IMPLEMENTING SPACE SYNTAX IN AN OPEN SOURCE GIS: GRASS GIS approach

100

Wen-Chihe (Jeffrey) Wang
Chaoyang University of Technology
Hsin-Ju Liao
Chaoyang University of Technology

Keywords:

Space syntax
GIS
GRASS
Open source
Network analysis

Wen-Chihe (Jeffrey) Wang
Chaoyang University of Technology, 168 Gifeng East Road, Wufeng Township,, Taichung County, 41349, Taiwan
jeffwang@cyut.edu.tw
Hsin-Ju Liao
Chaoyang University of Technology, 168 Gifeng East Road, Wufeng Township,, Taichung County, 41349, Taiwan

Abstract

Being a set of techniques for analyzing spatial patterns in buildings and cities, space syntax has been implemented in various software of spatial analysis. Due to the common requirement of handling spatial information, a significant portion of this spatial analysis software is built upon a geographic information system (GIS). Although these spatial analysis software may be available free of charge for academic or non-commercial use, they have to run inside a proprietary GIS such as MapInfo or ArcView, or a proprietary CAD such as AutoCAD or Microstation.

The emergence of free and open source software is one of the most significant computer-related developments in recent years. The philosophy is about freedom, which means users of such a computer program have the freedom to run it, study how it works and adapt it to their needs, redistribute it, and improve it and release the improvement to the public. This philosophy is particularly attractive to academia in that knowledge embedded in such software will be preserved, disseminated, and further developed regardless the commercial viability of the software. Linux operating system and Apache HTTP server are two most well known examples of free and open source software. In terms of GIS, Geographic Resources Analysis Support System (GRASS) is the oldest and most popular full-fledged GIS released as free and open source software. With the advent of GRASS 6 and its much improved vector map processing capabilities, it became feasible to implement the space syntax techniques in an open source GIS.

This paper describes an experimental implementation of the space syntax techniques in GRASS 6 that relies particularly on its vector network analysis modules and scripting capability. To test the effectiveness of the implementation, a case study is conducted using the same chosen study area but with two different street configurations, which are before and after the development of a special district for a high-speed railway station in the central part of Taiwan. The result shows that given the enormous capacities and flexibility of GRASS, the implementation is rather straightforward and works as expected. This demonstrates the true value of taking the open-source approach.

Introduction

How to analyze the built environment in a systematic manner has long been the subject of research in related fields of various scales ranging from interior design through architecture and landscape architecture to urban design and planning. Space syntax is one of the popular

approaches used by researchers in recent years. It was first conceived by Bill Hillier, Julienne Hanson, and colleagues at The Bartlett, University College London (UCL) in the late 1970s. Being a set of theories and techniques for the analysis of spatial configurations, space syntax can not only be used as a tool to help architects simulate the likely social effects of their designs, but also used in fields where spatial configuration seems to play a significant role, such as transportation, archaeology, information technology, urban and human geography, and anthropology (Hillier, 1998; Space Syntax Laboratory 2004a).

100-02

The basic idea behind space syntax is that it identifies the spatial configuration of a study area as a network, where nodes represent a unit of "space" and links represent connections between units of spaces. Through this approach we can then treat the analysis of spatial configuration as the well-established network analysis problem. Mathematicians have been working on the network analysis problems as early as the 18th century and accordingly developed the graph theory, which is now commonly used in computer science and many other fields where network analysis problems exist (Wikipedia contributors 2006b). Therefore terms, concepts, and algorithms of graph theory apply to space syntax as well.

Researches at UCL have implemented the space syntax techniques in various spatial analysis software (Space Syntax Laboratory 2004b). There is even more related spatial network analysis software developed outside UCL (Wikipedia contributors 2006c). Although most of this software is available free of charge for academic and non-commercial use, only one is open source software. In addition, because of the close relationship between space syntax and graphic theory, topology, and geometry, many of the software work as add-on or plug-in modules to a popular commercial computer-aided design (CAD) or geographic information system (GIS) package to utilize their fundamental capabilities. Those CAD or GIS are all proprietary and none is open-sourced.

Table 1 summarizes the spatial network analysis software that is listed on the Wikipedia web page. Table 1 shows that out of 14 options, at least 8 of them required people who are interested in space syntax to work on one particular platform. And for those who could not afford or for whatever reasons would not work on a certain proprietary CAD or GIS package, their options are rather limited.

However, this is exactly how an open source implementation of the space syntax techniques can play a significant role. The emergence of free and open source software is one of the most significant computer-related developments in recent years. The philosophy is about freedom, which means users of such a computer program have the freedom to run it, study how it works and adapt it to their needs, redistribute it, and improve it and release the improvement to the public (Free Software Foundation, 2006). This philosophy is particularly attractive to academia in that knowledge embedded in such software will be preserved, disseminated, and further developed regardless the commercial viability of the software. Linux operating system and Apache HTTP server are two most well known examples of free and open source software.

