SQL is a Programming Language, Summer 2018

Handwriting Recognition
with SQL (and a tiny bit of web stuff)

Noah Doersing

A long time ago in a galaxy very, very close…

In the early 60s, keyboard proficiency was less widespread, so other input methods were explored.

- Alan Kay, "Doing with Images Makes Symbols" (1987):[1]
  "None of us can type, can you do something about that?"

[1] http://archive.org/details/AlanKeyD1987?start=1439.5

Instead of implementing an entire interface paradigm, only consider *core problem*: **Recognition of a single-stroke character.**

Slight modifications to some characters to make this work.

Simple D3.js-based web version, connected to a PostgreSQL instance running on my server. **Try it on your phones:**

`hejnoah.com/handwriting/`

It yields a JSON-encoded *time series of coordinate pairs*. In order to transfer this into a 🐘 database…

1. Define variable **pen** via command-line parameter.

```
psql -v pen='[{"x":1, "y":3},{"x":3, "y":7}]' -f handwriting.sql
```

2. Convert JSON array into tabular representation.

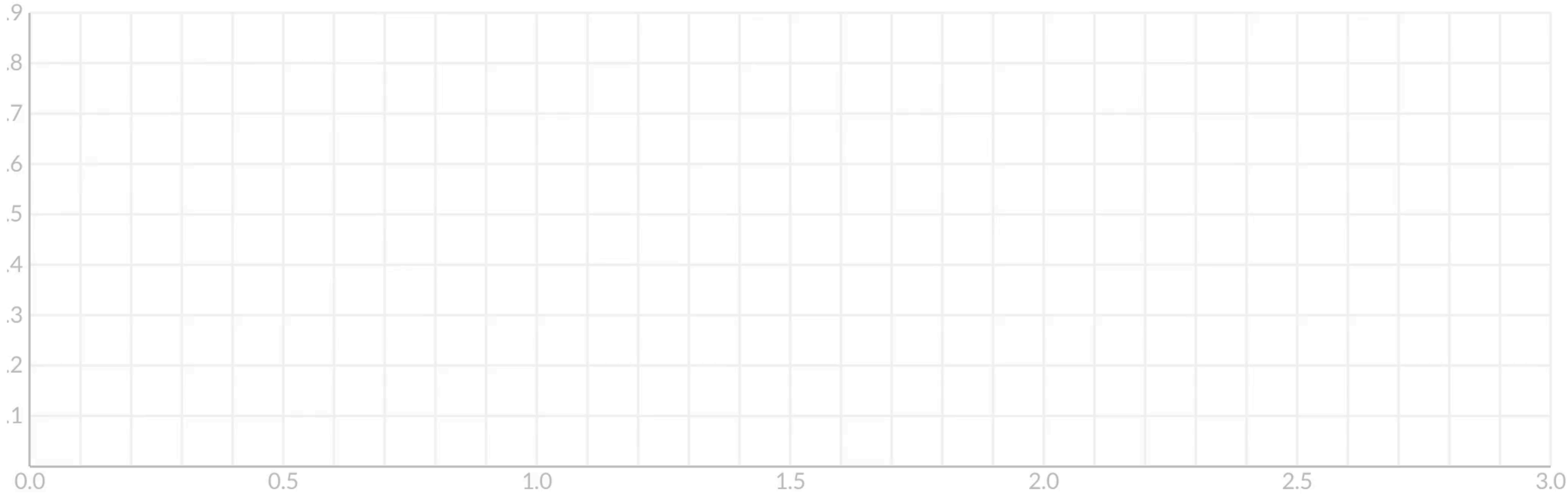
```
WITH RECURSIVE
tablet(pos, x, y) AS (
  SELECT ordinality AS pos, x, y
  FROM   ROWS FROM(jsonb_to_recordset(:'pen') AS (x int, y int)) WITH ORDINALITY
), ...
```

Approach[2]

1. Smoothing & thinning of stroke
2. Curvature & corner detection
3. Extraction of additional features
4. Descending a hardcoded *decision tree* mapping features to characters

[2] https://www.rand.org/pubs/research_memoranda/RM5016.html, http://jackschaedler.github.io/handwriting-recognition/

Smoothing: Removes *quantization noise*. Compute weighted average of most recently smoothed point and incoming point.



```
smooth(pos, x, y) AS (
  SELECT pos, x :: real, y :: real
  FROM   tablet
  WHERE  pos = 1

    UNION ALL

  SELECT t.pos,
         (:smoothingfactor * s.x + (1.0 - :smoothingfactor) * t.x) :: real AS x,
         (:smoothingfactor * s.y + (1.0 - :smoothingfactor) * t.y) :: real AS y
  FROM   smooth s, tablet t
  WHERE  t.pos = s.pos + 1
),
```

Variable **`smoothingfactor`** must be set before running the query.
Sensible values: between **`0.5`** and **`0.8`**.

