Austin Smothers

CISP 430

Assignment 16

I was aiming to make a happy little sky with

some lovely northern lights in it

I hope I did Bob Ross proud

If not this stuff could probably work for

a Colgate commercial

/\*

SSFEA (Super Simple Finite Element Analysis)

THE TEXT BASED VERSION

Original:

April 2016 3:00 AM

Dan Ross

Implements a TEXT output.

Suggest you change stuff here before messing with pixels.

UPDATED TO BOREALIS.cpp

THE TEXT BASED VERSION

Modified:

December 4, 2017

Austin Smothers

This program will use the emission spectra of both

Oxygen and Nitrogen, combined with the equations

governing light-emission as a result of solar wind

to simulate the phenomenon known as "Aurora Borealis"

flowQuantity = K \* (Q1 - Q2)

K = ProportionalityConstant----------------

A tribute to Bob Ross

\*/

#include <iostream>

#include <cmath>

#include <stdlib.h>

using namespace std;

#define PIE YUMMY // its absolutely true

#define MAX 32 // size of the matrix

#define ITERATIONS 100 // how many times the matrix is solved

#define K 0.025F // proportionality constant, set to .025 to adjust wavelength by 10 at first and slowly blend

#define SEED 700 // an initial value for the visible wavelength of oxygen/nitrogen

// the data

// how much "stuff" is in each element

float Q[MAX][MAX];

// 1 indicates that an element is blocked, 0 that it is free

bool B[MAX][MAX] = {

// 40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69

{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },

/\*this marks the split between the nitrogen emission spectra (top)

and the oxygen emission spectra (bottom)\*/

{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }

// 40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69

};

// text output

void printQ()

{

for (int row = 0; row < MAX; row++)

{

for (int col = 0; col < MAX; col++){

//check if we are allowed to print contents

if (Q[row][col])

printf("%1d ", (int)Q[row][col]);

}

printf("\n");

}

printf("\n");

}

/\*

Calculates how much "stuff" to move using a simple proportionality rule.

More sophisticated rules (differential equations) could also be used.

Moves "stuff" between a target and a neighbor cell.

\*/

void transfer(int tRow, int tCol, int nRow, int nCol)

{

float dQ; // flow quantity difference

// calculate how much Q to move (simple proportionality)

dQ = K \* (Q[tRow][tCol] - Q[nRow][nCol]);

Q[tRow][tCol] -= dQ; // adjust target

Q[nRow][nCol] += dQ; // adjust neighbor

}

/\*

Iterates thru elements of the Q matrix and updates the stuff in each element

\*/

void updateQ()

{

// for each target element

for (int row = 0; row < MAX; row++)

{

for (int col = 0; col < MAX; col++)

{

// NORTH

if (!B[row][col] && !B[row - 1][col]) // if target and its neighbor are not blocked

transfer(row, col, row - 1, col); // move stuff

// EAST

if (!B[row][col] && !B[row][col + 1])

transfer(row, col, row, col + 1);

// SOUTH

if (!B[row][col] && !B[row + 1][col])

transfer(row, col, row + 1, col);

// WEST

if (!B[row][col] && !B[row][col - 1])

transfer(row, col, row, col - 1);

}

}

}

int main()

{

for (int col = 0; col < MAX; col++) {

for (int row = 0; row < 8; row++) {

if (!B[row][col])

Q[row][col] = SEED; // seed value for nitrogen

}

for(int row = 17; row < 24; row++){

if (!B[row][col])

