Implementing the Space Syntax Techniques:
a GRASS application for the analysis of spatial configurations

Wen-Chihe (Jeffrey) Wang*, Hsin-Ju Liao*

* Chaoyang University of Technology
168 Gifeng E. Rd, Wufeng, Taichung, Taiwan
Tel: +886 4 2332 3000 -7663
Fax: +886 4 2374 2385
E-mail: jeffwang@cyut.edu.tw

Abstract

Space syntax techniques were developed by Professor Bill Hillier and colleagues at University College London (UCL) in the 1980s to analyze spatial configurations. Based on the theory of graph, space syntax techniques measure the relative connectivity of different spaces in the built environment such as a city. By analyzing the accessibility at local street level, the space syntax is able to simulate the likely effects of different street patterns.

Researches at UCL have implemented the space syntax techniques in various spatial analysis software. Although all those software is available free of charge for academic use, none is open source software. Some even have to run inside a proprietary GIS such as MapInfo or ArcView.

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.

1 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 to early 1980s [11]. 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 [2][6].

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 [9]. 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 [7]. There are even more related spatial network analysis software developed outside UCL [10]. Although most of those software are available free of charge for academic and non-commercial use, none is open source software. In addition, because of the close relationship between space syntax and graphic theory, topology, and geometry, many of those 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.

However, this is exactly how an open source implementation of the space syntax techniques can play a significant role. Given the capabilities, flexibility, and popularity of GRASS [5], 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 [4]. 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 Bash scripts used. Finally it applies this experimental implementation to a real urban environment in order to test its effectiveness.

2 Basic Concepts

2.1 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 the floor plan shown in figure 1 for example. There are some spatial elements that can be easily identified, such as those 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 [3]. In mathematics a convex set means a set of points containing all line segments between each pair of points [8]. Convex space is then defined as "an occupiable void where, if imagined as a wireframe diagram, no line between two of its points goes outside its perimeter" [11]. 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. For example, those shaded boxes in figure 1 identify the spatial elements in the floor plan.

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 of mostly buildings and streets, such as figure 2 shows, however, 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 2, it is more convenient to identify them as axial space, a straight sight-line that also represents a possible path of movement [11]. 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 2 shows the axial lines represent the same spatial elements identified in the left side.

