International Journal of Artificial Intelligence & Applications (IJAIA), Vol.2, No.4, October 2011

DIRECTION ORIENTED PATHFINDING IN VIDEO GAMES

Xiao Cui[1] and Hao Shi[2]

[1]School of Engineering and Science, Victoria University, Melbourne, Australia
xiao.cui1@live.vu.edu.au
[2]School of Engineering and Science, Victoria University, Melbourne, Australia
hao.shi@vu.edu.au

ABSTRACT

Pathfinding has been one of major research areas in video games for many years. It is a key problem that most video games are confronted with. Search algorithm such as Dijkstra's algorithm and A algorithm are representing only half of the picture. The underlying map representations such as regular grid, visibility graph and navigation mesh also have significant impact on the performance. This paper reviews the current widely used solutions for pathfinding and proposes a new method which is expected to generate a higher quality path using less time and memory than other existing solutions. The deployment of methodology and techniques is described in detail. The aim and significance of the proposed method in future video games is addressed and the conclusion is given at the end.*

KEYWORDS

Pathfinding, A algorithm, Map representation, Optimal path*

1. INTRODUCTION

The video game business is a multi-billion-dollar industry and still growing. In the past, better graphics have been one of the main driving forces of sales; however, this is no longer the case and good graphics alone is not sufficient to fuel sales. Instead, game players are looking for a more realistic gaming experience. Game artificial intelligence (AI) is playing an increasingly important role in the success of games. As a fundamental AI problem, pathfinding has an unneglectable indirect impact on gaming experience. It determines how a game character can move to a desired destination. Such problem has to be solved in real time, often under constraints of limited memory and CPU resources. Thus, a big challenge is how to balance solution quality against required t. This paper proposes a new method called Direction Oriented Pathfinding (DOP) which can provide a high quality solution without sacrificing speed.

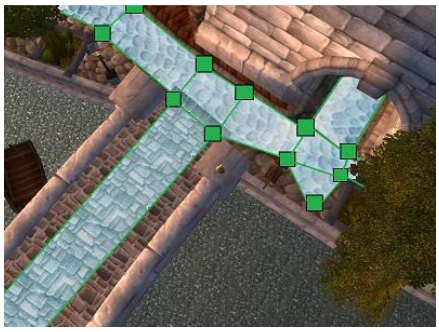
2. BACKGROUND

Pathfinding generally refers to find an optimal path between any two locations. In the context of video games, an optimal path between any two locations is the least cost path rather than the shortest path. That means a path along the road may be better than a shorter path through a hill in terms of time as the cost of climbing a hill is obviously higher than the cost of walking along a road. However, the actual game world as shown in Figure 1a doesn't contain such information. The general solution is to overlay another map with cost information on the actual game world as shown in Figure 1b and adopt a search algorithm on it to find the least cost path.

Current solutions for pathfinding, in the context of video games, either provide a high speed search by sacrificing accuracy or produce an optimal path but using more time and resources [1]. How to find an optimal path more efficiently is still an area of study.

DOI : 10.5121/ijaia.2011.2401

a) Part of Stormwind City in World of WarCraft



b) The same area annotated with a navigation mesh

Figure 1. Pathfinding in video games [2]

2.1. Search Algorithms

Dijkstra's algorithm and A* algorithm are two classic graph search algorithms in geometry, which can find the shortest path between any two nodes on a weighted graph as shown in Figure 2. In weighted graphs, the shortest path exactly equals the least cost path [3]. That means finding an optimal path on a weighted graph equals finding the shortest path on it.

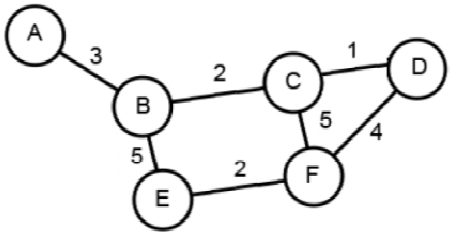
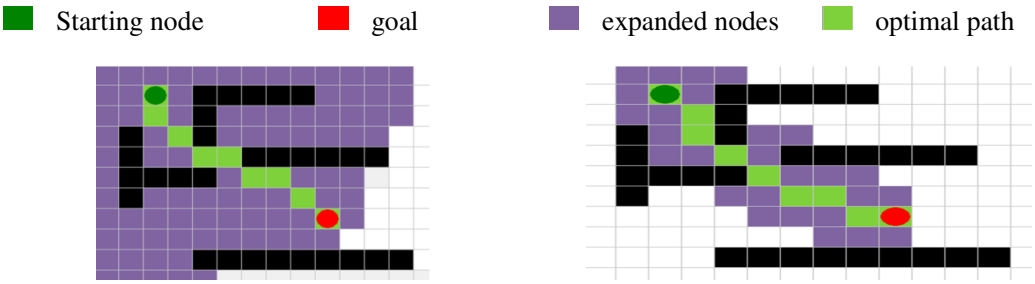


