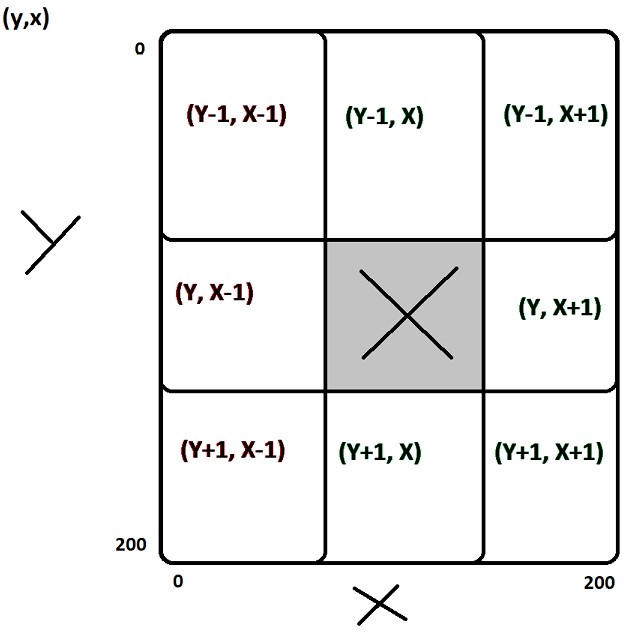
C Programming

Peak of Mountain

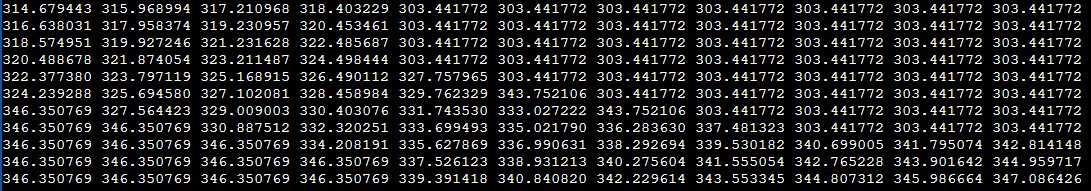
Cian Herlihy – R00205604

# X,Y Diagram



I used this as a reference of what the x and y values should do when travelling in a certain direction.

# False Peak



I found a false peak with 343.752106 with no higher values adjacent to it. This should not have been possible and forced me to change my program completely to check areas nearby for higher values rather than touching units. My first iteration of the program I created did not need a check surrounding areas until I met this false peak.

# gradient\_sol.c

#include "gradient.h"

path\_point find\_highest\_point(){

path\_point my\_point;

float my\_view[VIEW\_SIZE][VIEW\_SIZE];

int y, x, xValue, yValue, xCounter, yCounter;

int option = 0;

int lastOption = 0;

//Start at (70,70) in hopes of finding mountain quikcer

my\_point.x = 70;

my\_point.y = 70;

generate\_view(my\_view, my\_point.y, my\_point.x);

// Reset to 0 before loop starts

xValue = my\_point.x = 5;

yValue = my\_point.y = 5;

xCounter = 5;

yCounter = 5;

while(1)

{

float value = my\_view[yValue][xValue]; // Current Value

float t\_val = my\_view[yValue-1][xValue]; // Top Value

float tr\_val = my\_view[yValue-1][xValue+1]; // Top Right Value

float r\_val = my\_view[yValue][xValue+1]; // Right Value

float br\_val = my\_view[yValue+1][xValue+1]; // Bottom Right Value

float b\_val = my\_view[yValue+1][xValue]; // Bottom Value

float bl\_val = my\_view[yValue+1][xValue-1]; // Bottom Left Value

float l\_val = my\_view[yValue][xValue-1]; // Left Value

float tl\_val = my\_view[yValue-1][xValue-1]; // Top Left Value

//Array full of values

float arrayVals[] = {t\_val, tr\_val, r\_val, br\_val, b\_val, bl\_val, l\_val, tl\_val};

int highestVal = 0;

for (int i=0; i<5; i++)

{

if (highestVal < arrayVals[i])

{

highestVal = arrayVals[i];

}

}

/\*

Checks if it is at the edge of the view and then proceeds to check certain directions.

Sends option number to switch case to iterate the x or y value depending on what direction it went.

