Multiple-Level Grid Algorithm for Getting 2D Road Map in 3D Virtual Scene

Jiangchun Wang[1], Shensheng Zhang[1], and Jianqiang Luo[1]

[1]CIT Lab, Department of Computer Science and Engineering
Shanghai Jiao Tong University, P.R. China
scott751111@sjtu.edu.cn

Abstract. It is a common requirement to build a 2D map of a 3D virtual scene for path planning, but it is difficult or tedious. So we propose a multiple-level grid algorithm to solve the problem. The original idea lies in the new method on distinguishing a grid unit's accessibility. We argue for an approach by employing two navigating lines, which are right-angle intersection and stand for the unit's accessibility in four directions. Moreover, another advantage of it is to dynamically divide the virtual scene into different sizes of grids at different precisions. With these multi-precision grids, we can approach objects at any granularity. Lastly, experiments are performed to test the effectiveness of the algorithm.

1 Introduction

Recent advances in graphic engines and software tools have facilitated the development of visual interfaces based on 3D virtual environments (VEs) [1]. These interfaces use interactive 3D graphics to represent visual and spatial information and allow natural interaction with direct object manipulation, such as VRML (Virtual Reality Modeling Language) [2]. People can get a lot of shared resource from the Internet to construct their own virtual world. Moreover, when a virtual scenic file is acquired, they usually also need assistant information, such as the scenic 2D map, for imbedding and navigating our own virtual objects. Unfortunately, it happens rarely. A solution probably uses the projection image, the screen snapshot, from top view in the virtual environment browser. But this simple way comes with so much misunderstanding that it is not reliable. For example, a room, which may be entered in, is displayed as impassable object in the top view projection image. To acquire a real 2D road map, this paper proposes a Multiple-level Grid Algorithm.

2 Related Work

Let us briefly talk about some previous work on this problem. To draw virtual scenic map is similar to drawing real world map. Many researches had been completed on how to get these practical maps [3,4,5]. The aerial survey mapping is efficient way for large area survey like the top view projection method in our virtual environment.

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2659, pp. 264–274, 2003.

Many advantages had been presented in document [6]. But this method is unsuitable for our problem, for so much redundant information acquired probably. For example, an image of the top view projection would show some area as road area because of no objects here. But in fact, these areas maybe are embraced by other obstacles and can never be arrived at. We only want to know where can be reached in a virtual scene. The superabundance can only mislead us. To ransack a scene to find a road map belongs to the traversal problem. Many traversal algorithms had been proposed in the graph theory. Document [7] is a good reading material for this problem. Koucky proposed a universal traversal approach [8]. In some documents [9,10], this problem is called 'terrain acquisition'. Gonzalez had proposed a complementary regions algorithm for such a surface filling [11]. But their approaches are limited at precision for relying on the size of a gauge, which may be a mobile robot or a block. Our problem has partial similarity with the all of above, and its special attributes are described in next section.

Our method is based on the simulation of user's behavior. When a user is navigating in a virtual scene from a browser, such as a VRML browser, the interactive device is commonly a keyboard. The OS receives user input and only sends some keyboard messages to the browser. For example, Up-key message stands for user's idea of 'advancing', Left-key message means 'turning left', and so on. Therefore, we can substitute for OS to send same messages to the browser to simulate user behavior. Our program simulates a user movement in the virtual scene. With the help of embedded JAVA applet in the VRML file, the map recorder can receive the motion data of the user's viewpoint. Figure 1 illustrates this architecture. So the trail of user's viewpoint stands for the passage. How to compose a map with these points is our kernel problem. Because we have little interest in the difference of the motion trail on altitude, such difference is neglected and planar data are only kept, such as recorded x, z coordinates and neglected y (means height in the VRML).

