We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

8

Path-Finding Algorithm Application for Route-Searching in Different Areas of Computer Graphics

Csaba Szabó and Branislav Sobota
Technical University of Košice
Slovak Republic

1. Introduction

Common graphical notations for graphs use a set of points interconnected by lines. Points are called vertices. The lines called edges can also have in some cases a direction (orientation) specified.

Paths are sequences of vertices and edges between them. Path-finding between two locations in a graph is used in many areas like GPS navigation, CAD systems (auto-routing in design of circuit boards), and applications of artificial intelligence, computer games, virtual reality, military forces or robotics. The main aim of path-finding is to avoid collisions with obstacles and safely pass through the virtual world (or the real one) along the path found by the selected algorithm. Examples of this need are autonomous robots on distant planets without the possibility to be controlled in real time because of latencies in signal sending, or automatic vehicles control presented in (Simon et al., 2006) and (Kuljić et al., 2009). Another examples of path-finding are strategic computer games, mostly with computer opponent. Path-finding is also widely used in finding best routers interconnection for data transmission in many kinds of computer networks.

To be successful in path-finding, the need of reliable algorithm and reliable implementation of that algorithm is enormous. Another important thing is the need of some user-friendly visualization of results of path-finding, which is realized by application of computer graphics techniques (Vokorokos et al., 2010).

A geographical information system (GIS) as defined in (Tuček, 1998) is an information system (IS) designed to work with data that represent geographical or spatial coordinates. These data can be further analyzed and statistically evaluated.

Another way to specify a GIS is that it is a specialized IS defined as a collection of computer hardware (as noted by (Vokorokos et al., 2004)) and software and geographical data designed for effective gathering, retaining, editing, processing, analysis and visualization of all kinds of geographical information. Generally, it is an IS designed as specialized for spatial data, or an IS developed to such specialization during its life cycle by extensions and/or refactoring steps presented by (Szabó & Sobota, 2009).

Being an IS, GIS has to provide information in graphical (e.g. location of the hotel on the map) and non-graphical (e.g. fees, room equipment) way that supports fast searching

and actualization of data. Nowadays, graphical information are stored as vector graphics (two-dimensional), but the trends are clearly forecasting the second generation GIS as an IS working with 3D objects and surfaces as in (Szabó et al., 2010).

It is useful to extend the implementation of the above-mentioned searching of objects by route-searching between two selected objects. As the map is the core of any GIS, a possible way of implement path-finding on graphs within such a system is the mapping of these maps onto search-able graphs that will be shown in next sections of this chapter.

Section 2 presents the theoretical background needed in algorithm descriptions in the other following sections. In Sections 3–4, terms such as map and route are defined. Blind-search and A* algorithms are introduced in Section 5. Next section (Section 6) presents the test cases and results: results on static two-dimensional maps, results from tests where three-dimensional (city) maps were used, results of testing with dynamics in the maps. Tests also demonstrate the tested 3D city information system, especially its user interface, and hardware configurations of host computers used during testing. Section 7 ends the chapter by concluding it and pointing out future work.

2. Graph theory basics

Leonhard Euler is considered the founder of graph theory, since he solved the problem known as the Seven Bridges of Königsberg in 1736. Graph theory as a science discipline started with first graph theory monographs written by mathematicians (Kőnig, 1936) and (Berge, 1958). The next part of this section relies on definitions in (Bučko & Klešč, 1999).

Let V be a final non-empty set, i.e. the set of vertices, and let E be the set of edges defined as:

$$E = \{e | e = \{v_1, v_2\}; v_{1,2} \in V\}. \tag{1}$$

Obviously, $E \subseteq \binom{V}{2}$, where $\binom{V}{2}$ is a set of all two-element subsets of V:

$$\binom{V}{2} = \{A | A \subset V \wedge |A| = 2\}. \tag{2}$$

Considering these two sets presented in Equation 2 and Equation 1, the couple $G = (V, E)$ defines graph G with the corresponding vertices and edges.

Path is a sequence of vertices where an edge exists for any two consecutive vertices of this sequence. Degree of a vertex is the count of edges outgoing from this vertex, e.g. the degree of any internal vertex in a path is not less than two, the end vertices have a degree at least one.

The graph G is a coherent graph if all vertices are reachable, i.e. for all vertices exists at least one path to any other vertex.

Let $G = (V, E)$ be a coherent graph. Distance $d(u, v)$ between vertices u and v of graph G is the length of shortest route linking vertices u, v, i.e. the shortest path. Vertices distance $d(u, v)$ of graph G has the following properties:

$$\begin{aligned} d(u,v) &\geq 0; d(u,v) = 0 \Leftrightarrow u = v \\ d(u,v) &= d(v,u) \\ \forall z \in V : d(u,v) &\leq d(u,z) + d(z,v) \end{aligned} \tag{3}$$

These properties are metrics axioms; so ordered pair (V, d) is a metrical space. Nevertheless V is a vertical set of graph $G = (V, E)$, d is a metric – vertices distance in graph G. The advantage of these properties is the fact that they consist even after isomorphism is done. One result of this is the fact, that also notions defined by metric have the same advantage.

Let $G = (V, E)$ be a coherent graph, then:

1. For every vertex $u \in V$ the number $e(u, G) = \max d(u, v)$, $v \in V$ is called eccentricity of vertex u.
2. Number $P(G) = \max e(v, G)$, $v \in V$ is average of graph G.
3. Number $r(G) = \min e(v, G)$, $v \in V$ is radius of graph G.
4. The center of graph G is every vertex u of graph G with eccentricity equal to radius $e(u, G) = r(G)$.

Graph diameter can also be defined as $P(G) = \max d(u, v)$; $u, v \in V$.

Graph $G = (V, E)$ is an Eulerian graph, if this graph is coherent and every one of its vertices has even degree. Graph can be covered by a single enclosed stroke (i.e. Eulerian circuit) only if it is an Eulerian graph. Graph $G = (V, E)$ can be covered by a single unenclosed stroke only if it is a coherent graph with two vertices of degree 2 (opened stroke starts in one of these vertices and ends in another).