In terms of GIS, Geographic Resources Analysis Support System (GRASS) is the oldest and most popular full-fledged GIS released as free and open source software (Neteler & Mitasova 2002). Given the capabilities, flexibility, and popularity of GRASS, it is a no-brainer to choose GRASS as the platform to implement the space syntax techniques. It is even more so after the release of GRASS 6 in 2005, which has much improved vector map processing capabilities,

including those critical vector-based network analysis modules (GRASS Development Team, 2005).

The following portion of this paper first explains the basic concepts that are necessary to implement the space syntax in GRASS 6. It then describes the actual implementation in terms of the GRASS 6 modules and other UNIX utilities used. Finally it applies this experimental implementation to a real urban environment in order to test its effectiveness.

Basic Concepts

Definition of Space

The first step of using space syntax techniques to analyze spatial configuration is to identify the "space" in the study area. In other words, the study area needs to be break down into "spatial elements" in order to analyze their configuration. Here a spatial element means a convex empty area enclosed by objects such as wall, column, furniture, or plants in terms of architecture or landscape architecture. Take a floor plan of an apartment for example. There are some spatial elements that can be easily identified, such as rooms, while there are some spatial elements that cannot be identified easily, such as the corridor that provides the common access to all rooms. Therefore the originators of space syntax identify the spatial elements using the definition of convexity in mathematics and therefore call them convex space (Hillier, 1996). In mathematics a convex set means a set of points containing all line segments between each pair of points. Convex space is defined as "an occupiable void where, if imagined as a wireframe diagram, no line between two of its points goes outside its perimeter" (Wikipedia contributors 2006d). If the points represent people in a place, a convex space means that the line of sight between any two persons will never be blocked by the edge of the space, i.e. all people can see each other.

The same technique can be used at various scales from the interior of a small building to the open spaces in a city. To analyze the spatial configuration of an urban environment comprising mostly of buildings and streets, such as figure 1 shows, the way of identifying spatial elements can be further adapted. Although identifying long narrow streets as spatial elements works all right as shown in the left side of the figure 1, it is more convenient to identify them as axial space, a straight sight-line that also represents a possible path of movement. Therefore, in an urban environment consisting of mostly streets, a spatial element means a straight section of a street that is visible from one end to the other without obstruction. The right side of the figure 1 shows the axial lines represent the same spatial elements identified in the left side.

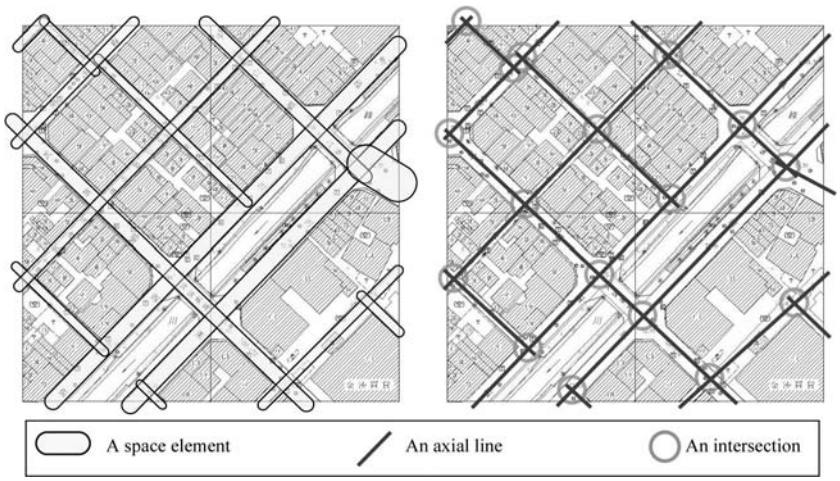


Figure 1:

Spatial configurations of urban streets

Connectivity of Space

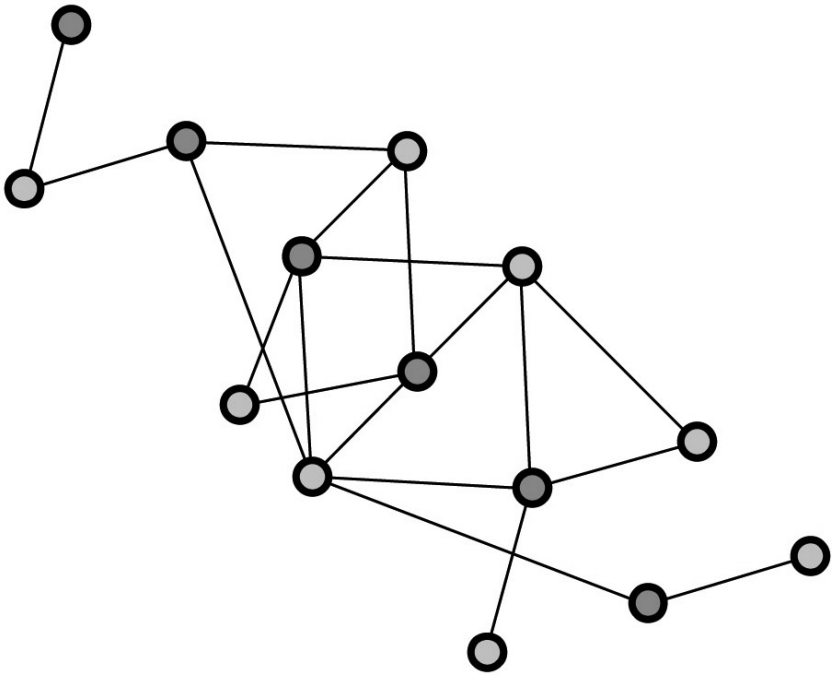
The second step of using space syntax techniques to analyze spatial configuration is to establish the "connectivity" of the spatial elements identified in the first step. For a study area such as a floor plan, identifying the connections or links among spaces is straightforward. One needs simply look for doors or gates in the floor plan. If the study area is in an urban environment, the connections or links among spaces are where the two spaces overlap, or where two axial lines cross, as identified by light gray circles in the right side of the figure 1. After both spatial elements and their connectivity are identified, Hillier (1996) suggested using a diagram called the justified graph, j-graph, to clearly depict the spatial configuration of the study area. A j-graph is similar to the diagram used in graph theory and network analysis that uses points and lines to represent nodes and links, respectively, in a network.

