
# Space-Filling Curves and The Art of Composition

**FOSS4G NA 2019 -- Jim Hughes**
and Chris Eichelberger

"I remember you had a talk today.  I know it will go well, how could it not?"
-- Mom

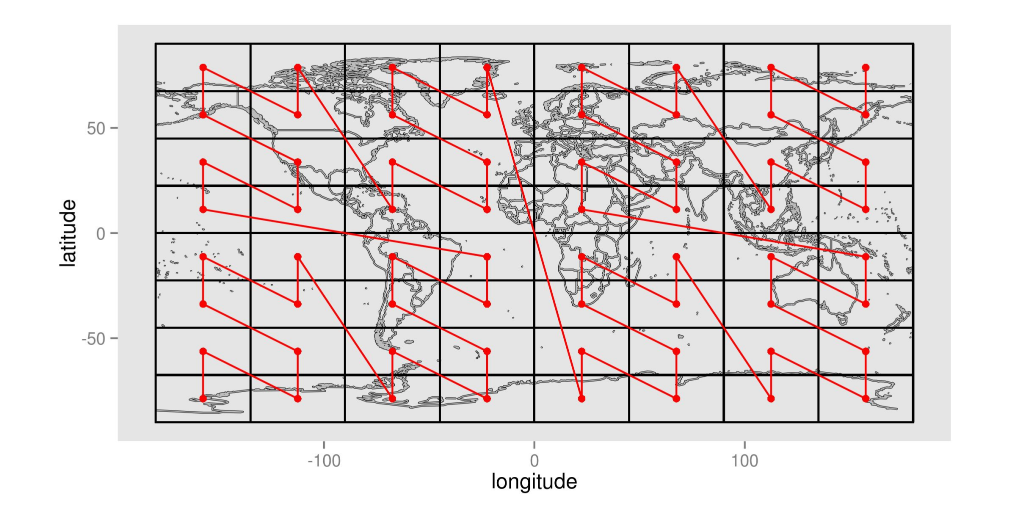
# outline

- motivation
- composition
- implementation
- analysis
- closing thought

# motivation

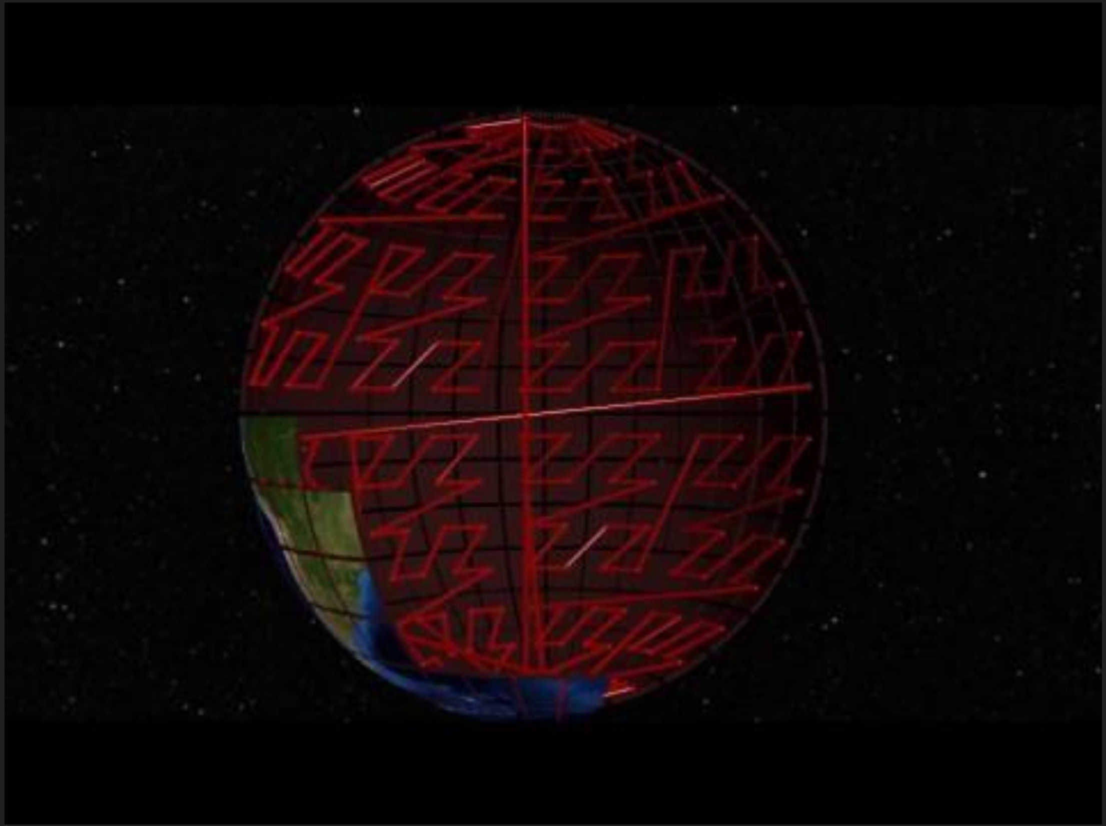
# why should I care about space-filling curves?