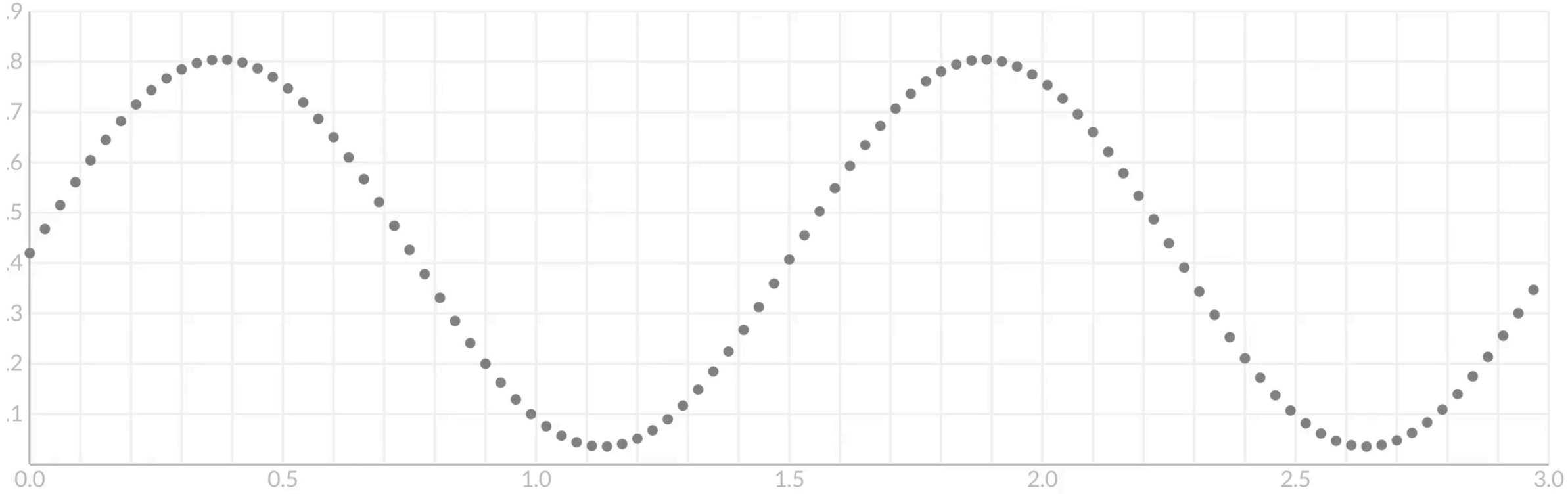
```
smooth(pos, x, y) AS (
  SELECT pos, x :: real, y :: real
  FROM   tablet
  WHERE  pos = 1

    UNION ALL

  SELECT t.pos,
         (:smoothingfactor * s.x + (1.0 - :smoothingfactor) * t.x) :: real AS x,
         (:smoothingfactor * s.y + (1.0 - :smoothingfactor) * t.y) :: real AS y
  FROM   smooth s, tablet t
  WHERE  t.pos = s.pos + 1
),
```

Variable **smoothingfactor** must be set before running the query.
Sensible values: between **0.5** and **0.8**.

```
smooth(pos, x, y) AS (
  SELECT pos, x :: real, y :: real
  FROM   tablet
  WHERE  pos = 1

    UNION ALL

  SELECT t.pos,
         (:smoothingfactor * s.x + (1.0 - :smoothingfactor) * t.x) :: real AS x,
         (:smoothingfactor * s.y + (1.0 - :smoothingfactor) * t.y) :: real AS y
  FROM   smooth s, tablet t
  WHERE  t.pos = s.pos + 1
),
```

Variable **smoothingfactor** must be set before running the query.
Sensible values: between **0.5** and **0.8**.

Thinning: Eases further *processing requirements*. Reject points within a certain distance from the most recent accepted point.



```
thin(pos, x, y) AS (
  SELECT *
  FROM   smooth
  WHERE  pos = 1

    UNION ALL

  SELECT *
  FROM   (
    SELECT s.pos, s.x, s.y
    FROM   thin t, smooth s
    WHERE  s.pos > t.pos
    AND    :thinningsize < |/ (s.x - t.x)^2 + (s.y - t.y)^2
    ORDER BY s.pos
    LIMIT 1
  ) AS _
),
```

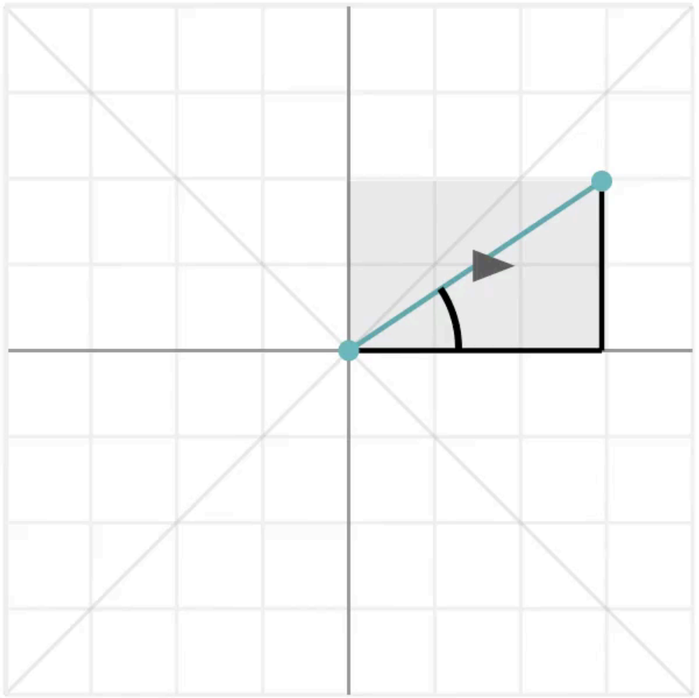
```
thin(pos, x, y) AS (
  SELECT *
  FROM   smooth
  WHERE  pos = 1

    UNION ALL

  SELECT *
  FROM   (
    SELECT s.pos, s.x, s.y
    FROM   thin t, smooth s
    WHERE  s.pos > t.pos
    AND    :thinningsize < |/ (s.x - t.x)^2 + (s.y - t.y)^2
    ORDER BY s.pos
    LIMIT 1
  ) AS _
),
```

```
thin(pos, x, y) AS (
  SELECT *
  FROM   smooth
  WHERE  pos = 1

    UNION ALL

  SELECT *
  FROM   (
    SELECT s.pos, s.x, s.y
    FROM   thin t, smooth s
    WHERE  s.pos > t.pos
    AND    :thinningsize < |/ (s.x - t.x)^2 + (s.y - t.y)^2
    ORDER BY s.pos
    LIMIT 1
  ) AS _
),
```

Curvature detection: Compute *cardinal directions* ▲▼◀▶ of line segments between point pairs. Discard sequential duplicates.

Originally done using a set of *inequalities*, but *trigonometry* is more elegant and reasonably fast on modern hardware.

