KTH Computer Science
and Communication

Visibility in PostGIS

Determining the visible buildings in a city environment using a spatial database.

KARL JOHAN ANDREASSON, CHRISTIAN WEMSTAD

Batchelor's Thesis at NADA
Supervisor: Michael Minock
Examiner: Mårten Björkman

Abstract

To be able to determine which building a pedestrian can see by the naked eye for any given coordinate is the core of this paper. The project is a part of the SpaceBook project, which caters to help tourists and urban workers in a city environment. Therefore it is important that the program has a tight correspondence to the real world.

Maps are exported from OpenStreetMap and imported into PostGIS, an extension to PostgreSQL. Functions are constructed in PostGIS to determine which buildings are visible from a given GPS-coordinate. Checking the center of the buildings as well as all the corners of said building does this. The program has inherited execution time constraints from the SpaceBook project. These are met for relatively small radius around the pedestrian (500 meter radius).

Optimization of the program is hard to accomplish due to the rather complex building placement in a city. There is no safe assumption of the city environment to be made that can be used to further accelerate the execution time of the program.

Referat

Synlighet i PostGIS

Att kunna avgöra vilka byggnader en fotgängare kan se för någon given GPS-koordinat är den centrala delen i denna rapport. Projektet är en del av SpaceBook-projektet vilket har som mål att hjälpa turister och stadsarbetare i en stadsmiljö. Det är därför viktigt att programmet har en tydlig verklighetsanknytning.

Kartor exporterades från OpenStreetMap och importerades in i PostGIS, ett tillägg till PostgreSQL. Funktioner konstruerades i PostGIS för att avgöra vilka byggnader som är synliga från en given koordinat. För att uppnå detta kontrolleras synligheten av centrum samt alla hörn på byggnaden. Det finns tidskrav på exekveringstiden på grund av SpaceBook-projektet. Dessa uppnås när en relativt liten radie runt positionen väljs (ca 500 meter).

Optimering av programmet är komplicerat på grund av den komplexa placeringen av byggnader i en stad. Därför är det svårt att hitta säkra antaganden som kan utnyttjas för att optimera programmet ytterligare.

Statement of collaboration

The work done for this project can be divided into three parts:

1. Possibility and background work
2. Developing of program and functions
3. Report writing

In the first part work was mostly done together. The second part was divided so that Karl Johan was more focused on the database functions while Christian mostly focused on developing the framework for testing the functions. In the report writing part the work was divided evenly. Karl Johan focused on results and discussions while Christian focused on the implementation. Though this was just for the first draft of the report, the other person proofread and extended the text when needed. Proofreading of the whole report was done together.

Contents

Chapter 1

Introduction

For a pedestrian to be able to navigate an unknown city, or part of a city, and get instant information about the buildings around her that are in sight. To help make this a reality was the purpose of this project.

This approach, giving feedback based on location, makes it possible to let tourists wander off and explore without the need of organized tours. This is unheard of for the broad masses; the only tourist market, which has been active in this regard, is in the museums where they provide a headset for the visitors with pre-recorded information. To scale this approach up for a whole city is a challenge.

To be able to determine the visibility for a specific point in a city in an effective manner, spatial requests are needed. To make this effective, an obvious approach was to do this on a database level using a spatial database. A spatial database is a database that is optimized for querying data regarding relations in space. This can be done with points as well as polygons in the space, polygons that could represent the outlines of buildings.

1.1 Overview

In Chapter 2 an explanation of the context of the project is presented along with relevant information of the situation as well as necessary information of the resources used in the project.

Chapter 3 focuses on what was done in the project and answers questions of the reader might have of what the actual approach to the problem at hand is and what is done to solve the problem.

The section 4.1 Implementation provides a more in depth view of how the program solves the problem was implemented. A thorough review of the code is presented with illustrations to give an increased understanding of the implementation.

The following section Results all the results are presented, this includes the raw data such as timings of the implementation with different settings activated.

In the section 4.3 Discussion all the data presented in Results are reviewed and discussed. For example what is the maximum distance a tourist is interested in a

building? Is this the same as an urban worker? Another aspect is the question whether there are any ways to improve our algorithm? Can one assume anything of building arrangement in a city environment?

