Fly Catcher

**M5X01038 / M2X8832**

*Lecturer:* Andrew Baker

*Author:* Michael Thomas Rhys Johns

P147264

*Date:* 27/04/2017

# Contents

[Contents 2](#_Toc481039107)

[Introduction 3](#_Toc481039108)

[1.1 Running the Software 3](#_Toc481039109)

[Changing fly amount 3](#_Toc481039110)

[Enabling rotation, scaling and transformation 4](#_Toc481039111)

[2. Code – Header Files 4](#_Toc481039112)

[2.1 – Calculations.h 4](#_Toc481039113)

[2.2 – cRenderClass.h 5](#_Toc481039114)

[2.3 – FlyManager 6](#_Toc481039115)

[2.4 – globals.h 7](#_Toc481039116)

[2.5 – Matrix.h 7](#_Toc481039117)

[2.6 – stdafx.h 8](#_Toc481039118)

[2.7 - Vector.h 9](#_Toc481039119)

[3. Code - .CPP Files 9](#_Toc481039120)

[3.1 – Calculations.cpp 9](#_Toc481039121)

[3.2 – cRenderClass.cpp 14](#_Toc481039122)

[3.3 – FlyManager 29](#_Toc481039123)

[3.4 – GraphicsTemplate.cpp 31](#_Toc481039124)

[3.5 – Matrix.cpp 37](#_Toc481039125)

[3.6 – stdafx.cpp 41](#_Toc481039126)

[3.7 – Vector.cpp 41](#_Toc481039127)

[Conclusion 42](#_Toc481039128)

[References 43](#_Toc481039129)

# Introduction

This document includes a copy of the code created while making a custom maths library, implementation of the library and functions. Included in the custom Math library are several functions including, custom line drawing algorithms, circle drawing, slope calculations and line equation functions and a custom convex hull using Jarvis march.

# 1.1 Running the Software

To run the software, start up the file “GraphicsTemplate.sln” within the main folder.

### Changing fly amount

To change fly amount, change the value within FlyManager.cpp line 6.



### Enabling rotation, scaling and transformation

To enable rotation, scaling and transform function, uncomment the lines in the main GraphisTemplate.cpp.

# 2. Code – Header Files

This section of the report will contain source code copied directly from visual studio assignment project.

## 2.1 – Calculations.h

#pragma once

class Calculations

{

private:

const float PI = 3.141592f;

public:

Calculations();

~Calculations();

void init();

Vector \*vectorResult;

Matrix \*translation;

Matrix \*scale;

Matrix \*rotate;

Matrix \*result;

Vector \*vector;

int convert;

float sinLUT[1024];

float cosLUT[1024];

void multiplyMatrixByMatrix(Matrix \*, Matrix \*);

void multiply4By4byVector(Matrix \*, Vector \*);

void translate(float, float, float, Vector \*);

void rotateX(float, Vector \*);

void rotateY(float, Vector \*);

void rotateZ(float, Vector \*);

void scaleUniform(float, Vector \*);

void scaleNonUniform(float, float, float, float, Vector \*);

float degreesToRadians(float);

float radiansToDegrees(float);

float dotProduct2D(Vector \*, Vector \*);

void createSinCosLUT();

};

## 2.2 – cRenderClass.h

class cRenderClass

{

public:

cRenderClass(const int, const int);

~cRenderClass();

void create(int argc, \_TCHAR\* argv[]);

void loop();

void clear();

void render();

void setColour(float, float, float);

void setPointSize(int);

void drawPixel(int, int);

// Work out line equation

float lineEquation(float, float, float, float);

float lineSlope(float, float, float, float);

// Line Draw functions

void lineDirectScan(float, float, float, float);

void lineDDA(float, float, float, float);

void lineBresenhams(float, float, float, float);

// Circle Draw

void lineCircle(float, float, float);

// Jarvis March Convex Hull

FlyManager\* hull = nullptr;

int orientation(Vector p, Vector q, Vector r);

void jarvisMarchHull(FlyManager \*);

private:

int m\_sw, m\_sh;

int m\_point\_size;

float \*m\_buffer;

sRGB m\_colour;

// initialise maths variables

float m = 0.0f;

float b = 0.0f;

float step = 0.0f;

float y = 0.0f;

};

## 2.3 – FlyManager

#pragma once

class FlyManager

{

public:

int size;

int leftFlyIndex;

Vector \*flyArray;

Vector \*direction;

FlyManager(int);

bool createFlyArray(int);

void setFlyVectors();

void moveDirection();

int leftFly();

~FlyManager();

};

## 2.4 – globals.h

////////////////////////////////////////////////////////////////////////////////

// externals

////////////////////////////////////////////////////////////////////////////////

extern void winReshapeFunc(GLint w, GLint h);

extern void renderScene();

extern void update();

////////////////////////////////////////////////////////////////////////////////

// constants

////////////////////////////////////////////////////////////////////////////////

const int SCREEN\_WIDTH = 800;

const int SCREEN\_HEIGHT = 800;

////////////////////////////////////////////////////////////////////////////////

// structures

////////////////////////////////////////////////////////////////////////////////

struct sRGB

{

float r, g, b;

};

## 2.5 – Matrix.h

#pragma once

class Matrix

{

public:

// array to store matrix values for object

float\* matrixArray;

// Default Matrix constructor

Matrix();

// 3x1 Matrix Constructor

Matrix(float, float, float);

// 2x2 Matrix Constructor

Matrix(float, float, float, float);

// 3x3 Matrix Constructor

Matrix(float, float, float, float, float, float, float, float, float);

// 4x4 Matrix Constructor

Matrix(float, float, float, float, float, float, float, float, float, float, float, float, float, float, float, float);

// Matrix Destructor

~Matrix();

void identityMatrix();

// initial matrix creator

bool createMatrix(int);

void print();

};

## 2.6 – stdafx.h

// stdafx.h : include file for standard system include files,

// or project specific include files that are used frequently, but

// are changed infrequently

//

#pragma once

#include <iostream>

using std::endl;

using std::cout;

#include <tchar.h>

#include <windows.h>

#include <math.h>

#include "Matrix.h"

#include "Vector.h"

#include "Calculations.h"

#include "FlyManager.h"

#include "glut.h"

#include "globals.h"

#include "cRenderClass.h"

## 2.7 - Vector.h

#pragma once

class Vector

{

public:

float x, y, z, w;

Vector();

Vector(float, float, float, float);

~Vector();

void print();

};

# 3. Code - .CPP Files

This section of the report will contain source code copied directly from visual studio assignment project.