Recall: $\boldsymbol{tangent} = \dfrac{\boldsymbol{opposite\ leg}}{\boldsymbol{adjacent\ leg}}$



1. Compute angle between every pair of thinned points using window function **`lag()`**:

```
curve(pos, x, y, direction) AS (
  SELECT pos, x, y,
         degrees(-atan2( y - lag(y) OVER (ORDER BY pos),
                        -x + lag(x) OVER (ORDER BY pos))) + 180
  FROM   thin
),
```

2. Define appropriate **ENUM** type with fancy Unicode triangles:

```
CREATE TYPE cardinal_direction AS ENUM('▶', '▲', '◀', '▼');
```

3. Use **enum_range()** function (returns the values of its argument's **ENUM** type as an array) to convert **angle / 90** into arrow symbols.

```
cardinal(pos, direction) AS (
  SELECT pos,
         (enum_range(NULL :: cardinal_direction))[(direction / 90) :: int % 4
                                                  + 1]
  FROM   curve
),
```

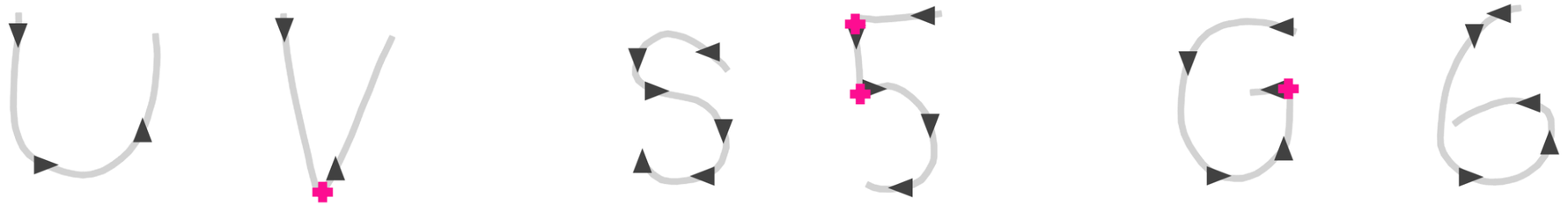
4. Only keep changes of the cardinal direction, i.e. a *new* direction that, additionally, occurs at least *twice in succession*.

```
cardinal_change(pos, direction) AS (
  SELECT pos, direction
  FROM   cardinal
  WHERE  COALESCE(lag(direction, 2) OVER win <> lag(direction) OVER win,
                  true)  -- Prevent NULLs from escaping this term.
  AND    lag(direction) OVER win = direction
  WINDOW win AS (ORDER BY pos)
),
```

…except this doesn't work because window functions cannot be used in **WHERE** clauses. Resort to an awkward subquery.

```
cardinal_change(pos, direction) AS (
  SELECT pos, direction
  FROM   (SELECT pos, direction,
                 COALESCE(lag(direction, 2) OVER win <> lag(direction) OVER win,
                          true)  -- Prevent NULLs from escaping this term.
                 AND lag(direction) OVER win = direction
          FROM cardinal
          WINDOW win AS (ORDER BY pos)) AS _(pos, direction, is_new)
  WHERE  is_new
),
```

Corner detection: Allows discerning between *similar characters*.



Corners lie between two line segments going in the *same direction* and two segments going in a *wildly different direction*, with an optional in-between "turn" segment.

1. Define function that computes angle difference (trivial for angle pairs like 10° and 20°, but less so for 350° and 30°).

```
CREATE OR REPLACE FUNCTION angdiff(alpha double precision,
                                   beta double precision) RETURNS real AS $$
  SELECT abs(degrees(atan2(sin(radians(alpha - beta)),
                           cos(radians(alpha - beta))))) :: real;
  --SELECT (180 - abs(abs(alpha - beta) - 180)) :: real;
$$ LANGUAGE SQL IMMUTABLE;
```

2. Detect corners with the help of another awkward subquery.

```
corner(pos, x, y) AS (
  SELECT pos, x, y
  FROM   (SELECT pos, x, y, (
                   angdiff(lag(direction, 2) OVER win, lag(direction) OVER win) < 22.5
                   AND angdiff(lag(direction) OVER win, direction) > :cornerangle
                   AND angdiff(direction, lead(direction) OVER win) < 22.5
                 ) OR (  -- Immediate direction change OR one-segment turn.
                   angdiff(lag(direction, 3) OVER win, lag(direction, 2) OVER win) < 22.5
                   AND angdiff(lag(direction, 2) OVER win, direction) > :cornerangle
                   AND angdiff(direction, lead(direction) OVER win) < 22.5
                 ) AS is_corner
          FROM   curve
          WINDOW win AS (ORDER BY pos)) AS _(pos, x, y, is_corner)
  WHERE  is_corner
),
```

2. Detect corners with the help of another awkward subquery.

```
corner(pos, x, y) AS (
  SELECT pos, x, y
  FROM   (SELECT pos, x, y, (
                   angdiff(lag(direction, 2) OVER win, lag(direction) OVER win) < 22.5
                   AND angdiff(lag(direction) OVER win, direction) > :cornerangle
                   AND angdiff(direction, lead(direction) OVER win) < 22.5
                 ) OR (  -- Immediate direction change OR one-segment turn.
                   angdiff(lag(direction, 3) OVER win, lag(direction, 2) OVER win) < 22.5
                   AND angdiff(lag(direction, 2) OVER win, direction) > :cornerangle
                   AND angdiff(direction, lead(direction) OVER win) < 22.5
                 ) AS is_corner
          FROM   curve
          WINDOW win AS (ORDER BY pos)) AS _(pos, x, y, is_corner)
  WHERE  is_corner
),
```