- space-filling curves are one way to create a geographical index
    - they flatten 2- or 3-D continuous spaces into single-dimensional cell sequences




- "where do you live?"  "Cell #26/64"

# how does this work on a round Earth?



# isn't there a FOSS library for this?

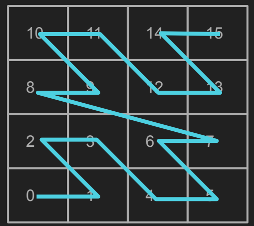
- yes, yes there is...
  - SFCurve:  a LocationTech technology within the Eclipse Foundation
- real FOSS projects use it...
  - GeoMesa
  - GeoWave
  - GeoTrellis
  - ...
- it's excellent, and enables large-scale geo-indexing and querying

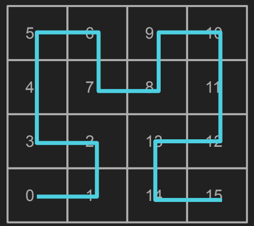
# what are the main duties of an SFC?

- index:  given a point in user-space, return the ordinate of the corresponding cell in index space
  - English:  tell me which box number my point is in
- inverse index:  given a cell in index space, describe its bounds in user space
  - English:  given my box number, what's its minimum and maximum longitude, latitude, time, ...
- find query ranges:  given a shape in user space, return the list of contiguous ordinates in index space
  - English:  given my county, tell me what index ranges I need to look up in my database

# what are SFCurve's limitations?

- only quadtree curves are supported, and only Z-order and Hilbert





- dimensions are fixed:  Z(x,y); Z(x,y,t); H(x,y)
  - only the number of bits precision can be changed
- these curves and dimensions don't always match the shape of the data you have or the type of queries you often perform
- fortunately, there was a way to solve these issues...

# composition

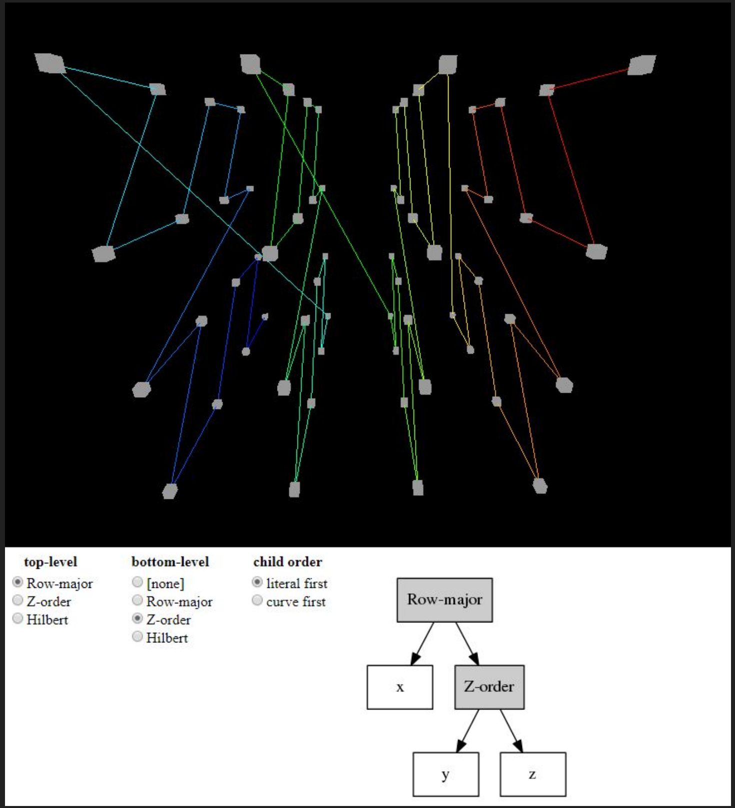
# composition... you mean like math?

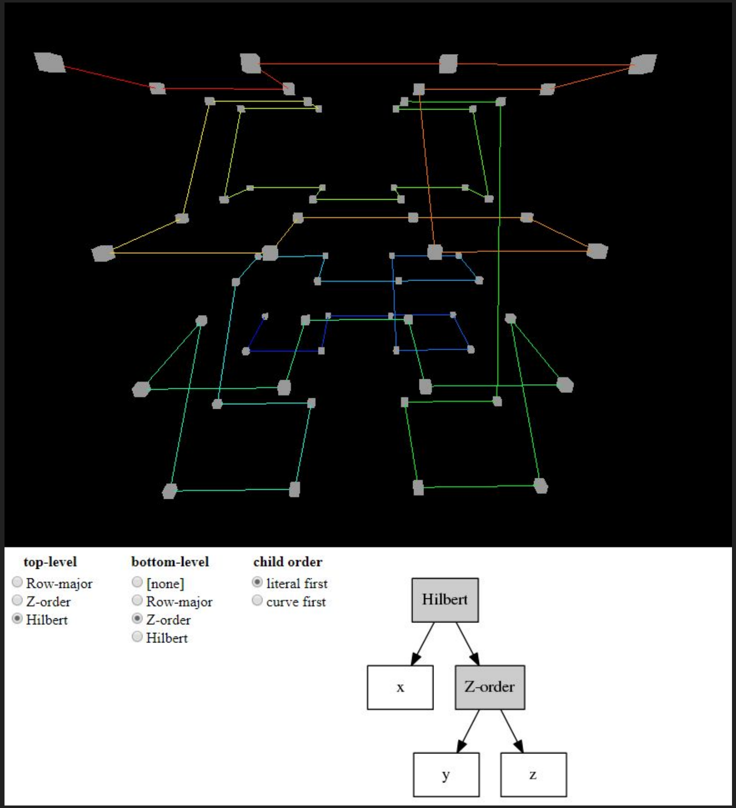
- say you have two functions
  - f(x) = x * x
  - g(y) = x + 1
- composition means nesting them together
  - f(g(w)) = (x + 1) * (x + 1) = x^2 + 2x + 1
  - g(f(w)) = x^2 + 1
- note:  the order of nesting can change the result!
- composition results in a *new function*

