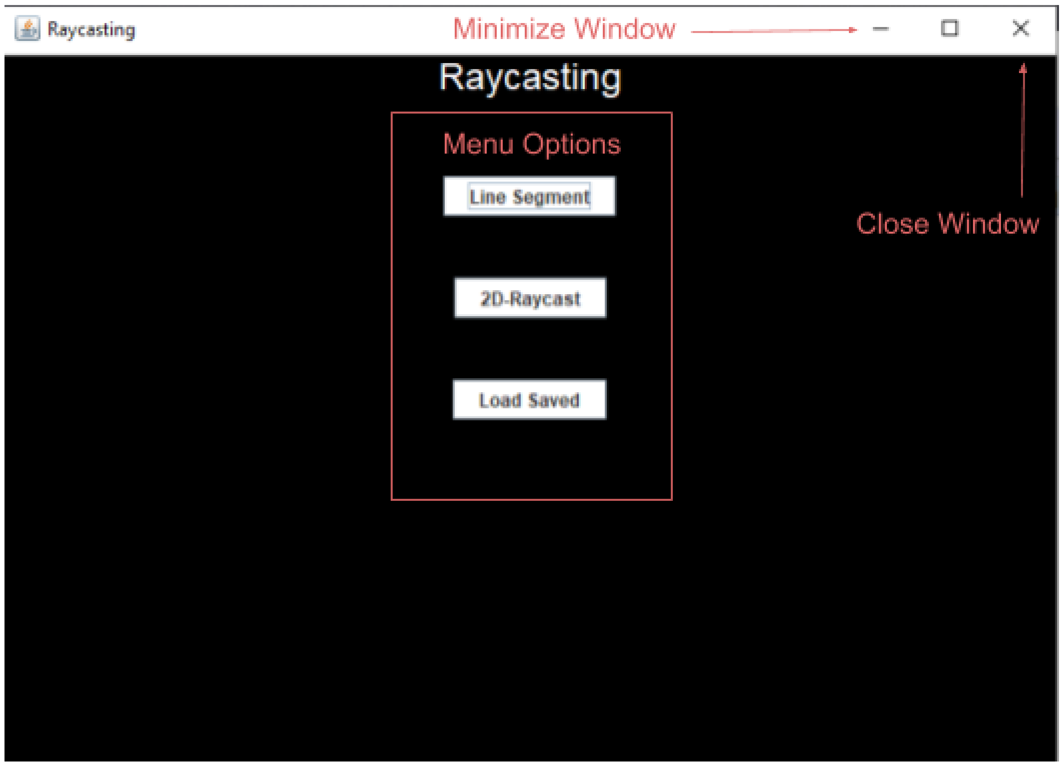
Raycasting Deployment and Maintenance

Lukas Jonca
2020-05-14
ICS4U

Table of Contents

Introduction: Raycasting is the process of casting rays to detect collision between rays and border, which can be used for simple 3D rendering, although this project will only focus on 2D raycasting.There are many practical applications of the concept from simple game AI, light simulation and 3D rendering. The program presented is meant as a demonstration of the concept.

Loading Instructions:
First the user will need to download a copy of proprietary demo software from a credible source after downloading they can simply run the program where they will be prompted with a main



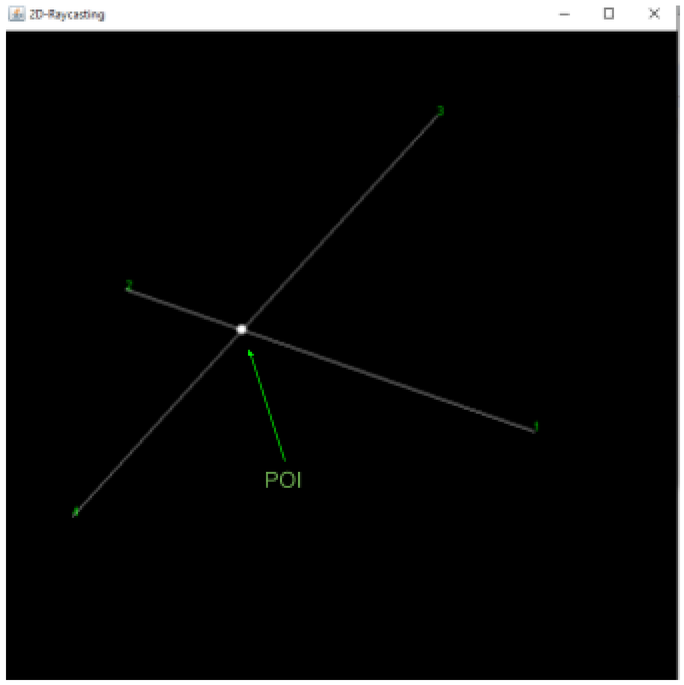
menu as no installation is required.
The demos main menu design is simple and straightforward to understand. On the top right corner are buttons to close the menu, following 3 white buttons of the middle of the screen. The menu prompts the user with 3 options.

Line Segment: will launch a simple demonstration of calculating the intersection of 2 random;y placed line segments.

