Implementation of Complex Aggregates in PostgreSQL

Subramanian Viswanathan, Sneha Rajan, Raviteja Yerrapati
Computer Science Department
University of South Florida, Tampa

ABSTRACT:

Aggregate functions in PostgreSQL can simplify and speed up several applications by a huge extent. They can increase the accuracy and speed of the application. Supporting complex aggregates is a major functionality of modern DBMSs like PostgreSQL. The complex aggregates can be used in several use cases. In this paper we are computing the spatial histogram in the 3D space histogram. This serves as the complex aggregates in our research.

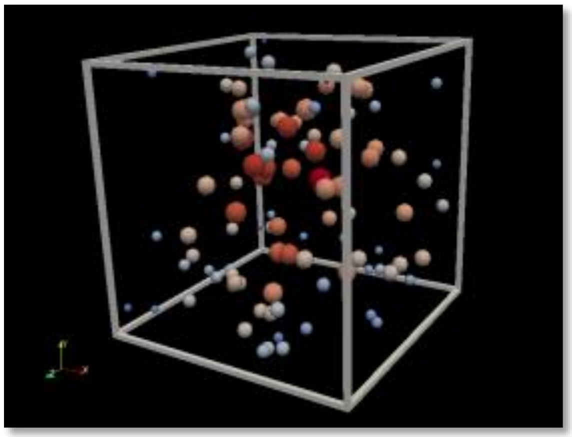


Fig 1: The figure shows a finite number particles from an infinite space interacting with each other.

This paper aims at implementing complex aggregates in PostgreSQL while computing the spatial distance histogram of a set of 3D points. Given the coordinates of the N number of 3d particles in a box, and a user-defined distance, we calculate the number of particle to particle distances falling into separate categories (buckets) based on the series of distance ranges between the particles. The expected result of the current work is to achieve a histogram consisting the number of particle-to-particle distances falling into a series of ranges of the bucket width.

1. INTRODUCTION

Complex Aggregates insist the SQL server to handle some kinds of calculations. In some complex applications, customized results can be achieved from multiple tables for analytical and reporting purposes. The quality of the results that are returned from the database can be greatly enhanced. We can avoid writing unnecessary lines of code for small, simple calculations or looping within the intermediate result set by using the aggregates in a useful manner. Creation of Complex aggregates can help us do the things faster in future. In case we need to compute a histogram for a different set of particles and bucket width, we can simply trigger the aggregate with the new arguments. As the existing aggregates like SUM, MAX, MIN, AVG and COUNT do their corresponding tasks for any input we give. Similarly, in this paper we show how to create an aggregate of our own which performs the same way as the remaining aggregates but in-return computes a spatial data histogram for the given input.

In this project we build a complex aggregate which runs a PostgreSQL function which is built on top of an executable object file containing the actual code. This object file can be generated in a language of our choice. We just need to compile the code in the environment in which the code is written and create a shared executable object file which can be used in PostgreSQL.

We have chosen to write a function in C language and use the C shared object file to build a PostgreSQL function on top of which a complex aggregate is built. The user simply gives the number of particles in the space and his own choice of bucket width. We define the 3D box size in the program and the 3D space is divided into some equal sized buckets based on the bucket width entered by the user. The expected output of the project is to generate a spatial distance histogram of the given set of 3D particles. We display 5 buckets in each row so as to get a feel of a histogram.

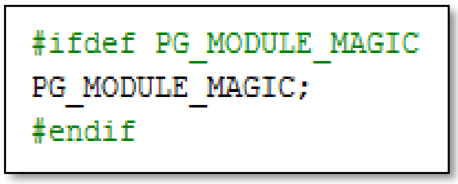
The major challenge in this project is to integrate a code written in C with PostgreSQL. Fetching arguments and returning the output is not the same as we do in a normal programming language. Care should be taken that the arguments are properly handled. The function should be compiled into dynamically loaded objects. Platform-specific shared library file name extension generally a '.so' file should be created which is executed when the aggregate is called. We will discuss about PostgreSQL supported C language functions in the further sections of the paper. The rest of the paper is organized as below. The methodology with various techniques is discussed in section 2. Sections 3 consists the results obtained after implementing complex aggregates. We derive the future work, conclusions, and references in section 4, section 5 and section6 respectively.

