



TRIBHUVAN UNIVERSITY

Institute of Engineering

Pulchowk Campus

Department of Electronics and Computer Engineering

Computer Graphics Project

On

“3D Maze”

Submitted

By

Baibhav L. R. [061BCT506] E-mail: baibhavr@gmail.com

Ph: 9841592453

Vivek Gyawali [061BCT547] E-mail: leovivu@hotmail.com

Ph: 9841413967

Submitted

To

Department of Electronics and Computer Engineering

Pulchowk Campus

8th January 2008

Acknowledgement

This project is carried out as per requirement of computer graphics course in our sixth semester.

We extend our first gratitude to our teacher and mentor Er. Bikash Shrestha who guided us and gave valuable suggestions throughout the project duration. We feel very much indebted for his excellent cooperation, which we received during his lectures in the classroom.

Our sincere thank goes to all our friends who helped us directly and indirectly in the completion of this project.

Also we welcome anyone with their valuable criticism and suggestions on the project we have accomplished.

Baibhav Lal Rajbhandary (061BCT506)

Vivek Gyawali (061BCT547)

Abstract

A maze is a complex tour puzzle in the form of a complex branching passage through which the solver must find a route. This is different from a labyrinth, which has an actual through-route and is not designed to be difficult to navigate (despite the common uses of the word to indicate various complex, confusing structures). The pathways and walls in a maze or labyrinth are fixed (pre-determined).

This project, **3D-Maze**, is a simulation of a maze consisting of different colored wall. User can move around the maze using the keyboard, which includes front, back, left and right movement depending upon the keyboard input.

Here first we defined a maze in 2 dimensional map and later on it is converted into three dimensional maze.

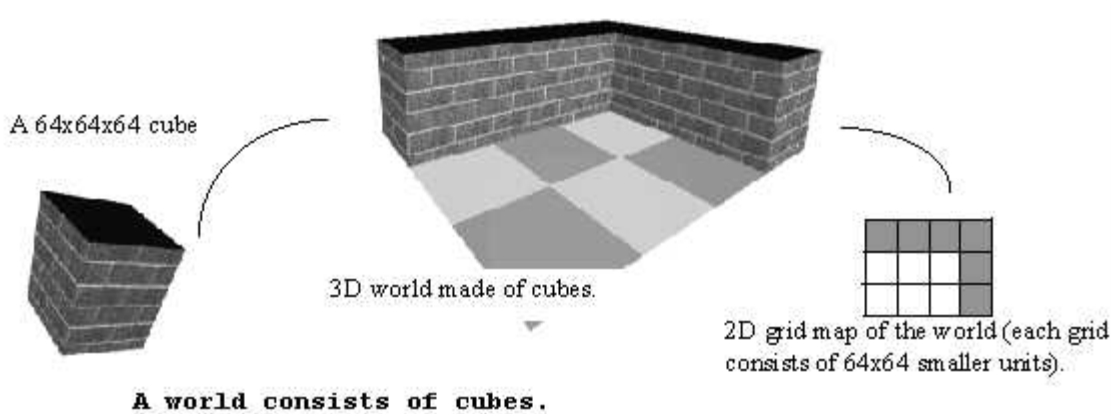
Table of Contents

Acknowledgement	i
Abstract	ii
Table of Contents	iii
Methodology	1
Development Tools	4
Output	5
Limitations	6
Conclusion	6
Bibliography	7

Methodology

2D-Maze

Here the maze map is defined in two-dimensional array consisting of numbers from 0 to 7. Number 0 represent no wall area i.e. free space where user can move and numbers other than 0 represent the wall. Different number assigned for the wall denotes its texture.



Player position and camera plane

After we define the maze on two dimensional array, player position and direction vector along with the camera's plane or projection plane is defined. In our case the camera plane is always perpendicular to the direction vector as shown in the figure below. Here the angle made by two extreme right and left ray is Field of Vision (FOV). It defines the viewable portion of the maze as seen by the user.

