想

**李逸飞：**

组队上应该自由结组，但是设定男女比例上限。一个组至少要有个女生，有个男生，这样对于思想的碰撞非常的有利。

展示方面可以拓宽到线上，先让同学们展示自己的成果，再去选择是否听他们的报告和展示。展示内容录成视频可能会更好。

希望学弟学妹不畏艰难，能够在大作业的完成中获得编程的快乐和知识上的提高。开开脑洞，不要被我们学长学姐的工作限制住了!!!

**毋轩琦：**

大作业不一定要被前人的想法束缚，可以完全推到重来。想法才是关键，至于怎么实现，交给大腿吧 233333

**华思博：**

建议大作业的时间调整一下吧，尽量不要与期中冲突，现在被期中考试和大作业加在中间，很是纠结到底该怎么办……

**聂劭质：**

希望同学们写代码时规范好格式，并写好注释，这将给整合程序的同学带来极大的便利。善于将独立的功能分成独立的函数也是极好的。

**廖丝丝：**

希望大家能够在这样的过程中互相学习，互相成长，还有就是在选择创意时，不放更大胆一点，没试过怎么知道会不会成功。

**云沿淞：**

这次画风景画结果远低于我原本的预期，主要的教训是格式的统一性，风格的统一性，分工的细致要求，以及我在画图时没有考虑到最后整合代码会遇到相当大的困难，以及小组人人码代码造成了最后景物风格不协调，难以拼凑的恶果，以后会吸取教训。之后再码代码会注意命名的规范性，变量名称的合理选取，以及递归次数的合理范围！

## 14.5致谢

感谢 Ghostscript 的开发企业 L. Peter Deutsch 和阿拉丁企业；
感谢生命游戏和骑士巡游的创意带给大家灵感；
感谢国粹——中国象棋给骑士巡游的走法带来的灵感激发；
感谢小组成员不辞辛劳地撕逼与熬夜；
感谢组长的英明指导；
感谢组员地积极配合；
感谢老师提供地（撕逼）机会；
感谢学长学姐的经验（黑历史）。

# 14.6参考文献

用 Python 和 Pygame 写游戏-从入门到精通
http://eyehere.net/2011/python-pygame-novice-professional-index/
让 turtle 输出图片
https://github.com/renyuanL/pythonTurtleInChinese/blob/master/turtle_docstringdict_tc.py
EPS to JPEG or PNG by Python
http://xufive.blog.163.com/blog/static/172326168201211902967O9/
《分形插值算法在模拟自然景物中的应用》（黄天云 张传武）
http://trevorappleton.blogspot.com/2013/07/python-game-of-life.html
生命游戏的引用
http://trevorappleton.blogspot.com/2013/07/python-game-of-life.html

# 15 U 组

# 数据结构与算法课程

## 递归视觉艺术实习作业报告

## （星空外的希望）

（池昱霖*，朱英杰，郭惠昀，武于靖，李海川，卢伟杰）

摘要：创意来自于课件里面的一幅图片，基于我们能够实现的技术，进行艺术性的创造性组织；涉及到的递归过程主要体现在蒲公英、树、花和星空的实现，集中表现在银河系的作图上（整幅画包括了四种递归）；实现原理主要包括利用随机数实现随机分布和利用斐波那契数列控制线条长度，以及两者有机结合的自然模拟算法；涉及的结构主要为列表和树。不仅运用递归绘制图形，而且运用递归计算出图形的分布（即画面构图），整体来说，作品效果已经达到了我们的最初的设想。并且具有一定的哲学和科学模拟意义。

关键字：动态星空、银河分形、列表的应用、随机递归、双重递归、自然模拟利用斐波那契数列和随机递归分布模拟自然

## 15.1创意过程与递归思路

### 15.1.1作品总体介绍

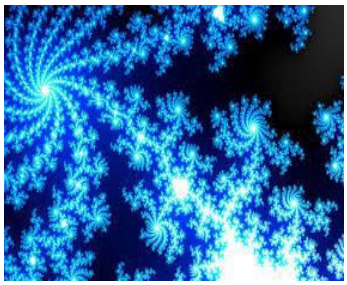
（简要介绍作品的整体内容，作品中体现的递归思想，以及采用的主要数据结构与算法，实现作品的技巧）

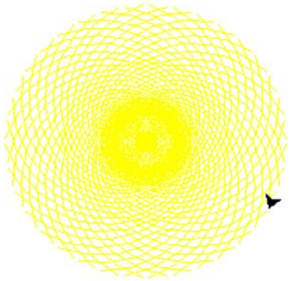
广袤无垠的星空，闪烁着令人迷醉的光辉，在遥远的星空之外，有着我们憧憬的美好，花开叶绿，银河升落，一条河流穿越无尽的疆域，流向远方，满天飘散的蒲公英携带着希望，等待着坚守者的到来，无尽的时间与空间阻挡一切物质，却阻挡不了饱含希望的心。涉及到的递归过程主要体现在蒲公英、树、花和星空的实现，集中表现在银河系的作图上（整幅画包括了四种递归）。算法主要包括利用随机数实现随机分布和利用斐波那契数列控制线条长度，以及两者有机结合的自然模拟算法；涉及的数据结构为列表和树，其中列表主要应用于对颜色和位置的确定；实现作品的技巧为背景调色、不同方式的递归调用（核心为随机模拟）以及黑科技（秘密）

### 15.1.2创意来源

（介绍创意产生的过程，是何种因素激发，以及组内合作分工确定选题的过程）

创意来源于两点，其一是陈斌老师上课展示的一幅图片（如下左图），其二是我们写出的第一个函数的运行结果（如下右图），我们脑洞大开便想到了星空，可是地球上看到的星空太过复杂，我们便把目标转向了银河系。由于银河系整体具有分形的特征，而且较为容易实现，故我们很快便达成了一致。





关于创意的进化，则是 4 月 5 号 PKU 夜奔专场——“蓝色行动”给我们的灵感。这次夜奔关注的是自闭症儿童，他们也被成为“来自星星的孩子”，幻想一下，如果他们真的是来自银河系外的一个星球，每天看着闪烁着蓝光的银河升落变幻，内心那无尽的孤独该怎样消除？所以我们想到了象征希望的蒲公英，我们在那个星球上放了蒲公英，希望无论在这个宇宙的何处，无论历经多长的时间，希望永不枯萎。

## 15.1.3递归思路

（分条、举例说明作品中包含的递归思想）

递归的调用是我们这次作业的一大亮点。总结说来，这幅图片中有四种递归方式：

第一种，在树的作图中的递归，在设置一个最小规模之后，在每一次作图中，都调用下一级的作图函数，实现了递归。

第二种，在银河的作图中的递归，双重递归，简化来说，就是在运行 A 函数时调用了 B 函数，而 B 函数又调用了下一级的 A 函数，从而实现了递归。

第三种，在星空的作图中的递归，集群随机分布递归，在这里我们用到了随机分布函数，在背景上随机选取一个点进行绘图之后，利用递归的思想，在该点附近进行相同过程：选取随机点，进行绘图。图中每种颜色的星星都进行了上千次递归，从而实现了远看平均分布，近看集群分布的效果，这也是我们的**自然模拟算法的精髓所在**：宇宙万物的分布其实也是如此：**分布概率的波函数远看是均匀的，近看又是不均匀的，最简单的例子：叶子要分布在树枝附近，树枝在树附近，树又要在土壤附近集群分布，土壤又集群地分布在一些随机地点。世界就是依靠这样的随机分布和集群分布不断嵌套而成的。**因此随机分布递归在某种程度上具有重大的意义

第四种，数值递归，即斐波那契数列的生成，由于斐波那契数列的特性，其本身就符合递归思想，我们在途中多处用到了斐波那契数列，利用其数值递归的性质，结合随机分布函数产生了一定的随机性，更好地模拟自然。

# 15.2程序代码说明

## 15.2.1数据结构说明

所用到的基本数据结构有：
①栈（Stack）
②树（tree）
所用到的类有：
①海龟（Turtle）
②Python 内置基本类（列表、元组等）
③Image（读取图片数据生成灰度值列表的时候使用了 PIL 模块）

## 15.2.2函数说明

①circle(R=1,angle=90.0,direct=0,exact=1.0,pensize=1,color='black',t=t):
画圆的函数，是所有绘图函数的基本，有 7 个参数可供调节，分别是：
半径、圆心角、方向、细分程度、笔墨粗细、颜色、使用海龟的编号
②Fibonacci(n):#返回斐波那契数列第 n 项的数值
使用递归的方法，返回斐波那契数列第 n 项的数值
③drawFibonacci(R=10,n=5,exact=1.0,pensize=1,color='black',direct=0):#R 为画出的第一个斐波那契数列圆的半径，n 为画到第几项的半径
利用画圆的函数和斐波那契数列生成函数，整合成为画出斐波那契展开线的函数
④drawFibonacciGalaxy(R=10,n=5,direct=0,tense=1.0,pensize=1,color='black',ifback=0,t=t):
利用斐波那契展开线函数，进一步整合，向各个方向画出展开线，形成旋臂状展开线组

⑤getRandom(x,sigmax,num = 1): #num 指返回满足条件的随机分布点的个数，x 为期望,sigma 是正态分布函数中的标准差
为本算法当中的核心函数，围绕一个平均值生成正态分布数值点
⑥
drawFibonacciRandom2(R=10,n=5,angle=90,exact=1.0,pensize=1,color='black',direct=0,sigmax=5,drange=50):#R 为画出的第一个斐波那契数列圆的半径，n 为画到第几项的半径
利用随机分布函数，对斐波那契展开线进行随机化，使其更接近自然
⑦
drawFibonacciGalaxyRandomRecursive2(R=10,n=5,direct=0,tense=1.0,pensize=1,color='black',drange=10,armanglerange=5,ninetydegreerange=15,ifback=0,angle=75):
利用随机分布函数把斐波那契展开线组随机化之后，模拟银河系的结构
⑧

randomDrawNature(center=(0,0),level=100,vertical=50,times=5,R=10,n=5,object='randomGalaxy',color1='purple',color2='orange',direction=0,tense=1/6):
输出主要调用函数
（还有部分代码在“实现技巧”板块）

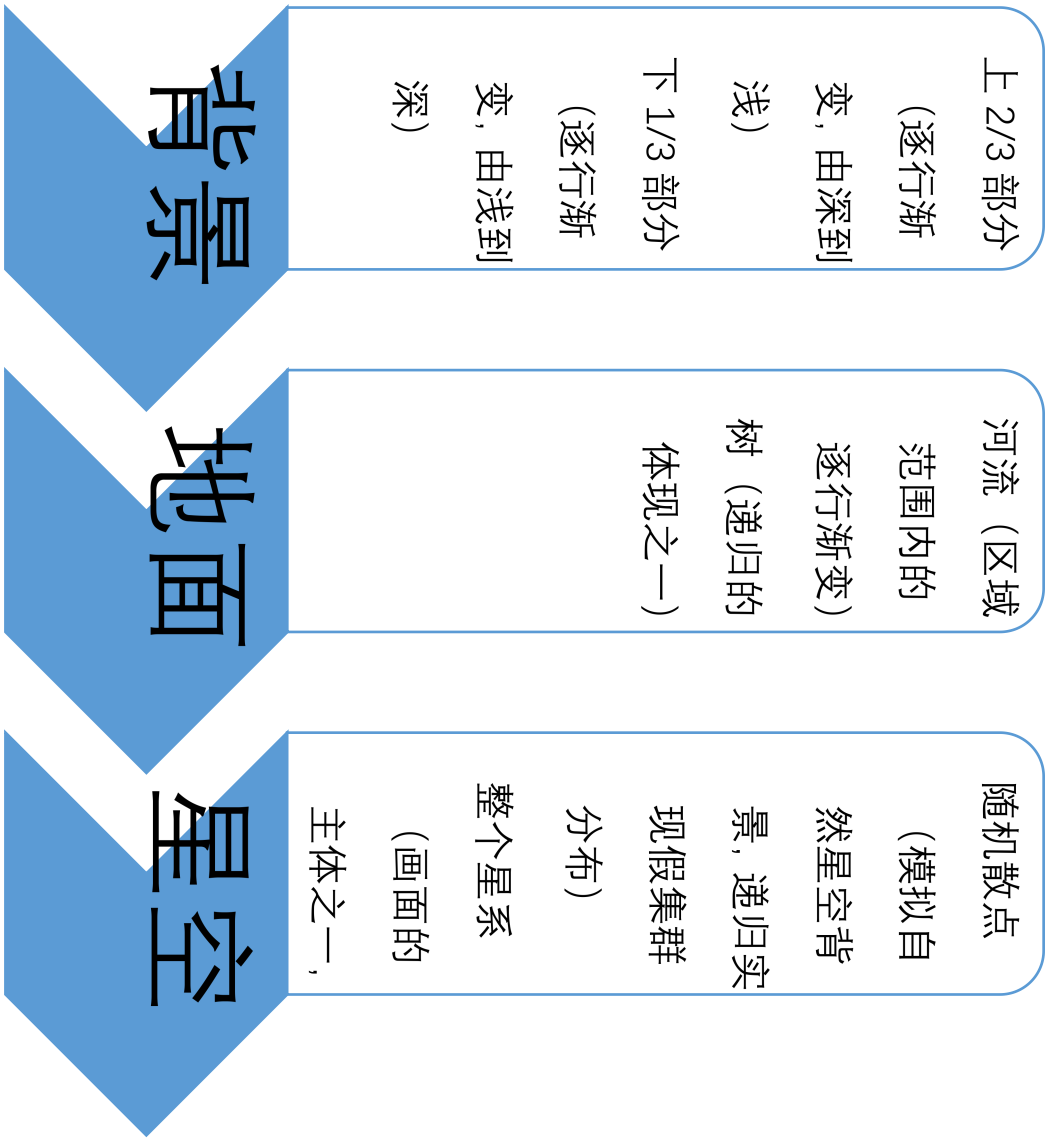
自然模拟画图算法，利用随机分布函数在画布上围绕某一个点开始递归画图，使形成的图像具有集群效应又具有随机性，每次画出的图片都有规律又有所不同，能整体预测但不可以精确预测，符合自然多体系统的规律
除此之外，还有大量辅助画图函数未列出