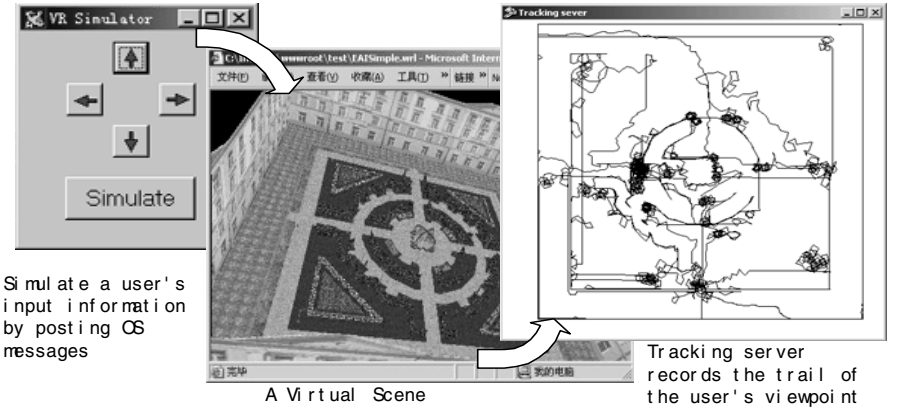

Fig. 1. The way for getting the accessible data

3 Virtual Terrain Acquisition Problem

In this section, we present the virtual terrain acquisition problem of the map in the virtual scene. In a 3D virtual scene, we are asked for to get out its 2D planar path map. Our problem is different from other path-plan problems [12,13], because the scenic environment is almost unknown and we need not only get out where are obstacles or roads, but must cover all position in it. The scenic information is very little except the scenic scope. While our avatar is walking, the data of viewpoint position can be acquired. The data don't stand for a set of small regions, but a set of sample points, which are infinitesimal. Due to the difficulty of walking on every point in the scenic map, it is impossible to use exhaustive approach to distinguish an obstacle or a road. So we propose a multiple-level grid algorithm to deal with it.

3.1 Problem Formulation

We assume that if a point x is of a road, its *neighborhood* $\Gamma(x)$, which is tightly close to it, is also of the road. Firstly, we divide the map into many small rectangles or blocks, which are of the same size. Figure 2 illustrates that.

Before we formulate the problem, let us define the following notation:

- $n =$ Number of small rectangle in a dividing grid.
- $a_i =$ The ith small rectangle in a dividing grid where $i = 1,2,...,n$.

If the n is big enough, in other words, there are enough many small rectangles divided from a scenic map, we can think the rectangle region as a road or an obstacle on the condition that whether or not a viewpoint can step into it from its adjacent rectangles. Here we define the neighborhood $\Gamma(x)$ of x, any point in the scenic map, as the small rectangle a_i:

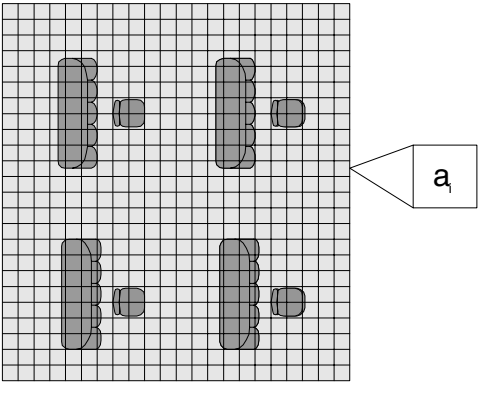

Fig. 2. The dividing grid for a map

If $x \in a_i, i = 1,..,n(n - \propto)$ and x of a road, then a_i is of a road, and vice versa. There are 4 edges in a_i — Up, Down, Left and Right. We statistically know if

a user can steps into a small rectangle a_i in these four directions along with the mid-points of every edge, then, a_i is almost of a road. So we simplify the determinant with a definite precision. Figure 3 illustrates some samples to tell which rectangle region is a road or an obstacle.

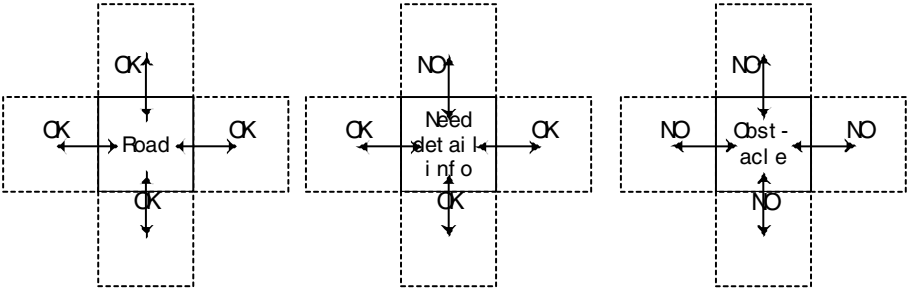

Fig. 3. Samples for describing the rectangle property of accessibility

Then, we use a graph notation to present a scenic map. Given a graph $G = (V, E)$, V represents the set of small rectangles in the dividing grid; E represents the set of edges such that an edge $e_{i,j} \in E$ represents that the small rectangle a_i is next to a_j. Because the number of small rectangles adjacent to a_i is 4 at most, we can construct such a graph like Fig. 4. The problem is converted to solve the accessibility of every edge in graph G.