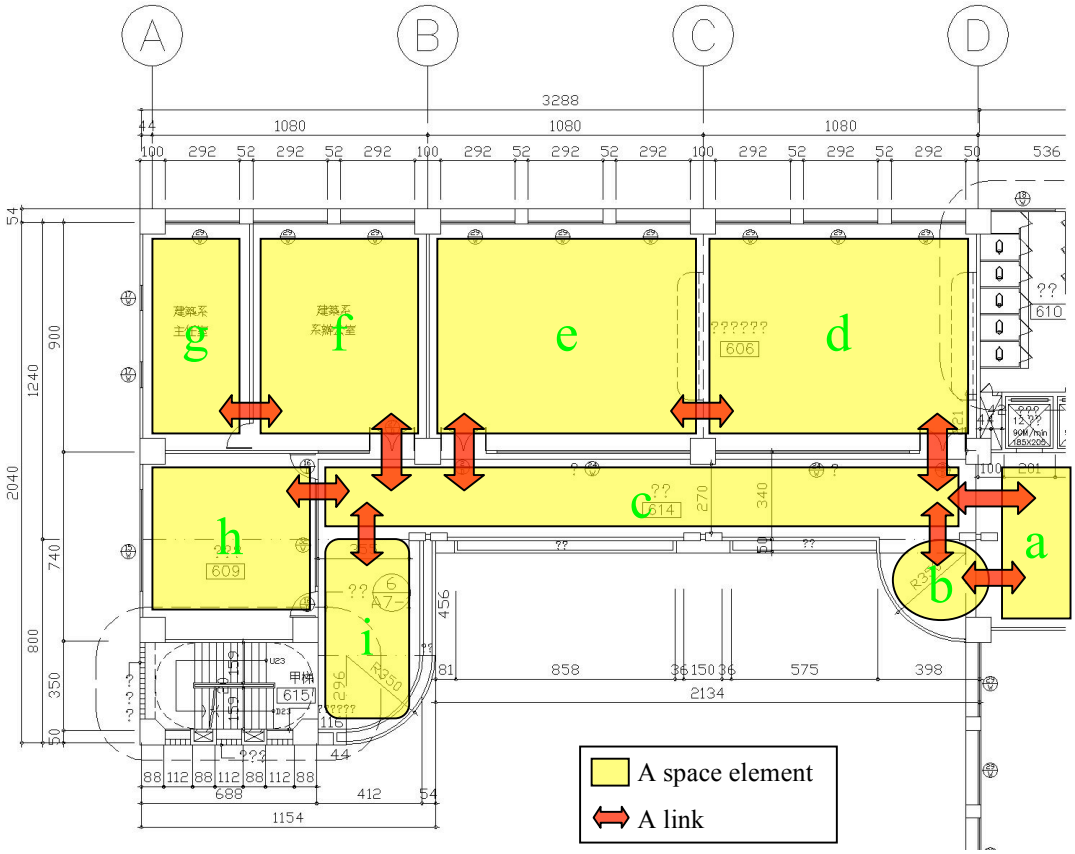

Figure 1 Analysis of spatial configuration

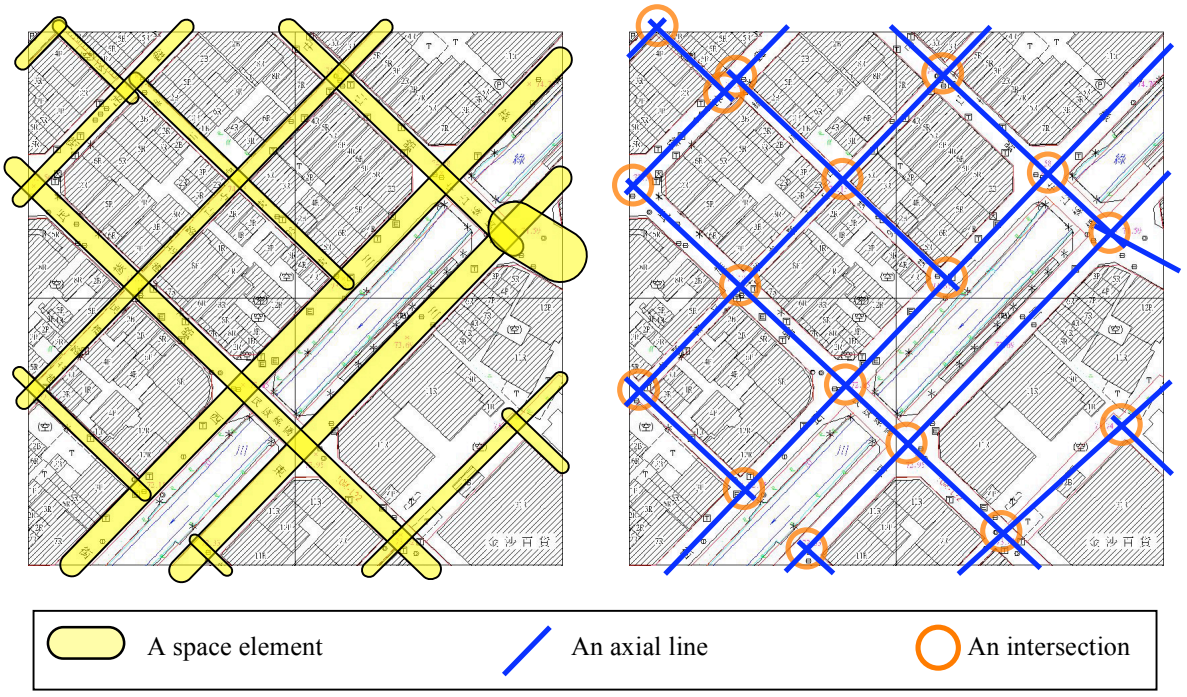

Figure 2 Spatial configurations of urban streets

2.2 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 the floor plan shown in figure 1, identifying the connections or links among spaces is straightforward. One needs simply look for doors or gates in the floor plan, as the arrows in the figure show. If the study area is in an urban environment, the connections or links among spaces locate where the two spaces overlap, or where two axial lines cross, as identified by orange circles in the right side of the figure 2.

After both spatial elements and their connectivity are identified, Hillier [3] suggested using a diagram he called the justified graph or simply the 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. For example, if we treat the spaces shown in figure 1 as nodes and the connections shown in the same figure as links, a j-graph that represents the spatial configuration of the floor plan can be drawn as the one shown in figure 3.