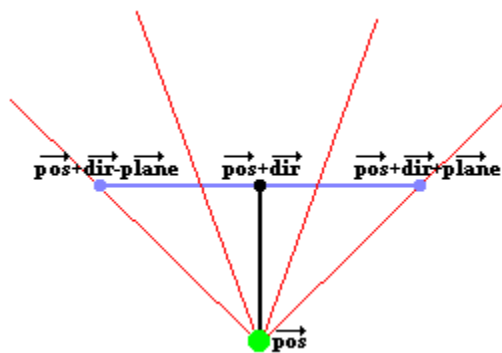


Fig: player position, direction and camera plane vector

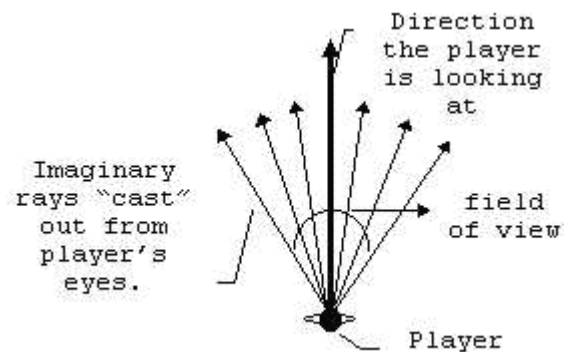


Fig: Ray casting from player position

Visible Surface Detection

Now for the detection of walls we used ray casting method. Ray casting method works pretty much opposite to the natural process of detecting visible surface area. In ray casting method, for every point in the camera plane a ray is cast and first object the ray hits is the visible surface for that point. And for the visible surface detection we use DDA based algorithm. DDA (Digital Differential Analysis) is a fast algorithm typical used on square grid to find out which squares a line hit.

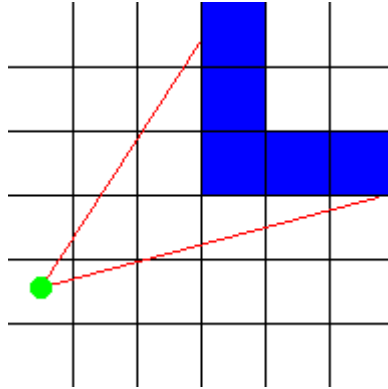


Fig: ray hitting the wall

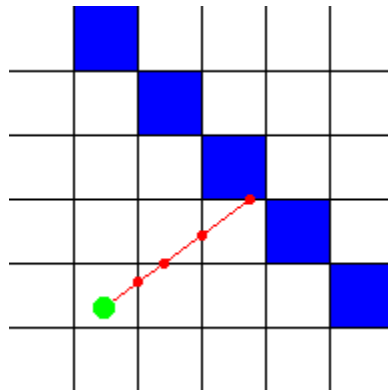


Fig: DDA algorithm implementation for surface detection

Movement of the player

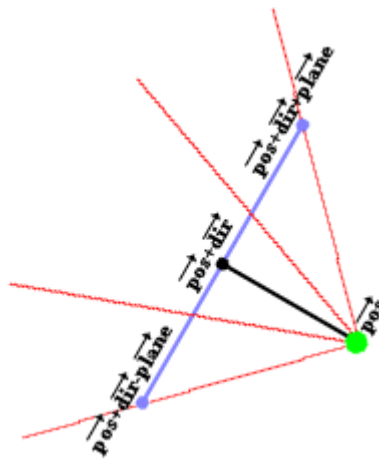
Since the map underlying the maze is two dimensional map, the user can move left, right, forward and backward but not up and down.

When the player rotates, the camera has to rotate, so both the direction vector and the plane vector have to be rotated. Then, the rays will all automatically rotate as well. But the rotation is not possible in z-axis i.e. along the viewing direction.

To rotate a vector, multiply it with the rotation matrix

$$\begin{bmatrix} \cos(a) & -\sin(a) \end{bmatrix}$$

$$\begin{bmatrix} \sin(a) & \cos(a) \end{bmatrix}$$



Wall projection

After we find out at which position the ray hits the wall, the height and the color texture is calculated based on the distance between the player position and the position where the ray hit the wall. Here the height of the wall is inversely proportional to the distance between the player position and the wall.

For the floor, first the position of the floor right in front of the wall is calculated, and there are 4 different cases possible depending if a north, east, south or west side of a wall was hit. After this position and the distances are set, the for loop in the y direction that goes from the pixel below the wall until the bottom of the screen starts, it calculates the current distance, out of that the weight, out of that the exact position of the floor, and out of that the pixel coordinate on the texture. With this info, both a floor and a ceiling pixel can be drawn

Development Tools

Simple DirectMedia Layer (SDL)