Figure 2. Weighted graph

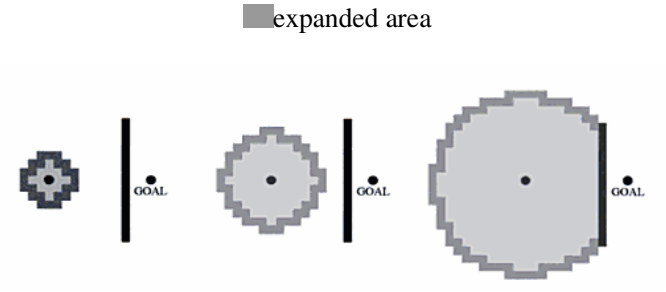
In the past, Dijkstra's algorithm was the only choice for pathfinding until last two decades A* algorithm had become the mainstream solution for pathfinding in video games [4]. Dijkstra's algorithm can find the shortest path from the source node to all the other nodes on the graph and A* algorithm produces the shortest path for a pair of nodes on the graph.



a) Dijkstra's pathfinding b) A* pathfinding

Figure 3. Comparison between Dijkstra's and A*

Unlike Dijkstra's algorithm where an exhaustive search of the entire graph is required as shown in Figure 3a, A* algorithm uses a heuristic method to reduce the research space as shown in Figure 3b. That means only the promising nodes are examined. Figure 4 shows an example of how a heuristic method can affect the search space.



a) The heuristic cost of zero

b) The heuristic cost using Euclidean distance

c) The heuristic cost overestimated

Figure 4. Comparison between different heuristic cost functions [5]

2.2. Map Representation

Neither Dijkstra and A* can work directly with the actual game world. An overlay map with cost information is required. The most widely used map representation in current video games include regular grid, visibility graph and navigation mesh [6]. Different graphs have different impact on the performance of pathfinding in terms of time, memory space and path quality. Normally, large graphs will most likely require more memory to store the nodes and edges and

also result in a slower search. Smaller simpler graphs leave smaller memory footprint but run the risk of not accurately representing the game world.

2.6.1. Find Path on Regular Grid

Figure 5 shows an example of square grid in Civilization IV, which is a well-known turn-based strategy game released in 2005.

Figure 5. Square grid (Civilization IV) [7]

Such a representation has been widely used in many top-down perspective strategy games. The advantage is that it is easy to generate automatically [8]. The limitation is that such a representation often requires a large number of cells to adequately represent the game world. Thus, regular grid was soon overwhelmed by the sheer exponential growth in the complexity of the game world such as a complex 3D world. To find a path between any two cells on a regular grid, search algorithms like A* algorithm can be adopted on it and each cell on a regular grid is a node on a weighted graph.

2.6.2. Find Path on Visibility Graph

A visibility graph consists of a set of waypoints which are placed at the corners of the obstacles, with edges between the waypoints as shown in Figure 6.

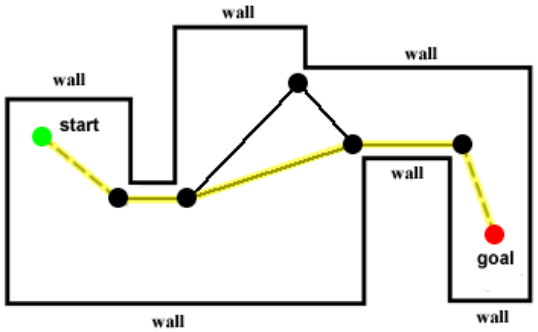


Figure 6. A visibility graph for a first-person shooter game

Although visibility graph overcomes the exponential increase in the number of cells in regular grid when representing a complex game world like 3-dimension (3D) world, the most compelling disadvantage of such a representation is that generating a visibility graph has $O(n^2)$ complexity (n is the number of nodes). It is not the best solution for pathfinding especially in an outdoor area as shown in Figure 7. Visibility graph with A* algorithm is guaranteed to find the shortest path if one exists on the graph [9].