## 3.1 – Calculations.cpp

#include "stdafx.h"

Calculations calc;

Calculations calc1;

Calculations calc2;

Calculations::Calculations()

{

this->init();

}

Calculations::~Calculations()

{

delete this->result;

this->vectorResult;

this->translation;

this->scale;

this->rotate;

}

// initialise objects (should be in its own function as nothing that could fail directly in constructor)

void Calculations::init()

{

this->result = new Matrix();

this->vectorResult = new Vector();

this->translation = new Matrix();

this->scale = new Matrix();

this->rotate = new Matrix();

createSinCosLUT();

}

void Calculations::multiplyMatrixByMatrix(Matrix\* matrix1, Matrix\* matrix2)

{

// Row 1 of resulting Matrix

this->result->matrixArray[0] = (matrix1->matrixArray[0] \* matrix2->matrixArray[0]) + (matrix1->matrixArray[1] \* matrix2->matrixArray[4]) + (matrix1->matrixArray[2] \* matrix2->matrixArray[8]) + (matrix1->matrixArray[3] \* matrix2->matrixArray[12]);

this->result->matrixArray[1] = (matrix1->matrixArray[0] \* matrix2->matrixArray[1]) + (matrix1->matrixArray[1] \* matrix2->matrixArray[5]) + (matrix1->matrixArray[2] \* matrix2->matrixArray[9]) + (matrix1->matrixArray[3] \* matrix2->matrixArray[13]);

this->result->matrixArray[2] = (matrix1->matrixArray[0] \* matrix2->matrixArray[2]) + (matrix1->matrixArray[1] \* matrix2->matrixArray[6]) + (matrix1->matrixArray[2] \* matrix2->matrixArray[10]) + (matrix1->matrixArray[3] \* matrix2->matrixArray[14]);

this->result->matrixArray[3] = (matrix1->matrixArray[0] \* matrix2->matrixArray[3]) + (matrix1->matrixArray[1] \* matrix2->matrixArray[7]) + (matrix1->matrixArray[2] \* matrix2->matrixArray[11]) + (matrix1->matrixArray[3] \* matrix2->matrixArray[15]);

// Row 2 of resulting Matrix

this->result->matrixArray[4] = (matrix1->matrixArray[4] \* matrix2->matrixArray[0]) + (matrix1->matrixArray[5] \* matrix2->matrixArray[4]) + (matrix1->matrixArray[6] \* matrix2->matrixArray[8]) + (matrix1->matrixArray[7] \* matrix2->matrixArray[12]);

this->result->matrixArray[5] = (matrix1->matrixArray[4] \* matrix2->matrixArray[1]) + (matrix1->matrixArray[5] \* matrix2->matrixArray[5]) + (matrix1->matrixArray[6] \* matrix2->matrixArray[9]) + (matrix1->matrixArray[7] \* matrix2->matrixArray[13]);

this->result->matrixArray[6] = (matrix1->matrixArray[4] \* matrix2->matrixArray[2]) + (matrix1->matrixArray[5] \* matrix2->matrixArray[6]) + (matrix1->matrixArray[6] \* matrix2->matrixArray[10]) + (matrix1->matrixArray[7] \* matrix2->matrixArray[14]);

this->result->matrixArray[7] = (matrix1->matrixArray[4] \* matrix2->matrixArray[3]) + (matrix1->matrixArray[5] \* matrix2->matrixArray[7]) + (matrix1->matrixArray[6] \* matrix2->matrixArray[11]) + (matrix1->matrixArray[7] \* matrix2->matrixArray[15]);

// Row 3 of resulting Matrix

this->result->matrixArray[8] = (matrix1->matrixArray[8] \* matrix2->matrixArray[0]) + (matrix1->matrixArray[9] \* matrix2->matrixArray[4]) + (matrix1->matrixArray[10] \* matrix2->matrixArray[8]) + (matrix1->matrixArray[11] \* matrix2->matrixArray[12]);

this->result->matrixArray[9] = (matrix1->matrixArray[8] \* matrix2->matrixArray[1]) + (matrix1->matrixArray[9] \* matrix2->matrixArray[5]) + (matrix1->matrixArray[10] \* matrix2->matrixArray[9]) + (matrix1->matrixArray[11] \* matrix2->matrixArray[13]);

this->result->matrixArray[10] = (matrix1->matrixArray[8] \* matrix2->matrixArray[2]) + (matrix1->matrixArray[9] \* matrix2->matrixArray[6]) + (matrix1->matrixArray[10] \* matrix2->matrixArray[10]) + (matrix1->matrixArray[11] \* matrix2->matrixArray[14]);

this->result->matrixArray[11] = (matrix1->matrixArray[8] \* matrix2->matrixArray[3]) + (matrix1->matrixArray[9] \* matrix2->matrixArray[7]) + (matrix1->matrixArray[10] \* matrix2->matrixArray[11]) + (matrix1->matrixArray[11] \* matrix2->matrixArray[15]);

// Row 4 of resulting Matrix

this->result->matrixArray[12] = (matrix1->matrixArray[12] \* matrix2->matrixArray[0]) + (matrix1->matrixArray[13] \* matrix2->matrixArray[4]) + (matrix1->matrixArray[14] \* matrix2->matrixArray[8]) + (matrix1->matrixArray[15] \* matrix2->matrixArray[12]);

this->result->matrixArray[13] = (matrix1->matrixArray[12] \* matrix2->matrixArray[1]) + (matrix1->matrixArray[13] \* matrix2->matrixArray[5]) + (matrix1->matrixArray[14] \* matrix2->matrixArray[9]) + (matrix1->matrixArray[15] \* matrix2->matrixArray[13]);

this->result->matrixArray[14] = (matrix1->matrixArray[12] \* matrix2->matrixArray[2]) + (matrix1->matrixArray[13] \* matrix2->matrixArray[6]) + (matrix1->matrixArray[14] \* matrix2->matrixArray[10]) + (matrix1->matrixArray[15] \* matrix2->matrixArray[14]);

this->result->matrixArray[15] = (matrix1->matrixArray[12] \* matrix2->matrixArray[3]) + (matrix1->matrixArray[13] \* matrix2->matrixArray[7]) + (matrix1->matrixArray[14] \* matrix2->matrixArray[11]) + (matrix1->matrixArray[15] \* matrix2->matrixArray[15]);

this->result->print();