100-04

To depict the spatial configuration of an urban environment in a j-graph, the concept of nodes and links may need some adjustment. Take the study area shown in figure 1 for example. Although spatial elements are represented by axial lines in the right side of the figure 1, they should be represented by nodes instead, while connections are represented by links as usual. A diagram that represents the spatial configuration of the urban environment can be drawn as the one shown in figure 2. This diagram is a modified j-graph that uses dark gray circles to represent the northwest-southeast direction axial lines and light gray circles to represent northeast-southwest direction axial lines for easier identification of the original streets. This is where the j-graph deviates from the diagram of typical network analysis such as a road network where axial lines of roads are links and intersections of axial lines are nodes.

Figure 2:

A spatial configuration diagram of the area shown in figure 1



Depth of Space

A j-graph of the study area reveals two types of information. First, it shows the hierarchy of the spatial elements in terms of "depth" from one particular element. Secondly the "total depth" of each node, i.e. spatial element, in the j-graph can be calculated. According to Hillier (1996), the depth from one node to the other is the sum of the number of links in the shortest path between these two nodes. The total depth

of a node is simply the sum of the depths to all other nodes. This is the second point where the use of j-graph deviates from a typical diagram for network analysis. In a regular network the length or travel cost of a link is usually important, in space syntax however, the link is only used to identify connectivity between spatial elements. Thus the lengths or travel costs of links, which is called "depth" in space syntax, are all treated the same and set to 1 (unity).

Empirical studies have shown that in a study area a spatial element having a lower total depth means it is easier to navigate (Hillier 1996, 1998). If we color the spatial elements in figure 1 according to their total depth, similar to those shown in figure 6 and 7, we can easily discern how easily navigable a spatial element is. Hillier and others further found that easy navigability is not only useful for settings where way-finding is a significant issue, such as the design of museums, airports, and hospitals, but also applicable to predict the correlation between spatial layouts and social effects such as crime, traffic flow, sales per unit area, etc (Wikipedia contributors 2006d). For example, we can infer from figure 6 and 7 that where one is most likely to meet and interact with other people in that study area. Such a map of total depth shows the value of space syntax and explains why space syntax has been a popular approach in analyzing built environment since its inception.

Implementation

The Algorithm

In short, the ability of space syntax to measure the relative connectivity of different spatial elements in a study area is through the calculation of total depth for each element. The steps to calculate and analyze the total depth of spatial elements in a study area are listed as follows.

1. Identify spatial elements in a study area and draw them as axial lines.
2. Identify connections between spatial elements and draw them as links that connect each intersecting axial line pair by their midpoint.
3. From the finished graph of network, which is called "justified graph," or j-graph, find all shortest paths between each pair of nodes.
4. The depth of a node to another node is the distance of the shortest path between them, which is also the sum of the number of individual links within the path because all links' distances are set to 1.
5. The total depth of a node, i.e. spatial element, is the sum of its depths to all other nodes.
6. Inscribe the total depth of each node to the network, i.e. the j-graph. Present the j-graph in a thematic manner such as graduated colors or symbols.
7. Those spatial elements that have lower values of total depths mean they have higher potential to be used by pedestrian because of higher accessibility.

A GRASS Approach

Although the algorithm listed above is not complicated, when performed manually it will be quite tedious and quickly become insurmountable as the number of spatial elements increase. Therefore it is necessary to have a computerized solution to carry out the calculation of the total depths in order to use space syntax practically.

As mentioned before, there have been many solutions for this purpose. What this study does is not simply to reinvent the wheel but to develop a solution under an open-source framework so that all benefits of open-source software apply.