2. METHODOLOGY

In this section, we will discuss about all the techniques implemented to create a complex aggregate which on calling, computes a spatial data histogram for the user given arguments. As discussed earlier the major challenge is to select the C language functions which help in integrating it with the PostgreSQL. Let us now see how the complex aggregate is designed.

A. *DESIGN OF THE ALGORITHM WRITE IN C LANGUAGE:*

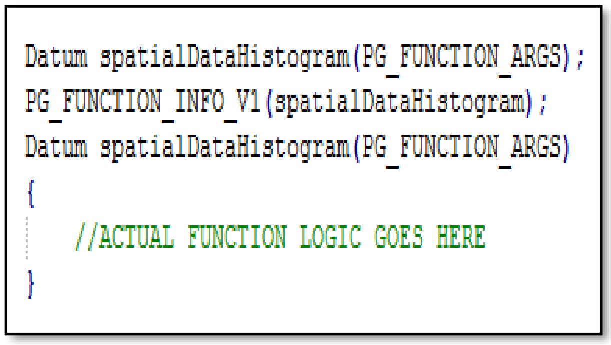
We have designed a user defined function in C named 'spatialDataHistogram.c' which is compiled into a dynamically loadable object and is loaded by the PostgreSQL server on demand. PostgreSQL will not compile a C function automatically. The object file must be compiled before it is referenced in a function. To ensure that a dynamically loaded object file is not loaded into an incompatible server, we add a magic block. This allows the server to detect obvious incompatibilities. The magic block is initialized in the C file as below:



```
#ifdef PG_MODULE_MAGIC
PG_MODULE_MAGIC;
#endif
```

Fig 2.1a: The above code snippet shows how to include a magic block.

'Postgres.h' should always be included first in any source file in a C file when we intend to build a function. It declares a number of things that you will need are discussed below. Initially, A macro is created. The calling convention relies on macros to suppress most of the complexity of passing arguments and results. The calling convention is indicated by writing a PG_FUNCTION_INFO_V1() macro call for the function. The macro call is done in the name of the source file. It is generally written just before the function itself. The actual function is now described and in our case the function name is spatialDataHistogram. PG_FUNCTION_ARGS accepts the runtime arguments given by the user and it should be given during the function declaration. The whole process goes as below:

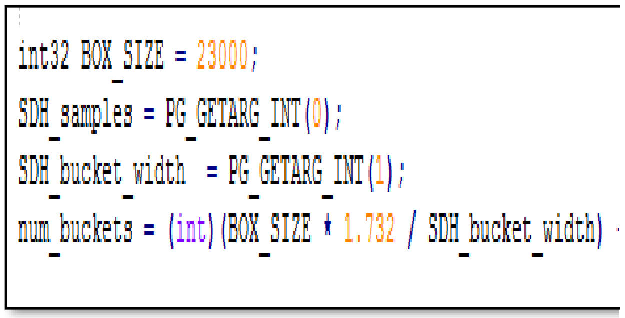


```
Datum spatialDataHistogram(PG_FUNCTION_ARGS);
PG_FUNCTION_INFO_V1(spatialDataHistogram);
Datum spatialDataHistogram(PG_FUNCTION_ARGS)
{
    //ACTUAL FUNCTION LOGIC GOES HERE
}
```

Fig 2.1b: The above code snippet shows how to include a magic block.

In the function, initially, the size of the 3D space considered is initialized. We can also make it a run time argument but as per this project, it has been defined as a simple variable in the function. The function accepts two arguments which are the number of samples and a user defined bucket width. PG_GETARG_XXX() macro is used to fetch the arguments. XXX can be replaced with the data type of the input arguments. For example: It can be set to INT if the input argument provided by the user is an integer and TEXT if the input argument provided by the user is text. It is a 3D Box. We just make sure if the arguments provided by the user are valid. Here, it is checked if both are proper positive integer values as below: (SDH_samples > 0 && SDH_bucket_width > 0).