Let $G = (V, E)$ be a graph, $|V| = n$, $n \geq 3$. Let the degree of every vertex of graph G be at least $n/2$. Then G is a Hamilton graph. If graph $G = (V, E)$ is a Hamilton graph, this graph has to be terminal, non-empty, coherent and may not consist of bridges. Nevertheless fulfillment of these conditions does not guarantee existence of Hamilton graph.

Oriented graph G is defined as ordered pair (V, E), where V is the set of vertices and E is set of ordered element pairs of set V. Elements of V are called graph vertices and elements of E are called oriented graph edges.

3. Maps

The core element of any GIS is the map. All objects have spatial coordinates defining their placement on this map. On other hand, maps are also basic elements if it comes to path-finding or animations. Architecture of such a system for animations based on drawing objects on a 2D map is shown on Fig. 1, its output is presented on Fig. 2.

The specifics and role of maps in the selected domain areas are presented in the following subsections.

3.1 Maps in virtual reality

Simulations use 2D and 3D maps that are dynamically changing due simulation state changes. This type of maps is also used in any interactive graphics such virtual reality city walkthroughs or computer games.

Dynamic of the maps' change is expressed by structural changes in the maps, i.e. at least one object's spatial coordinates have been changed. Changes near found paths might indicate the need of re-running the path-finding algorithm on the new map segments.

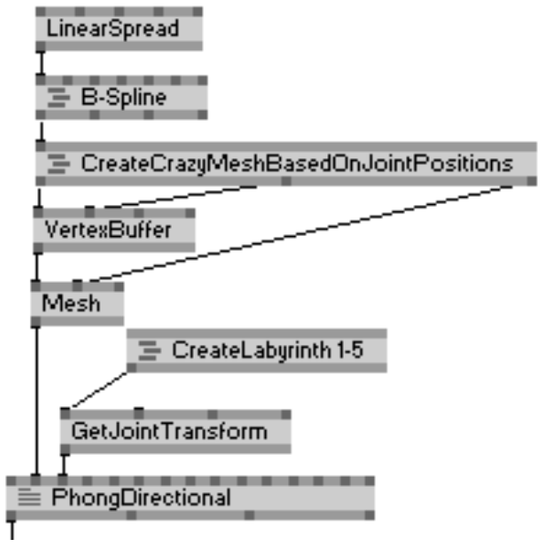


Fig. 1. Architecture of example system for 3D maze drawing using a 2D map

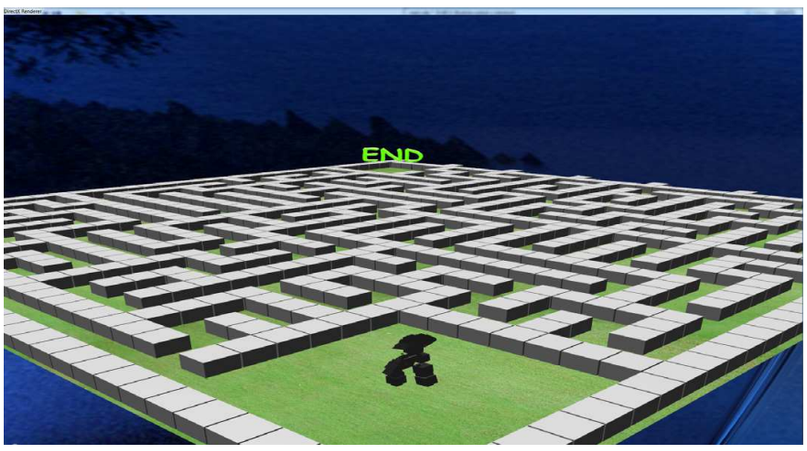


Fig. 2. Example system output with animated actor

3.2 Maps in GIS

Geographical information systems mainly use static maps. These maps could be both 2D or 3D or combined. Mostly, these maps are segments of cadastral maps, where the most important factor is the proper positioning of the segment root (see Fig. 3) that could be achieved by manual operation or using image filters (Póth, 2007).

3.3 Circuit boards as maps

Printed Circuit Boards (PCB) are well tested and proven technology. A manufacturer of electronics cannot imagine manufacturing of electronic equipment without this technology. Reliable operation of the current integrated circuits with high operating frequencies requires a minimum length of printed circuit conductors. Therefore, it is necessary to pay close attention to the PCB design mainly in the case of the optimal length of the printed circuit conductors. The conductive pattern may be on one side or both sides of the board and might be linked

Fig. 3. Correct (A) and incorrect (B) position of a segment of the city map designated for GIS internal map creation

in various ways. In the case of multilayer PCB, it might be between a few plates of copper conductive patterns. The creation and routing of conductors on PCB is the key of PCB design. In principle, the addressed issues are the issues discussed in this chapter.

The algorithm used in PCB design applications is the principal factor of their success. The most used algorithms in this area are heuristic algorithms, channel algorithm, rip-up algorithm (rip and reroute) and maze algorithms. These algorithms include, for example Lee router or also algorithms such as Blind search or A* described and compared later in this chapter.

In short, Lee router was first published more than 40 years ago and it was successfully implemented 20 years ago as a maze algorithm (Brown & Zwolinsky, 1990). In its simplest form, this router represents the algorithm to find a path between two points in 2D area with various obstacles. Its main features are ability to found always a path between the points, if any, within the limits of the work area size (PCB size) and this path is always the shortest route between two points.

However, it is very slow and, moreover, cannot organize its data structure to minimize gaps, and so it is being replaced by above mentioned algorithms now. Lee router organizes PCB space to the network of points. All unfilled points in the basic network are unmarked during algorithm initialization. The algorithm consists of two phases: an exploratory phase and a processing phase. Exploratory phase is similar with seed-fill algorithm used in computer graphics. The target point is on last place in the list and we travel back over the list of unfilled points to the original seeds in the subsequent processing phase. This original seed represents a starting point. Very simple and efficient improvement is the possibility to use 45 degrees angles too.

3.4 From map to graph representation

The basic working structure of algorithms is the map, which is represented as a set $M = N \times N$ (two-dimensional quadratic grid). Square is a member of set M. Every square represents position in map and has its own coordinates $[x, y]$ according to beginning P_m. Squares $[x1, y1]$ and $[x2, y2]$ are neighbors, if ($|x1 - x2| = 1$ and $|y1 - y2| <= 1$) or ($|x1 - x2| <= 1$ a $|y1 - y2| = 1$) (they are neighbors geometrically). The beginning in map P_m is a defined square. This representation is called map representation.