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 2 for example. Although spatial elements are represented by axial lines in the right side of the figure 2, they should be represented by nodes instead, while connections are represented by links as usual. A j-graph that represents the spatial configuration of the urban environment can be drawn as the one shown in figure 4. 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.

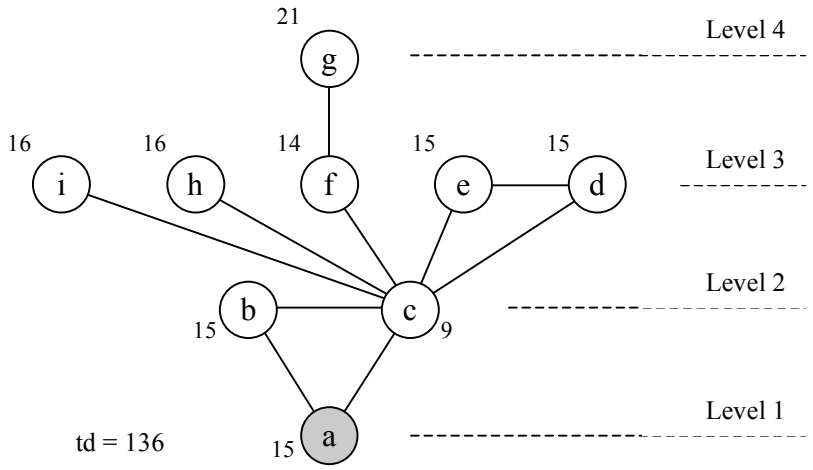

Figure 3 A j-graph of the spatial configuration in figure 1

2.3 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. For example, in figure 3 the waiting area outside elevators, as represented by node "a," is set as the entrance of the study area and all other spatial elements' depth from the entrance is clearly depicted.

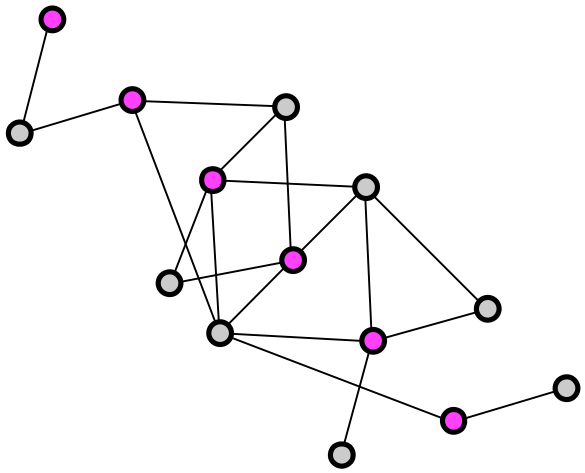

Figure 4 A j-graph of the spatial configuration in figure 2

Secondly the "total depth" of each node, i.e. spatial element, in the j-graph can be calculated. According to Hillier [3], the depth from one node to the other is the sum of the number of links in the shortest path between these two nodes. Take the entrance node "a" in figure 1 for example. The depth from the entrance node to node X is X as listed in table 1. The total depth of a node is simply the sum of the depths to all other nodes. In figure 3 the number next to each node is their total depth.

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 [2][3]. If we color the spatial elements in figure 2 according to their total depth, as shown in figure 8 and 10, 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 [11]. For example, we can infer from figure 8 and 10 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.

Table 1 Total depth calculation of the spatial configuration in figure 1

	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**Sum**
a	0	1	1	2	2	2	3	2	2	**15**
b	1	0	1	2	2	2	3	2	2	**15**
c	1	1	0	1	1	1	2	1	1	**9**
d	2	2	1	0	1	2	3	2	2	**15**
e	2	2	1	1	0	2	3	2	2	**15**
f	2	2	1	2	2	0	1	2	2	**14**
g	3	3	2	3	3	1	0	3	3	**21**
h	2	2	1	2	2	2	3	0	2	**16**
i	2	2	1	2	2	2	3	2	0	**16**
Sum	**15**	**15**	**9**	**15**	**15**	**14**	**21**	**16**	**16**	**136**

3 Implementation

3.1 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 is listed as follows.

1. Identify spatial elements in a study area and draw them as nodes.
2. Identify connections between spatial elements and draw them as links.
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.

