Calculating Spatial Distance Histograms For 3D Points

Luis Quezada
University of South Florida, Tampa
luis4@usf.edu

Mushfiq Mahmud
University of South Florida, Tampa
mushfiq@usf.edu

Tommy Truong
University of South Florida, Tampa
ttruong@usf.edu

***Abstract*—Complex aggregates are functions offered by database management systems to do rigorous calculations without redundancy in code. Database management systems such as PostgreSQL offers users the ability to implement and integrate additional complex aggregates. The project goal was to compute the spatial histogram in 3D space. We used 3D spatial distance histogram implementation and generated a quadtree index of the points. Result of the SDH query was stored into a table.**

I. Introduction

Aggregation is the formation of a number of things into a cluster such as adding all values of a column to find the sum. Database management systems (DBMS) such as PostgreSQL supports such complex aggregates to help the user perform complex analytical task with their SQL server. PostgreSQL currently supports various aggregates such as SUM, MAX, MIN, AVG, and COUNT. If a aggregate is not supported, PostgreSQL provides the ability for users to implement their own aggregate function in a compatible language such as C and integrate it into the codebase.

A. Problem Statement

A spatial distance histogram (SDH) is the histogram of distances between all pairs of particles in the system [2]. While the current codebase handles SDH queries for a set of 2D points, our task is to extend the complex aggregates in another language to handle a set of 3D points and integrate it into the codebase.

II. Methodology

In this section, we will discuss how to integrate with PostgreSQL, how quadtrees can be used for indexing and the algorithm behind SDH.

A. PostgreSQL Integration

PostgreSQL allows users to extend functionality by creating custom attributes to complement the database server such as custom types, aggregate functions and general purpose functions. Moreover, while the PostgreSQL source code is written in the C language, we have the freedom of using any external language that can be integrated into the source code including the procedural language provided by PostgreSQL themselves called `plpgsql`. We have made use of these to create external functions in both C as well as `plpgsql`.

At a high level, the custom source code is dynamically loaded via shared library object files which are OS specific. As long as the absolute path to these `.so` files are provided or they are available in the dynamic library path, these files can be referenced inside a database session in PostgreSQL using SQL functions. PostgreSQL provides some much needed abstractions in the form of C macros to aid in retrieving user data such as arguments to a function or data inside of SQL tables.

1) Custom Type: We start with creating a custom type called `Point3D` to store our 3 dimensional data. `Point3D` is defined as follows:

```
typedef struct Point3D
{
    double x;
    double y;
    double z;
} Point3D;
```

As can be seen from the interface, each coordinate is stored as a double numeric type. All other custom functions that we have are created on top of `Point3D`.

2) Custom Functions: We have a general purpose function named `distance3d` to calculate the euclidean distance between any 2 given `Point3D`s.

We also have the aggregate function SDH that populates the histogram and stores each bucket as its own row.

B. QuadTrees/Octrees

QuadTree is a tree like data structure which has exactly four children. It's used to store 2D points that can be traversed efficiently in logarithmic time [3]. QuadTrees are commonly used to index geographical data by partitioning two-dimensional space into four quadrants and recursively subdividing it until all points are mapped. Octrees are an extension of quadtrees, octrees have eight children and are often used to partition three-dimensional spaces and store 3D points, such as images. For our uses, since we are just calculating the distance of a spatial histogram, a quad tree is more than enough to store the data.

C. Spatial Distance Histogram Calculation

For our project, we went with the brute force method [4],[1] as shown in Algorithm 1.

```
Algorithm 1: Spatial Distance Histogram
Input: Set of 3D points
Output: Set of distances of 3D points between
        buckets in a histogram
arr = Set of 3D points
histogram = list of buckets
for $i \in numSamples$ do
    for $j \in numSamples$ do
        $x_1, y_1, z_1 = arr[i]$
        $x_2, y_2, z_2 = arr[j]$
        dist = euclidean distance between
          $(x_1, y_1, z_1)\ and\ (x_2, y_2, z_2)$
        $h_pos = \lfloor dist/bucketWidth \rfloor$
        histogram[$h_pos$].distCount++
    end for
end for
return histogram
```