\*/

if (xValue == 0 || xValue == 10 || yValue == 0 || yValue == 10)

{

generate\_view(my\_view, yCounter, xCounter);

xValue = my\_point.x = 5;

yValue = my\_point.y = 5;

}

else if (value < br\_val && br\_val < 999) // Bottom Right Value

{

option = 0;

lastOption = 0;

}

else if (value < b\_val && b\_val < 999) // Bottom Value

{

option = 1;

lastOption = 1;

}

else if (value < r\_val && r\_val < 999) // Right Value

{

option = 2;

lastOption = 2;

}

else if (value < tl\_val && tl\_val < 999) // Top Left Value

{

option = 3;

lastOption = 3;

}

else if (value < l\_val && l\_val < 999) // Left Value

{

option = 4;

lastOption = 4;

}

else if (value < t\_val && t\_val < 999) // Top Value

{

option = 5;

lastOption = 5;

}

else if (value < bl\_val && bl\_val < 999) // Left Value

{

option = 6;

lastOption = 6;

}

else if (value < tr\_val && tr\_val < 999) // Top Value

{

option = 7;

lastOption = 7;

}

else if (value == t\_val && value == r\_val && value == b\_val && value == l\_val)

{

option = 9;

}

else if (value == b\_val && lastOption != 5)

{

option = 1;

lastOption = 1;

}

else if (value == l\_val && lastOption != 2)

{

option = 4;

lastOption = 4;

}

else if (value == t\_val && lastOption != 1)

{

option = 5;

lastOption = 5;

}

else if (value == r\_val && lastOption != 4)

{

option = 2;

lastOption = 2;

}

else if (value >= highestVal) // Has found false peaks in the past so I have it do option 9 if it is not the true peak. Example shown in document

{

if(declare\_peak(xCounter, yCounter) == 1)

{

my\_point.x = xCounter;

my\_point.y = yCounter;

return my\_point;

}

else

{

option = 9;

}

}

else

{

option = 9;

}

switch (option)

{

case 0: // Bottom Right Value

xValue++;

yValue++;

xCounter++;

yCounter++;

break;

case 1: // Bottom Value

yValue++;

yCounter++;

break;

case 2: // Right Value

xValue++;

xCounter++;

break;

case 3: // Top Left Value

xValue--;

yValue--;

xCounter--;

yCounter--;

break;

case 4: // Left Value

xValue--;

xCounter--;

break;

case 5: // Top Value

yValue--;

yCounter--;

break;

case 6: // Bottom Left Value

yValue++;

yCounter++;

xValue--;

xCounter--;

break;

case 7: // Top Right Value

yValue--;

yCounter--;

xValue++;

xCounter++;

break;

case 9: // Checks nearby areas when in the middle of plateau

/\*

generate view to the Top Right

\*/

generate\_view(my\_view, yCounter-10, xCounter+10);

int highTRx=0, highRx=0, highBRx=0, highBx=0, highBLx=0, highLx=0, highTLx=0, highTx=0, highCx=0;

int highTRy=0, highRy=0, highBRy=0, highBy=0, highBLy=0, highLy=0, highTLy=0, highTy=0, highCy=0;

float highTRval=0, highRval=0, highBRval=0, highBval=0, highBLval=0, highLval=0, highTLval=0, highTval=0, highCval=0;

for (y=0; y<VIEW\_SIZE; y++)

{

for (x=0; x<VIEW\_SIZE; x++)

{

if (highTRval < my\_view[y][x] && my\_view[y][x] < 999)

{

highTRx = x;

highTRy = y;

highTRval = my\_view[y][x];

}

}

}

/\*

generate view to the Right

\*/

generate\_view(my\_view, yCounter, xCounter+10);

for (y=0; y<VIEW\_SIZE; y++)

{

for (x=0; x<VIEW\_SIZE; x++)

{

if (highRval < my\_view[y][x] && my\_view[y][x] < 999)

{

highRx = x;

highRy = y;

highRval = my\_view[y][x];

}

}

}

/\*

generate view to the Bottom Right

\*/

generate\_view(my\_view, yCounter+10, xCounter+10);

for (y=0; y<VIEW\_SIZE; y++)

{

for (x=0; x<VIEW\_SIZE; x++)

{

if (highBRval < my\_view[y][x] && my\_view[y][x] < 999)

{

highBRx = x;

highBRy = y;

highBRval = my\_view[y][x];

}

}

}

/\*

generate view to the Bottom

\*/

generate\_view(my\_view, yCounter+10, xCounter);

for (y=0; y<VIEW\_SIZE; y++)