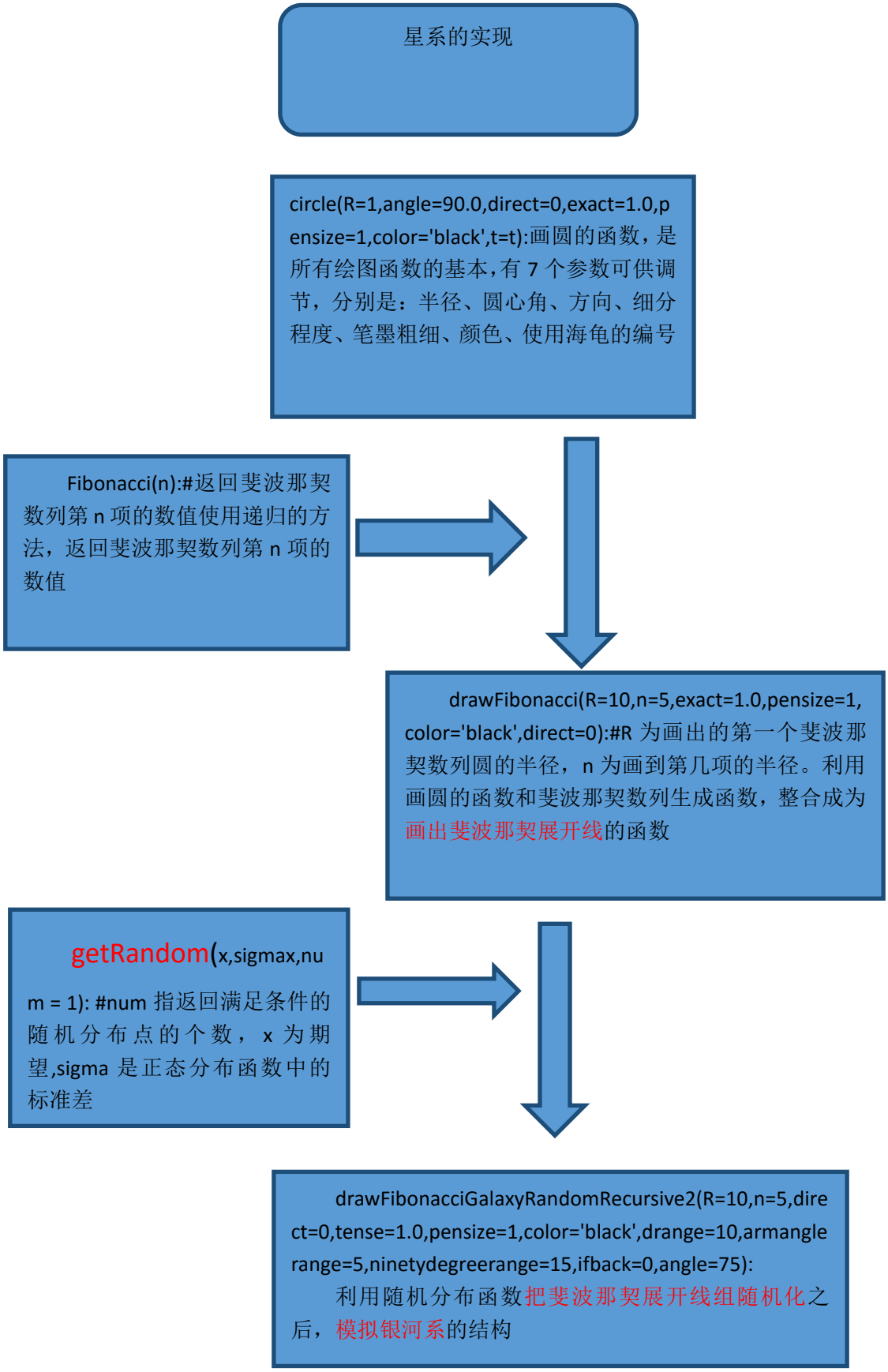
### 15.2.3程序限制

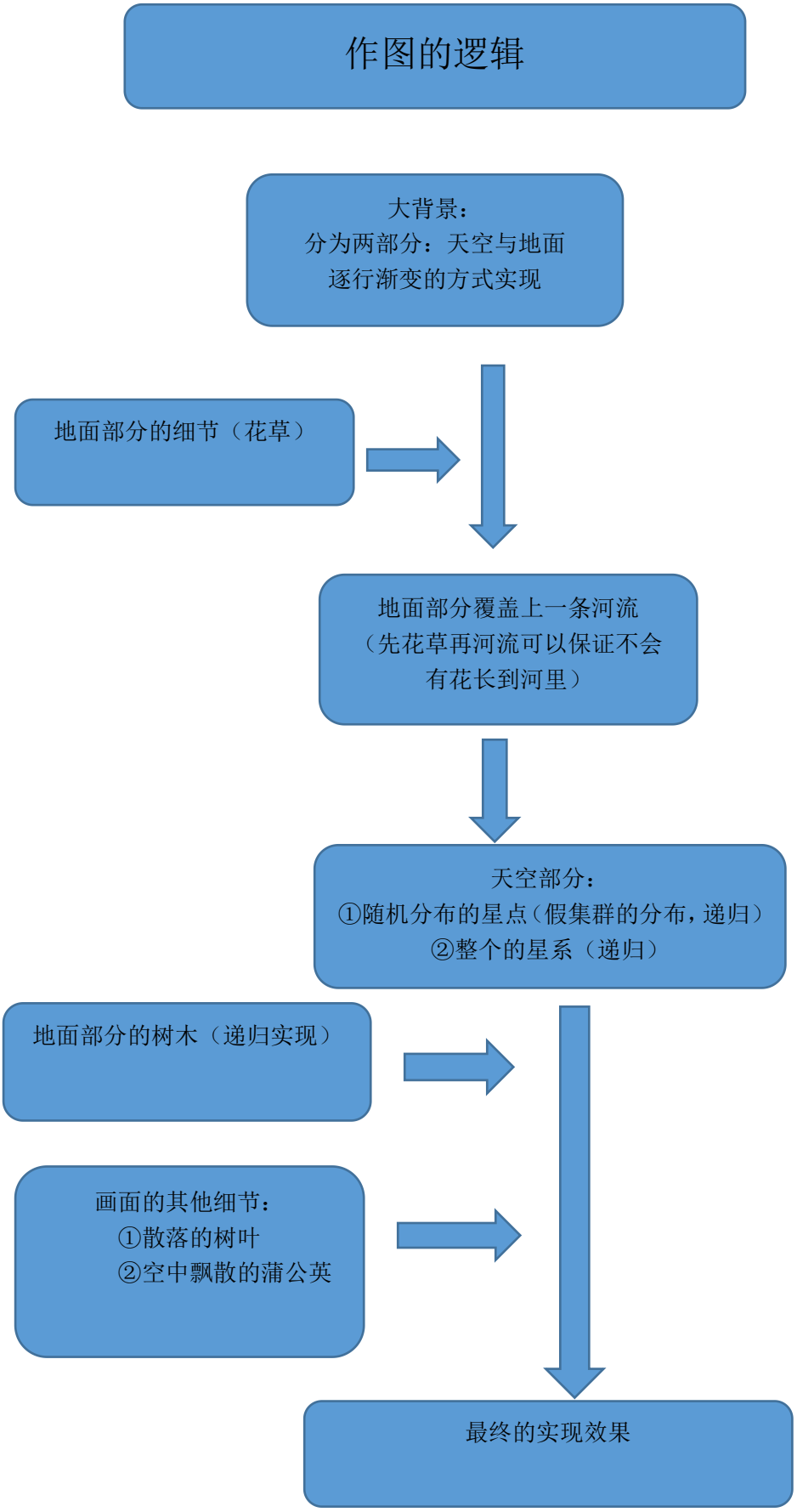
①当使用银河系递归画图方法时，如果输入的半径 R 过大，会造成递归操作次数呈指数级上升，超出递归最高栈深度限制或者超出内存限制，产生错误；
②同样的自然模拟递归在 R 过大时也可能会出现同样的情况
③tense 参量如果过大，会造成画出的曲线过于密集，造成海龟移动缓慢难以绘图，同时大幅加大画图时间，最后造成死机错误。
④输错指示型字符串会造成画图失败

### 15.2.4算法流程图

## 算法流程图（U）


背景
上 2/3 部分（逐行渐变，由深到浅）
下 1/3 部分（逐行渐变，由浅到深）
地面
河流（区域范围内的逐行渐变）
树（递归的体现之一）
星空
随机散点（模拟自然星空背景，递归实现假集群分布）
整个星系（画面的主体之一，


星系的实现
circle(R=1,angle=90.0,direct=0,exact=1.0,pensize=1,color='black',t=t):画圆的函数，是所有绘图函数的基本，有 7 个参数可供调节，分别是：半径、圆心角、方向、细分程度、笔墨粗细、颜色、使用海龟的编号
Fibonacci(n):#返回斐波那契数列第 n 项的数值使用递归的方法，返回斐波那契数列第 n 项的数值
drawFibonacci(R=10,n=5,exact=1.0,pensize=1,color='black',direct=0):#R 为画出的第一个斐波那契数列圆的半径，n 为画到第几项的半径。利用画圆的函数和斐波那契数列生成函数，整合成为画出斐波那契展开线的函数
getRandom(x,sigmax,num = 1): #num 指返回满足条件的随机分布点的个数，x 为期望,sigma 是正态分布函数中的标准差
drawFibonacciGalaxyRandomRecursive2(R=10,n=5,direct=0,tense=1.0,pensize=1,color='black',drange=10,armanglerange=5,ninetydegreerange=15,ifback=0,angle=75):
利用随机分布函数把斐波那契展开线组随机化之后，模拟银河系的结构


作图的逻辑
大背景：
分为两部分：天空与地面
逐行渐变的方式实现
地面部分的细节（花草）
地面部分覆盖上一条河流
（先花草再河流可以保证不会
有花长到河里）
天空部分：
①随机分布的星点（假集群的分布，递归）
②整个的星系（递归）
地面部分的树木（递归实现）
画面的其他细节：
①散落的树叶
②空中飘散的蒲公英
最终的实现效果

# 15.3实验结果

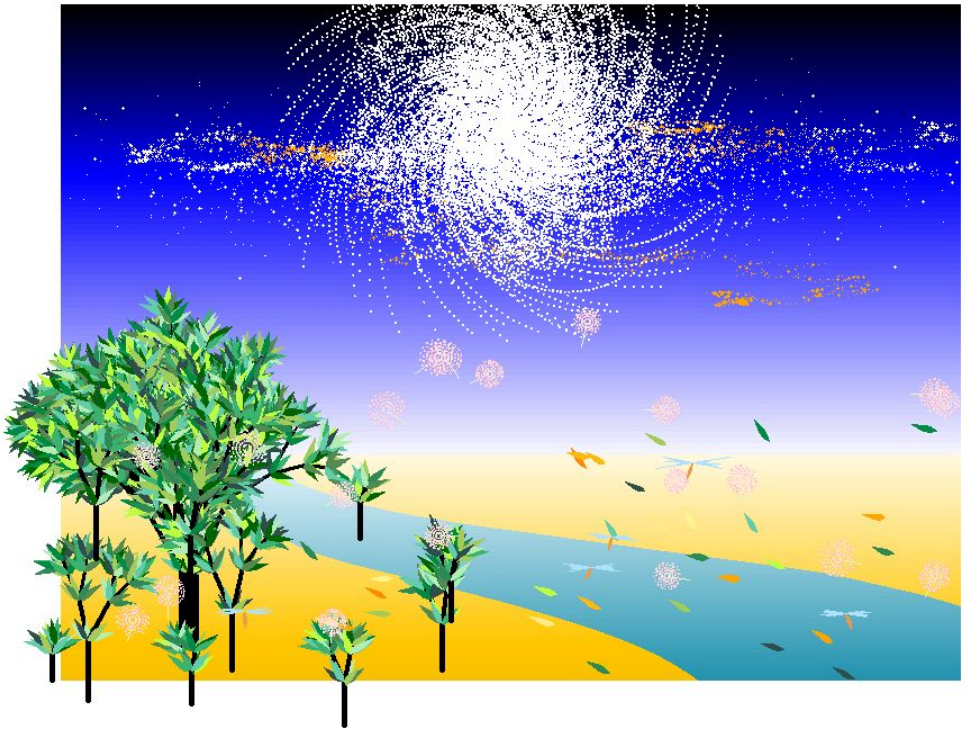
## 15.3.1实验数据

实验环境说明：

- 硬件配置：（CPU/内存）Intel(R)Atom(TM)x7-Z8700 CPU @ 1.60GHz/1.60GHz　内存 4GB
- 操作系统：（名称/版本）windows 8.1 专业版
- Python 版本：（版本号）3.5

## 15.3.2作品描述

（详细说明作品呈现的内容、特色，具有交互式功能的作品可以分步骤展示，建议配以图片）
内容：图片大致可分为上下两部分，上部是图片的主体部分，也是我们花费精力最多的部分，应用前面所提到的种种办法，我们最终模拟出了比较符合自然的星空图景，下部则是一些相对来说简单一些的景物，比如树木、花草、飘散的蒲公英，这些主要是改进了树的算法。
特色：
第一点，背景颜色的渐变，为了让背景看起来更加自然，我们采用了一行一行码代码的方式；
第二点，对自然景物的充分模拟，无论是随机选取，还是集群分布，我们用了很多种方法，让图片呈现出真正的自然混乱而又有序的特点；
第三点，也是最重要的一点，由于运用了随机函数，**我们的图片每次画出来都不一样，**在不同的运行环境中得到的图片在保持整体结构不变的情况下，会有较大的差异，这也是我们的程序的**一大亮点，具有一定的变化性，而不是一成不变。**
有什么自己看图片喽~~~




## 15.3.3实现技巧

（如果在实习过程中发现本作品有令人惊奇的表现或是有特殊的秘密技巧（黑科技等），可在本节中加以描述）

动态化：

```
def
circle_alter(sh,R=1,angle=90.0,direct=0,exact=1.0,pensize=1,color='black',t=t):#直接开始画，不会回到起始点,0 表示左转，1 表示右转
    t.pu()
    step=0.5*3.14159265358979*R/(angle*exact)
    for i in range(int(angle*exact)):#exact 表示画图精细度，数值越大，时间越长，画的图越圆，\
                                     # 默认 1 一般可以接受
        t.forward(step)
        if direct==0:
            t.left(1/exact)
            x = getRandom(t.xcor(),7)[0] #用随机数处理一下位置
            y = getRandom(t.ycor(),7)[0]
            p = ((x+0.5,y+0.5),(x-0.5,y+0.5),(x-0.5,y-0.5),(x+0.5,y-0.5))
#建立了一个四边形对象 p
            sh.addcomponent(p,color) #把 p 添加到了 sh 里面，sh 是一个
compound 类型的 Shape 对象
        else:
            t.right(1/exact)
            x = getRandom(t.xcor(),7)[0] #用随机数处理一下位置
            y = getRandom(t.ycor(),7)[0]
            p = ((x+0.5,y+0.5),(x-0.5,y+0.5),(x-0.5,y-0.5),(x+0.5,y-0.5))
#建立了一个四边形对象 p
            sh.addcomponent(p,color)#把 p 添加到了 sh 里面，sh 是一个
compound 类型的 Shape 对象
    t.color('black')#程序结束改回黑色

def circle_move(R=1,angle=90.0,direct=0,exact=1.0,t=t):
    step=0.5*3.14159265358979*R/(angle*exact)
    for i in range(int(angle*exact)):#exact 表示画图精细度，数值越大，时间越长，画的图越圆，\
                                     # 默认 1 一般可以接受
        t.forward(step)
        if direct==0:
            t.left(1/exact)
        else:
            t.right(1/exact)

def
backDrawFibonacciRandom_alter(sh,R=10,n=5,angle=90,exact=1.0,pensize=2,color='black',direct=0,sigmax=5,drange=50):
    out=getRandom(angle,drange,1)#新增参数 angle 默认值为 90，表示每个半圆的圆心角度数
```

```
    circle_alter(sh,R*Fibonacci(n),out[0],direct,exact,pensize,color)
    for i in range(1,n):#direct 表示方向，0←    1→
        lst=getRandom(angle,sigmax,1)#为了取一个随机角度，画圆的圆心角为90度左右的随机数，sigmax 为标准差，num 为取一个数

circle_alter(sh,R*Fibonacci(n-i),lst[0],direct,exact,pensize,color)

def
drawFibonacciRandom_alter(sh,R=10,n=5,angle=90,exact=1.0,pensize=2,color='black',direct=0,sigmax=5,drange=50):#R 为画出的第一个斐波那契数列圆的半径，n 为画到第几项的半径
    for i in range(1,n):#direct 表示方向，0←    1→
        lst=getRandom(angle,sigmax,1)#为了取一个随机角度，画圆的圆心角为90度左右的随机数，sigmax 为标准差，num 为取一个数
        circle_alter(sh,R*Fibonacci(i),lst[0],direct,exact,pensize,color)
        #heading=t.heading()
        #drawFibonacciGalaxy(3,4,tense=1/12)
        #t.setheading(heading)
    out=getRandom(angle,drange,1)#新增参数 angle 默认值为 90，表示每个半圆的圆心角度数
    circle_alter(sh,R*Fibonacci(n),out[0],direct,exact,pensize,color)

def
drawFibonacciGalaxyRandomRecursive_alter(sh,R=10,n=5,direct=0,tense=1.0,pensize=0.1,color='black',drange=10,armanglerange=5,ninetydegreerange=15,angle=75):
    position=t.position()#参数 angle 表示画出圆的圆心角度数，默认 75 度
    heading=t.heading()
    current=R
    for i in range(int(36*tense)):#sigmax 表示旋臂之间角度差的标准差,d 表示旋臂长度之间的标准差,通过改变最后一个大圆圆心角来实现，默认标准差为 10 度,a 表示组成每条旋臂的半圆圆心角和 90 度偏差的标准差
        t.up()#画出 360 度等角度旋转下的斐波那契曲线的集合，tense=1 时以 10 度为间隔画出 36 条，tense 加倍时条数加倍
        t.setpos(position)#tense 减半时条数减半，但是要注意 tense 乘以 36 之后需要为整数
        t.down()#形状如同银河系的旋臂
        t.setheading(heading)
        lst=getRandom(10*i/tense,armanglerange,1)#取一个随机数，从中心开始画斐波那契数列弧线时转过的角度不再均匀
        t.right(lst[0])

drawFibonacciRandom_alter(sh,R,n,angle,0.0625,pensize,color,direct,ninetydegreerange,drange)#斐波那契弧线的臂长变为随机
```

```
def
drawFibonacciRandom2_alter(sh,R=10,n=5,angle=90,exact=1.0,pensize=0.1,co
lor='black',direct=0,sigmax=5,drange=50,color2='blue'):#R 为画出的第一个
斐波那契数列圆的半径，n 为画到第几项的半径
    current=R
    for i in range(1,n):#direct 表示方向，0←    1→
        lst=getRandom(angle,sigmax,1)#为了取一个随机角度，画圆的圆心角为
90 度左右的随机数，sigmax 为标准差，num 为取一个数
        circle_alter(sh,R*Fibonacci(i),lst[0],direct,exact,pensize,color)
        if current>12:
          heading=t.heading()
          position=t.position()

backDrawFibonacciRandom_alter(sh,current-5,n-2,70,0.0625,pensize,color2,
direct)

drawFibonacciGalaxyRandomRecursive_alter(sh,current-5,n=n-2,tense=1/12,c
olor='white',direct=direct,pensize=pensize)
          t.setheading(heading)
          t.up()
          t.setpos(position)
          t.down()
    out=getRandom(angle,drange,1)#新增参数 angle 默认值为 90，表示每个半圆
的圆心角度数
    circle_alter(sh,R*Fibonacci(n),out[0],direct,exact,pensize,color)

def
drawFibonacciGalaxyRandomRecursive2_alter(sh,R=10,n=5,direct=0,tense=1.0,
pensize=0.05,color='purple',drange=10,armanglerange=5,ninetydegreerange=
15,ifback=0,angle=75,color2='orange'):
    t.pu()
    t.home()
    position=t.position()
    heading=t.heading()
    for i in range(int(36*tense)):#sigmax 表示旋臂之间角度差的标准差,d 表
示旋臂长度之间的标准差,通过改变最后一个大圆圆心角来实现，默认标准差为 10
度,a 表示组成每条旋臂的半圆圆心角和 90 度偏差的标准差
        #画出 360 度等角度旋转下的斐波那契曲线的集合，tense=1 时以 10 度为
间隔画出 36 条，tense 加倍时条数加倍
        t.setpos(position)#tense 减半时条数减半，但是要注意 tense 乘以 36
之后需要为整数
        t.setheading(heading)
```

```
        lst=getRandom(10*i/tense,armanglerange,1)#取一个随机数，从中心开始画斐波那契数列弧线时转过的角度不再均匀
        t.right(lst[0])

drawFibonacciRandom2_alter(sh,R,n,angle,0.5,pensize,color,direct,ninetydegreerange,drange,color2)#斐波那契弧线的臂长变为随机
    if ifback==1:
        t.setpos(position)
        t.setheading(0)
def movingGalaxy(xpos,ypos,heading = 0,x_length = 1,y_length = 1,R=1,angle=90.0,direct=0,exact=1.0):
    w.tracer(0)
    sh = turtle.Shape("compound")

drawFibonacciGalaxyRandomRecursive2_alter(sh,16,5,0,2,0.05,color2=(252,250,242),angle=40,ninetydegreerange=5,color=(250,214,137),)
    w.register_shape("galaxy",sh) #注册了这个 Shape，使得他能够被 turtle 调用
    t.setheading(heading)
    t.shape("galaxy")#turtle 的样子变成了 Galaxy
    t.setpos(xpos,ypos)
    t.shapesize(x_length,y_length) #对 turtle 做了纵向的压缩
    t.speed(0)
    w.tracer(2,1)
    circle_move(R,angle,direct,exact)
随机数：def getRandom(x,y,sigmax,sigmay,num = 1): #num 指返回满足条件的随机分布点的个数，x,y 分别为最初点的坐标,|
    # sigmax,sigmay 是正态分布函数中的标准差
    import random
    l = []
    for i in range(num):
        newX,newY = round(random.normalvariate(x,sigmax),2),\
                    round(random.normalvariate(y,sigmay),2) #这里用 round 函数取了两位小数
        l.append((newX,newY))
    return l
```