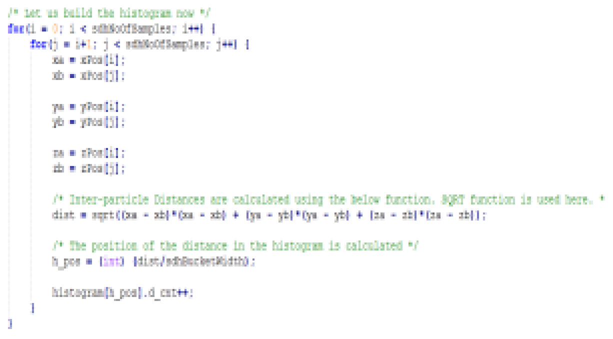
If the arguments are properly given, calculate number of buckets(equal sized) in the histogram based on the second argument (bucket width) given by user:



```
int32 BOX_SIZE = 23000;
SDH_samples = PG_GETARG_INT(0);
SDH_bucket_width  = PG_GETARG_INT(1);
num_buckets = (int)(BOX_SIZE * 1.732 / SDH_bucket_width) ·
```

Fig 2.1c: The above code snippet shows how the number of buckets in Histogram are set

Srand() and rand() function is used to assign the coordinates of the particles in the box. all the X, Y, Z coordinates of the 3D particles are assigned and stored in the corresponding arrays x_pos[], y_pos[], z_pos[].// Inter Particle Distances are calculated using the below function. SQRT function is used here and thus math.h is included in the code.



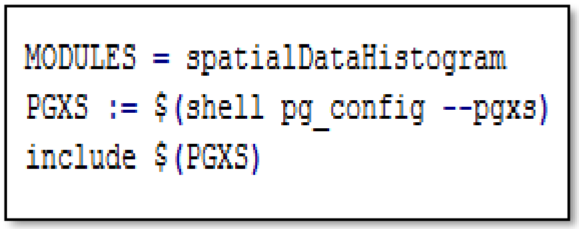
```
/* let us build the histogram now */
for(i = 0; i < sdhNoOfSamples; i++) {
    for(j = i+1; j < sdhNoOfSamples; j++) {
        xa = xPos[i];
        xb = xPos[j];

        ya = yPos[i];
        yb = yPos[j];

        za = zPos[i];
        zb = zPos[j];

        /* Inter-particle Distances are calculated using the below function. SQRT function is used here. *
        dist = sqrt((xa - xb)*(xa - xb) + (ya - yb)*(ya - yb) + (za - zb)*(za - zb));

        /* The position of the distance in the histogram is calculated */
        h_pos = (int) (dist/sdhBucketWidth);

        histogram[h_pos].d_cnt++;
    }
}
```

Fig 2.1d: The above code snippet shows the distance calculation using complex aggregates.

Now, all the inter particle distances are stored in the histogram and it needs to be returned when the aggregate is used. To accomplish this we need to move all the elements in the array into a single text field and simply return it. PG_RETURN_XXX is used to return a value which is the result of calling an aggregate. Again, XXX can be replaced with the datatype of the input arguments. For example: It can be set to INT if the input argument provided by the user is an integer and TEXT if the input argument provided by the user is text.

B. INTEGRATION OF C CODE INTO POSTGRES BY GENERATING AN EXECUTABLE SHARED OBJECT FILE:

As we already discussed, major challenge in this project is to integrate a code written in C with PostgreSQL. Fetching arguments and returning the output is not the same as we do in a normal programming language. Care should be taken that the arguments are properly handled. The function should be compiled into dynamically loaded objects. Platform-specific shared library file name extension generally a '.SO' file should be created which is executed when the aggregate is called. The .SO file is created as follows:

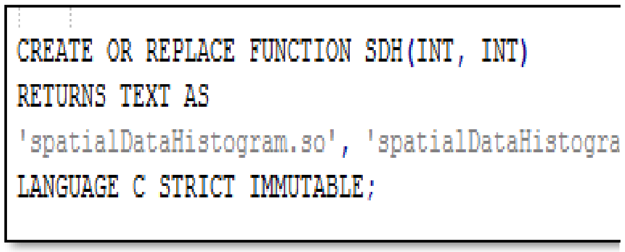


```
MODULES = spatialDataHistogram
PGXS := $(shell pg_config --pgxs)
include $(PGXS)
```

Fig 2.2: The above code snippet the generation of a .SO file with same name as the function in C file.

C. BUILD A FUNCTION IN POSTGRES ON TOP OF THE EXECUTABLE SHARED OBJECT FILE:

Now, the code is compiled and the executable shared object file spatialDataHistogram.so is ready. A Function in PostgreSQL can now be built on top of this object file. The function in the project is created in the following way. We can see that the input arguments which the shared object file expects are two integers and the output is a single text. The file is built in C and thus it needs to be mentioned in the function creation.