}

void Calculations::multiply4By4byVector(Matrix \*matrix, Vector \*vector)

{

// Row 1 of resulting Matrix

this->vectorResult->x = ((matrix->matrixArray[0] \* vector->x) + (matrix->matrixArray[1] \* vector->y) + (matrix->matrixArray[2] \* vector->z) + (matrix->matrixArray[3] \* vector->w));

this->vectorResult->y = ((matrix->matrixArray[4] \* vector->x) + (matrix->matrixArray[5] \* vector->y) + (matrix->matrixArray[6] \* vector->z) + (matrix->matrixArray[7] \* vector->w));

this->vectorResult->z = ((matrix->matrixArray[8] \* vector->x) + (matrix->matrixArray[9] \* vector->y) + (matrix->matrixArray[10] \* vector->z) + (matrix->matrixArray[11] \* vector->w));

this->vectorResult->w = ((matrix->matrixArray[12] \* vector->x) + (matrix->matrixArray[13] \* vector->y) + (matrix->matrixArray[14] \* vector->z) + (matrix->matrixArray[15] \* vector->w));

\*vector = \*vectorResult;

}

void Calculations::translate(float x, float y, float z, Vector \*vector)

{

this->translation->identityMatrix();

this->translation->matrixArray[3] = x;

this->translation->matrixArray[7] = y;

this->translation->matrixArray[11] = z;

multiply4By4byVector(this->translation, vector);

}

void Calculations::rotateX(float angle, Vector \*vector)

{

this->convert = angle \* 2.84444444;

this->rotate->identityMatrix();

this->rotate->matrixArray[5] = (this->cosLUT[convert]);

this->rotate->matrixArray[6] = (this->sinLUT[convert]);

this->rotate->matrixArray[9] = (-(this->sinLUT[convert]));

this->rotate->matrixArray[10] = (this->cosLUT[convert]);

multiply4By4byVector(this->rotate, vector);

}

void Calculations::rotateY(float angle, Vector \*vector)

{

this->convert = angle \* 2.84444444;

this->rotate->identityMatrix();

this->rotate->matrixArray[0] = (this->cosLUT[convert]);

this->rotate->matrixArray[2] = ((-(this->sinLUT[convert])));

this->rotate->matrixArray[8] = (this->sinLUT[convert]);

this->rotate->matrixArray[10] = (this->cosLUT[convert]);

multiply4By4byVector(this->rotate, vector);

}

void Calculations::rotateZ(float angle, Vector \*vector)

{

this->convert = angle \* 2.84444444;

this->rotate->identityMatrix();

this->rotate->matrixArray[0] = (this->cosLUT[convert]);

this->rotate->matrixArray[1] = (this->sinLUT[convert]);

this->rotate->matrixArray[4] = (-(this->sinLUT[convert]));

this->rotate->matrixArray[5] = (this->cosLUT[convert]);

multiply4By4byVector(this->rotate, vector);

}

void Calculations::scaleUniform(float w, Vector \*vector)

{

this->scale->identityMatrix();

this->scale->matrixArray[15] = w;

this->multiply4By4byVector(this->scale, vector);

vector->x = (vector->x / vector->w);

vector->y = (vector->y / vector->w);

vector->z = (vector->z / vector->w);

vector->w = (vector->w / vector->w);

}

void Calculations::scaleNonUniform(float x, float y, float z, float w, Vector \*vector)

{

this->scale->identityMatrix();

this->scale->matrixArray[0] = x;

this->scale->matrixArray[5] = y;

this->scale->matrixArray[10] = z;

this->scale->matrixArray[15] = w;

this->multiply4By4byVector(this->scale, vector);

vector->x = (vector->x / vector->w);

vector->y = (vector->y / vector->w);

vector->z = (vector->z / vector->w);

vector->w = (vector->w / vector->w);

}

float Calculations::degreesToRadians(float degrees)

{

//cout << (degrees \* (this->PI / 180)) << endl;

return (degrees \* (this->PI / 180));

}

float Calculations::radiansToDegrees(float radians)

{

return (radians \* (180 / this->PI));

}

// Procedure to find the dot product of two vectors

float Calculations::dotProduct2D(Vector \*vector1, Vector \*vector2)

{

// Multiply the first vector by the second vector

// Add each above answer together

return (vector1->x \* vector2->x +

vector1->y \* vector2->y);

}

void Calculations::createSinCosLUT()

{

sinLUT[0] = 0.00436331; ……. Refer to source file (omitted from report due to length)

}

## 3.2 – cRenderClass.cpp

/////////////////////////////////////////////////////////////////////////////////

// cRenderClass.cpp

/////////////////////////////////////////////////////////////////////////////////

#include "stdafx.h"

/////////////////////////////////////////////////////////////////////////////////

// Declarations

/////////////////////////////////////////////////////////////////////////////////

cRenderClass graphics( SCREEN\_WIDTH, SCREEN\_HEIGHT );

extern Calculations calc;

/////////////////////////////////////////////////////////////////////////////////

// cRenderClass() - constructor

/////////////////////////////////////////////////////////////////////////////////

cRenderClass::cRenderClass(const int w, const int h)

{

m\_sw = w;

m\_sh = h;

int total\_count = w\*h\*3;

m\_buffer = new float[total\_count];

for( int i=0; i<total\_count; i++ )

{

m\_buffer[i] = 0;

}

m\_point\_size = 1;

}

/////////////////////////////////////////////////////////////////////////////////

// cRenderClass() - destructor

/////////////////////////////////////////////////////////////////////////////////

cRenderClass::~cRenderClass()

{

}

/////////////////////////////////////////////////////////////////////////////////

// loop() - enters game loop

/////////////////////////////////////////////////////////////////////////////////

void cRenderClass::loop()

