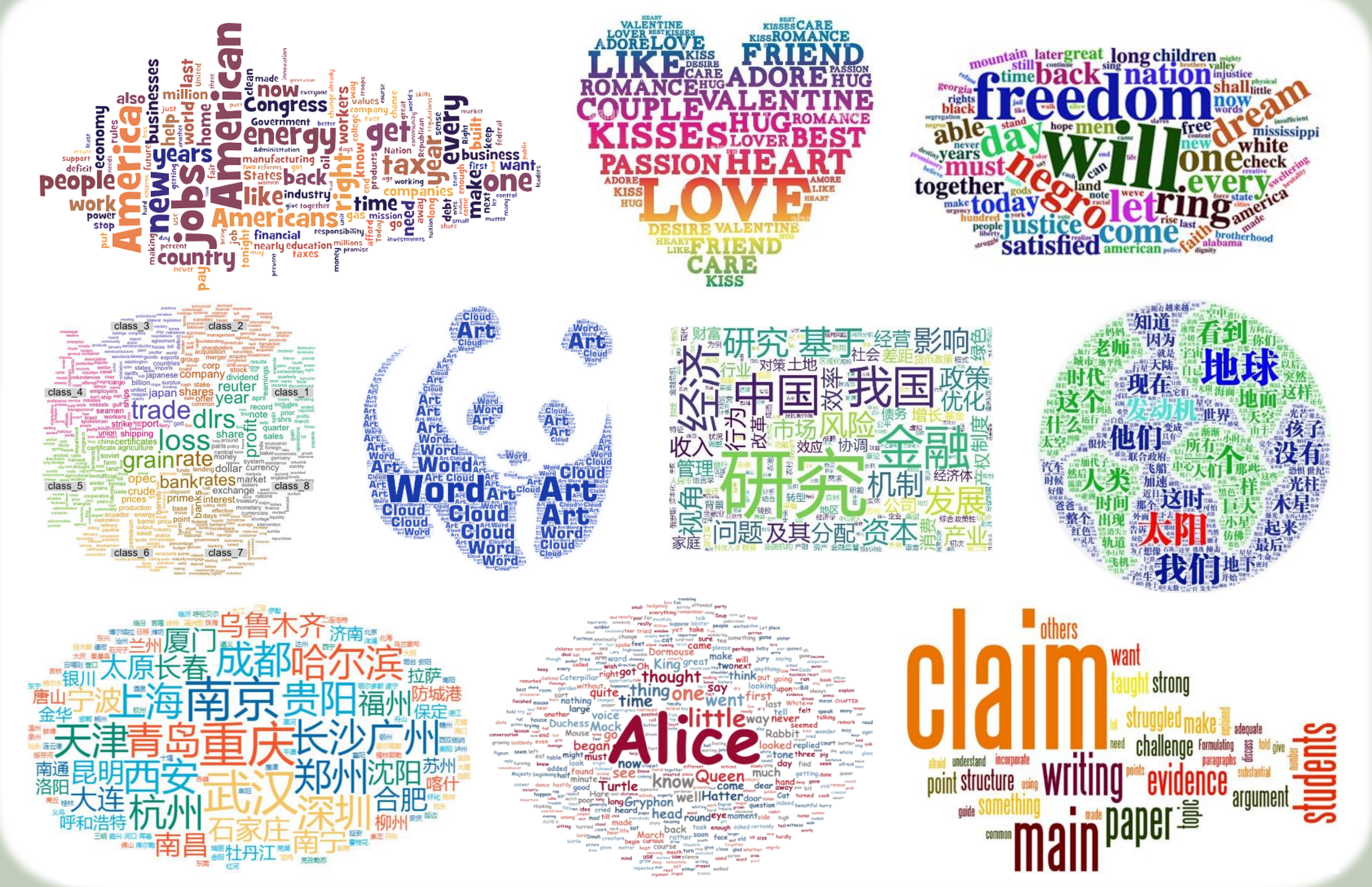
WordCloud.jl

用 Julia 生成漂亮的词云

GuoYongzhi
郭勇智

词云是什么



American
America
jobs
energy
Congress
people
Americans
country
LOVE
KISSES
LIKE
COUPLE
PASSION
HEART
FRIEND
ADORE
VALENTINE
freedom
will
negro
dream
justice
satisfied
Word
Art
Cloud
trade
loss
grain
rate
研究
金融
中国
我国
经济
地球
太阳
我们
南京
上海
重庆
武汉
天津
青岛
Alice
little
claim
main
writing
paper
students
evidence

词云的特点

词云是一种文本数据的可视化方式，一般通过不同的文字大小来体现关键字的重要性。

此外词云的形状，文字的颜色、位置、方向等既可为美观设计也可用于体现一些其它信息。

信息 **可视化的信息量** 单词的大小反映其重要程度

密铺 **词云之所以为云** 单词分布紧凑而不重叠

艺术 **丰富的样式** 各种形状、颜色、字体、角度

WordCloud.jl的特点

WordCloud.jl是完全基于julia开发的一个功能强大的词云生成工具。

项目地址： https://github.com/guo-yong-zhi/WordCloud.jl

准确

单词尺寸严格正比于权重 单词不重复，权重不失真

高效

Julia实现的高效布局算法 充分优化，无惧大图

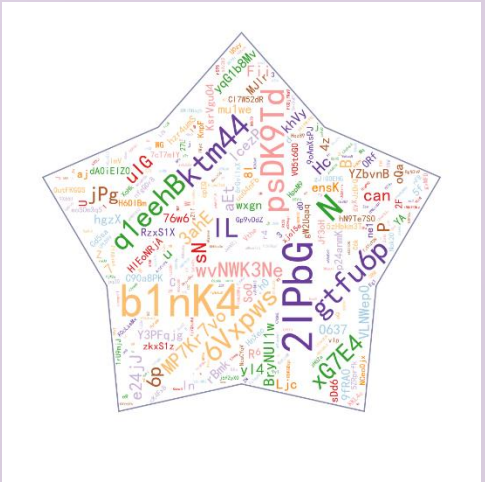
灵活

最少的限制最大的自由 各种风格，一网打尽

- 功能介绍
- 实现介绍

●基本用法 ●第一个例子

```
using WordCloud
using Random
words = [randstring(rand(1:8)) for i in 1:300]
weights = randexp(length(words))
wc = wordcloud(words, weights)
generate!(wc)
paint(wc, "random.svg")
```