Next, a function is defined:

$$w_m : M \to \mathbb{R}^+ \cup \{+\infty\}. \tag{4}$$

This function assigns regress (e.g. weight, terrain cost) to every square. This function determines the difficulty of input for each square (or square traversing). If the regress is $+\infty$ then the square is impassable and contains an obstacle (black color nodes in Fig. 4, white color indicates passable vertices). This function is simple only if it contains a two-element set 1 and $+\infty$. In the map, the start square of the route is marked A, the target square is marked B.

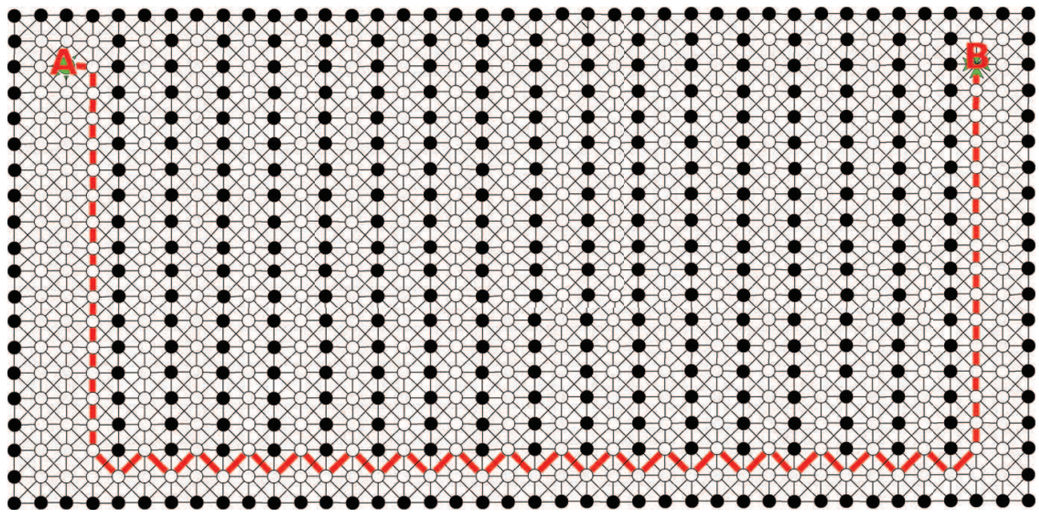


Fig. 4. Example graph of a maze with walls (black color), lanes (white color) and a route (red color)

4. Routes

Route in map is a sequence of map squares $(m_1, m_2, \ldots, m_n)$ in which every next square is neighbor to previous square on the route. There are no squares on the route that are included more than once. The route length is the sum of regresses of all route squares:

$$d = \sum_i w_m(m_i). \tag{5}$$

5. Algorithms

The algorithm task is to find the shortest route in map from the start to the target, i.e. from A to B.

The map representation is a special graph example. Vertices correspond to squares. The bijection transforming the graph to squares will be called map. Transition between neighbor squares corresponds to graph edge and initial vertex in graph P_g corresponds to initial vertex in map P_m. Next function to define is w, which will perform weight calculation on the graph. Function w_g is defined as representation from set of edges to set of non-negative real numbers with $+\infty$:

$$w_g : E \to \mathbb{R}^+ \cup \{+\infty\}. \tag{6}$$

For edge $e = (p, q)$, function w_g is defined as:

$$w_g(e) = w_m(map(q)). \tag{7}$$

Heuristics is defined as function $h : V \to \mathbb{R}^+ \cup \{+\infty\}$, which assigns the expected distance to target to vertex v. It estimates the length of shortest route from vertex v to the target.

The most common heuristics used for path-finding in 2D static maps are:

- Euclidean distance or Euclidean metric,
- Manhattan method that uses $\mathrm{d}x + \mathrm{d}y$, and
- the method of the maximum that uses $\max(\mathrm{d}x, \mathrm{d}y)$ as heuristics value.

There are several algorithms for route-searching in static maps:

- blind-search
- divide & conquer
- breadth-first search
- bidirectional breadth-first search
- depth-first search
- iterative depth-first search
- Dijkstra's algorithm
- best-first search
- Dijkstra's algorithm with heuristics (A*)

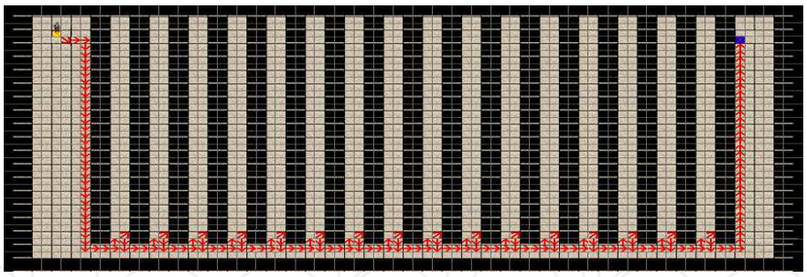
In our tests, the blind-search algorithm and Dijkstra's algorithm with Manhattan method of heuristics (A*) were used.

5.1 Blind-search algorithm

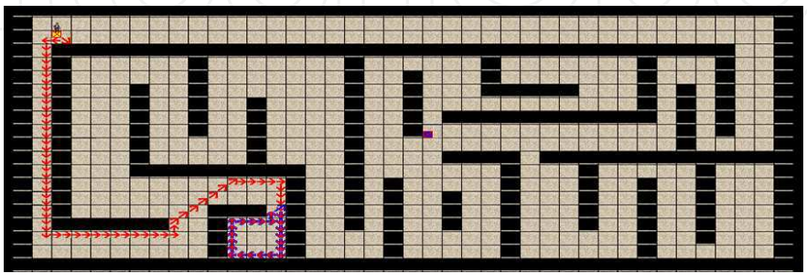
The blind-search algorithm (example results are shown in Fig. 5(a)–5(c)) is based on progress to the target, until it collides with obstacle, then the direction of movement is changed and the algorithm tries to move along nearest to get around the obstacle. This algorithm works with aim on one square and usually does not take into consideration the whole evaluation function w, but only its two states represented by values $+\infty$ or 1 (i.e. other than $+\infty$). So it works only with simple evaluating function and does not count with regress. This algorithm does not guarantee the finding of shortest route and can be used on special maps because of its speed, low memory requirements and simple implementation.