# 15.4实习过程总结

## 15.4.1分工与合作

（说明小组分工，合作与交流的方式，历次组会记录（照片!））

我们在确定了所要创作的图形后，结合已经掌握的编程知识，提出了需要对代码进行改进的

地方，加上一些非编程性的任务，确定了每个人的分工。具体分工如下：

池昱霖　改进原有画图函数，使其可以在任意一点作画
　　研究 COLOURMODE 函数的使用，为图片上色做基础
朱英杰　引入适合小组创意的随机数程序，作为其他函数的基础
　　研究 Python 文档，寻找黑科技~~~调整参数以实现图像的变形
郭惠昀　改进画树的函数
　　编写实现背景颜色渐变的函数
武于靖　利用随机数程序调整 DRAWFIB
李海川　会议记录，程序进展实时跟进
卢伟杰　总结，写报告

至于交流方式，由于时间并不统一，主要是线上交流，在遇到瓶颈时会大家聚集在一起讨论，确定下一步的工作和具体细节。（P.S. 晚上的松林好适合讨论，虽然有点吵）

第一次组会

第二次组会


最后一次组会

## 15.4.2经验与教训

（总结实习过程中的工作经验，有哪些方面是得意之处，哪些方面可以改进）

经验：说到经验，印象最深刻的就是组长在一开始对保障互相之间的交流协作的一些措施，首先就是函数的命名，由于我们的任务分散，函数嵌套和互相调用的情况很多，组长在一开始就规定了函数命名的具体规则，保证了不会出现两个人的函数出现混乱的局面。其次，则是有人负责流程把控，不仅实时汇报各个函数整合时的先后顺序，还监督各个任务的完成情况。
教训：由于线上任务分配交流较少，有时候出现不清楚任务的情况。

### 15.4.3建议与设想

（请对本次实习作业提出建议，在组队、创意、展示等哪些方面可以改进？对选修明年课程学弟学妹的寄语；实习作业后续工作的大胆设想）
建议有三个：一个，不要对自己的实力太过自信，选取一个比较简单的最好，由简单到复杂，否则会因为想法太过复杂一次次改变（手动哀伤）；第二个，在一开始确定分工，明确每个人的任务；第三个，在做出一个好看的图片时截屏！截屏！截屏！在展示的时候很有用，而且很有可能再也做不出来了……
寄语：让每个人都能参与，是小组活动的意义所在
设想：利用海龟做出包含更多景物和演变过程的更复杂的图.。

## 15.5致谢

（请感谢下对本小组实习过程有贡献的人和事）
感谢陈春含学姐在关于颜色的函数方面的指导，李然学姐在树的具体实现方面的指导。
特别感谢组长池昱霖的无私奉献与辛勤工作，带领我们做出了这么美丽的作业，组长不仅自己完成了较大部分的工作，而且还指导每一个人，从简单到复杂逐级向每个人分配了各自的任务，组织合作编程，尽自己最大的努力让每一个人都有强烈的参与感和锻炼。
同时感谢朱英杰同学做出的巨大贡献，每一次程序出现问题，他都冲在第一线，并且完成了随机数和动态化两大技术难点。同时还编写了预告 ppt 并且上台展示。
当然，在编程方面做出贡献的还有武于靖和郭惠昀同学，她们两位尽职尽责，无论是颜色还是位置，都把程序调试到尽善尽美，郭惠昀在编程的过程中还发挥了谈迁重写《国榷》的精神，在背景涂色程序意外丢失的情况下，重头再来，并且把新程序做得更好……
最后要感谢负责掌控流程图和艺术设计的李海川同学以及编辑撰写报告和最终展示 ppt 并上台展示的卢伟杰同学，从流程图所占的页数以及制作的精美的 ppt 本报告。就可以看出他的努力。
总之，每个人都发挥了自己最大的力量，感谢 U 组！

## 15.6参考文献

（列出实习过程中用到的参考资料、网站链接等）
（1） https://docs.python.org/3.5/library/turtle.html
（2） http://nipponcolors.com/

北京大学

# 16　　V 组

# 数据结构与算法期中作业

题目：　航　行


组号：　V

组长姓名：　秦珺峰

院　　系：　地空学院

主管老师：　陈斌

二〇一六年四月

# 16.1两次组会的讨论记录

## 16.1.1第一次组会内容记录

**时间 ：** 2016 年 4 月 5 日 13:00--14:00

**地点：** 6 号机房西北角

**人员：** 全员到齐

**组会记录：** 吴浩波

**主题：** 确定所做的主题、分配各自的任务

**内容：**

组会之前组长秦珺峰在群内通知每个人准备各自的创意和想法，并于 4 月 5 日中午一点在 6 号机房西北角召开第一次组会。

一点钟组会开始，首先讨论作业主题。

孟浩瀚提出以富士山为主题，加上各种樱花之类的点缀，因为最近正值春季，网上最新的富士山樱花图片也有很多，可以有很多借鉴的地方。秦珺峰、李寅煌提出反对意见，认为山脉一方面新意不足，另一方面山要求连贯性很强，需要个人完成，不易于分配任务。投票决定后没有采纳富士山主题。

李寅煌提出可以以动漫卡通人物为主题，比如可以以数码宝贝或者神奇宝贝的进化为主题，呈现出通过不断完善画图来表现数码宝贝的进化过程。该提议在新意上很突出，但是在操作性上还是存在问题。吴浩波提出反对意见，一方面要成现动态变化的难度是很大的，另一方面还是存在任务分配问题，这种图片还是要一个人来画，没有办法对整个程序拆分。投票决定后没有采纳动漫任务主题。

秦珺峰提出以辽宁舰为原型进行刻画，最后要表现一副辽宁舰出海时的画面。该想法在新意上比动漫主题差一些，但是可操作性强，分工比较容易，最后程序也容易契合。李寅煌提出了反对意见，其他人比较支持，秦珺峰陈述了选题理由：第一是整个画面可以被分为天空以及天上的云朵、海面以及海上的浪花、辽宁舰航母和航母行驶所作出的水纹四个部分，任务很好分配，最后大家也不冲突。第二是关于选题的定位，由于组内并无大腿，也坚决抵制四抱一的大腿模式，所以选题以让每一个成员都要做事为第一原则，充分发挥每个人的主观能动性。之后进行了投票表决，全员通过了航母模型。

之后进行了任务分配，首先秦珺峰选择了画航母主体，剩下三个人为保证公平公正原则采用了 random 的方式，孟浩瀚画天空和云朵，黄杰画水纹（侧边和后侧两部分水纹），李寅煌画海面以及浪花，定于周五（4 月 8 日）晚 11 点之前在微信群内汇报进度和代码，周六（4 月 9 日）晚 11 点前完善并自己的代码。

**结论：**

1. 确定辽宁舰主题
2. 秦珺峰画航母主体，孟浩瀚画天空和云朵，黄杰画水纹（侧边和后侧两部分水纹），李寅煌画海面以及浪花，吴浩波记录会议并撰写报告
3. 4 月 8 日晚 11 点前汇报进度，4 月 9 日晚 11 点前提交代码

## 16.1.2第二次组会内容记录

**时间：** 2016 年 4 月 9 日晚 7 点

**地点：** 40 楼 106 寝室

**人员：** 全员到齐

**组会记录：** 吴浩波

**主题：** 代码的交流进度和改进

## 内容：

组会主要对第一次组会的分工的任务进行讨论，对写代码过程中遇到的问题进行交流讨论。

秦珺峰首先对整个进度做了汇报，航母主体的代码已经大致完成，还需要一些后续的修缮。天空的代码还可以，但是云朵的随机性处理的不好，水纹的切合度很好，但是大海的颜色以及渐变都有问题，进度有些慢。

黄杰交流了关于水纹的编码，浪花的编码过程还是先要注意航母的几个基本的坐标点，保证最后的几段程序可以契合在一起。

李寅煌对海面和浪花的程序进行了总结，提出了海面的渐变难题比较难处理，小组做了后续的讨论。

孟浩瀚对云朵的随机性的编码也遇到了困难，组会讨论了一会，没有直接解决问题，暂定于网上讨论。

整个的初步代码呈现的效果并不理想，经过讨论后定于 4 月 12 日晚之前将修改的代码提交到组会群内。

## 结论：

**对天空、云朵、海浪、海面的代码进行再修改，定于 4 月 12 日晚之前将修改的代码进行提交。**

# 16.2数据结构与算法期中报告

**摘要：**首先介绍作业图片的表现含义，然后是做作业过程中遇到的困难和障碍，之后是对代码的各个部分的画法和想法进行诠释，最后总结了整个作业的绘画过程里出现的问题和原因。

## 关键词：障碍 代码诠释 问题总结

## 内容：

此次作业我们的选题是航行，我们每个人的生命都是在向着远方航行，我们都像是在海面行驶的船，每个人都是自己的航空母舰，为了自己的梦想战斗，所以我们选择了一副航母在行驶的画面。

这次的作业主要是围绕使用迭代算法来进行图片的绘画，在刚开始接触的时候只是听老师讲觉得这个东西非常有意思，但是实际操作起来还是有一些困难和障碍，主要表现在三个方面：

首先是选题的方面。选题一方面要考虑到题目的新颖，同时由于我们组内并无大神，采用的是多人完成程序的方式，所以在选题的方面也着重考虑了画图的时候比较好实现的方式，比如说刚开始选择樱花绽放这种题材，虽然好看一些，但是程序代码的分配很困难，最终的选择方案其实是偏向于实际操作性强一点的航母版。但是航母版的创新度和新颖程度都一般，并不如其他的两个创意，但是它的操作性强，可以分解成几个模块来做，所以左后选择了航母版来做。

其次是编码的难度问题，组员在做各自的任务时都没有非常顺利，最后在第二次组会的时候还是出现了很多的代码不理想。比如海面的渐变效果完全没法实现，云朵的随机性做的也没有达到目标。不过在最终的作品里海面渐变的效果完成了实现，通过了线条的粗细来对比产生深度感。

最后是代码的契合问题。几个人的代码有时候会突然出现一些奇怪的 bug，出现不能重合的情况，所以在后期的代码整合调整的过程里也花费了很多的精力。

不过即使遇到了上述这么多的问题，代码最后还是很成功的。下面对代码的各个部分的想法一一进行诠释：

首先是云朵，我们在画云的时候采用的是画半圆的方法，云朵的画法采用的是迭代的算法，对半圆的迭代，不断的画半圆，然后当画点的纵坐标等于初始的纵坐标时，画一条横线回来，就得到了云朵。不过云朵的随机性和深度感做的不理想，最后调整后比之前要好一些。

然后是天空和海面，天空是直接做了色彩的填充。海面我们也是首先做演的填充，而海面的纵深感是通过线条的粗细来实现的，我们在下方的海面做随机的粗线条并配上白色的浪花，模拟我们在近处看海面的感觉，在上方的海面我们用的是随机的细线条配白色的附加线，这样海面就产生了由近到远的感觉。

航母主题的画法首先是采用直接调整各点的方式画出了外轮廓和甲板上的起飞线，画甲板上的建筑采用的是迭代的算法，让画点不断的画平行四边形，每次画完一个后调整初始点的纵坐标向下移动一点，直到画出主体。

航母航行的水纹也是通过迭代的算法进行实现，顺着航母底部的直线，从纵坐标最小的点开始在它的附近不断的做随机画白点，然后将初始点的纵坐标向上移动，用同样的画法做随机白点，整个出现的效果就是航行的水纹。

整个的绘画过程是首先进行天空和海面的底色涂色，用半圆画法画云朵，然后画海面的蓝白线条，做海平面的纵深感，最后画航母的主体。在迭代算法的使用过程中，还是会有一些力不从心的感觉，虽然算法都懂，但是要画出漂亮的图片则需要不断的调整角度，每次迭代区间的大小，比如云朵的画法是个很简单的迭代，但是要画很漂亮的、很自然的云朵则需要对每次画的半圆的大小和角度进行不断的改进。

总体来讲，每个组员在各自的任务方面都比较努力，但是局限于水平的差异，最后的图片还是出现有些部分精细有些部分粗糙的情况，在整个完成作业的过程

里还是交流太少，即使每个人分配了不同的任务，还是要多交流，才能保证最后图片的一致性，相信这些也是老师希望我们在完成作业中可以学到的东西。

## 参考文献：

百度百科
军事网 铁血网
《Python 算法教程》[挪威]赫特兰、 凌杰 陆禹淳 顾俊

# 17 W 组

# 数据结构与算法课程

## 递归视觉艺术实习作业报告

## （圣诞夜·雪）

（实习小组成员名单：苏培臻＊，粟恒奕，张浩源，伍晗，邬科元，杨子浩）

摘要：
创意来源：寻找自相似图形，最后确立雪花为主体，圣诞夜为主题
实现原理：自相似图形，缩小规模调用自身；改变 turtle 形状，从而呈现动态效果
涉及的数据结构与算法：递归，线性，拟合
作品呈现效果概述：一个宁静而祥和的圣诞夜里，月光洒满了大地，圣诞老人驾着麋鹿在雪夜中给千家万户送去礼物与祝福，一颗圣诞树上落满了闪亮　的星星和浓浓圣诞节气氛的铃铛，雪花，就在这样美妙的夜晚，悄然降临。