{

for (x=0; x<VIEW\_SIZE; x++)

{

if (highBval < my\_view[y][x] && my\_view[y][x] < 999)

{

highBx = x;

highBy = y;

highBval = my\_view[y][x];

}

}

}

/\*

generate view to the Bottom Left

\*/

generate\_view(my\_view, yCounter+10, xCounter-10);

for (y=0; y<VIEW\_SIZE; y++)

{

for (x=0; x<VIEW\_SIZE; x++)

{

if (highBLval < my\_view[y][x] && my\_view[y][x] < 999)

{

highBLx = x;

highBLy = y;

highBLval = my\_view[y][x];

}

}

}

/\*

generate view to the Left

\*/

generate\_view(my\_view, yCounter, xCounter-10);

for (y=0; y<VIEW\_SIZE; y++)

{

for (x=0; x<VIEW\_SIZE; x++)

{

if (highLval < my\_view[y][x] && my\_view[y][x] < 999)

{

highLx = x;

highLy = y;

highLval = my\_view[y][x];

}

}

}

/\*

generate view to the Top Left

\*/

generate\_view(my\_view, yCounter-10, xCounter-10);

for (y=0; y<VIEW\_SIZE; y++)

{

for (x=0; x<VIEW\_SIZE; x++)

{

if (highTLval < my\_view[y][x] && my\_view[y][x] < 999)

{

highTLx = x;

highTLy = y;

highTLval = my\_view[y][x];

}

}

}

/\*

generate view to the Top

\*/

generate\_view(my\_view, yCounter-10, xCounter-10);

for (y=0; y<VIEW\_SIZE; y++)

{

for (x=0; x<VIEW\_SIZE; x++)

{

if (highTval < my\_view[y][x] && my\_view[y][x] < 999)

{

highTx = x;

highTy = y;

highTval = my\_view[y][x];

}

}

}

/\*

generate view to the Centre

\*/

generate\_view(my\_view, yCounter, xCounter);

for (y=0; y<VIEW\_SIZE; y++)

{

for (x=0; x<VIEW\_SIZE; x++)

{

if (highCval < my\_view[y][x] && my\_view[y][x] < 999)

{

highCx = x;

highCy = y;

highCval = my\_view[y][x];

}

}

}

// Gathers all highest values found and puts into an array

float platCheckVals[] = {highTRval, highRval, highBRval, highBval, highBLval, highLval, highTLval, highTval, highCval};

float max = 0;

int arrayPos = 0;

for (int i=0; i<9; i++) // Iterates through array and picks highest option and travels to that altitude to carry on search

{

if (max < platCheckVals[i] && platCheckVals[i] > 1)

{

max = platCheckVals[i];

arrayPos = i;

}

}

if (max < 1)

{

arrayPos = 9;

}

// Switch case to handle x and y values being plus/minus respectively

switch (arrayPos)

{

case 0: // Top Right

yCounter = yCounter - (15 - highTRy);

xCounter = xCounter + 5 + highTRx;

break;

case 1: // Right

yCounter = yCounter - 5 + highRy;

xCounter = xCounter + 5 + highRx;

break;

case 2: // Bottom Right

yCounter = yCounter + 5 + highBRy;

xCounter = xCounter + 5 + highBRx;

break;

case 3: // Bottom

yCounter = yCounter + 5 + highBy;

xCounter = xCounter - 5 + highBx;

break;

case 4: // Bottom Left

yCounter = yCounter + 5 + highBLy;

xCounter = xCounter - (15 - highBLx);

break;

case 5: // Left

yCounter = yCounter - 5 + highLy;

xCounter = xCounter - (15 - highLx);

break;

case 6: // Top Left

yCounter = yCounter - (15 - highTLy);

xCounter = xCounter - (15 - highTLx);

break;

case 7: // Top

yCounter = yCounter - (15 - highTy);

xCounter = xCounter - 5 + highTx;

break;

case 8: // Centre

yCounter = yCounter - 5 + highTy;

xCounter = xCounter - 5 + highTx;

break;

case 9: // Randomise if no higher value found

xValue = my\_point.x = random()%100;

yValue = my\_point.y = random()%100;

xCounter = xValue;

yCounter = yValue;

generate\_view(my\_view, my\_point.y, my\_point.x);

break;

}

generate\_view(my\_view, yCounter, xCounter);

xValue = my\_point.x = 5;

yValue = my\_point.y = 5;

break;

}

}

return my\_point;

}