The evaluation process is as follows:

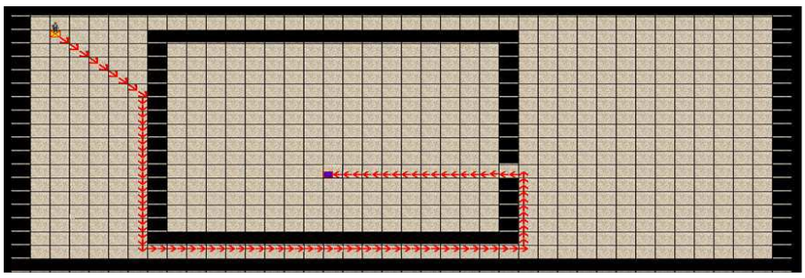
$$abs\left[(e_x - t_x) + (e_y - t_y)\right], \tag{8}$$



(a) Path found on Map1



(b) Trouble on Map2



(c) Path found on Map3

Fig. 5. Example paths found by the blind-search algorithm

where $e_{x,y}$ are the $[x, y]$ coordinates of the actually evaluated square and $t_{x,y}$ are coordinates of target square on the map.

The square with the lowest evaluation is chosen and set as new starting square for the next iteration. Continuing using this mechanism, the algorithm moves to target with the fastest possible approach. Because of the use of a simple evaluation, this algorithm is one of the fastest algorithms for path-finding.

Problem occurs when the evaluated square is impassable. If this situation occurs, the algorithm stops to evaluate squares and remembers the obstacle direction and then tries to avoid this obstacle. The avoid process is done by choosing the nearest available square, which is gained by turning the movement direction into right. This square is the new starting square, which is not evaluated but only checked if the direction with obstacle is not released. If yes, then the square in this direction is set as new starting square and the evaluation process starts again. If not, then the algorithm continues with new direction and tests first route. By this movement the algorithm can collide with new obstacles. It remembers again only the direction with obstacle and sets new course. This new course is tested and if it is released, then the

original direction is tested. If the algorithm collides with obstacle it stops to evaluate squares and tests transitivity of only one square. This can fasten the algorithm but can also become the algorithm into endless loop (see Fig. 5(b)).

5.2 A* algorithm

The A* algorithm (test results on Fig. 6(a)–6(c)) is a natural generalization of Dijkstra's algorithm and the scanning is based only on heuristics. In A* the element classification process is done by a binary heap. All square near the evaluated square are also evaluated in eight directions. Function g(x) is evaluated from the starting square. To work with integer values, a modification is made to the representation of Euclidean distance as follows:

- If the evaluated square is in the diagonal, then the number 14 is assigned to this square, and
- if not, then number 10 is assigned to this square.

As heuristics, the Manhattan method is used:

$$10 * \left[abs(e_x - t_x) + abs(e_y - t_y) \right], \tag{9}$$

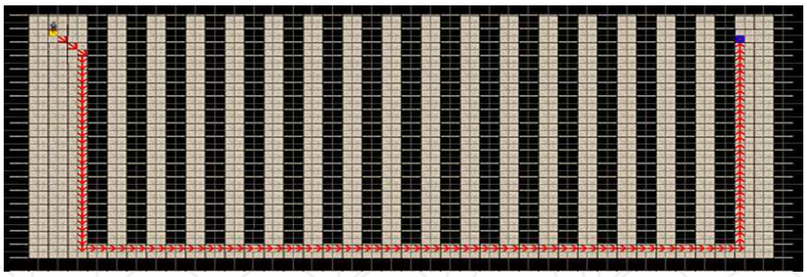
i.e. the distance between dx + dy is computed. Sequence function is defined as $f(x) = g(x) + h(x)$. The algorithm description follows.

Assume that graph $G = (V, E)$ has already evaluated edges, i.e. the mapping and weight computation is already done. Two values for every already processed vertex will be stored: $d[v]$ will store the length of the shortest path to the vertex, $p[v]$ will be the vertex before v on the path. The algorithm also uses two sets: $OPEN$ and $CLOSED$. $OPEN$ is the set of vertices to process, $CLOSED$ includes vertices already processed as presented in (Sobota, Szabó & Perháč, 2008):

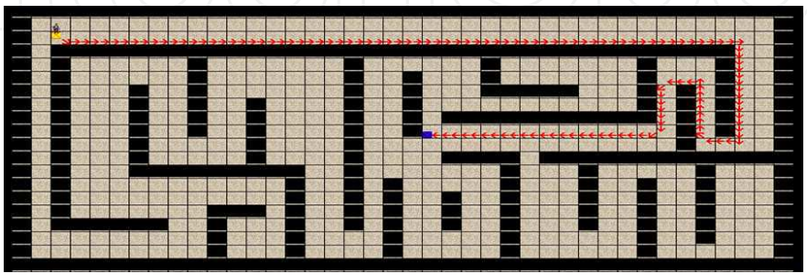
1. At the beginning, the starting vertex is inserted into $OPEN$.
2. The iteration cycle starts with the choose of the vertex n with best value of $f(n)$. This vertex will be inserted into set $CLOSED$ and all from n reachable vertices v are selected, and if they are not in set $OPEN$, they will be inserted with value $p[v] = n$ and the corresponding value of $d[v] = d[n] + l(n, v)$, where $l(n, v) = w_g(v)$ is the cost function for the selected edge from n. If v was already a member of $OPEN$, a test is needed if the new path is shorter than the old one by comparing $d[v] < d[n] + l(n, v)$. If the comparison fails, update of the values $p[v]$ and $d[v]$ is performed.
3. The presented cycle continues until the target is found or the set $OPEN$ is empty.
4. After the algorithm had finished, the path is reconstructed using values of $p[v]$.

Fig. 6(a)–6(c) show examples of usage of the A* algorithm on 2D static maps to compare with the results of the blind-search algorithm from the previous section.

