Assignment 15

Finite Limit Elements

```
/*
    SSFEA (Super Simple Finite Element Analysis)

    THE PIXEL BASED VERSION

    April 2016 3:00 AM
    Dan Ross

    flowQuantity = K * (Q1 - Q2)

    K = ProportionalityConstant

    Implements a PIXEL output.
      If you make changes, you should start with the TEXT version then
      port your changes to the pixel version.

*/

#include <windows.h>
#include <iostream>
#include <cmath>
#include <stdlib.h>
#include <time.h> // Include for srand

using namespace std;

HDC mydc;
HWND myconsole;

#define MAX 100                 // size of the matrix
#define CUBE_SIZE 10    // size of the cube
#define SPEED 1         // speed of the cube

// the data
// color representation: 0 for blue background, 1 for red cube
int color[MAX][MAX];

// pixel output
void draw()
{
    for (int row = 0; row < MAX; row++)
    {
        for (int col = 0; col < MAX; col++)
        {
            COLORREF COLOR;
            COLOR = RGB(100, 255 - color[row][col], 0); // Gradient effect from blue
to green
            SetPixel(mydc, col, row, COLOR);
            COLOR = RGB(0, 0, 255);
        }
    }
}

void clearscreen(void)
{
    for (int y = 0; y < MAX; y++)
        for (int x = 0; x < MAX; x++)
```

```
            color[y][x] = 255 * y / MAX; // Smooth transition from blue (255) to
green (0)
}

// Function to simulate the wave-like propagation of color change
void propagateColor(int row, int col, int value)
{
    if (row >= 0 && row < MAX && col >= 0 && col < MAX && color[row][col] > value) {
        color[row][col] = value;
        propagateColor(row - 12, col, value + 12); // Propagate the color change
upward
        propagateColor(row + 12, col, value + 12); // Propagate the color change
downward
        propagateColor(row, col - 12, value + 12); // Propagate the color change to
the left
        propagateColor(row, col + 12, value + 12); // Propagate the color change to
the right
    }
}

int main()
{
    // Get a console handle
    myconsole = GetConsoleWindow();
    // Get a handle to device context
    mydc = GetDC(myconsole);
    // White out the screen
    clearscreen();

    srand(time(NULL)); // Initialize random seed

    // Initial position and direction of the first cube
    int cubeX1 = MAX / 2;
    int cubeY1 = MAX / 2;
    int dirX1 = rand() % 3 - 1; // Random direction (-1, 0, 1)
    int dirY1 = rand() % 3 - 1;

    // Initial position and direction of the second cube
    int cubeX2 = MAX / 4;
    int cubeY2 = MAX / 4;
    int dirX2 = rand() % 3 - 1;
    int dirY2 = rand() % 3 - 1;

    // Initial position and direction of the third cube
    int cubeX3 = MAX * 3 / 4;
    int cubeY3 = MAX * 3 / 4;
    int dirX3 = rand() % 3 - 1;
    int dirY3 = rand() % 3 - 1;

    // do it forever
    for (int i = 0; 1; i++)
    {
        clearscreen(); // Clear the screen for each frame

        // Update first cube position
        cubeX1 += dirX1 * SPEED;
        cubeY1 += dirY1 * SPEED;
```

```
// Update second cube position
cubeX2 += dirX2 * SPEED;
cubeY2 += dirY2 * SPEED;

// Update third cube position
cubeX3 += dirX3 * SPEED;
cubeY3 += dirY3 * SPEED;

// Check boundaries to make the cubes bounce and handle collisions
if (cubeX1 <= 0 || cubeX1 >= MAX - CUBE_SIZE)
    dirX1 *= -1; // Change direction on hitting the boundaries
if (cubeY1 <= 0 || cubeY1 >= MAX - CUBE_SIZE)
    dirY1 *= -1;

if (cubeX2 <= 0 || cubeX2 >= MAX - CUBE_SIZE)
    dirX2 *= -1;
if (cubeY2 <= 0 || cubeY2 >= MAX - CUBE_SIZE)
    dirY2 *= -1;

if (cubeX3 <= 0 || cubeX3 >= MAX - CUBE_SIZE)
    dirX3 *= -1;
if (cubeY3 <= 0 || cubeY3 >= MAX - CUBE_SIZE)
    dirY3 *= -1;

// Check for collisions between cubes and change direction if they touch
if (abs(cubeX1 - cubeX2) < CUBE_SIZE && abs(cubeY1 - cubeY2) < CUBE_SIZE) {
    dirX1 *= -1;
    dirY1 *= -1;
    dirX2 *= -1;
    dirY2 *= -1;
}

if (abs(cubeX1 - cubeX3) < CUBE_SIZE && abs(cubeY1 - cubeY3) < CUBE_SIZE) {
    dirX1 *= -1;
    dirY1 *= -1;
    dirX3 *= -1;
    dirY3 *= -1;
}

if (abs(cubeX2 - cubeX3) < CUBE_SIZE && abs(cubeY2 - cubeY3) < CUBE_SIZE) {
    dirX2 *= -1;
    dirY2 *= -1;
    dirX3 *= -1;
    dirY3 *= -1;
}

// Draw the first cube
for (int row = cubeY1; row < cubeY1 + CUBE_SIZE; row++)
{
    for (int col = cubeX1; col < cubeX1 + CUBE_SIZE; col++)
    {
        color[row][col] = 1; // Set pixels within cube bounds to red
    }
}

// Draw the second cube
for (int row = cubeY2; row < cubeY2 + CUBE_SIZE; row++)
{
```

```
            for (int col = cubeX2; col < cubeX2 + CUBE_SIZE; col++)
            {
                color[row][col] = 1; // Set pixels within cube bounds to red
            }
        }

        // Draw the third cube
        for (int row = cubeY3; row < cubeY3 + CUBE_SIZE; row++)
        {
            for (int col = cubeX3; col < cubeX3 + CUBE_SIZE; col++)
            {
                color[row][col] = 1; // Set pixels within cube bounds to red
            }
        }

        // Propagate color change from the position of the cubes
        propagateColor(cubeY1, cubeX1, 0);
        propagateColor(cubeY2, cubeX2, 0);
        propagateColor(cubeY3, cubeX3, 0);



draw();

        // Pause to control frame rate (adjust as needed)
        Sleep(50);

    }

    ReleaseDC(myconsole, mydc);
    cin.ignore();
}
```