生成词云需要的基本输入

- 单词向量 words
- 权重向量 weights

●基本用法 ●不同的方式

- **从字符串**

```
wc = wordcloud("It's easy to generate word clouds") |> generate!
```

- **从文本文件**

```
wc = wordcloud(open(./alice.txt")) |> generate!
```

- **从列表**

```
wc = wordcloud(["中文", "需要", "提前", "分词"]) |> generate!
```

- **从Pair或者Dict**

```
wc = wordcloud(["the"=>1.0, "to"=>0.51, "and"=>0.50,
                "of"=>0.47, "a"=>0.44, "in"=>0.33]) |> generate!
```

●丰富的自定义

```
using WordCloud
stopwords = WordCloud.stopwords_en ∪ ["said"]
textfile = pkgdir(WordCloud)*"/res/alice.txt"
wc = wordcloud(
    processtext(open(textfile), stopwords=stopwords),
    mask = ngon,  npoints = 6,
    masksize = (500, 500),
    maskcolor = "black",
    backgroundcolor = 1.0,
    outline = 15,
    linecolor = "#808080",
    colors = (0.95, 0.925, 0.90, 0.825, 0.85),
    angles = 0:90,
    font = "Impact",
    density = 0.7,
    spacing = 0) |> generate!
paint(wc, "alice.svg")
```

自定义

- 停止词
- 掩膜形状
- 掩膜颜色
- 掩膜尺寸
- 背景色
- 轮廓线宽
- 轮廓颜色
- 文字颜色
- 文字角度
- 字体
- 密度
- 间隔

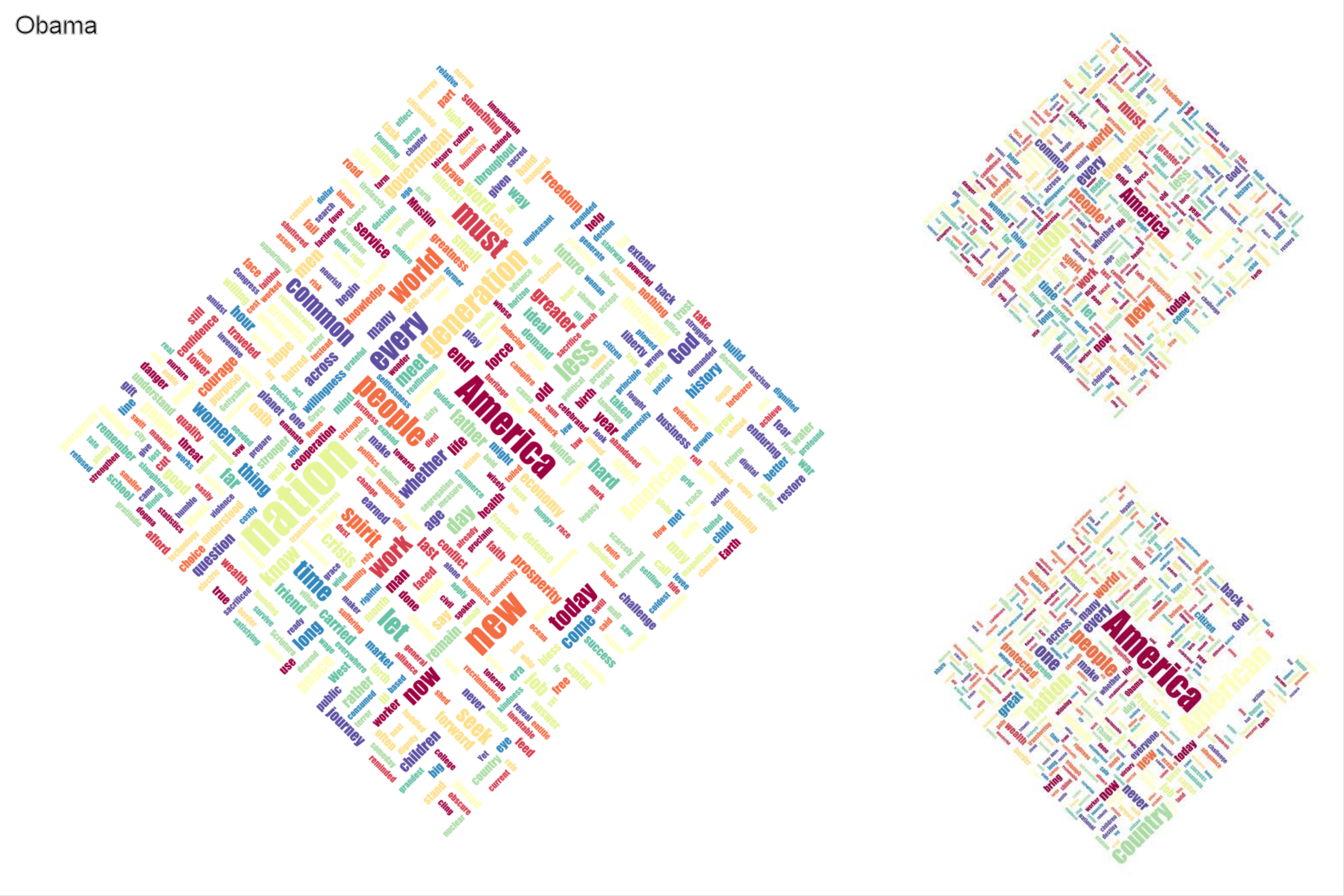
●丰富的自定义

```
using WordCloud
stopwords = WordCloud.stopwords_en ∪ ["said"]
textfile = pkgdir(WordCloud)*"/res/alice.txt"
maskfile = pkgdir(WordCloud)*"/res/alice_mask.png"
wc = wordcloud(
    processtext(open(textfile), stopwords=stopwords),
    mask = maskfile,
    maskcolor = "#faeef8",
    outline = 4,
    linecolor = "purple",
    colors = :Set1_5,
    angles = (0, 90),
    font = "Tahoma",
    density = 0.55) |> generate!
paint(wc, "alice.png", ratio=0.5)
```