2. Detect corners with the help of another awkward subquery.

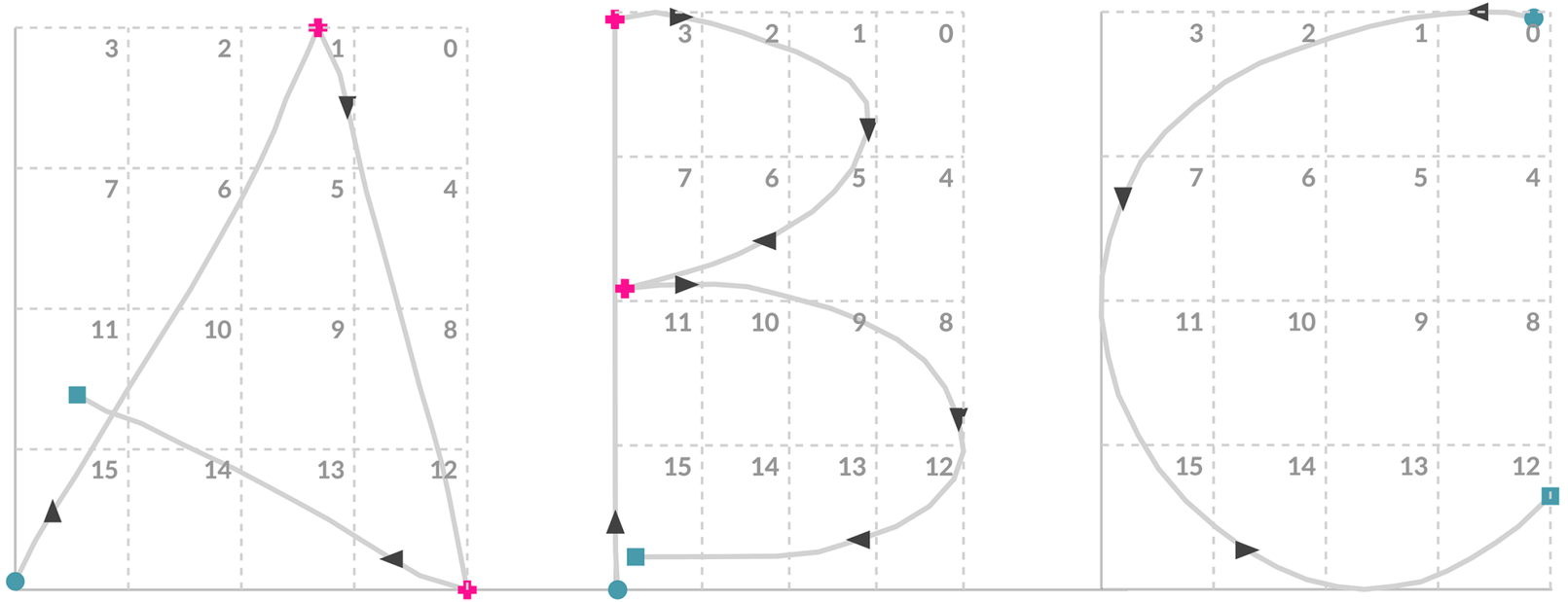
```
corner(pos, x, y) AS (
  SELECT pos, x, y
  FROM   (SELECT pos, x, y, (
                   angdiff(lag(direction, 2) OVER win, lag(direction) OVER win) < 22.5
                   AND angdiff(lag(direction) OVER win, direction) > :cornerangle
                   AND angdiff(direction, lead(direction) OVER win) < 22.5
                 ) OR (  -- Immediate direction change OR one-segment turn.
                   angdiff(lag(direction, 3) OVER win, lag(direction, 2) OVER win) < 22.5
                   AND angdiff(lag(direction, 2) OVER win, direction) > :cornerangle
                   AND angdiff(direction, lead(direction) OVER win) < 22.5
                 ) AS is_corner
          FROM   curve
          WINDOW win AS (ORDER BY pos)) AS _(pos, x, y, is_corner)
  WHERE  is_corner
),
```

2. Detect corners with the help of another awkward subquery.

```
corner(pos, x, y) AS (
  SELECT pos, x, y
  FROM   (SELECT pos, x, y, (
                   angdiff(lag(direction, 2) OVER win, lag(direction) OVER win) < 22.5
                   AND angdiff(lag(direction) OVER win, direction) > :cornerangle
                   AND angdiff(direction, lead(direction) OVER win) < 22.5
                 ) OR (  -- Immediate direction change OR one-segment turn.
                   angdiff(lag(direction, 3) OVER win, lag(direction, 2) OVER win) < 22.5
                   AND angdiff(lag(direction, 2) OVER win, direction) > :cornerangle
                   AND angdiff(direction, lead(direction) OVER win) < 22.5
                 ) AS is_corner
          FROM   curve
          WINDOW win AS (ORDER BY pos)) AS _(pos, x, y, is_corner)
  WHERE  is_corner
),
```

Feature extraction: Extract some *supplementary features* that will help discern between different characters later on: Start point, end point, aspect ratio, …

Transform from absolute pixel positions to 4x4 grid segments.



1. Define *axis-aligned bounding box* around pen stroke and gather some statistics.

```
aabb(xmin, xmax, ymin, ymax, aspect, width, height, centerx, centery) AS (
  SELECT min(x),
         max(x),
         min(y),
         max(y),
         (max(y) - min(y)) / greatest(1, (max(x) - min(x))),  -- Prevent n/0.
         max(x) - min(x),
         max(y) - min(y),
         min(x) + (max(x) - min(x)) / 2,
         min(y) + (max(y) - min(y)) / 2
  FROM   smooth
),
```

