初探地图类 APP 后端那些事

空间索引算法

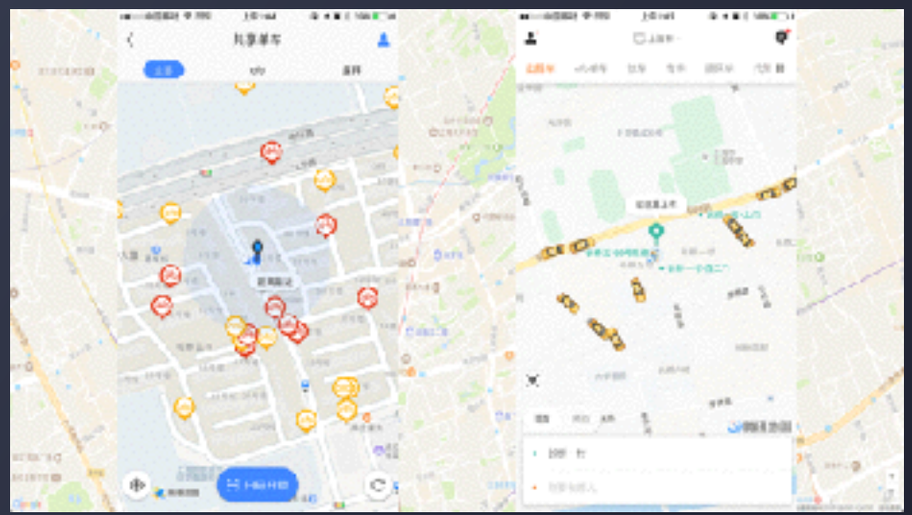
WELCOME

自我介绍

ID：一缕殇流化隐半边冰霜（花名：霜菜）

了解 iOS 开发、前端小白、后端新手
略懂 JavaScript、Go、C、C++、Objective-C、Swift

Github：@halfrost
微博：@halfrost



左边是静止的，右边的车是动态的。还有大众点评周围的商家，饿了么骑手动态化的智能调度。

PRESENTATION AGENDA

01 Geohash

02 Space filling curve

03 Google S2

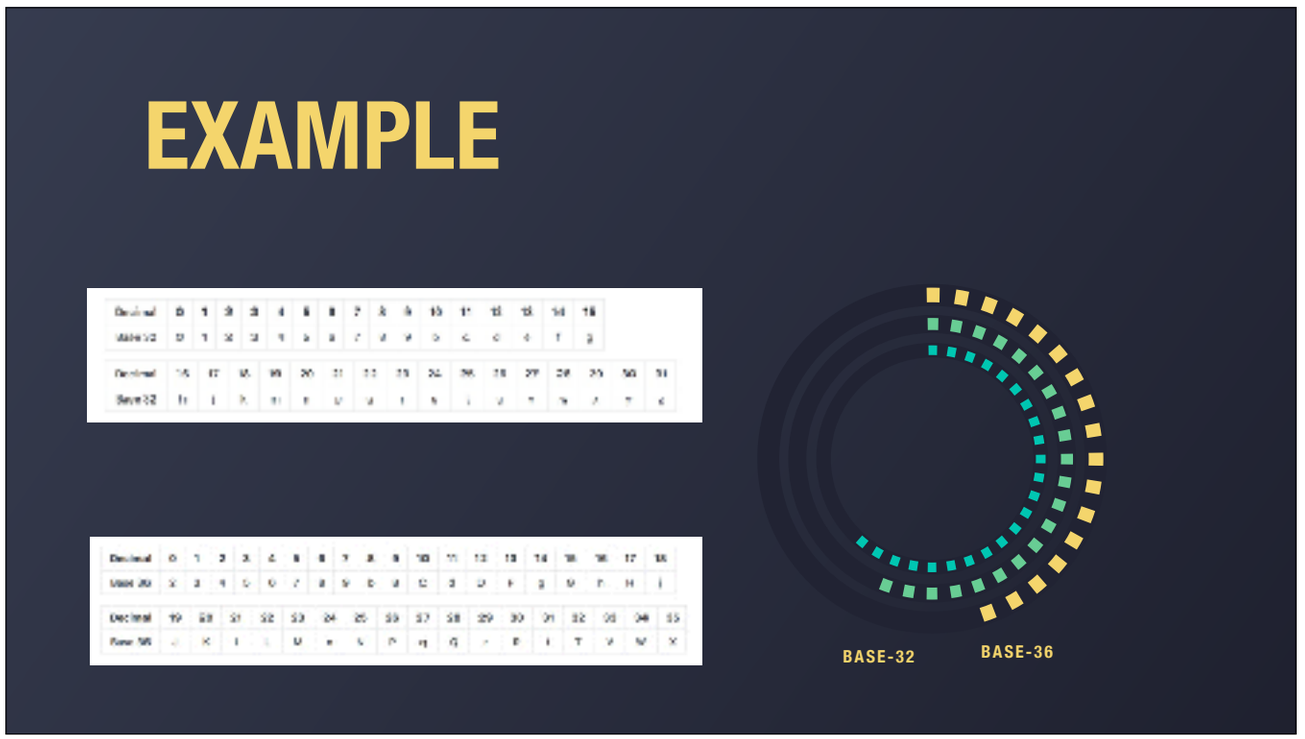
04 Application

SPATIAL INDEX

GEOHASH

NEXT SLIDE

#01



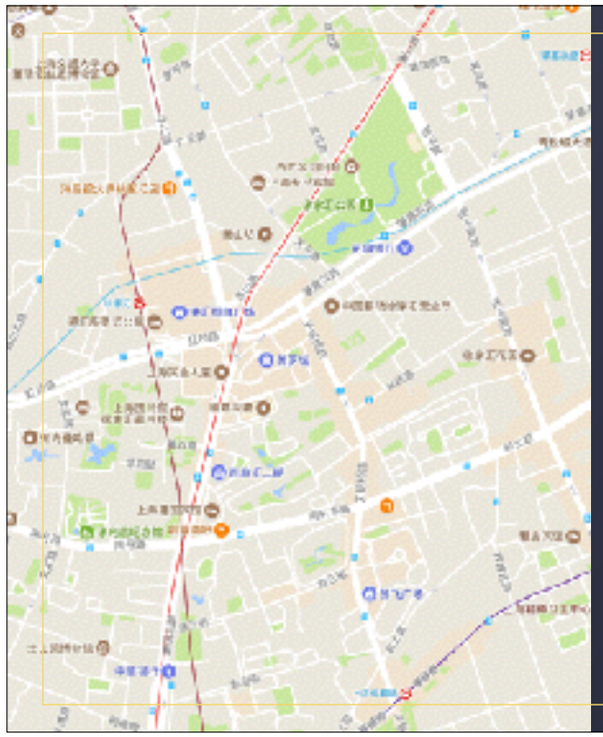
base 36 的版本对大小写敏感，用了36个字符，“23456789bBCdDFgGhHjJKlLMnNPqQrRtTVWX”。

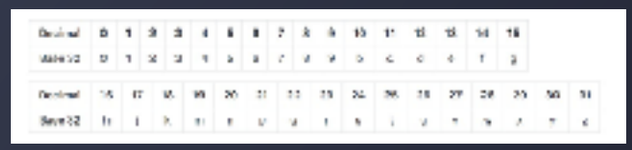
EXAMPLE


wtw37p
wtw37r
wtw37x
wtw37n
wtw37q
wtw37w
wtw37j
wtw37m
wtw37t

level 6 ，意味着字符串长度为6，由于是 base-32，2的5次方，所有每5个二进制位表示一个 base-32，6个 base-32 需要30个二进制位，除以2等于15，所以每边二分都是15次。

GEOHASH (LEVEL-6)





11100 11001 11100 00011 00111 10110

28 25 28 3 7 22

wtw37q

NEXT

从地图上可以看出，这邻近的9个格子，前缀都完全一致。都是wtw37。

当Geohash 增加到7位的时候，网格更小了，美罗城的 Geohash 变成了 wtw37qt。

可以看到中间大格子的 Geohash 的值是 wtw37q，那么它里面的所有小格子前缀都是 wtw37q。可以想象，当 Geohash 字符串长度为5的时候，Geohash 肯定就为 wtw37了。

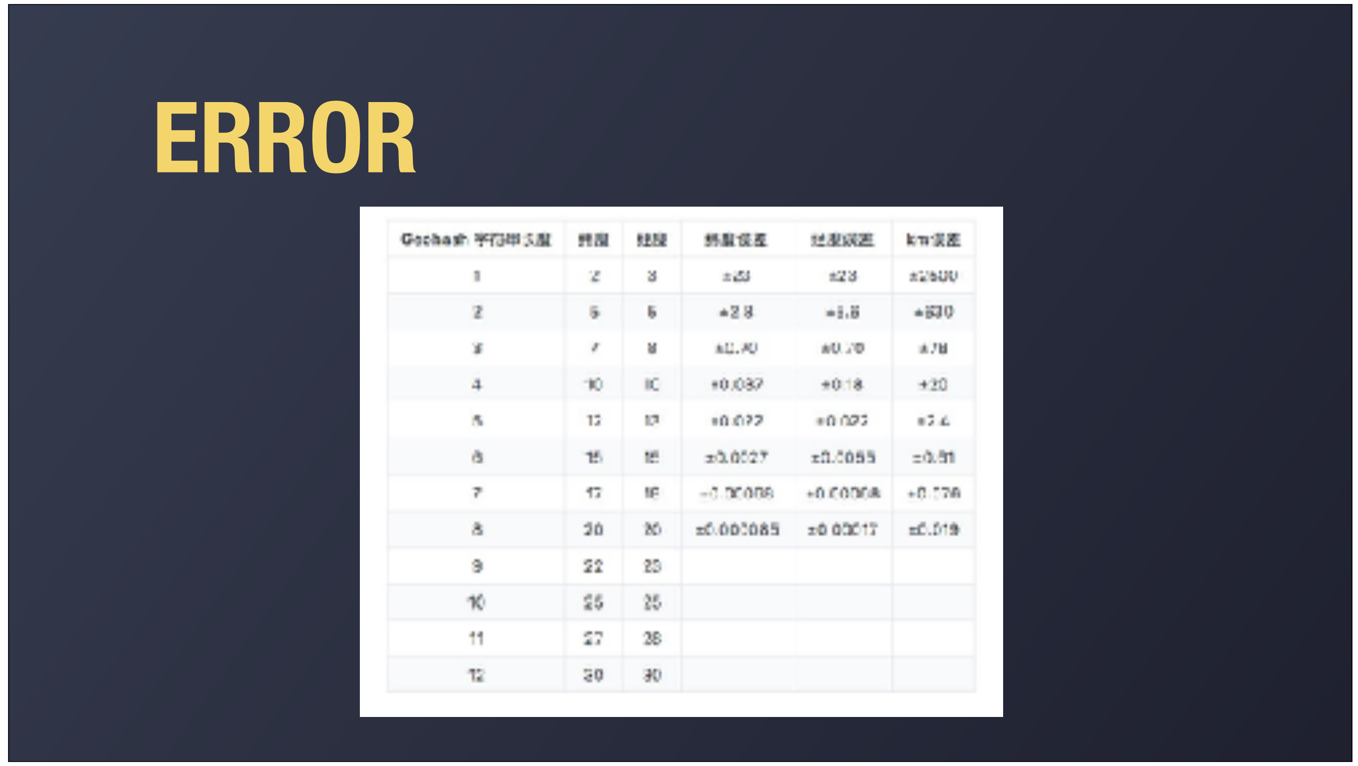
CODE

```go
package geohash

import (
    "bytes"
)

const (
    BASE32               = "0123456789bcdefghjkmnpqrstuvwxyz"
    MAX_LATITUDE  float64 = 90
    MIN_LATITUDE  float64 = -90
    MAX_LONGITUDE float64 = 180
    MIN_LONGITUDE float64 = -180
)

var (
    bits   = []int{16, 8, 4, 2, 1}
    base32 = []byte(BASE32)
)

type Box struct {
    MinLat, MaxLat float64 // 纬度
    MinLng, MaxLng float64 // 经度
}

func (this *Box) Width() float64 {
    return this.MaxLng - this.MinLng
}

func (this *Box) Height() float64 {
    return this.MaxLat - this.MinLat
}
```

```go
// 输入值：纬度，经度，精度(geohash的长度)
// 返回geohash，以及该点所在的区域
func Encode(latitude, longitude float64, precision int) (string, *Box) {
    var geohash bytes.Buffer
    var minLat, maxLat float64 = MIN_LATITUDE, MAX_LATITUDE
    var minLng, maxLng float64 = MIN_LONGITUDE, MAX_LONGITUDE
    var mid float64 = 0

    bit, ch, length, isEven := 0, 0, 0, true
    for length < precision {
        if isEven {
            if mid = (minLng + maxLng) / 2; mid < longitude {
                ch |= bits[bit]
                minLng = mid
            } else {
                maxLng = mid
            }
        } else {
            if mid = (minLat + maxLat) / 2; mid < latitude {
                ch |= bits[bit]
                minLat = mid
            } else {
                maxLat = mid
            }
        }
        isEven = !isEven
        if bit < 4 {
            bit++
        } else {
            geohash.WriteByte(base32[ch])
            length, bit, ch = length+1, 0, 0
        }
    }
    b := &Box{
        MinLat: minLat,
        MaxLat: maxLat,
        MinLng: minLng,
        MaxLng: maxLng,
    }
    return geohash.String(), b
}
```

