One goal of computer graphics is to generate realistic scenes that imitate the real properties of light. The primary method for doing this is ray tracing, wherein rays are cast for each pixel on the screen to discover where they collide with geometry and their relation to light sources when they do so. Ray tracing is capable of producing exceptionally realistic results, especially when more advanced techniques are introduced, such as path tracing to simulate radiosity and softer shadows. Unfortunately, ray tracing is often too inefficient to utilize within real-time contexts, such as video games. Most video games run at framerates greater than or equal to 30 frames per second, with 60 frames per second becoming increasingly prevalent with modern hardware capability. Rendering scenes as complex as those necessitated by modern video games upwards of 30 times per second is impractical, and, with almost all commercially available graphics hardware, impossible. Recently, however, much attention has been paid to real-time raytracing hardware. Nvidia and AMD, the two prevailing GPU manufacturers for consumers, have both released cards with hardware focused on real-time ray tracing. Although these cards do not yet have a strong presence in modern video game development, early adoption has produced promising results for how this technology may be used to improve the realism of games, both modern and classic (https://www.youtube.com/watch?v=vrq1T93uLag&t=93s). However, while such results produce an impressively immersive effect, it’s mostly atmosphere that benefits from the hardware upgrade. Light reflects more realistically off of water, explosions and gun shots reflect spread light on their surroundings, and shadows appear softer and more realistic. Aesthetic upgrades are important, and video game developers are always looking for methods to increase the graphical fidelity of their titles, but it remains largely unexplored whether realistic lighting techniques can be used to
influence and improve gameplay in an interactive experience. The goal of this project is to explore the potential of ray tracing to be used as a gameplay element in an interactive experience. That is, this project will seek to demonstrate several ways in which realistic light simulation can be used to design puzzles in a short video game.

This project will be a small proof-of-concept video game demo where the player controls a robot trying to escape from a carnival fun house, filled with optical illusions and puzzles. Players will be able to move the robot or adjust the angle of its head (the camera through which the player views the world). Both of these actions are taken discretely, meaning that the player will input commands, then see the camera feed update once – one render per action. While the project won’t utilize real-time raytracing, it will hopefully demonstrate the use of raytracing as a gameplay element, even in a slower, point-and-click manner. While I haven’t determined the exact forms of any puzzles yet, I intend to utilize implicit surfaces to replicate carnival fun house mirrors and create a mirror maze the player will attempt to navigate. I also would like to play with the concept of lenses distorting something behind them, requiring the player to gain a new vantage point to determine how they must proceed.

This project will be an in-depth exploration of ray tracing techniques, seeking to create a system that can implement basic materials systems (specular, diffuse, glass), as well as textures, bump mapping, implicit surfaces, and possibly 3D models. In order to ensure that renders are created as quickly as possible to minimize player wait time, the game will be written in C++. I will also attempt to make effective use of canonical acceleration techniques, such as creating a uniform grid. While I’m still exploring options for libraries to use while writing the ray-tracer, my current plan is to use Simple Directmedia Layer (SDL), which provides “low level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D,” per its website (https://www.libsdl.org/).

Currently I am exploring two methods of interaction with the game for the player:
a) The player will control the robot by interacting (clicking on) with on-screen buttons. There will be buttons that allow the player to rotate the camera, both parallel to the ground and in the up-down direction, and a set of buttons that allow the player to move in one of 8 cardinal directions. The benefit of this input system is that predetermined locations in the world can be decided during development that keep playthroughs fairly quick and guarantee that players move to points of interest. This does, however, potentially cheapen the effect of raytracing, as each screen could theoretically be rendered and stored, though this might require dozens of angles per location. It also has the benefit of keeping player interaction completely within the game window.

b) The player will control the robot via the command prompt, issuing commands like “move N” to move the robot to the north. Commands such as “angle lat 45” may also be given to specify the angle of the camera precisely. This control method has the benefit of allowing the user to specify more directly the angles they want to view, increasing the impact of the generated renders. It does have several drawbacks, however. First, it might prove annoying to have to switch between the game window and the command prompt to control the game. Second, without specific points of interest designated beforehand, it might become easy for the player to get lost or struggle to find the proper camera positions to solve puzzles.

It might also prove beneficial to combine these two methods, and provide a brief description of player position via the command prompt, with possible input options that the player can choose.

The final project will be composed of an executable for the game engine, as well as additional files for room description and possibly models. I also hope to produce a report for each playthrough detailing which positions in the world the player moved to, and which angles the camera was positioned at. I hope to perform some rudimentary analysis on this data to judge the efficacy of the game’s design, and possibly to recognize any trends across different players’ experiences.