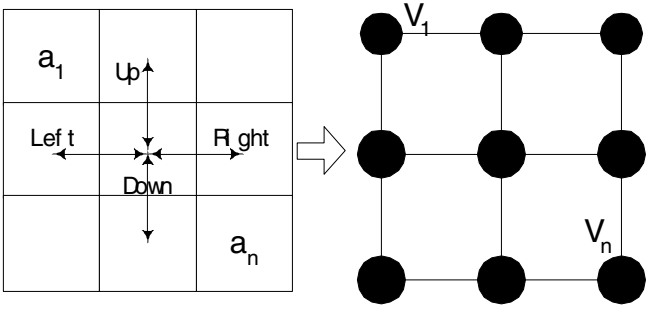

Fig. 4. Transferring from a grid to a graph

3.2 Random Walk Approach

One way to work out every small rectangles state is by random walk approach, that is, given n rectangles of the whole grid, which is a n-vertices graph, and then the viewpoint can start at any point and jump to any edge at every node. Let $|n|$ denote the

total linked edges for detecting every node; thus, here is 4^n. If scenic scope is 100*100 and the required precision is 0.1, so $n = 10^6$, the random walk approach impossibly reach our anticipation.

3.3 Multiple-Level Grid Algorithm

On the above, we propose the multiple-level grid algorithm (MLG) to solve this virtual terrain acquisition problem.

We need define some data packets:

- Edge=(V_i, Vj, Passed) An *Edge* is a triple, where V_i, $Vj \in V$ and *Passed* stands for whether we can enter in the rectangle a_j, which the vertex V_j correspond to, from a_i (which the vertex V_i correspond to). The optional value of *Passed* is among *'T'*(true)□*'F'* (false) and *'U'*(untouched). Because the road is two directions, if *x=(V_i,V_j,Passed)* and *y=(V_i,V_j,Passed)*, then the *Passed(X) = Passed(Y)*.
- Node=(V_i,Front_Edge,Back_Edge,Left_Edge, Right_Edge ,Status) An *Node* includes seven elements, where $V_i \in V$; *Front_Edg*, *Back_Edge, Left_Edge, Right_Edge* are the type of *Edge*. *Front_Edge* stands for passing-through nature between V_i and its adjacent vertex in the up direction, and *Down_Edge* is for the down direction, the third for left, the fourth for the right; the *Status* is for whether the rectangle is region of a road or an obstacle, which value is among *'R'*(road), *'O'*(obstacle), *'NFI'*(need detail information), or *'U'*(undone).

We also need define some functions:

- *Passed(Edge)* It get result from a *Edge* to tell if the *Edge* can be pass through.
- *Status(Node)* It get result from a *Node* to tell whether the rectangle is of a road, an obstacle or other conditions.

Multiple-Level Grid Algorithm:

1. begin

2. Define the max value *R* for the resolution and initial resolution $r_0 < R$, and iteration K = 1;
3. Create a global stack M to contain the need-checking region;
4. Let $r = 1, i = 0$ and push the whole scenic map as a graph, denoted by A_0, and a start point s_0 into stack M;
5. **while** ($r < R$){
6. $i = i + 1, r = r \cdot r_0, j = 0$;
7. **while** (Stack M is not empty){
8. Get out a graph A_j from the stack M;
9. Use the *grid-covering* algorithm to deal with A_j at the resolution r and start point s_i, then work out the status for every node of the graph.;

10. $j = j + 1$;}
11. $j = 0$;
12. **For** every node of all graphs {
Uses the *adjacent-edge rule* to check them.;
If a node status is NFI **then**
{ $j = j + 1$, mark it with A_j and push it and a point s_i ($s_i \in A_i$) into stack M;}
} }
13. Integrate all nodes of every level to compose a whole road map *A* for the scene;
14. Scan all divided smallest rectangles in A to find a new start point s_K.
{ **If** found(s) **then**
let $s_0 = s_K$ and K=K+1, goes to step 4
Else
final road map A is result;
/* The smallest rectangle must be on border including the new start point. */ }
15. **end**

The *multiple-level grid algorithm* is iterative, the complexity in an iteration cycle is $O(n)$. The *algorithm* has two key components, namely, 1) the *grid-covering algorithm* (GC), and 2) the *adjacent edge rule* (AER). The GC algorithm uses a dived-and-conquer approach to filter out road blocks and obstacle blocks. The AER is based on the experiential assumption. In what follows, we describe each of these two components.