关键字：圣诞夜、雪花、圣诞树、月光、递归

## 17.1创意过程与递归思路

### 17.1.1作品总体介绍

一个宁静而祥和的圣诞夜里，月光洒满了大地，圣诞老人驾着麋鹿在雪夜中给千家万户送去礼物与祝福，一颗圣诞树上落满了闪亮的星星和浓浓圣诞节气氛的铃铛，雪花，就在这样美妙的夜晚，悄然降临。寻找自相似图形，最后确立雪花为主体，圣诞夜为主题。自相似图形，缩小规模调用自身；改变 turtle 形状，从而呈现动态效果。

### 17.1.2创意来源

确定主题的过程：第一个思路：网上找现成的图片，名画之类的模仿作画，但感觉名画效果不易呈现，自然风景的山水花草树木过于普通，难以创新，因此放弃；第二个思路，从可以用递归画出的东西入手，也就是找有分形性质的物体，海岸线，闪电，云，雪花，还有最常见的树，讨论后选定了雪花。确定了主体，一起构思了其他要素，如森林，圣诞树，房子，雪人等，构成了我们的圣诞夜雪景主题。

### 17.1.3递归思路

作品中递归的运用主要在雪花，树和月亮上。

1st 其中远景的树是简单的通过描绘多边形，并缩小规模调用自身，直到参数小于设定值时结束，这样画出的多个多边形组成的，是典型的递归思想。

2nd 而月亮按原来的设想是不需要用递归的，在创作过程中我们想到，也许可以用颜色的渐变来模仿月光的效果，因此，绘制月亮的函数中除了坐标参数和控制大小的参数，还有控制颜色的参数，每次调用时先画一个圆并根据输入的参数填充颜色，然后调用自身时的参数有所变化：半径减小，颜色略微改变，直到半径参数小于设定值时结束，这样可以画出大小和颜色渐变的多个同心圆，以模拟月亮的月光对周围产生影响的效果。

3rd 主体之一圣诞树是典型的运用递归画出的物体，本作品中的树特殊之处在于与传统的分叉的树形状差别较大，圣诞树的形状近似于一个三角形，并且是较为对称的，而传统的树一般是呈伞状的，二者有所区别。因此此作品中树的主干与枝干不是用同一种方法实现的，主干即一条较粗的直线，我们让主干每前进一定距离，随即向左或向右转过一定角度然后调用枝干的函数，这样就实现了分叉，接下来画枝干时就与传统的树较为接近了，每个枝干都可以看做“一棵树”，且是较瘦长的树，就可以利用树枝自相似的性质，采用递归调用自身的方法画出枝干了，此时在其中加入画树叶的函数，就可以利用递归多次调用自身的优势，同时得到许多树叶，以达到所需要的茂密的效果。完成一组枝干、主干再前进一定距离后，再调用画枝干的函数时参数需要减小，以达到树的整体呈三角形的效果。由于作画方式的不同，我们的圣诞树自身整体并不是自相似的，而是每一个枝干都可以看做一棵具有自相似性质的树。其中还运用了随机数以更贴近现实。

## 17.2程序代码说明

### 17.2.1数据结构说明

无自定义类及数据结构改进。

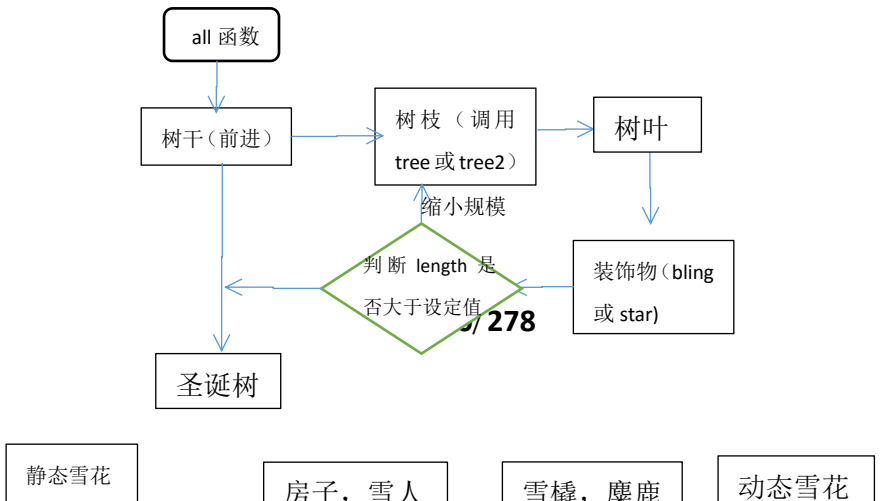
## 17.2.2函数说明

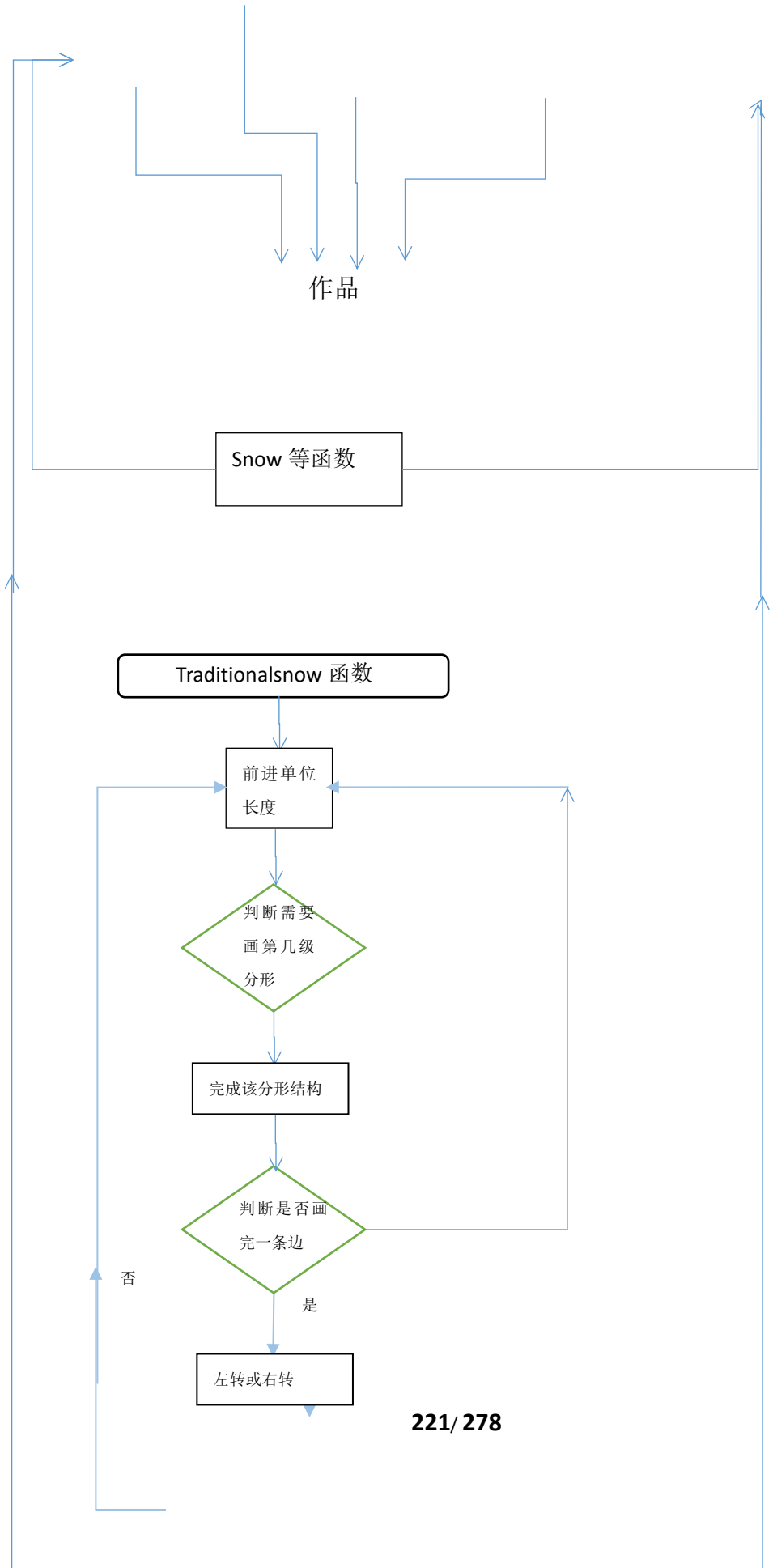
树：（接口 t——turtle，length——长度，size——大小）
tree：产生树的函数，通过缩小规模，每长一段树干就分叉产生树枝，调用自身——递归
add：产生叶子，在每一次调用 tree 函数结束时，生成叶子
star：在每一次 tree 结束之后，产生一个随机数，若在范围内，就画一个星星
bling：在每一次 tree 结束之后，产生一个随机数，若在范围内，就画一个铃铛
fall：产生两个随机数作为横纵坐标，在该位置随机画一个星星或铃铛
（说明算法中各主要函数的接口、功能、采用的算法等）
雪花：（size：雪花的大小，step：雪花最短边的长度，限制 size＝step＊3^n）
partsnow：通过奇偶性的判断从而确定分形结构，进而构建出一个雪花
traditionalsnow：通过分形作出的雪花
snow1/2/3:通过拟合作出的雪花
snow background：随机产生动态和静态的雪花及其位置，并尽量使之不和其他景物产生重复

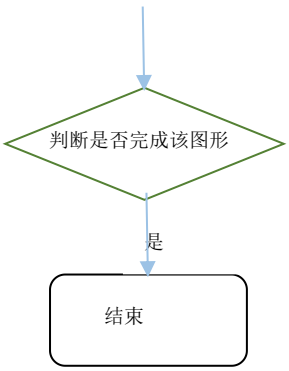
## 17.2.3程序限制

尚且未知。

## 17.2.4算法流程图




作品
Snow 等函数
Traditionalsnow 函数
前进单位长度
判断需要画第几级分形
完成该分形结构
判断是否画完一条边
否
是
左转或右转




# 17.3实验结果

## 17.3.1实验数据

实验环境说明：

- 硬件配置：英特尔酷睿 i5，6G
- 操作系统：windos8
- Python 版本：3.5.2

## 17.3.2作品描述

（1）一轮明月，通过递归展现的柔和月光，铺满白雪的大地，远处若隐若现的森林。

（2）一座房子，一个雪人，通过递归勾画出的雪花图腾。

（3）一棵采用非典型递归算法画出的圣诞树，拟合与分形并有的精致雪花，通过改变 turtle 形状实现的雪花纷纷扬扬漫天飞舞梦幻场景，以及圣诞老人驾着麋鹿雪橇从天边飞过的奇妙想象。

# 17.4实习过程总结

## 17.4.1分工与合作

（一）小组分工：

（1）程序设计：雪花——粟恒奕
圣诞树——张浩源
雪橇——邬科元
麋鹿&背景树——苏培臻
房子&雪人——杨子浩

（2）报告：伍晗

（3）ppt：苏培臻

（4）报告展示：苏培臻，粟恒奕，张浩源，杨子浩，邬科元，伍晗

（二）小组活动记录：

（1）第一次小组会议

1.讨论作画的主题和内容*。

2.确定画中元素，主要有树（包括近距离能看清细节以及原处只需要轮廓的），雪花（同样由远近大小），麋鹿，房子，雪人等。

3.确定各个要素设计者的分工。

*确定主题的过程：首先提出的第一个思路：网上找现成的图片，名画之类的模仿作画，但感觉名画效果不易呈现，自然风景的山水花草树木过于普通，难以创新，因此放弃；第二

个思路：从可以用递归画出的东西入手，也就是找有分形性质的物体，海岸线，闪电，云，雪花，还有最常见的树，讨论后选定了雪花。确定了主体，一起构思了其他要素，如森林，圣诞树，房子，雪人等，构成了我们的圣诞夜雪景主题。

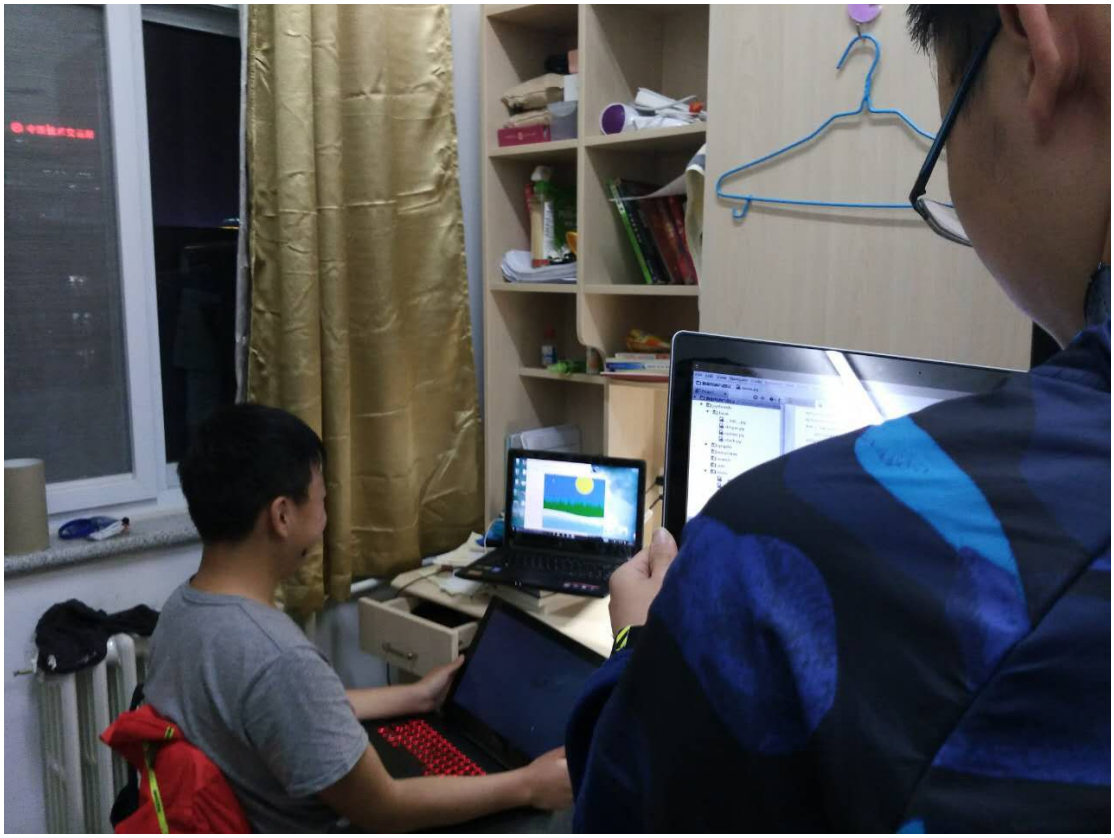
（2）第二次小组会议（2 人加开）

苏培臻，张浩源就其中较难的树的画法的讨论。基本确定的解决方法。

（3）第三次小组会议

1.分享各自画图过程中的遇到的困难，确立周四预告内容。

2..研究报告模板，共同整理总结报告编写所需要的材料。



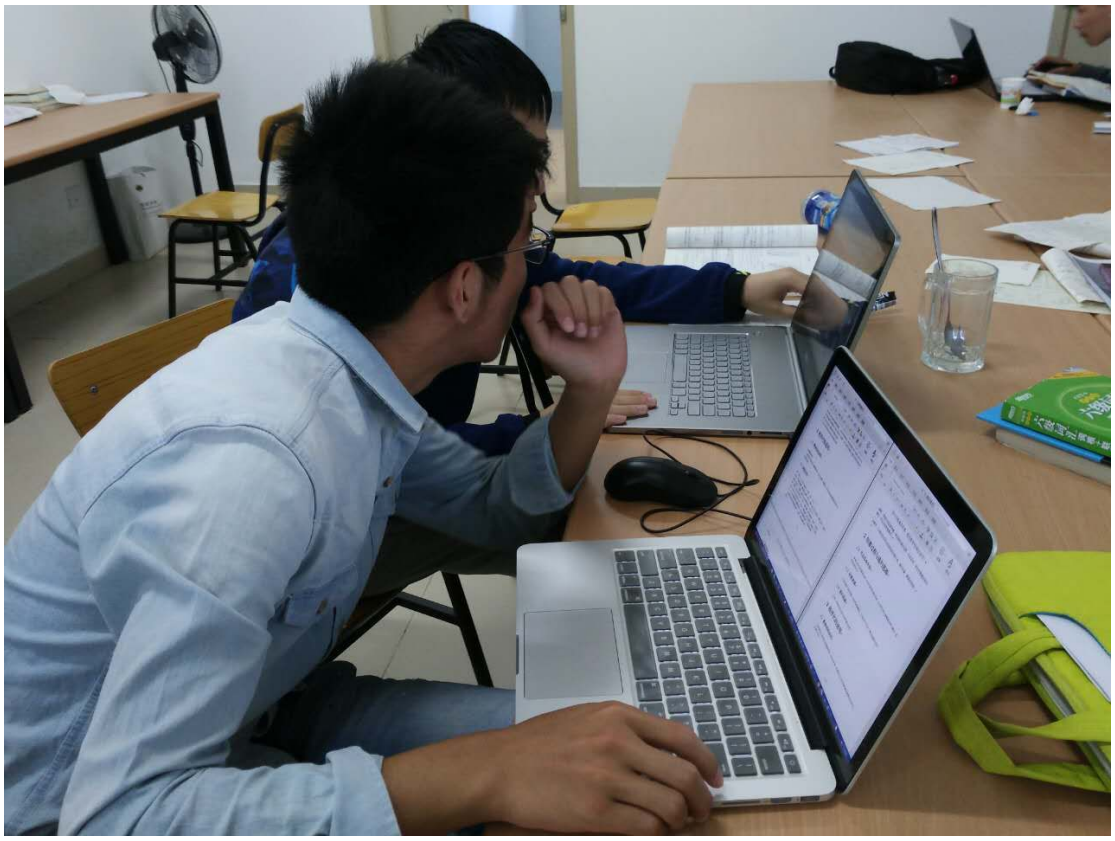
（4）第四次小组会议

1.研究报告模板，共同整理总结报告编写所需要的材料。

2.初步完成程序的整合。

3.就其中运行中的问题进行讨论改进。

4.初步确定展示形式，排练展示过程。



### 17.4.2经验与教训

不足与教训：随机数的应用使得并非每次运行效果都尽如人意，不过总体效果良好；麋鹿、圣诞老人、雪橇，是通过繁琐的程序画出，编写时几乎是一笔一画画出来的，效率低，费时费力，又无法找到好的解决方法；由于基础薄弱以及时间精力有限，无法对一些较高级的模块和复杂的数据结构进行研究学习和运用。

