

3D Maze Wall Through

Guohua Hao

haog@eecs.oregonstate.edu

<http://web.engr.orst.edu/~haog/cs550/>

1. Project Description:

In this project, I implemented a single level 3D maze navigation project, which is a simple instance of 3D wall through problem. A single level maze is generated randomly according to the user's specification of the number of rows and columns in the maze. The starting point is always at the bottom left corner and the ending point is always at the upper right corner. Once the program starts, the user is placed at the starting point, surrounded by the walls. Then the user can navigate the maze by turning to the left, to the right and going forward. But the user is not allowed to walk through an existing wall in the maze. And it can not climb up the walls either. The user has a headlight on his helmet. But due to the fog, he can not see very far away. He has to go close to find out if there is a wall ahead or not. If the user gets lost in the maze, he can take a look at the top view of the whole maze. There is a red point at the starting point and a green point at the ending point in the top view. At the ending point there is teapot. So the user knows where the ending point is. Once the user reaches the cell containing the teapot, he wins.

2. Implementing details

2.1. Maze generation algorithm

We use the Depth First Search (DFS) algorithm to generate the maze with a give number of rows and columns. Each cell in the maze has four walls and the cell is set to be unvisited at the beginning. Randomly choose a cell to start with and use the following algorithm recursively:

Mark the current cell as visited;

If the current cell has any neighbors that are not visited

 Choose one of them

 Knock down the wall between the chosen node and the current node

 Mark the chosen cell as visited

 Call this function on the chosen cell

This algorithm can guarantee that there is always a path between any two cells although the maze it generates is relatively simple.

2.2. Texture mapping and display list

We use texture mapping to both the walls and the floor. The texture images are in *.bmp format with resolution of 256*256. The function used to read the texture images into buffers are adapted from the one provide by Prof. Mike Bailey in CS 519 course. One texture object is generated for each texture to speed up the performance. Display lists are also used to show the top view in orthogonal view and the inside view of the maze, i.e. walls and the floor in perspective view. The teapot is shown by calling `glutSolidTeapot()` in the display list.

2.3. Fogging

To simulate the effect that the user can not see far away in the maze, fogging technique is used. Fogging is enabled by `glEnable(GL_FOG)` and disabled by `glDisable(GL_FOG)`. The exponential mode is chosen with density 0.30 and fog color of black.

2.4. Lighting

Implementing a light source above the head of the user pointed in the direction of viewing. The parameters are as follows:

```
float light_diffuse[] = {1.0, 1.0, 1.0, 1.0};
float light_ambient[] = {0.7, 0.7, 0.7, 1.0};
float light_specular[] = {1.0, 1.0, 1.0, 1.0};
float light_pos[] = { 0, 0, 0, 1};
```

and the material properties of the teapot are as follows:

```
float teapot_ambient[] = { 0.3, 0.3, 0.1, 1.0};
float teapot_diffuse[] = {0.27, 0.51, 0.70, 1};
float teapot_specular[] = {0.27, 0.51, 0.70, 1};
float teapot_shine = 100.0;
```

The light source moves with the user during the navigation. So it has to be transformed correctly.

2.5. Walk through detection

Here we use a simple walk through detection algorithm. If the line segment from the previous position to the current position intersects with some walls, then the user does not move. Otherwise the user moves to the current position.

3. Usage summery

3.1. specify the maze:

Input two integers at the start of the program, which are the number of rows and columns in the maze

3.2. navigation:

✧ left arrow key to turn left by 30 degrees

- ◇ right arrow key to turn right by 30 degrees
 - ◇ up arrow key to go forward
- 3.3. view changing:
- ◇ 't' to change to the top view;
 - ◇ 'p' to change to the inside perspective view
 - ◇ 'q' to quit the program
- 3.4. others
- ◇ The console window outputs the current location of the user and the index of the row and column of the current cell.
 - ◇ When the user reaches the ending point, console window outputs 'WIN'

4. Further discussion

For simplicity, I do not consider the thickness of the wall, since it makes the wall through detection algorithm more complicated. More complex maze generation algorithms can also be used to generate more interesting mazes. We can also introduce some score to measure how fast the user finds the exit and give a penalty if user switches to the top view of the maze.

Reference:

- [1] <http://www.siteexperts.com/tips/functions/ts20/page3.asp>
- [2] <http://www.cse.cuhk.edu.hk/~csc3550/gallery/2001/>
- [3] Interactive Computer Graphics: A Top Down Approach Using OpenGL by Edward Angel
- [4] OpenGL Programming Guide by Woo, Neider, Davis and Schreiner The Art of 3-D Computer Animation and Imaging, Second Edition by Isaac Victor Kerlow