2D-Raycasting: will launch the full demonstration, generating 10 random walls and showing how the rays interact.

Load: will load in any previously saved files into the program and display it.

Line Segment: Once the line segment option is clicked the program will open a new window where it will generate 2 random lines and determine if they intersect and where.



The demo is simple and has no user input, the purpose of it was simply to be able to effectively test the formulas and to show the user the idea behind the concept.

The way the program works is simply to find first whether the line segments intersect and if they do next, finding the points at which they intersect and drawing an oval. This is done using simple equations and straight forward logic.

The concepts used are not only applicable to raycasting but also collision detection of lines in general.

Equations: The equation used to find whether the points intersect are listed below if 0<t<1 and 0<u<1.

$$t = \frac{(x_1 - x_3)(y_3 - y_4) - (y_1 - y_3)(x_3 - x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}$$

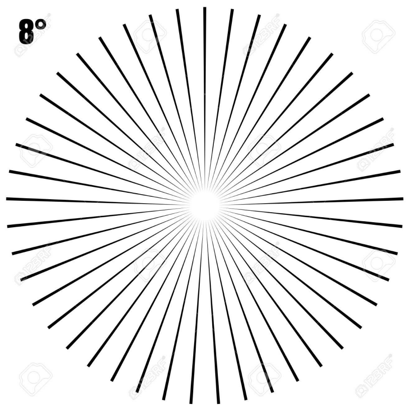
$$u = -\frac{(x_1 - x_2)(y_1 - y_3) - (y_1 - y_2)(x_1 - x_3)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}$$

$$Px = \frac{(x_1 y_2 - y_1 x_2)(x_3 - x_4) - (x_1 - x_2)(x_3 y_4 - y_3 x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}$$

$$Py = \frac{(x_1 y_2 - y_1 x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 y_4 - y_3 x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}$$

Then the exact points of intersection can be found using the Px and Py equations above. The equations will be used to compare every ray to each individual wall, this will be completed after every time the player is moved.

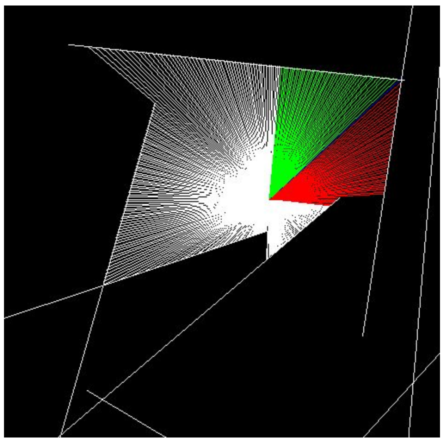
2D Raycasting: To create a 2 raycasting program it simply builds off of the line segment intersection completing the calculations for many more lines using array list, for loops and more complex logic.



The first step in the program is to draw the actual rays, this is done using degrees, drawing a line every 1 degree. The line is calculated using tan then an equation for the line is made and 2 points on the grid are calculated, one at origin the other far away.

Once the rays are drawn walls must be generated using a random number generator and drawing them on the screen.

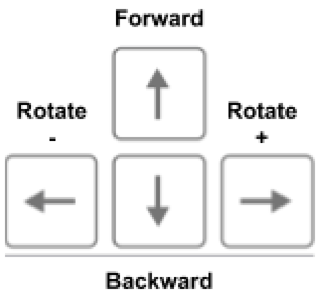
Then the program must calculate whether each ray intersects with a wall on the screen and if it does stopping the ray there.



This will create an image like this representing the players lines of sight.

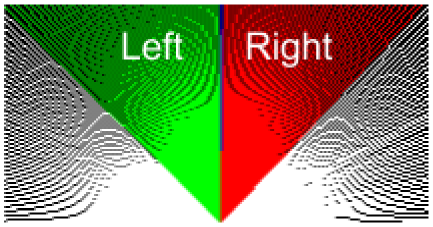
After every single time the player moves the rays will need to be recalculated and reprinted on screen.

Player Controls: The user can interact with the player object on the screen by simply using their arrow keys.



Up Arrow: moves the player forward (direction of blue ray).
Down Arrow: moves the player backward (opposite of blue ray).
Left Arrow: rotates player right (positive direction).
Right Arrow: rotates player left (negative direction).

Player FOV: Although the player FOV has no practical use in the demo it is an important concept for ray casting.



The Player FOV is set to 90 in the demo and is represented by green and red rays. Red represents the right side of the FOV and green represents the left.

The FOV can later be used to display a 3D render of the player's view.

Saving File: The save function under file in the menu bar allows for the state of the player (rays and walls) to be saved for later use. Because all variables are stored in the player object it allows for the state of the program to easily be saved using serialization.

Loading File: Using the load function a previously saved player can be accessed and displayed on the GUI.

Conclusion: The 2D Raycasting demo is a visualization of a simple concept that can be further implemented for more complicated uses such as simple 3D rendering which will be explored further in the future but falls outside the scope of this project.

APA Citations:

Line–line intersection. (2020, May 2). Retrieved May 15, 2020, from https://en.wikipedia.org/wiki/Line–line_intersection