CS535: Assignment #2 – Beam Tracing

Out: February 10, 2026
Back/Due: February 24, 2026

Objective:
This objective of this assignment is to obtain a good understanding of tracing. While "ray tracing" is something you have probably heard and have perhaps implemented previously, its cousins path tracing and beam tracing are more efficient. Beam tracing can be quite efficient for small to medium complexity polygonal models. Beam tracing can be used for rendering/shading/illumination and also for auralization (i.e., audio). Here are some recent path/beam tracing papers that also use deep learning:

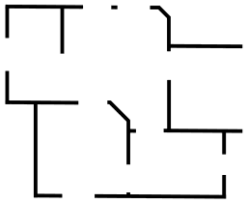
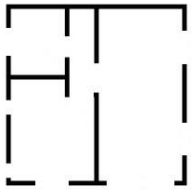
- "High-Fidelity 4× Neural Reconstruction of Real-time Path Traced Images", WACV 2025.
- "Real-Time Markov Chain Path Guiding for Global Illumination and Single Scattering", I3D 2025.
- "A beam tracing approach to acoustic modeling for interactive virtual environments", SIGGRAPH '98
- "Real-Time Object Tracking with On-Device Deep Learning for Adaptive Beamforming in Dynamic Acoustic Environments", Journal of Supercomputing, 2026.

Unfortunately, there is not enough time to do a full path/beam tracing implementation, so instead we you will do a mini 2D version using interior space floor plans with a receiver (i.e., "eye"). In the design below, the receiver can quantify the amount of light received or the amount of sound received – only the amount of light received is required for the assignment, but the extension to include sound is quite simple, fun, and extra credit.

Specifics:
(0) **(5%) Floor plan creation**. You are to create two interior 2D floor plans. The floor plan only needs to define wall structures. On the left is a nice 3D model…too much work for now, so just create things like on the right – only wall structures; define a text-file format to store the wall segments such

```
n
(x1, y1, x2, y2)
(x1, y1, x2, y2)
(x1, y1, x2, y2)
…
```

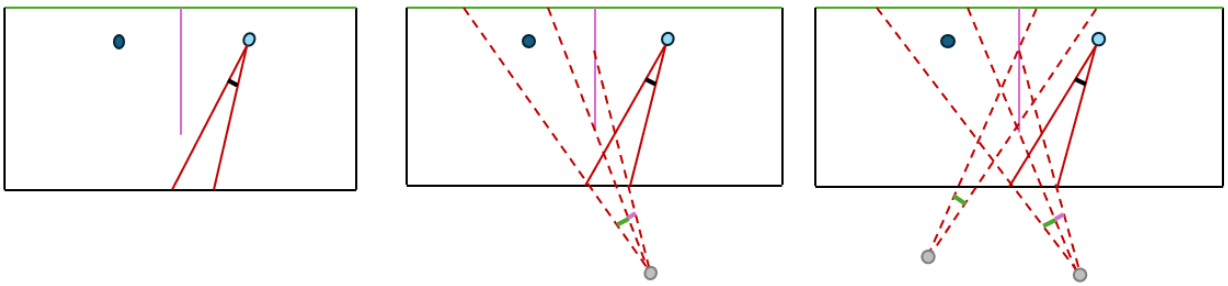
where n is the number of wall segments and each line has the end points of the line segments.

(1) **(10%) Source and Receiver Renderer**. Next, define at a random location inside the floor plan (but not "on a wall"), initial location of one source, and initial location of one receiver.
   a. **Source**. It is a dot somewhere in the floor plan (maybe 5x5 pixels).
   b. **Receiver**. It is also a dot somewhere in the floor plan but it has a "field of view" (e.g., 30-degrees). The field of view (FOV) should be displayed a simple pair of lines that converge on the receiver (e.g., a "wedge").
   c. **GUI**. Using either keys or mouse, you should be able to translate in XY the source or receiver. You should also be able to "rotate" the receiver and the receiver's FOV should visually rotate.

(2) **(65%) Beam Tracer**. The main aspect is tracing beams from the receiver outwards. There are two ways you can think of doing the beam tracing. Both are relatively short in terms of amount of code and exploit a recursive mentality.
   a. **Option A** (can be done fully with CPU rendering)
      i. (5%) *Receiver*: define an "image line-segment" (e.g., an image plane but in 2D) at some distance from the receiver, perpendicular to the line of sight and of width dictated by the FOV.
      ii. (30%) *Beam Shooting*: shoot a "beam" from the receiver outwards and project all line segments onto the image line-segment; store only the closest projections; this defines how "far" the beam reaches. *Draw lines representing the traced beams.*
      iii. (20%) *Beam Reflecting*: for each line segment on the image line-segment, compute the reflected beam using a standard reflection vector computation.
      iv. (10%) *Recursion*: for each reflected beam, call the above "Beam Shooting"; repeat until a maximum number of reflections or until the beam shoots outside the floorplan and hits nothing.
   b. **Option B** (can use OpenGL pipeline but requires some ingenuity)
      i. (5%) *Receiver*: define a "camera" at the receiver with the specified FOV
      ii. (30%) *Beam Shooting*: render floor plan to camera view
      iii. (20%) *Beam Reflecting*: for each rendered segment, compute the reflected view direction and setup the next camera
      iv. (10%) *Recursion*: for each reflected beam, call the above "Beam Shooting"; repeat until a maximum number of reflections or until the beam shoots outside the floorplan and hits nothing.

(receiver: light blue dot, source: dark blue dot)



Initial beam | First reflected beam | Second reflected beam

(3) **(20-30%) Distance Estimator**. As you trace the beams, you accumulate total "distance" d and when the FOV of the beam contains the source, you add in the distance from the last reflection to the source, you stop recursion and you report back to the receiver the total distance value. If multiple beams hit the same source, store multiple total distance values at the receiver.

a. **(20%) Illumination**: as minimum requirement, you can imagine the source is a light of intensity h, so you multiple the light intensity value by the inverse total-distance-squared: i.e., $light = Kh/d^2$ where K is some scalar to bring the illumination value into a nice range, e.g., [0, 255]. Then you "render" the intensity (and size?) of the receiver based on the value of "light". This way when you rotate/translate the receiver close to the source, it becomes bigger/brighter. If you do not ever see the source, it stays a minimum size and brightness.

b. **(10%) Extra credit**: when you receive the value of "light", instead you produce an audible source – here, brighter means louder. You just did a minimal source renderer! Now, when your receiver translates through the floorplan, the volume of sound changes!

Grading:
Your program will be tested against the aforementioned functionality and your code will be inspected.

Turn-in:
To give in the assignment, please use Brightspace. Give in

- **a zip file with your complete project (project files, source code, and precompiled executable).**
- **a short video of your program working**

It is your responsibility to make sure the assignment is delivered/dated before it is due.

If you implement the extra credit, please ensure instructions are given on how to use it --- put such in the GUI or have clear instructions printout – do not assume we will read your code and decipher how to use your extra credit.