3.3.1 Grid Covering (GC) Algorithm

The main idea of the grid-covering algorithm is to divide the scene of a virtual environment into grid array $A(a_j \in A, j = 1,2,..., n)$ and then to construct the corresponding graph $G = (V, E)$. In the grid- covering algorithm, we first set the resolution r and a start position s_0, which are inputted from outside, for this level grid.

Grid Covering Algorithm:
1. begin
2. According to the scenic size and the resolution r, divide the scene with a set of small rectangles $a_i (i = 1,2,..., n)$;
3. According to a_i, construct a set of vertexes to compose an array, denoted by *V*, and create a new array of *Node*, denoted by M. The number of elements in V is n, and same to array M. V_i of $M[p]$ $(p = 1,2...,n)$ is belong to V;
4. Set initial value for array M;
/*Set *Status* of $M[p](p = 1,2...,n)$ with '*U*';*/
5. From start vertex V_0 (It's a seed), which includes the start position s_0. Let $V_{curr} = V_0$;
6. From *Front_Edge* to *Right_Edge* of V_{curr}, check its *Edge* element *X*(*X*∈ *Front_Edge*, *Back_Edge,Left_Edge,Right_Edge*).

If *Passed*(*X*)=*U* then goes to step 6, or goes to step 8;
7. Check user can enter in V_{curr} 's neighbor V_{next} through the *Edge X*. If can not, set variable *Passed* of *Edge X* and V_{next} 's corresponding *Edge* element with value '*F*', and goes to step 5; set variable *Passed* of *Edge X* and V_{next} 's corresponding *Edge* element with value '*T*';
8. Let $V_{curr} = V_{next}$, goes to step 4;
9. If *Edge X* is *Right_Edge*, backwards to the previous *Node* V_{pre} . Let $V_{curr} = V_{pre}$ and goes to step 9, else goes to step 4;
10. If $V_{curr} = V_0$, goes to step 10, or goes to step 4;
11. **end**.

The purpose of the grid-covering algorithm is to get all real connections of every block in the grid. In the view of graph, we want to produce a connected graph so that we can perform the checking step by the adjacent-edge rule. This algorithm can check every node, which stands for the accessible area.

3.3.2 Adjacent-Edge Rule

The idea of the *adjacent-edge rule* is described as follows:

In the view of graph, if a node V_i can be reached from all the $e_{i,j} \in E$, we think that it stands for a road area; if just from some $e_{i,j} \in E$, we think it need-checking with detail information; if from none, it must be an obstacle area. We name the method as an adjacent-edge rule. Figure 5 illustrates these instances.

The function *Status(Node)* is based on above. After the implementation of GC algorithm, we set nodes state obeyed the *adjacent-edge rule*.

3.3.3 The Complexity of the Multiple-Level Grid Algorithm

The *multiple-level grid algorithm* is a recursive approach based on the GC and AER. GC algorithm makes a viewpoint move in the scenic graph along with a definite route. The most time cost exists in walking away from all edges. The dropping back step needs reset the viewpoint's position, which also shares cost, and then sometimes the viewpoint need backward many nodes to find a new acceptable edge. For discussing the complexity of our algorithm, we define these notations as following:

- $N_k = p_k \times q_k$ This means in the k th level grid, the whole region is divided into p_k rows and q_k columns of rectangles.
- α_k = The average cost of passing through an edge between two nodes in the k th level grid.
- φ_k = The average cost of the dropping back step in the k th level grid.

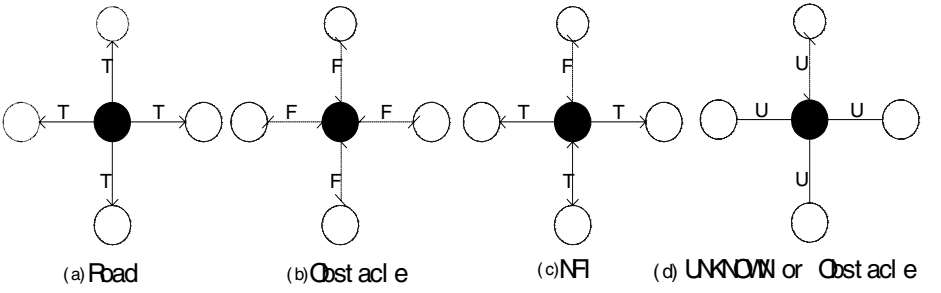

Fig. 5. (a) That all connections of a node are 'T' means it is a road area. (b) That no connections of a node are 'T' means it is an obstacle area. (c) That there are different statuses among connections of a node means it is NFI (need further information for checking). (d) That all connections of a node are 'U'(untouched) means that it is isolated as an obstacle in the smallest grid, or is unknown in other level grids

Lemma 1. *The complexity of Grid Covering Algorithm is* $O(N)$.