In the last chapter a conclusion of the project is presented, this includes whether or not it was possible to achieve the SpaceBook project goals. The chapter also contains a summary of what has been done to achieve this conclusion. Possible source of errors regarding the implementation are also presented.

Chapter 2

Background

2.1 The SpaceBook Project

This project is a part of the SpaceBook project, which is a "speech-driven, hands-free, eyes-free device for pedestrian navigation and exploration". The main target group of the SpaceBook project is tourists and urban workers [2].

2.2 Tourism

Tourism has always been a huge market and easing the exploration of the city for the tourists has been and always will be relevant. In Stockholm alone the tourists stayed at sleeping accommodations over 10.5 million times during 2011. This is an increase of 5% compared to 2010. This ascending trend is not exclusive to Stockholm. For all of Sweden the guest nights increased with 1% [3].

Sightseeing buses and other travelling services with a guide describing the surroundings to the tourists is a common way of exploring the city. Designing the tour themselves and executing it in their own pace makes the tour more interesting for the tourists. This is because they are able to focus on what they think is the more interesting parts of the tour. They will also not have to follow the time schedule for when the tour buses and making the starting times of sightseeing. This is done by letting them wander off with a device that takes the tourists GPS position and determines the buildings visible to her with additional information about the buildings.

2.3 Computer Games

Similar challenges are present in computer games. Determining the visibility in a game is commonly done for person playing a game in a three dimensional simulation of the game environment. The way most computer games tackle this issue is to load in the whole environment and then model the view of the player based on that.

This is not possible for this project implementation because the environment the program is to model is significantly larger than the counterpart in computer games.

Also computer games have a quite long loading phase of the data that is to be used in the program, this is not desirable in the program because of the way it is supposed to be used on the move. If the program takes too long to load the tourist might be discouraged to use the whole final product.

2.4 Resources

To store the data about the buildings and surroundings in the city, a database is necessary. The database of choice in this project is PostgreSQL. PostgreSQL is released under MIT-License and thus is a free and open source database. PostgreSQL is an object-relational database management system that uses the Structured Query Language and implements most of the SQL:2008 standard, which is the ISO standard [4].

Just PostgreSQL alone is not enough to complete the questions to the database, also called queries. To make the database able to answer the required queries an extension to PostgreSQL called PostGIS is needed. PostGIS enables the database to act as a spatial database and thus enabling spatial queries. A spatial database is designed to store and query data of their relations with other objects in space. A spatial query is a query of relations between objects in space [1].

OpenStreetMap is an open source map of the whole world that makes it free to use and edit as you please, this also implies that the data entered to OpenStreetMap are from volunteers. The license used by OpenStreetMap is Creative Commons Attribution-ShareAlike 2.0 [5]. To convert the data extracted from OpenStreetMap to SQL queries an open source Java program was used. This conversion was done outside of this project.

Chapter 3

Approach

This project was created to see if there was a possibility to create a program determining the visibility in a city. Due to the mobile nature of the SpaceBook project there are time constraints that needed to be taken into consideration.

The time constraints promoted an approach in a low level nature. A natural calculations level for this was therefore on a database level. To determine the possibility of the project and efficiently implement functions a spatial database was needed due to the spatial requests. The spatial database of choice in this project was PostGIS, an extension to PostgreSQL. When determining the possibility of the project, the approach was to explore the library functions available in both PostGIS and PostgreSQL.

Being able to make accurate decisions of visibility in a city environment the data used was based on data extracted from OpenStreetMap using an open source Java program. To avoid total dependence on the data given by the extraction program, a set of results will be double-checked in the actual place by visual determination.

The Java-program used to extract data from OpenStreetMap exports had no functional way of determining the height of a building. This was because the map in OpenStreetMap does not always have a height value. Due to this every building was considered to be of infinite height.

How far can one see in a city environment and after what distance are the buildings not relevant to the observer any more. These questions were rather specific to the city and the approach was to let the end user decide at what distance buildings should be evaluated. Now a user or an application using this implementation could decide which distance is reasonable of how far a tourist or an urban worker can see and still find the information useful.