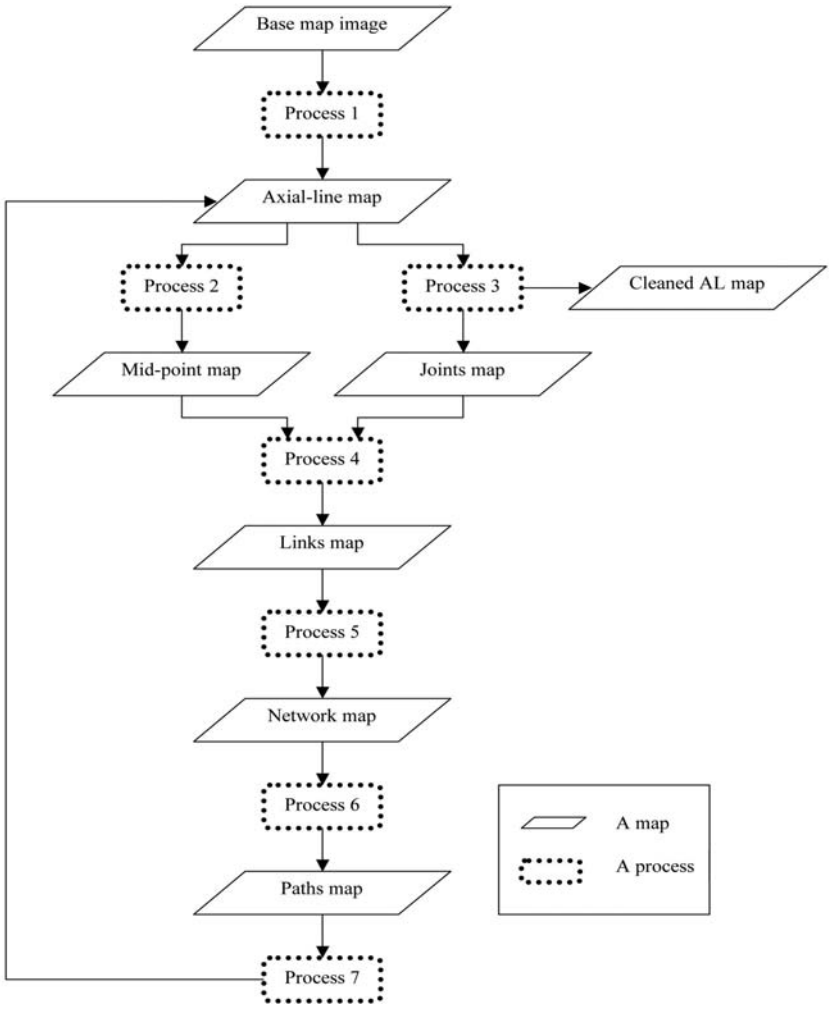
Given the aforementioned algorithm and the capabilities provided by GRASS 6, this study implements a computerized space syntax solution in the form of a standard operation procedure (SOP) to be carried out in GRASS 6. The SOP is then automated by putting together necessary commands and turning them into a UNIX Bash shell script. Since it is a prototype, it is not yet a fully automated solution and requires a few manual operations, such as digitizing the axial lines. However it does perform similarly to another solution based on ArcView, a popular proprietary GIS system, described by Batty (1998).

100-06

Figure 3 summarizes the SOP that implements the algorithm described in section 3.1 for calculating the total depth of a study area. In figure 3, a parallelogram represents an input or output map, while a rounded rectangle represents a process that includes several GRASS commands or operations of related UNIX utilities. The following text describes those processes in the order presented in figure 3.

Figure 3:

The standard operation procedure



Process 1

Process 1 creates the axial line map. This process first takes an image of the base map as the input and starts the digitizing function to let the user identify axial lines on the base map. Note that while in the digitizing operation, the user should not create the map's associated attribute table. The user should also choose "insert no category" mode

and turn off the “insert new record into table” option. The category number of each identified axial line is added by the v.category command. Also note that when digitizing, in order to avoid ambiguity and computing errors the endpoints of a line should not snap to any other endpoints, that is, no endpoint is connected by more than one line. The actual code listing follows.

```
d.mon x0
r.in.gdal base_map.jpg out=base_map
v.digit -n temp bgcmd="d.rgb red=base_map.red \
    green=base_map.green blue=base_map.blue"
v.category temp out=axial_map option=add
g.remove vect=temp
v.category axial_map option=report
d.vect axial_map display=shape,cat lcolor=black
```


Process 2

Process 2 creates the mid-point map. This process first sets the database connection for all attribute tables to SQLite, which is a full-function SQL open source database engine supported by GRASS. It then adds the axial line map’s attribute table and prepares necessary data. Finally it creates the map of midpoints of all axial lines from that table using a UNIX utility called awk. The actual code listing follows.

```
db.connect driver=sqlite \
    database='$GISDBASE/$LOCATION_NAME/$MAPSET/sqlite
.db'
db.connect -p
v.db.addtable axial_map columns="L double,HL double, \
    X1 double,Y1 double,X2 double,Y2 double, \
    M double,B double,depth_sum int"
v.info axial_map
v.to.db axial_map type=line option=length col=L unit=me
v.db.update axial_map col=HL value=L/2
v.db.select -c axial map | awk '
    BEGIN {FS="|"} {print "P", $1, $1, $3}' | \
    v.segment axial_map out=am_mp
d.vect am_mp display=shape,cat icon=basic/circle color=green
lcolor=green
```

