ECE3090 **Fall Semester, 2012**

Wireless Path Loss

Assigned: March 6, 2012 *Due: March 16, 2012, 11:59pm*

Given a wireless base station with a given transmitter power P_t, we want to compute the received power at a wireless mobile device at every point in a building. The formula we will use to compute this is:

$$P_r = P_t + G_t + G_r - 10\log_{10}(\frac{4\pi f}{C}) - 20\log_{10}R - \sum_{n=0}^{n=k-1} X_n$$

Where:

P_r is the received power at the wireless mobile device (dBm)

P_t is the transmit power at the wireless base station (dBm)

G_t is the antenna gain factor for the transmitter (dBm)

G_r is the antenna gain factor for the receiver (dBm)

f is the carrier frequency for the signal (hz)

C is the speed of light (meters/sec)

R is the distance from the transmitter to the receiver (meters)

k is the number of walls in the building

X_n is the path loss through wall n (dBm).

For this assignment, use the following values:

P_t = 20dBm

G_t = 0

G_r = 0

f = 2.4GHz

$C = 3 \times 10^8$ meters/sec

X_n = 5 dBm if the line segment from the transmitter to the receiver intersects wall n, 0 otherwise.

Problem Description. Assume a square, single story building of 256 meters by 256 meters in size. The input file walls.txt describes the walls found in the building. Each line in the file describes one wall, specified as the starting and ending point of the wall in x and y coordinates, in units of meters. For example:

10 50 10 100

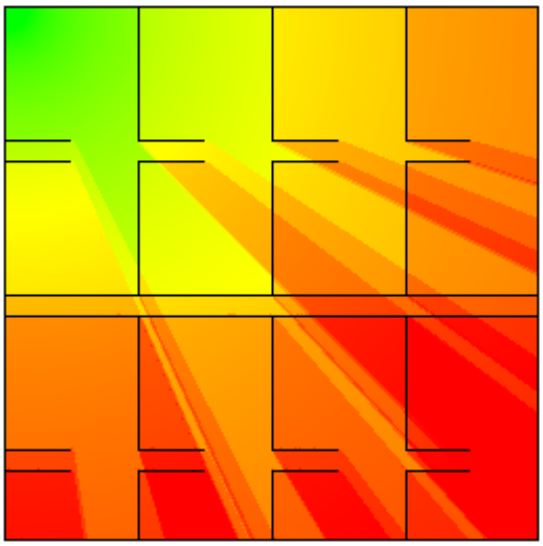
describes a vertical wall that is 50 meters long from (x=10,y=50) to (x=10,y=100). For simplicity, in this assignment all walls will be either horizontal or vertical. Note that the "final point" (10/100 in the above example) is *one beyond the end*, meaning that you don't put a pixel at the last point. There will likely be a corner there, and the same pixel will be the starting point of another line.

Your program is to compute the P_r at every point in the building (on a 1 meter granularity) that is NOT inside a wall, using the above formula, and display a graphical representation of the calculated strength. A green pixel would mean "good" signal, yellow means "marginal", and red means "poor".

Program Details. Your program is to do the following using the QDisplay API that we have discussed previously.

1. Create an blank 276 pixel by 276 pixel window using the supplied `qtwindow` graphics library, and specify a color depth of 32 bits. This represents a geographic region of 256 meters by 256 meters, with a 10 pixel border on all sides that is not part of the region. This means that each pixel is equivalent to 1 meter of distance.
2. **NOTE:** In the ImageFiltering assignment our pixels were 8–bit gray scale, so we used a `unsigned char*` to point to the pixel data returned by `ImageData`. Here, we are using 32-bit RGB color pixel reprentations. The correct type for the pixel array is `QRgb*`. There is an example below that shows how to both define the pixel array and how to create a pixel of the desired RGB color.
3. Read the walls.txt file and draw the walls on the window using *black pixels*. Assume each pixel represents one square meter. Of course this means that we have walls that are 1 meter thick!
4. Assume the transmitter is at location (1,1) (NOT (0,0) which is inside a wall!)..
5. **For the next steps, use 16 threads cooperating to perform the computations.**
6. For every square meter (pixel) in the region that is not inside a wall, calculate the received signal power per the above formula.
7. Color the pixel on a scale from completely green (for P_r = -20dBm) down to completely red (for P_r = -80dBm), with solid yellow at the value of -50dBm. Use linear interpolation for values in between these specified ones. For any calculated P_r less than -80dBm, use solid red, and for any calculated P_r more than -20dBm, use solid green.

The expected results are shown below.



Program Design Hints. You will likely want a class `Point` representing a single point in the x/y plane, and a class `Line` representing the two endpoints of a line segment. You should create a `vector` of lines to represent the walls. The only tricky part of this assignment is determining if a line segment from the transmitter to a receiver intersects one or more walls. To do this, make a subroutine called `Intersects` (or something like that) with two `Line` arguments that returns a boolean `true` if the line segments intersect, and `false` otherwise. Compute the slopes of the two lines, and if they are parallel (same slope) they do not intersect unless they are part of the same line. Non–parallel lines, by definition, have an intersection point. Find the intersection point of the two lines and determine if the point is on both of the two line segments. If the intersection point is on both line segments, the line segments intersect, otherwise they do not. To calculate this, you might want a subroutine `PointOnLine` to find if a given point is on a given line segment. One complication is if one of the lines has infinite slope, which requires a bit of extra coding.

You will need to create the 32–bit RGB values to store in the pixel buffer. The easiest way to do this is to use the `QColor` helper class, as follows (for example):

```
QRgb* p = (QRgb*)d.ImageData();
p[i] = QColor(255, 0, 0).rgb();
```

The above example uses the QColor constructor that accepts three 8–bit values representing the magnitude of the red, green, and blue components respectively. The QColor object has an `rgb()` method to convert the information to a 32–bit pixel representation. This examples sets the i'th pixel to bright red.

Resources. All skeleton files are in the usual place, on `jinx` in the `PathLoss` subdirectory.

Turning in your Project. Run the usual `riley-turnin` procedure as always.