收获：通过改变 turtle 的形状从而营造动态效果，如雪花飘舞、圣诞老人的移动；在研究圣诞树画法的过程中对传统的画树的递归算法进行改进，达到了预期效果；小组中的每个人虽然以前编程基础都很薄弱，这次分组因为没有“大腿”使得每个“小白”组员们都参与了程序设计编写，并在此过程中得到学习锻炼，受益匪浅。

### 17.4.3建议与设想

建议：由于基础薄弱以及时间精力有限，无法对一些较高级的模块进行研究学习和运用，希望以后有机会老师或助教可开一些课外讨论会帮助编程小白们。同时我们认为以大腿为核心的建队方式，编程报告展示分工明确，有其优势，而编程基础较弱的同学其实也可以多参与编程，未必把任务都交给编程高手，毕竟只有不断实践才能取得进步，这点此次作业中我们感触颇深。

设想：利用课余时间学习一些 python 中较高级的模块，更好地实现雪花降落、飘舞，甚至积雪的逐渐产生等各种各样的动态过程。

寄语：生活中充满了递归之美，多学些编程知识才能更好地将它们在屏幕上实现！

## 17.5参考文献

https://docs.python.org/2/library/turtle.html

# 18 X 组

# 数据结构与算法课程

## 递归视觉艺术实习作业报告

## （花树鸟巢藏）

（杨昕陶*，金婧，项楷，崔博，李锴，商修齐）

摘要：我们从鸟巢、树、倒影以及烟花上看到了优美的景观与 python-turtle 代码的兼容性。我们利用递归绘出了七夜繁花（花树）和漫天璀璨（烟花），涉及到接近 20 个函数，涉及到 turtle、math、Pillow 等库。通过图形叠加、乌龟隐现等多种技巧，最终呈现出三千弱水、七夜繁花、漫天璀璨、岿然鸟巢的效果。
关键字：花树鸟巢、烟花花树、集合结构、递归结构、pillow 图形处理

## 18.1创意过程与递归思路

### 18.1.1作品总体介绍

作品《花树鸟巢藏》展现一幅别开生面的渐变图景——从一树到一巢再到点点烟花的动态变化，出人意料而寓意其中。构图运用树木交错的枝条和深沉的色彩，先声夺人地描绘的树木的繁盛。树木之中怎样藏入鸟巢，怎样体现生命的丰富多彩，成了最富新意的亮点所在。《花树鸟巢藏》巧妙地联想到了北京奥林匹克锦标赛的主会场——鸟巢的结构设计。作品运用鸟巢纵横交错的构架与树木的局部重合性，富有创意地通过擦除轮廓地方法，勾勒出鸟巢的外轮廓，进而通过添加内部的支柱和装饰彩灯，生动地还原了夜晚鸟巢的美丽效果。随着主画面的展开，夜空中升起了点点烟花，更生姿色和欢喜。

作品中处处体现递归思想，布景地设计、树枝的伸展、烟花彩灯的光束，玲珑的水波都运用到了递归调用自身的核心算法，通过将相似问题的持续分解，优雅精妙地解决了问题。

主要数据结构：列表、元组

实现作品的技巧：递归算法、pillow 库、turtle 库

### 18.1.2创意来源

创意来自课上用 turtle 设计的树木，联想到鸟巢繁复的构架与树木有异曲同工之妙，进而改进树木的构图，给欣赏者以惊喜。小组讨论时曾有第二方案——绘制蒙娜丽莎的表情包。但蒙娜丽莎的表情包，虽然有趣，能博众一笑，但却不能带来持久的惊喜感。对比两个创意，

综合考虑了精彩度、欣赏性及可行性，我们最终选定了《花树鸟巢藏》这幅作品。

组内的分工协作为：

| | |
|---|---|
| 金婧 | 书面报告（主） |
| 商修齐 | 书面报告（副）/个别代码（倒影） |
| 杨昕陶 | 码农（鸟巢） |
| 崔博 | 码农（烟花） |
| 李锴 | 码农（小树） |
| 项楷 | 码农（动态激发） |

（介绍创意产生的过程，是何种因素激发，以及组内合作分工确定选题的过程）

### 18.1.3递归思路

1.乱树生成：依靠递减的分裂次数作为递归的变量
2.行道树生成：依靠递减的分裂次数作为递归的变量
3.石子路：依靠递增的 X 坐标作为递归的变量
4.背景：依靠递减的 Y 坐标作为递归的变量

## 18.2程序代码说明

### 18.2.1数据结构说明

1.颜色输入元组
2.烟花大小调节列表
3.烟花乌龟列表

### 18.2.2函数说明

- Def boom(turtle,number):

定义 乱树爆炸
#输入乌龟和分裂次数，生成乱树

- def mirrorlength(coordinate,r=0.618,l=-70.0)

定义 镜面长度函数
#传入像的坐标，压缩比例，以及对称面 y 坐标，生成镜像，作为倒影原材料。

- Def mirrorangle(angle,r=0.618)

定义 角度镜

#此函数通过 math 库将岸上 turtle 的方向倒映到水中，通过数学计算可知满足：tan α*0.618=tan β，从而返回镜像中需要的偏转角度

- Def nestanditsopposition(t)

定义 鸟巢外部轮廓及其倒影(指定海龟)

#描绘鸟巢外部轮廓及其倒影：同时使用两个 turtle，在描绘岸上风景时，利用上述 mirrorlength 和 mirrorangle 函数同时描绘水中的倒影。

- Def nestcrossanditsopposition(t)

定义 鸟巢背面钢筋及其倒影(指定海龟)

#鸟巢背面交叉的钢筋及其倒影：同上使用两个 turtle，在描绘岸上风景时，利用上述 mirrorlength 和 mirrorangle 函数同时描绘水中的倒影。

- def nestcrossanditsopposition(t):

#鸟巢背面交叉的东西 cross 及其倒影

- Def buildinganditsopposition(t)

定义 建筑物(赛场和观众席)及其倒影(指定海龟)

#建筑物(赛场和观众席)及其倒影：使用 begin_fill 和 end_fill 函数填色，并利用上述 mirrorlength 和 mirrorangle 函数同时描绘水中的倒影。

- Def lightanditsopposition(t)

定义 灯光及其倒影(指定海龟)

#灯光及其倒影：一共九灯，照耀运动场

- Def barrieranditsopposition(t)

定义 鸟巢前面交叉的钢筋及其倒影(指定海龟)

#鸟巢前面交叉的钢筋及其倒影

- def bkcolor(y):

定义 背景颜色生成函数

#生成 y 值对应的颜色，从而实现背景颜色渐变

- def backg(bkturtle,x,y,st,end,size,length):

定义 背景生成函数

#利用递归以及 bkcolor 函数，传入起始坐标和区域长度、单线条尺寸，实现区域背景着色

- def combk():

#执行背景绘制

- def comol():

#执行鸟巢绘制

- def theTree(x,y,size):

定义 产生树函数

#传入树底坐标和和尺寸，利用 brunch 函数、递归和随机数产生树

- def brunch(t1,t2,length,n):

定义 分枝函数

#传入乌龟、分枝长度和分枝数，产生枝杈

- def theStoneRoad(t,x,y,end):

定义 石子路生成器

#传入石子路起始坐标，利用递归生成石子路

- def comtheStone():

定义 石子路生成器执行函数

#选择恰当的参数执行石子路生成器，生成石子路

- def firework(number,pathlen,setx,sety,fwSize):

定义 烟花

# 传入烟花上升总步数参数 number、每部高度参数 pathlen、烟花底部位置参数 setx、sety、烟花爆开大小参数 fwSize 来画出烟花。需要调用下方的 firework2 函数

- def fireworks2(num,tg,tga,pathlen,r,FireworkSize):

定义 烟花 2

#传入参数烟花上总步数 num、乌龟列表 tg tga（在 firework 函数中产生传入 此函数）烟花每步上升高度 pathlen、烟花上升过程中三个圆点大小列表 r，以及 烟花大小 FireworkSize

- def rannum():/def ran01():/def rannum3():/def rancolor():

各种随机函数，用于产生各种随机数或随机数组：

- def getButton():

定义 按钮生成器

#生成两个按钮，以便借助 button_for_fireworks 函数控制烟花的发射和消失

- def button_for_fireworks(x,y):

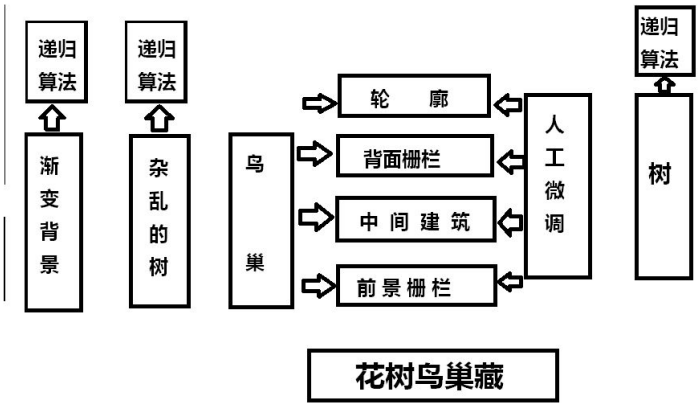
定义 烟花控制器

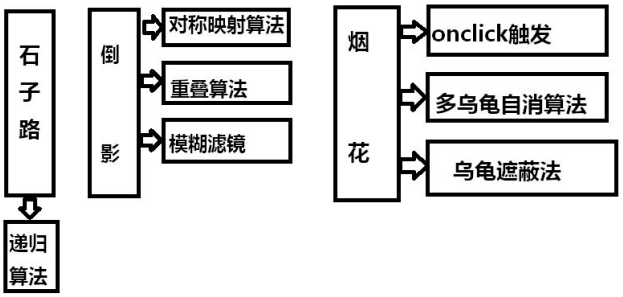
#使用两个按钮控制烟花的发射和消失

### 18.2.3程序限制

1. 传入参数 number、pathlen 必须为正，fireSize 以 100 左右为宜
2. 由于一个烟花需要超过 50 只乌龟，烟花运行之后未进行 Reset，内存未能释放，导致烟花函数运行越来越慢。
3. 由于在实现鸟巢及树的波纹、模糊倒影效果时使用了 pillow 库，处理之后文件变为一张图片，无法再实现烟花的释放等各种动画效果。

### 18.2.4算法流程图




## 18.3实验结果

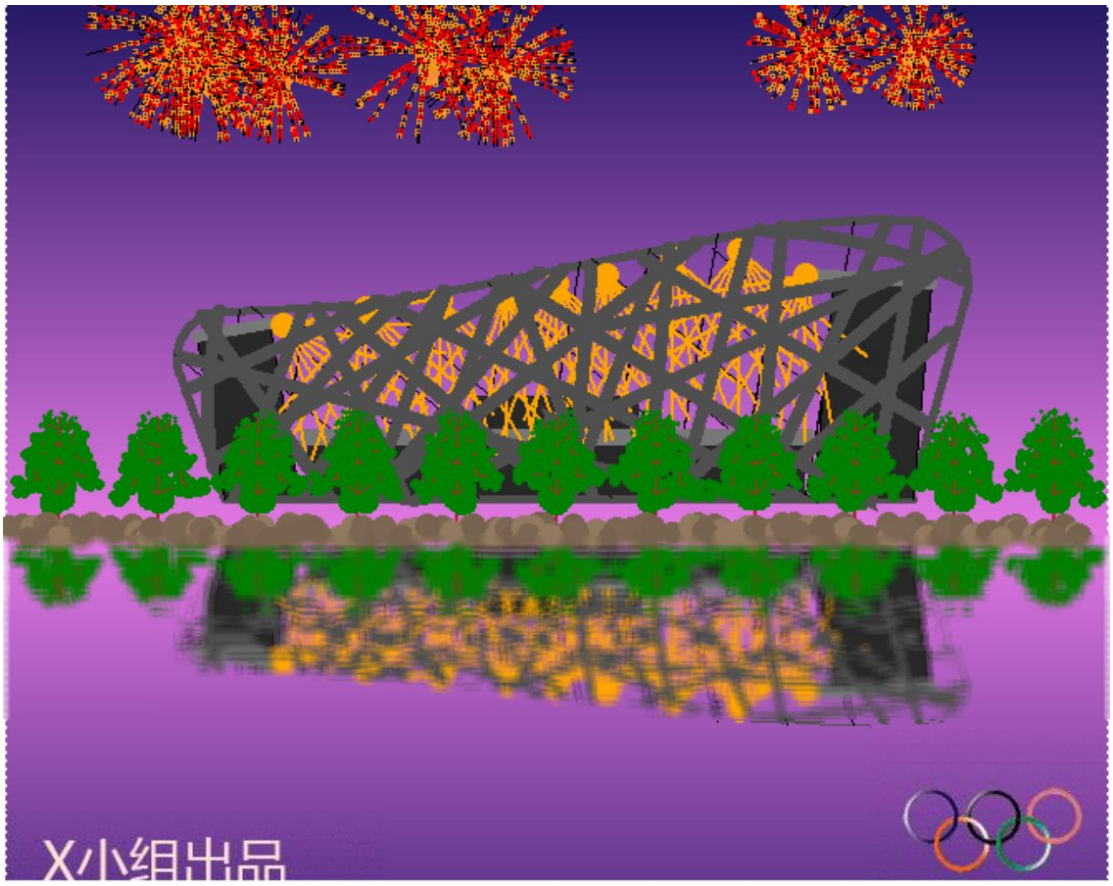
### 18.3.1作品描述

《花树鸟巢藏》巧妙地联想到了北京奥林匹克锦标赛的主会场——鸟巢的结构设计。作

品运用鸟巢纵横交错的构架与树木的局部重合性，勾勒出鸟巢的外轮廓，进而通过添加内部的支柱和装饰彩灯，生动地还原了夜晚鸟巢的美丽效果。随着主画面的展开，夜空中升起了点点烟花，生动地再现了夜晚鸟巢的美丽效果。

作品特色：

1. 晦明变换。作品背景晦明变换，通过色彩渐变，展现了鸟巢亮灯的美丽夜景。
2. 对称美感。倒影的设计，通过模糊化的效果，画面构图稳定，富有对称的美感。
3. 动静结合。静态的鸟巢、树林配以动态绽放的烟花，动中有静，静中有动。
4. 网状结构。网状结构的鸟巢和交错的灯光光线相印成趣，明暗对比，画面饱满。




## 18.3.2实现技巧

1.鸟巢裁剪

Turtle 中的 clear 功能只能针对某个 turtle 的所有“画作”进行删除，不能达到分区域裁剪的效果，所以我们通过用两个 turtle 分别绘制“乱树”和“鸟巢”，然后删除“乱树”，实现“花树鸟巢藏”的效果。