自定义

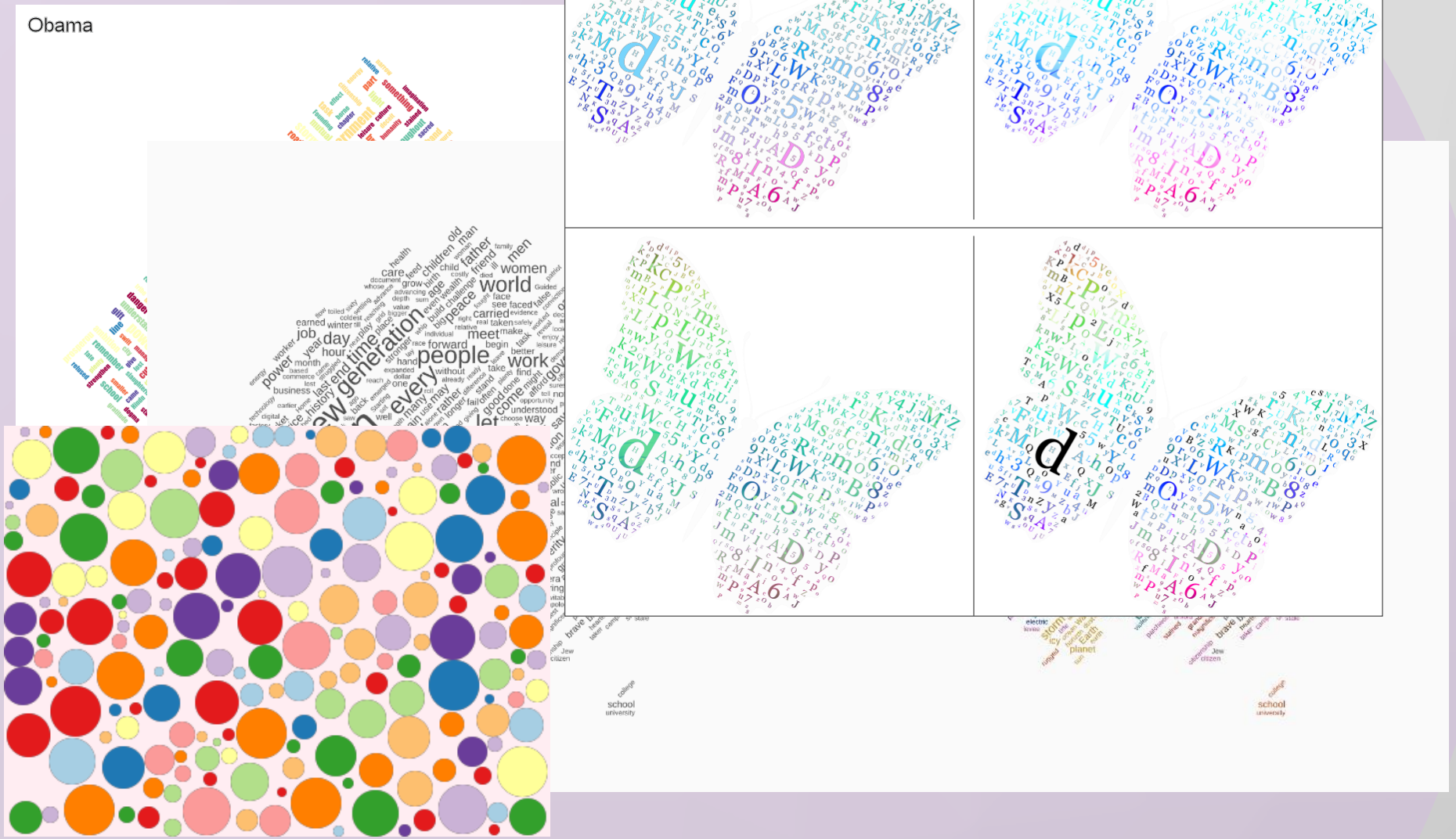
- 停止词
- 掩膜文件
- 掩膜颜色
- 轮廓线宽
- 轮廓颜色
- 文字颜色
- 文字角度
- 字体
- 密度