Q[row][col] = SEED; // seed value for oxygen

}

}

// do it a buncha times

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

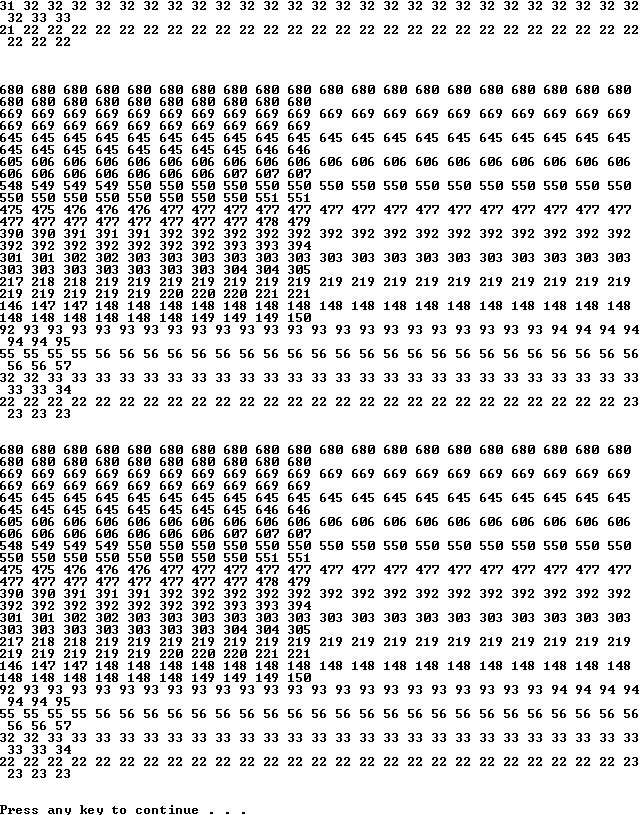
{

printQ();

updateQ();

}

}



/\*

SSFEA (Super Simple Finite Element Analysis)

THE PIXEL BASED VERSION

April 2016 3:00 AM

Dan Ross

Implements a PIXEL output.

If you make changes, you should start with the TEXT version then

port your changes to the pixel version.

MODIFIED TO BOREALIS.cpp

Modified:

December 4, 2017

Austin Smothers

This program will use the emission spectra of both

Oxygen and Nitrogen, combined with the equations

governing light-emission as a result of solar wind

to simulate the phenomenon known as "Aurora Borealis"

flowQuantity = K \* (Q1 - Q2)

K = ProportionalityConstant----------------

A tribute to Bob Ross

\*/

#include<windows.h>

#include<iostream>

#include <cmath>

#include <stdlib.h>

using namespace std;

HDC mydc;

HWND myconsole;

#define PIE YUMMY // its absolutely true

#define MAX 160 // size of the matrix

#define K 0.25F // proportionality constant (CAN ADJUST THIS)

#define SEED 1000 // an initial wavelength for nitrogen

#define SEED2 530 // an initial wavelength for nitrogen

#define SEED3 430 // an initial wavelength for nitrogen

#define SEED4 1200 // an initial wavelength for oxygen

#define SEED5 560 // an initial wavelength for oxygen

#define SEED6 500 // an initial wavelength for oxygen

// the data

// how much "stuff" is in each element

float Q[MAX][MAX];

// 1 indicates that an element is blocked

bool B[MAX][MAX] = { 0 };

/\* pixel output

Heavily modified for use with the aurora borealis

emission spectra\*/

void drawQ()

{

//Hi there, so glad you could join us this evening

int red = 0;

int green = 0;

int blue = 0;

/\*Today we've got a real treat for ya! We're gonna be painting with two palettes.

Now, before I get too crazy, let me explain what I've got going on over here:

We've got our standard 32x32 stretched canvas, but you can use whatever SIZE

you like. And let's go ahead and have them run the colors we'll be using

across the screen graphically.\*/

//the range of color emissions based on wavelength for nitrogen

//Nitrogen Pallet

for (int row = 1; row < (MAX/2); row++)

{

for (int col = 1; col < (MAX - 1); col++)

{

//Midnight Black

if (Q[row][col] <= 400) {

red = 0;

green = 0;

blue = 0;

}

//Prussian Blue

else if (Q[row][col] <= 464) {

red = 0;

green = 49;

blue = 83;

}

//Phthalo Blue

else if (Q[row][col] <= 570) {

red = 42;

green = 100;

blue = 173;

}

//Mountain Blue

else if (Q[row][col] <= 600) {

red = 0;

green = 156;

blue = 255;

}

//Soft Turquoise

else if (Q[row][col] <= 610) {

red = 0;

green = 128;

blue = 128;

}

//Soft Mauve

else if (Q[row][col] <= 690) {

red = 150;

green = 120;

blue = 182;

}

else {

red = 0;

green = 0;

blue = 0;

}

COLORREF COLOR = RGB(red, green, blue);

SetPixel(mydc, col, row, COLOR);