Process 3

Process 3 creates the joints map. This process first adds the midpoint map’s attribute table and prepare axial line map’s attribute table for further processing. It then identifies which line intersects with which other lines and create the map comprising of intersection points of all axial lines, the joints map. The actual code listing follows.

```
v.db.addtable am_mp columns="X double,Y double"
v.to.db am_mp option=coor col=X,Y
db.select am_mp -c > am_mp_tab.txt
v.to.db axial_map type=line option=start column=X1,Y1
v.to.db axial_map type=line option=end column=X2,Y2
v.db.update axial_map col=M value="(Y2-Y1)/(X2-X1)"
v.db.update axial_map col=B value="Y1-(M*X1)"
db.select axial_map
v.clean axial_map out=am_cleaned error=am_err tool=break
v.category am_err out=am_joints option=add
g.remove vect=am_err
d.vect am_joints display=shape,cat icon=basic/box \
    color=orange lcolor=orange
```

Process 4

Process 4 takes both mid-point map and joints map as inputs to create the links map. This process checks each point on the joints map to find out the two axial lines passing through that point. It then connects each pair of passing axial lines by their midpoints to create the links map. The actual code listing follows.

```
v.db.addtable am_joints columns="X double,Y double"
v.to.db am_joints option=coor col=X,Y
db.select am_joints
rm am_joints.txt
np=`v.category am_joints option=report | awk '/point/ {print $2}'`
i=1
while [ $i -le $np ]
do
    echo "$i" > am_j_lines.txt
    echo "SELECT cat FROM axial_map WHERE
        (abs((SELECT Y FROM am_joints WHERE cat=$i) - \
        (M * (SELECT X FROM am_joints WHERE cat=$i)) -
B) < \
        0.0000001);" | sqlite3
$GISDBASE/$LOCATION_NAME/ \
        $MAPSET/sqlite.db >> am_j_lines.txt
    cat am_j_lines.txt | tr '\n' ' ' >> am_joints.txt
    echo "" >> am_joints.txt
    i=`expr $i + 1`
done
rm am_links.txt
nl=`awk 'END {print NR}' am_joints.txt`
i=1
while [ $i -le $nl ]
do
    P1=`awk '{if ($1 == I) {p1 = $2}} END {print p1}' I=$i
am_joints.txt`
    P2=`awk '{if ($1 == I) {p2 = $3}} END {print p2}' I=$i
am_joints.txt`
    echo "SELECT * FROM am_mp WHERE cat=$P1 OR
cat=$P2" | \
        db.select -c | awk '
        BEGIN { FS="|" ; print "L", "2", "1" }
        { print $2, $3 }
        END { print "1", I }' I=$i >> am_links.txt
    i=`expr $i + 1`
done
v.in.ascii -n am_links.txt out=am_links format=standard
d.vect am_links display=shape,cat color=blue
```

Process 5

Process 5 creates the network map required by the GRASS network analysis modules. It then prepares the network map's associated attribute table for further processing. Note that the number of points in the network map must equal to the number of lines in the axial line map and the number of points in the mid-point map. Otherwise it means that there are errors during the digitizing operation and therefore the user should go back to fix the axial line map and restart from process 1. The actual code listing follows.

```
v.net -c am_links out=am_net
d.erase
d.vect am_net display=shape,cat
d.vect am_net layer=2 display=shape,cat icon=basic/circle \
```

```
        color=blue llayer=2 lcolor=blue
    v.db.addtable am_net columns="arc_cost int"
    v.db.update am_net column=arc_cost value=1
    v.db.addtable am_net table=am_net_pt layer=2 columns="X
    double,Y double,depth_sum int"
    v.to.db am_net layer=2 option=coor col=X,Y
    v.category am_net option=report
    v.out.ascii am_net format=standard > am_net.txt
```

Process 6

Process 6 creates the final path map. This process first creates a text file that lists all possible origin-destination combinations in the network map. It then takes this file as input to create the path map showing the shortest path between each origin-destination pair and that path's distance. Note that the user should look at the path map's attribute table to see if there are unreachable destinations, which usually imply some problems in the previous processes. The actual code listing follows.


```
    rm am_od_pairs.txt
    na=`v.category axial_map option=report | awk '/line/ {print $2}'`

    count=0
    i=1
    while [ $i -le $na ]
    do
        j=`expr $i + 1`
        while [ $j -le $na ]
        do
                count=`expr $count + 1`
                echo "$count $i $j" >> am_od_pairs.txt
                echo "$count $i $j"
                j=`expr $j + 1`
        done
        i=`expr $i + 1`
    done
    cat am_od_pairs.txt | v.net.path -s am_net out=am_path
    afcol=arc_cost
    db.select am_path > am_path_tab.txt
```