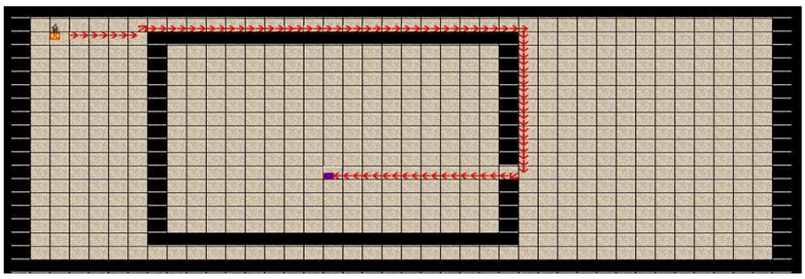
There are possible modifications to this algorithm that can decrease memory usage and fasten the A* algorithm. The most common techniques optimize the number of vertices within the sets $OPEN$ and $CLOSED$ by discarding the less useful ones. This technique limits the count of the elements of each set separately to get best performance but do not miss some possible paths. The common problem is the grid resolution selection: high resolution rapidly slows down the algorithm execution; low resolution may cause loss on important information as smaller buildings and/or gates. The maximum sizes of sets $OPEN$ and $CLOSED$ were



(a) Path found on Map1



(b) Path found on Map2



(c) Path found on Map3

Fig. 6. Example paths found by the A* algorithm

examined and experimentally defined in (Sobota, Szabó & Perháč, 2008) and (Vlasova, 2008) as 1000 and 3000.

6. Algorithm use cases and evaluation results

6.1 Test results for static 2D maps - the maze example continued

A couple of tests have been run on two-dimensional static maps to compare the blind-search and A* algorithms, which implementations were based on (Adams, 2003; Barron, 2003). Ten (10) maps were tested.

Routes found differ in length (compare results on Fig. 5(c) and Fig. 6(c)). The route always starts with red arrow. If the route is returning by the same route backward, then the arrow is changed to yellow and the route continues with yellow arrow. If the route crosses itself (blind-search algorithm), then it is in cycle and the arrow will be marked by boxed blue color (see 5(b)). Yellow square is start square, blue square is target square.

The efficiency of blind-search algorithm is only 40% because it found a route only in four of 10 maps. A* algorithm searched in all maps 14 895 squares together. Blind-search needed to search 83 928 squares, because it used to cycle itself (see Fig. 5(b)) and the condition of cycle detection was at 10000 cycles. The blind-search algorithm searched 5.64 times more squares than A* algorithm. Considering only those maps, where both algorithms succeeded, blind-search algorithm would be the better one in speed and number of squares searched, respectively.

We present the test results of both algorithms on Fig. 7. It is obvious that blind-search was not so effective like the A* algorithm when compared the count of squares searched on all maps. The speeds on all maps are not comparable due to the low efficiency of the blind-search algorithm.

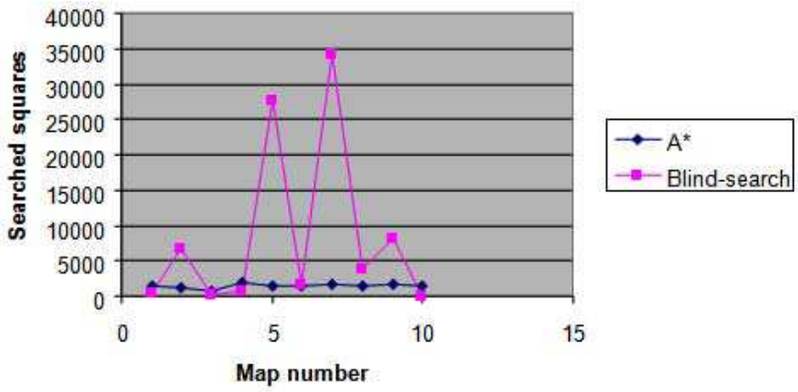


Fig. 7. Statistical results comparison for both algorithms gathered on 10 maps

6.2 Testing with static 3D city maps

Two-dimensional maps, as presented above, are a good testing tool, but in real systems three-dimensional ones are more frequently asked. Representing higher fidelity to real world, virtual reality techniques help to build more user-friendly and more precise information systems.

Dealing with geographical information, a three-dimensional map (or more precisely, the visualization of it) introduces a few new problems into the path-finding by extending the amount of information to be searched.

Therefore, tests on more complicated surfaces and maps with building objects, hills etc. were also run. The result is shown on Fig. 8 in the form of a route found from point A to point B between the buildings of a randomly generated city.

Tests on optimizing of sizes for sets *OPEN* and *CLOSED* were also performed. As already mentioned, in (Vlasova, 2008) were made some experiments under the supervision of the authors. The testing conditions were as follows:

- each test case was run separately on the same hardware (no sequences etc.),
- visualization (rendering) speed was set to minimum, and
- each test case was run ten times and results were averaged.

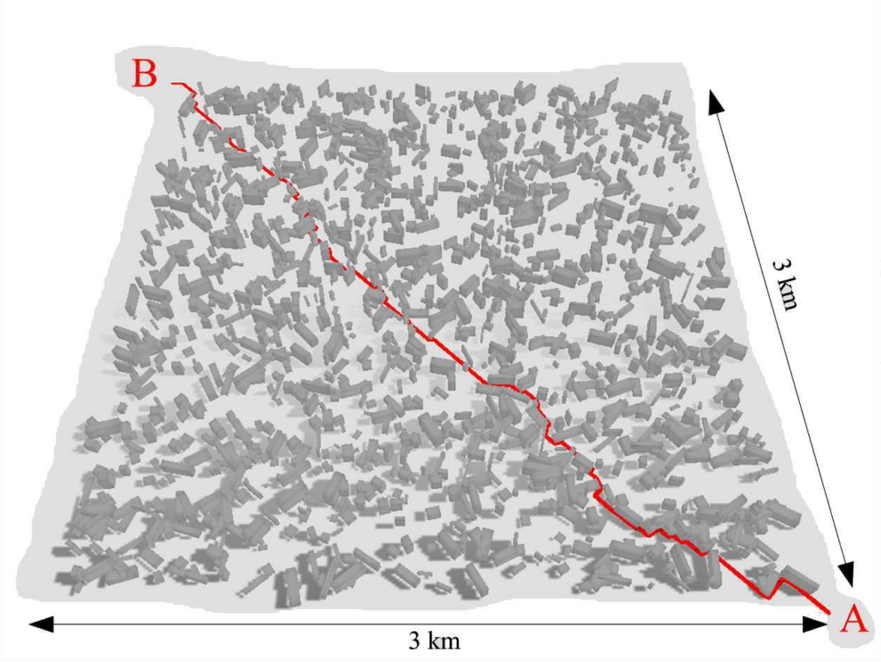


Fig. 8. Path found by A* algorithm on a random 9km^2 city map

Optimization technique	Average execution time [s]
none	489
CLOSED	458
OPEN	368
OPEN & *CLOSED*	346

Table 1. Speed test results for A* algorithm optimization

The test results are shown on Table 1 and Table 2. The best performance was achieved by dual optimizing (i.e. using both limits), but the optimization of size of set *OPEN* had stronger effect on execution speed than the size optimization of set *CLOSED*.