{

glutMainLoop();

}

/////////////////////////////////////////////////////////////////////////////////

// initialize glut stuff

/////////////////////////////////////////////////////////////////////////////////

void cRenderClass::create(int argc, \_TCHAR\* argv[])

{

// initialise the glut library

glutInit(&argc, argv);

// set up the initial display mode

// need both double buffering and z-buffering

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

// set the initial window position

glutInitWindowPosition(100, 100);

// set the initial window size

glutInitWindowSize(SCREEN\_WIDTH, SCREEN\_HEIGHT);

// create and name the window

glutCreateWindow("Wow!");

// reshape callback for current window

glutReshapeFunc(winReshapeFunc);

// set display callback for current window

glutDisplayFunc(renderScene);

// set up the global idle callback

glutIdleFunc(update);

}

/////////////////////////////////////////////////////////////////////////////////

// clear() - clears out the render buffer

/////////////////////////////////////////////////////////////////////////////////

void cRenderClass::clear()

{

memset( m\_buffer, 0, sizeof(float)\*m\_sw\*m\_sh\*3 );

}

/////////////////////////////////////////////////////////////////////////////////

// setPointSize() - set the pixel size

////////////////////////////////////////////////////////////////////////////////

void cRenderClass::setPointSize(int size)

{

m\_point\_size = size;

}

/////////////////////////////////////////////////////////////////////////////////

// drawPixel() -

/////////////////////////////////////////////////////////////////////////////////

void cRenderClass::drawPixel(int x, int y)

{

int start, end;

start = -(m\_point\_size/2);

end = (int)((m\_point\_size/2.0f) + 0.5);

for( int i=start; i<end; i++ )

{

for( int j=start; j<end; j++ )

{

if( (x+j < 0) || (x+j >= m\_sw) )

continue;

if( (y+i < 0) || (y+i >= m\_sh) )

continue;

m\_buffer[(((y+i)\*m\_sw+(x+j))\*3) + 0] = m\_colour.r;

m\_buffer[(((y+i)\*m\_sw+(x+j))\*3) + 1] = m\_colour.g;

m\_buffer[(((y+i)\*m\_sw+(x+j))\*3) + 2] = m\_colour.b;

}

}

}

/////////////////////////////////////////////////////////////////////////////////

// Line Equation calculation

/////////////////////////////////////////////////////////////////////////////////

float cRenderClass::lineEquation(float p0X, float p0Y, float p1X, float p1Y)

{

// calculate slope

m = (p1Y - p0Y) / (p1X - p0X);

// calculate incept point of Y

b = p0Y - m \* p0X;

//cout << "slope = " << m << ", intercept = " << b << endl;

// return equation of a straight line (y = mx + b)

return (m\*p0X+b);

}

/////////////////////////////////////////////////////////////////////////////////

// Line slope calculation

/////////////////////////////////////////////////////////////////////////////////

float cRenderClass::lineSlope(float p0X, float p0Y, float p1X, float p1Y)

{

// calculate slope

m = (p1Y - p0Y) /(p1X - p0X);

return m;

}

/////////////////////////////////////////////////////////////////////////////////

// Direct scan line draw algorithm function - horizontal lines only

/////////////////////////////////////////////////////////////////////////////////

void cRenderClass::lineDirectScan(float p0X, float p0Y, float p1X, float p1Y)

{

if (p0X <= p1X)

{

step = p0X;

while (step <= p1X)

{

//cout << "Step = " << step << ", P0X = " << p0X << endl;

y = lineEquation(step, p0Y, p1X, p1Y);

//cout << "Y = " << y << endl;

drawPixel(step, y);

step++;

}

}

else

{

step = p1X;

while (step < p0X)

{

//cout << "Step = " << step << ", P1X = " << p1X << endl;

y = lineEquation(step, p0Y, p1X, p1Y);

//cout << "pixel position (" << step << ", " << y << ")" << endl;

drawPixel(step, y);

step++;

}

}

}

/////////////////////////////////////////////////////////////////////////////////

// DDA Line drawing algorithm function (currently left to right only on points and positive slopes)

/////////////////////////////////////////////////////////////////////////////////

void cRenderClass::lineDDA(float p0X, float p0Y, float p1X, float p1Y)

{

float step = 1;

// Calculate line equation ( m = slope, b = Y intercept)

lineEquation(p0X, p0Y, p1X, p1Y);

// Calculate deltas

float dy = p1Y - p0Y;

float dx = p1X - p0X;

dy = m \* dx;

dx = dy / m;

float x = p0X;

float y = p0Y;

// if slope (m) is less than or equal to 1

if (m <= 1.0f)

{

while (x < p1X)

{

drawPixel(x, y);

y = y + m;

x++;

}

}

// If slope is greater than 1 ( use y increment value instead of x)

else

{

while (y < p1Y)

{

drawPixel(x, y);

x = x + (1 / m);

y++;

}

}

}

/////////////////////////////////////////////////////////////////////////////////

// Bresenham's line drawing algorithm

/////////////////////////////////////////////////////////////////////////////////

void cRenderClass::lineBresenhams(float p0X, float p0Y, float p1X, float p1Y)

{

int x, y, xEnd, yEnd;

int i = 0;

// Calculate deltas

int dx = p1X - p0X; // length of line on X

int dy = p1Y - p0Y; // length of line on Y

// absolute values to check length even with negative length

int dxAbs; // = fabs(dx);

int dyAbs; // = fabs(dy);

// if less than zero, convert to absolute (positive) value to compare length

if (dx < 0)

{

dxAbs = (dx)-(dx + dx);

//cout << "dxAbs = " << dxAbs << endl;

}

else

{

dxAbs = dx;

//cout << "dxAbs = " << dxAbs << endl;

}

if (dy < 0)

{

dyAbs = (dy)-(dy + dy);

//cout << "dyAbs = " << dyAbs << endl;

}

else

{

dyAbs = dy;

//cout << "dyAbs = " << dyAbs << endl;

}

int dy2Minusdx2 = 2 \* (dyAbs - dxAbs);

int dx2Minusdy2 = 2 \* (dxAbs - dyAbs);