Process 7

Process 7 calculates the total depth of each axial line. Because each link's distance used to compute the shortest path in the path map has been set to 1, the sum of each node's shortest distances to all other nodes is also the total depth of that node's corresponding axial line. The final calculation is to find the corresponding axial line of each node, which is that axial line's midpoint, and store the total depth information into the axial line map's attribute table for display and analysis. The actual code listing follows.

```
    na=`v.category axial_map option=report | awk '/line/ {print $2}'`
    i=1
    while [ $i -le $na ]
    do
        sum=`db.select am_path | awk '
                BEGIN { FS = "|" }
                {if (($3 == point) || ($4 == point)) {s = s + $6}}
                END {print s}' point=$i`
        v.db.update am_net layer=2 column=depth_sum value=$sum
    where="cat=$i"
        echo "Line $i depth sum = $sum"
        i=`expr $i + 1`
```

```
done
db.select am_net_pt -c > am_net_pt_tab.txt
na=`v.category axial_map option=report | awk '/line/ {print $2}'`
i=1
while [ $i -le $na ]
do
    X=`echo "SELECT X FROM am_mp WHERE cat=$i" | db.select -c`
    Y=`echo "SELECT Y FROM am_mp WHERE cat=$i" | db.select -c`
    DS=`cat am_net_pt_tab.txt | awk '
        BEGIN {FS="|"}{if (($2 == x) && ($3 == y)) {ds = $4}}
        END {print ds}' x=$X y=$Y`
    v.db.update axial_map column=depth_sum value=$DS where="cat=$i"
    echo $X $Y $DS
    i=`expr $i + 1`
done
db.select axial_map > axial_map_tab.txt
d.erase
d.vect.thematic -l axial_map type=line column=depth_sum \
    themetype=graduated_lines size=11 maxsize=3 nint=5
d.vect.thematic -l axial_map type=line column=depth_sum \
    colorscheme=red-blue nint=5
```


Verification

The Site

In order to test the effectiveness of the standard operation procedure described in the previous section, we applies the experimental implementation to the same chosen study area but with two different street configurations, which are before and after the development of a high-speed railway station. Figure 4 shows a recent satellite imagery of the study area that has been developed according to the special district plan of the Wu-ri high-speed railway station, while figure 5 shows the old street map of the same study area before developed.

Figure 4:

An old street map of the study area (©http://www.urmap.com)

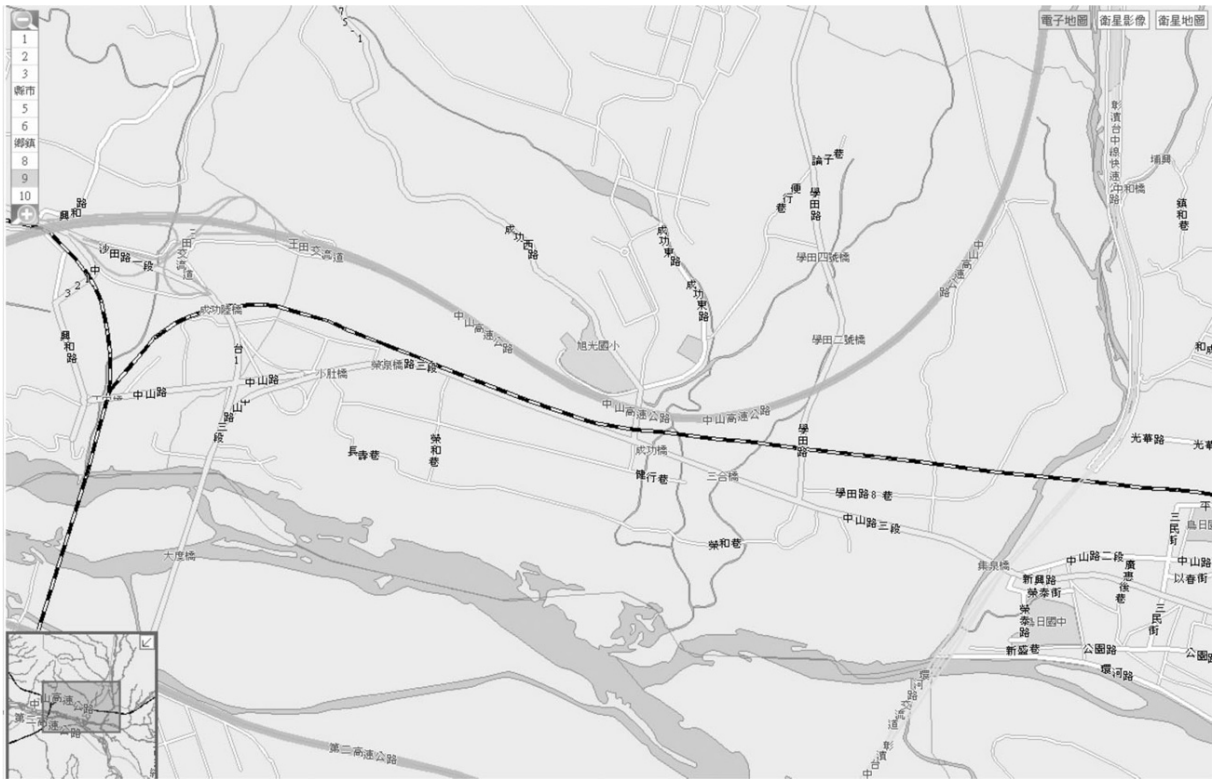


Figure 5:

A recent satellite imagery of the study area (©http://www.urmap.com)

Results

Figure 6 through 8 show the results of calculating total depth of the two different street configurations of the same study area using the SOP described in session 3.

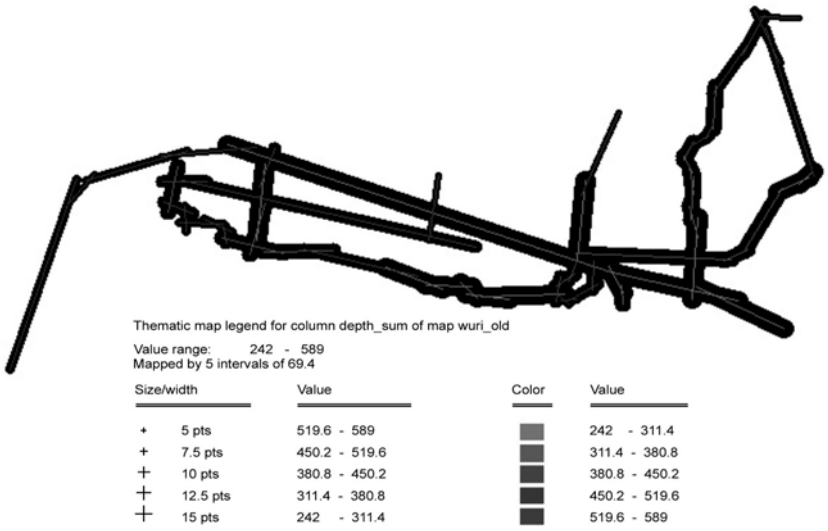


Figure 6:

A thematic map of the total depth of the old street configuration

Figure 7:

A thematic map of the total depth of the new street configuration

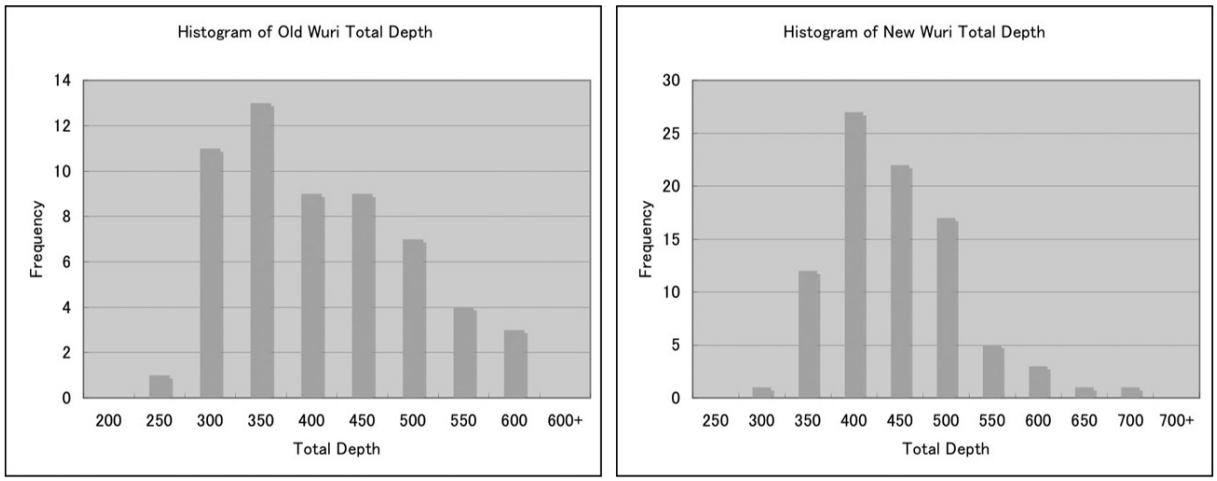


Figure 8:

Comparison of the total depth between old and new configurations

Discussion

The verification process shows that the implementation does work as expected. However, the bottleneck of running the calculation turns out to be portions of the Bash script, such as updating tables in the database or calculating all possible combinations of the origin-destination pairs. The v.net.path command operation that is originally thought to be both resource-demanding and time-consuming run surprisingly quick and efficient.

A significant issue of the current implementation is that digitizing the axial lines is a tedious work and prone to error due to the current user interface of the v.digit module. A temporary solution may be digitizing the axial line using other tools, such as the open source Quantum GIS. Besides, due to the current limitation of the d.vect.thematic module, the presentation of the final results is not easily discernable, as evident in figure 6 and 7. It would also be nice to see the thematic vector display command be further refined soon.