3.2 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. Since it is a prototype, it is not yet a fully automated solution and requires quite a few manual operations. However it does perform similarly to another solution based on ArcView, a popular proprietary GIS system, described by Batty [1]. The following text describes what GRASS 6 commands and/or Bash scripts should be performed in each step of the above algorithm.

3.2.1 The 1st step of the aforementioned algorithm can be executed using the following GRASS 6 commands.

```
r.in.gdal basemap_image out=basemap                                    (1)

v.digit -n axial_line bgcmd="d.rgb red=basemap.red
        green=basemap.green blue=basemap.blue"                         (2)

v.category axial_line option=report                                    (3)

v.category axial_line out=axial_line_tmp option=del                    (4)

g.remove vect=axial_line                                               (5)

v.category axial_line_tmp out=axial_line option=add                    (6)

v.db.droptable axial_line                                              (7)

v.db.addtable axial_line columns="depth_sum int"                       (8)
```

```
v.category axial_line option=report                                    (9)
g.remove vect= axial_line_tmp                                          (10)
```

3.2.2 The 2nd step of the aforementioned algorithm can be executed using the following GRASS 6 commands.

```
v.clean axial_line out=axial_line_tmp error=intersection
        tool=break                                                     (11)
v.category axial_line_tmp option=report                                (12)
v.digit -n j-graph bgcmd="d.vect -c axial_line display=shape,cat;
        d.vect interaction icon=basic/circle"                          (13)
v.category j-graph option=report                                       (14)
v.category j-graph out=j-graph_tmp option=del                          (15)
g.remove vect= j-graph                                                 (16)
v.category j-graph_tmp out=j-graph option=add                          (17)
v.db.droptable j-graph                                                 (18)
v.db.addtable j-graph                                                  (19)
v.category j-graph option=report                                       (20)
g.remove vect= j-graph_tmp                                             (21)
```

3.2.3 The 3rd step of the aforementioned algorithm can be executed using the following GRASS 6 commands and Bash scripts that utilize some UNIX tools.

```
v.net -c j-graph out=j-graph_net                                       (22)
v.out.ascii j-graph_net format=standard > j-graph_net.txt              (23)
d.erase                                                                (24)
d.vect j-graph_net display=shape,cat                                   (25)
d.vect j-graph_net layer=2 display=shape,cat icon=basic/circle
        color=blue llayer=2 lcolor=blue                                (26)
v.db.addtable j-graph_net table=j-graph_pt layer=2
        columns="depth_sum int"                                        (27)
v.db.addcol j-graph_net columns="arc_cost int"                         (28)
v.db.update j-graph_net column=arc_cost value=1                        (29)
v.category j-graph_net option=report                                   (30)
limit=`v.category axial_line option=report | awk '/line/ {print
        $2}'`                                                          (31)
limit=$1                                                               (32)
rm wuri_points.txt                                                     (33)
count=0                                                                (34)
i=1                                                                    (35)
while [ $i -le $limit ]                                                (36)
do                                                                     (37)
   j=`expr $i + 1`                                                     (38)
```

```
   while [ $j -le $limit ]                                          (39)
   do                                                               (40)
      count=`expr $count + 1`                                       (41)
      echo "$count $i $j" >> node_pairs.txt                         (42)
      echo "$count $i $j"                                           (43)
      j=`expr $j + 1`                                               (44)
   done                                                             (45)
   i=`expr $i + 1`                                                  (46)
done                                                                (47)
cat node_pairs.txt | v.net.path -s j-graph_net out=short_path
       afcol=arc_cost                                               (48)
v.db.select short_path > short_path_tab.txt                         (49)
```

3.2.4 The 4th and 5th steps of the aforementioned algorithm can be executed using the following GRASS 6 commands and Bash scripts that utilize some UNIX tools.

```
i=1                                                                 (50)
while [ $i -le $limit ]                                             (51)
do                                                                  (52)
   sum=`v.db.select short_path | awk '                              (53)
      BEGIN { FS = "|" }                                            (54)
      {if (($3 == point) || ($4 == point)) {s = s + $6}}            (55)
      END {print s}' point=$i`                                      (56)
   v.db.update j-graph_net layer=2 column=depth_sum value=$sum
        where="cat=$i"                                              (57)
   v.db.update axial_line column=depth_sum value=$sum
        where="cat=$i"                                              (58)
   echo "Line $i depth sum = $sum"                                  (59)
   i=`expr $i + 1`                                                  (60)
done   (60)
v.db.select j-graph_net layer=2 > j-graph_pt_tab.txt                (61)
v.db.select axial_line > axial_line_tab.txt                         (62)
```