Optimization technique	Path length [m]	$\|OPEN\|$	$\|CLOSED\|$
none	5210	6022	5210
CLOSED	5210	6022	3000
OPEN	5210	1000	5263
OPEN & *CLOSED*	5210	1000	3000

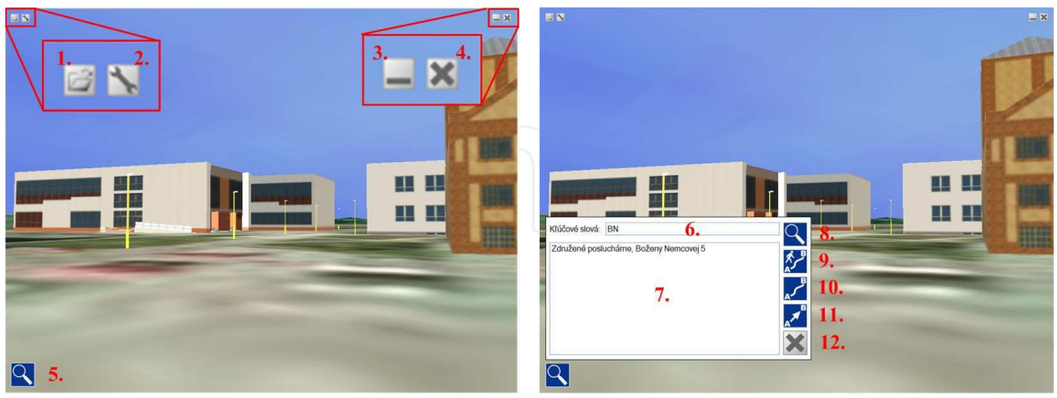
Table 2. Set sizes and path lengths during A* algorithm optimization

6.2.1 The 3D City IS test case

The tested A* algorithm was implemented within a GIS called the 3D city IS, details of the system can be found in (Sobota et al., 2010; Sobota, Korečko & Perháč, 2009; Sobota, Perháč & Petz, 2009; Sobota, Szabó, Perháč & Ádám, 2008; Sobota, Szabó & Perháč, 2008).

Fig. 9(a) shows the main screen of the tested GIS. There are a few building objects shown on the base map. In the upper right corner are navigation buttons (ref. no. 3 for minimize, ref.

no. 4 – close). Upper left corner contains also two buttons: ref. no. 1 is for opening a map; ref. no. 2 point on the options menu. 'Options' are e.g. map resolution. The ref. number 5 indicates the search button, which functionality was tested in the test cases.



(a) The main screen (b) Path searching dialog

Fig. 9. 3D city IS test case setup

Selecting this last option, a search dialog opens that is shown on Fig. 9(b). As any classical search dialogs, it has a field for the search query (reference number ref. no. 6), results displaying area (ref. no. 7), search execution button (ref. no. 8), animation button (ref. no. 9), path display button (ref. no. 10), object display button (ref. no. 11), and close dialog button (ref. no. 12).

The testing procedure is simple:

1. Open a map, e.g. in this case each of the following maps was used:
   - a map with some known labeled objects randomly generated by other tool,
   - the real (but yet only partial) city map, and
   - the map of the university campus.
2. Type a search phrase and evaluate what has been found and displayed.
3. Now follows the test of path-finding abilities.
   - Function go-to-object only moves the camera to the selected target.
   - Function show-path opens after a few minutes the resulting route from the actual position on the map to the selected target as shown on Fig. 8.

The speed is faster with less resolution, but some narrow streets have been missed. With the campus map are nearly real-time results due to the small count of objects.

The virtual reality oriented part of the tested geographical information system is the animation function. The path-finding module was shared with the other already mentioned parts. The test case had to ensure the quality of the obstacle detection. This part offers path visualization by circles and the animated walkthrough of the scene rendered from the map and additional object information stored in the GIS database. Fig. 10 shows the animation screen. Buttons labeled 13–16 are classical navigation buttons for movement within the animation. This test case is evaluated by expressing the quality and feel by human testers; and were used in the debugging phase of development of the path animation part of the system.

Fig. 10. Walk-through animation of 3D city information system

6.3 Path-finding results in dynamic 2D maps

In virtual reality applications (also in computer simulators and games), the effect of reality is significantly achieved by dynamics of the scene.

For problem simplification, the next test case uses pseudo 3D solution, meaning that the scene is a composition of 2D (map) and 3D objects. Note that in this case path-finding algorithms have to deal with less data, i.e. weight function and final calculations could be performed faster. The same simplification could be made in the case of any type of systems listed above.

There were 10 maps created with different amount and positions of static obstacles. Tests were provided for 1, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 192 and 384 active moving objects called units. Test cases were oriented on calculation time (e.g. speed of path-finding in relation to CPU) and visualization time (e.g. the role and impact of GPU) while comparing the two selected algorithms.

Tables 3 and 4 show the contrast between the algorithms during the testing process. While considering all maps as in Table 3 A* algorithm seems to be better in every case, the fact is that there are averaged values for all maps, where blind-search algorithm failure to find a path incorrectly implies being this algorithm the slower one. Considering only calculation time on maps, where both algorithms succeed shown in Table 4 reflect that blind-search algorithm is faster but less stable.

Next group of tests was designed to compare speeds of both path-finding algorithm implementations on different hardware CPUs. Table 5 shows the test configurations used.