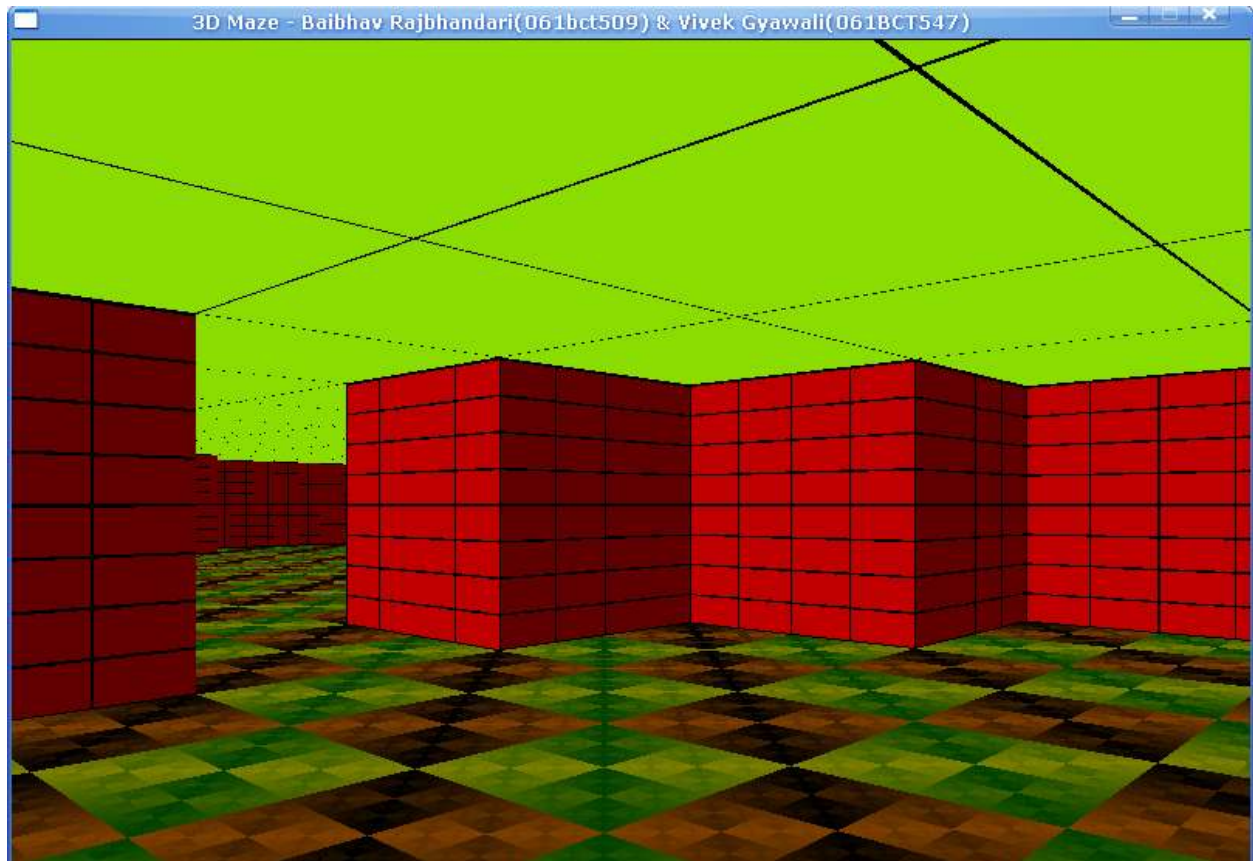
Simple DirectMedia Layer (SDL) is a cross-platform, multimedia, free software library written in C that creates an abstraction over various platforms' graphics, sound, and input APIs, allowing a developer to write a computer game or other multimedia application once and run it on many operating systems.

Here in our project we have used SDL for generating video only i.e. window initialization and drawing the pixel.

Dev C++

For the implementation of above discuss methodology we choose C++ programming language. SDL can be easily integrated into C++ programming environment. Here we have used Dev C++ compiler to accomplish the project.

Output



Limitations

Every software has its limitations and this is no exception, so here are the list of some limitations our software possess

- The use can't move up or down in the vertical direction
- Rotation is not possible in z-axis

Conclusion

Hence, we created a 3D Maze as required for the computer graphics course. We do believe that we have gained a lot of knowledge about working with different graphics and learnt a great deal of algorithms by implementing them as implementations teach us more than just studying a subject matter.

Bibliography

1. Computer Graphics, C Version.

Donald Hearn (Author), M. Pauline Baker (Author)

2. <http://www.libsdl.org/>

3. <http://www.permadi.com/tutorial/raycast/>

4. <http://en.wikipedia.org>

5. Class notes by our lecturer Er. Bikash Shrestha