SCALE

字符串长度		cell 宽度		cell 高度
1	≤	5,000km	×	5,000km
2	≤	1,250km	×	625km
3	≤	156km	×	156km
4	≤	39.1km	×	19.5km
5	≤	4.89km	×	4.89km
6	≤	1.22km	×	0.61km
7	≤	153m	×	153m
8	≤	38.2m	×	19.1m
9	≤	4.77m	×	4.77m
10	≤	1.19m	×	0.596m
11	≤	149mm	×	149mm
12	≤	37.2mm	×	18.6mm



Geohash 字符串长度	纬度	经度	纬度误差	经度误差	km误差
1	2	3	±23	±23	±2500
2	5	5	±2.8	±5.6	±630
3	7	8	±0.70	±0.70	±78
4	10	10	±0.087	±0.18	±20
5	12	13	±0.022	±0.022	±2.4
6	15	15	±0.0027	±0.0055	±0.61
7	17	18	±0.00068	±0.00068	±0.076
8	20	20	±0.000085	±0.00017	±0.019
9	22	23			
10	25	25			
11	27	28			
12	30	30			

由于有奇偶数分配不均匀的关系，所以存在误差

1.Geohash 是一种地理编码，由 Gustavo Niemeyer 发明的。它是一种分级的数据结构，把空间划分为网格。Geohash 属于空间填充曲线中的 Z 阶曲线（Z-order curve）的实际应用。上图就是 Z 阶曲线。这个曲线比较简单，生成它也比较容易，只需要把每个 Z 首尾相连即可。

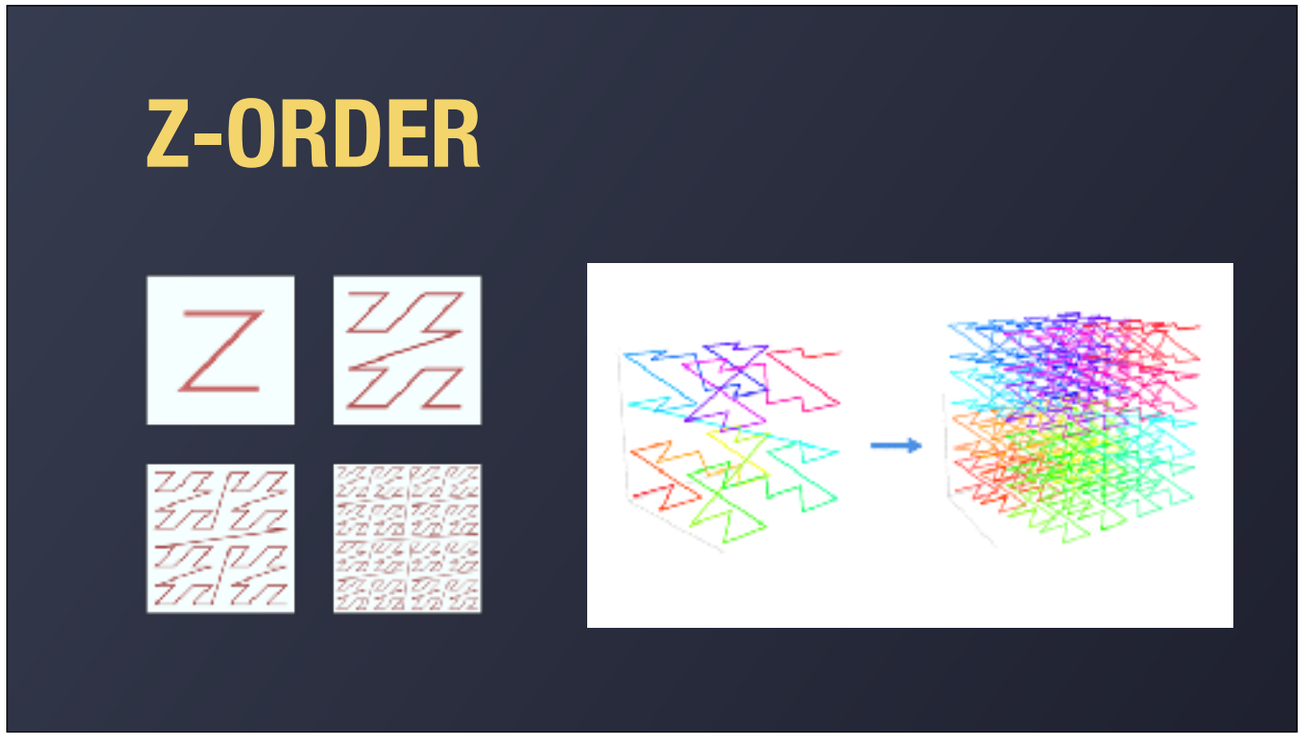
SPATIAL FILLING CURVE

SPACE FILLING CURVE

NEXT SLIDE

#02

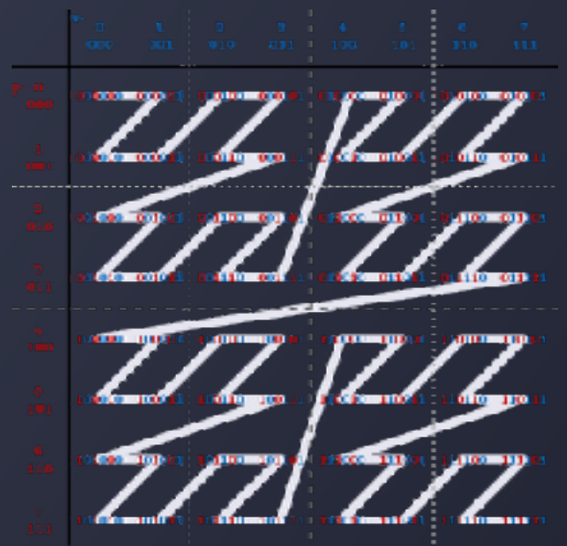
欧几里得几何有时就指二维平面上的几何，即平面几何。三维空间的欧几里得几何通常叫做立体几何。高维的情形叫欧几里得空间。黎曼几何大部分都是广义相对论的四维研究对象。所以 n 维空间应该是欧几里得空间，这些数学空间也被叫做n维欧几里得空间（甚至简称 n维空间）或有限维实内积空间。n 维时空对应的是黎曼几何。



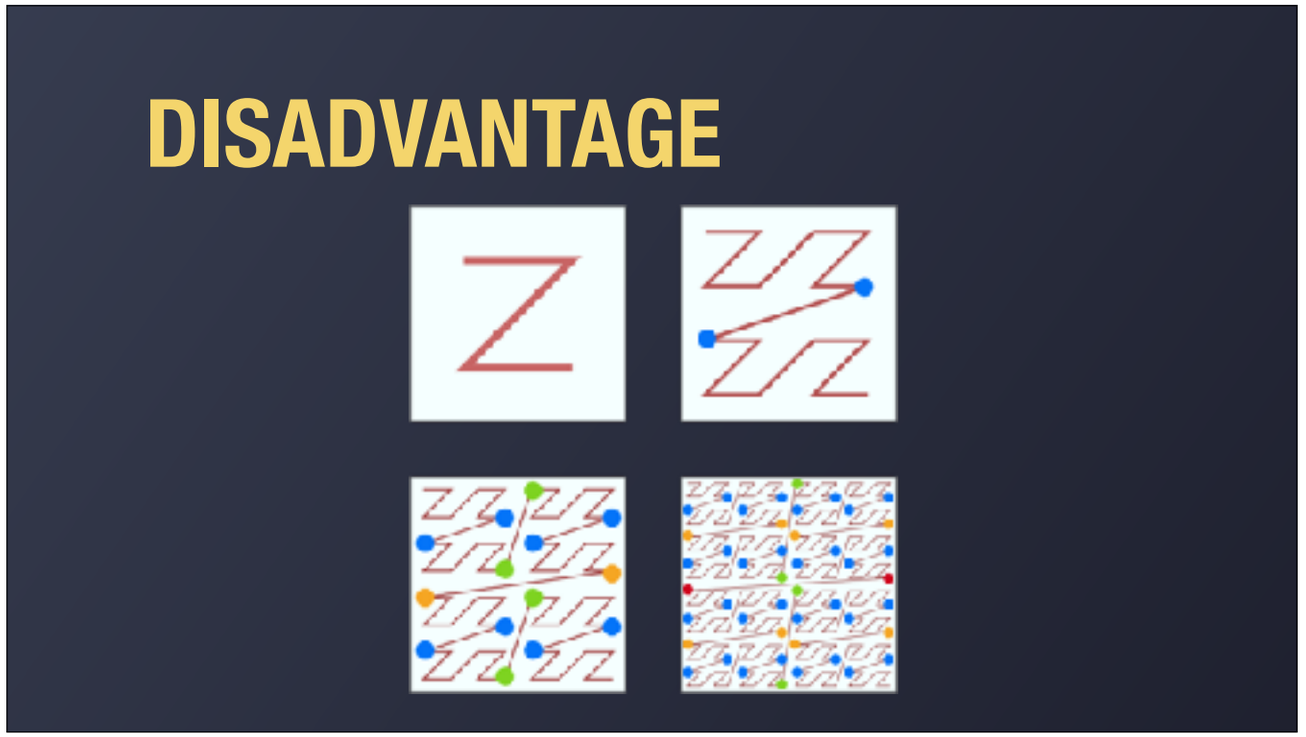
上图就是 Z 阶曲线。这个曲线比较简单，生成它也比较容易，只需要把每个 Z 首尾相连即可。Z 阶曲线同样可以扩展到三维空间。只要 Z 形状足够小并且足够密，也能填满整个三维空间。

Geohash 有一个和 Z 阶曲线相关的性质，那就是一个点附近的地方(但不绝对) hash 字符串总是有公共前缀，并且公共前缀的长度越长，这两个点距离越近。越接近的点通常和目标点的 Geohash 字符串公共前缀越长（但是这不一定，也有特殊情况，下面举例会说明）

Z-ORDER



Z 阶曲线的理论来源



它利用 Z 阶曲线进行编码。而 Z 阶曲线可以将二维或者多维空间里的所有点都转换成一维曲线。在数学上成为分形维。并且 Z 阶曲线还具有局部保序性。 Geohash 的另外一个优点，搜索查找邻近点比较快。Z 阶曲线有一个比较严重的问题，虽然有局部保序性，但是它也有突变性。在每个 Z 字母的拐角，都有可能出现顺序的突变。

DISADVANTAGE

EXAMPLE

如果选择 Geohash 字符串为6的话，就是蓝色的大格子。红星是美罗城，紫色的圆点是搜索出来的目标点。如果用 Geohash 算法查询的话，距离比较近的可能是 wtw37p，wtw37r，wtw37w，wtw37m。但是其实距离最近的点就在 wtw37q。如果选择这么大的网格，就需要再查找周围的8个格子。

如果选择 Geohash 字符串为7的话，那变成黄色的小格子。这样距离红星星最近的点就只有一个了。就是 wtw37qw。

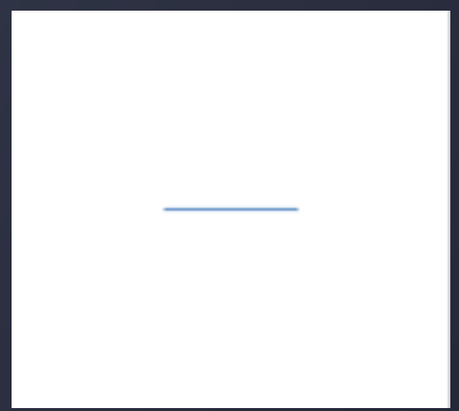
如果网格大小，精度选择的不好，那么查询最近点还需要再次查询周围8个点。

PEANO CURVE

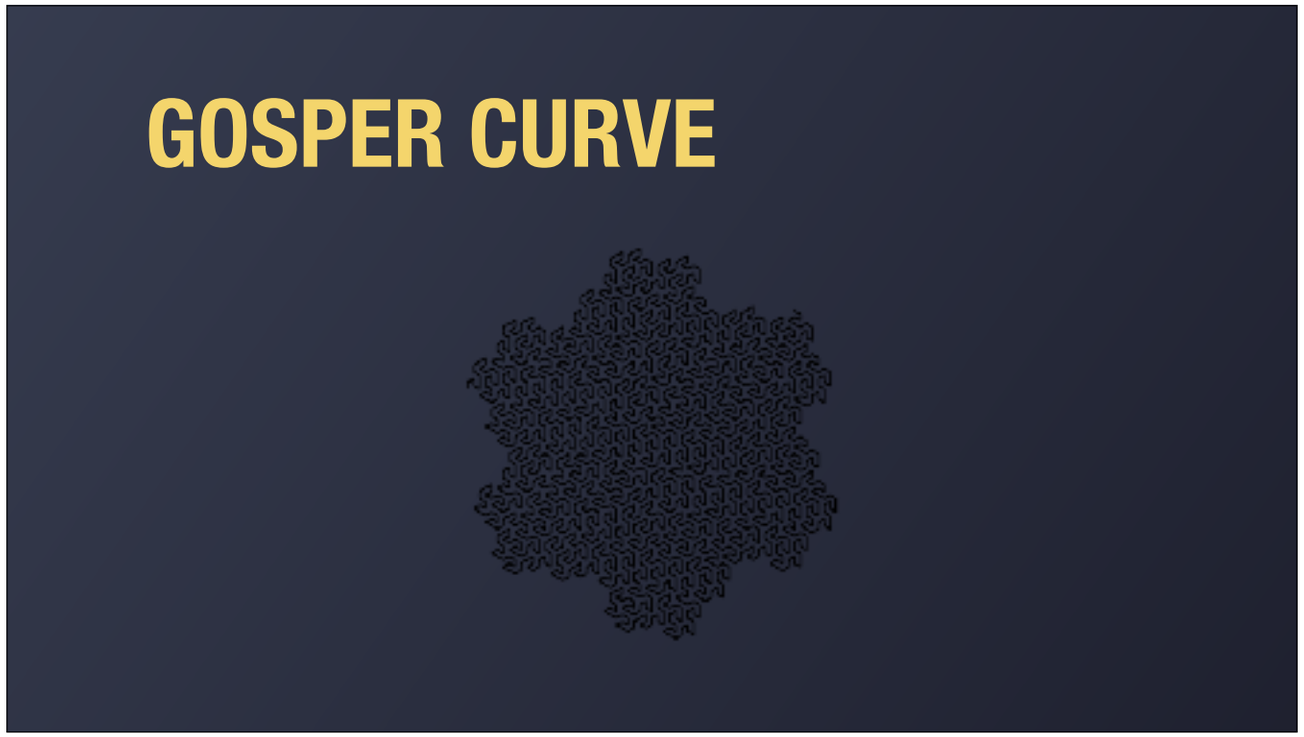
皮亚诺曲线是一条连续的但处处不可导的曲线。

在1890年，Giuseppe Peano 发现了一条连续曲线，现在称为 Peano 曲线，它可以穿过单位正方形上的每个点。他的目的是构建一个可以从单位区间到单位正方形的连续映射。 Peano 受到 Georg Cantor 早期违反直觉的研究结果的启发，即单位区间中无限数量的点与任何有限维度流型（manifold）中无限数量的点，基数相同。 Peano 解决的问题实质就是，是否存在这样一个连续的映射，一条能填充满平面的曲线。上图就是他找到的一条曲线。

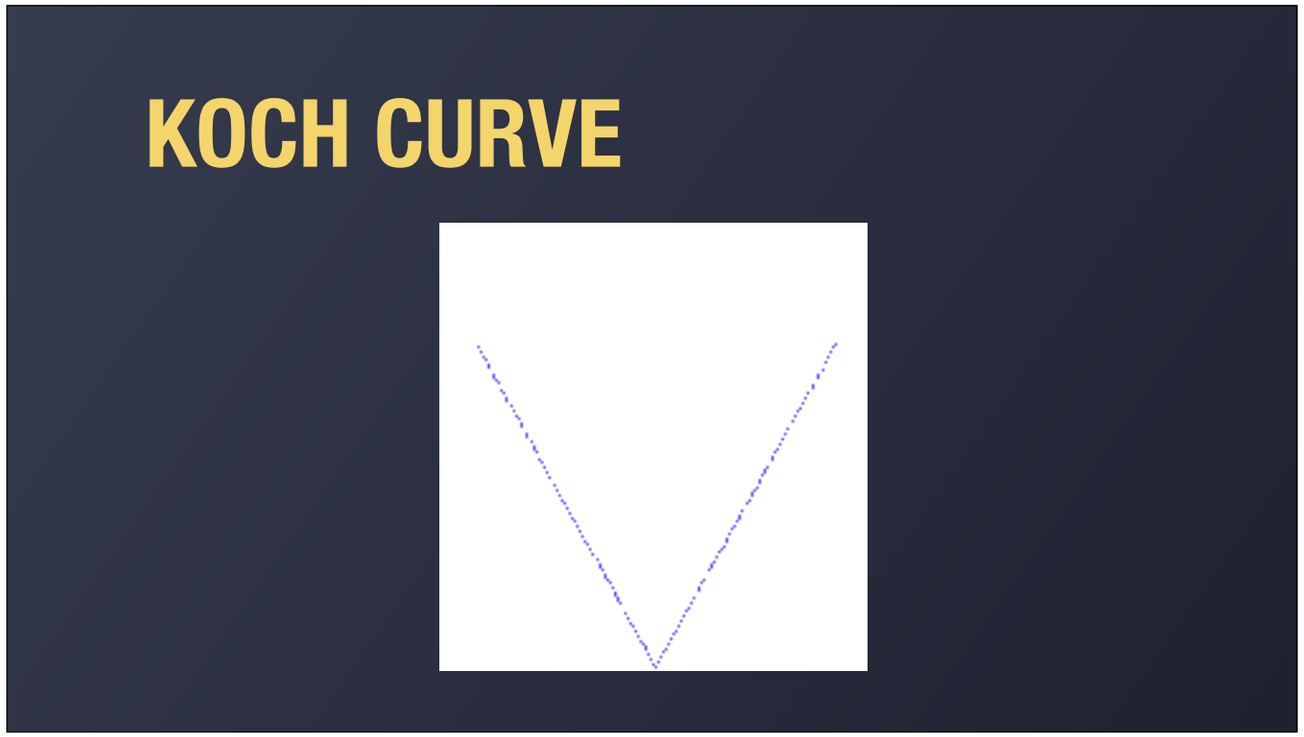
DRAGON CURVE



龙曲线(Dragon curve)

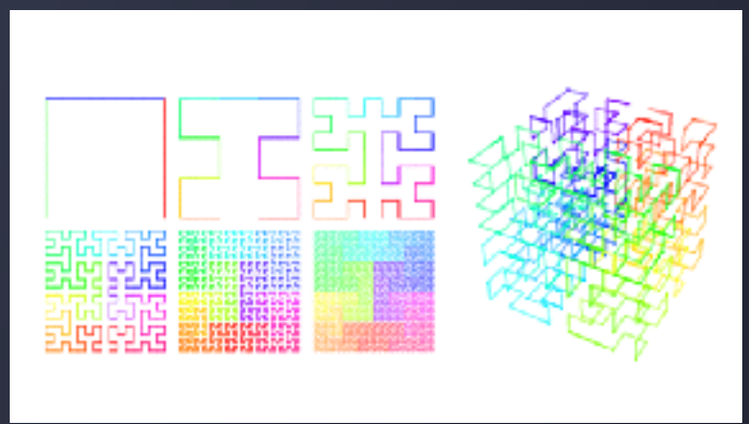


高斯帕曲线(Gosper curve)



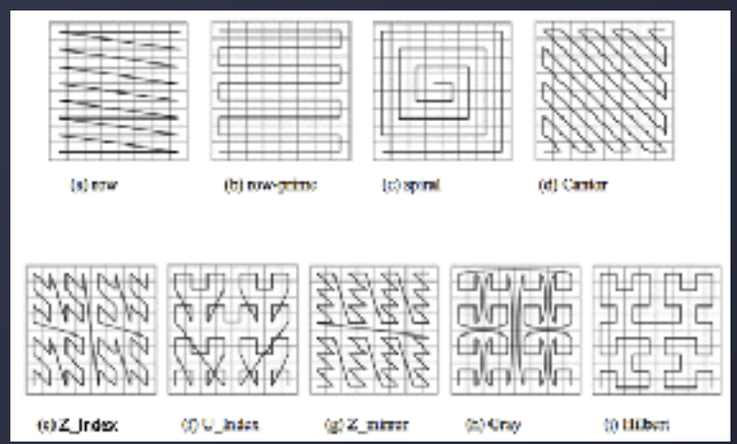
Koch曲线(Koch curve)

MOORE CURVE

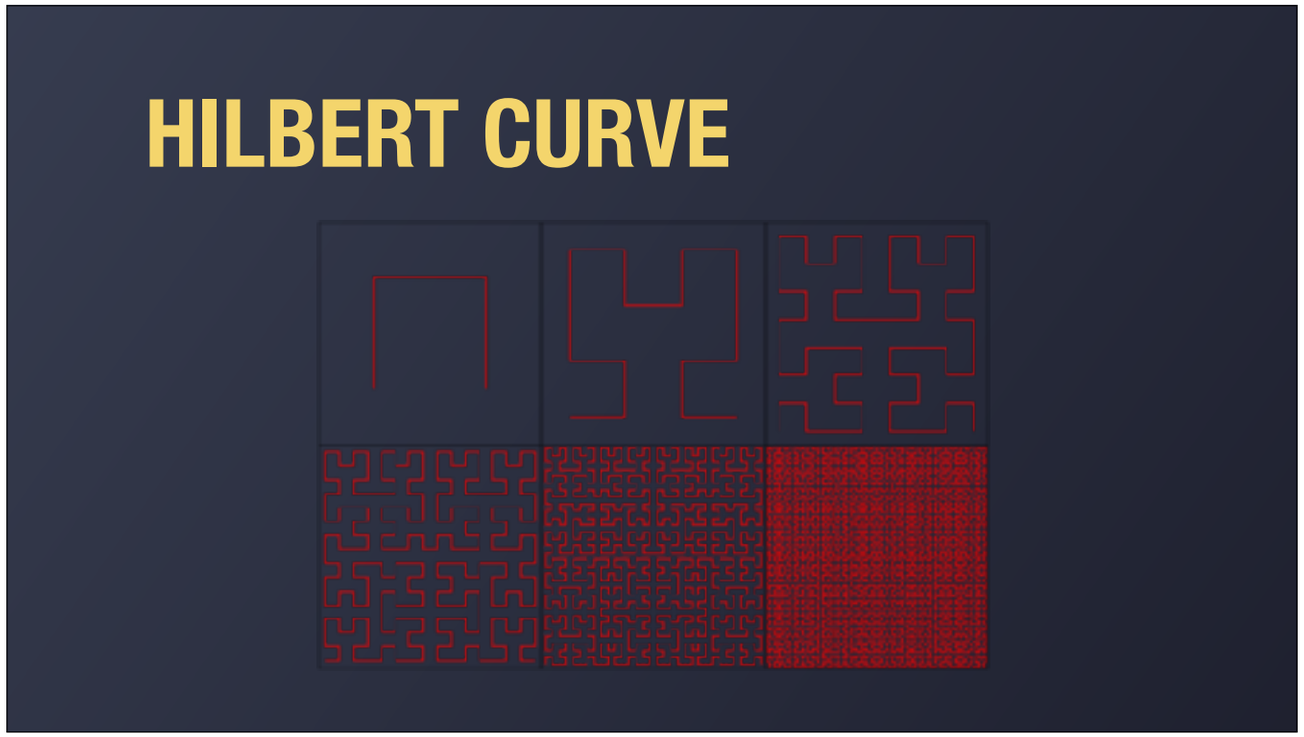


摩尔定律曲线(Moore curve)

SIERPIŃSKI CURVE

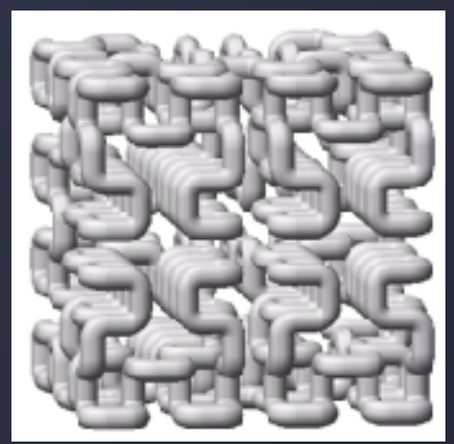


谢尔宾斯基曲线(Sierpiński curve)、奥斯古德曲线(Osgood curve)



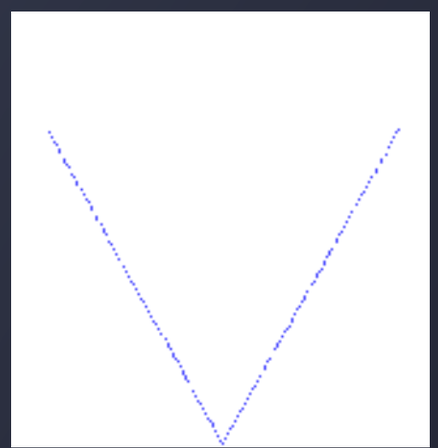
一年后，即1891年，希尔伯特就作出了这条曲线，叫希尔伯特曲线（Hilbert curve）。

HILBERT CURVE



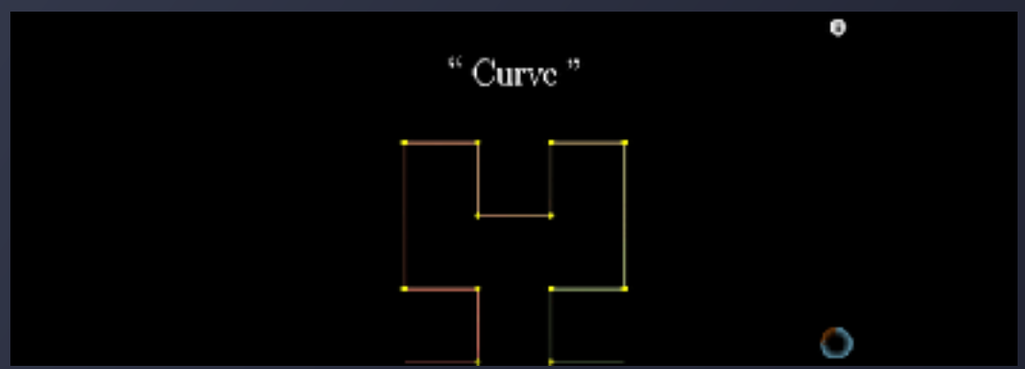
一年后，即1891年，希尔伯特就作出了这条曲线，叫希尔伯特曲线（Hilbert curve）。

HAUSDORFF FRACTALS DIMENSION

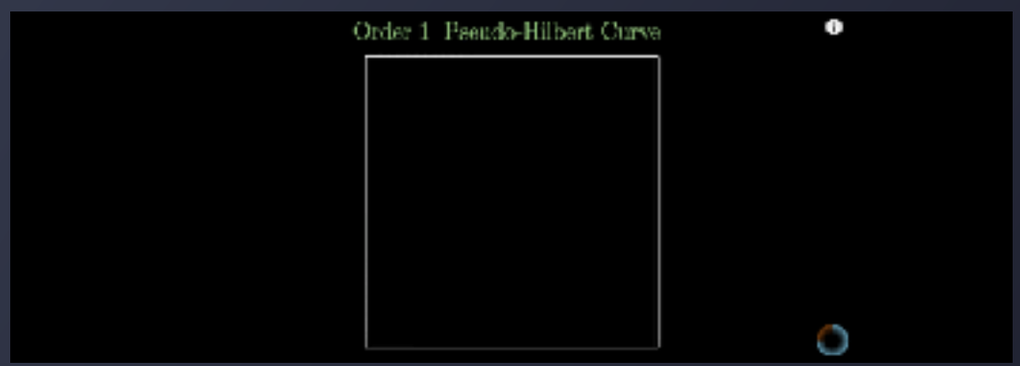


豪斯多夫分形维(Hausdorff fractals dimension)和拓扑维数

HILBERT CURVE

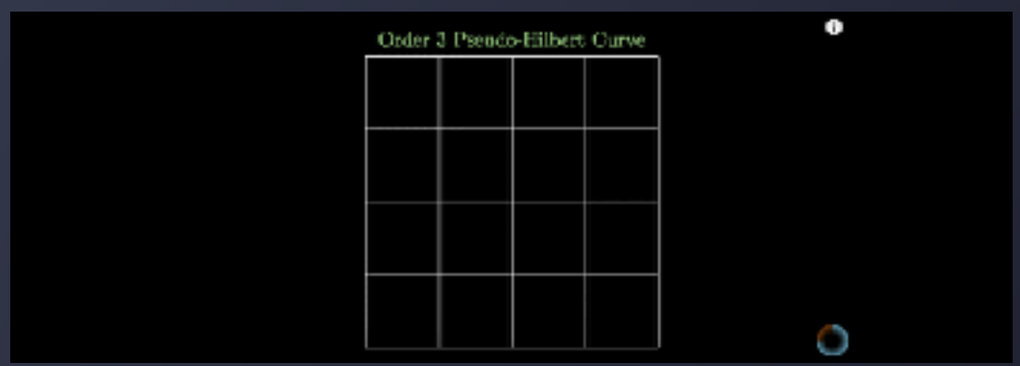


HILBERT CURVE



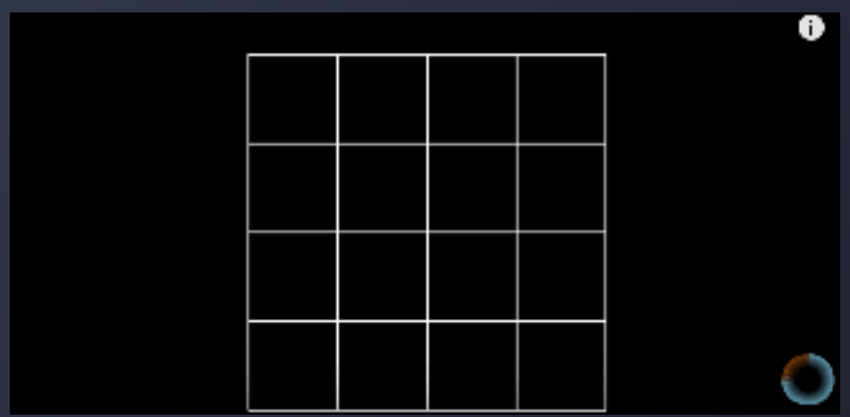
HILBERT CURVE

HILBERT CURVE



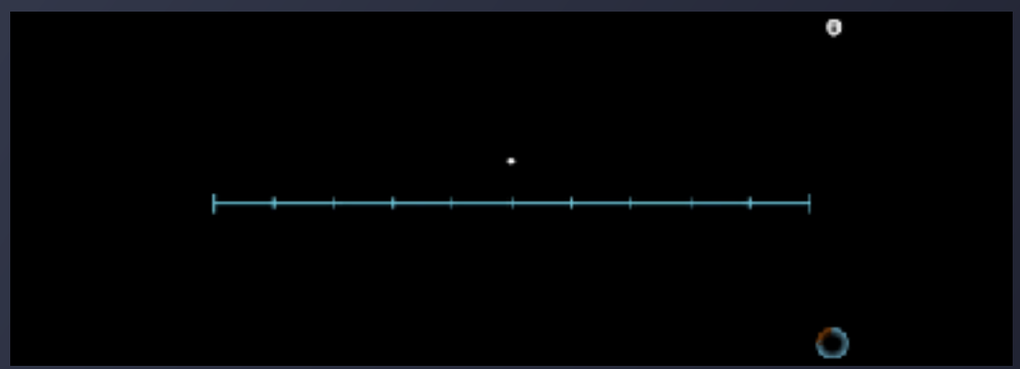
HILBERT CURVE

HILBERT CURVE

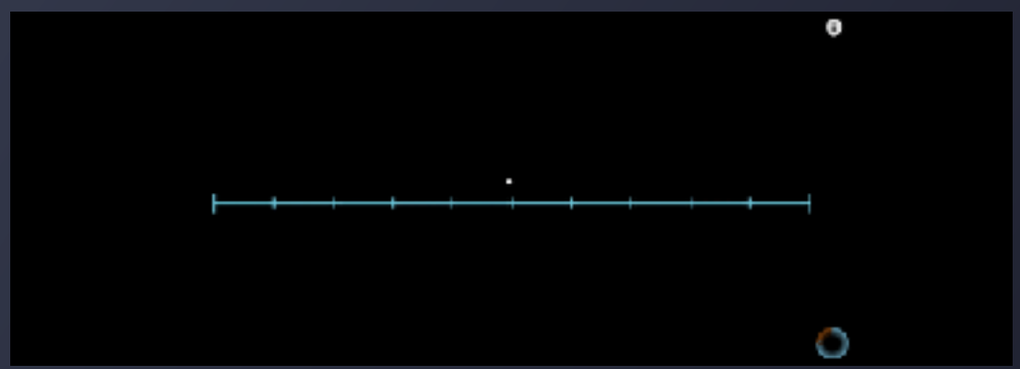


HILBERT CURVE

HILBERT CURVE



HILBERT CURVE



CHARACTERISTIC

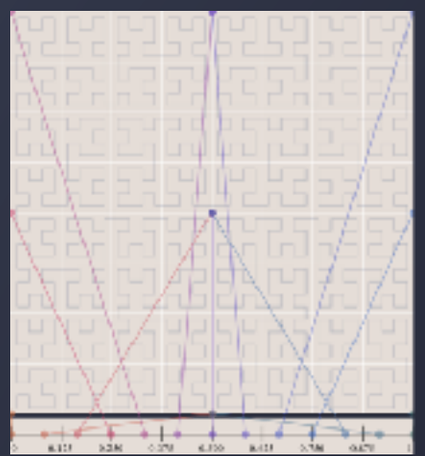
降维

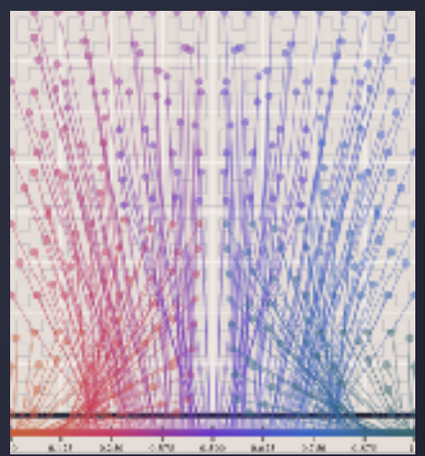
稳定

连续

希尔伯特曲线是连续的，所以能保证一定可以填满空间。连续性是需要数学证明的。具体证明方法这里就不细说了，感兴趣的可以点文章末尾一篇关于希尔伯特曲线的论文，那里有连续性的证明。

CONTINUITY





THANKS FOR YOUR ATTENTION!

ANY QUESTIONS?

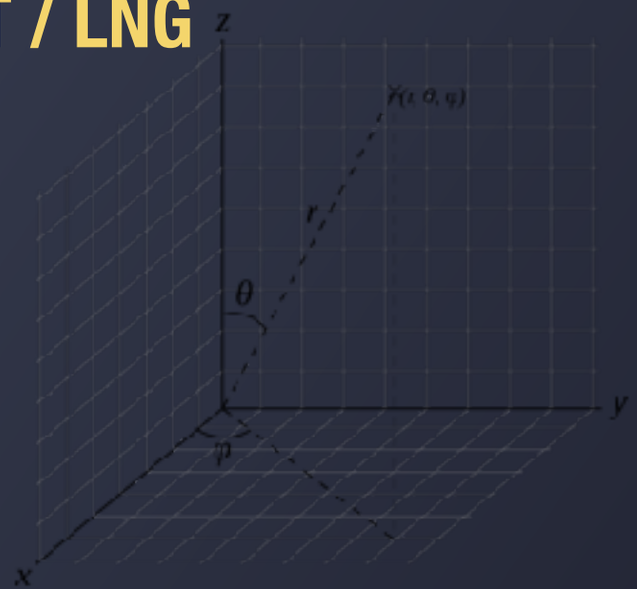
NEXT SLIDE

处理多维空间的思路，先考虑如何降维，再考虑如何分形。

众所周知，地球是近似一个球体。球体是一个三维的，如何把三维降成一维呢?

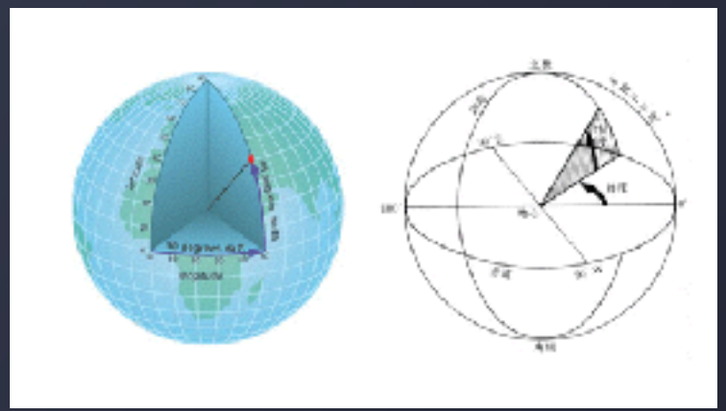
球面上的一个点，在直角坐标系中，可以这样表示

LAT / LNG



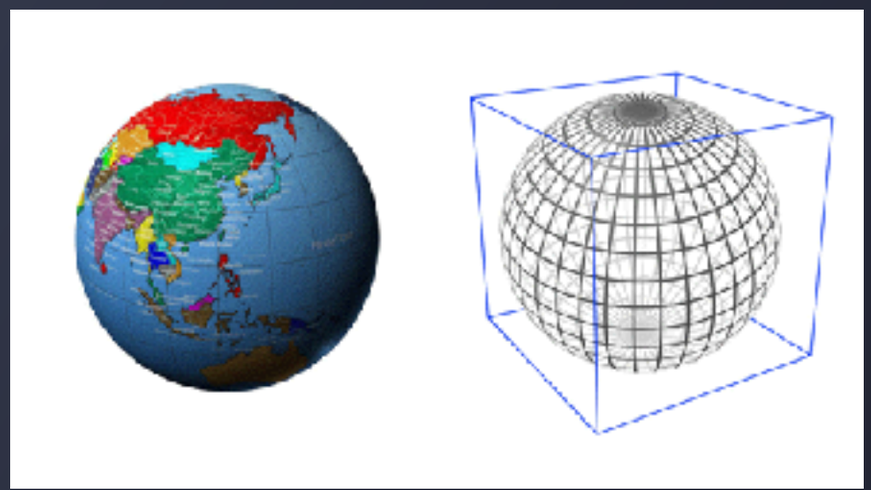
$$x = r * \sin\theta * \cos\phi$$
$$y = r * \sin\theta * \sin\phi$$
$$z = r * \cos\theta$$

LAT / LNG

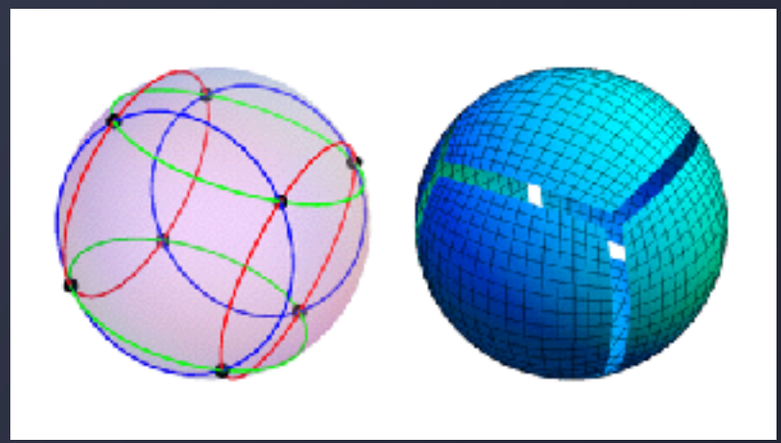


s(lat,lng) -> f(x,y,z)

PROJECTION

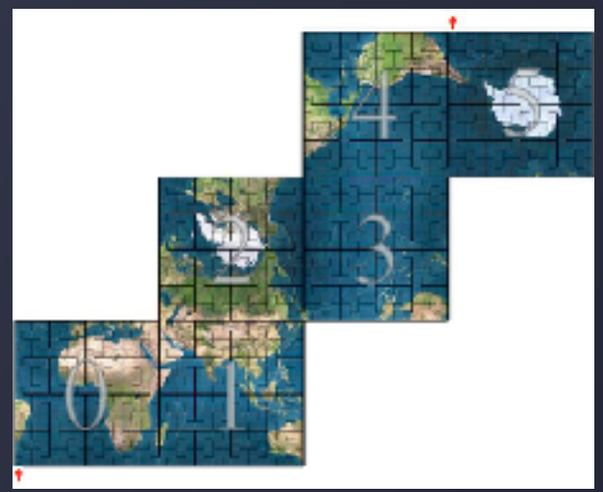


PROJECTION

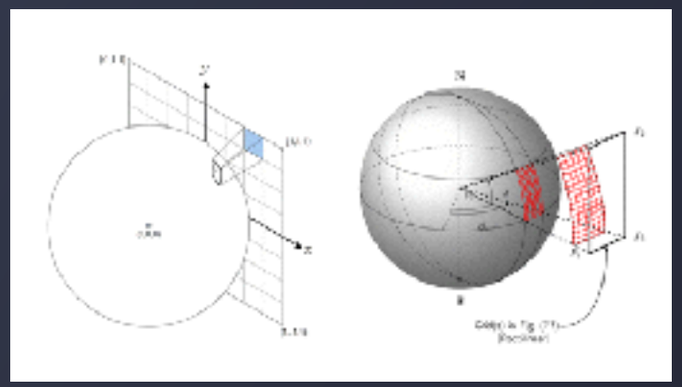


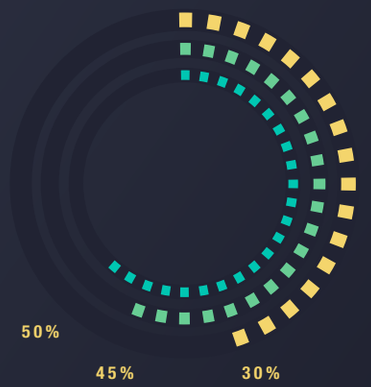
FRACTAL

FRACTAL



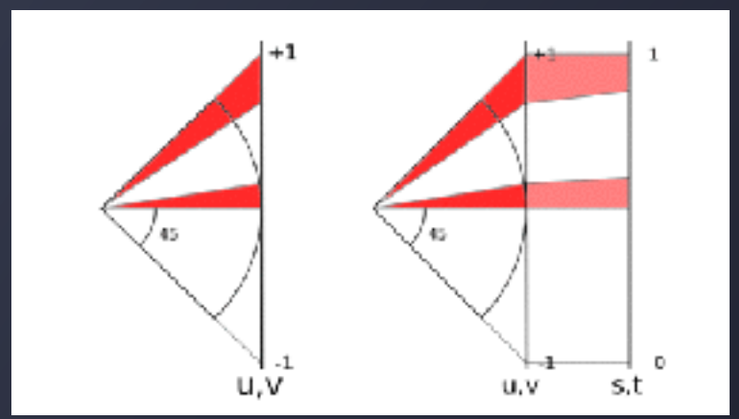
FRACTAL



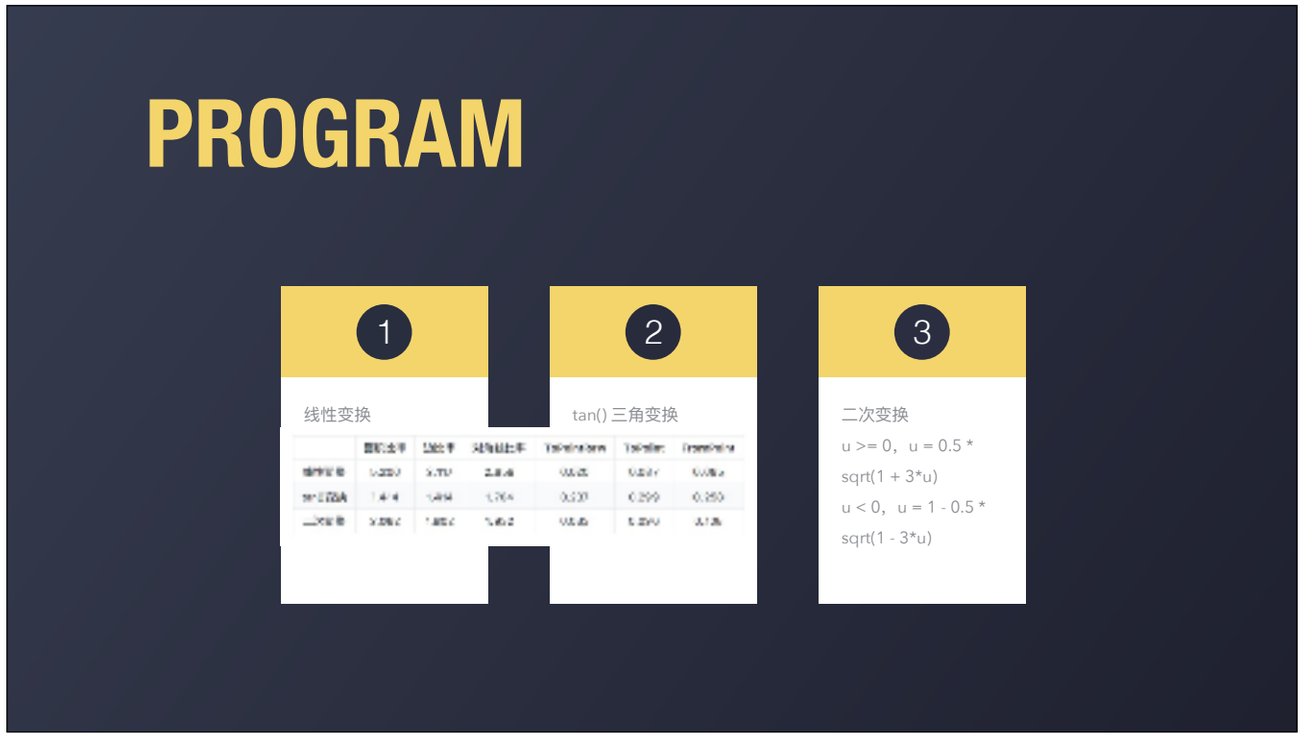


f(x,y,z) -> g(face,u,v)

FIXED

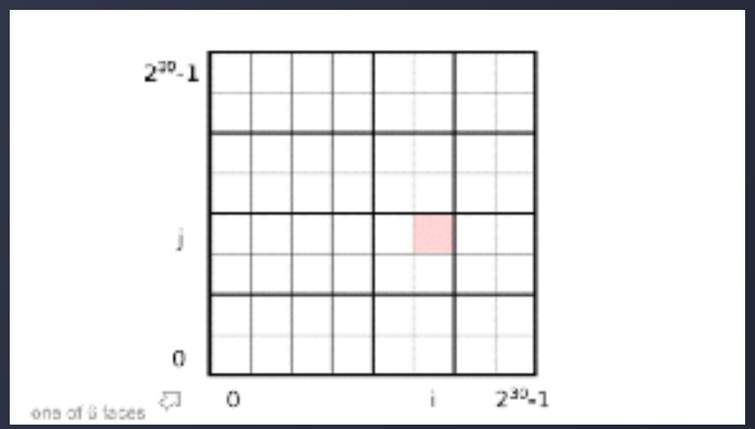


g(face,u,v) -> h(face,s,t)



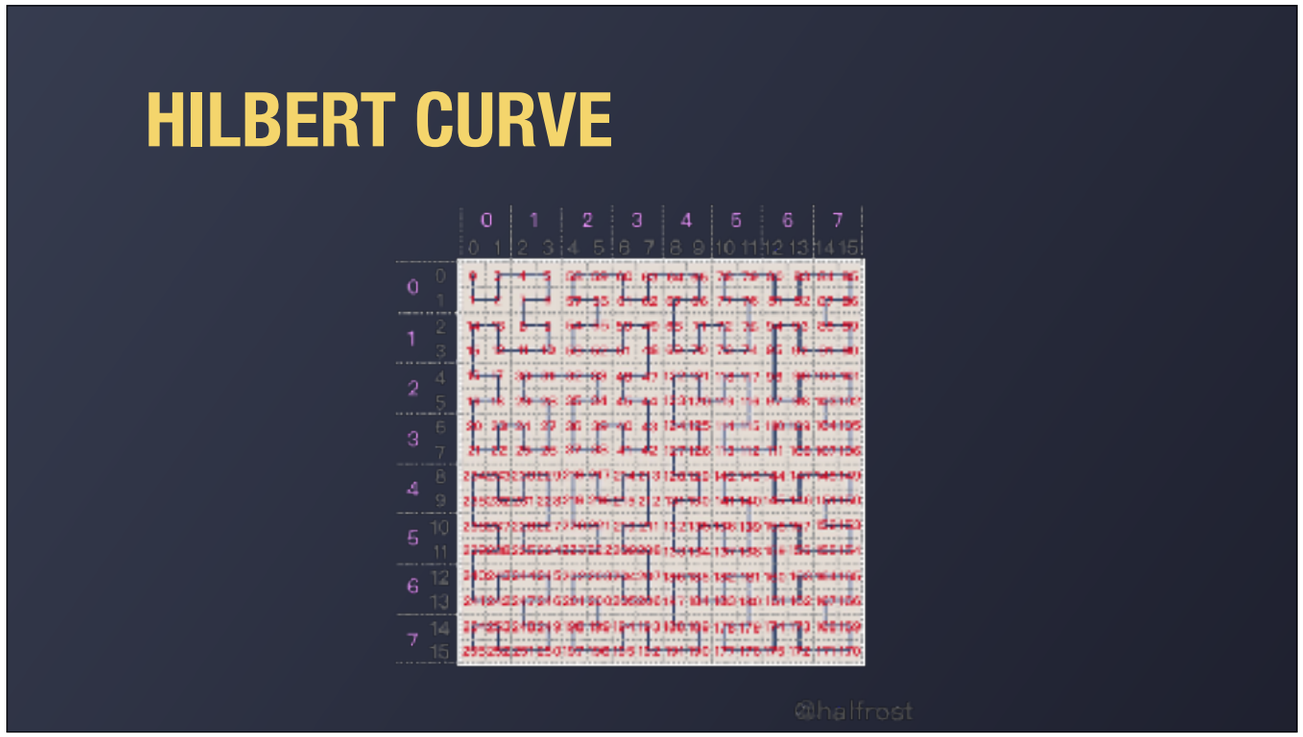
最后谷歌选择的是二次变换，这是一个近似切线的投影曲线。它的计算速度远远快于 tan() ，大概是 tan() 计算的3倍速度。生成的投影以后的矩形大小也类似。不过最大的矩形和最小的矩形相比依旧有2.082的比率。

TRANSFORM



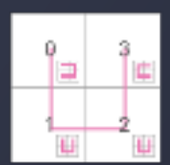
h(face,s,t) -> H(face,i,j)

s，t的值域是[0,1]，现在值域要扩大到[0,2^30^-1]



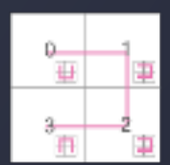
注意：由于 CellID 是64位的，头三位是 face ，末尾一位是标志位，所以中间有 60 位。i，j 转换成二进制是30位的。7个4位二进制位和1个2位二进制位。4*7 + 2 = 30 。iijjoo ，即 i 的头2个二进制位和 j 的头2个二进制位加上 origOrientation，这样组成的是6位二进制位，最多能表示 2^6^ = 32，转换出来的 pos + orientation 最多也是32位的。即转换出来最多也是6位的二进制位，除去末尾2位 orientation ，所以 pos 在这种情况下最多是 4位。iiiijjjjpppp，即 i 的4个二进制位和 j 的4个二进制位加上 origOrientation，这样组成的是10位二进制位，最多能表示 2^10^ = 1024，转换出来的 pos + orientation 最多也是10位的。即转换出来最多也是10位的二进制位，除去末尾2位 orientation ，所以 pos 在这种情况下最多是 8位。

HILBERT CURVE



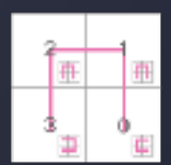
标准顺序

图0



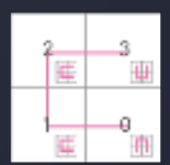
轴旋转

图1



上下倒置

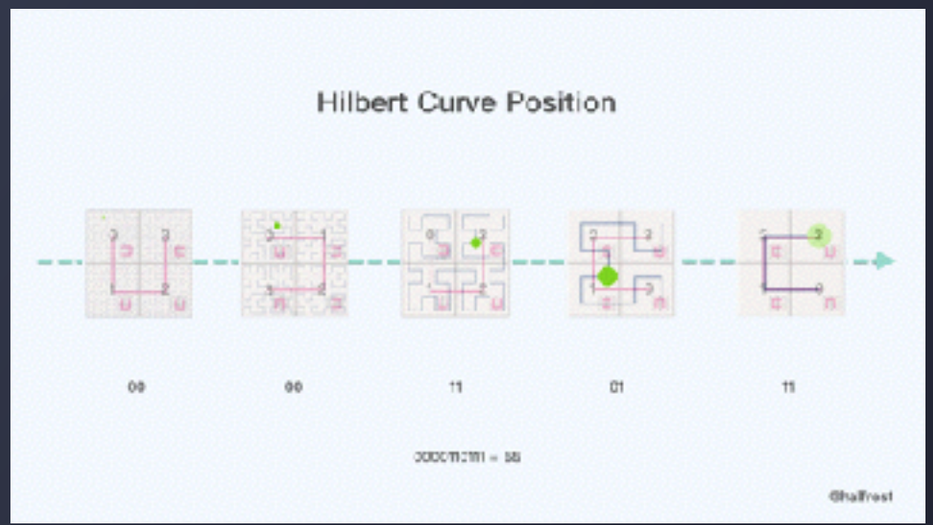
图2



轴旋转左右倒置

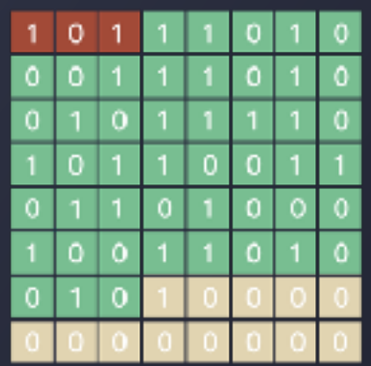
图3

HILBERT CURVE



HILBERT CURVE LEVEL

[0,2^30−1] * [0,2^30−1]

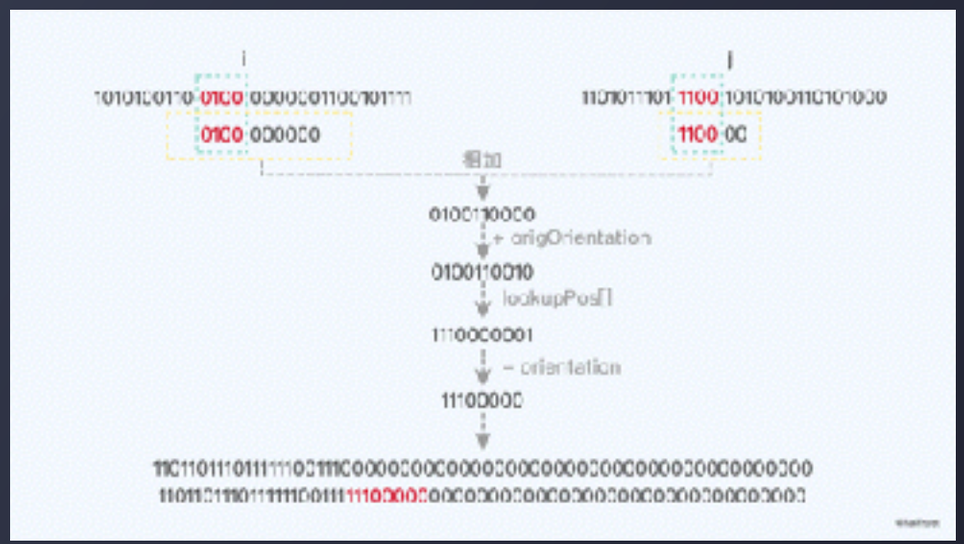


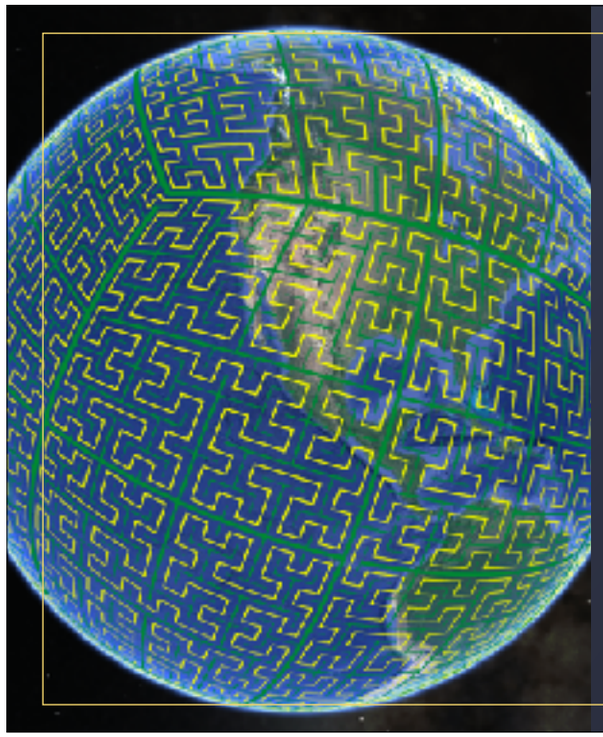
[0,2^24−1] * [0,2^24−1]

H(face,i,j) -> CellID

@halfrost

CELL ID

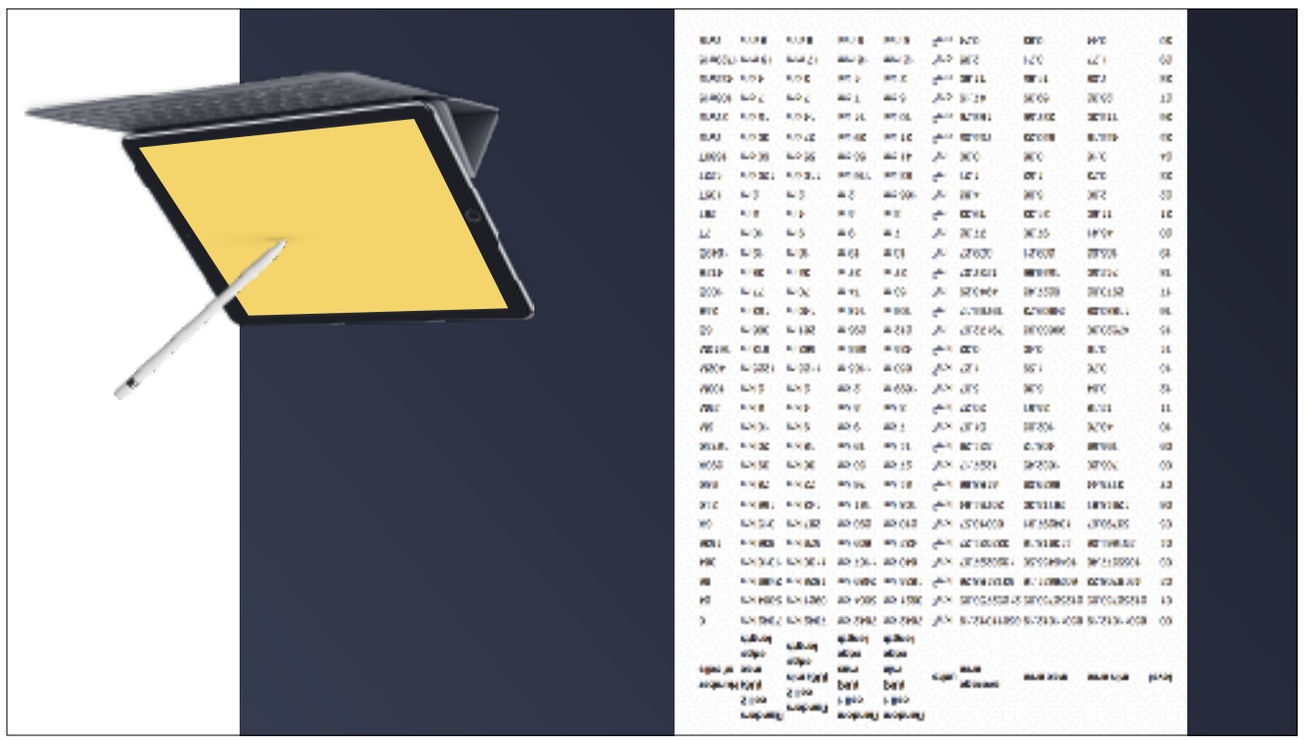

0100110000
+ origOrientation
0100110010
lookupPos[]
111000001
- orientation
11100000



GOOGLE S2

S(lat,lng) -> f(x,y,z) -> g(face,u,v) ->
h(face,s,t) -> H(face,i,j) -> CellID

NEXT SLIDE



GEOHASH VS GOOGLE S2

LEVEL 精细度

Geohash 有12级，从5000km 到 3.7cm。中间每一级的变化比较大。有时候可能选择上一级会大很多，选择下一级又会小一些。比如选择字符串长度为4，它对应的 cell 宽度是39.1km，需求可能是50km，那么选择字符串长度为5，对应的 cell 宽度就变成了156km，瞬间又大了3倍了。Geohash 需要 12 bytes 存储

突变性

S2 有30级，从 0.7cm² 到 85,000,000km² 。S2 的存储只需要一个 uint64 即可存下

几何计算

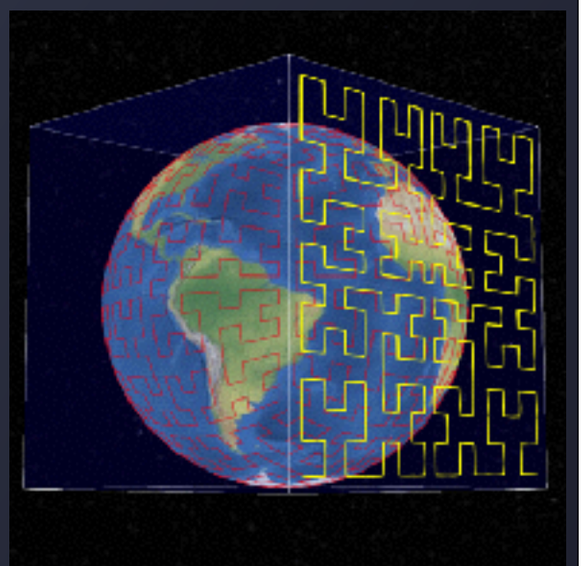
各种向量计算，面积计算，多边形覆盖，距离问题，球面球体上的问题

多边形覆盖

S2 还能解决多边形覆盖的问题

GOOGLE S2

1. 涉及到角度，间隔，纬度经度点，单位矢量等的表示，以及对这些类型的各种操作。
2. 单位球体上的几何形状，如球冠（“圆盘”），纬度 - 经度矩形，折线和多边形。
3. 支持点，折线和多边形的任意集合的强大的构造操作（例如联合）和布尔谓词（例如，包含）。
4. 对点，折线和多边形的集合进行快速的内存索引。
5. 针对测量距离和查找附近物体的算法。
6. 用于捕捉和简化几何的稳健算法（该算法具有精度和拓扑保证）。
7. 用于测试几何对象之间关系的有效且精确的数学谓词的集合。
8. 支持空间索引，包括将区域近似为离散“S2单元”的集合。此功能可以轻松构建大型分布式空间索引。



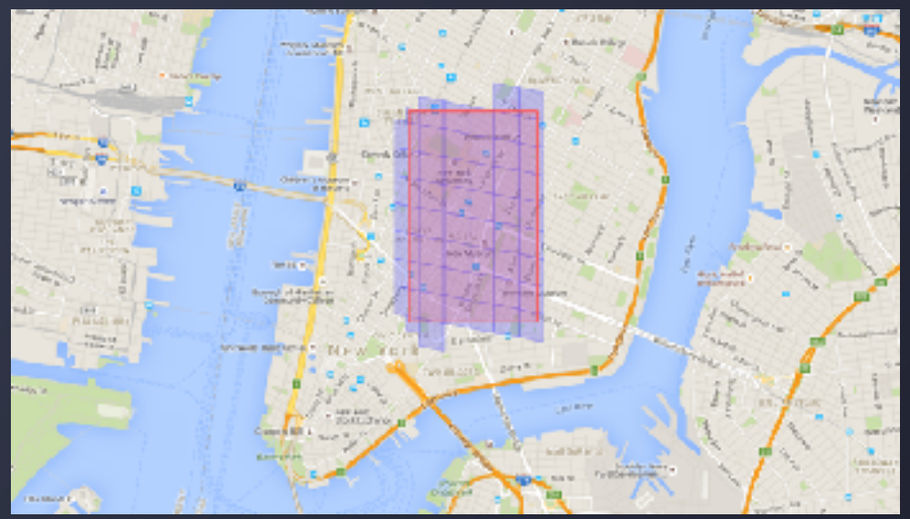
FINAL

APPLICATION

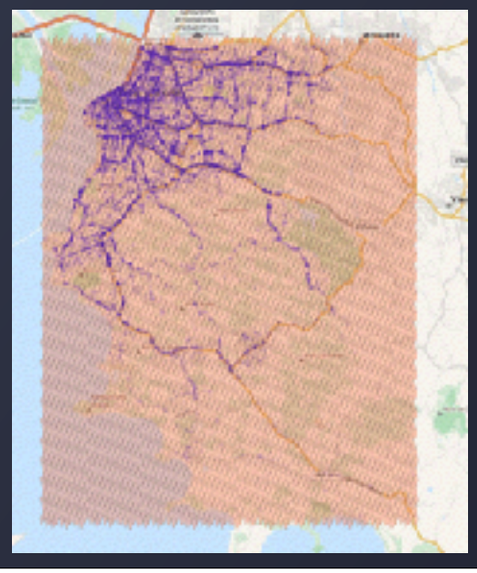
NEXT SLIDE

#04

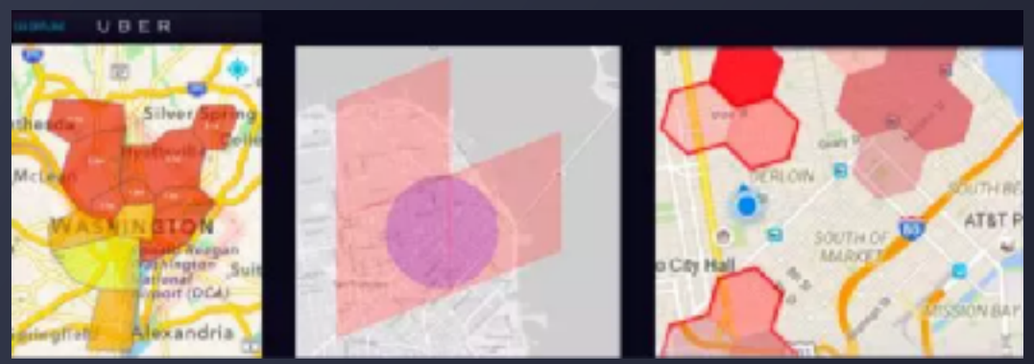
APPLICATION



APPLICATION

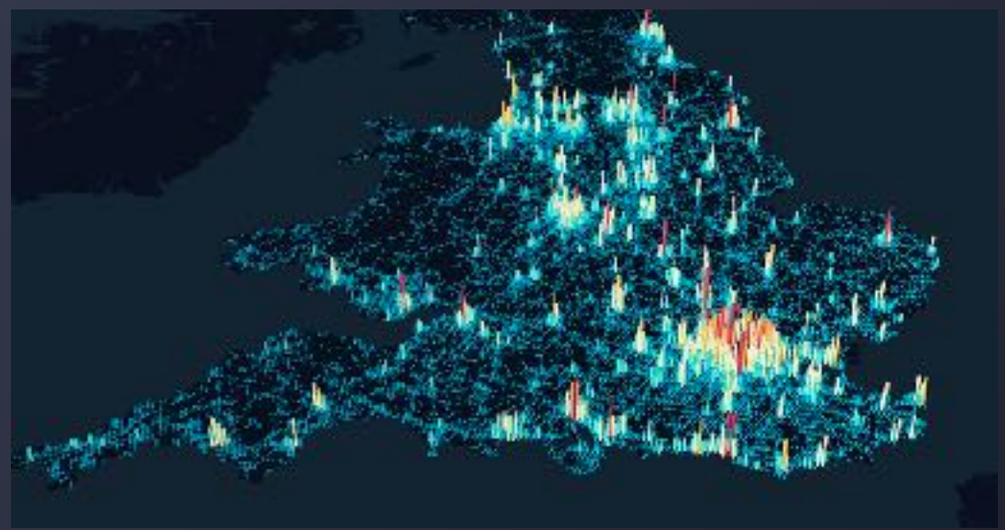


APPLICATION

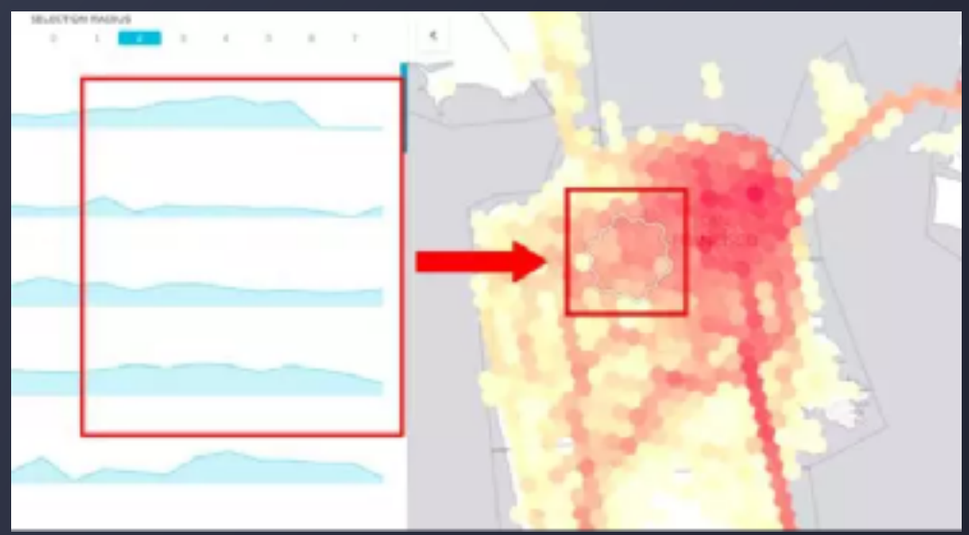


APPLICATION

APPLICATION



APPLICATION



APPLICATION

http://uber.github.io/deck.gl/#/
http://vonwolfehaus.github.io/von-grid/editor/

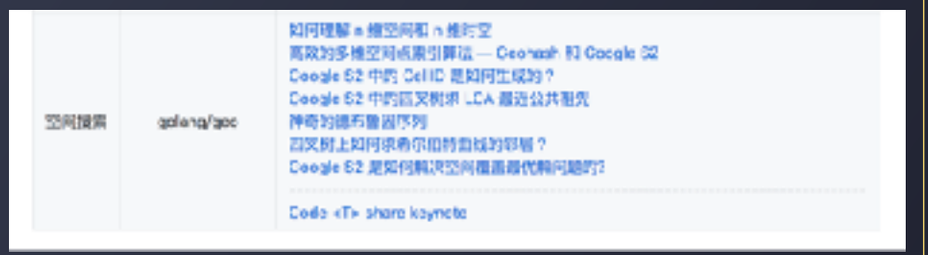
APPLICATION

流量是每秒钟大概数万条消息，一天大概是几亿，并且每条消息包含几十个字段

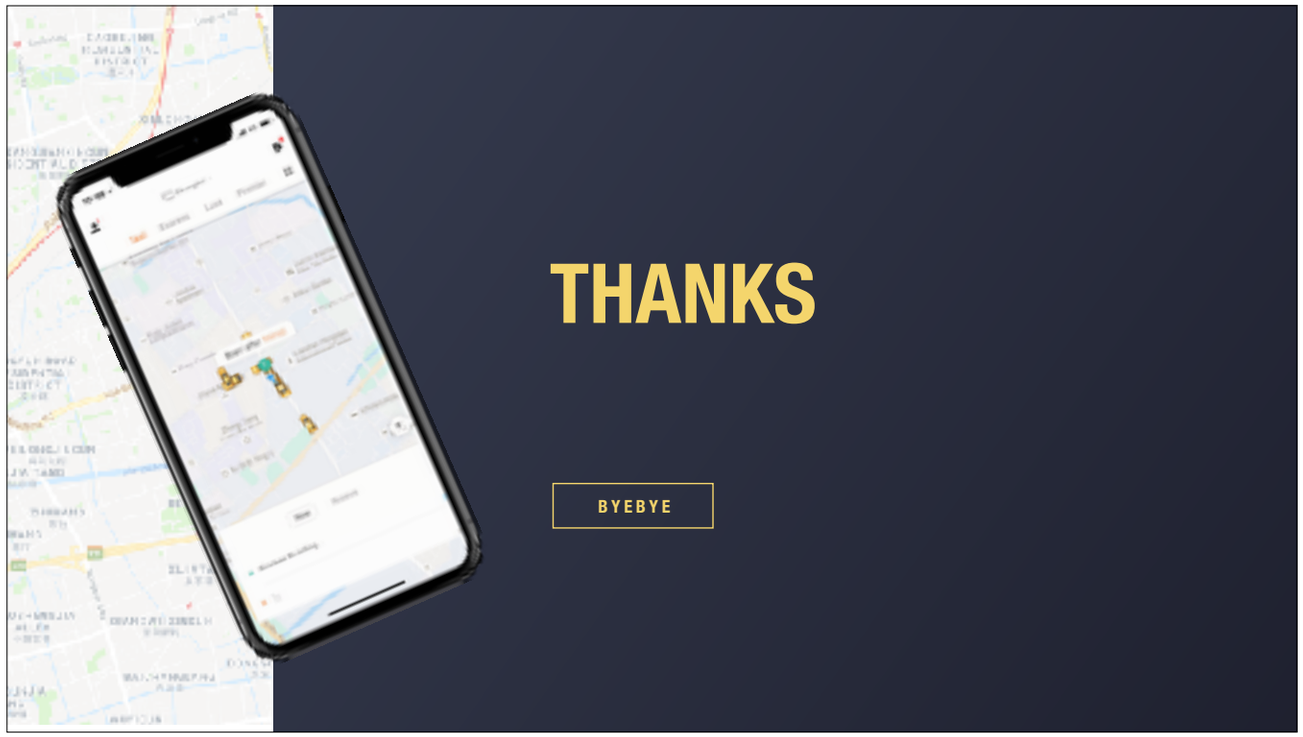
1.支持时序和地理空间的切片
2.支持大流量数据
3.支持秒级(毫秒级？)查询
4.支持原始数据查询

ElasticSearch + Kafka

ONE MORE THING



https://github.com/halfrost/Halfrost-Field



THANKS

BYEBYE