•不止于此

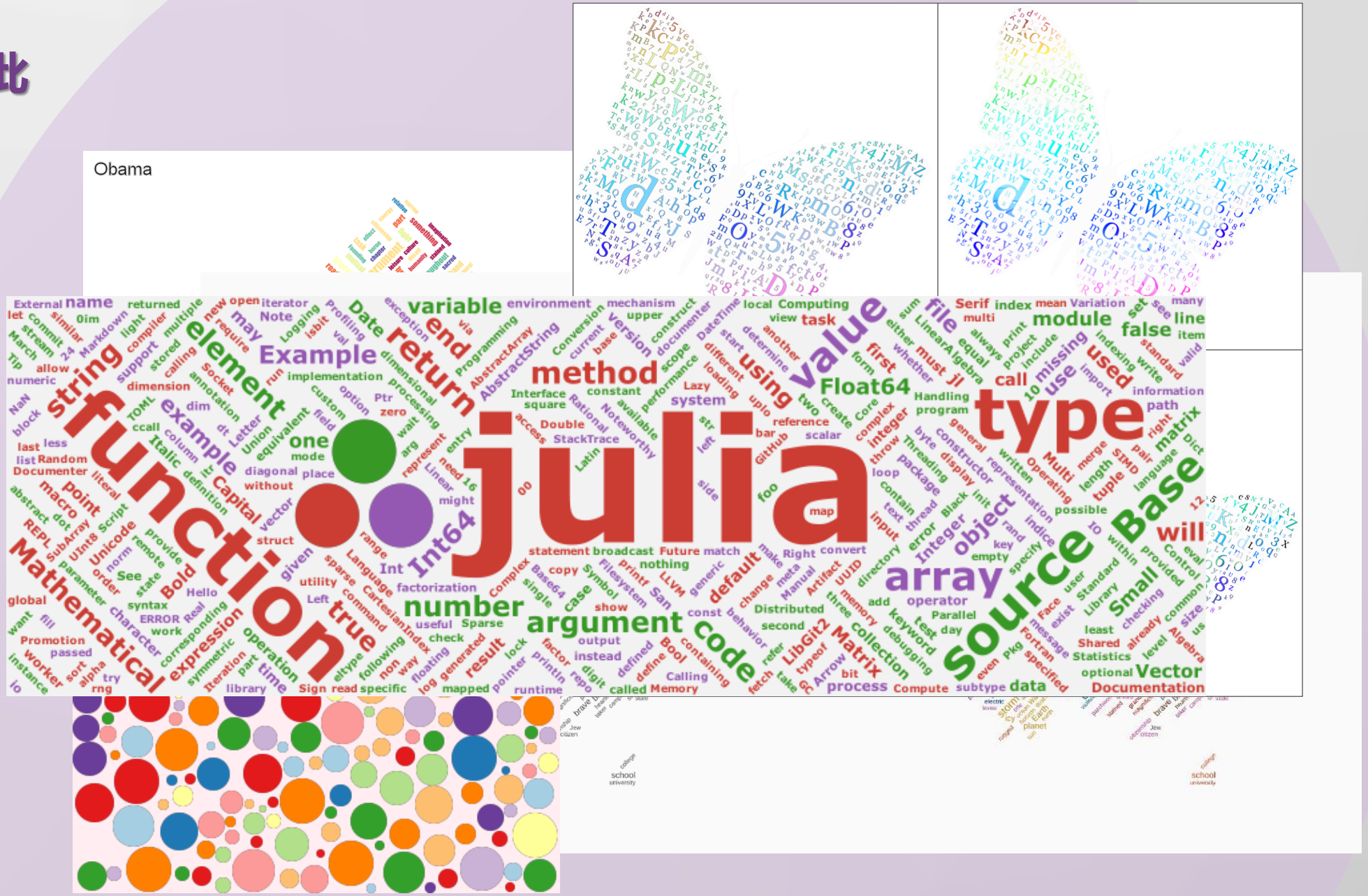

Obama
nation
America
new
people
every
generation
must
common
world
time
today
American
country

●不止于此

●不止于此



•不止于此


Obama
julia
function
Mathematical
string
element
Example
method
value
type
array
source
Base
number
argument
code
Int64
Float64
Matrix
Vector
Documentation
school
university

- 功能介绍
- 实现介绍

●项目整体结构

Stuffing 布局算法	碰撞检测、不规则排料问题
Strategy 策略	密度估计、字号估计、输出尺寸估计
Artist 主题方案	配色、字体、形状等
TextProcessing 文本处理	单词切分、计数、词形还原、大写还原
Render 图形渲染	字体和形状渲染、图片的叠加、填充、描边等

●布局算法

词云布局属于二维不规则排料问题

(nesting problems)

- 常规的贪心算法
  - 螺线法
  - 矩阵法
- 我们的非贪心算法

●常规的贪心算法

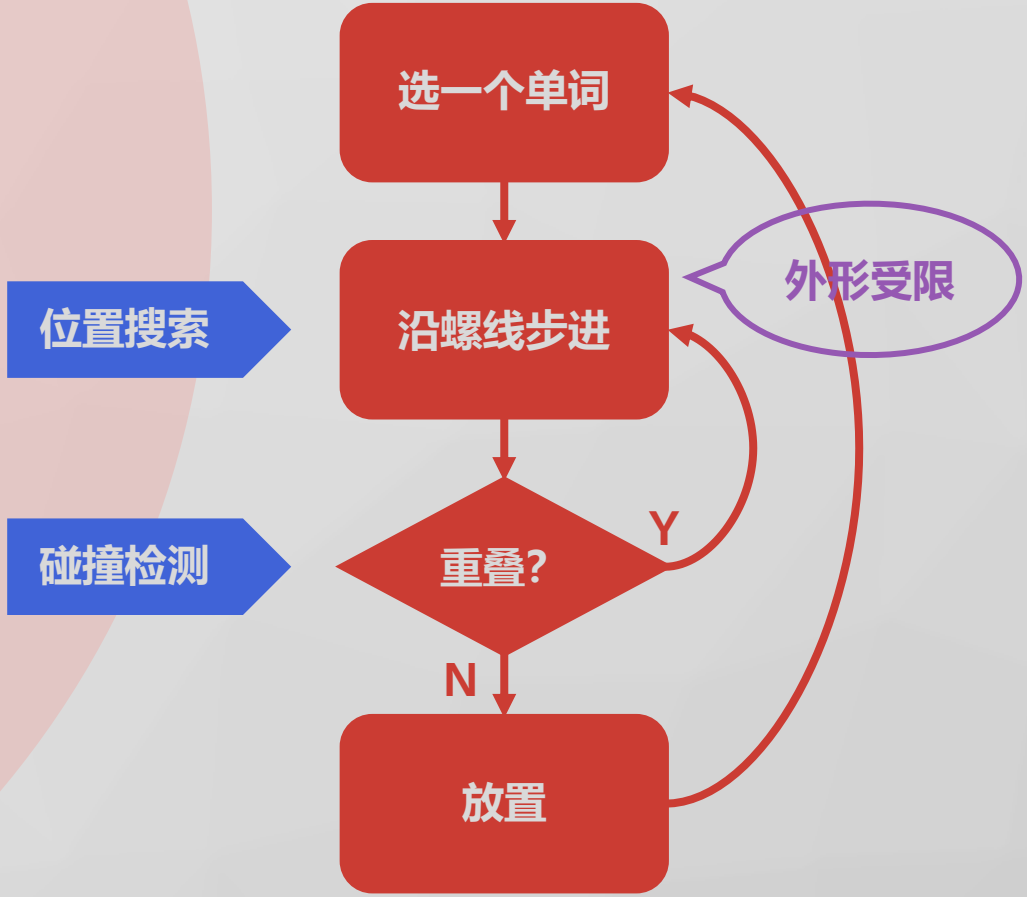
生成词云需要几步?