2. Define function for *transforming* coordinates of start, end and corners into grid positions.

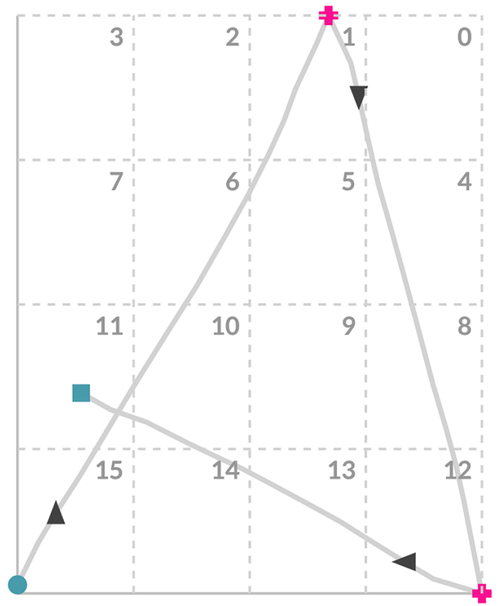
```
CREATE OR REPLACE FUNCTION gridpos(width real, height real,
                                   xmin real, ymin real,
                                   x real, y real) RETURNS int AS $$
  SELECT greatest(0,
                  15 - (      (floor(4 * (x - xmin) / (width  + 1)) :: int)
                        + 4 * (floor(4 * (y - ymin) / (height + 1)) :: int)));
$$ LANGUAGE SQL IMMUTABLE;
```

3. Apply this function and thus create CTEs **start_grid**, **stop_grid** and **corner_grid**. Not shown.

4. Accumulate extracted features. This single-row table contains **all information required to identify the drawn character**.

```
features(directions, start, stop, corners, width, height, aspect, center) AS (
  SELECT (SELECT array_agg(direction ORDER BY pos)
          FROM   cardinal_change),
         (TABLE start_grid),
         (TABLE stop_grid),
         (SELECT COALESCE(array_agg(n ORDER BY pos), '{}')
          FROM   corner_grid),
         width,
         height,
         aspect,
         point(centerx, centery)
  FROM   aabb
),
```

directions	start	stop	corners	width	height	aspect	center
{▲,▼,◀}	15	11	{1,12}	36.7595	66.1965	1.8008	(55.3797,64.0982)



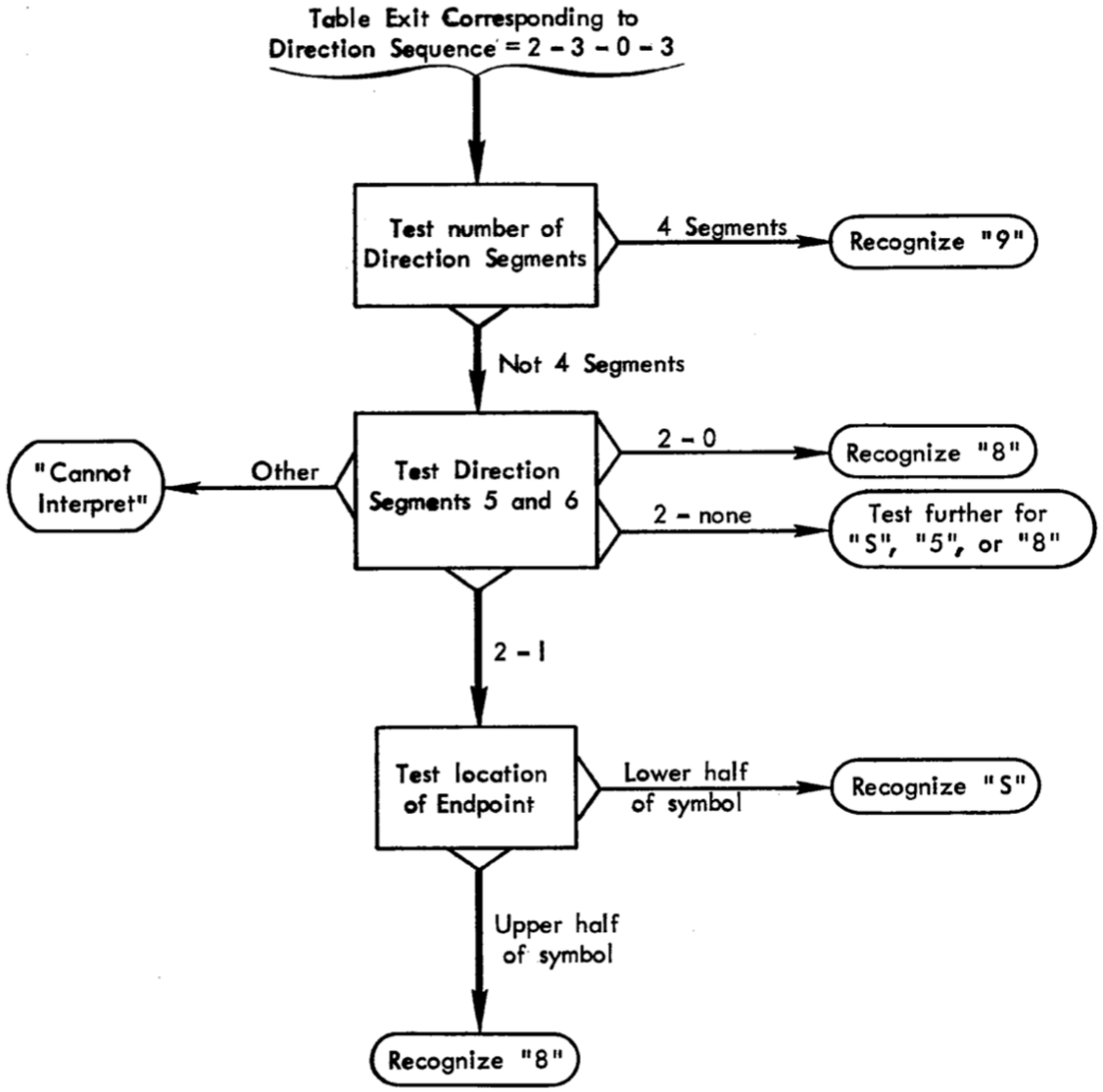


FIG. 6 - Recognition of the Track of Figure 2.

Decision tree: Depending on the first four cardinal directions, descend custom-tailored subtree.

Incredibly *awkward* in SQL. Imagine 1000+ lines of deeply nested and probably buggy `CASE WHEN`s.

A better approach is to do this in *two discrete stages*:

1. Determine **candidate characters** based on the pen stroke's *first four cardinal directions*.
2. Settle on **best fit** by matching *selected stroke features*.