A function that determines the visibility of a building from a position in the city was central in this project. The definition of visibility used in this project was that if one point of the building was in sight, the building was deemed visible. Being able to determine the visibility of a building the central point of the building along with the corners should be checked. If one of these points was visible the building was deemed visible from that position. To be able to determine which buildings

are visible around the coordinate the distance to every building is calculated. The buildings that are within the given distance are then evaluated for visibility.

The testing and benchmarking of the program was done on a laptop with as few other processes running as possible. The mean value of several measurements was used to further strengthen the results.

Chapter 4

Analysis

4.1 Implementation

The first line of action when trying to solve this problem was to write a function to determine if the center of a building is visible. The central point of the building is never in sight from the position of the pedestrian. However, if no buildings intersects a line between the center and the position of the pedestrian, one part of the building must be in sight. This is the part of the wall that blocks the center of building from being in sight.

The center of the building was located and the buildings that intersects with a line between the center and the point where the pedestrian was standing were identified (Seen as the red point in the figure 4.1). Accessing the center of the building was achieved by using the built-in PostGIS function called **ST_Centroid(geometry)**. This function takes a standard PostGIS geometry and locates the center point in the geometry (Seen as the blue point in the figure 4.1). From this point a line was created using **ST_MakeLine(geometry, geometry)**. This function returns a PostGIS LineString from the two PostGIS geometry parameters (Seen as the black line in the figure 4.1).

Figure 4.1. Visibility of the center point of building V.

To find all the intersections on the line between the building center and the pedestrian, the function **ST_Intersects(geometry, geometry)** was used. The intersects function takes the LineString as the first parameter and a building as the second parameter and returns true if the building intersects with the line. The intersects function is looped through all the buildings in the database, except the building the pedestrian is looking at, to find possible intersects (An intersection is shown in figure 4.2).

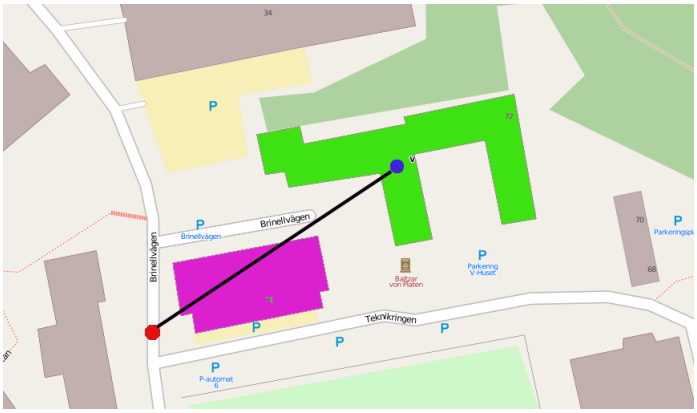

Figure 4.2. A building is intersecting the visibility of the building V.

Since the visibility of a building is not only determined if the center of a building is visible or not; an extension to the first algorithm was made. In this extension a function for checking every corner of the desired building was included, to get a more precise determination of visibility. Using the same reasoning as for the center of the building, a line was defined between the pedestrian and the corner. If no other buildings intersects this line, the corner was deemed visible since either the corner is in sight or the part of the wall that blocks the corner is.

To achieve this, the program first needed to find out how many corners the desired building had. Using the buildings polygon as the parameter to the PostGIS function **ST_ExteriorRing(geometry)** an outline of the polygon was returned as a LineString. This LineString is used in the PostGIS function **ST_NumPoints(geometry)** and the function will then return the number of corners of the building. When the program has the number of corners it will loop a new query for every corner, containing the function **ST_PointN(geometry, int)** with the current corner as a parameter. The PointN function will then return the point of the current corner. Using this point and the pedestrian point with the functions **ST_MakeLine(geometry, geometry)** and **ST_Intersects(geometry, geometry)** in the same way as for the center point will return whether each corner is visible (As illustrated in the figure 4.3).