# SFCs are like functions

- the most abstract signature of an SFC is
  - index:  <values> → integer
  - inverse index:  integer → <values>
- so if we have
  - z2:  a two-dimensional Z-order curve
  - z3:  a three-dimensional Z-order curve
  - h:  a two-dimensional Hilbert curve
  - x, y, t:  single dimensions
- we could write
  - z3(x, y, t)
  - z2(h(x, y), t)
  - h(x, z2(y, t))
  - ...

# what do composed SFCs look like?



# implementation

# new interfaces (at least the important ones)

- DimensionLike[T]
  - can convert a (value, extent, cardinality) into a bin index
  - can convert a (bin index, extent, cardinality) into an Extent
- Discretizor(arity, cardinality)
  - can convert a sequence of values, and return a bin index
  - can convert a bin index into a Cell (a collection of Extents)
  - can convert a rectangle into an Iterator[IndexRange]
- Dimension[T : DimensionLike](cardinality) extends Discretizor(1, cardinality)
  - has static instances like Longitude(cardinality), Latitude(cardinality) from [-a, a]
  - has static instances like LonNonNeg(cardinality), LatNonNeg(cardinality) from [0, 2a]
- RangeConsolidator
  - given Iterator[IndexRange] and maxGap, return the minimum Iterator[IndexRange]
  - there's a tree-based, static instance InMemoryRangeConsolidator for convenience

# changes to existing interfaces

- SFC extends Discretizor with RangeConsolidator
  - has children, each of which is a Discretizor
    - could be a Dimension
    - could be another SFC
  - defines new, abstract functions that must be overridden
    - fold: Seq[Long] → Long
    - unfold: Long → Seq[Long
  - redefines old functions in terms of the new capabilities
    - index(Seq[_]) → Long: defers to **fold** after asking children to convert values to child indexes
    - inverseIndex(Long) → Seq[_]: defers to **unfold**, and then asks children to convert child indexes back into value ranges

# this is a little bit boring

here's a dancing clown GIF...

courtesy of https://gifimage.net/clown-gif-3/

# analysis

# what does composition do for us?

- our initial limitations are all solved
  - only quadtree curves are supported, and only Z-order and Hilbert
    - supports curves with arbitrary cardinality, like Row-Major (any) and Peano (base 3)
  - dimensions are fixed:  Z(x,y); Z(x,y,t); H(x,y)
    - dimensions can be *anything*, so long as it's discretizable
  - these curves and dimensions don't always match the shape of the data you have or the type of queries you often perform
    - you can now choose to learn, craft, or fit a composed SFC to fit your data and queries
- we gain some new capabilities
  - non-linear discretizations (per dimension) are easy
  - index spaces no longer need be rectangular

# TANSTAAFL: computing query ranges slows down

- de-composing query ranges is expensive
  - consider R(Z(x,a), H(y, t, b))
  - try to query over [x0..x1, a0..a1, y0..y1, t0..t1, b0..b1]
    - first, you need the query ranges for Z; call the result z0..z1
      - these are not single values; each z-item is a contiguous range of indexes
    - second, you need the ranges for H; call the result h0..h1
      - these are not single values; each h-item is a contiguous range of indexes
    - to compute ranges in R, you must
      - compute the cross-product of each of the z-ranges with each of the h-ranges
      - compute the R-ranges for each of the contiguous regions within the intersection of this single member of the z/h cross-product
      - consolidate the ranges, probably ordering them post-generation
- contiguous ranges must now be emitted in order; that's not always the fastest way to compute them
  - worst case, they have to be sorted while they are consolidated (which is what the InMemoryRangeConsolidator does)

# TANSTAAFL:  you can have dead cells

- imagine a curve like Z(x, H(t,x))

| | | |
|---|---|---|
| x | 1 | 2 |
| | 0 | 3 |
| | t | |

| | | | | |
|---|---|---|---|---|
| | 5 | 6 | 9 | 10 |
| | 4 | 7 | 8 | 11 |
| H(t,x) | 3 | 2 | 13 | 12 |
| | 0 | 1 | 14 | 15 |
| | | x | | |

- if t=0, then H(t,x)={2,3}
  - this implies that, no matter the value of x, Z(x, H(t,x)) cannot return 0-3 or 12-15
  - this isn't Earth-shattering, but it can be exceedingly wasteful at high cardinality

closing thought

# ... and how much longer is this relevant?

SFCurve and composition are tools with a lot of horsepower...

... just like horse-drawn carriages.

How much longer until *learned indexes* supplant artisanal, imperative indexes?