// starting value for decision parameter (error value check on line)

int parameterX = 2 \* dyAbs - dxAbs; // for values with x axis being used as decision parameter

int parameterY = 2 \* dxAbs - dyAbs; // for values with y axis being used as decision parameter

// if y step is smaller or equal to x step

// then step along x

if (dyAbs <= dxAbs)

{

// if x step is negative

if (dx < 0)

{

x = p1X;

y = p1Y;

// xEnd will be left of starting pixel

xEnd = p0X;

}

// else x step is postive

else

{

x = p0X;

y = p0Y;

// xEnd will be right of starting pixel

xEnd = p1X;

}

// draw first pixel

drawPixel(x, y);

//cout << "PX = " << parameterX << endl;

//cout << "( " << x << ", " << y << ")" << endl;

// step through from x to xEnd point

for (i = 0; x < xEnd; i++)

{

x++;

// if x decision parameter is less than zero

if (parameterX < 0)

{

//cout << "parameterX < 0 --" << endl;

// decision parameter now equals previous parameterX + deltaY\*2

parameterX += 2 \* dyAbs;

}

// if dx and dy are anything other than zero

else

{

if ((dx<0 && dy<0) || (dx>0 && dy>0))

{

//cout << "dx dy not equal zero --" << endl;

// next plot is (x, y + 1)

y++;

}

else

{

//cout << "parameterX > 0 or higher --" << endl;

// next plot is (x, y - 1)

y--;

}

// decision parameter now equals previous parameterX + deltaY\*2 - DeltaX\*2

parameterX += dy2Minusdx2;

}

//cout << "PX after = " << parameterX << endl;

//cout << "( " << x << ", " << y << ")" << endl;

drawPixel(x, y);

}

}

// else x step is larger than y step

// same as above but for decision parameter to be performed on Y axis

else

{

// check if y step is positive or negative and set end point

if (dy < 0)

{

x = p1X;

y = p1Y;

yEnd = p0Y;

}

else

{

x = p0X;

y = p0Y;

yEnd = p1Y;

}

// draw first pixel

drawPixel(x, y);

// while y is not at right hand point of line

for (i = 0; y < yEnd; i++)

{

//cout << "else y++ " << endl;

// increment y each step

y++;

// if y decision parameter is less than or equal to zero

if (parameterY <= 0)

{

//cout << "parameterY <= 0 --" << endl;

// decision parameter now equals previous parameterY + deltaX\*2

parameterY += 2 \* dxAbs;

}

// if dx and dy are anything other than zero

else

{ // else / if indentation matters due to parameter being set

// - this took a while to notice...

if ((dx<0 && dy<0) || (dx>0 && dy>0))

{

//cout << "dx dy not equal zero -ELSE-" << endl;

// next plot is (x + 1, y)

x++;

}

else

{

//cout << "parameterY > 0 or higher --" << endl;

// next plot is (x - 1, y)

x--;

}

// decision parameter now equals previous parameterY + deltaX\*2 - DeltaY\*2

parameterY += dx2Minusdy2;

}

//cout << "else ( " << x << ", " << y << ")" << endl;

// draw decided pixel

drawPixel(x, y);

}

}

}

/////////////////////////////////////////////////////////////////////////////////

// Circle Drawing - Optimized for reflection

/////////////////////////////////////////////////////////////////////////////////

void cRenderClass::lineCircle(float radius, float x, float y)

{

float xNew;

float yNew;

int i;

for (i = 0; i < 360; i++)

{

xNew = x + (radius \* calc.cosLUT[i]);

yNew = y + (radius \* calc.sinLUT[i]);

drawPixel(xNew, yNew); // octant 0 ( 0 to 45 degrees) ( quadrant 0 with 360 loops)

//drawPixel(yNew, xNew); // octant 1 ( 45 to 90 degrees)

xNew = x - (radius \* calc.cosLUT[i]);

yNew = y + (radius \* calc.sinLUT[i]);

drawPixel(xNew, yNew); // octant 3 ( 135 to 180 degrees) ( quadrant 1 with 360 loops)

//drawPixel(yNew, xNew); // octant 7 ( 270 to 315 degrees)

xNew = x + (radius \* calc.cosLUT[i]);

yNew = y - (radius \* calc.sinLUT[i]);

drawPixel(xNew, yNew); // octant 8 ( 315 to 360 degrees) ( quadrant 4 with 360 loops)

//drawPixel(yNew, xNew); // octant 2 ( 90 to 135 degrees)

xNew = x - (radius \* calc.cosLUT[i]);

yNew = y - (radius \* calc.sinLUT[i]);

drawPixel(xNew, yNew); // octant 4 ( 180 to 225 degrees) ( quadrant 3 with 360 loops)

//drawPixel(yNew, xNew); // octant 5 ( 225 to 270 degrees)

/////////////////////////

// Commented draws was an attempt to get 180 loop working

// caused issues on translation of a circle

////////////////////////

//cout << "( " << xNew << ", " << yNew << ")" << endl;

}

}

///// Reference [1]

int cRenderClass::orientation(Vector p, Vector q, Vector r)

{

int val = (q.y - p.y) \* (r.x - q.x) -

(q.x - p.x) \* (r.y - q.y);

if (val == 0)

{

return 0; // colinear

}

return (val > 0) ? 1 : 2; // clock or counterclock wise

}

///// Reference [1], [2], [3], [4]

// Prints convex hull of a set of n points.

void cRenderClass::jarvisMarchHull(FlyManager\* flies)

{

int size = flies->size;

int leftFly = 0;

int p = 0;

int q = 0;

// count vector coords added to hull array

int count = 0;

// Initialize Result

if (hull == nullptr)

{

//cout << "making hull" << endl;

hull = new FlyManager(size);

}

// Calculate left most point

for (int i = 1; i < size; i++)

{

if (flies->flyArray[i].x < flies->flyArray[leftFly].x)

{

leftFly = i;

}

}

// set p as left fly index

p = leftFly;

// loop vector coords

do

{

// Add vector coord to hull array

hull->flyArray[count] = flies->flyArray[p];

// increment flies in hull count

count++;

q = (p + 1);

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

{

// If i is more counterclockwise than current q, then

// update q

if (orientation(flies->flyArray[p], flies->flyArray[i], flies->flyArray[q]) == 2)

q = i;