1. 依次选择未放置的单词
2. 找一个空位
3. 放置

●常规的贪心算法 ●螺线法

生成词云需要几步?

1. 依次选择未放置的单词
2. 找一个空位
3. 放置



●常规的贪心算法 ●螺线法 ●位置搜索

位置搜索

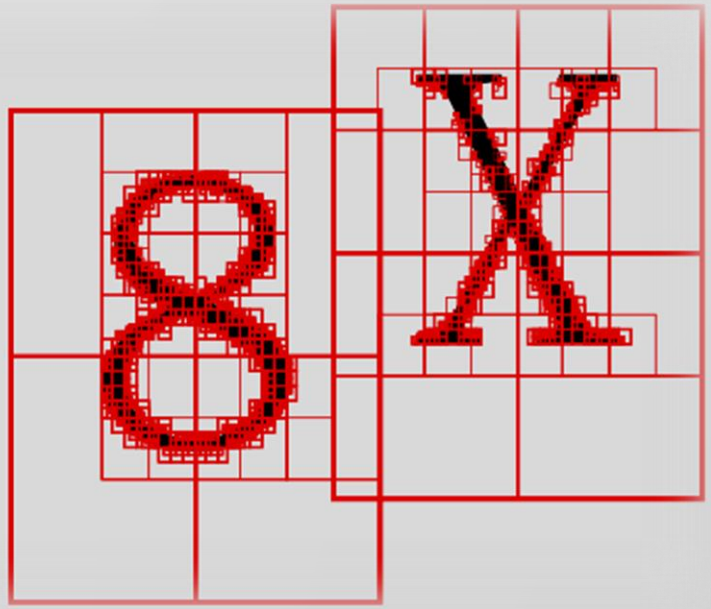
- 沿着螺旋线
- 从中央向外扩张
- 每次挪动一个步长

** Beautiful Visualization. Chapter 3. Wordle. Jonathan Feinberg*

●常规的贪心算法 ●螺线法 ●碰撞检测

碰撞检测

- 层次包围盒
- 链接四叉树
- 自顶而下
- 存储结点坐标

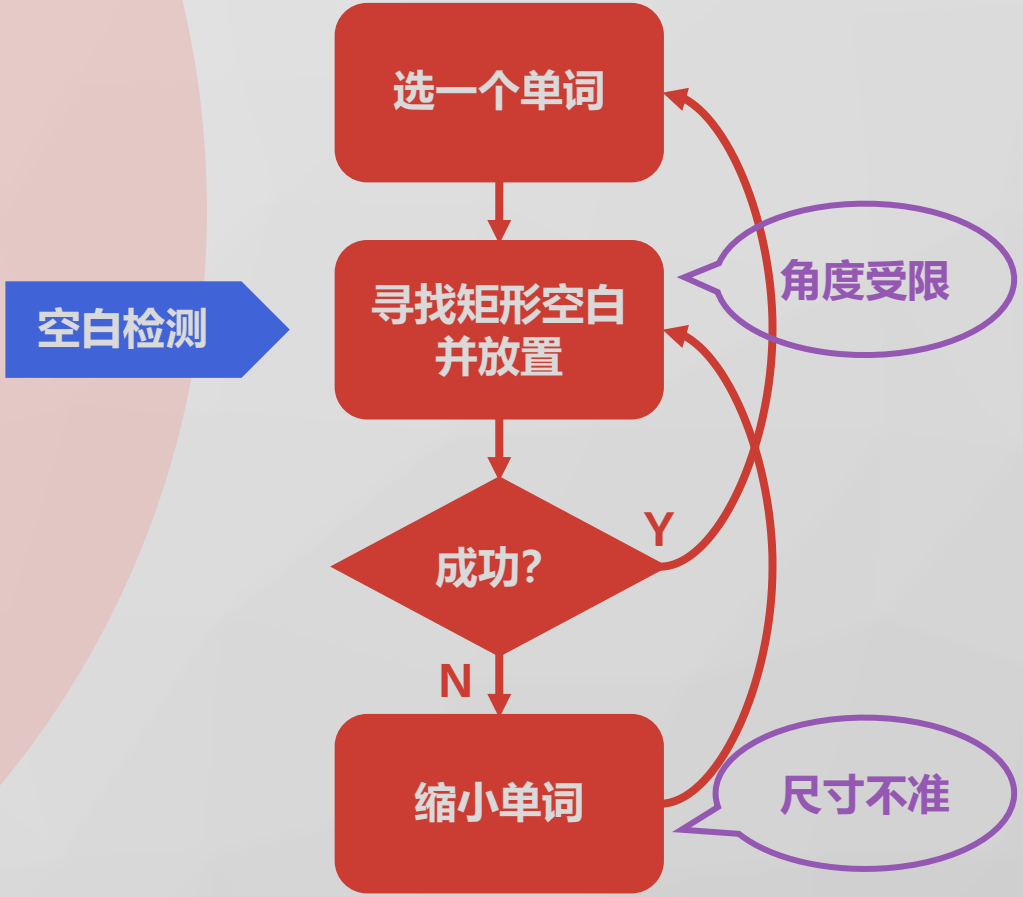


** https://www.jasondavies.com/wordcloud/about/*

●常规的贪心算法 ●矩形法

生成词云需要几步?