# links and contact information:  find us!

- the SFCurve project within LocationTech is supported by
  - CCRi
  - Azavea
- SFCurve is used for indexing by
  - GeoMesa
  - GeoTrellis
  - GeoWave
  - ...
- for more information, contact:


Jim Hughes
jim.hughes@ccri.com

Chris Eichelberger
cne1x@ccri.com

# BACKUP SLIDES

# geo-temporal indexing

- space-filling curves are useful for geo-indexing
  - GeoMesa
  - GeoTrellis
  - GeoWave
  - ...
- CCRi and Azavea created the SFCurve project under LocationTech to share a library / implementation

# so... isn't this a solved problem?

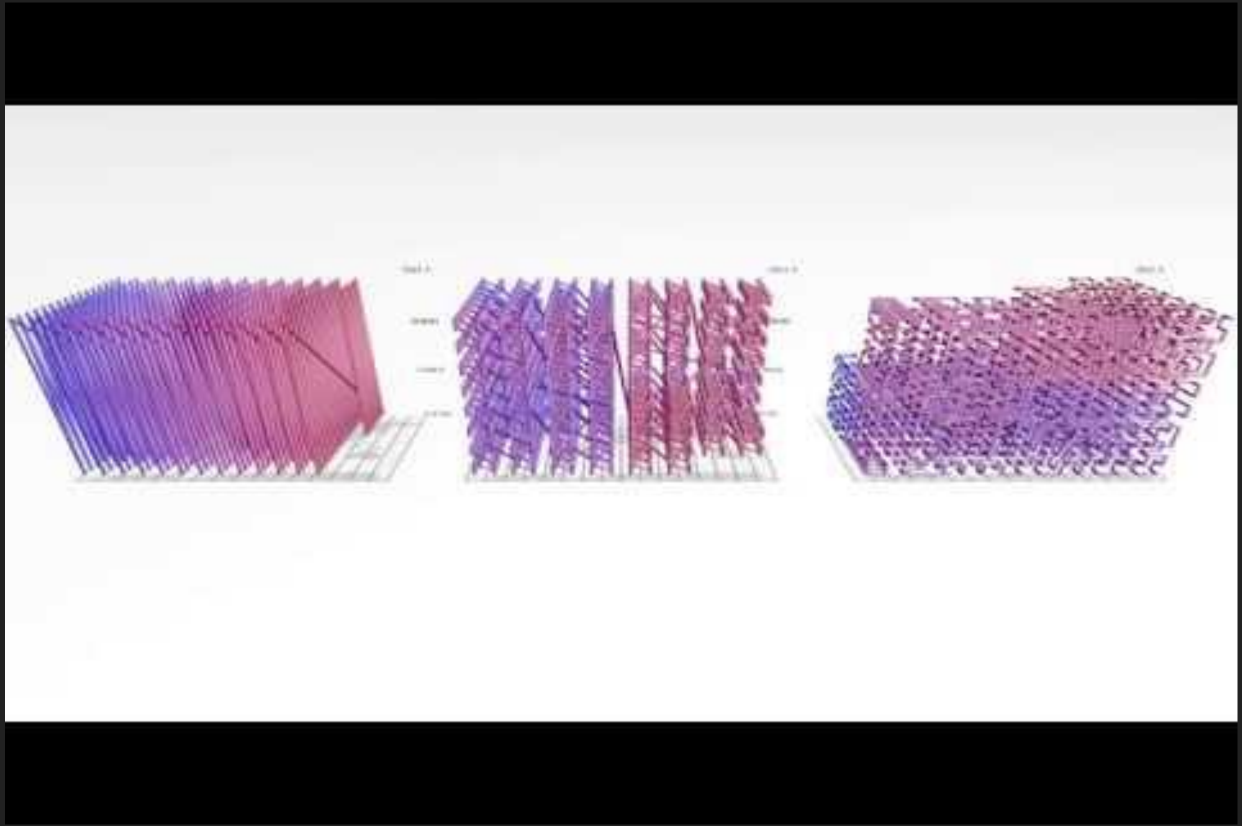
- it's code... it can always be better
- GeoMesa is *using* composed SFCs now, but only implicitly
  - no, that's not an intentional Scala thing
  - please don't tell Anthony; he gets upset about this being true-but-pointless
- we tend to describe curves with bits-precision rather than cardinality
  - this is an artificial constraint on what curve types we can use, because not all are quad-trees
- Jim, Tim, and Eichelbooger have been plotting for *years*

# the resulting TO DO list

- introduce a new abstraction into SFCurve that
  - makes dimensions first-class objects
  - allows for the arbitrary composition of dimensions and SFCs
  - supports more general SFCs than simply quad-trees
- change the existing public API as little as possible
- "standard" software development best practices
  - use this presentation as the basis for a design document to review *before* the MR
  - regression tests must pass
    - if there are no regression tests, they must be added
  - every new / modified capability must have a unit test
  - the performance impact must be minimal

# main ideas

# animation of R, Z, H curves

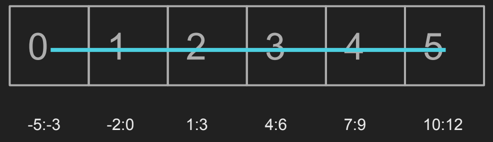


# space-filling curves

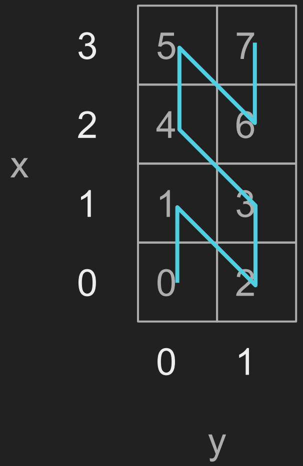
- divide a continuous space into discrete cells
- provide an explicit ordering of those cells