}

// Set p as q for next iteration

// add to hull

p = q;

} while (p != leftFly); // While we don't come to first point

// draw convex hull

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

{

// if end of array reached, connect first(most left point) to current

if (i + 1 >= count)

{

lineBresenhams(hull->flyArray[i].x, hull->flyArray[i].y, hull->flyArray[0].x, hull->flyArray[0].y);

}

else

{

lineBresenhams(hull->flyArray[i].x, hull->flyArray[i].y, hull->flyArray[i + 1].x, hull->flyArray[i + 1].y);

}

}

}

/////////////////////////////////////////////////////////////////////////////////

// render() - renders this buffer to screen

/////////////////////////////////////////////////////////////////////////////////void cRenderClass::render()

{

// clear the back buffer

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glDrawPixels( m\_sw, m\_sh, GL\_RGB, GL\_FLOAT, m\_buffer );

// swap the buffers of the current window

glutSwapBuffers();

// clear out the buffer

clear();

}

/////////////////////////////////////////////////////////////////////////////////

// setColour() - sets the current colour

/////////////////////////////////////////////////////////////////////////////////

void cRenderClass::setColour(float r, float g, float b)

{

m\_colour.r = r;

m\_colour.g = g;

m\_colour.b = b;

}

/////////////////////////////////////////////////////////////////////////////////

// winReshapeFunc() - gets called initially and whenever the window get resized

// resizing has been locked

////////////////////////////////////////////////////////////////////////////////

void winReshapeFunc(GLint w, GLint h)

{

// specify current matrix

glMatrixMode(GL\_PROJECTION);

// load an identity matrix

glLoadIdentity();

// create a projection matrix... i.e. 2D projection onto xy plane

glOrtho( -SCREEN\_WIDTH, SCREEN\_WIDTH, -SCREEN\_HEIGHT, SCREEN\_HEIGHT, -100, 100);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

// set up the viewport

glViewport

(

0, // lower left x position

0, // lower left y position

(GLsizei) SCREEN\_WIDTH, // viewport width

(GLsizei) SCREEN\_HEIGHT // viewport height

);

}

## 3.3 – FlyManager

#include "stdafx.h"

#include <stdlib.h>

#include <time.h>

// create fly objects here

FlyManager flies(10);

// Must pass in array size

FlyManager::FlyManager(int size)

{

this->size = size;

this->createFlyArray(size);

}

// Create Fly Array

bool FlyManager::createFlyArray(int size)

{

bool result = false;

this->flyArray = nullptr;

this->flyArray = new Vector[this->size];

this->direction = nullptr;

this->direction = new Vector[this->size];

if (this->flyArray != nullptr && this->direction != nullptr)

{

result = true;

}

else

{

// Result already set to false

}

return result;

}

void FlyManager::setFlyVectors()

{

// start points within the window

int spawnMaxX = SCREEN\_WIDTH - 20; // max X offset

int spawnMinX = SCREEN\_WIDTH - SCREEN\_WIDTH + 20; // Min X offset

int spawnMaxY = SCREEN\_HEIGHT - 20; // max Y offset

int spawnMinY = SCREEN\_HEIGHT - SCREEN\_HEIGHT + 20; // Min Y offset

for (int i = 0; i < this->size; i++)

flies.flyArray[i] = Vector(rand() % spawnMaxX + spawnMinX, rand() % spawnMaxY + spawnMinY, 1.0f, 1.0f);

}

// random fly velocity

void FlyManager::moveDirection()

{

srand(time(NULL));

for (int i = 0; i < this->size; i++)

{

flies.direction[i] = Vector(rand() % 3 + 1, rand() % 3 + 1, 1.0f, 1.0f);

}

}

// attempted to control left fly outside of hull.

// starts ok but currently causing unknown bug that multiplies files

int FlyManager::leftFly()

{

for (int i = 1; i < size - 1; i++)

{

if (flyArray[i].x < flyArray[leftFlyIndex].x)

leftFlyIndex = i;

//cout << "Size = " << leftFlyIndex << endl;

}

//cout << "Left fly index = " << leftFlyIndex << endl;

return leftFlyIndex;

}

FlyManager::~FlyManager()

{

delete[] this->flyArray;

delete[] this->direction;

}

## 3.4 – GraphicsTemplate.cpp

// GraphicsTemplate.cpp

/////////////////////////////////////////////////////////////////////////////////

#include "stdafx.h"

/////////////////////////////////////////////////////////////////////////////////

// externals

/////////////////////////////////////////////////////////////////////////////////extern cRenderClass graphics;

extern Calculations calc;

extern FlyManager flies;

/////////////////////////////////////////////////////////////////////////////////

// renderScene() - render the scene

/////////////////////////////////////////////////////////////////////////////////

void renderScene()

{

// set a colour

graphics.setColour( 1,0,0 );

// set pixel size..

graphics.setPointSize(3);

//graphics.drawLine(&p0, &p1);

// draw to an off screen buffer

//graphics.drawPixel(p0.x, p0.y);

//graphics.drawPixel(p1.x, p1.y);

//graphics.drawPixel(p2.x, p2.y);

//graphics.drawPixel(p3.x, p3.y);

// render the scene

graphics.render();

}

/////////////////////////////////////////////////////////////////////////////////

// update() - update function

/////////////////////////////////////////////////////////////////////////////////

void update()

{

// add any update code here...