1. 依次选择未放置的单词
2. 找一个空位
3. 放置



●常规的贪心算法 ●矩形法 ●空白检测

空白检测

- 积分矩阵
- 二维前缀和
- 仅能检测矩形空白

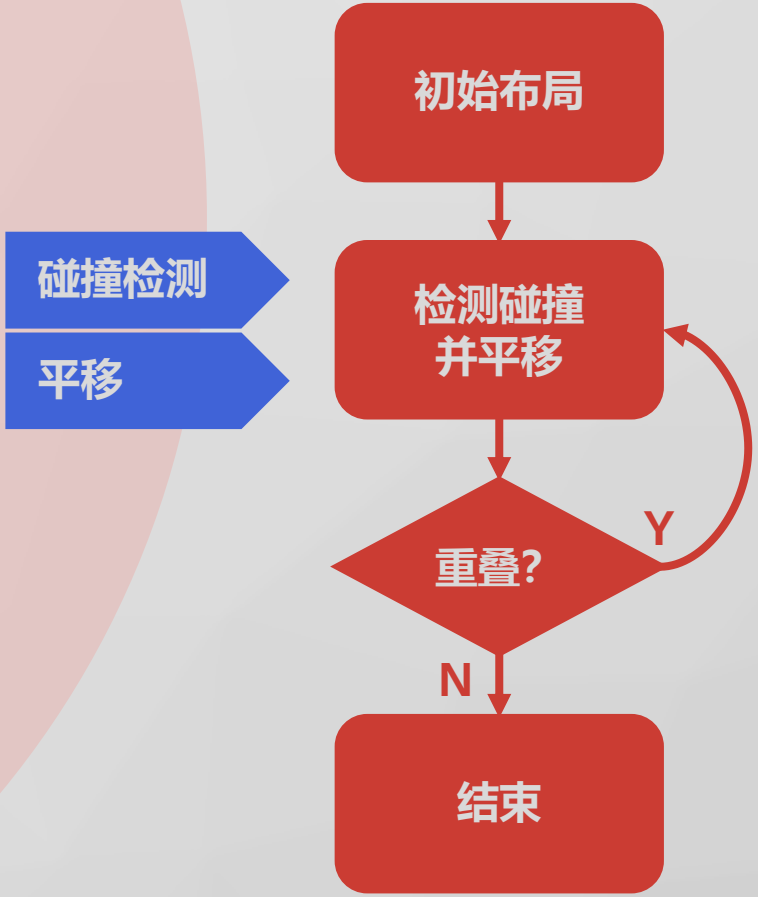
11 - 6 - 9 + 4 = 0

** https://github.com/amueller/word_cloud*

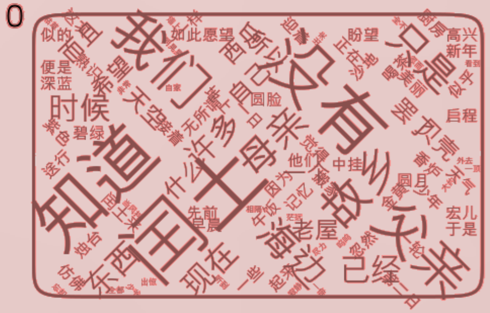
●我们的非贪心算法

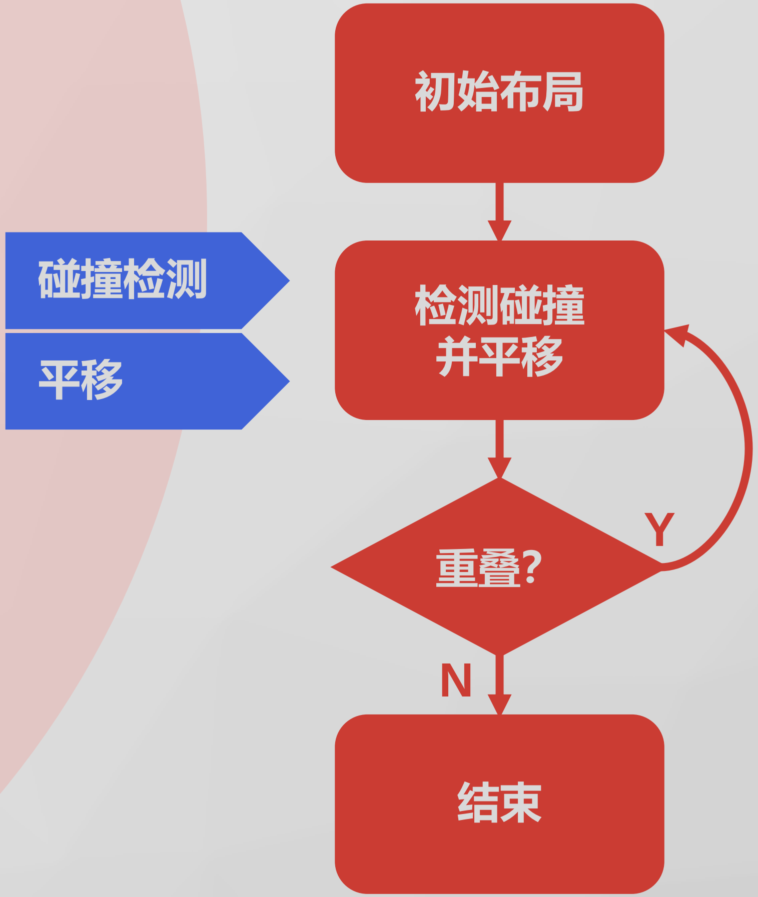
生成词云需要几步?