III. Results

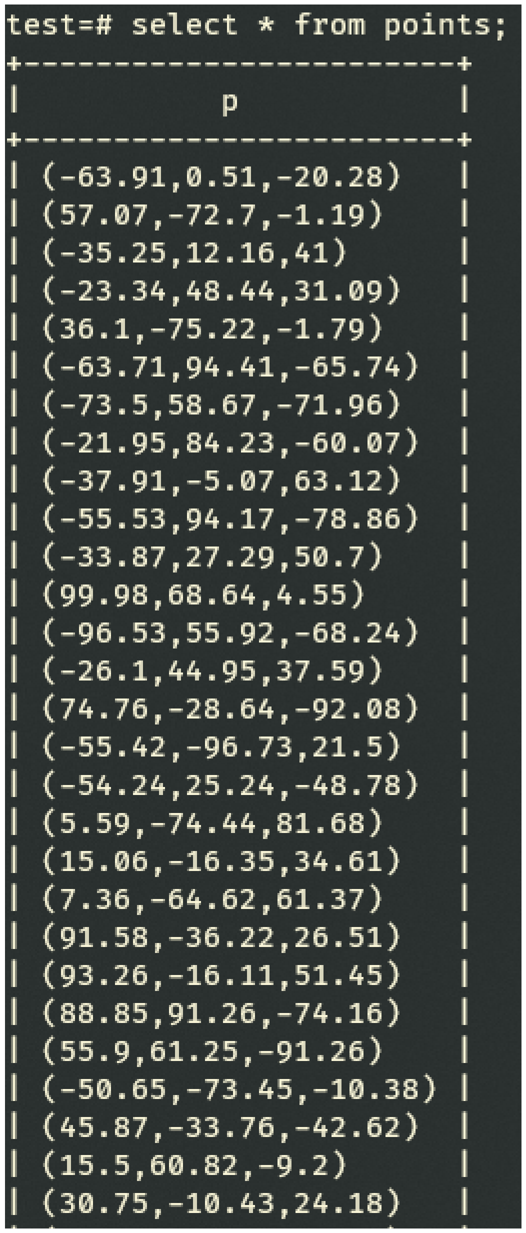
For the calculation of the SDH query, we were able to create a list of 3d points and stream them into the table such as shown on Fig. 1.

Once the 3d point table was populated, it was passed into the SDH custom aggregate that goes row by row to calculate the point-to-point distances and update the histogram distribution count as per Algorithm 1. The resulting SDH is shown on Fig. 2.

IV. Future Work

Since the scope of the project was small, we had to implement the ideas that were feasible while discarding the rest for a later time. Some of the future work that we could do is has the 3D point accept any datatype, on top that we would also like to be able to pass in a whole set of 3D points at once rather than one at a time. From the SDH standpoint, another future work is to implement the faster method [2] of SDH instead of the brute force method. The faster algorithm on top of doing a full scale octree instead of quadtree for indexing would speed up performance of the whole program.

While researching the SDH problem, we found an external library call PostGIS which makes solving the SDH problem easier for future integration. PostGIS is a spatial database extender for PostgreSQL object-relational database. It adds support for geographic objects allowing location queries to be run in SQL. What made PostGis useful is that it had an implementation of SDH that was more robust and adaptable to different scenario.