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DRAW FLY CODE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for (int i = 0; i < flies.size; i++)

{

// draw flies as pixels

graphics.drawPixel(flies.flyArray[i].x, flies.flyArray[i].y);

// Translate flies position

calc.translate(flies.direction[i].x, flies.direction[i].y, 0, &flies.flyArray[i]);

// change direction with reflection law if sides of screen hit

if (flies.flyArray[i].x < (SCREEN\_WIDTH - SCREEN\_WIDTH + 2) ^ flies.flyArray[i].x > (SCREEN\_WIDTH - 2))

{

flies.direction[i].x = -flies.direction[i].x;

}

// change direction with reflection law if top or bottom of screen hit

if (flies.flyArray[i].y < (SCREEN\_HEIGHT - SCREEN\_HEIGHT + 2) || flies.flyArray[i].y > (SCREEN\_HEIGHT - 2))

{

flies.direction[i].y = -flies.direction[i].y;

}

// uncomment to show changes in fly direction

//cout << "flies direction[" << i << "] = (" << flies.direction[i].x << ", " << flies.direction[i].y << ")" << endl;

// uncomment to show changes in fly location

//cout << "flies location[" << i << "] = (" << flies.flyArray[i].x << ", " << flies.flyArray[i].y << ")" << endl;

//calc.rotateX(3, &flies.flyArray[i]);

//calc.rotateY(3, &flies.flyArray[i]);

//calc.rotateZ(3, &flies.flyArray[i]);

//calc.scaleUniform(0.01f, &flies.flyArray[i]);

}

// encapsulate all flies inside a convex hull

graphics.jarvisMarchHull(&flies);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* OTHER MATH LIBRARY FEATURES

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//graphics.lineCircle(100, p0.x, p0.y);

//graphics.lineEquation(p0.x, p0.y, p1.x, p1.y);

//graphics.directScan(p0.x, p0.y, p1.x, p1.y);

//graphics.directScan(p1.x, p1.y, p2.x, p2.y);

//graphics.directScan(p3.x, p3.y, p2.x, p2.y);

//graphics.directScan(p3.x, p3.y, p0.x, p0.y);

//graphics.lineDDA(p0.x, p0.y, p1.x, p1.y);

//graphics.lineDDA(p1.x, p1.y, p2.x, p2.y);

//graphics.lineDDA(p3.x, p3.y, p2.x, p2.y);

//graphics.lineDDA(p0.x, p0.y, p3.x, p3.y);

//graphics.lineBresenhams(p2.x, p2.y, p0.x, p0.y);

//graphics.lineBresenhams(p3.x, p3.y, p2.x, p2.y);

//graphics.lineBresenhams(p2.x, p2.y, p1.x, p1.y);

//graphics.lineBresenhams(p3.x, p3.y, p1.x, p1.y);

//graphics.lineBresenhams(p3.x, p3.y, p0.x, p0.y);

/\*/ Rotation

p0.x -= centre.x;

p0.y -= centre.y;

p0.z -= centre.z;

p0.w -= centre.w;

p1.x -= centre.x;

p1.y -= centre.y;

p1.z -= centre.z;

p1.w -= centre.w;

p2.x -= centre.x;

p2.y -= centre.y;

p2.z = centre.z;

p2.w -= centre.w;

p3.x -= centre.x;

p3.y -= centre.y;

p3.z -= centre.z;

p3.w -= centre.w;

//calc.rotateZ(3, &p0);

//calc.rotateZ(3, &p1);

//calc.rotateZ(3, &p2);

//calc.rotateZ(3, &p3);

calc.rotateY(1, &p0);

calc.rotateY(1, &p1);

calc.rotateY(1, &p2);

calc.rotateY(1, &p3);

calc.rotateX(5, &p0);

calc.rotateX(5, &p1);

calc.rotateX(5, &p2);

calc.rotateX(5, &p3);

calc.rotateX(5, &centre);

p0.x += centre.x;

p0.y += centre.y;

p0.z += centre.z;

p0.w += centre.w;

p1.x += centre.x;

p1.y += centre.y;

p1.z += centre.z;

p1.w += centre.w;

p2.x += centre.x;

p2.y += centre.y;

p2.z += centre.z;

p2.w += centre.w;

p3.x += centre.x;

p3.y += centre.y;

p3.z += centre.z;

p3.w += centre.w;

\*/

/\*

// Transform

calc.translate(0, 1, 0, &p0);

calc.translate(1, 0, 0, &p1);

calc.translate(1, 0, 0, &p2);

calc.translate(1, 0, 0, &p3);

\*/

/\*

calc.scaleNonUniform(1, 1, 1, 1, &p0);

calc.scaleNonUniform(0.999, 1, 1, 1, &p1);

calc.scaleNonUniform(1, 1, 1, 1, &p2);

calc.scaleNonUniform(0.999, 1, 1, 1, &p3);

\*/

/\*

calc.scaleUniform(-0.999, &p0);

calc.scaleUniform(-0.999, &p1);

calc.scaleUniform(-0.999, &p2);

calc.scaleUniform(-0.999, &p3);

calc.scaleUniform(-0.999, &centre);

\*/

/\*

//p0.print();

//p1.print();

//p2.print();

//p3.print();

\*/

// always re-render the scene..

renderScene();

}

/////////////////////////////////////////////////////////////////////////////////

// \_tmain() - program entry point

/////////////////////////////////////////////////////////////////////////////////

int \_tmain(int argc, \_TCHAR\* argv[])

{

// init glut stuff..

graphics.create(argc, argv);

// good place for one-off initialisations and objects creation..

// randomly generate fly positions

flies.setFlyVectors();

// randomly generate fly direction (velocity)

flies.moveDirection();

// enter game loop..

graphics.loop();

return 0;

}

/\*

interpolation - fill a triangle with color (rasterization)

\*/

## 3.5 – Matrix.cpp

#include "stdafx.h"

Matrix::Matrix()

{

// create matrix(size) for array (made within parent Matrix class)

this->createMatrix(16);

}

Matrix::~Matrix()

{

delete[] this->matrixArray;

}

void Matrix::identityMatrix()

{

// create matrix(size) for array (made within parent Matrix class)

this->createMatrix(16);

// Set values in matrix (within parent class Matrix.cpp)

this->matrixArray[0] = 1;

this->matrixArray[1] = 0;

this->matrixArray[2] = 0;

this->matrixArray[3] = 0;

this->matrixArray[4] = 0;

this->matrixArray[5] = 1;

this->matrixArray[6] = 0;

this->matrixArray[7] = 0;

this->matrixArray[8] = 0;

this->matrixArray[9] = 0;

this->matrixArray[10] = 1;

this->matrixArray[11] = 0;

this->matrixArray[12] = 0;

this->matrixArray[13] = 0;

this->matrixArray[14] = 0;

this->matrixArray[15] = 1;

}

// Create 3x1 Matrix

Matrix::Matrix(float r1, float r2, float r3)