1. 将所有单词放到初始位置
2. 检测碰撞
3. 挪动（平移）



- 我们的非贪心算法





●我们的非贪心算法 ●几何表示

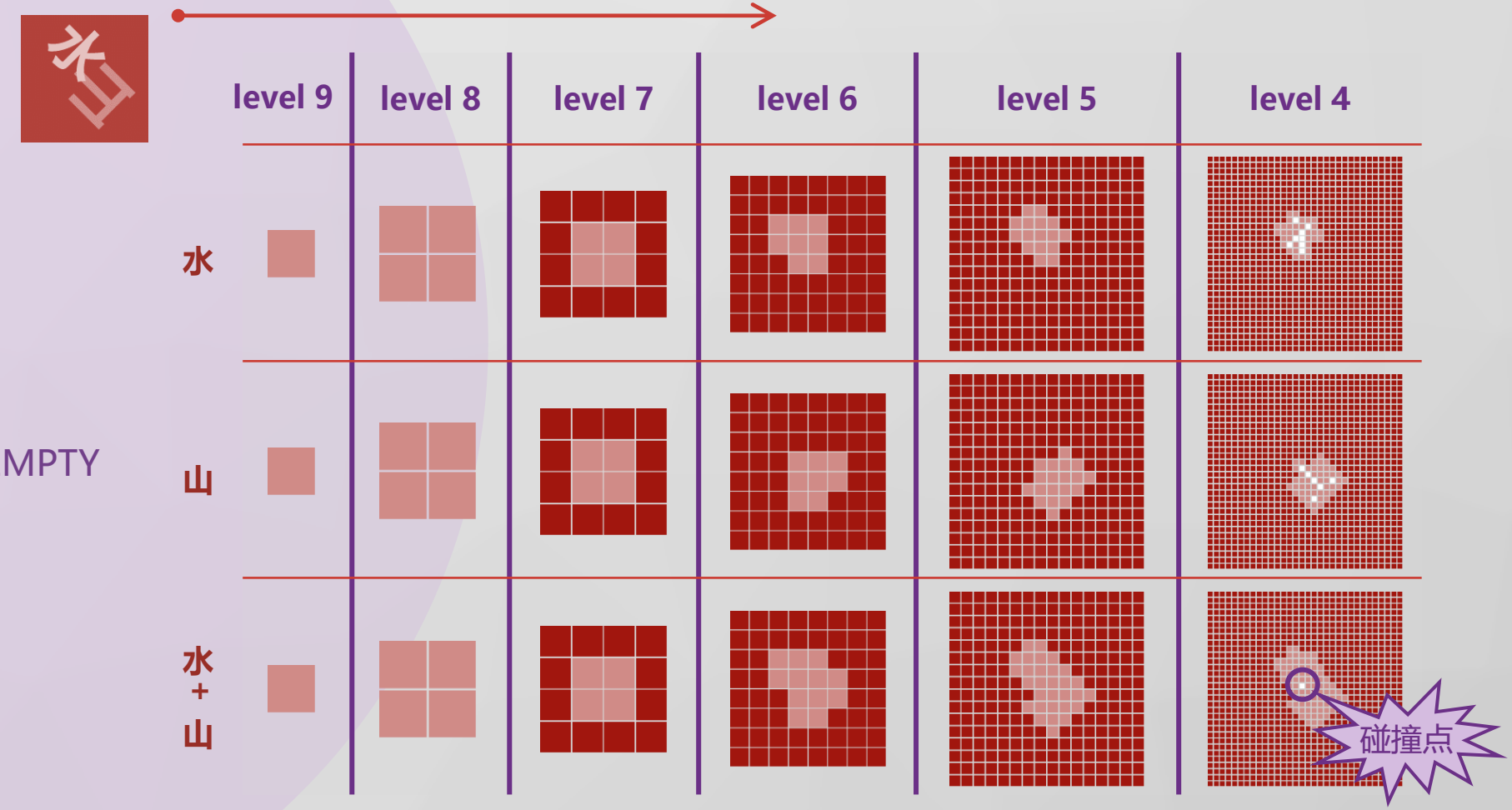
三值栅格金字塔

- 是降采样
- 是层叠四叉树
- 也是对齐的层次包围盒

●我们的非贪心算法 ●碰撞检测

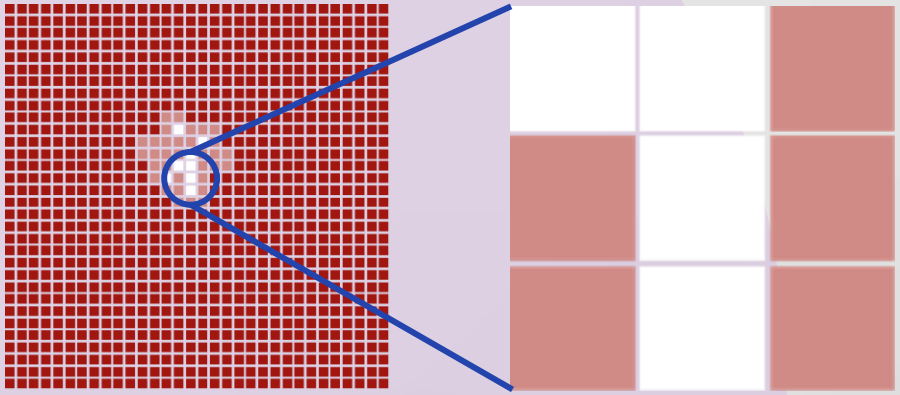
自顶而下搜索

- 在某一点
- 一者为FULL
- 另一者不为EMPTY



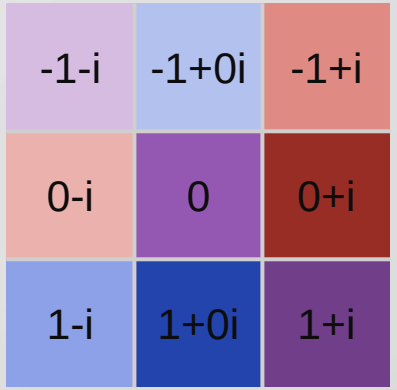
●我们的非贪心算法 ●平移

水

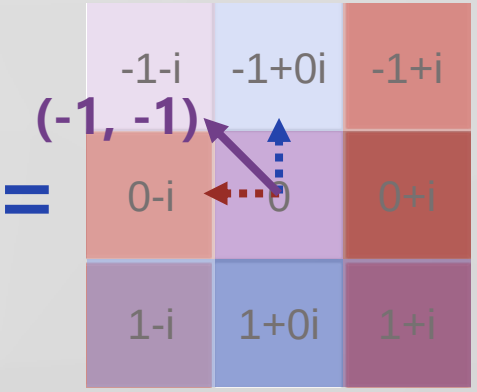


碰撞的层

局部灰度

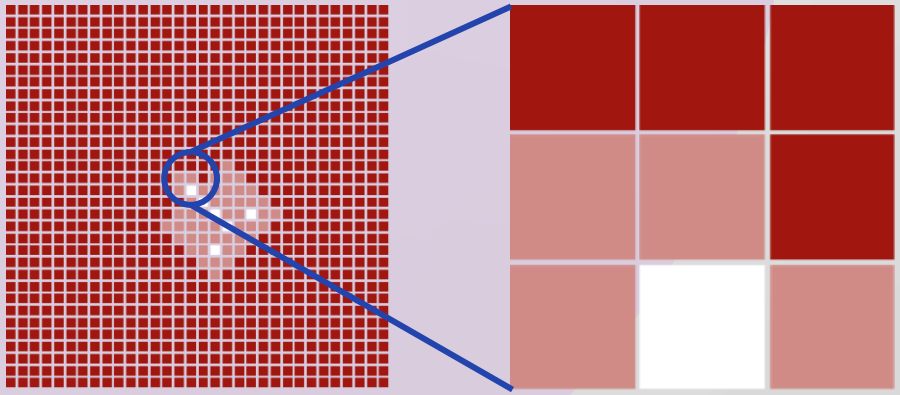


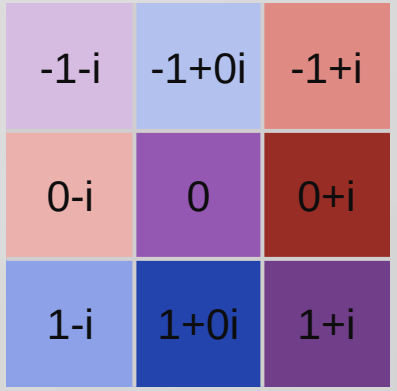
微分算子

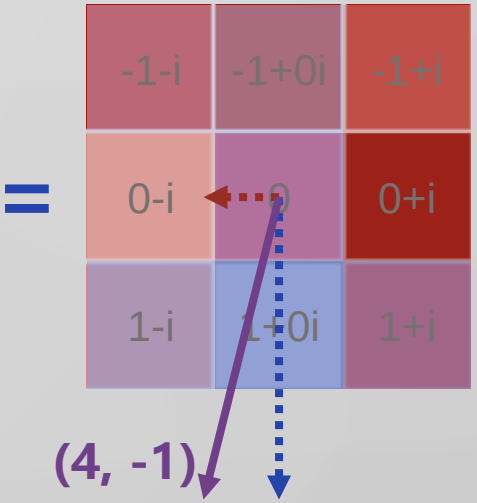