Although the implementations should not be affected by the hardware and results are not used as CPU metrics, Tables 6–7 present the values. The only conclusion to make is that the slower A* algorithm could be evaluated faster on a fast CPU than the faster blind-search algorithm using a slower CPU.

Units	1	8	16	24	32	40	48	56
A* [s]	0.022	0.146	0.281	0.381	0.514	0.658	0.849	0.966
BS [s]	0.043	0.42	0.853	1.262	1.707	2.303	2.756	3.233

Units	64	72	80	88	96	192	384	
A* [s]	1.077	1.126	1.324	1.453	1.665	3.169	6.343	
BS [s]	3.717	4.247	4.705	5.173	5.671	11.376	23.799	

Table 3. Average time for 10 maps for A* and blind-search (BS) algorithm in relation to No. of units

Units	1	8	16	24	32	40	48	56
A* [s]	0.009	0.058	0.112	0.152	0.206	0.263	0.340	0.386
BS [s]	0.001	0.008	0.016	0.025	0.032	0.044	0.052	0.058

Units	64	72	80	88	96	192	384	
A* [s]	0.431	0.450	0.530	0.581	0.666	1.268	2.537	
BS [s]	0.065	0.075	0.081	0.087	0.097	0.189	0.390	

Table 4. Average time for A* and blind-search (BS) algorithm in relation to No. of units (for maps, where BS succeed)

Name	CPU	GPU	RAM
TConf1	AMD Duron 1200 MHz	Geforce MX400/64MB	256 MB
TConf2	AMD Athlon 1700+	Radeon 9200SE	512 MB
TConf3	AMD Athlon 2000+	Geforce MX460/64MB	512 MB
TConf4	AMD Barton 2500+	Geforce 5900	2 GB
TConf5	Intel P4 Centrino 1.7 GHz	Intel GMA 900	512 MB

Table 5. Hardware test configurations

Units	1	8	16	24	32	40	48	56
TConf1 [s]	0.000	0.016	0.031	0.047	0.047	0.062	0.094	0.094
TConf2 [s]	0.002	0.016	0.020	0.032	0.042	0.053	0.066	0.074
TConf3 [s]	0.002	0.010	0.018	0.029	0.039	0.046	0.056	0.065
TConf4 [s]	0.002	0.007	0.017	0.023	0.031	0.039	0.047	0.055
TConf5 [s]	0.000	0.016	0.015	0.016	0.031	0.031	0.047	0.047

Units	64	72	80	88	96	192	384	
TConf1 [s]	0.109	0.125	0.125	0.140	0.156	0.328	0.640	
TConf2 [s]	0.086	0.090	0.106	0.117	0.125	0.252	0.504	
TConf3 [s]	0.074	0.085	0.095	0.104	0.113	0.226	0.452	
TConf4 [s]	0.065	0.073	0.078	0.085	0.092	0.189	0.377	
TConf5 [s]	0.047	0.062	0.062	0.078	0.078	0.157	0.297	

Table 6. Average times for A* algorithm using different hardware configurations

Last group of tests was aimed to measure average time for animations of unit movement on path from A to B. In test realization, for places A and B constant locations were used. Time was measured including path-finding, but the additional condition was to achieve the same formation of the units in both locations, e.g. for 96 units a 8×12 rectangle. Test results are shown in Table 8 and Table 9.

Units	1	8	16	24	32	40	48	56
TConf1 [s]	0.000	0.000	0.000	0.016	0.031	0.031	0.031	0.031
TConf2 [s]	0.000	0.007	0.014	0.012	0.017	0.021	0.027	0.035
TConf3 [s]	0.000	0.004	0.008	0.012	0.017	0.022	0.027	0.032
TConf4 [s]	0.000	0.003	0.006	0.009	0.013	0.016	0.020	0.023
TConf5 [s]	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.015

Units	64	72	80	88	96	192	384	
TConf1 [s]	0.047	0.047	0.063	0.063	0.063	0.109	0.250	
TConf2 [s]	0.040	0.044	0.045	0.052	0.055	0.117	0.229	
TConf3 [s]	0.035	0.040	0.042	0.046	0.051	0.104	0.209	
TConf4 [s]	0.029	0.032	0.035	0.039	0.040	0.084	0.168	
TConf5 [s]	0.015	0.031	0.031	0.031	0.031	0.078	0.156	

Table 7. Average times for blind-search algorithm using different hardware configurations

Units	1	96	192	384
TConf1 [s]	39	63	84	124
TConf2 [s]	31	31	35	50
TConf3 [s]	15	19	30	35
TConf4 [s]	13	13	25	28
TConf5 [s]	22	28	43	50

Table 8. Average animation times for A* algorithm using different hardware configurations

Units	1	96	192	384
TConf1 [s]	55	92	117	167
TConf2 [s]	43	43	49	70
TConf3 [s]	21	27	42	49
TConf4 [s]	18	19	34	39
TConf5 [s]	31	39	60	68

Table 9. Average animation times for blind-search algorithm using different hardware configurations

7. Conclusion

This chapter presents a view on graph theory from point of view of virtual reality. Applications and testing of blind-search algorithm and the modified Dijkstra's A* algorithm of path-finding in graphs within a geographical information system were presented.

From the test results some conclusions can be made. The blind-search algorithm is fast but is not accurate enough. It can be used in maps with low number of obstacles only when the results have to be gained as fast as possible and the route does not have to be the shortest one.

A* algorithm is the most used algorithm and showed its qualities in our tests as well. Despite time results (the speed of this algorithm is not the fastest), this algorithm is reliable and has provided better results. The optimization techniques of set size restrictions were useful and brought higher speed into execution of path-finding.

The implementation of both algorithms showed up as useful, but the significant difference of applicability between them excludes the blind-search algorithm from the set of candidates of route-searching algorithms for a GIS.

Algorithms were implemented into a 3D city information system as a prototype of a GIS. The included functionalities except those presented include the object browser and map editor. The future plan is to interconnect the system with other systems for geographical data processing such points clouds analyzers, video or ortophoto processors etc.

The similarity to today's GPS in the case of route- searching is intentional. According to the trends in the area, future navigation systems might offer three-dimensional interfaces as well. The only difference is in the type of the map, because our system uses a static map while a good navigation system uses a dynamic one that might future GIS use too.