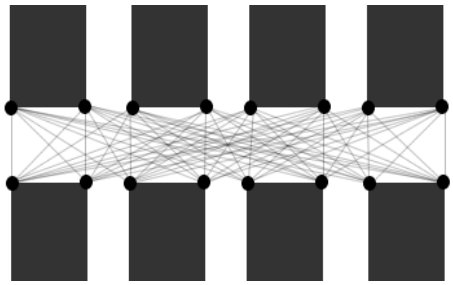


Figure 7. A visibility graph in an open area

2.6.3. Find Path on Visibility Graph

Navigation mesh (NavMesh) is a representation that only covers the walkable surfaces in the game world with convex polygons. Convex polygons guarantee that a character can freely walk from any location within a polygon to any other location with that same polygon. Figure 8 shows an example of using NavMesh to represent the game world. It overcomes the exponential increase in the number of edges in visibility graph when representing an outdoor environment [10]. Thus, one of the advantages of such a representation is that it can handle indoor environments and expansive outdoor areas equally well.

Figure 8. Navigation from A to B on a NavMesh

However, it is not trivial to generate a NavMesh both manually and automatically, especially a triangle-based NavMesh as shown in Figure 9 [9].

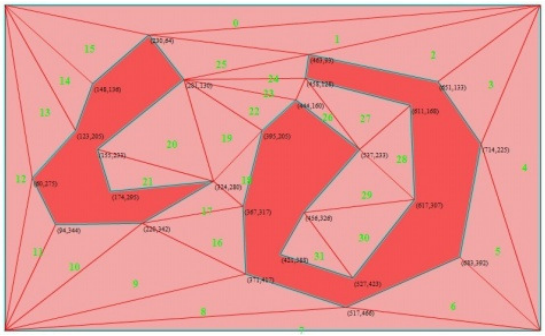


Figure 9. A triangle-based NavMesh

Figure 10 compares the paths found on a NavMesh through the polygon centres, the edge centres and the obstacle corners respectively. However, none of them can provide the shortest path and the paths generated by such methods have many abrupt twists and turns. So, the extra processing is required. String-pulling and funnel algorithm are two widely used techniques to smooth out the path [11].

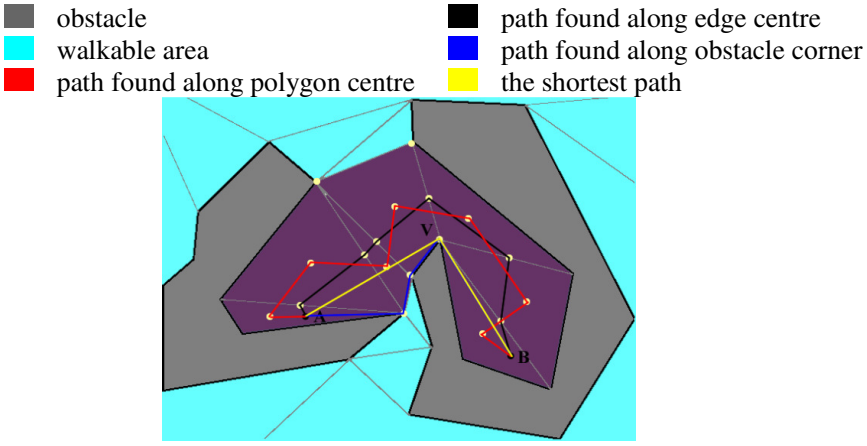
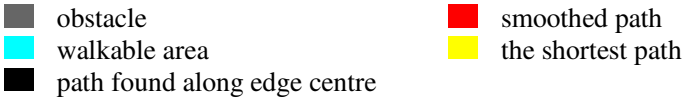


Figure 10. NavMesh pathfinding

Figure 11 shows an example of using string-pulling to smooth out the path. Let P be the path and N be the number of nodes on P. A new path is generated by removing any P_i (P_i is the i^{th} node from the starting node P_1) from P when it is possible to get from P_{i-1} to P_{i+1} directly. A smoother path is given after string-pulling but it still may not be the shortest path. An explanation for the failure to find the shortest path may be that string-pulling as a post-processing technique only optimizes the path itself rather than the way to find the path.