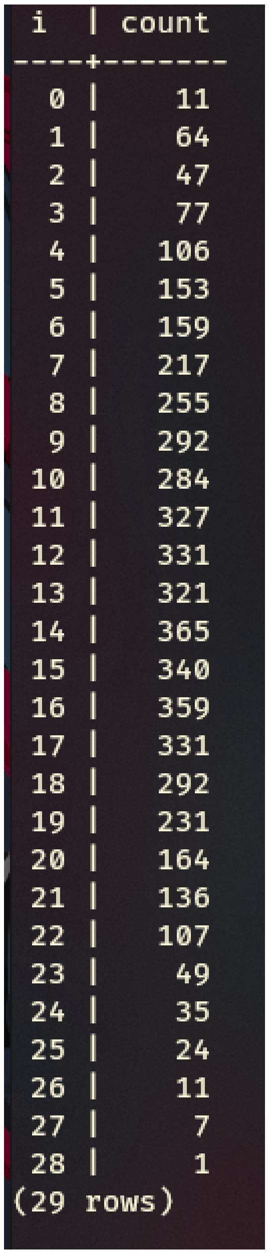


```
test=# select * from points;
+-----------------------+
|           p           |
+-----------------------+
| (-63.91,0.51,-20.28)  |
| (57.07,-72.7,-1.19)   |
| (-35.25,12.16,41)     |
| (-23.34,48.44,31.09)  |
| (36.1,-75.22,-1.79)   |
| (-63.71,94.41,-65.74) |
| (-73.5,58.67,-71.96)  |
| (-21.95,84.23,-60.07) |
| (-37.91,-5.07,63.12)  |
| (-55.53,94.17,-78.86) |
| (-33.87,27.29,50.7)   |
| (99.98,68.64,4.55)    |
| (-96.53,55.92,-68.24) |
| (-26.1,44.95,37.59)   |
| (74.76,-28.64,-92.08) |
| (-55.42,-96.73,21.5)  |
| (-54.24,25.24,-48.78) |
| (5.59,-74.44,81.68)   |
| (15.06,-16.35,34.61)  |
| (7.36,-64.62,61.37)   |
| (91.58,-36.22,26.51)  |
| (93.26,-16.11,51.45)  |
| (88.85,91.26,-74.16)  |
| (55.9,61.25,-91.26)   |
| (-50.65,-73.45,-10.38) |
| (45.87,-33.76,-42.62) |
| (15.5,60.82,-9.2)     |
| (30.75,-10.43,24.18)  |
```

Fig. 1. Table storing 3d points

V. Conclusion

Overall this project was nourishing in what it set for us to do and how to accomplish it. In the pursuit of implementing a complex aggregates, we learned the inner workings of PostgreSQL codebase. Since the codebase was highly modularized, it was simple to see how to plug in our complex aggregate implementation. We learned how to dynamically link a function with the source code and we learned a little bit about the complexity of indexing in a DBMS. Some of the problems however were mostly in project requirement. Since the project description wasn't very clear, there was some confusion as to what a spatial distance histogram was since we never learned it in class, how the SDH query gets tested,



```
 i  | count
----+-------
  0 |    11
  1 |    64
  2 |    47
  3 |    77
  4 |   106
  5 |   153
  6 |   159
  7 |   217
  8 |   255
  9 |   292
 10 |   284
 11 |   327
 12 |   331
 13 |   321
 14 |   365
 15 |   340
 16 |   359
 17 |   331
 18 |   292
 19 |   231
 20 |   164
 21 |   136
 22 |   107
 23 |    49
 24 |    35
 25 |    24
 26 |    11
 27 |     7
 28 |     1
(29 rows)
```

Fig. 2. Table showing the output of the SDH for 100 sample 3D points with a bucket width of 100

and what are the expected results. Another issue we had is that the description said there was an initial SDH codebase that handled 2D points but that wasn't provided beforehand nor was it in PostgreSQL codebase so we had to do some investigation to find it. However once those were cleared up, we had no issue doing the project.

References

[1] Michael Ankerst, Gabi Kastenmuller, Hans-Peter Kriegel, and Thomas Seidl. 3d shape histograms for similarity search and classification in spatial databases. July 1999.

[2] Anand Kumar, Vladmir Grupcev, Yongke Yuan, Jin Huang, Yi-Cheng Tu, and Gang Shen. Computing spatial distance histograms for large scientific datasets on-the-fly. October 2014.

[3] Chengceng Mou. A comparative study of dual-tree algorithms for computing spatial distance histogram, November 2015.

[4] Yi-Cheng Tu, Shaoping Chen, and Sagar Pandit. Computing spatial distance histograms efficiently in scientific databases. pages 796–807, November 2009.