Less flexible than the procedure presented in the original memo, but significantly more *idiomatic* and concise.

1. Determine initial set of potential characters.

```
CREATE TABLE lookup_candidates (
  first_four_directions cardinal_direction[],
  candidate_characters  char[]
);
INSERT INTO lookup_candidates VALUES
  ('{"▼"}',                '{"I"}'),
  ('{"▼","◀"}',            '{"J"}'),
  ('{"▼","◀","▲"}',        '{"O","J","X","U"}'),
  ('{"▼","◀","▲","▶"}',    '{"X","O","U"}'),
  ('{"▼","◀","▶","▲"}',    '{"X"}'),
  ('{"▼","▶"}',            '{"L"}'),
  ('{"▼","▶","◀"}',        '{"6"}'),
  ('{"▼","▶","◀","▼"}',    '{"4"}'),
  ('{"▼","▶","▲"}',        '{"O","U"}'),
  ...
```

2. Find best fit by feature matching. Read **NULL**s as "don't care".

```
CREATE TABLE lookup_bestfit (
  candidate_characters char[],
  character            char,
  start                int,
  stop                 int,
  corners              int[],
  aspect_range         numrange
);
INSERT INTO lookup_bestfit  -- All single-character patterns from first lookup.
  ...
INSERT INTO lookup_bestfit VALUES
  ('{"M","N"}', 'M', NULL, 12, NULL, NULL),  -- End point bottom right.
  ('{"M","N"}', 'N', NULL, 0,  NULL, NULL),  -- End point top right.
  ...
```

3. Tie it all together: Consult *both* lookup tables in order.

```
character(character) AS (
  SELECT l.character
  FROM   features f, lookup_candidates c, lookup_bestfit l
  WHERE  f.directions[1:4] = c.first_four_directions
  AND    c.candidate_characters = l.candidate_characters
  AND    (l.start IS NULL          OR f.start = l.start)
  AND    (l.stop IS NULL           OR f.stop = l.stop)
  AND    (l.corners IS NULL        OR f.corners = l.corners)
  AND    (l.aspect_range IS NULL   OR l.aspect_range @> f.aspect :: numeric)
),
```

3. Tie it all together: Consult *both* lookup tables in order.

```
character(character) AS (
  SELECT l.character
  FROM   features f, lookup_candidates c, lookup_bestfit l
  WHERE  f.directions[1:4] = c.first_four_directions
  AND    c.candidate_characters = l.candidate_characters
  AND    (l.start IS NULL          OR f.start = l.start)
  AND    (l.stop IS NULL           OR f.stop = l.stop)
  AND    (l.corners IS NULL        OR f.corners = l.corners)
  AND    (l.aspect_range IS NULL   OR l.aspect_range @> f.aspect :: numeric)
),
```

3. Tie it all together: Consult *both* lookup tables in order.

```
character(character) AS (
  SELECT l.character
  FROM   features f, lookup_candidates c, lookup_bestfit l
  WHERE  f.directions[1:4] = c.first_four_directions
  AND    c.candidate_characters = l.candidate_characters
  AND    (l.start IS NULL           OR f.start = l.start)
  AND    (l.stop IS NULL            OR f.stop = l.stop)
  AND    (l.corners IS NULL         OR f.corners = l.corners)
  AND    (l.aspect_range IS NULL    OR l.aspect_range @> f.aspect :: numeric)
),
```

3. Tie it all together: Consult *both* lookup tables in order.

```
character(character) AS (
  SELECT l.character
  FROM   features f, lookup_candidates c, lookup_bestfit l
  WHERE  f.directions[1:4] = c.first_four_directions
  AND    c.candidate_characters = l.candidate_characters
  AND    (l.start IS NULL          OR f.start = l.start)
  AND    (l.stop IS NULL           OR f.stop = l.stop)
  AND    (l.corners IS NULL        OR f.corners = l.corners)
  AND    (l.aspect_range IS NULL   OR l.aspect_range @> f.aspect :: numeric)
),
```

3. Tie it all together: Consult *both* lookup tables in order.

```
character(character) AS (
  SELECT l.character
  FROM   features f, lookup_candidates c, lookup_bestfit l
  WHERE  f.directions[1:4] = c.first_four_directions
  AND    c.candidate_characters = l.candidate_characters
  AND    (l.start IS NULL          OR f.start = l.start)
  AND    (l.stop IS NULL           OR f.stop = l.stop)
  AND    (l.corners IS NULL        OR f.corners = l.corners)
  AND    (l.aspect_range IS NULL   OR l.aspect_range @> f.aspect :: numeric)
),
```

Performance: Most algorithms designed in the 60's will run on modern hardware *just fine* — this one is no exception:

The whole chain of queries terminates after **~15 ms**.