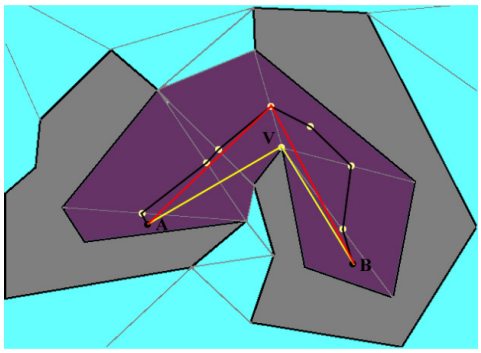
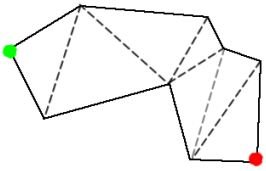


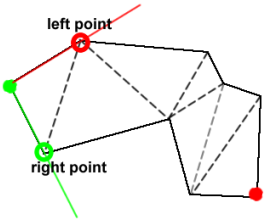
Figure 11. String-pulling

Figure 12 shows an example of using funnel algorithm to find the shortest path on a simple polygon which is divided by a set of triangles. The shortest path can be found within linear time using funnel algorithm. It is a key algorithm for single source shortest path problem in computational geometry and has been integrated into the usual pathfinding approaches. The limitation of such a method is that it only works on a triangle-based NavMesh.

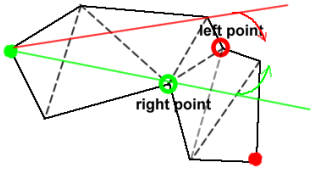
How to find the shortest path on an n-sided-poly-based NavMesh is still worthy of study as such a representation is much easier to auto-generate than triangle-based NavMesh where all polygons must be triangles and it also requires smaller memory footprint than triangle-based NavMesh. Polygons on an n-sided-poly-based NavMesh can have any number of sides as long as remain convex.



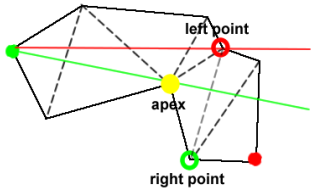
Step 1: divide the polygon into a set of triangles



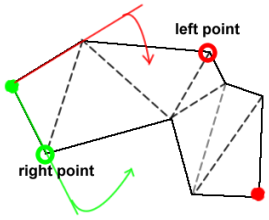
Step 2: find the left and right points



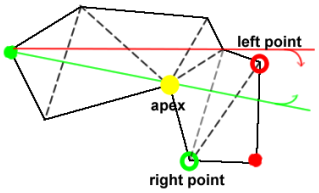
Step 5: repeat step 3 and 4, until either the next right point or the next left point is outside the limits



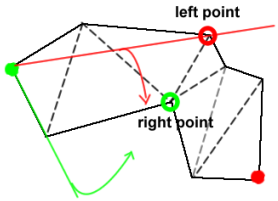
Step 6: if the next right point is outside the right limit, do not update right limit, set the current right point as an apex point



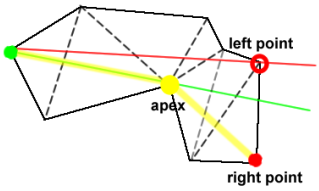
Step 3: if the next left point inside both the left and right limits, update the left limit



Step 7: repeat step 3 and 4, until either the next right point or the next left point is outside the limits



Step 4: if the next right point inside both the left and right limits, update the right limit



Step 8: if the next left point is outside both the left and right limits, terminate the loop and find the shortest path through apex points (or if the next right point is the goal, terminate the loop and find the shortest path through apex points)

Figure 12. Find the shortest path on a set of triangle-based polygons using funnel algorithm

3. AIM AND SIGNIFICANCE

The main objective of this research is to propose a new pathfinding method called Direction Oriented Pathfinding (DOP) which can find the shortest path on an n-sided-poly-based NavMesh more efficiently in terms of running time and space consumption. The specific objectives include:

- Evaluate the efficiency and effectiveness of the proposed method DOP. Show that DOP will be better than existing solutions.

- Adopt the proposed method DOP to real game scenarios and other domains such as robot path planning.