梯度（平移方向）

山





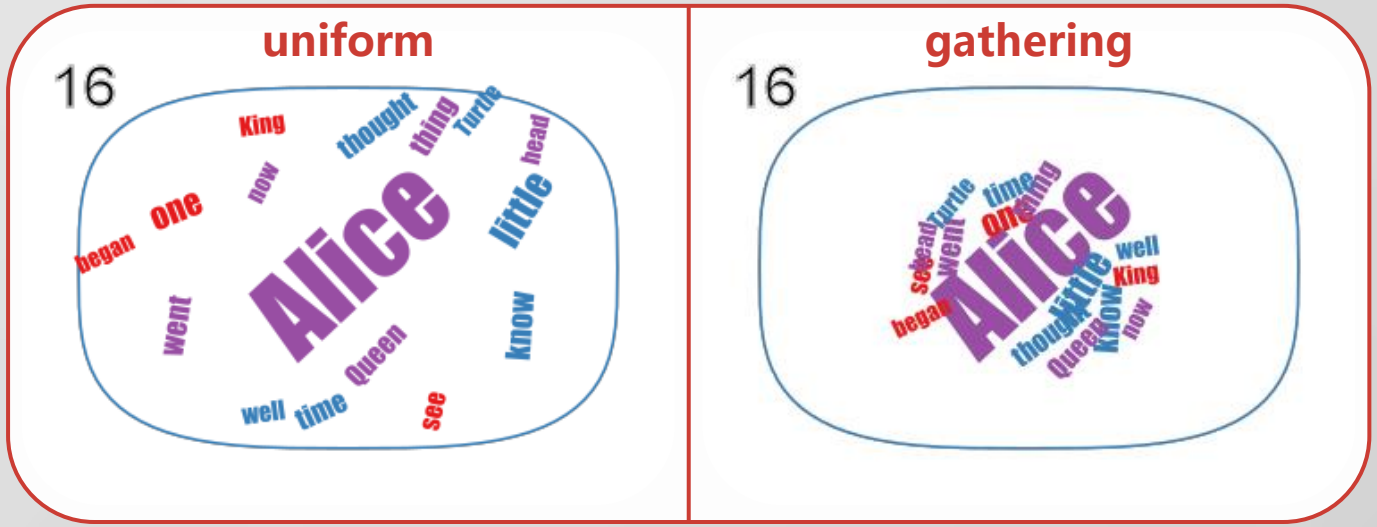


还不够快

- ShiftedQtree——空白处不存储像素，平移时仅修改偏移量
- 层次子区域划分——碰撞不会发生在不同的子区域中
- 近期碰撞记忆——重叠的词平移后很可能依然重叠
- 梯度动量——平滑运动，减少震荡，加速优化
- 并行化碰撞检测——发挥现代计算机的多核优势

初始布局

- 基本方法——贪心法，均匀随机放置在空白的区域
- 风格化——大词靠近中心，或者其它基于语义的位置初始化



怎么跳出局部最优

- 碰撞点随机化——随机广度优先搜索
- 移动顺序随机化——每次梯度检测和平移前洗牌
- 位移随机化——在梯度向量基础上加一个随机向量
- 重新放置——把频繁碰撞的对象重放到空旷地带

解存在吗

- 合适的密度——谨慎的画幅尺寸、字号、密度预估策略
- 最大迭代次数——兜底策略，必有终止
- 提前终止——碰撞数难以下降时终止迭代
- 降低密度——布局失败后统一缩小字号并重试

感谢围观