SQL is, *after all*, a programming language.

```
                                                   QUERY PLAN
CTE Scan on debug  (cost=601.52..601.60 rows=4 width=172) (actual time=10.615..10.705 rows=1 loops=1)
  CTE tablet
    ->  Function Scan on jsonb_to_recordset  (cost=0.00..1.00 rows=100 width=16) (actual time=0.104..0.117 rows=64 loops=1)
  CTE smooth
    ->  Recursive Union  (cost=0.00..36.53 rows=101 width=16) (actual time=0.115..2.662 rows=64 loops=1)
          ->  CTE Scan on tablet  (cost=0.00..2.25 rows=1 width=16) (actual time=0.114..0.161 rows=1 loops=1)
                Filter: (pos = 1)
                Rows Removed by Filter: 63
          ->  Hash Join  (cost=0.33..3.23 rows=10 width=16) (actual time=0.025..0.037 rows=1 loops=64)
                Hash Cond: (t.pos = (s.pos + 1))
                ->  CTE Scan on tablet t  (cost=0.00..2.00 rows=100 width=16) (actual time=0.000..0.011 rows=64 loops=64)
                ->  Hash  (cost=0.20..0.20 rows=10 width=16) (actual time=0.002..0.002 rows=1 loops=64)
                      Buckets: 1024  Batches: 1  Memory Usage: 9kB
                      ->  WorkTable Scan on smooth s  (cost=0.00..0.20 rows=10 width=16) (actual time=0.000..0.001 rows=1 loops=64)
  CTE thin
    ->  Recursive Union  (cost=0.00..474.82 rows=11 width=16) (actual time=0.003..2.681 rows=23 loops=1)
          ->  CTE Scan on smooth  (cost=0.00..2.27 rows=1 width=16) (actual time=0.002..0.021 rows=1 loops=1)
                Filter: (pos = 1)
                Rows Removed by Filter: 63
          ->  Limit  (cost=47.22..47.22 rows=1 width=16) (actual time=0.113..0.113 rows=1 loops=23)
                ->  Sort  (cost=47.22..47.50 rows=112 width=16) (actual time=0.109..0.109 rows=1 loops=23)
                      Sort Key: s_1.pos
                      Sort Method: quicksort  Memory: 25kB
                      ->  Nested Loop  (cost=0.00..46.66 rows=112 width=16) (actual time=0.017..0.084 rows=39 loops=23)
                            Join Filter: ((s_1.pos > t_1.pos) AND ('5'::double precision < (|/ ((((s_1.x - t_1.x))::double precision ^ '2'::double precision) + (((s_1.y - t_1.y))::double precision ^ '2'::double precision)))))
                            Rows Removed by Join Filter: 25
                            ->  WorkTable Scan on thin t_1  (cost=0.00..0.20 rows=10 width=16) (actual time=0.000..0.000 rows=1 loops=23)
                            ->  CTE Scan on smooth s_1  (cost=0.00..2.02 rows=101 width=16) (actual time=0.000..0.016 rows=64 loops=23)
  CTE curve
    ->  WindowAgg  (cost=0.41..0.88 rows=11 width=24) (actual time=2.846..3.071 rows=23 loops=1)
          ->  Sort  (cost=0.41..0.44 rows=11 width=16) (actual time=2.797..2.810 rows=23 loops=1)
                Sort Key: thin.pos
                Sort Method: quicksort  Memory: 26kB
                ->  CTE Scan on thin  (cost=0.00..0.22 rows=11 width=16) (actual time=0.004..2.702 rows=23 loops=1)
  CTE cardinal
    ->  CTE Scan on curve  (cost=0.00..0.36 rows=11 width=12) (actual time=3.256..3.804 rows=23 loops=1)
  CTE cardinal_change
    ->  Subquery Scan on _  (cost=0.41..0.80 rows=6 width=12) (actual time=3.959..4.054 rows=3 loops=1)
          Filter: _.is_new
          Rows Removed by Filter: 20
          ->  WindowAgg  (cost=0.41..0.69 rows=11 width=13) (actual time=3.949..4.045 rows=23 loops=1)
                ->  Sort  (cost=0.41..0.44 rows=11 width=12) (actual time=3.929..3.933 rows=23 loops=1)
                      Sort Key: cardinal.pos
                      Sort Method: quicksort  Memory: 26kB
                      ->  CTE Scan on cardinal  (cost=0.00..0.22 rows=11 width=12) (actual time=3.260..3.842 rows=23 loops=1)
  CTE corner
    ->  Subquery Scan on __1  (cost=0.41..2.61 rows=6 width=16) (actual time=0.451..0.566 rows=2 loops=1)
          Filter: __1.is_corner
          Rows Removed by Filter: 21
          ->  WindowAgg  (cost=0.41..2.50 rows=11 width=17) (actual time=0.287..0.554 rows=23 loops=1)
                ->  Sort  (cost=0.41..0.44 rows=11 width=24) (actual time=0.102..0.106 rows=23 loops=1)
                      Sort Key: curve_1.pos
                      Sort Method: quicksort  Memory: 26kB
                      ->  CTE Scan on curve curve_1  (cost=0.00..0.22 rows=11 width=24) (actual time=0.001..0.055 rows=23 loops=1)
  CTE aabb
    ->  Aggregate  (cost=6.57..6.61 rows=1 width=44) (actual time=2.792..2.792 rows=1 loops=1)
          ->  CTE Scan on smooth smooth_1  (cost=0.00..2.02 rows=101 width=8) (actual time=0.116..2.707 rows=64 loops=1)
  CTE start_grid
    ->  Limit  (cost=3.56..3.61 rows=1 width=12) (actual time=0.113..0.114 rows=1 loops=1)
          ->  Result  (cost=3.56..8.86 rows=101 width=12) (actual time=0.106..0.106 rows=1 loops=1)
                ->  Sort  (cost=3.56..3.81 rows=101 width=32) (actual time=0.097..0.097 rows=1 loops=1)
                      Sort Key: s_2.pos
                      Sort Method: top-N heapsort  Memory: 25kB
                      ->  Nested Loop  (cost=0.00..3.05 rows=101 width=32) (actual time=0.007..0.053 rows=64 loops=1)
                            ->  CTE Scan on aabb a  (cost=0.00..0.02 rows=1 width=16) (actual time=0.001..0.002 rows=1 loops=1)
                            ->  CTE Scan on smooth s_2  (cost=0.00..2.02 rows=101 width=16) (actual time=0.001..0.017 rows=64 loops=1)