Proof. In the *i*th level grid, the all edges is $4N_k - 2p_k - 2q_k + 12$. When a collision happens, we need reset viewpoint to an acceptable edge. The number of reset times is same as the number of edges at most. The whole cost is based on the following formula:

$$C_k = (a_k + \varphi_k) \times (4N_k - 2p_k - 2q_k + 12) \tag{1}$$

Therefore, the complexity of Grid Covering Algorithm is $O(N_k)$.

GC algorithm is recursively used in the *multiple-level grid algorithm*. Therefore, the complexity of the *multiple-level grid algorithm* is $O(\prod N_k)$ at most. As we known,

$$\prod N_k = n \tag{2}$$

So the complexity of an iteration cycle in the *multiple-level grid algorithm* is $O(n)$; and in the worst case, the complexity is $O(n^2)$, which needs n iteration cycles. If we only check the untouched Edges in every iteration cycle with the help of status marks, the complexity will be decreased to $O(n)$. In fact, because of many similar areas, only a few rectangles need to be divided into small enough pieces. The convergence speed of our algorithm is higher than theoretic one.

4 Experiments

In this section, we select the VRML 97 and Cortona VRML Client 4.0 [14] on HP workstation x4000 as our test-bed. To evaluate the algorithm discussed in the previous section, we apply it to the different scales of virtual environment. For the small virtual scene, since the scenic space is manageable, we can give the comparison of the behavior of our algorithm with a random approach. In the large virtual scene, we compare the convergence speed of it with different number and size of obstacles.

4.1 Experiment 1: Small Virtual Scene

In this experiment, we use a small virtual scene with a dimension 1*1. The precision is defined as 0.1, and the obstacles are five boxes. Because the random walk approach is not convergent, we only describe the access percentage of the road area. The curve of random approach is very rough and its searching speed is slow because of a lot of repetitive access. By contrast, our MLG algorithm accesses every site faster in the virtual scene, and the curve is smoother. From the two trend curves, we found that our algorithm is so stable and efficient. Figure 7 list the different effect of our algorithm and random walk approach. The curve of the random approach is convex. With time elapsing, the curve is approximate to the end line (at 100%) infinitely. The approach works out the data of road area fast at the beginning, but later it is running more and more slowly. The curve of our MLG algorithm is concave and convergence. It arrives at the end line stably.

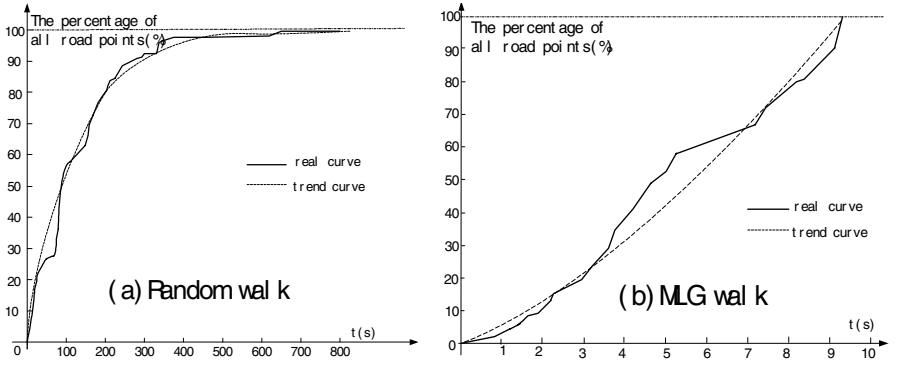

Fig. 6. The comparison of walk effects between two approaches in a small virtual scene

4.2 Experiment 2: Large Virtual Scene

Here, we use a large virtual scene with a dimension 100*100. The precision is also defined as 0.1. We vary the number and size of obstacles to compare the behavior of the multiple-level grid algorithm. In Fig. 7, we apply MLG as two levels of grid to a virtual scene. The horizontal axis stands for different ratio, which is 10^{-4}-10^{4}, between n_1 and n_2. (n_1 is the number of rectangles in the first level grid; n_2 is the number of rectangles in the first level grid.) The left vertical axis stands for its convergence time and the right vertical axis stands for accuracy ratio. The accuracy curve fluctuates in a limited range (96-100%). By contrast, the convergence-time curve changes much more (219-1250s). We recommend that a suitable r be in [10,100].

