VoxelGrid 3D

Inputs:

Curve (represents the lot curve)

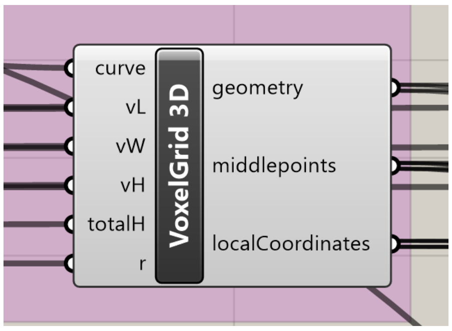
vL (the requested voxel length)

vW (the requested voxel width)

vH (the requested voxel height)

totalH (the requested total height of the pointcloud)

r (the requested rotator)



```
If:

 one of the dimension inputs equals 0 return warning message

Else:

Create 3 empty lists for the point coordinates in the world space, the point coordinates in the local space and for the voxel geometry.

Create a rotating plane that rotates based on the rotator input.

Create a bounding box around de lot curve based on the rotating plane

Create a  orientating surface using the rotator based on the corner points of the rotating plane

Create a surface from the lotcurve and translate it to -2 and extrude it by 4 to create an evaluation box for the first floor of the lot.

Create counters for the grid creation by dividing the requested input length and width by the length and width of the orientating surface.

Create the counter of the height by dividing the input height by the requested input total height.

For every length counter:

        For every width counter:

                create a baseplane point on the rotating baseplane based on the counters times the input dimensions

                    if this point is inside of the evaluation box

                        For every height counter:

                                create a point based on the baseplane point and the height counter times the input dimensions

                                create a tuple for the localcoordinates based on the counters

                                add the point to the point coordinates in world space list
                                add the tuple to the point coordinates in the local space list

foreach point in the pointlist in world space
        a rotating plane based on the rotator input with the point as centrepoint

        create a x,y & z interval based on the input variables

        create the voxel on the rotating plane based on the x,y & z intervals

        add the voxels to the voxellist
```

---

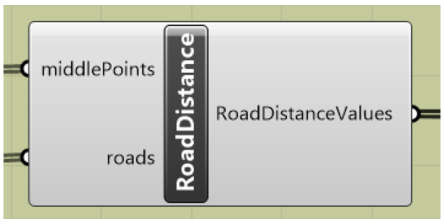
outputs:

geometry (the created voxels)

middlepoints (the created pointcloud based on the world coordinates)

localCoordinates (the created list of tuples with localcoordinates)

RoadDistance



Inputs:

MiddlePoints (from the Voxelgrid 3D node)

Roads (curves representing the roads in the model)

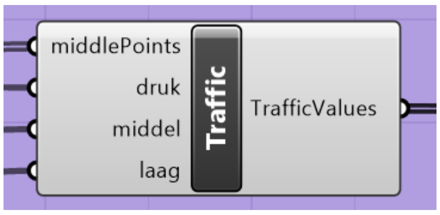
```
Create new list the size of the input middlepoints list filled with 0 that represents the distance to a road from those points.

Create a pointindex counter and set to 0

For each point in the input input middlepoints

    For each line in the roads list

        calculate the distance from the point to the closest point on the line.

        if this distance that is already saved for point at the pointindex is bigger than the calculated distance OR the distance that is saved is 0.

        replace the distance at the pointindex with the calculated distance.

    Add 1 to the pointindex

Normalize the values in the calculated distance to road list.
```

Outputs:

RoadDistanceValues (the list with the normalized roaddistances)

Traffic

Inputs:



MiddlePoints (from the Voxelgrid 3D node)

druk (roadcurves representing roads with a high crowdedness)

middle(roadcurves representing roads with a avarage crowdedness)

laag (roadcurves representing roads with a low crowdedness)

Create new list the size of the input middlepoints list filled with 0 that represents the crowdedness index at those points.

Per crowdedness index use the method to update the crowdedness index list. The inputs variate from index levels in the curves and the curve values that are used

Druk – input druk roadcurves and a curve value of 100

Middel – input middle roadcurves and a curve value of 50

Laag – input laag roadcurves and a curve value of 20

Method is on the next page

```
Method listupdater:
Create a pointindex counter and set to 0
For each point in the input middlepoints
    create a distance number and set to 10000
    For each line in the roads list
        calculate the distance from the point to the closest point on the line
        if the calculated distance is smaller than the distance number
            set a temporary crowdedness value to: curvevalue – the distance*1.5
                if the saved crowdedness value is smaller than the temporary one
                    replace the crowdedness value with the temporary one
                set the distance number to the line length of the used distance

normalize these values
```

---

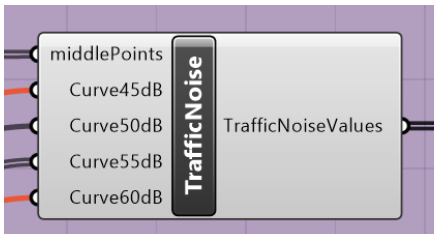
outputs:

trafficValues (the normalized crowdedness index at a point list)

TrafficNoise

Inputs:

MiddlePoints (from the Voxelgrid 3D node)



Curve45dB (roadcurves representing roads that have an average sound pressure of 45dB)

Curve50dB (roadcurves representing roads that have an average sound pressure of 50dB)

Curve55dB (roadcurves representing roads that have an average sound pressure of 55dB)

Curve60dB (roadcurves representing roads that have an average sound pressure of 60dB)

The source of the sound pressure: https://www.atlasleefomgeving.nl

---

Create new list the size of the input middlepoints list filled with 0 that represents the sound pressure at a those points

Per soundpressure group use the method to update the soundpressure at point list. The inputs variate from soundpressure group in the soundpressure and the roadrepresenting curves.

Curve45dB (Curve45dB roadcurves and a soundpressure of 45dB)

Curve50dB (Curve50dB roadcurves and a soundpressure of 50dB)

Curve55dB (Curve55dB roadcurves and a soundpressure of 55dB)

Curve60dB (Curve60dB roadcurves and a soundpressure of 60dB)

Method is on the next page

```
Method to update the soundpressure:
Create a pointindex counter and set to 0
    For each point in the input middlepoints
        Set tempdistance to 0
        For each line in the roadrepresenting curves list
            calculate the distance from the point to the closest point on the line
            if the calculated distance is bigger than the tempdistance
                set a temporary dB value to: soundpressure – $\log^{10}(4 * \pi * \text{calculated distance}^2 / 4)$
                    if the temporary dB value is bigger than the stored dB value at the pointindex
                    replace the stored dB value with the temporary one
                    set tempdistance to the calculated distance
      pointindex + 1
normalize these values
```

Outputs:

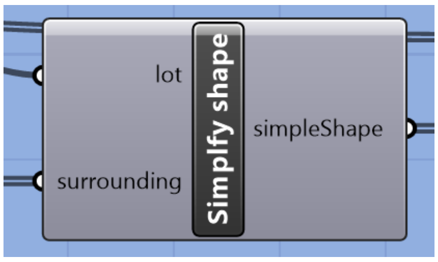
TrafficNoiseValues (the normalized dB at a point list)

Simplify Shape

Inputs:

Lot (represents the lot curve)

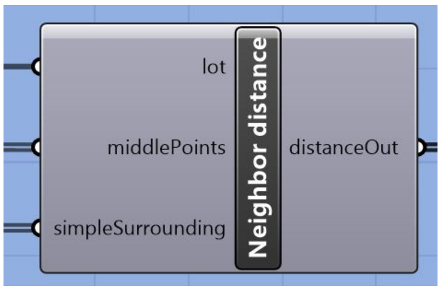
Surrounding (neighboring geometry from the model representing adjacent buildings)



```
Create a empty list for the simplified shapes

Create a minimum length threshold
Create a minimum height threshold

Join the surrounding breps to create one brep from adjacent smaller breps

For each brep of the joined breps

    get the edges of the brep

        For each line in the edges of the brep

            if the line is longer than the length threshold AND horizontal

            Save the line


    Create a vector based on this line

    Create a perpendicular vector to the first one

    Create a new local plane based on the two vectors with the start of the line as center point

    Create a bounding box based on the local plane around the brep

    Create a box based on the bounding box and the world plane

    A dd the box to the simplified shapes list
```

Outputs: simpleShape (the list of simplified shapes)

Neighbor distance



Inputs:

Lot (represents the lot curve)

MiddlePoints (from the Voxelgrid 3D node)

simpleSurrounding (from Simplify Shape node)

```
for every box in the simplesurrounding list
    change the box to a brep
    create a list of all the brepfaces
    For every face in the list of brepfaces
        create a line between the centerpoint of the face and the centrepoint of the lotcurve
        Store this line
    For each line in the stored lines
        Calculate the amount of intersections with the brep
        If there is only 1 intersection:
            foreach point in MiddlePoints
                calculate the distance from the point to the surface
                    if the distance stored for the point is bigger than the newly
                calculated distance OR the distance is equal to 0
                    update the distance with the newly calculated distance
normalize all the distances
```

Outputs:

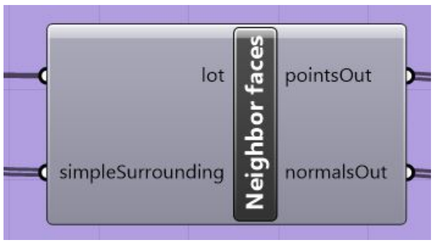
DistanceOut (list of the normalized distances)

Neighbor Faces

Inputs:

Lot (represents the lot curve)

simpleSurrounding (from Simplify Shape node)



```
create a u distance and a v distance variable for the grid dimensions
for every box in the simplesurrounding list
        change the box to a brep
        create a list of all the brepfaces
        For every face in the list of brepfaces
                create a line between the centerpoint of the face and the centrepoint of the lotcurve
                Store this line
        For each line in the stored lines
            Calculate the amount of intersections with the brep
            If there is only 1 intersection:
            Add the face to the list of stored faces


For each face in the stored face list
        Get the width and the length of the surface
        If the width is smaller than the set u distance
                Set the u division to 1
        Else
                Set the u division to the with/u distance

        If the length is smaller than the set v distance
                Set the v division to 2
        Else
                Set the v division to height/v distance

        For all the u divisions
                For all the v divisions
                        Place a point at the u,v coordinate
                        Create a vector based on the 1,1 location
```

Store the point and the vector

Outputs:

pointsOut(the list of the stored points)

normalsOut(the list of the stored vectors)

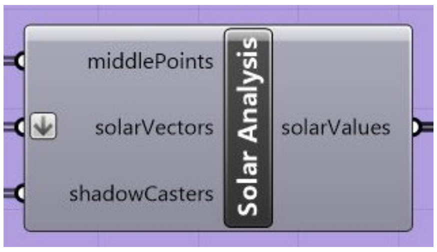
Solar Analysis

Inputs:

middlePoints(from the Voxelgrid 3D node)

solarVectors(sunVec from the solarTimeAngles node)

shadowCasters(ShadowCasting Buildings from the model)



```
set a evaluation line length
for each point of middlePoints
    set a temporary solar index of 0
    for each vector of SolarVectors
        create a line from the point in the negative direction of the vector and with a length
        equal to the evaluation line length
        for each building in the shadowcasters list
            check if the created line intersects the building
            count the amount of intersections
        if there are no intersections
            add 1 to the temporary solar index
    add the temporary value to a list of solar values
normalize the solar value list
```

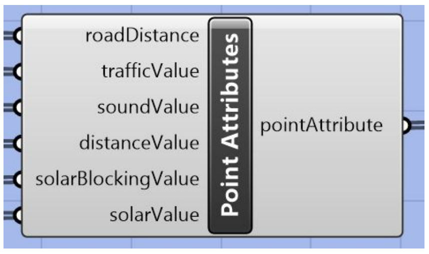
Outputs:

solarValues (the normalized solar value list)

SolarEnvelope

This node was edited to incorporate the shadows of the shadow casting buildings. The added code is similar to that of the solar analysis and due to that fact it will not be further described.

Point Attributes



Inputs:

roadDistance (RoadDistanceValues from the RoadDistance node)

trafficValue (TrafficValues from the Traffic node)

soundValue (TrafficNoiseValues from the TrafficNoise node)

distanceValue (distanceOut from the Neighbor distance node)

solarBlockingValue (Nblockage from the SolarEnvelope node)

SolarValue (solarValues from the Solar Analysis node)

```
For every index
        Create a tuple of all the values in all the lists at the index number
```

Outputs:

pointAttrinute (the list of tuples)

PreviewSwitcher

Inputs:

Info_1(RoadDistanceValues from the RoadDistance node)

Info_2(TrafficValues from the Traffic node)

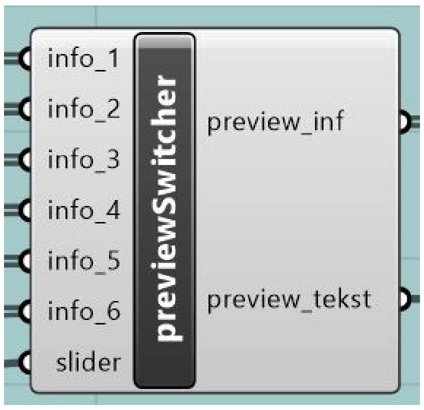
Info_3(TrafficNoiseValues from the TrafficNoise node)

Info_4(distanceOut from the Neighbor distance node)

Info_5 (solarValues from the Solar Analysis node)

Info_6(solarBlockingValue (Nblockage from the SolarEnvelope node)

Slider(generic slider with values between 1 to 6)



If the slider is the same as the info number output the values related to the info and output the info name

Output:

preview_inf (the values of the selected info)

preview_tekst (the name of the selected info)

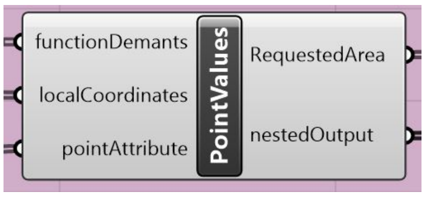
PointValues

Inputs:

functionDemants (a datatree from a opened excel file)

localCoordinates (localCoordinates from the VoxelGrid3D node)

pointAttribute (pointAttribute from the Point Attribute node)



```
extract the weights from the datatree and put every weight per function in a list
isolate the Areas and put in a list
for each function
    for each item in pointAttrubute
        multiply or divide the related value to the related weight so that it follows the formula:
        RoadDistance*w + distanceValue*w + SolarValue*w  – ( NoiseValue / 45 )*w
        + 1.5 * TrafficValue*w
        Save the value
    Store the list of values in another list representing the function
Normalize the values
```

Output:

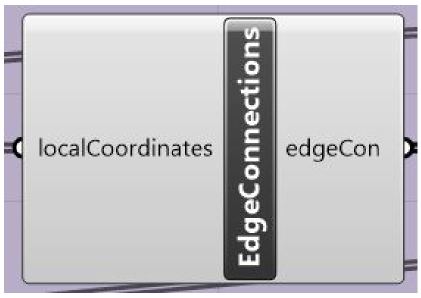
RequestedArea(the isolated Area list)

nestedOutput(the pointvalues per function)

EdgeConnections

Inputs:

localCoordinates (localCoordinates from the VoxelGrid3D node)



```
For each coordinate1 in localCoordinates
    For each coordinate2 in local coordinates
        If coordinate1 is not the same as coordinate2
            If the coordinates have the same I and j coordinate and the k coordinate is 1 of
                Make a tuple of coordinate 1 and 2
            Else If the coordinates have the same j and k coordinate and the I coordinate is 1 of
                Make a tuple of coordinate 1 and 2
            Else if the coordinate have the same I and k coordinate and the j coordinate is 1 of
                Make a tuple of coordinate 1 and 2
```

Outputs:

edgeCon(list of tuples of local coordinates)

Growing

Inputs:

pointValues (nestedOuput of the PointValues node)

pointRcoordinates (localCoordinates from the VoxelGrid 3D node)

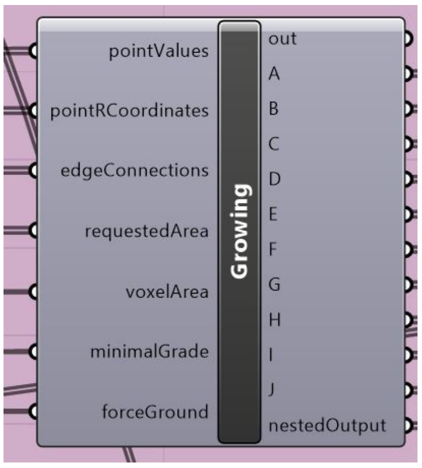
edgeConnections (edgeCon of the EdgeConnections node)

requestedArea (RequestedArea from the PointValues node)

voxelArea (Area from the voxel Info node)

minimalGrade (number from slider)

forceGround (Boolean toggle)



```
set the amount of iterations

create a occupied list the size of the pointRcoordinates

for each function

        if force ground

                for each point

                        store the index and the value and set to occupied if the value is bigger than
                        the already stored one and the point is not occupied and on ground level

        else

                for each point

                        store the index and the value and set to occupied if the value is bigger than
                        the already stored one and the point is not occupied


for each iteration until the max amount of iterations is reached

        for each function in pointValues

                for each point in occupiedlist

                        for each connection in edgeConnections

                                if the first item of het connection is the same as the index of the
                                point in the occupiedlist and the value of the connected point is
                                higher than the minimalGrade
```

```
                    if the maximum area is not yet reached
                        store point and set occupied
    store new points
```

Outputs:

A t/m J (output per function)

nestedOutput (the nested points)