As mentioned in the introducing section, the use of graph theory becomes common in modern computer graphics and virtual reality systems. Collision detection is the main area of it, but, as this article implies, the information system area should follow these trends as well by implementing its algorithms into the visualization and searching modules.

8. Acknowledgements

Authors thank R. Švábý, M. Švábý (M. Vlasova) and V. Kríž for their constructive work in the implementation and debugging phase of development of the presented computer graphics software applications.

This work was supported by VEGA grant project No. 1/0646/09: "Tasks solution for large graphical data processing in the environment of parallel, distributed and network computer systems."

9. References

Adams, J. (2003). *Advanced Animation with DirectX*, Premier Press, a division of Course Technology.

Barron, T. (2003). *Strategy Game Programming with DirectX 9.0*, Wordware Publishing Inc.

Berge, C. (1958). *Théorie des graphes et ses applications*, Vol. II of *Collection Universitaire de Mathématiques*, Dunod, Paris 1958 (English edition, Wiley 1961; Methuen & Co, New York 1962; Russian, Moscow 1961; Spanish, Mexico 1962; Roumanian, Bucharest 1969; Chinese, Shanghai 1963; Second printing of the 1962 first English edition. Dover, New York 2001).

Brown, A. & Zwolinsky, M. (1990). Lee router modified for global routing, *CAD* (5): 296–300.

Bučko, M. & Klešč, M. (1999). *Discrete Mathematics, Diskrétna matematika (orig. Slovak title)*, Academic Press elfa, Košice.

Kőnig, D. (1936). *Theorie der Endlichen und Unendlichen Graphen: Kombinatorische Topologie der Streckenkomplexe*, Akad. Verlag, Leipzig.

Kuljić, B., Simon, J. & Szakáll, T. (2009). Pathfinding based on edge detection and infrared distance measuring sensor, *Acta Polytechnica Hungarica* 6(1): 103–116.

Póth, M. (2007). Comparison of convolutional based interpolation techniques in digital image processing, *Proc. of the 5th Int. Symposium on Intelligent Systems and Informatics, SISY 2007*, Subotica, Serbia, pp. 87–90.

Simon, J., Szakáll, T. & Čović, Z. (2006). Programming mobile robots in ANSI C language for PIC MCU's, *Proc. of the 4th Serbian-Hungarian Joint Symposium on Intelligent Systems, SISY 2006*, Subotica, Serbia, pp. 131–137.

Sobota, B., Hrozek, F. & Szabó, Cs. (2010). 3D visualization of urban areas, *Proc. of the 8th Int. Conf. on Emerging eLearning Technologies and Applications, ICETA 2010*, Stará Lesná, Slovakia, pp. 277–280.

Sobota, B., Korečko, Š.. & Perháč, J. (2009). 3D modeling and visualization of historic buildings as cultural heritage preservation, *Proc. of the Tenth Int. Conf. on Informatics, INFORMATICS 2009*, Herlany, Slovakia, pp. 94–98.

Sobota, B., Perháč, J. & Petz, I. (2009). Surface modelling in 3D city information system, *J. of Comp. Sci. and Control Systems* 2(2): 53–56.

Sobota, B., Szabó, Cs., Perháč, J. & Ádám, N. (2008). 3D visualisation for city information system, *Proc. of Int. Conf. on Applied Electrical Engineering and Informatics, AEI 2008*, Athens, Greece, pp. 9–13.

Sobota, B., Szabó, Cs. & Perháč, J. (2008). Using path-finding algorithms of graph theory for route-searching in geographical information systems, *SISY 2008 – Proc. of the 6th International Symposium on Intelligent Systems and Informatics*, Subotica, Serbia, pp. 1–6.

Szabó, Cs. & Sobota, B. (2009). Some aspects of database refactoring for GIS, *Proc. of the Fourth Int. Conf. on Intelligent Computing and Information Systems, ICICIS 2009*, Cairo, Egypt, pp. 352–356.

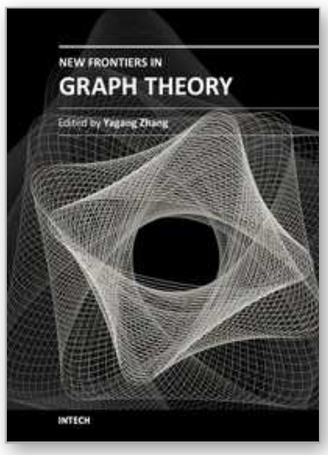
Szabó, Cs., Sobota, B. & Kiš, R. (2010). Terrain LoD in real-time 3D map visualization, *8th Joint Conference on Mathematics and Computer Science, Selected Papers*, J. Selye University, Komárno, Slovakia, pp. 395–402.

Tuček, J. (1998). *Geographical Information Systems, Geografické informační systémy (orig. Czech title)*, Computer Press, Praha.

Vlasova, M. (2008). *3D city information system*, Master's thesis, DCI FEEaI TU, Košice, Slovakia.

Vokorokos, L., Bliš'an, P., Petrík, S. & Ádám, N. (2004). Utilization of parallel computer system for modeling of geological phenomena in GIS, *Metalurgy J.* 43(4): 287–291.

Vokorokos, L., Danková, E. & Ádám, N. (2010). Parallel scene splitting and assigning for fast ray tracing, *Acta Electrotechnica et Informatica* 10(2): 33–37.



New Frontiers in Graph Theory
Edited by Dr. Yagang Zhang

ISBN 978-953-51-0115-4
Hard cover, 526 pages
Publisher InTech
Published online 02, March, 2012
Published in print edition March, 2012

Nowadays, graph theory is an important analysis tool in mathematics and computer science. Because of the inherent simplicity of graph theory, it can be used to model many different physical and abstract systems such as transportation and communication networks, models for business administration, political science, and psychology and so on. The purpose of this book is not only to present the latest state and development tendencies of graph theory, but to bring the reader far enough along the way to enable him to embark on the research problems of his own. Taking into account the large amount of knowledge about graph theory and practice presented in the book, it has two major parts: theoretical researches and applications. The book is also intended for both graduate and postgraduate students in fields such as mathematics, computer science, system sciences, biology, engineering, cybernetics, and social sciences, and as a reference for software professionals and practitioners.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen