Alon S. Levin

Final Project: 3-D Labyrinth

Computer Graphics, Spring 2019

Abstract

For this project, I chose to implement a 3D maze, with the labyrinth spanning all three axes. A multi-floored maze is generated randomly based on a "Growing Tree" algorithm[1], and it is the goal of the user to find the way from the top floor to the exit on the bottom, indicated by a light blue wall. The user is at all times provided with his current room's coordinates on the current floor, the current floor number, and the compass direction the user is currently facing. As well, the user has access to a pop-up map of the current floor; depending on the difficulty setting, either the whole map is shown or only the "visited" rooms are.

Operating Instructions

This application is hosted at https://ee.cooper.edu/~levin2/3D%20Maze/, and works on all four web browsers I tested it on (Chrome, Firefox, Internet Explorer, and Microsoft Edge). Movement is controlled by the arrow keys; all other functions are performed via the buttons on the page.

Maze Generation

The maze was generated using the "Growing Tree" algorithm, modified to branch out above and below the current floor to create a multiple floor maze. This algorithm can be summarized as follows:

1. Generate an empty list of cells. Generate a "seed" cell within the list, with desired features listed.
2. Choose a random cell in the list. Choose a random direction that has not been dug into and create a new cell at that position; add this new cell to the list. If all directions have been dug into from the current cell, remove the current cell from the list.
3. Repeat step 2 until the list is empty.

Additional logic is required to ensure that, when digging up or down, the slope (1:3) being generated does not run into any existing cells on either floor or go out of the bounds of the maze.

Each cell is stored as an integer within a 3-dimensional array titled "Maze". This method makes reading cell information very efficient, as the individual bits can be accessed and used as flags for each parameter of interest. The mapping of bits to flags is as follows:

```
0:    East opening
1:    North opening
2:    West opening
3:    South opening
4:    Vertical offset position, B0
5:    Vertical offset position, B1
6:    Direction of downwards facing slope, B0
7:    Direction of downwards facing slope, B1
```

[1] http://weblog.jamisbuck.org/2011/1/27/maze-generation-growing-tree-algorithm

```
8:   Bottom landing of slope
9-15: Unused
```

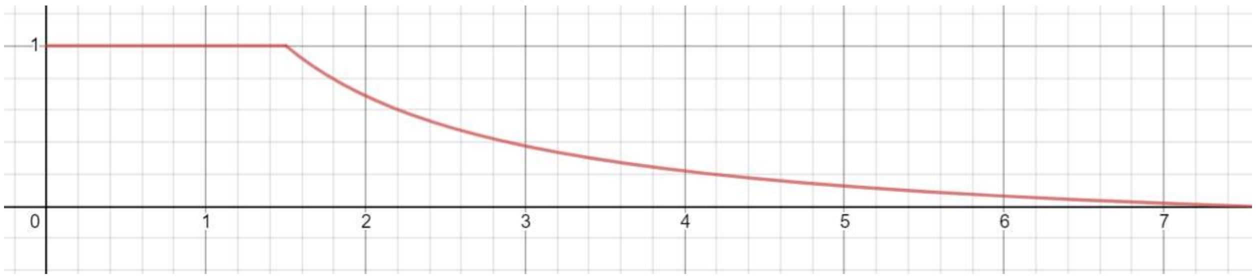
Such data storage allows for simple bitwise access and manipulation of all information regarding how a cell interacts with its neighbors; this is instrumental in the creation of coordinates and identifying wall, ceiling, and floor structures for coloring and mapping purposes.

Coloring is performed by means of randomly assigning colors to walls, ceilings, and floors of sections of the maze. Ten color options for each of these three basic structures are randomly generated, and likewise randomly chosen for different sections of the labyrinth.

Lighting

Lighting is performed as function of distance to the location of the observer. Once the original colors of the surfaces are mapped, the distance from the observer to each fragment is calculated, and a multiplier is calculated using the following relationship:

$$m = \begin{cases} 1.0, x \in [0, 1.5] \\ \dfrac{\frac{1.5}{x} - 0.2}{0.8}, x \in [1.5, \infty) \end{cases}$$



This creates a gradient on surfaces from the perspective of the user, with fragments further from the user appearing much darker than fragments closer to the user.

Maze Traversal

The user moves through the maze using the arrow keys, with up & down corresponding to taking a step of 0.1 units forward & back and left & right corresponding to turning 15^{o} counterclockwise & clockwise. Object collision is performed by detecting whether the observer's x- and y- coordinates are within a threshold (0.15 units) of a wall after each step; if this is the case, the user's position is moved to the threshold limit before rendering takes place for a smooth transition.

One thing that should be noted is that the application renders only when an arrow key is pressed, as that is the only time when there is any change in any of the variables in the application. This was done in order to optimize the runtime of the program, removing any unnecessary renderings and keeping the screen static after each motion by the observer until the next.

[1] http://weblog.jamisbuck.org/2011/1/27/maze-generation-growing-tree-algorithm

Alon S. Levin
Final Project: 3-D Labyrinth
Computer Graphics, Spring 2019

Additional Features

The location relative to the maze and the orientation (represented as a compass direction) of the observer is updated at every render. The pop-up map, which reflects either the entire current floor or only the cells visited by the user depending on the user's choice, can be access by means of a button on the page, at the cost of blocking the observer's view and preventing motion while the map is being viewed.

The maze can be exported as a JSON file, in which all maze variables (sans the color scheme) and observer variables are stored. This JSON file can be loaded into the program to continue at the same exact point that the file was generated, with observer position and orientations preserved.

[1] http://weblog.jamisbuck.org/2011/1/27/maze-generation-growing-tree-algorithm