3.2.5 The 6th and 7th steps of the aforementioned algorithm can be executed using the following GRASS 6 commands.

```
d.vect.thematic -l axial_line type=line column=depth_sum
       themetype=graduated_lines size=11 maxsize=3 nint=5           (63)
d.vect.thematic -l axial_line type=line column=depth_sum nint=5
       colorscheme=red-blue                                         (64)
```

4 Verification

4.1 The site

In order to test the effectiveness of the standard operation procedure described in the previous section, this study applies this 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 5 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 6 shows the old street map of the same study area before developed.

Figure 5 A satellite imagery of the study area (©http://www.urmap.com)

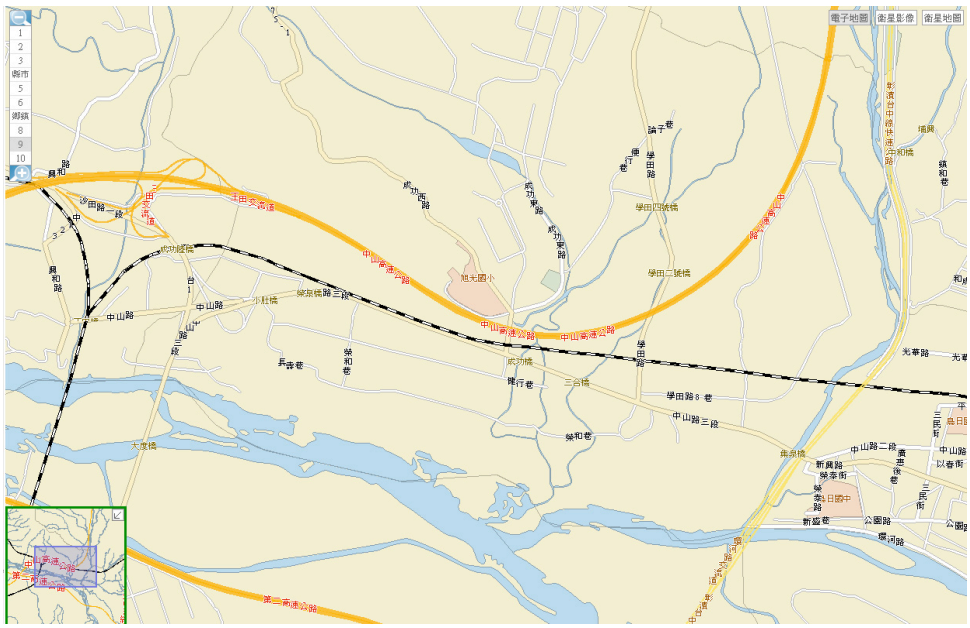

Figure 6 An old street map of the study area (©http://www.urmap.com)

4.2 Results

Figure 7 through 11 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.