2.鸟巢立体感营造

线条绘制的鸟巢始终囿于平面的感觉，为了实现其立体感，我们最初尝试使用颜色填充，但是颜色填充只能单色，渐变色绘制又过于复杂（边界不规则），所以我们想到直接模拟鸟巢真实结构，由背面、内部到正面依次绘制，实现立体感。

3.烟花处理

Turtle 是单线程轨迹绘制，难以直接实现烟花的上升以及绽放的效果，我们采取多 turtle 短距交替绘制，并借助 tracer 函数加速 turtle 的转换过程，实现烟花的发射和绽放

4.烟花控制

使用 onclick 函数设置按钮来控制烟花的释放和消失。但是绘制烟花的 for 循环中，所申请的 turtle 没有被保存（开玩笑....好几十个 turtle 怎么存），所以不能直接对烟花进行操作。同时，由于背景是渐变色，也不能直接用实心多边形对烟花进行遮盖，所以我们想到用一个“背景形状”的 turtle，利用 hideturtle 和 showturtle 函数控制其显隐，来使烟花消失。

5.倒影制作

简单的对称和拉伸形成的倒影缺少倒影的荡漾感，我们通过 python 的 pillow 库对图像继续进行了碎片化处理和模糊滤镜，使得倒影更具真实感。

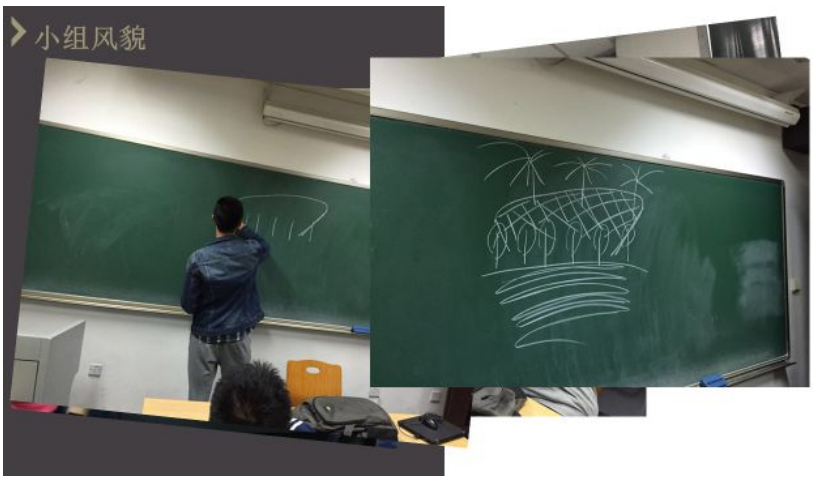
# 18.4实习过程总结

## 18.4.1分工与合作

分工协作的内容如下，通过将画面构图进行模块化，再分配，更好地保证了小组整体的进度和执行力。

| | |
|---|---|
| 金婧 | 书面报告（主） |
| 商修齐 | 书面报告（副）/个别代码（倒影） |
| 杨昕陶 | 码农（鸟巢） |
| 崔博 | 码农（烟花） |
| 李锴 | 码农（小树/倒影） |
| 项楷 | 码农（onclick） |

构思草图的过程和参考图：




线下讨论：

第一次组会：确定框架和内容

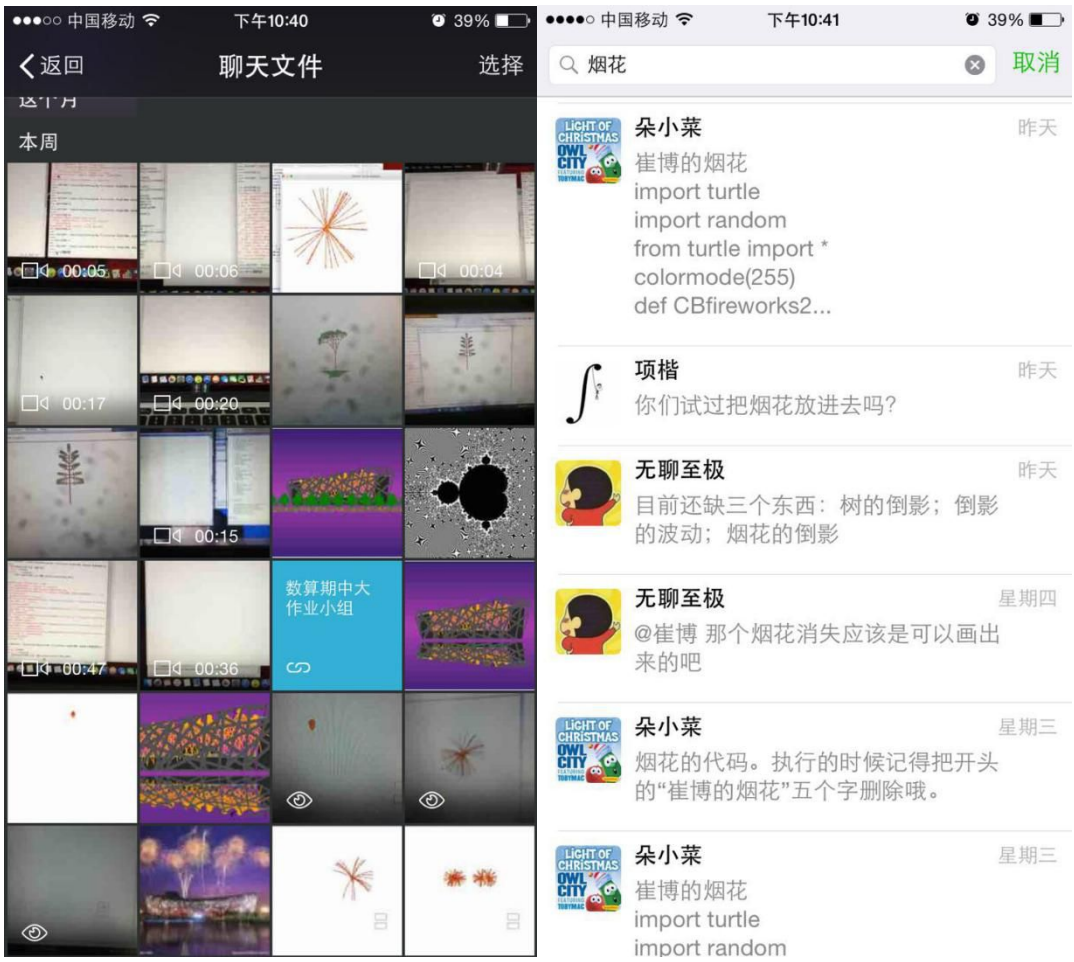
第二次组会：讨论分工和技术学习细节

第三次组会：跟进完成进度，交流面对的问题

第四次组会：完成作品主要成份，准备展示

线上讨论：

仅仅是关于烟花一项，就有数百条聊天讨论记录之多！回顾聊天文件，看到作品逐渐孕育出来，问题一个个被解决，成就感满满！




或许时隔多年之后，依然记得我们熬过的夜，我们喝的咖啡。那自习室的灯光，将照亮我们未来的回忆，告诉我们青春的足迹。

## 18.4.2经验与教训

1. 多向老师、助教请教

   很多东西我们缺少经验，仅靠网上资源不能成功，要多请老师、助教请教。

   例如：之前安装 Pycairo 和 Pillow 时自己摸索半天都毫无头绪，老师一两句话问题就迎刃而解。

2. 大作业书写前充分熟悉函数

   我们在书写大作业前由于没有充分熟悉 turtle 库里的所有函数，导致走了不少弯路，最终效果也不好。

3. 分工计划不佳

   对大作业的分工不是很好，导致没有充分发挥每个人的作用，效率较低

4. 不抛弃不放弃

即便在最后的日子里，我们还想做很多工作，但是没有完成我们就不能放弃。我们也曾一起熬夜编代码，那是名为倔强的日子。

### 18.4.3建议与设想

组队方面：每队中都有乐于写代码的人，都有善于写报告的人，也总有能学习新东西的人。我们分组的主要目的应该是激发每个人的潜力，而不是一味“抱大腿”。范冰冰能说“我不嫁豪门，我就是豪门”，我们也应该有“我不抱大腿，我就是大腿”的无畏无惧。在我们组中，大家的水平参差不齐，既有 quickbird 的大腿组长，也有不善编程的小白。但经过我们分工合作，大腿们努力编程，编程小白也开始学会如何去看文档，如何获取新知识，这是我们的收获，是我们的胜利！

如果给出建议的话，还是希望组队规模能更小一些，让大家能有更自由的发挥空间。

创意方面：更应鼓励多样化的创意，在课堂作业介绍时应当给予指引。例如，本次作业如果画出分形图形，并能介绍分形几何的变化，就能将课堂展示提高一个档次。

展示方面：希望第一次展示的时候，各组在耍宝和卖关子之余，可以通过实际内容的构想给其他小组拓展他们的创意更多的启发，增加落地内容的分享，希望不要让展示喧宾夺主，让数算回归代码。不过同时大家以多种多样的形式各领风骚，也算是一堂精彩的展示课。

寄语：就在你所在的地方生根发芽，开出漫天七夜繁花，这是你们的舞台，你，准备好了吗？！

大胆设想：在实习作业结束后，能进一步了解 python 的画图工具，能够在不久的将来用更好的方法展示更漂亮的鸟巢！！！

## 18.5致谢

感谢陈斌老师给了我们这次小组合作完成有趣大作业的机会
感谢助教，谢谢未名一塔湖图的好榜样，给了我们非常多的启发。
感谢陈斌老师和易超助教、陈春含助教在我们遇到困难时对我们的帮助和启发。
最后，也是最重要的是，感谢所有成员热情地出谋划策，尽心尽力地完成自己的任务，团结协作，不怕困难，完成了这件说不上优秀但却凝结了我们心血的大作业

## 18.6参考文献

1.python 官方手册——turtle 部分
https://docs.python.org/3.3/library/turtle.html#turtle.hideturtle

2.脚本之家
http://www.jb51.net

3.pillow 官方使用手册
http://pillow.readthedocs.org/en/latest/reference/Image.html

4.电脑编程网
http://biancheng.dnbcw.info/python

# 19 Y 组

# 数据结构与算法课程

## 递归视觉艺术实习作业报告

## 《但是你没有》

周哲* 仲子奇 方先君 濮心翼 万紫荆 李华成

摘要：谢尔宾斯基三角形让我们发现对一个图形进行自相似的无限分割，可以分到非常高的精度，我们想到可以对一张图片进行同样操作。而栅格数据结构则告诉我们将图片细分，对每一个小格子填色，就可以高精度的还原出图片的原貌，最终我们选取了“但是你没有”这一温情漫画，展现美好的爱情。

关键字：递归 自相似 谢尔宾斯基三角形 python 作图 漫画 长方形

## 19.1创意过程与递归思路

### 19.1.1作品总体介绍

我们组的作品的主题是“但是你没有”，我们让小海龟用一系列的漫画，描绘出一个温情的爱情故事。主要原理是我们任意给定一张图片，我们就可以利用 Python 中的海龟，快速识别并清晰的画出来。

这其中用到的递归思想并不复杂，就是定义一个函数，让他把一幅图分割为四个长方形，再把每个长方形分成四个小长方形，以此类推，直到分到我们需要的精度为止。海龟识别原图和再现原图都是利用这个递归函数，在识别时，我们用一个列表来储存细分后每个小长

方形的颜色，画图时在一一填上对应的颜色。

最初我们使用三角形作为基本单元，后来发现画图速度过慢，于是改成用长方形作为基本单元，程序运行速度加快了许多。

### 19.1.2创意来源

创意的来源主要是谢尔宾斯基三角形和栅格结构。谢尔宾斯基三角形让我们发现对一个图形进行自相似的无限分割，可以分到非常高的精度，我们想到可以对一张图片进行同样操作。而栅格数据结构则告诉我们将图片细分，对每一个小格子填色，就可以高精度的还原出图片的原貌。关于用怎样的一系列漫画，我们组进行了激烈的讨论，最初想用“友谊的小船说翻就翻”，然而组长认为文字太多，后来又想到可以画“李克强总理到北大”，也被否决。最终我们选取了“但是你没有”这一温情漫画，展现美好的爱情。

### 19.1.3递归思路

把原图片分成四个长方形，把每个长方形再分成四个长方形，以此类推，从这个半完成图中，可以看出递归的痕迹。

例如一种极端情况，图片大小为 $2^{20}*2^{20}$，首先将其分成 4 个 $2^{19}*2^{19}$ 大小的区域，再将 4 个区域分成 4*4 个 $2^{18}*2^{18}$ 大小的区域，直至划分后的区域边长<2 或 3，

以其中一个坐标对应的颜色作为整个小区域的颜色，从而画出整个图片。

## 19.2程序代码说明

### 19.2.1数据结构说明

在程序开头，除去必须用到的 turtle 模块，还引入了两个新的模块，分别为 numpy 和 image 模块。主要通过 image 模块完成对图片中每个坐标像素点的 RGB 值进行扫描，然后通过 numpy 模块中的多维数组对象对图片像素点的 RGB 值进行简化运算，最终通过 turtle 表现出新的较为清晰的图片。

下面介绍一下算法中采用的新的数据结构：numpy 数组。

numpy 数组是一个多维数组对象。其由两部分组成：实际的数据以及描述这些数据的元数据。大部分操作仅针对于元数据，而不改变底层实际的数据。关于 numpy 数组有几点必需了解的：numpy 数组的下标从 0 开始，同一个 numpy 数组中所有元素的类型必须是相同的。

numpy 数组属性，numpy 数组的维数称为秩（rank），一维数组的秩为 1，二维数组的秩为 2，以此类推。在 numpy 中，每一个线性的数组称为是一个轴（axes），秩其实是描述轴的数量。比如说，二维数组相当于是两个一维数组，其中第一个一维数组中每个元素又是一个一维数组。所以一维数组就是 numpy 中的轴（axes），第一个轴相当于是底层数组，第二个轴是底层数组里的数组，而轴的数量——

秩，就是数组的维数。

在这个算法中，由于处理的 RGB 值为（x，y）格式，所以采用的 numpy 数组的秩为 2，通过这种数组，可以对 RGB 值进行简单的加减运算及向量式的收缩运算。

### 19.2.2函数说明

1. 普通方法：

abs(x)：函数返回 x（数字）的绝对值或向量的模，在本次算法中，主要采用后者。

2. turtle 模块中的方法：

forward(n); backward(n)：爬行

left(a); right(a)：转向

penup(); pendown(); pensize(); pencolor(c)：笔触

turtle.Turtle()：生成一只海龟

turtle.Screen()：获取屏幕对象，用于最后的点击自动关闭窗口

hideturtle()：将海龟隐藏

w.exitonclick()：作图完毕，欣赏结束后可以点击关闭窗口

3. image 模块中的类方法：

getpixel(self,x,y)：获取图片中坐标（x，y）像素点的 RGB 值。参数格式为（x，y）

size(self)：获取指定图片的长和宽，以图片左上角为原点，竖直向下为 x 轴正方向，水平向右为 y 轴正方向。若对于图片 p，M=p.size()，

那么 M(0),M(1)分别表示图片的长和宽。

4. numpy 模块中的方法：

array()：创建 numpy 数组。使用 array 函数创建时，参数必须是由方括号括起来的列表，而不能使用多个数值作为参数调用 array，可以用双重序列来表示二维的数组，三重序列表示三维数组，以此类推。也可以在创建时显式指定数组中元素的类型。通常，刚开始时数组的元素未知，而数组的大小已知。因此，numpy 提供了一些使用占位符创建数组的函数。这些函数有助于满足数组扩展的需要，同时降低了繁杂的运算，缩短图片处理的时间。

5. 新定义函数：

dist 函数：用于计算两点间的距离，便于后面对两点间差距的判断。

scan 函数：通过对多个坐标的递归计算，找出分割后小区域的代表坐标，再读取其像素值充当这一小区域的像素值，若将这一过程颠倒，程序会因占用内存过大而崩溃。

其主要思想是对一个区域的划分，例如将一个长方形区域等分成四个小长方形，再对四个小长方形进行分割，直至其足够小。

该函数有 7 个参数，其中 p0，p1，p2 表示长方形的三个拐角的坐标，可以确定一个长方形；T 为列表，用于储存扫描后的数据，包括最小长方形的三个拐角坐标和小长方形中的颜色；pic 为待处理的图片；R= [1,-1]，用于转变坐标系，使 size 函数的坐标转变成 turtle 中的正常坐标；A 用于将图片转移到程序运行界面的中央。具体如下：

```
M = pic.size
T = []
R = numpy.array([1,-1])
A = numpy.array([-M[0]/2,M[1]/2])
p0 = numpy.array([0,0])
p1 = numpy.array([0,M[1]-1])
p2 = numpy.array([M[0]-1,0])
```