}

}

//the range of color emissions based on wavelength for oxygen

//Oxygen Pallet

for (int row = MAX / 2; row < (MAX - 1); row++)

{

for (int col = 1; col < (MAX - 1); col++)

{

//Midnight Black

if (Q[row][col] <= 400) {

red = 0;

green = 0;

blue = 0;

}

//Phthalo Green

else if (Q[row][col] <= 520) {

red = 18;

green = 53;

blue = 36;

}

//Soft Viridian Green

else if (Q[row][col] <= 580) {

red = 0;

green = 155;

blue = 56;

}

//Sap Green

else if (Q[row][col] <= 600) {

red = 97;

green = 255;

blue = 0;

}

//Soft Cadmium Red

else if (Q[row][col] <= 690) {

red = 227;

green = 0;

blue = 34;

}

//And I've gone ahead and just covered the canvas

//with a thin layer of Black Gesso

else {

red = 0;

green = 0;

blue = 0;

}

COLORREF COLOR = RGB(red, green, blue);

SetPixel(mydc, col, row, COLOR);

}

}

}

/\*

Calculates how much "stuff" to move using a simple proportionality rule.

More sophisticated rules (differential equations) could also be used.

Moves "stuff" between a target and a neighbor cell.

And just blend it all together with light, vertical brush strokes.

Just be ever so gentle.

\*/

void transfer(int tRow, int tCol, int nRow, int nCol)

{

float dQ; // flow quantity difference

// calculate how much Q to move (simple proportionality)

dQ = K \* (Q[tRow][tCol] - Q[nRow][nCol]);

// Move the stuff

Q[tRow][tCol] -= dQ; // adjust target

Q[nRow][nCol] += dQ; // adjust neighbor

}

/\*

Iterates thru elements of the Q matrix and updates the stuff in each element

\*/

void updateQ()

{

// for each target element

for (int row = 0; row < MAX; row++)

{

for (int col = 0; col < MAX; col++)

{

// NORTH

if (!B[row][col] && !B[row - 1][col]) // if target and its neighbor are not blocked

transfer(row, col, row - 1, col); // move stuff

// EAST

if (!B[row][col] && !B[row][col + 1])

transfer(row, col, row, col + 1);

// SOUTH

if (!B[row][col] && !B[row + 1][col])

transfer(row, col, row + 1, col);

// WEST

if (!B[row][col] && !B[row][col - 1])

transfer(row, col, row, col - 1);

}

}

}

void clearscreen(void)

{

COLORREF WHITE = RGB(255, 255, 255);

for (int y = 0; y < MAX; y++)

for (int x = 0; x < MAX; x++)

SetPixel(mydc, x, y, WHITE);

}

void initB()

{

// borders

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

{

B[0][i] = 1;

B[MAX - 1][i] = 1;

B[i][0] = 1;

B[i][MAX - 1] = 1;

}

//The Starting Color Bands for Nitrogen and Oxygen

for (int col = 0; col < (MAX - 1); col++)

{

//nitrogen

for (int row = 0; row < (MAX/2 - 1); row++)

{

if (row < (MAX / 8))

Q[row][col] = SEED;

else if (row < (MAX / 4))

Q[row][col] = SEED2;

else if (row < (MAX \* 3/8))

Q[row][col] = SEED3;

}

//oxygen

for (int row = (MAX/2 + 1); row < (MAX - 1); row++)

{

if (row < (MAX \* 5 / 8))

Q[row][col] = SEED4;

else if (row < (MAX \* 3 / 4))

Q[row][col] = SEED5;

else if (row < (MAX \* 7 / 8))

Q[row][col] = SEED6;

}

}

// Divide the bands

for (int row = (MAX/2 - 1); row < (MAX/2 + 1); row++)

{

for (int col = 0; col < MAX; col++)

{

B[row][col] = 1;

}

}

}

int main()

{

// Get a console handle

myconsole = GetConsoleWindow();

// Get a handle to device context

mydc = GetDC(myconsole);

// White out the screen

clearscreen();

initB(); // be carefull not to overwrite array bounds when changing MAX

// do it forever

for (int i = 0; 1; i++)

{

drawQ();

updateQ();

}

ReleaseDC(myconsole, mydc);

cin.ignore();

}