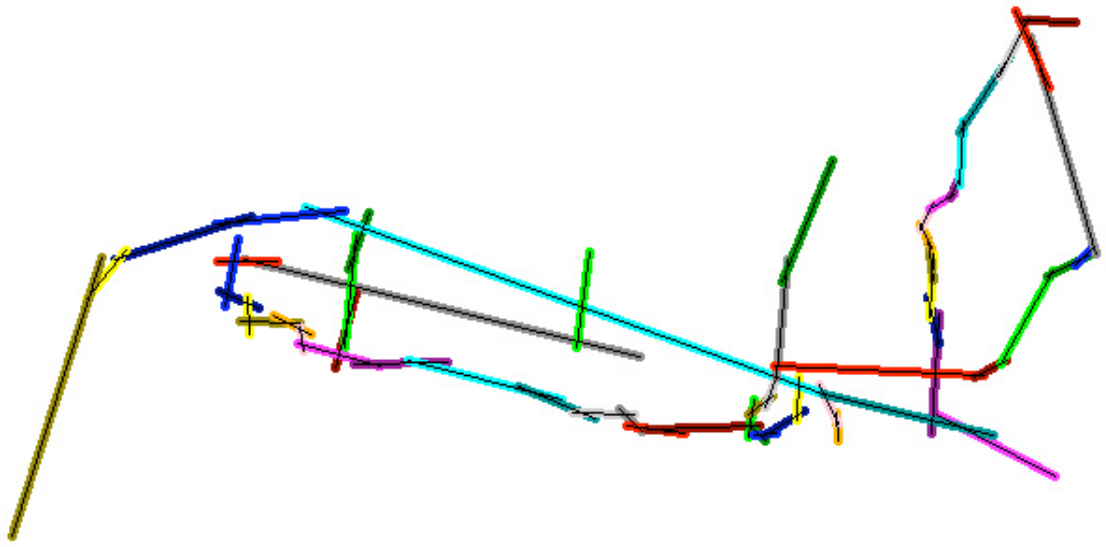

Figure 7 Identified axial lines of the old street configuration in random colors

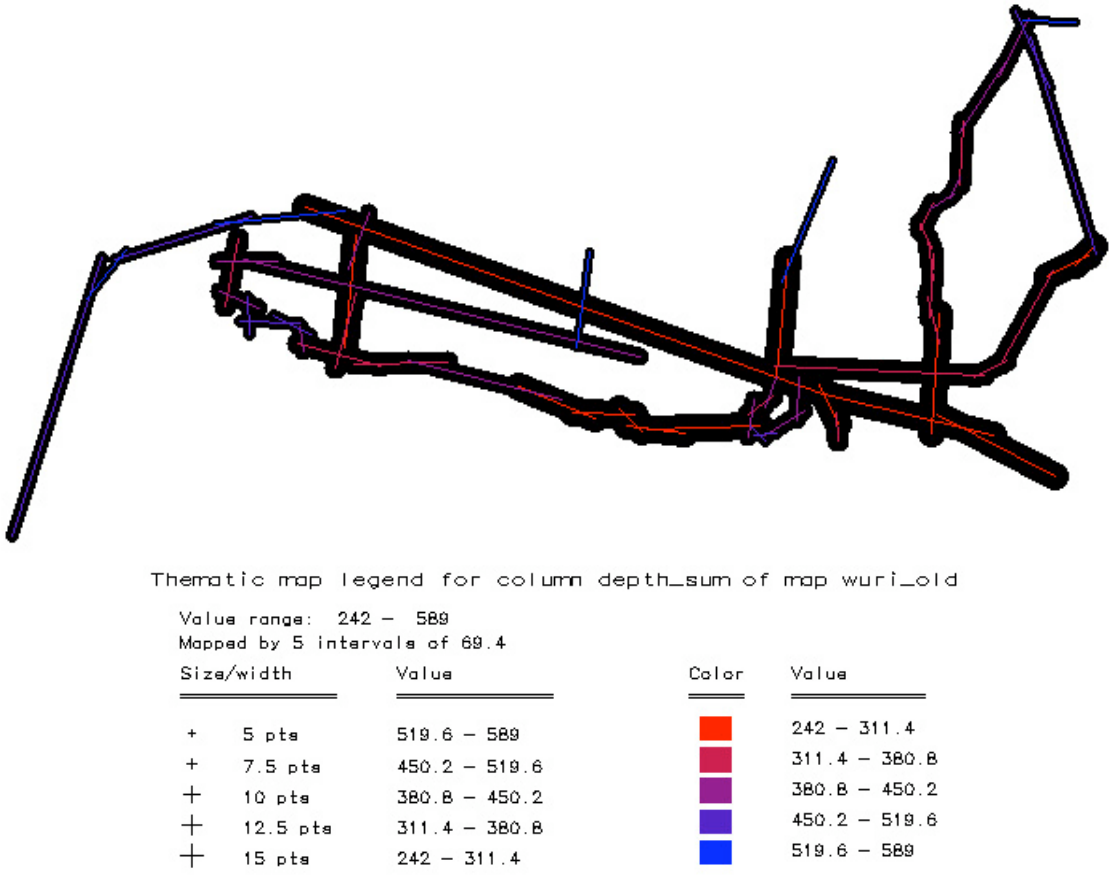

Figure 8 A thematic map of the total depth of the old street configuration

Figure 9 Identified axial lines of the new street configuration in random colors

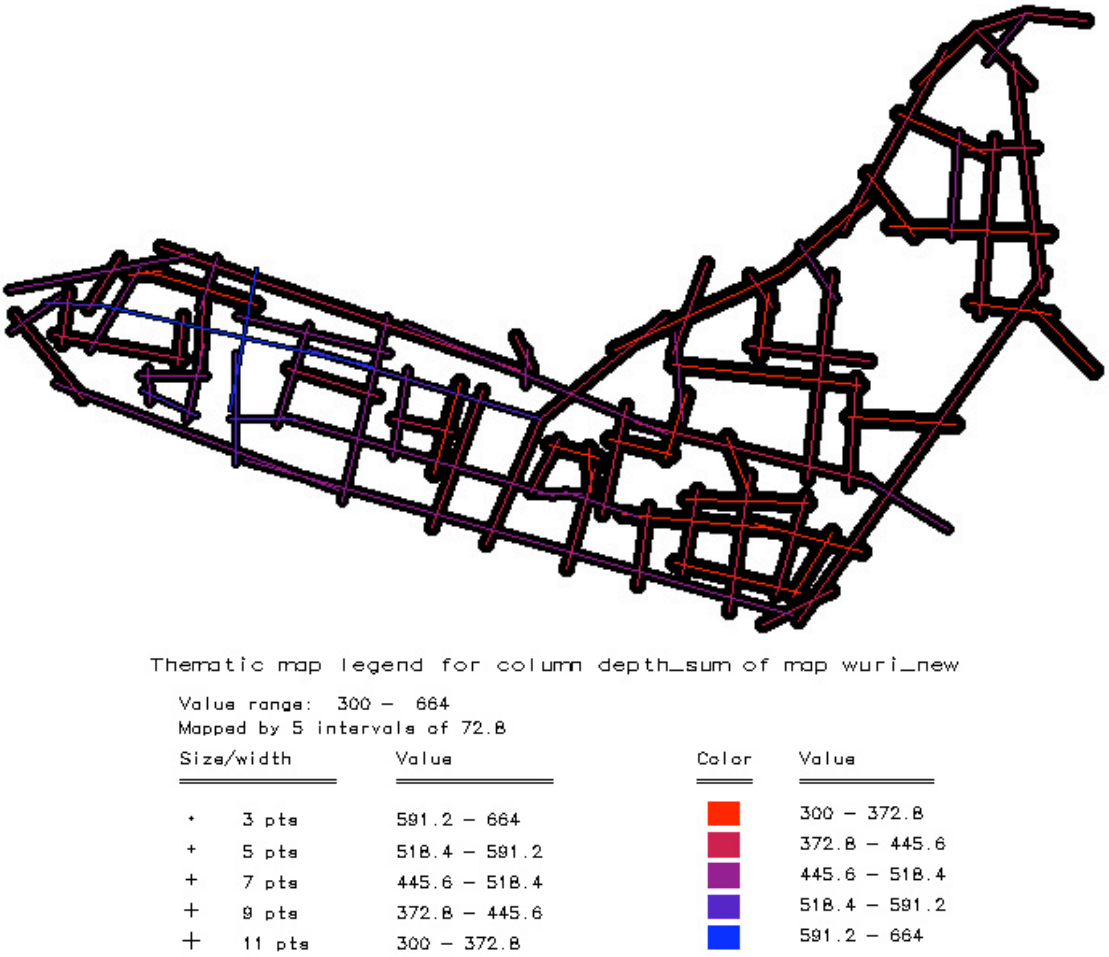

Figure 10 A thematic map of the total depth of the new street configuration

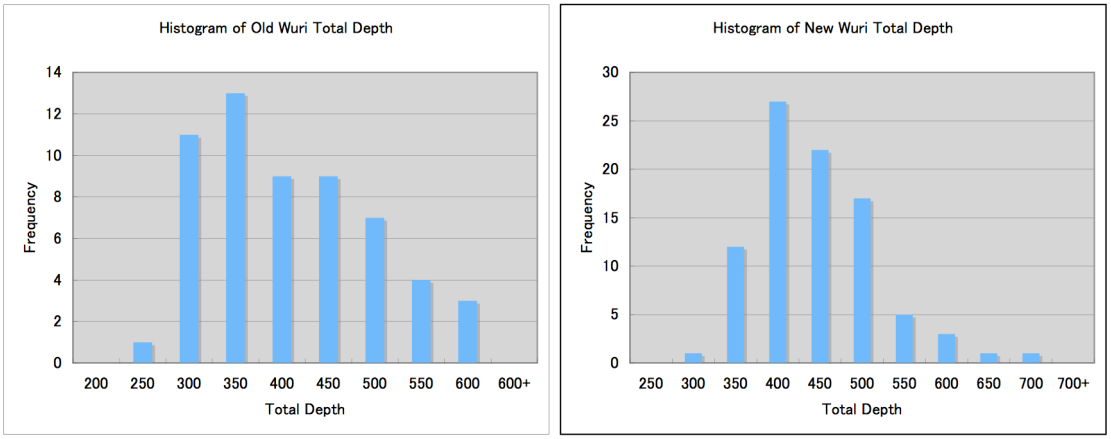

Figure 11 Comparison of the total depth between old and new configurations

4.3 Discussion

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

The biggest issue of the current implementation is that digitizing the axial lines as well as the j-graph is a tedious work and prone to error. Thus more automation is highly desirable. Some key functionalities necessary for further automation, nonetheless, are not yet available as vector commands as of GRASS version 6.1. This makes a more automated implementation of space syntax must be in the form of a custom programmed GRASSS module written in a lower-level programming language such as C. But again, suitable application programming interface (API) for doing this does not seem to be easily accessible in GRASS 6.1. Besides, due to the current limitation, it would also be nice to have the thematic vector display command be further refined.

5 Conclusion