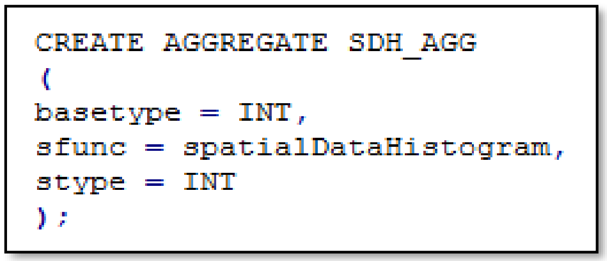


```
CREATE OR REPLACE FUNCTION SDH(INT, INT)
RETURNS TEXT AS
'spatialDataHistogram.so', 'spatialDataHistogra
LANGUAGE C STRICT IMMUTABLE;
```

Fig 2.3: The above code snippet shows the creation of a Function on the Object File

D. BUILD AN AGGREGATE IN POSTGRES ON TOP OF THE ABOVE FUNCTION:

Now, the Function is in place which returns the text value calculated by the C file. An aggregate needs to be built which calls the above function and generates a spatial Data Histogram. The Aggregate can be built as below:



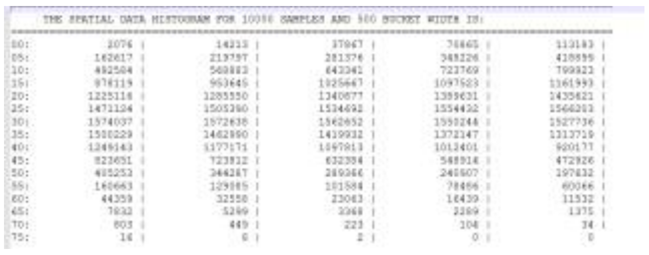
```
CREATE AGGREGATE SDH_AGG
(
basetype = INT,
sfunc = spatialDataHistogram,
stype = INT
);
```

Fig 2.4: The above code snippet shows the creation of the COMPLEX AGGREGATE which call the function built in Section 2.2

3. RESULTS

The function created accepts two arguments which are the number of samples and a user defined bucket width. Let us see the results returned by the complex aggregate for various input arguments passed. The expected output of the project is to generate a spatial distance histogram of the given set of 3D particles. We display 5 buckets in each row so as to get a feel of a histogram. The output is displayed in the PostgreSQL console and the output is returned in a way similar as that of any normal aggregate like SUM, MIN etc.

For instance, 10000 samples with a bucket width of 500 are given as the input arguments and the aggregate returns the following Histogram:

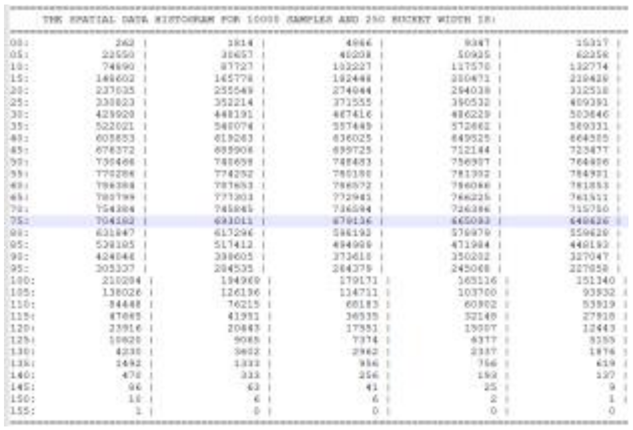


```
   THE SPATIAL DATA HISTOGRAM FOR 10000 SAMPLES AND 500 BUCKET WIDTH IS:
=========================================================================
00:        2076 |      14213 |      37967 |      76665 |     113183 |
05:      162617 |     213797 |     281376 |     348226 |     418899 |
10:      492504 |     568883 |     643341 |     723769 |     799823 |
15:      878119 |     953645 |    1025667 |    1097523 |    1161993 |
20:     1225118 |    1285550 |    1340677 |    1389631 |    1435621 |
25:     1471124 |    1505380 |    1534692 |    1554432 |    1566203 |
30:     1574037 |    1572638 |    1562652 |    1559244 |    1527736 |
35:     1500229 |    1462890 |    1419932 |    1372147 |    1313719 |
40:     1249143 |    1177171 |    1097813 |    1012401 |     920177 |
45:      823651 |     723812 |     632384 |     548914 |     472826 |
50:      405253 |     344287 |     289366 |     240507 |     197832 |
55:      160663 |     129985 |     101584 |      78486 |      60066 |
60:       44359 |      32558 |      23063 |      16439 |      11532 |
65:        7832 |       5289 |       3368 |       2289 |       1375 |
70:         803 |        449 |        223 |        104 |         34 |
75:          16 |          0 |          2 |          0 |          0
```