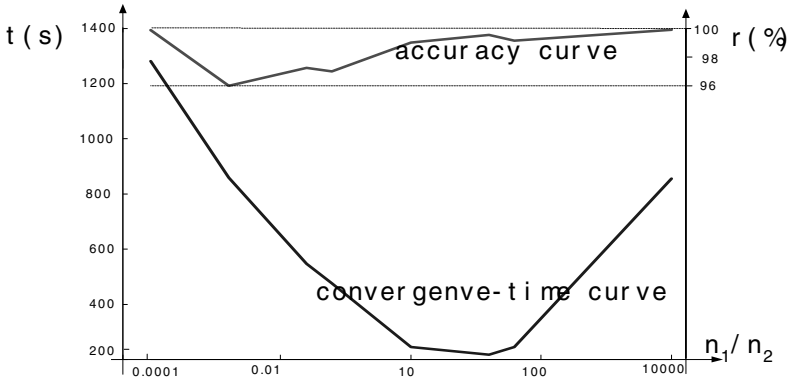

Fig. 7. The behavior of MLG algorithm in different large virtual scene

5 Conclusion and Future Work

This paper deals with the problem extracting the 2D planar map from 3D virtual scene to facilitate using the shared source of the virtual reality scene on Internet. An efficient algorithm is presented and experiments are carried out to testify its efficiency. We also adapt the algorithm to parallel domain. The algorithm is good at the convergence and reliable to versatile scenes with the acceptable recognition accuracy.

But our method only thinks about the same ground plane without caring for the little difference on the altitude, and we get the rough data of a map. Further researches on this project are how to composite a useful map with them and how to navigate the intelligent robot in a virtual scene with the map. The current research is good test-bed for robots navigate approach.

Acknowledgements. This work was supported by China National Science Foundation under grant No: 59789502, and by the National High Technology Plan 863/ CIMS under the grant No: 863-511-030-007-9.

References

1. Tiziana Catarci, Thomas, “Using 3D and Ancillary Media to Train Construction Workers”, Multimedia at Work, April 2002, pp. 88-92
2. `http://www.cs.nps.navy.mil/people/faculty/capps/4473/projects/VRML`
3. Illert, A. “Automatic Digitization of Large Scale Maps”, Technical Papers, ACSM ASPRS Annual Convention, 1991 (6)
4. Datta, A. and Parui, S.K., “A Robust Parallel thinning Algorithm for Binary Images”. Pattern Recognition, 1994, Vol. 27, No. 9,
5. Krakiwsky, Edward J., Mueller, Ivan I., “Toward world surveying and mapping education”. Report on the XIIIth North American surveying and mapping teachers conference, ACSM-ASPRS Annual Convention, Vol. 2., 1991, pp. 160–169

6. Mayer, Helmut, "Automatic object extraction from aerial imagery – a survey focusing on buildings", Computer Vision and Image Understanding, 1999 Acad Press Inc., pp. 138–149.
7. Aldous, D. and Fill, J.A., "Reversible Markov Chains and Random Walks on Graphs", http://www.stat.berkeley.edu/ users/aldous/book.html
8. Koucky, M., "Universal traversal sequences with backtracking", Computational Complexity, 16th Annual IEEE Conference, 2001, pp. 21–27.
9. Lurnelsky, V.J. and Mukhopadhyay, S., "Dynamic Path Planning on Sensor-Based Terrain Acquisition", IEEE Trans. on Robotics and Automation, Vol. 6, No 4,1990, pp. 462–472.
10. Cao, Z.L., Huang, Y., and Hall E.L., "Region filling operations with random obstacle avoidance for mobile robots", Journal of Robotics Systems, Vol. 5, No. 2,1988, pp. 87–102.
11. Gonzalez, E., Suarez, A., Moreno, C., and Artigue, F., "Complementary regions: a surface filling algorithm", IEEE International Conference, Vol. 1, 1996, pp. 909 –914
12. Alexopoulos, C, Griffin, P.M. "Path planning for a mobile robot". IEEE Trans. on System, Man, and Cybernetics, 1992, 22(2), pp. 318–322
13. Beom, H.B. "A sensor-based navigation for a mobile robot using fuzzy logic and reinforcement learning". IEEE Trans on SMC. 1995, 25(3), pp. 464–477.
14. http://www.parallelgraphics.com