In summary, space syntax is a popular tool for spatial configuration analysis. A robust implementation of space syntax techniques in GRASS could be both a proof of its capabilities and a boost for its adoption. However, it is not yet straightforward to do so. Although this study has shown that a proof-of-concept implementation of space syntax in GRASS 6 is feasible, automating the process of creating the j-graph should be implemented as soon as possible if it would really be put to use in a real world situation. In the long run, a refined implementation that can automate the process of identifying the axial lines will surely be a well-received improvement.

References

[1] M. Batty, M Dodge, B Jiang, and A. Smith, 1998. GIS and Urban Design, Working Paper Series, Paper 3. Centre for Advanced Spatial Analysis, University College London, London, UK. Available at: http://www.casa.ucl.ac.uk/urbandesifinal.pdf, accessed 16 August 2006.

[2] B. Hillier, 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, UK. Available at: http://www.spacesyntax.org/publications/commonlang.html, accessed 16 August 2006.

[3] B. Hillier, 1996. Space Is the Machine: A configurational theory of architecture. Cambridge University Press, Cambridge, UK.

[4] GRASS Development Team, 2005. GRASS 6.0 Users Manual. ITC-irst, Trento, Italy. Available at: http://grass.itc.it/grass60/manuals/html_grass60. Accessed August 16, 2006.