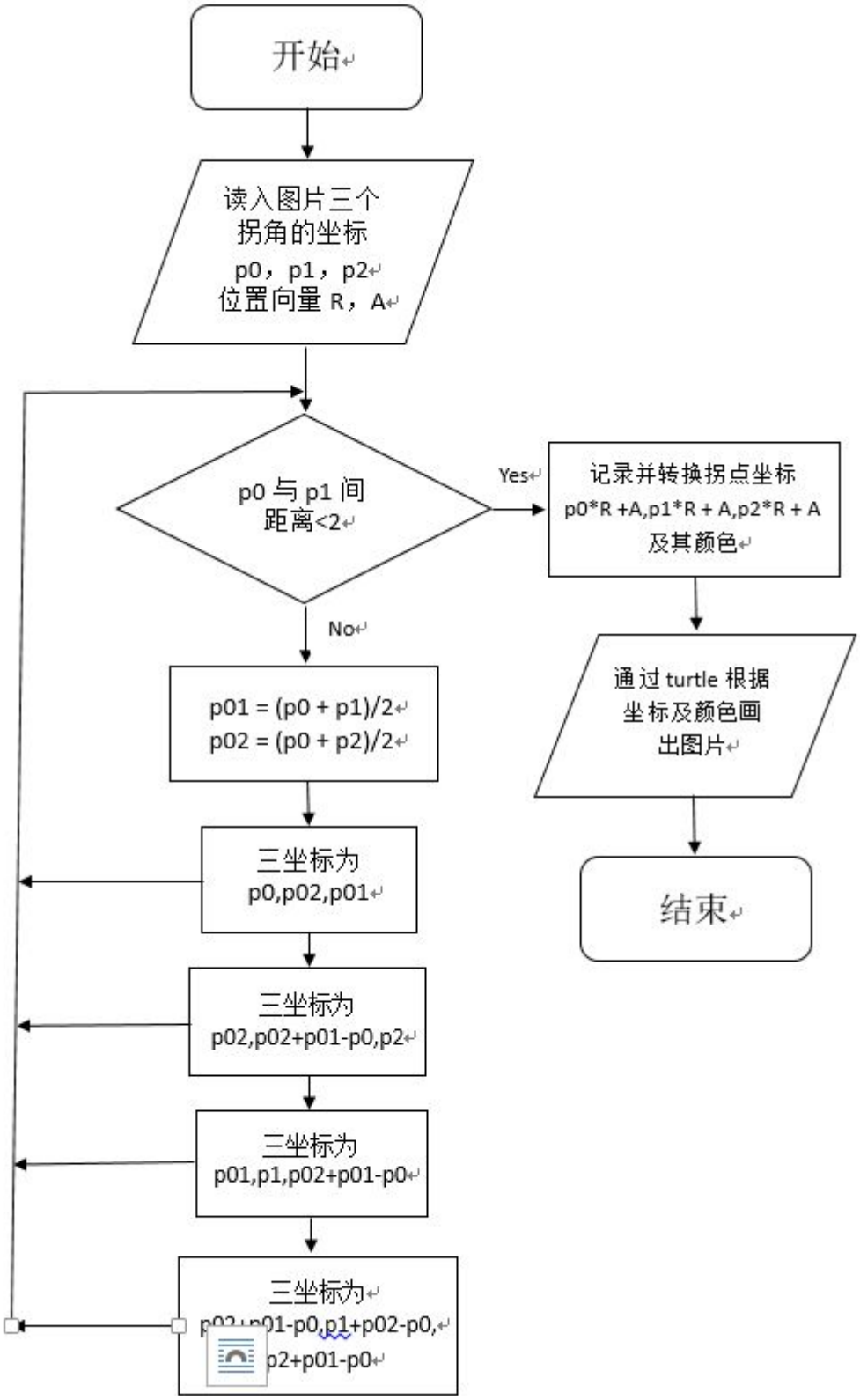
下面为完整的 scan 函数，若将 dist 上限减少，可增加作图的准确度。

```
def scan(p0,p1,p2,T,pic,R,A):
    if dist(p1,p0)< 2 and dist(p2,p0) < 2 :
        T.append((p0*R +A,p1*R + A,p2*R + A,pic.getpixel(tuple(p0))))
    else:
        p01 = (p0 + p1)/2
        p02 = (p0 + p2)/2
        scan(p0,p02,p01,T,pic,R,A)
        scan(p02,p02+p01-p0,p2,T,pic,R,A)
        scan(p01,p1,p02+p01-p0,T,pic,R,A)
        scan(p02+p01-p0,p1+p02-p0,p2+p01-p0,T,pic,R,A)
```

### 19.2.3程序限制

如果图片尺寸过大（尺寸小，分辨率高，程序也可运行），程序会因为占用内存过大而崩溃，出现下面的错误。

## 19.2.4算法流程图


开始
读入图片三个
拐角的坐标
p0，p1，p2
位置向量 R，A
p0 与 p1 间
距离<2
Yes
记录并转换拐点坐标
p0*R +A,p1*R + A,p2*R + A
及其颜色
No
通过 turtle 根据
坐标及颜色画
出图片
结束
p01 = (p0 + p1)/2
p02 = (p0 + p2)/2
三坐标为
p0,p02,p01
三坐标为
p02,p02+p01-p0,p2
三坐标为
p01,p1,p02+p01-p0
三坐标为
p01-p0,p1+p02-p0,
p2+p01-p0

## 19.3实验结果

### 19.3.1实验数据

实验环境说明：

- 硬件配置：CPU: Intel Core i5-5200U 2.20GHz \ 8GB RAM
- 操作系统：Windows 10 专业版
- Python 版本：python 2.7.11

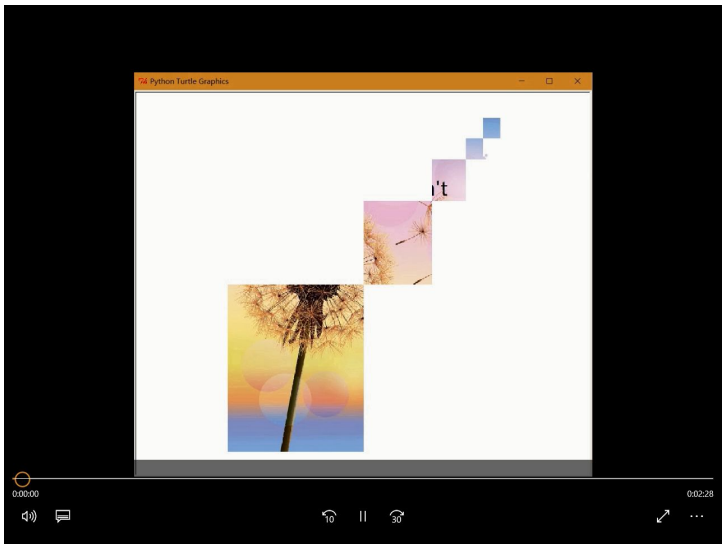
### 19.3.2作品描述

**作品名称**：《但是你没有》

通过 Turtle 作图模块，利用递归的思路来将导入的图片用海龟画出来。如果将一组漫画导入 turtle 中，并将作画的过程录下来，再配上乐，就可以用 turtle 来讲故事啦！

我们选用的是一首感人的爱情诗《但是你没有》，这首诗的作者是一位普通的美国妇女，她的丈夫应征去了越南战场，后来不幸阵亡了。这个噩耗对她无疑是一种心灵上最痛苦的折磨，但是她不得不面对现实，但丈夫生前对她的爱一幕幕涌现，使她决定终身守寡，最后年老病逝。有一天，她的女儿在整理遗物时发现了母亲当年写给父亲的这一首未曾发出去的情诗，便是这首《但是你没有》。

**作品展示：**




• 选取其中的一帧处暂停，可以看到递归的自相似效果。


• 诗名《但是你没有》


• 随着《kiss the rain》凄美的音乐声响起，turtle 逐渐画出了第一幅漫画：记得那一天，我借用你的新车，我撞凹了它，我以为你一定会杀了我的


• 但是你没有


• 记得那一天，我在你的新地毯上吐了满地的草莓饼，我以为你一定会厌恶我的


• 但是你没有

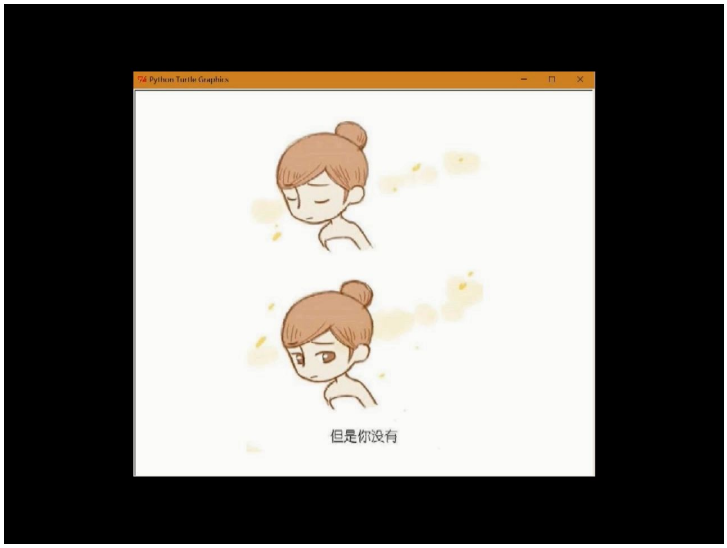

• 记得那天，我拖你去海滩，而它真如你所说的下了雨


• 我以为你会说“我告诉过你”


• 但是你没有


• 记得那天，我忘了告诉你那个舞会是要穿礼服，而你却穿了牛仔裤


• 我以为你一定要抛弃我了




• 但是你没有

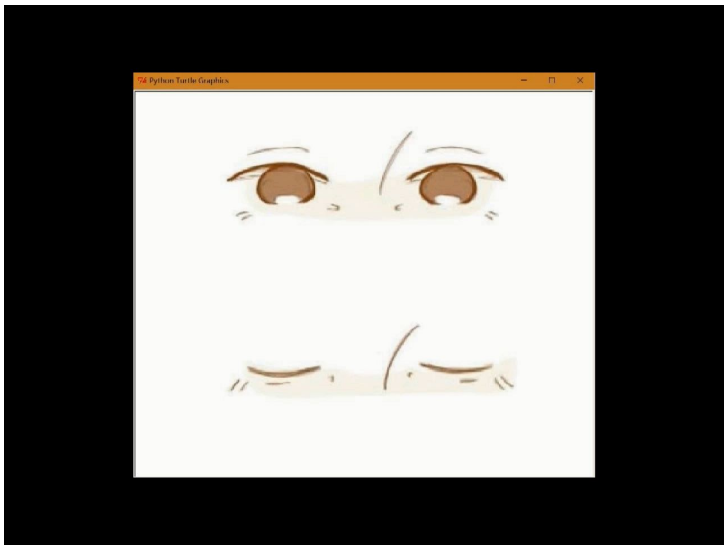

• 是的，有许多的事你都没有做


• 而你容忍我


• 钟爱我 保护我


• 有许多的事情我要回报你


• 等你从越南回来




• 但是你没有

（完）

以上就是我们展示的漫画了，运用递归算法画出来的漫画不但可以展示内容，而且自带渐入效果，利用视频编辑软件反转一下后还有渐出的效果，从而造成一种连环的感觉。

### 19.3.3实现技巧

1、可以通过调节 tracer（）函数来设置 turtle 模块每走 2000 步刷新一次，这样占用缓存比较少，看起来画的比较快一点

2、可以设置每画 600 个正方形之后删除该海龟，并生成一个新的海龟，这样可以释放内存，加快作图。

3、运用 numpy 科学算法模块来替代自定义的向量运算法可以大大节约运算时间。

## 19.4实习过程总结

### 19.4.1分工与合作

分工：组长及台上展示：周哲

编程：李华成

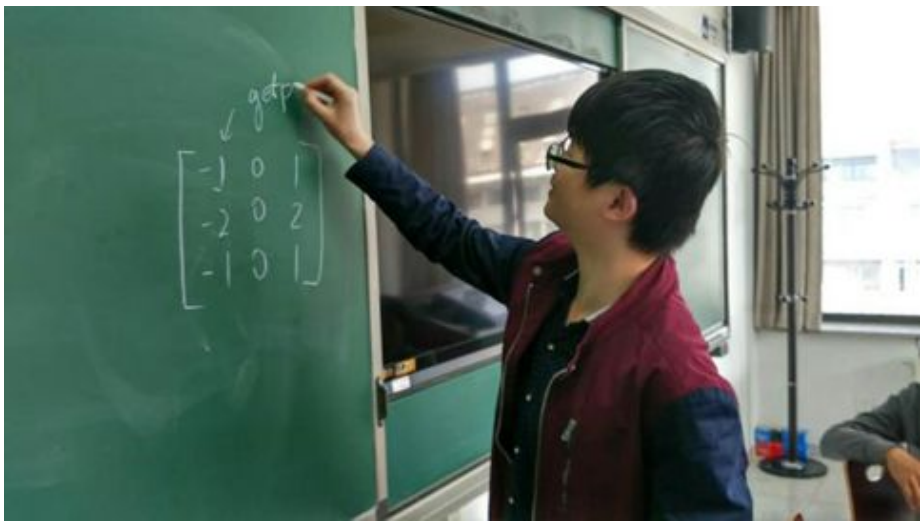
ppt 制作：万紫荆

报告撰写：仲子奇、濮心翼、方先君

创意来源及表现：全体组员

合作：微信交流（包括素材收集与分享、代码解释）、当面交流（即

组会）、课下交流（在各种课后回宿舍路上的交流）



第一次组会（李华成大神在讲解他的思路）

第一次组会（组员们在用组长的电脑看初步程序）

第二次组会（程序的修改与完善及为初步展示做准备）

第三次组会（讨论最终展示的形式以及脑洞碰撞，时间最长的组会大家一起做着各自负责的事为最后展示的视频及报告及 ppt 准备）（都没有看镜头）

## 19.4.2经验与教训

得意之处：

1.作业要求我们必须以递归算法为核心主题。相比于一些为了满足这个要求强行将循环改为递归的较为简单的想法，我们小组没有采用，可谓是一开始就是坚决以递归为主题的以谢尔宾斯三角形为灵感，

又红又专。

2.从作业要求中我们可以看到内容形式可以选择分形结构、艺术图像、动画视频、交互作品等。受上届画树的影响，我们组长预测肯定会有相当数量的小组会选择继续画树或者画与树相关的花花草草之类的，再加点个人元素。果不其然，在第一次展示上确实有很多小组选择画树及相关物体。我们得承认他们的确实很美，但是我们想突出的是我们组的创意，独特不重复。同样是画图，我们想的是利用谢尔宾斯三角形对总区域的递归划分来得到足够小的区域从而能以各种效果画出图片及抹掉图片。

3.为了更好的表现我们组的创意效果我们还展开讨论要以什么样的形式做什么样有故事的视频来展现我们小组的创意。这种视频故事的创新更是一种绝佳的表现方式。

改进之处：

1.谢尔宾斯三角形的思路依旧有着缺陷。第一，它画的为三角形，需要画中间的一条斜线，所花费的时间更多，第一次跑的时候花了几十分钟上小时的时间。第二，我们采用的是自定义的向量计算方法，同样拖慢了时间。通过思索，编程大牛将三角形改为长方形，这时 turtle 将只用转 90 度来作图。以及发现可以通过调用模块来进行科学计算。这时画图所用时间大大减小。

2.画图效果单一。一开始我们只从右下角开始画左上角结束。后来经过讨论大家认为就像 ppt 的图片插入与消退效果一样，我们的 turtle 画图也应该能实现各种形式的画图以及各种形式的抹图。这样

视觉效果会更好。以一个代码为母本能演化出多个代码，能根据心意改变画图部分的顺序来隐藏信息等等。

### 19.4.3建议与设想

建议：

1.关于组队。仍存在大腿抱堆、编程新手面面相觑的情况。为了能更加让每个小组均衡发展，建议先根据之前作业情况确定组长，不采取自报组长的制度，然后组员自报这样的形式。

2.关于创意展示。老师在提树的同时可提其他别的创意递归，避免大家被树的固有思维囿住。

3.关于寄语。学弟学妹们，数算这门课程是对能力要求蛮高的一门课程。可以在做期中大作业之前多练练 python，多看看课外的东西，在作业中多体现自己的想法。

设想：

1.大众投票评比。可以选出最具人气的以及逼格最高的以及最有情调的 python 期中大作业作品，这个可以在期中大作业里占分，从而促进大家投入的热情和编程展示效果的提升。