  CTE stop_grid
    ->  Limit  (cost=3.56..3.61 rows=1 width=12) (actual time=0.190..0.191 rows=1 loops=1)
          ->  Result  (cost=3.56..8.86 rows=101 width=12) (actual time=0.190..0.190 rows=1 loops=1)
                ->  Sort  (cost=3.56..3.81 rows=101 width=32) (actual time=0.185..0.185 rows=1 loops=1)
                      Sort Key: s_3.pos DESC
                      Sort Method: top-N heapsort  Memory: 25kB
                      ->  Nested Loop  (cost=0.00..3.05 rows=101 width=32) (actual time=0.002..0.105 rows=64 loops=1)
                            ->  CTE Scan on aabb a_1  (cost=0.00..0.02 rows=1 width=16) (actual time=0.001..0.001 rows=1 loops=1)
                            ->  CTE Scan on smooth s_3  (cost=0.00..2.02 rows=101 width=16) (actual time=0.000..0.022 rows=64 loops=1)
  CTE corner_grid
    ->  Unique  (cost=0.52..0.55 rows=6 width=12) (actual time=0.803..0.807 rows=2 loops=1)
          ->  Sort  (cost=0.52..0.53 rows=6 width=12) (actual time=0.802..0.804 rows=2 loops=1)
                Sort Key: (GREATEST(0, (15 - ((floor((('4'::double precision * (c.x - a_2.xmin)) / (a_2.width + '1'::double precision))))::integer + (4 * (floor((('4'::double precision * (c.y - a_2.ymin)) / (a_2.height + '1'::double precision))))::integ
                Sort Method: quicksort  Memory: 25kB
                ->  Nested Loop  (cost=0.00..0.44 rows=6 width=12) (actual time=0.473..0.593 rows=2 loops=1)
                      ->  CTE Scan on aabb a_2  (cost=0.00..0.02 rows=1 width=16) (actual time=0.003..0.004 rows=1 loops=1)
                      ->  CTE Scan on corner c  (cost=0.00..0.12 rows=6 width=16) (actual time=0.454..0.571 rows=2 loops=1)
  CTE features
    ->  CTE Scan on aabb  (cost=0.34..0.36 rows=1 width=100) (actual time=8.370..8.371 rows=1 loops=1)
          InitPlan 12 (returns $13)
            ->  Aggregate  (cost=0.14..0.15 rows=1 width=32) (actual time=4.150..4.150 rows=1 loops=1)
                  ->  CTE Scan on cardinal_change  (cost=0.00..0.12 rows=6 width=12) (actual time=3.963..4.062 rows=3 loops=1)
          InitPlan 13 (returns $14)
            ->  CTE Scan on start_grid  (cost=0.00..0.02 rows=1 width=4) (actual time=0.148..0.149 rows=1 loops=1)
          InitPlan 14 (returns $15)
            ->  CTE Scan on stop_grid  (cost=0.00..0.02 rows=1 width=4) (actual time=0.193..0.194 rows=1 loops=1)
          InitPlan 15 (returns $16)
            ->  Aggregate  (cost=0.14..0.15 rows=1 width=32) (actual time=0.952..0.953 rows=1 loops=1)
                  ->  CTE Scan on corner_grid  (cost=0.00..0.12 rows=6 width=12) (actual time=0.805..0.813 rows=2 loops=1)
  CTE character
    ->  Hash Join  (cost=22.22..40.22 rows=1 width=8) (actual time=0.560..1.346 rows=1 loops=1)
          Hash Cond: (l.candidate_characters = c_1.candidate_characters)
          Join Filter: (((l.start IS NULL) OR (f.start = l.start)) AND ((l.stop IS NULL) OR (f.stop = l.stop)) AND ((l.corners IS NULL) OR (f.corners = l.corners)) AND ((l.last_direction IS NULL) OR (f.directions[array_length(f.directions, 1)] = l.last_
          ->  Seq Scan on lookup_bestfit l  (cost=0.00..15.60 rows=560 width=116) (actual time=0.096..0.389 rows=353 loops=1)
          ->  Hash  (cost=22.17..22.17 rows=4 width=108) (actual time=0.270..0.270 rows=1 loops=1)
                Buckets: 1024  Batches: 1  Memory Usage: 9kB
                ->  Hash Join  (cost=0.03..22.17 rows=4 width=108) (actual time=0.209..0.230 rows=1 loops=1)
                      Hash Cond: (c_1.first_four_directions = f.directions[1:4])
                      ->  Seq Scan on lookup_candidates c_1  (cost=0.00..18.80 rows=880 width=64) (actual time=0.033..0.057 rows=70 loops=1)
                      ->  Hash  (cost=0.02..0.02 rows=1 width=76) (actual time=0.034..0.034 rows=1 loops=1)
                            Buckets: 1024  Batches: 1  Memory Usage: 9kB
                            ->  CTE Scan on features f  (cost=0.00..0.02 rows=1 width=76) (actual time=0.006..0.007 rows=1 loops=1)
  CTE debug
    ->  Hash Join  (cost=0.07..29.58 rows=4 width=172) (actual time=10.580..10.669 rows=1 loops=1)
          Hash Cond: (lookup_candidates.first_four_directions = features.directions[1:4])
          InitPlan 18 (returns $19)
            ->  CTE Scan on "character"  (cost=0.00..0.02 rows=1 width=8) (actual time=0.568..1.356 rows=1 loops=1)
          InitPlan 20 (returns $21)
            ->  CTE Scan on "character" character_1  (cost=0.00..0.02 rows=1 width=8) (actual time=0.001..0.001 rows=1 loops=1)
          ->  Seq Scan on lookup_candidates  (cost=0.00..18.80 rows=880 width=64) (actual time=0.022..0.046 rows=70 loops=1)
          ->  Hash  (cost=0.02..0.02 rows=1 width=100) (actual time=8.474..8.474 rows=1 loops=1)
                Buckets: 1024  Batches: 1  Memory Usage: 9kB
                ->  CTE Scan on features  (cost=0.00..0.02 rows=1 width=100) (actual time=8.380..8.382 rows=1 loops=1)
          SubPlan 19
            ->  Aggregate  (cost=1.75..1.76 rows=1 width=32) (actual time=0.278..0.278 rows=1 loops=1)
                  ->  Function Scan on unnest  (cost=0.00..1.00 rows=100 width=40) (actual time=0.103..0.104 rows=1 loops=1)
Planning time: 5.633 ms
Execution time: 12.077 ms
```

Handwriting Recognition with SQL
Noah Doersing