Fig 3.a: Result returned by the complex aggregate for 10000 points and a Bucket width of 500

In the above screenshot, we can see that the spatial data histogram with 80 buckets is generated. Each row contains five buckets. The bucket width is defined by the user. Total box size is defined in the program. Based on these two values the number of buckets is calculated to 80 in this case. Each bucket shows the number of particle-to-particle distances falling into a series of ranges of the bucket width. We will now see some more results for different values of the input arguments and see how the histogram looks like.

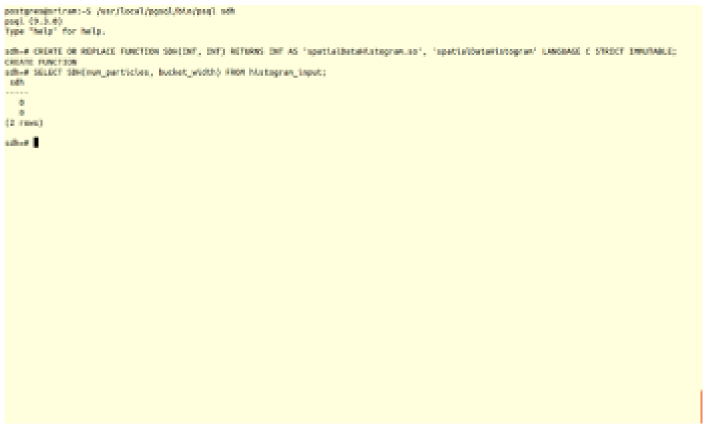
For instance, 10000 samples, bucket width of 250 are given as the input arguments. It is obvious that we get double buckets in this case which is 160 as the bucket width is reduced to half of that in the initial case. The aggregate returned the Histogram below:



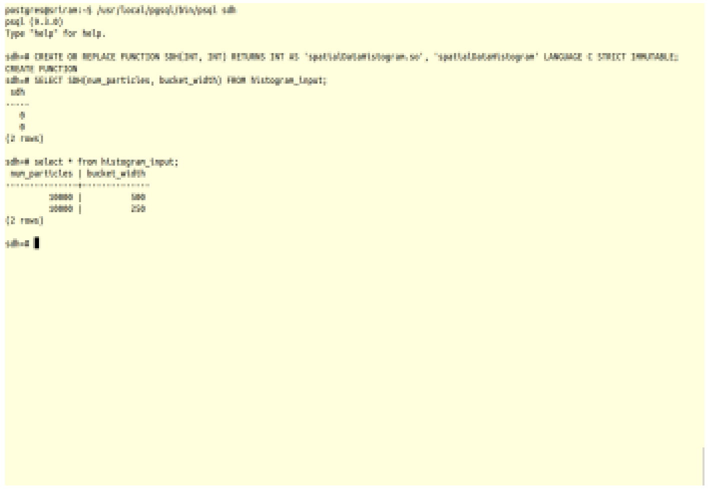
```
===========================================================================================
     THE SPATIAL DATA HISTOGRAM FOR 10000 SAMPLES AND 250 BUCKET WIDTH IS:
===========================================================================================
00:             262 |          1814 |          4866 |          9347 |         15317 |
05:           22550 |         30657 |         40208 |         50935 |         62256 |
10:           74690 |         87727 |        103227 |        117570 |        132774 |
15:          148602 |        165778 |        182448 |        200471 |        219429 |
20:          237035 |        255549 |        274044 |        294038 |        312518 |
25:          330823 |        352214 |        371555 |        390532 |        409391 |
30:          429920 |        448191 |        467416 |        486229 |        503846 |
35:          522021 |        540074 |        557449 |        572662 |        589331 |
40:          605853 |        619263 |        636025 |        649525 |        664585 |
45:          676372 |        685908 |        699725 |        712144 |        723477 |
50:          730468 |        741659 |        748483 |        756937 |        764486 |
55:          770286 |        774252 |        780188 |        781392 |        784931 |
60:          786388 |        787653 |        786672 |        786068 |        781853 |
65:          783799 |        777303 |        772981 |        766225 |        761511 |
70:          754384 |        745845 |        736594 |        726386 |        715756 |
75:          704182 |        693011 |        679136 |        665093 |        649626 |
80:          631847 |        617296 |        596192 |        579979 |        559629 |
85:          538185 |        517412 |        494989 |        471984 |        449193 |
90:          424046 |        398605 |        373618 |        350202 |        327047 |
95:          305337 |        284535 |        264378 |        245068 |        227058 |
100:          210284 |        194969 |        179171 |        165116 |        151340 |
105:          138026 |        126136 |        114711 |        103700 |         93932 |
110:           84448 |         76215 |         68183 |         60902 |         53919 |
115:           47065 |         41951 |         36535 |         32148 |         27918 |
120:           23916 |         20843 |         17551 |         15007 |         12443 |
125:           10620 |          9065 |          7374 |          6377 |          5155 |
130:            4230 |          3602 |          2962 |          2337 |          1876 |
135:            1492 |          1333 |           956 |           766 |           619 |
140:             470 |           333 |           256 |           193 |           137 |
145:              96 |            63 |            41 |            25 |             9 |
150:              16 |             6 |             6 |             2 |             1 |
155:               1 |             0 |             0 |             0 |             0
===========================================================================================
```

Fig 3.b: Result returned by the complex aggregate for 10000 points and a Bucket width of 250

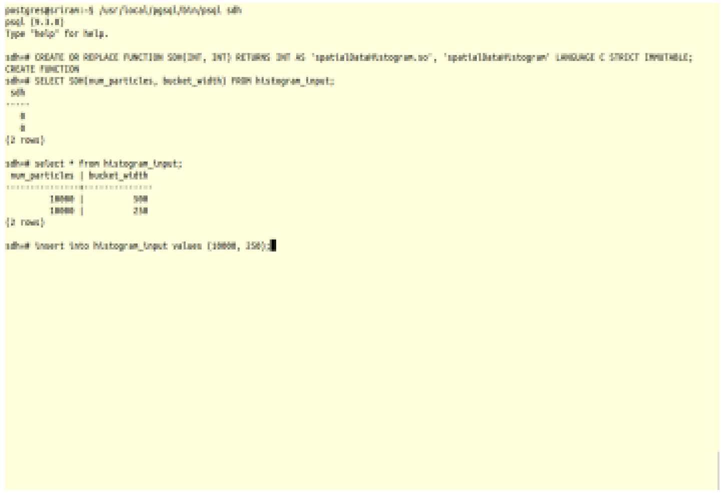
4. SCREENSHOTS



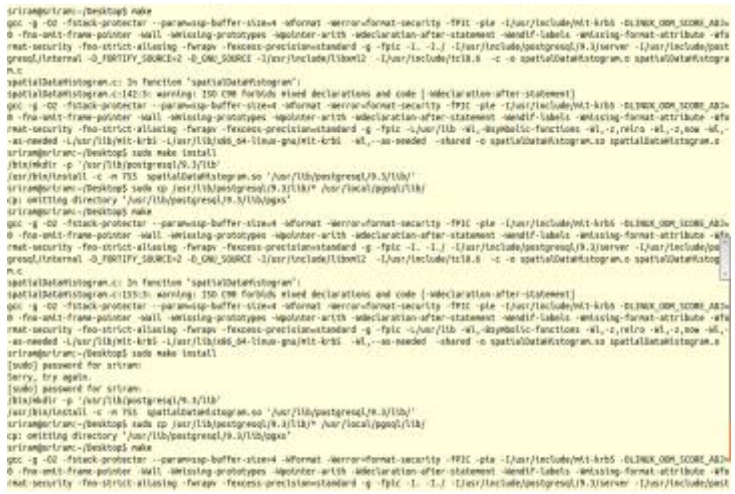
```
postgres@sriram:~$ /usr/local/pgsql/bin/psql sdh
psql (9.3.0)
Type "help" for help.

sdh=# CREATE OR REPLACE FUNCTION SDH(INT, INT) RETURNS INT AS 'spatialDataHistogram.so', 'spatialDataHistogram' LANGUAGE C STRICT IMMUTABLE;
CREATE FUNCTION
sdh=# SELECT SDH(num_particles, bucket_width) FROM histogram_input;
 sdh
-----
   0
   0
(2 rows)

sdh=# █
```



```
postgres@sriram:~$ /usr/local/pgsql/bin/psql sdh
psql (9.3.0)
Type "help" for help.

sdh=# CREATE OR REPLACE FUNCTION SDH(INT, INT) RETURNS INT AS 'spatialDataHistogram.so', 'spatialDataHistogram' LANGUAGE C STRICT IMMUTABLE;
CREATE FUNCTION
sdh=# SELECT SDH(num_particles, bucket_width) FROM histogram_input;
 sdh
-----
   0
   0
(2 rows)

sdh=# select * from histogram_input;
 num_particles | bucket_width
---------------+--------------
         10000 |          500
         10000 |          250
(2 rows)

sdh=# █
```



```
postgres@sriram:~$ /usr/local/pgsql/bin/psql sdh
psql (9.3.0)
Type "help" for help.

sdh=# CREATE OR REPLACE FUNCTION SDH(INT, INT) RETURNS INT AS 'spatialDataHistogram.so', 'spatialDataHistogram' LANGUAGE C STRICT IMMUTABLE;
CREATE FUNCTION
sdh=# SELECT SDH(num_particles, bucket_width) FROM histogram_input;
 sdh
-----
   0
   0
(2 rows)

sdh=# select * from histogram_input;
 num_particles | bucket_width
---------------+--------------
         10000 |          500
         10000 |          250
(2 rows)

sdh=# insert into histogram_input values (10000, 250);█
```



```
sriram@sriram:~/Desktop$ make
gcc -g -O2 -fstack-protector --param=ssp-buffer-size=4 -Wformat -Werror=format-security -fPIC -pie -I/usr/include/mit-krb5 -DLINUX_OOM_SCORE_ADJ=
0 -fno-omit-frame-pointer -Wall -Wmissing-prototypes -Wpointer-arith -Wdeclaration-after-statement -Wendif-labels -Wmissing-format-attribute -Wfo
rmat-security -fno-strict-aliasing -fwrapv -fexcess-precision=standard -g -fpic -I. -I./ -I/usr/include/postgresql/9.3/server -I/usr/include/post
gresql/internal -D_FORTIFY_SOURCE=2 -D_GNU_SOURCE -I/usr/include/libxml2  -I/usr/include/tcl8.6  -c -o spatialDataHistogram.o spatialDataHistogra
m.c
spatialDataHistogram.c: In function 'spatialDataHistogram':
spatialDataHistogram.c:142:5: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
gcc -g -O2 -fstack-protector --param=ssp-buffer-size=4 -Wformat -Werror=format-security -fPIC -pie -I/usr/include/mit-krb5 -DLINUX_OOM_SCORE_ADJ=
0 -fno-omit-frame-pointer -Wall -Wmissing-prototypes -Wpointer-arith -Wdeclaration-after-statement -Wendif-labels -Wmissing-format-attribute -Wfo
rmat-security -fno-strict-aliasing -fwrapv -fexcess-precision=standard -g -fpic -L/usr/lib -Wl,-Bsymbolic-functions -Wl,-z,relro -Wl,-z,now -Wl,-
-as-needed -L/usr/lib/mit-krb5 -L/usr/lib/x86_64-linux-gnu/mit-krb5  -Wl,--as-needed  -shared -o spatialDataHistogram.so spatialDataHistogram.o
sriram@sriram:~/Desktop$ sudo make install
/bin/mkdir -p '/usr/lib/postgresql/9.3/lib'
/usr/bin/install -c -m 755  spatialDataHistogram.so '/usr/lib/postgresql/9.3/lib/'
sriram@sriram:~/Desktop$ sudo cp /usr/lib/postgresql/9.3/lib/* /usr/local/pgsql/lib/
cp: omitting directory '/usr/lib/postgresql/9.3/lib/pgxs'
sriram@sriram:~/Desktop$ make
gcc -g -O2 -fstack-protector --param=ssp-buffer-size=4 -Wformat -Werror=format-security -fPIC -pie -I/usr/include/mit-krb5 -DLINUX_OOM_SCORE_ADJ=
0 -fno-omit-frame-pointer -Wall -Wmissing-prototypes -Wpointer-arith -Wdeclaration-after-statement -Wendif-labels -Wmissing-format-attribute -Wfo
rmat-security -fno-strict-aliasing -fwrapv -fexcess-precision=standard -g -fpic -I. -I./ -I/usr/include/postgresql/9.3/server -I/usr/include/pos
gresql/internal -D_FORTIFY_SOURCE=2 -D_GNU_SOURCE -I/usr/include/libxml2  -I/usr/include/tcl8.6  -c -o spatialDataHistogram.o spatialDataHistog
m.c
spatialDataHistogram.c: In function 'spatialDataHistogram':
spatialDataHistogram.c:135:5: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
gcc -g -O2 -fstack-protector --param=ssp-buffer-size=4 -Wformat -Werror=format-security -fPIC -pie -I/usr/include/mit-krb5 -DLINUX_OOM_SCORE_ADJ=
0 -fno-omit-frame-pointer -Wall -Wmissing-prototypes -Wpointer-arith -Wdeclaration-after-statement -Wendif-labels -Wmissing-format-attribute -Wfo
rmat-security -fno-strict-aliasing -fwrapv -fexcess-precision=standard -g -fpic -L/usr/lib -Wl,-Bsymbolic-functions -Wl,-z,relro -Wl,-z,now -Wl,-
-as-needed -L/usr/lib/mit-krb5 -L/usr/lib/x86_64-linux-gnu/mit-krb5  -Wl,--as-needed  -shared -o spatialDataHistogram.so spatialDataHistogram.o
sriram@sriram:~/Desktop$ sudo make install
[sudo] password for sriram:
Sorry, try again.
[sudo] password for sriram:
/bin/mkdir -p '/usr/lib/postgresql/9.3/lib'
/usr/bin/install -c -m 755  spatialDataHistogram.so '/usr/lib/postgresql/9.3/lib/'
sriram@sriram:~/Desktop$ sudo cp /usr/lib/postgresql/9.3/lib/* /usr/local/pgsql/lib/
cp: omitting directory '/usr/lib/postgresql/9.3/lib/pgxs'
sriram@sriram:~/Desktop$ make
gcc -g -O2 -fstack-protector --param=ssp-buffer-size=4 -Wformat -Werror=format-security -fPIC -pie -I/usr/include/mit-krb5 -DLINUX_OOM_SCORE_ADJ=
0 -fno-omit-frame-pointer -Wall -Wmissing-prototypes -Wpointer-arith -Wdeclaration-after-statement -Wendif-labels -Wmissing-format-attribute -Wfo
rmat-security -fno-strict-aliasing -fwrapv -fexcess-precision=standard -g -fpic -I. -I./ -I/usr/include/postgresql/9.3/server -I/usr/include/post
```