2.作品集。将优秀的作品和写成集。

## 19.5致谢

感谢陈斌老师半学期的授课指导，帮助答疑。

感想各位助教对我们在 Python 上的引导和帮助。

感谢 Y 小组所有成员的辛勤付出，我们通过团队合作、各司其职成功地完成了这一次大作业。

## 19.6参考文献

http://www.wtoutiao.com/p/Qa86vO.html（视频素材来源）

https://docs.python.org/2.7/library/turtle.html（程序基础知识来源）

# 20 Z 组

# 数据结构与算法课程

## 递归视觉艺术实习作业报告

### (《魔法的城堡》)

（张家港*，刘宇飞，许午川，于曦彤，赵兴鑫，赵泽严）

**摘要**:《魔法的城堡》是由北京大学地球与空间科学学院 15 级本科生团队所完成的作品。该作品以迪士尼城堡为主题，仿照迪士尼电影公司所设计的动画，采用递归方式加以制作而成的，整体画面由基准塔、星点、烟花、艺术字以及幕布几个要素构成。这是 Z 团队从美学、心理学、几何学、信息科学、电影制作与处理学等角度考虑，用时三周竭力打造的大型卡通华美巨制，倾心打造，欢迎观赏！

**关键字**：递归、城堡、迪士尼、梦幻

## 20.1创意过程与递归思路

### 20.1.1作品总体介绍

总体宏观分析全局，整个作品的灵魂所在在于城堡的堆砌，以基准塔为核心，以递归为指导思想，通过对基准塔大小的调整以及位置的改变，加以修整，进而完成整座城堡的制作，辅助以星点、烟花、艺术字、渐变色幕布等，使整幅图景更具奇幻色彩。

以艺术的眼光看待这幅作品，确有独到之处：城堡错落有致，极具文艺复兴时期威尼斯当地画风特色；星点随意洒落天际，简洁而不失风雅；烟花则带来的是欢动的气氛，给作品增

添了一抹亮色；艺术字与渐变幕布是最后的画龙点睛之笔，升华整个作品。迪士尼城堡这幅作品集聚视觉美观性、布局合理性、拓扑关系明确性于一体，贴近生活，画风独到创新，不失为是浪漫主义画派之佳作、数据结构与算法课程之良品。

### 20.1.2创意来源

所谓创意，不单指无中生有的能力，可以将陈腔滥调写成别有洞天也是一种境界。而我组的创意，正是将不起眼的城塔楼阁堆砌成一幅奇景，一幅绚丽与内涵并存、华美与情怀相济的奇景。

基于已有的递归模型，我们起初的设想便是动画主题，完成一份富有童年色彩的作品。从数码宝贝的进化光圈到龙珠的龟派气功波、天津饭的分身等，我们曾设想了许多种独具创意的方案，但由于人物绘画具有一定的局限性，再加以美观性、可行性、时效性等多方面的考察，都没有得到大家的一致认可，思路一时间陷入僵局。

继而，我们的思路又回归到编程的最本源去，期望可以通过最基本、最直接的方式去达到最打动人的效果。

线条，建筑，城堡，迪士尼。

创意，有时正是源于最简单、最原始。

寥寥数行代码，运行勾勒笔划。天公抖擞献创意，腐朽蜕变迪士尼。

### 20.1.3递归思路

**作品主要有两处使用递归：**

**城堡**：城堡来自于楼阁规律的堆砌，相同的基准塔经过大小的变化，位置的改变，仅仅通过简单的递归便可成为伟岸的城堡。就像谢尔宾斯基三角形一样，简单的三角形变成形态各异的塔楼。只需要三阶递归，城堡的雏形便可显现。

**烟花**：把一棵树转一圈难道不就是烟花吗？通过不同次数的递归可调节出不同尺寸、复杂程度的烟花，选择适当位置，作为点缀。

## 20.2程序代码说明

### 20.2.1数据结构说明

本程序在数据结构使用上比较基础，在此无须赘述。

自定义类的使用包括多种基准塔和城墙，详情请看程序源代码。

### 20.2.2函数说明

```
def draw_towerls(length, n, x,  y):
        t.up()
        t.goto(x,  y)
        t.down()
```

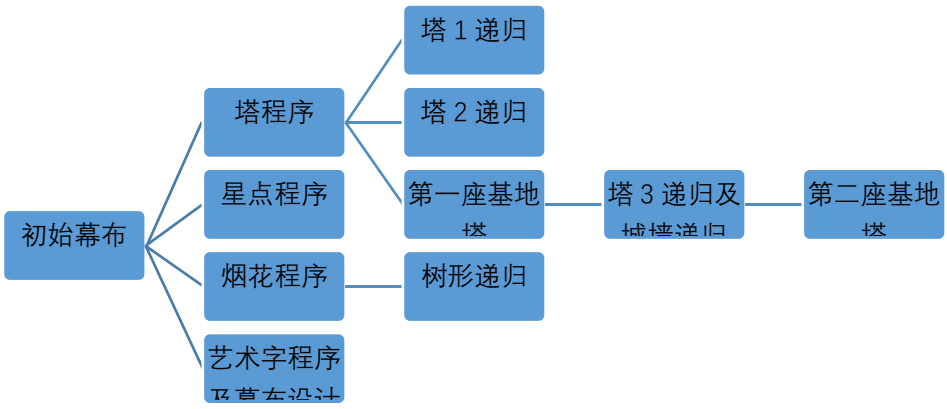
```
        if  n==1:
                tower1(length,  x,  y)
        else:
                tower1(length,  x,  y)
                cheng_qiang(10,  int(7.2*n*n*length/20)+1,  2,  x-3.6*n*n*leng
th+10,  y-7*length)
                draw_tower1s(1.3*length,  n-1,  x-3.6*n*n*length,  y-7*length)
                draw_tower1s(1.3*length,  n-1,  x+3.6*n*n*length,  y-7*length)

def draw_tower2s(length,  n,  x,  y):
        if  n==1:
                tower2(length,  x,  y)
        else:
                tower2(length,  x,  y)
                draw_tower2s(0.6*length,  n-1,  x-10*(4-n)*length,  y)
                draw_tower2s(0.6*length,  n-1,  x+10*(4-n)*length,  y)

def draw_tower3s(m,  n,  x,  y):
        if  n==1:
                tower3(m,  x,  y)
        else:
                tower3(m,  x,  y)
                draw_tower3s(0.7*m,  n-1,  x-m*180,  y-m*200)
                draw_tower3s(0.7*m,  n-1,  x+m*180,  y-m*200)
```

以上是三座塔的递归方式，通过调节参数可以对塔的位置、尺寸等进行调整。

## 20.2.3程序限制

**Caution**:Turtle 初始方向或是坐标设定异常会导致城堡扭曲或位置错乱。

## 20.2.4算法流程图



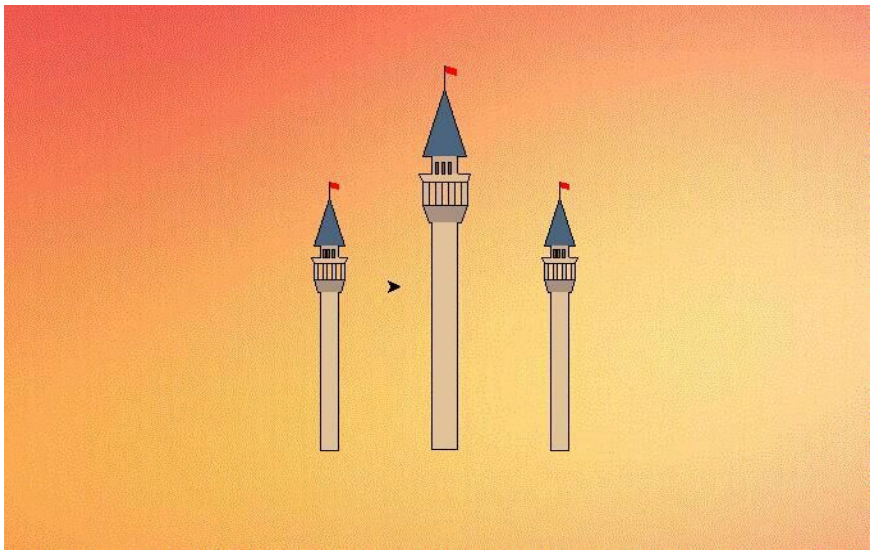
# 20.3实验结果

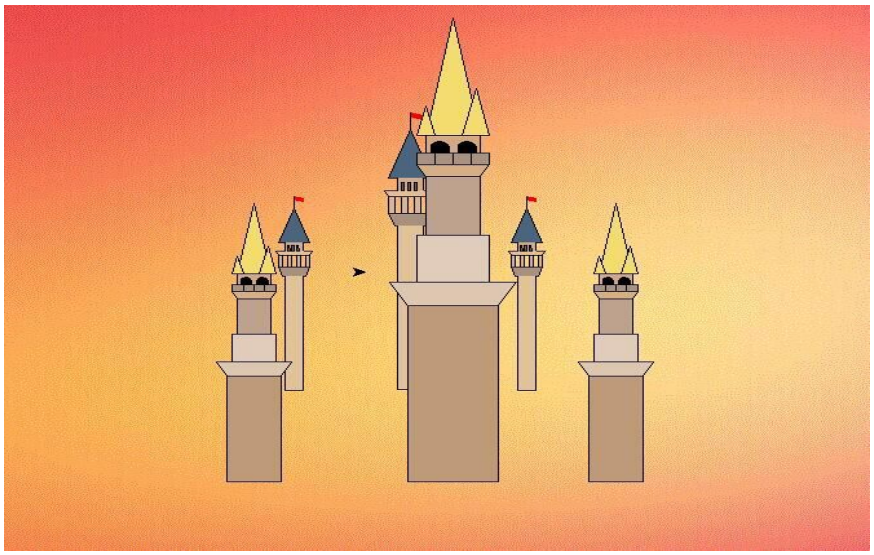
## 20.3.1实验数据

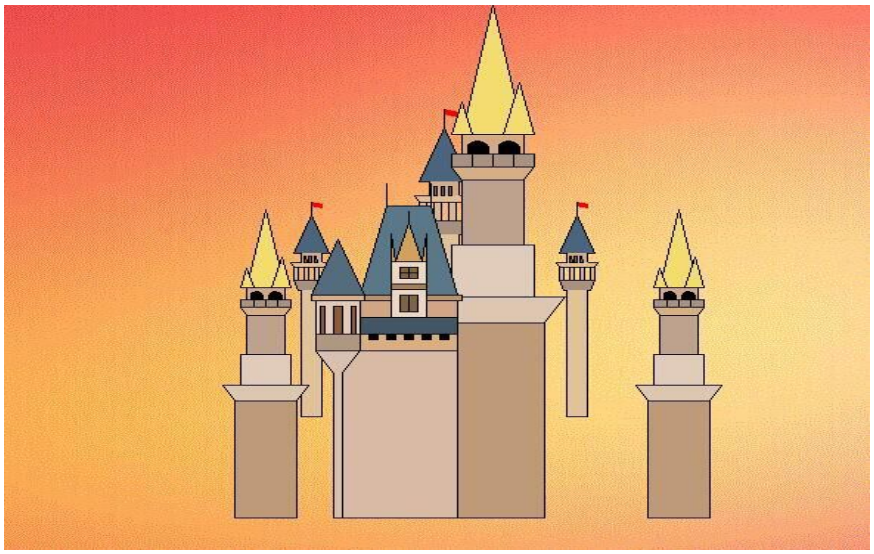
实验环境说明：

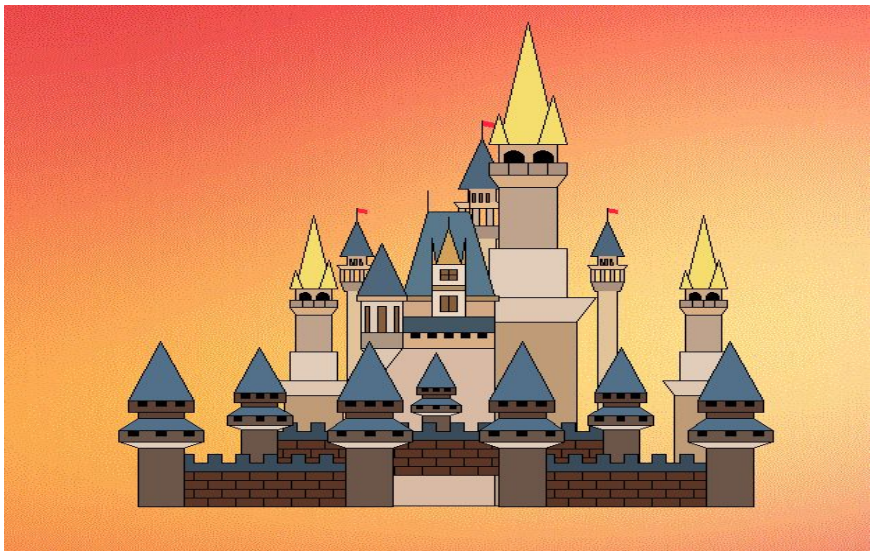
- 硬件配置： 2.2 GHz Intel Core i7/16 GB 1600 MHz DDR3
- 操作系统： ios X El Capitan
- Python 版本： python 3.5.1

## 20.3.2作品描述










以上是《魔法的城堡》程序运行时的全过程，将多部分自定义类所完成的图形叠加在一幅图上，通过对位置与尺寸的调节而得到的精美的画作。

### 20.3.3实现技巧

**三原色取色**：通过 photoshop 对原图进行色度临摹，选取某些特定点记录下该点 RPG 值，用于最后城堡等建筑物的取色。
**艺术字描点**：通过对原字体位置坐标的记录，细密取点，描绘出与原字体极为相近的艺术字。

## 20.4实习过程总结

### 20.4.1分工与合作

组长：张家港
编程：刘宇飞、许午川、赵兴鑫、赵泽严
文案：于曦彤
展示及创意：全体成员

第一次组会（2016/4/4）

第二次组会（2016/4/10）

第三次组会（2016/4/16）

### 20.4.2经验与教训

谈及经验，且说个人三周以来的体会吧。

何谓“大腿”？何谓“小腿”？皆是相对之谈。不论基础有无、实力强弱，期中大作业的初衷在于锻炼自我编程、组织、文案等各方面的能力，并非“抱大腿”即可躲避了事。大腿聚堆，组内各自分工，或许不能完全发挥各自的强项，达不到提高能力的最初目的；没有基础的同学们反而可以在没有大腿的情况下从零做起，通过不断地探索来丰富知识、提升能力，各展所长，发生更神奇的化学反应。大家起初都是零基础的，都需要通过独自摸索、长期训练来获得编程的技巧与能力，希望大家可以通过大作业来提升各方面的能力，披荆斩棘，勇克难关。

大作业完成的顺利与否在于团队协作，在这一点上，多开几次组会、多进行交流是有必要的。分工安排要明确，工作思路要捋顺清楚，定期检查、交流工作成果有利于下一阶段任务的开展。诚然，工作热情高涨固然是好，但需要有组织、有条理地完成任务，在某个工作纲领的指导下，方能化激情为时效，羽化而登仙。

### 20.4.3建议与设想

虽说建议在精不在多，但希望以下建议可以对这门课程日后的开展有所帮助。

组队方面，从防止大腿聚堆的角度来看，以后可以以寝室为单位，一至两个寝室为一组，方便交流且具有随机性，或是根据学号分组，都可以起到随机分组的效果。

创意方面，这学期的递归大作业相对上一年已有了大幅度的提升，由于不只限制于树与风景画，充分发挥了同学们的想象力，大家的作品会对学弟学妹们有很大的启发，他们的想象空间会更宽阔。

展示方面，展示形式可以多样化，例如：编排话剧，将程序所展示的内容编程故事形式，更加生动有趣；引入与展示内容相关的歌舞等节目，会得到意想不到的效果。这样，展示课不仅是学术工作的交流汇报活动，还是一展才华、开拓创新的发展平台。

最后，希望学弟学妹能够不断超越，无畏艰难，奋勇前行，挽狂澜于既倒，扶大厦之将倾，码实用创新之代码，任信息时代之栋梁！

## 20.5致谢

组长：张家港

编程：刘宇飞、许午川、赵兴鑫、赵泽严

文案：于曦彤

展示及创意：全体成员

# 20.6参考文献

http://www.tudou.com/programs/view/Kh1Vu9Hh9fs/