Gaming industry has become the fastest growing and most profitable industry. Global industry sales as a whole are expected to rise to $68 billion in 2012. With the growing demand for a more realistic gameplay experience, game artificial intelligence (AI) is playing an increasingly important role in the success of games. Pathfinding is one of the fundamental AI problems that most games are confronted with. The biggest challenge to game developers is to provide a practical pathfinding solution in real time under the constraints of memory and CPU resources. The computational effort required to find an optimal path, using current solutions, exponentially increases with size and complexity of the map.

This new proposed method for pathfinding Direction Oriented Pathfinding (DOP) will significantly speed up the pathfinding process and save resources like CPU time and memory space by using the new algorithms without the requirement of post-processing and the new map representations which are easier to generate and maintain. The proposed method DOP, finding an optimal path in real time, can make games, especially sports games like car racing games, more challenging and game players immersed in the game.

Besides that, the proposed method DOP can also be adopted to other industry domains such as robot path planning and plays significant role in future video games. For example, a wheeled

robot can be employed to the dangerous areas such as nuclear-contaminated sites and flooding areas, a strategy like DOP telling the robot where to move to reach the destination is essential.

4. METHOD AND EXPERIMENTAL DESIGN

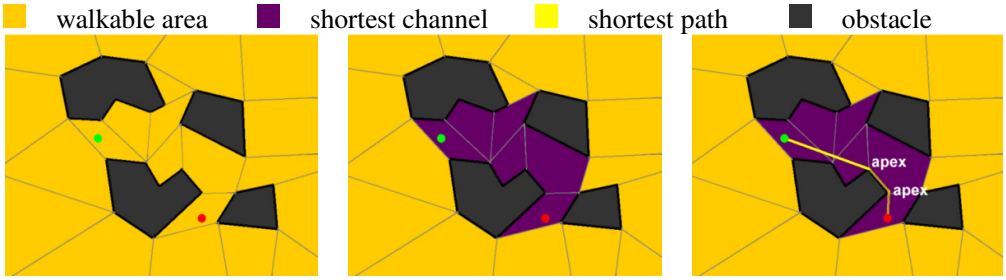
The following methodology has been considered for this research project.

4.1. Using Unity3D as A Testbed

Unity3D is an integrated development environment for 3D games [12]. An existing Unity3D project A* Pathfinding [13], which provides a basic implementation of A* pathfinding, is available for reuse. Algorithms and maps used in A* Pathfinding can be easily tailored to suit the needs of the experiments on DOP.

4.2. Finding Shortest Path on N-sided-poly-based NavMesh

As mentioned above, finding the shortest path on an n-sided-poly-based navigation mesh is still a problem. This research intends to adopt a funnel-like algorithm to n-sided-poly-based navigation mesh pathfinding. The proposed method DOP can be divided in three main steps. First, generate an n-sided-poly-based navigation mesh on the game world as shown in Figure13a. Second, find the shortest channel as shown in Figure 13b. A channel is composed of a set of adjacent polygons. It represents the areas where the shortest path may traverse. A* algorithm can be adopted to find it and each polygon on a NavMesh is a node on a weighted graph. Third, find the shortest path inside the channel as shown in Figure 13c. The key challenge of this research is to overcome the limitation of the original funnel algorithm and develop a new variant which can find the shortest path on n-sided-poly-based navigation mesh.



a) Generate n-sided-poly based NavMesh
b) Find the shortest channel
c) Find the shortest path on the channel

Figure 13. Find the shortest path (least cost path) on n-sided-poly-based NavMesh

4.3. Experiments

To evaluate the proposed method, a set of systematic experiments are required. Experiments will be conducted from two perspectives: efficiency and effectiveness. The efficiency of the proposed method DOP can be assessed by comparing the running time (how many milliseconds need to be took to find a path) and memory usage (how many kilobytes need to be used to store the pathfinding data) against other existing approaches such as A* pathfinding on visibility graph and A* pathfinding on triangle-based NavMesh. The effectiveness can be gained by simulating the proposed method DOP in various game worlds so as to prove that DOP can always find the shortest path if one exists on an n-sided-poly-based NavMesh. Examples of such

experiments testing the effectiveness of the proposed method DOP include indoor environment pathfinding, outdoor environment pathfinding, and random environment pathfinding and so on.