5. FUTURE WORK

In this paper we are computing spatial data histogram in the 3D space for given set of points. Up until the latest version of postgres for an aggregate function creation we can use only int, text, byte as the input only. In the case of only int datatype as the input it was challenging for us to create the aggregate function. In which case we have to create a function that works like an aggregate function. The histogram was printed on the console. As future work it would be beneficial to have the future releases of the Postgres to support all datatypes for the aggregate creation.

In our research we have written one function which takes int datatype as input and gives text as the output. In postgres when we create an aggregate we need to have two functions sfunc – transition function and ffunc – final function. In our case we are specifying input from a table and giving 3D histogram as output. Since we don't have different sfunc n ffunc it was challenging to execute the aggregate function.

6. CONCLUSION

In this paper we aimed at implementing complex aggregates in PostgreSQL while computing the spatial distance histogram of a set of 3D points. In this paper we have learnt how to use the postgres specific macros such as the PG_FUNCTION_ARGS, PG_FUNCTION_INFO_V1, PG_RETURN_NULL, PG_ARGISNUL that would help us run the C program on postgres. This gave us flexibility to explore the postgres sql codebase. Using these macros we were able to display 3D spatial histogram as a custom aggregate.

References:

[1] www.postgresql.org/docs/9.4/static/xaggr.html

[2] http://www.joeconway.com/plr/doc/plr-aggregate-funcs.html

[3] http://www.fi.muni.cz/usr/popelinsky/lectures /databaze/postgres/complex.sql