This solution did not consider the fact that different buildings have different height since we in the beginning of the project did not have access to a database containing the height of the buildings, meaning that every building is blocking the

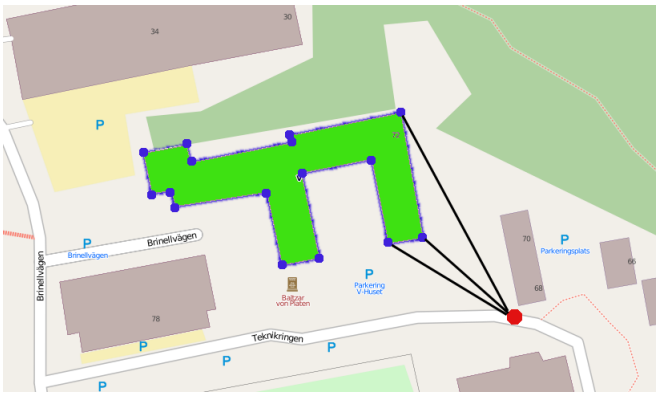

Figure 4.3. Visibility of the corner points of building V.

way even if it in real life is just a foot tall.

The next stage in the implementation was to be able to list all buildings that are visible from a specific location. This was achieved on the database level, using the database scripting language PL/pgSQL[6] inside the functions. A function was created that takes the input parameters distance and a position and returns the identification number of each building that are visible from that position. The function extracts a circular area around the position with the distance as radius. This area was then used to create a temporary view in the database. Further, in this temporary view the database then logs all the buildings, loops through them and checks if they are visible, with the same visibility check as uses before. Finally returning the result.

This version of the code worked as planned but it was not very time efficient. Locating the most time consuming factor in the code was done to improve the execution time. The most time consuming factor was located to be the **ST_Intersects(geometry, geometry)**, this is because it is heavy for the computer to calculate and because this function was called multiple times. Since the function is a standard function in PostGIS it was not possible to change it to make it faster, without rewriting the whole function. But it was possible to make the implementation execute faster by reducing the amount of input data to the function. It was also possible to reduce the number of times the function was called making the whole program go faster. These two enhancement where accomplished by introducing an extra table called blocked and an extra view. This table blocked contains all the buildings that are blocked by other buildings and the view all buildings inside the distance circle that are not in table blocked, therefore not blocked or not yet checked. If a building (Blue building in figure 4.4) was blocked it means that all its corners and center is being blocked by other buildings (Red buildings in figure 4.4). When checking the next building (Green building in figure 4.4) there is then no need to check if the green building is being blocked by the blue building since then the red buildings will also block the green building. By adding the blue blocked building

to the table blocked and only check new buildings against the view not containing blocked buildings the number of calls to **ST_Intersects(geometry, geometry)** is significantly reduced (Code for this in appendix A.2 function **allVisibleBuildings(numeric, numeric, numeric)**).

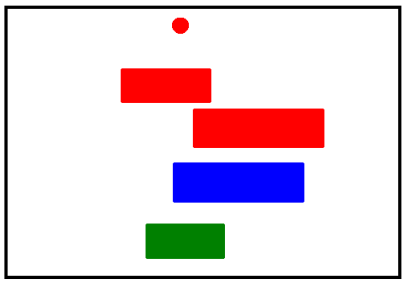

Figure 4.4. The green building is being blocked the red buildings. Checking the blue building is unnecessary.

Another optimization made in the function checking visibility on a single building. By changing the function to return true as soon as a part of the building was declared visible it was possible to reduce the numbers of calls to **ST_Intersects(geometry, geometry)** (This can be seen in appendix A.2 function **isVisi(numeric, numeric, numeric)**).

4.2 Results

The data returned from the implementation was deemed correct from observations in the real world. The comparisons were made at different locations in the campus of KTH.

The timings was done on a computer using i7-2630QM CPU @ 2.00GHz $*$ 8 and running 32-bit Ubuntu 11.10. PostgreSQL version was 9.1.3. Throughout this section the coordinate used to represent the person in the city will be consistent, namely $18.07247°, 59.3457°$. This point is shown in figure 4.5 together with the houses checked.

Average runtime while checking center and all the 24 corners of the building E was 69 milliseconds. Mean value of runtime while checking the center point and the 4 corners of the building *Kungliga Tekniska Högskolan* was 28 milliseconds. Average runtime while checking visibility of all buildings around the point in a 100 meter radius was 69 milliseconds.

Checking every building in a 500 meter radius the program executed in average in 3300 milliseconds. Determining visibility of every building in a 1000 meter radius has an average runtime of 12000 milliseconds.

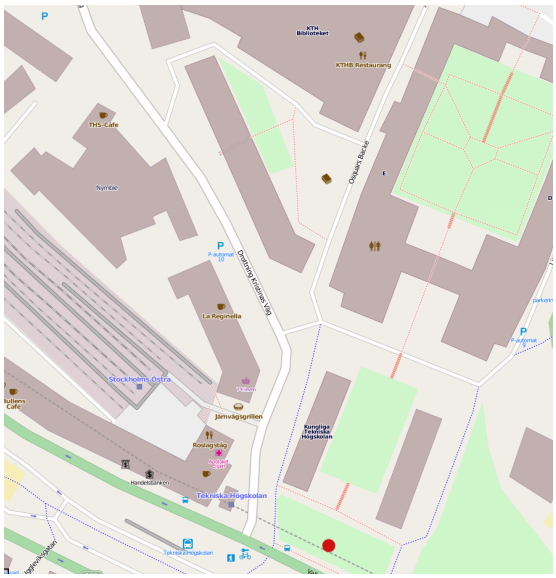

Figure 4.5. Settings for the results.

4.3 Discussion

The results in section 4.2 should be taken with moderation and are not absolute truths, mainly because the testing was used on computers running several background processes. These processes were impossible to manage and could affect the timings.

All the results of which buildings were visible presented in section 4.2 corresponds to our own measurements in the actual environment. The absolutely largest problem with the data presented was the fact that the program does not take into consideration that there might be ground level differences present in the city. Typically, hills cause visibility to increase or decrease in a city. Also the program considers every building to be of infinite height. This premise to the project causes the program to think that a house as small as a tool shed might block the visibility of the Stockholm Palace. This would of course not be the case in the real world. The problem with height of buildings and ground level is caused due to the fact that height of building and ground level is not present on every building in the maps on OpenStreetMap.

Another important aspect of our project was the catering to urban workers and tourists. Therefore it was important to consider whether or not a tourist is able to see a building from 1000 meters, or even further, away. This is generally not the case in the central parts of a city except for big landmarks such as the Eiffel tower. Though in regards to tall buildings the implementation is more restricted by the fact that no height values are existent in the calculations.

Maybe there is a optimal distance to the buildings which are to be checked but

this number is highly likely to differ from city to city and therefore we implemented the function, which finds the visible buildings (**allVisibleBuildings(longitude, latitude, distance)**), in such a manner that the maximum distance is a parameter to the function. In this way the user or application using the implementation can determine how distant buildings they want to be able to get information about.

Another way to optimize the execution time might be to be less strict of what makes a building visible. Either not checking the center, or not checking every corner can do this. Both solutions limit the amount of calls to **ST_Intersects(geometry, geometry)** that is the bottleneck in the current implementation.

Cities does not follow any kind of strict architectural design standard. Because of this making any kind of assumption about the visibility in a city is a hard task to do correctly. There is no way of to determine where buildings are located and in what shape they have. There are patterns, as in blocks, but no magic rule that can be used in an implementation. This limits the amount of possible optimizations drastically.

Chapter 5

Conclusion

To determine what is visible for a pedestrian in a city environment is possible using a spatial database. The approach to define functions in the database limits the overhead and increases speed compared to a implementation on application level.

The execution times are not unreasonable for an application connecting to a server, especially if a smaller radius of the area searched is used. The SpaceBook project set out to implement a program to ease the day of urban workers and tourist. This project is a step in right direction to accomplish this task.

One optimization implemented is using the fact that if a building is covered, i.e. not visible to the pedestrian, this building does not need to be checked if it blocks another building. Optimizing the program for faster execution, the amount of calls to the function **ST_Intersects(geometry, geometry)** has to be reduced even further. Making an assumption of the city layout and using it to eliminate buildings that could not possibly be visible from the current position can do this. Another approach is to change the requirements to make a building visible.

Sources of errors in the results from the program are most likely to come from the absence of checking height or ground level in the implementation. The timings of the program will differ with computer set-up and the amount of background processes running when executing this implementation. This can of course be eliminated completely by running it on a dedicated server, which should be the case in the SpaceBook project.

Bibliography

[1] *What is postGIS?* retrieved Feb. 9 2012. <http://postgis.refractions.net/>

[2] *SpaceBook* retrieved Apr. 1 2012. <http://spacebook-project.eu/>

[3] *Turismen i Stockholm* retrieved Apr. 1 2012. <http://www.stockholmtown.com/templates/page____19505.aspx>

[4] *PostgreSQL:About* retrieved Apr. 1 2012. <http://www.postgresql.org/about/>

[5] *OpenStreetMap:Copyright and License* retrieved Apr. 1 2012. <http://www.openstreetmap.org/copyright/en/>

[6] *PL/pgSQL - SQL Procedural Language* retrieved Apr. 10 2012. <http://www.postgresql.org/docs/8.3/static/plpgsql.html/>

Appendix A

Program

A.1 Example Output

Checking visibility of the building named *E*, the first boolean value corresponds to whether or not the center of the building is visible, and the rest 24 boolean values corresponds to the 24 corners of the building:

```
mindb=# select isVisible(18.07247,59.3457,'E');
 isvisible
-----------
 t
 f
 f
 f
 f
 f
 f
 t
 t
 t
 t
 t
 t
 t
 f
 f
 t
 t
 t
 t
 t
 t
 t
 f
```

```
 f
 t
 t
(25 rows)

Time: 70,654 ms
```

Whilst checking the building named *Kungliga Tekniska Högskolan* the output from the program might look like this, in this case there are only four corners of the building and therefore the first boolean is the center and the following four are the corners: Example output from program:

```
mindb=# select isVisible(18.07247,59.3457,'Kungliga Tekniska Högskolan');
 isvisible
-----------
 t
 t
 t
 t
 t
(5 rows)

Time: 27,319 ms
```

Checking all the possible buildings in a 100 meter radius from the coordinate representing the pedestrian the output consists of the id of the buildings visible. Example output looks like this:

```
mindb=# select allVisibleBuildings(18.07247,59.3457,100);
 allvisiblebuildings
---------------------
                  61
                 386
                 285
                 241
                  51
                 324
                 314
                 153
(8 rows)

Time: 65,615 ms
```

A.2 Code

```
CREATE OR REPLACE FUNCTION isVisible(numeric, numeric,
    numeric) RETURNS TABLE( visible boolean) as $$
DECLARE
distanceToGoal numeric;
--idOfBuilding integer;
numberOfCorners integer;
tmpint integer;
loopint integer;
longitude ALIAS FOR $1;
latitude ALIAS FOR $2;
idOfBuilding ALIAS FOR $3;
BEGIN
-- Determines the distance to the building
SELECT INTO distanceToGoal st_maxdistance(transform(
    st_setsrid(st_point(longitude,latitude),4326),3006),geom)
     FROM haspolygon WHERE id = idOfBuilding;
-- Creates tmpview which contains the buildings which should
     be checked if the block
EXECUTE 'CREATE OR REPLACE TEMP VIEW tmpview AS SELECT *
    FROM haspolygon WHERE st_distance(transform(st_setsrid(
    st_point('||longitude||','||latitude||'),4326),3006),
    geom) <= ' || distanceToGoal || ' AND id != ' ||
    idOfBuilding;
-- Checking middle.
SELECT INTO tmpint count(distinct(st_intersects(st_makeline
    ((SELECT st_centroid(geom) FROM haspolygon WHERE id =
    idOfBuilding), transform(st_setsrid(st_point(longitude,
    latitude),4326),3006)),geom))) FROM tmpview;
IF tmpint < 2 THEN -- If it was one it was visible.
        visible = true;
        return next;
ELSE
        visible = false;
        return next;
END IF;
-- Checking number of corners in the building.
SELECT INTO numberOfCorners st_numpoints(st_ExteriorRing(
    geom)) FROM haspolygon WHERE id = idOfBuilding;
-- Loops through the corners.
loopint := 1;
```

```
WHILE loopint < numberOfCorners LOOP
        SELECT INTO tmpint count(distinct(st_intersects(
            st_makeline((select pointn(st_ExteriorRing(geom),
             loopint) FROM haspolygon WHERE id = idOfBuilding
            ), transform(st_setsrid(st_point(longitude,
            latitude),4326),3006)),geom))) FROM tmpview;
        IF tmpint < 2 THEN -- If it was one it was visible.
                visible = true;
                return next;
        ELSE
                visible = false;
                return next;
        END IF;
        loopint := loopint + 1;
END LOOP;
return;
END;
$$ LANGUAGE 'plpgsql';


CREATE OR REPLACE FUNCTION allVisibleBuildings(numeric,
    numeric, numeric) RETURNS SETOF integer AS $$
DECLARE
longitude ALIAS FOR $1;
latitude ALIAS FOR $2;
distance ALIAS FOR $3;
bvar boolean;
count integer;
n integer;
BEGIN
-- Creates the two temporary views and the table needed.
EXECUTE 'CREATE OR REPLACE TEMP VIEW tmp1 AS SELECT
    st_distance(transform(st_setsrid(st_point('||longitude||'
    ,'||latitude||'),4326),3006), geom) AS dist, id, geom
    FROM haspolygon WHERE st_distance(transform(st_setsrid(
    st_point('||longitude||','||latitude||'),4326),3006),
    geom) <' || distance || ' ORDER BY dist, id';
EXECUTE 'CREATE TEMPORARY TABLE blocked (id integer)';
EXECUTE 'CREATE OR REPLACE TEMP VIEW tmp2 AS SELECT * FROM
    tmp1 WHERE id NOT IN (SELECT id FROM blocked)';

-- Loop through every possible building which can block the
    visibility.
FOR n IN SELECT id FROM tmp1 LOOP
```

```
        SELECT INTO bvar isVisi(longitude,latitude,n);
        IF bvar = true THEN
                return next n;
        ELSE
                EXECUTE 'INSERT INTO blocked VALUES (' || n
                    || ')';
        END IF;

END LOOP;

-- Drop the temporary views and the table used in the
    function.
EXECUTE 'DROP VIEW tmp3';
EXECUTE 'DROP VIEW tmp2';
EXECUTE 'DROP VIEW tmp1';
EXECUTE 'DROP TABLE blocked';
return;
END;
$$ LANGUAGE 'plpgsql';


CREATE OR REPLACE FUNCTION isVisi(numeric,numeric,numeric)
    RETURNS boolean AS $$
DECLARE
longitude ALIAS FOR $1;
latitude ALIAS FOR $2;
idOfBuilding ALIAS FOR $3;
count integer;
numberOfCorners integer;
loopint integer;
distanceToGoal numeric;
BEGIN

-- Calculate maximum distance to building
SELECT INTO distanceToGoal st_maxdistance(transform(
    st_setsrid(st_point(longitude,latitude),4326),3006),geom)
    FROM tmp2 WHERE id = idOfBuilding;

-- Use this distance to eliminate buildings which cannot
    block the view of the building.
EXECUTE 'CREATE OR REPLACE TEMP VIEW tmp3 AS SELECT * FROM
    tmp2 WHERE st_distance(transform(st_setsrid(st_point('||
    longitude||','||latitude||'),4326),3006), geom) <= ' ||
    distanceToGoal;
```

```
-- Checks visibility of center of building.
SELECT INTO count count(distinct(st_intersects(st_makeline((
    SELECT st_centroid(geom) FROM tmp3 WHERE id =
    idOfBuilding), transform(st_setsrid(st_point(longitude,
    latitude),4326),3006)),geom))) FROM tmp3 WHERE id !=
    idOfBuilding;
IF count < 2 THEN -- If it was one it was visible.
        return true; -- Return true immediately.
END IF;

-- Number of corners in the building.
SELECT INTO numberOfCorners st_numpoints(st_ExteriorRing(
    geom)) FROM tmp3 WHERE id = idOfBuilding;

-- Loops through the corners and checks visibility for each
    - if one corner is found visible the function returns
    true.
loopint := 1;
WHILE loopint < numberOfCorners LOOP
        SELECT INTO count count(distinct(st_intersects(
            st_makeline((SELECT pointn(st_ExteriorRing(geom),
             loopint) FROM tmp3 WHERE id = idOfBuilding),
            transform(st_setsrid(st_point(longitude,latitude)
            ,4326),3006)),geom))) FROM tmp3 WHERE id !=
            idOfBuilding;
        IF count < 2 THEN -- If it was one it was visible.
                return true; -- Return true immediately.
        END IF;
        loopint := loopint + 1;
END LOOP;
return false;
END;
$$ LANGUAGE 'plpgsql';
```