[5] M. Neteler, H. Mitasova, 2002. Open Source GIS: a GRASS GIS approach. Kluwer Academic Publishers, Boston, US.

[6] Space Syntax Laboratory, 2004. Introduction. Space Syntax Laboratory, University College London, London, UK. Available at: http://www.spacesyntax.org/introduction/index.asp. Accessed August 16, 2006.

[7] Space Syntax Laboratory, 2004. Spatial Analysis Software. Space Syntax Laboratory, University College London, London, UK. Available at: http://www.spacesyntax.org/software/index.asp. Accessed August 16, 2006.

[8] Wikipedia contributors. Convex. *Wikipedia, The Free Encyclopedia*. July 13, 2006, 03:20 UTC. Available at: http://en.wikipedia.org/w/index.php?title=Convex&oldid=63532177. Accessed August 17, 2006.

[9] Wikipedia contributors. Graph theory. *Wikipedia, The Free Encyclopedia*. August 16, 2006, 14:43 UTC. Available at: http://en.wikipedia.org/w/index.php?title=Graph_theory&oldid=70023273. Accessed August 17, 2006.

[10] Wikipedia contributors. Spatial network analysis software. *Wikipedia, The Free Encyclopedia*. July 21, 2006, 20:19 UTC. Available at: http://en.wikipedia.org/w/index.php?title=Spatial_network_analysis_software&oldid=65091697. Accessed August 17, 2006.

[11] Wikipedia contributors. Space syntax. *Wikipedia, The Free Encyclopedia*. March 14, 2006, 21:55 UTC. Available at: http://en.wikipedia.org/w/index.php?title=Space_syntax&oldid=43789685. Accessed August 17, 2006.