Finally because the current implementation is purely experimental and only tested in a few limited situations, the previously listed code may not be as robust as other currently available options. Because there is no any fault tolerance and exception handling mechanism in the code, they should be carefully examined before each execution.

Conclusion

This experimental implementation of space syntax techniques in GRASS proves that an open source alternative is a viable approach. It provides people another platform-independent solution to use space syntax techniques. Most importantly it is free, not only in terms of money but also in terms of liberty. Anyone who is computer literate and is given a couple hours of instruction can start to use this implementation on a well-prepared machine immediately. Those who are not afraid of using the command-line user interface and willing to take the time to download, install, and learn the freely available software can apply the space syntax techniques to their need, regardless their platform of choice. Over time this may increase the popularity of space syntax even further, especially in the developing countries. Finally, because this implementation is only a proof-of-concept, many improvements can be done. The greatest benefit of being an open-source solution is that anyone who is interested and willing to spend time on it can do his/her own improvement without getting other's permission or starting from scratch again.

References

Batty, M., Dodge, M., Jiang, B., Smith, A., 1998, "GIS and Urban Design", *Working Paper Series, paper 3*, Centre for Advanced Spatial Analysis, University College London, London, viewed 16 August 2006, <http://www.casa.ucl.ac.uk/urbandesifinal.pdf>.

Free Software Foundation, 2006, *The Free Software Definition, The GNU Project*, viewed 29 September 2006, <http://www.gnu.org/philosophy/free-sw.html>.

GRASS Development Team, 2005, *GRASS 6.0 Users Manual*, ITC-irst, Trento, viewed 16 August 2006, <http://grass.itc.it/grass60/manuals/html_grass60>.

Hillier, B., 1998, "The Common Language of Space: A Way of Looking at the Social, Economic and Environmental Functioning of Cities on a Common Basis", *Space Syntax Laboratory*, University College London, London, viewed 16 August 2006, <http://www.spacesyntax.org/publications/commonlang.html>.

Hillier, B., 1996, *Space is the Machine: A Configurational Theory of Architecture*, Cambridge University Press, Cambridge.

Neteler, M., Mitasova, H., 2002, *Open Source GIS: A Grass GIS Approach*, Kluwer Academic Publishers, Boston.

Space Syntax Laboratory, 2004a, "Introduction", *Space Syntax Laboratory*, University College London, London, viewed 16 August 2006, <http://www.spacesyntax.org/introduction/index.asp>.

Space Syntax Laboratory, 2004b, "Spatial Analysis Software", *Space Syntax Laboratory*, University College London, London, viewed 16 August 2006, <http://www.spacesyntax.org/software/index.asp>.

Wikipedia contributors, 2006a, "Convex", *Wikipedia: The Free Encyclopedia*, viewed 17 August 2006, <http://en.wikipedia.org/w/index.php?title=Convex&oldid=63532177>.

Wikipedia contributors 2006b, "Graph Theory", *Wikipedia: the Free Encyclopedia*, viewed 17 August 2006, <http://en.wikipedia.org/w/index.php?title=Graph_theory&oldid=70023273>.

Wikipedia contributors, 2006c, "Spatial Network Analysis Software", *Wikipedia: the Free Encyclopedia*, viewed 17 August 2006, <http://en.wikipedia.org/w/index.php?title=Spatial_network_analysis_software&oldid=65091697>.

Wikipedia contributors, 2006d, "Space Syntax", *Wikipedia: the Free Encyclopedia*, viewed 17 August 2006, <http://en.wikipedia.org/w/index.php?title=Space_syntax&oldid=43789685>.

Table 1:

Available spatial network analysis software for space syntax

	Space Syntax Software			Underlying Technology		
	Cost	Registration	Open Source	Type	Ownership	Platform
Ajanachara	Free	No	Yes (GPL)	Standalone	N/A	Linux, Windows
AJAX	Free	No	No	Standalone	N/A	Windows
AXess	Free	No	No	GIS: ArcView	Proprietary	Windows
Axman et al	Free*	Required	No	Standalone	N/A	Mac OS
Axwoman	Free	No	No	GIS: ArcView	Proprietary	Windows
Confeego	Free*	Required	No	GIS: MapInfo	Proprietary	Windows
Depthmap	Free*	Required	No	Standalone	N/A	Windows
Fathom	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown
Isovist Analyst	Free*	No	No	GIS: ArcView	Proprietary	Windows
Mindwalk	Free*	Required	No	Java	Proprietary (Free)	Independent
OmniVista	Unknown	Unknown	Unknown	Unknown	Unknown	Mac OS
OverView	Unknown	Unknown	Unknown	CAD: AutoCAD	Proprietary	Windows
Spatialist	Unknown	Unknown	Unknown	CAD: Microstation	Proprietary	Windows
Webmap	Free	Required	No	Java Applet	Proprietary (Free)	Independent

** Free of charge only for academic and non-commercial use.*