4.4. Implementation on Real Video Game Scenarios

Implementing DOP on real video game scenarios can be a good opportunity to evaluate the proposed method and get advices and criticism from the experience game players. Many commercial game companies release their software develop kit (SDK) to allow game players to rewrite or modify the game to their heart's content by creating custom Mods.
As the most well-known turn-based strategy game, Civilization 5 (Civ5) allows custom Mod to be created by game players to change the map of the game, the rules of the game and the AI behavior of the game. There are two official moddling tools available: ModBuddy and World Builder SDK. Posting a DOP pathfinding Mod on Civ5 can be a good opportunity to collect feedback from game players about the performance of DOP on real game scenarios.

Another famous city-building game SimCity 4 (SC4) also allows game players to create their own Mods to change the game. A Mod called NAM has successfully improved the built-in pathfinding system in SC4 by introducing a more accurate heuristic function [14]. Posting a DOP pathfinding Mod may make an improvement on the built-in pathfinding system from another angle.

5. CONCLUSIONS

This paper systematically reviews three widely used techniques for pathfinding, including A* pathfinding on regular grid, A* pathfinding on visibility graph and A* pathfinding on navigation mesh, and discusses two path smoothing techniques string-pulling and funnel algorithm. Then, a gap is identified and a new method DOP is proposed to fill the gap. This paper also describes how the proposed method will be assessed. The evaluation will be conducted from two perspectives efficiency and effectiveness. And it will also be implemented in real game scenarios by replacing the pathfinding systems in the existing games by DOP. The benefit DOP may bring is that it can make games more challenging and game players immersed in the game more likely through improving the built-in game AI system in the game.

REFERENCES

[1] V.Bultiko, Y.Bjornsson, N.R.Sturtevant and R.Lawrence, "Real-time Heuristic Search for Pathfinding in Video Games", Artificial Intelligence for Computer Games, Springer, 2011

[2] Game/AI, "Fixing pathfinding once and for all", http://www.ai-blog.net/archives/000152.html

[3] T.H. Cormen, C.E.Leiserson, R.L.Rivest and C.Stein, *Inroduction to Algorithms*, The MIT Press, 2001.

[4] I. Millington and J.Funge, *Artificial Intelligence for Games*, Morgan Kaufmann, 2009.

[5] S. Rabin, "A* speed optimizations", *Game Programming GEMS*, pp.264-271, Charles River Media, America, 2000.

[6] P. Tozour, "Search space representation", *AI Game Programming Wisdom 2*, pp.85-113, Charles River Media, America, 2003.

[7] Firaxis Games, "Sid Meier's Civilization V", http://www.civilization5.com , accessed October 24, 2010.

[8] Y. Bjornsson, M. Enzenberger, R. Holte, J. Schaejfer, and P. Yap, "Comparison of different grid abstractions for pathfinding on maps", In Proceedings of the 18th International Joint Conference on Artificial Intelligence, San Francisco, pp.1511-1512, 2003.

[9] K. Yu, "Finding a natural-looking path by using generalized visibility graphs", In Proceedings of the 9th Pacific Rim International Conference on Artificial Intelligence, Berlin, pp.170-179, 2006

[10] G. Snook, "Simplified 3D movement and pathfinding using navigation meshes", *Game Programming GEMS*, pp.288-304, Charles River Media, America, 2000

[11] D. Demyen and M. Buro, "Efficient triangulation-based pathfinding", In Proceedings of the 21st National Conference on Artificial Intelligence, Boston, pp.942-947, 2006

[12] Unity, "What's new in 3.0", http://unity3d.com/unity/whats-new/unity-3

[13] Aron Granberg, "A* Pathfinding Project", http://www.arongranberg.com/unity/a-pathfinding

[14] Sim City 4 Devotion Forums, "NAM Unified Traffic Simulator and Data View Help", http://sc4devotion.com/forums/index.php?topic=6812.0

Authors

Xiao Cui is a Ph.D student in School of Enignnering and Science at Victoria University, Australia. He completed his master degree in the area of Software Engineering at Australian National University and obtained his Bachelor of Computer Science degree at Victoria University. His research interests include object-oriented software engineering, pathfinding algorithms, database management and game programming.

Hao Shi is an Associate Professor in School of Engineering and Science at Victoria University, Australia. She completed her PhD in the area of Computer Engineering at University of Wollongong and obtained her Bachelor of Engineering degree at Shanghai Jiao Tong University, China. She has been actively engaged in R&D and external consultancy activities. Her research interests include p2p Network, Location-Based Services, Web Services, Computer/Robotics Vision, Visual Communications, Internet and Multimedia Technologies.