# space-filling curves

- divide a continuous space into discrete cells
- provide an explicit ordering of those cells
- examples:




range



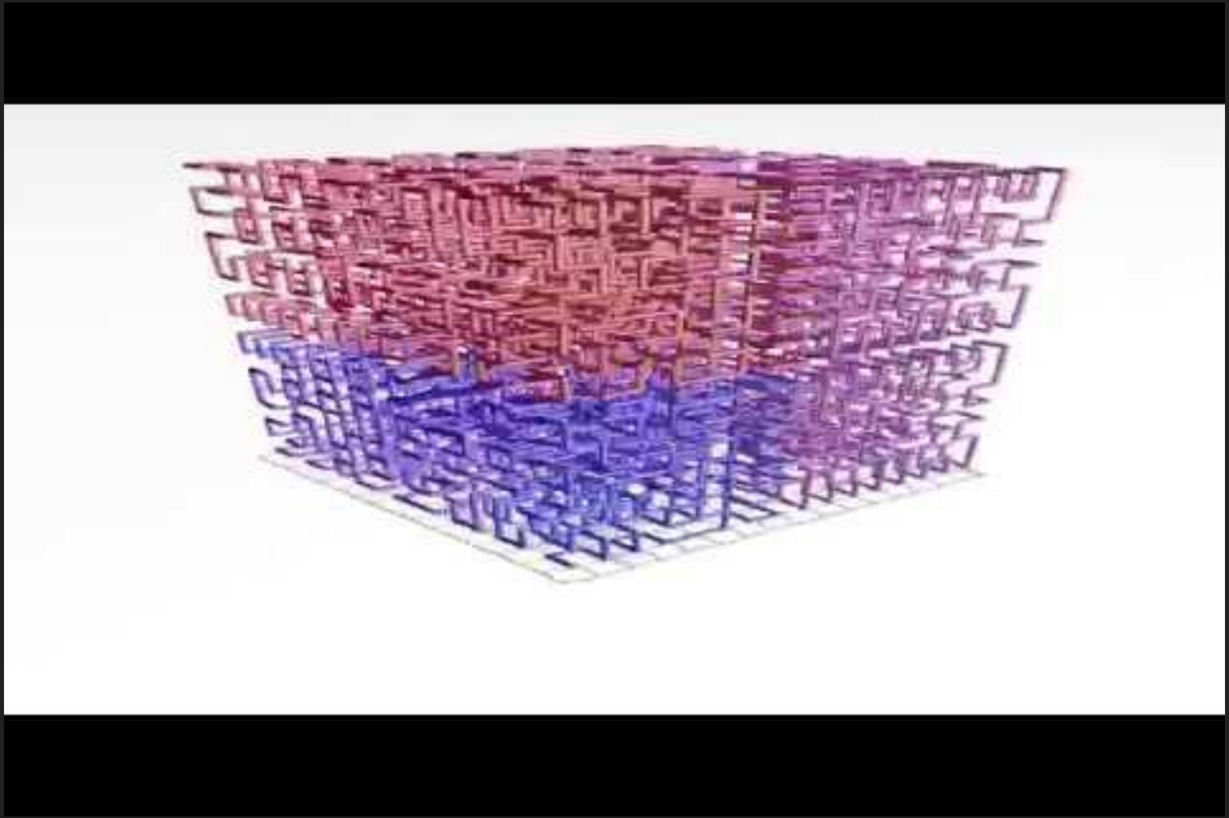
# space-filling curves, the usual suspects

- Row-Major: naive; "sort by latitude, then by longitude"
- Z-order / Morton: our for GeoMesa
  - quad-tree
  - fast to compute, because it's just interleaving bits
  - some yucky query-ranges, because of some large $|cell_n - cell_{n+1}|$ jumps
- (Compact) Hilbert: GeoWave's quad-tree choice
  - quad-tree
  - slow to compute, because it requires rotations and flips
  - nice query-ranges, because all $|cell_n - cell_{n+1}|$ transitions are one unit long
- Peano
  - base-3 recursive
- ...

# space-filling curves in GeoMesa

- allow (cells of) points to be ordered explicitly
- means that a query over latitude and longitude can be translated into one or more ranges of 1D keys assigned by Z(x,y)
- we use the (LocationTech) SFCurve library
  - Z2, Z3, H2 are fully implemented:  yay!
  - they all contain hard-coded dimensions:  boo!
  - they use precision in terms of bits, not cardinality:  boo!

# even sillier animation of query-range planning



# space-filling curves in GeoMesa *for realz*

- we use keys that have a time-value (ordinate) prefix and a geo(-time) suffix
- this isn't *really* a pure Z2 curve...  what is it?

# space-filling curves in GeoMesa *for realz*

- we use keys that have a time-value (ordinate) prefix and a geo(-time) suffix
- this isn't *really* a pure Z2 curve... what is it?
  - it's a a row-major curve that combines:
    - time
    - a Z2 over x and y
  - R(t, Z(x, y))

# space-filling curves in GeoMesa *for realz*

- we use keys that have a time-value (ordinate) prefix and a geo(-time) suffix
- this isn't *really* a pure Z2 curve...  what is it?
  - it's a a row-major curve that combines:
    - time
    - a Z2 over x and y
  - R(t, Z(x, y))

Why can't we do this more naturally?  Why do we have to hard-code it?

# proposal

# compose space-filling curves naturally

- discretizor*
  - maps between a continuous space and a discrete space
  - has cardinality
  - can compute query ranges
- dimension
  - a one-dimensional discretizor tied to a single type
- space-filling curve
  - a multi-dimension, untyped discretizor
  - has children that are discretizors:  dimension | SFC
  - can fold an ordered list of (child) bin-numbers into a single (parent) bin number
  - can unfold a single (parent) bin number into an ordered list of (child) bin numbers

*Apologies for the English.  I struggled to find a better term, and failed.  If you know of one, *please* share.  Terminology is fair game in a design / code review...

# compose space-filling curves naturally

- That's all it takes:
  - Discretizor
    - Dimension[T]
    - SpaceFillingCurve
- We can now recognize GeoMesa's scheme as:
  - R(dT(t), Z(dX(x), dY(y)))
- We can easily express more exotic curves, should we chose:
  - H(Z(dX(x), dT(t)), R(dT(t), dY(y)))

# why would I *ever* want to do this?!

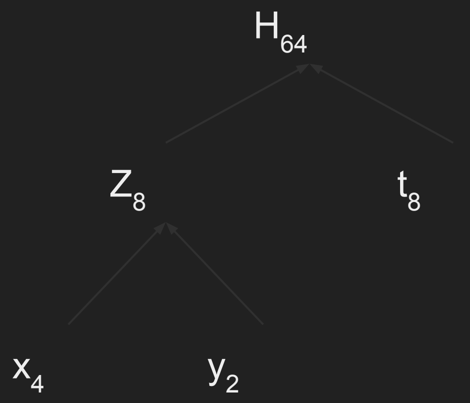
- why does GeoMesa already support so many indexes?
- ... because there isn't any single index that performs well for all queries
- see http://eichelberger.org/sfseize/index.html

simple walk-through

# simple walk-through

(pro tip:  never trust "simple")

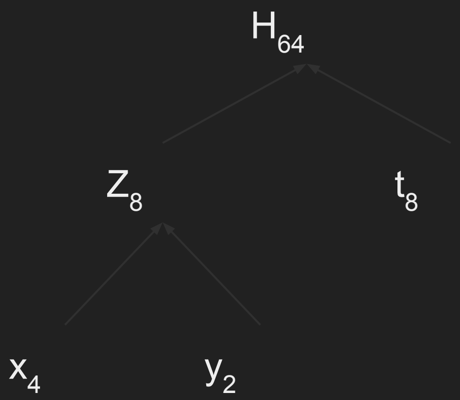
# simple walk-through




assume these cardinalities and ranges:

- $x_4$: longitude on [-180, 180), four bins:
  - [-180, -90)
  - [-90, 0)
  - [0, 90)
  - [90, 180)
- $y_2$: latitude on [-90, 90], two bins:
  - [-90, 0)
  - [0, 90]
- $t_8$:  time ranges from 2000-01-01 2100-12-31, eight (big!) bins:
  - [2000-01-01, 2012-07-01)
  - [2012-07-01, 2024-12-31)
  - [2024-12-31, 2037-07-01)
  - [2037-07-01, 2049-12-31)
  - [2049-12-31, 2062-07-01)
  - [2062-07-01, 2074-12-31)
  - [2074-12-31, 2087-07-01)
  - [2087-07-01, 2099-12-31)

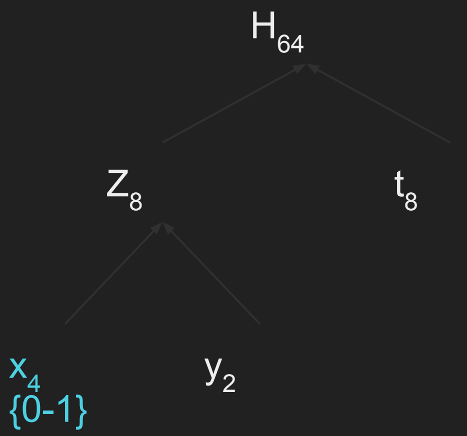
# simple walk-through




query:

DURING 2018-12-10 / 2018-12-14
AND geom.x BETWEEN -100 AND -80
AND geom.y BETWEEN 30 AND 40

# simple walk-through



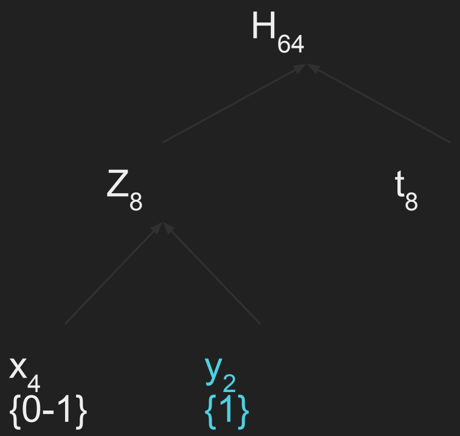

query:

DURING 2018-12-10 / 2018-12-14
AND geom.x BETWEEN -100 AND -80
AND geom.y BETWEEN 30 AND 40

The longitude range spans two discrete bins on this dimension:

0. [-180, -90)
1. [-90, 0)
2. [0, 90)
3. [90, 180)

# simple walk-through



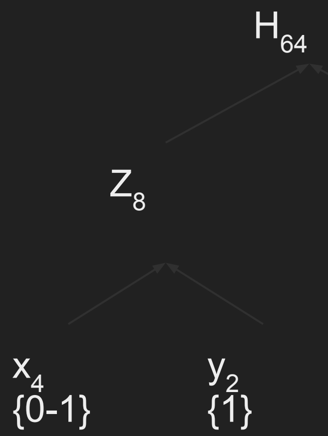

query:

DURING 2018-12-10 / 2018-12-14
AND geom.x BETWEEN -100 AND -80
AND geom.y BETWEEN 30 AND 40

The latitude range spans one full discrete bin on this dimension:

0. [-90, 0)
1. [0, 90]

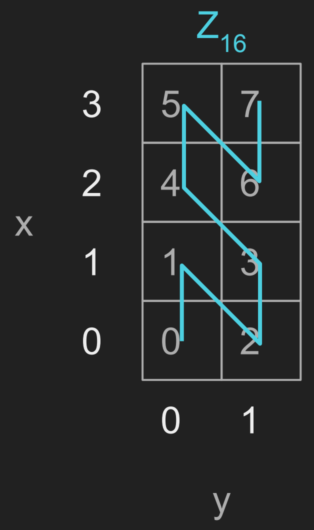
# simple walk-through



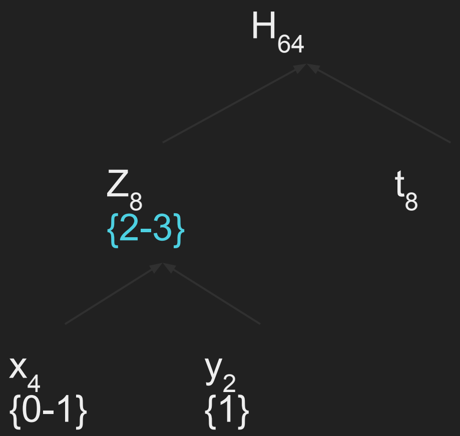

query:

DURING 2018-12-10 / 2018-12-14
AND geom.x BETWEEN -100 AND -80
AND geom.y BETWEEN 30 AND 40

The Z2 curve progresses like this:



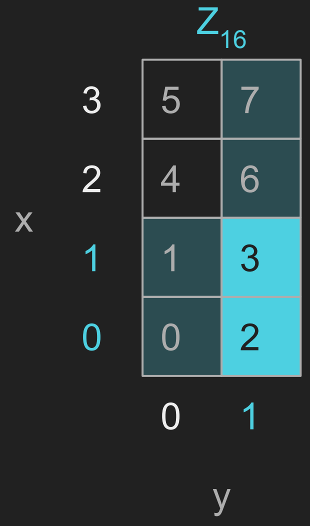
# simple walk-through



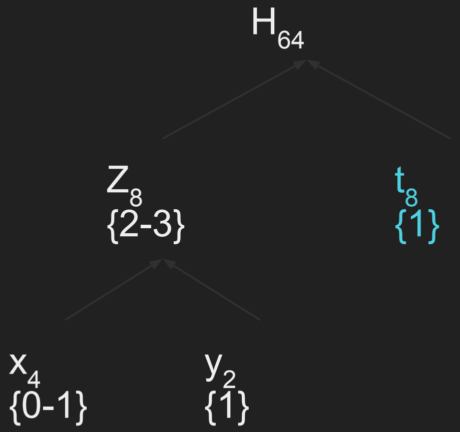

query:

DURING 2018-12-10 / 2018-12-14
AND geom.x BETWEEN -100 AND -80
AND geom.y BETWEEN 30 AND 40

The Z2 curve returns these query ranges:



# simple walk-through



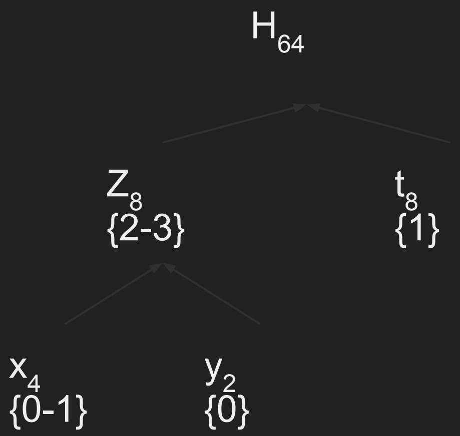

query:

DURING 2018-12-10 / 2018-12-14
AND geom.x BETWEEN -100 AND -80
AND geom.y BETWEEN 30 AND 40

The time range falls entirely within a single discrete bin on this dimension:

0. [2000-01-01, 2012-07-01)
1. [2012-07-01, 2024-12-31)
2. [2024-12-31, 2037-07-01)
3. [2037-07-01, 2049-12-31)
4. [2049-12-31, 2062-07-01)
5. [2062-07-01, 2074-12-31)
6. [2074-12-31, 2087-07-01)
7. [2087-07-01, 2099-12-31)

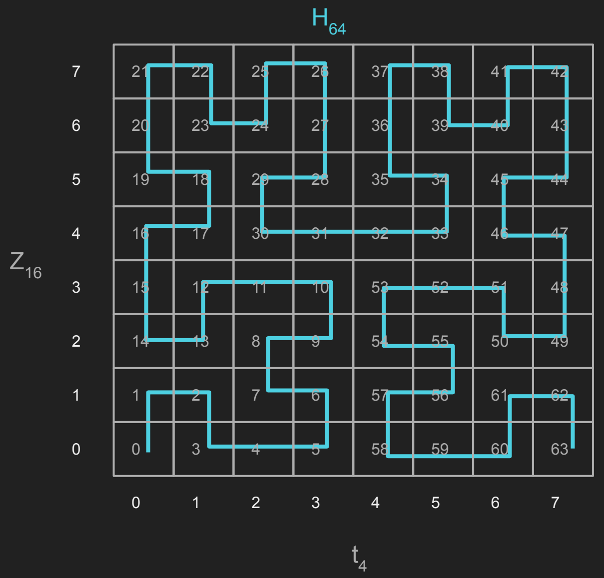
# simple walk-through



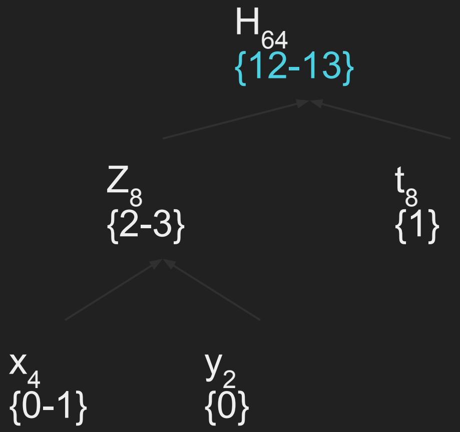

query:

DURING 2018-12-10 / 2018-12-14
AND geom.x BETWEEN -100 AND -80
AND geom.y BETWEEN 30 AND 40

The H2 curve progresses like this:



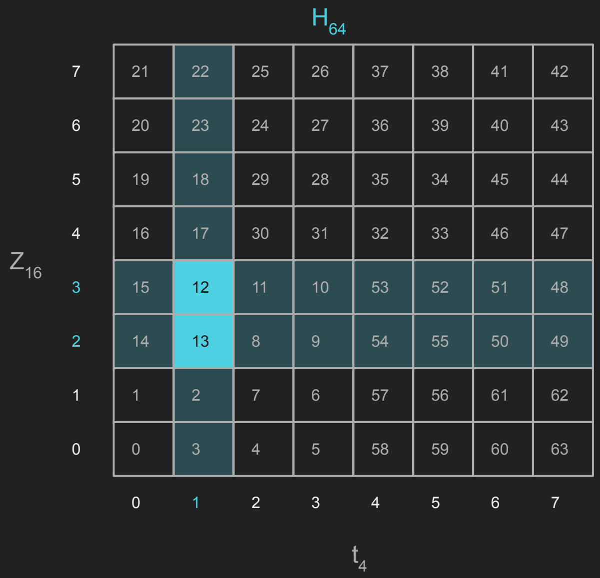
# simple walk-through



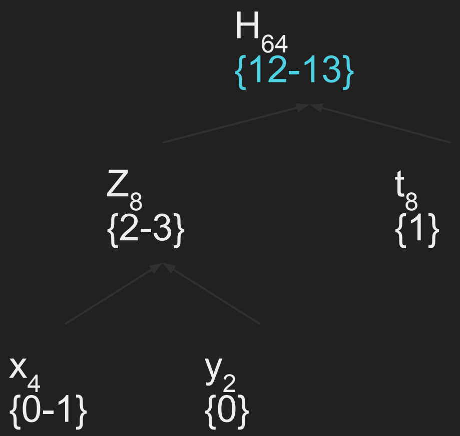

query:

DURING 2018-12-10 / 2018-12-14
AND geom.x BETWEEN -100 AND -80
AND geom.y BETWEEN 30 AND 40

The H2 curve returns these query ranges:



# simple walk-through




observation:

- the number of cells in a parent must equal the product of the number of cells across all children
  - x=2 * y=1 -> z=2
  - z=2 * t=1 -> h=2

$H_{64}$

| $Z_{16}$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 7 | 21 | 22 | 25 | 26 | 37 | 38 | 41 | 42 |
| 6 | 20 | 23 | 24 | 27 | 36 | 39 | 40 | 43 |
| 5 | 19 | 18 | 29 | 28 | 35 | 34 | 45 | 44 |
| 4 | 16 | 17 | 30 | 31 | 32 | 33 | 46 | 47 |
| 3 | 15 | 12 | 11 | 10 | 53 | 52 | 51 | 48 |
| 2 | 14 | 13 | 8 | 9 | 54 | 55 | 50 | 49 |
| 1 | 1 | 2 | 7 | 6 | 57 | 56 | 61 | 62 |
| 0 | 0 | 3 | 4 | 5 | 58 | 59 | 60 | 63 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

$t_4$

code review (**pre**-MR)

# SFCurve at LocationTech

- branch in progress: https://github.com/cne1x/sfcurve/tree/f_composition
- dimension-like type classes: https://github.com/cne1x/sfcurve/blob/f_composition/api/src/main/scala/org/locationtech/sfcurve/Dimensions.scala#L20
- Dimension: https://github.com/cne1x/sfcurve/blob/f_composition/api/src/main/scala/org/locationtech/sfcurve/Dimensions.scala#L83
- SFC: https://github.com/cne1x/sfcurve/blob/f_composition/api/src/main/scala/org/locationtech/sfcurve/Dimensions.scala#L159

# that's all, folks

- if you have any **questions** or **complaints**, please contact

    Jim Hughes
    jhughes@ccri.com

- if you have any **lauds** or **adulation**, please contact

    Chris Eichelberger
    cne1x@ccri.com