{

// create matrix(size) for array (made within parent Matrix class)

this->createMatrix(16);

// Set values in matrix (within parent class Matrix.cpp)

this->matrixArray[0] = r1;

this->matrixArray[1] = 0;

this->matrixArray[2] = 0;

this->matrixArray[3] = 0;

this->matrixArray[4] = r2;

this->matrixArray[5] = 0;

this->matrixArray[6] = 0;

this->matrixArray[7] = 0;

this->matrixArray[8] = r3;

this->matrixArray[9] = 0;

this->matrixArray[10] = 0;

this->matrixArray[11] = 0;

this->matrixArray[12] = 0;

this->matrixArray[13] = 0;

this->matrixArray[14] = 0;

this->matrixArray[15] = 1;

}

// Create 2x2 Matrix

Matrix::Matrix(float r1, float r2, float r3, float r4)

{

// create matrix(size) for array (made within parent Matrix class)

this->createMatrix(16);

// Set values in matrix (within parent class Matrix.cpp)

this->matrixArray[0] = r1;

this->matrixArray[1] = r2;

this->matrixArray[2] = 0;

this->matrixArray[3] = 0;

this->matrixArray[4] = r3;

this->matrixArray[5] = r4;

this->matrixArray[6] = 0;

this->matrixArray[7] = 0;

this->matrixArray[8] = 0;

this->matrixArray[9] = 0;

this->matrixArray[10] = 0;

this->matrixArray[11] = 0;

this->matrixArray[12] = 0;

this->matrixArray[13] = 0;

this->matrixArray[14] = 0;

this->matrixArray[15] = 1;

}

// Create 3x3 Matrix

Matrix::Matrix(float r1, float r2, float r3, float r4, float r5, float r6, float r7, float r8, float r9)

{

// create matrix(size) for array (made within parent Matrix class)

this->createMatrix(16);

// Set values in matrix (within parent class Matrix.cpp)

this->matrixArray[0] = r1;

this->matrixArray[1] = r2;

this->matrixArray[2] = r3;

this->matrixArray[3] = 0;

this->matrixArray[4] = r4;

this->matrixArray[5] = r5;

this->matrixArray[6] = r6;

this->matrixArray[7] = 0;

this->matrixArray[8] = r7;

this->matrixArray[9] = r8;

this->matrixArray[10] = r9;

this->matrixArray[11] = 0;

this->matrixArray[12] = 0;

this->matrixArray[13] = 0;

this->matrixArray[14] = 0;

this->matrixArray[15] = 1;

}

// Create 4x4 Matrix

Matrix::Matrix(float r1, float r2, float r3, float r4, float r5, float r6, float r7, float r8, float r9, float r10, float r11, float r12, float r13, float r14, float r15, float r16)

{

// create matrix(size) for array (made within parent Matrix class)

this->createMatrix(16);

// Set values in matrix (within parent class Matrix.cpp)

this->matrixArray[0] = r1;

this->matrixArray[1] = r2;

this->matrixArray[2] = r3;

this->matrixArray[3] = r4;

this->matrixArray[4] = r5;

this->matrixArray[5] = r6;

this->matrixArray[6] = r7;

this->matrixArray[7] = r8;

this->matrixArray[8] = r9;

this->matrixArray[9] = r10;

this->matrixArray[10] = r11;

this->matrixArray[11] = r12;

this->matrixArray[12] = r13;

this->matrixArray[13] = r14;

this->matrixArray[14] = r15;

this->matrixArray[15] = r16;

}

// initial matrix creation

bool Matrix::createMatrix(int size)

{

bool result = false;

this->matrixArray = nullptr;

this->matrixArray = new float[size];

if (this->matrixArray != nullptr)

{

result = true;

}

else

{

// Result already set to false

}

return result;

}

void Matrix::print()

{

cout << this->matrixArray[0] << "\t";

cout << this->matrixArray[1] << "\t";

cout << this->matrixArray[2] << "\t";

cout << this->matrixArray[3] << endl;

cout << this->matrixArray[4] << "\t";

cout << this->matrixArray[5] << "\t";

cout << this->matrixArray[6] << "\t";

cout << this->matrixArray[7] << endl;

cout << this->matrixArray[8] << "\t";

cout << this->matrixArray[9] << "\t";

cout << this->matrixArray[10] << "\t";

cout << this->matrixArray[11] << endl;

cout << this->matrixArray[12] << "\t";

cout << this->matrixArray[13] << "\t";

cout << this->matrixArray[14] << "\t";

cout << this->matrixArray[15] << endl << endl;

}

## 3.6 – stdafx.cpp

// stdafx.cpp : source file that includes just the standard includes

// opengl\_window.pch will be the pre-compiled header

// stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"

// TODO: reference any additional headers you need in STDAFX.H

// and not in this file

## 3.7 – Vector.cpp

#include "stdafx.h"

Vector::Vector()

{

this->x = 0;

this->y = 0;

this->z = 0;

this->w = 0;

}

Vector::Vector(float x, float y, float z, float w)

{

this->x = x;

this->y = y;

this->z = z;

this->w = w;

}

Vector::~Vector()

{

}

void Vector::print()

{

cout << this->x << endl << this->y << endl << this->z << endl << this->w << endl << endl;

}

# Conclusion

This document outlines the code creating a custom maths library, Implementation of the library and functions being used by an array of flies(pixels) moving around the screen. Included in the custom Math library are several functions including, custom line drawing algorithms, circle drawing, slope calculations and line equation functions and a custom convex hull using Jarvis march.

# References

[1]"Convex Hull | Set 1 (Jarvis's Algorithm or Wrapping) - GeeksforGeeks", GeeksforGeeks, 2017. [Online]. Available: http://www.geeksforgeeks.org/convex-hull-set-1-jarviss-algorithm-or-wrapping/. [Accessed: 11- Apr- 2017].

[2]Sedgewick., Algorithms in C, 1st ed. Addison-Wesley Professional, 2002, pp. 353-367.

[3]J. Hughes, Computer graphics, 1st ed. NewJersey: Prentice Hall Inc, 2002, pp. 600-609.

[4]K. Loudon, Mastering Algorithms with C, 1st ed. Sebastopol: O'Reilly Media, Inc., 2009, pp. 507-512.