Занятие №7.

18.10.2018.

Лабораторная работа №1.

1. Построение графика функции, заданной, в зависимости от варианта, одним из следующих способов:
    - параметрически заданная функция в декартовых координатах;
    - явно заданная функция в полярных координатах;
    - параметрически заданная функция в полярных координатах;
    - явно заданная функция в эллиптических координатах;
    - параметрически заданная функция в эллиптических координатах;
    - параметрически заданная функция в бицентрических координатах.
2. В модуль Data.h добавьте функцию, график которой необходимо построить, а также числовые переменные, задающие границы интервала изменения параметра.
3. Добавьте в класс Scene2D метод, осуществляющий построение графика функции, заданной в модуле Data.h. При построении графика функции желательно выбирать такой шаг параметра, который обеспечивает одинаковое расстояние между соседними узлами при любых масштабах и при любых разрешениях рабочей области окна. Общий подход должен быть таким: чем больше пикселей приходится на единицу масштаба мировой системы координат, т.е. чем больше величина $p_x = W / (R-L) = H / (T-B)$, тем меньше шаг.
4. В обработчик сообщения WM_PAINT добавьте вызов реализованного метода.

Занятие №8.

25.10.2018.

Лабораторная работа №1.

Завершение выполнения лабораторной работы №1.

Занятие №9.

01.11.2018.

Лабораторная работа №2.

1. Откройте проект MatrixExample.
2. Изучите структуру проекта:
    1) Matrix.h – модуль, содержащий шаблон класса Matrix для работы с квадратными матрицами;

2) AffineTransform.h – модуль, в котором реализуются функции, возвращающие матрицы базовых аффинных преобразований;
3) MatrixExample.cpp – главный модуль, в котором тестируются реализованные классы и функции.

3. Ознакомьтесь с файлом Matrix.h. Обратите внимание, что в данном модуле содержится шаблон класса Matrix с параметром Cell – так называется тип данных для хранения коэффициентов матрицы.
4. Изучите поля и методы шаблонного класса Matrix:
   - size – скрытое поле, размер квадратной матрицы;
   - cells – скрытое поле, двумерный динамический массив коэффициентов матрицы;
   - AllocateCells, FreeCells – скрытые методы, осуществляющие выделение и освобождение памяти для динамического массива коэффициентов (обратите внимание, что работа с памятью, а также изменение значения поля size осуществляется в этих двух методах и только в них);
   - конструкторы и деструкторы (включая конструктор создания матрицы из линейного списка);
   - оператор круглые скобки для взятия индекса;
   - перегрузка оператора присваивания и арифметических операторов;
   - перегрузка операторов потокового ввода/вывода в качестве дружественных функций.
5. Реализуйте недостающие методы шаблонного класса Matrix:
   - конструктор создания матрицы из линейного списка;
   - арифметические операторы вычитания и умножения матриц.

   Запустите проект и протестируйте реализованные методы.
6. Модифицируйте шаблонный класс Matrix так, чтобы он позволял работать с прямоугольными матрицами. Вместо поля size используйте поля rows и cols. Протестируйте получившийся шаблонный класс на прямоугольных матрицах.
7. Реализуйте в модуле AffineTransform.h функции, возвращающие матрицы базовых аффинных преобразований:
   - Identity() – тождественное аффинное преобразование;
   - Translation(x, y) – параллельный перенос на вектор с координатами (x, y);
   - Rotation(t) – поворот вокруг начала координат на угол t против часовой стрелки;
   - Rotation(c, s) – поворот вокруг начала координат на угол, косинус и синус которого пропорциональны c и s соответственно;
   - Scaling(kx, ky) – масштабирование вдоль координатных осей;

- функции, возвращающие матрицы отражения относительно координатных осей и начала координат, можно не реализовывать, так как к ним может быть сведена функция Scaling.

Занятие №10.

08.11.2018.

Лабораторная работа №2.

1. Добавьте в проект MatrixExample модуль Model2D.h. Для этого в окне обозревателя решений (Solution Explorer) найдите проект MatrixExample, кликните правой кнопкой мыши по разделу «Header Files» и выберите «Add», «New Item...». В открывшемся окне выберите «Header File (.h)», введите имя файла «Model2D.h» и нажмите кнопку «Add». Наконец, включите в проект код созданного модуля при помощи директивы #include, добавив в главный модуль MatrixExample.cpp строку

```
#include "Model2D.h"
```

2. Опишите в модуле Model2D.h класс Model2D:

```
#ifndef MODEL_2D_H
#define MODEL_2D_H

#include <string>
#include "Matrix.h"

class Model2D
{
private:
	Matrix<> Vertices;
	Matrix<int> Edges;
public:
	Model2D() : Vertices(), Edges() {}
	Model2D(const Matrix<> Vertices, const Matrix<int> Edges) :
		Vertices(Vertices), Edges(Edges) {}
	Matrix<> GetVertices() { return Vertices; }
	Matrix<int> GetEdges() { return Edges; }
};

#endif MODEL_2D_H
```

3. Добавьте в класс Model2D следующие конструкторы и методы:
   - Model2D(string, string) – конструктор создания модели по именам файлов, в которых лежат карта вершин и карта рёбер;
   - double GetVertexX(int) и double GetVertexY(int) – получение координат вершины модели с заданным номером;
   - void Apply(Matrix<>) – применение к модели аффинного преобразования, заданного своей матрицей.
4. Протестируйте созданный класс, добавив в главный модуль код, аналогичный данному:

```
double v[6] = {2, 1, 3, 0, 1, 1};
Matrix<> V(3, 2, v);
int e[2] = {1, 2};
```

```
Matrix<int> E(1, 2, e);
Model2D model(V, E);
cout
     << endl << "Edges:" << endl << model.GetEdges()
     << endl << "Vertices before AT:" << endl << model.GetVertices();
model.Apply(Translation(1, 2));
cout
     << endl << "Vertices after AT:" << endl << model.GetVertices();
```

Примените к модели несколько аффинных преобразований, в том числе составных.

5. Модифицируйте метод Apply с использованием накопленного аффинного преобразования. Для этого добавьте в класс Model2D скрытые поля:
   - CumulativeAT – накопленное аффинное преобразование, которое в конструкторах инициализируется тождественным аффинным преобразованием;
   - InitialVertices – исходные вершины модели, которые в конструкторах инициализируются такой же матрицей, что и Vertices.

   В методе Apply сначала пересчитывается матрица накопленного аффинного преобразования, а затем – матрица вершин.

Занятие №11.

15.11.2018.

Лабораторная работа №2.

1. Добавьте в проект Plot2DViewer модули Matrix.h, AffineTransform.h и Model2D.h. Надёжнее это делать следующем образом: в окне обозревателя решений (Solution Explorer) найдите проект Plot2DViewer, кликните правой кнопкой мыши по разделу «Header Files» и выберите «Add», «New Item...». В открывшемся окне выберите «Header File (.h)», введите имя файла (например, «Matrix.h») и нажмите кнопку «Add». Во вновь созданный файл скопируйте написанный ранее код. Наконец, включите в проект код созданного модуля при помощи директивы #include, добавив в главный модуль Plot2DViewer.cpp строки

   ```
   #include "Matrix.h"
   #include "AffineTransform.h"
   #include "Model2D.h"
   ```

2. Добавьте в класс Scene2D объект model класса Model2D. Предусмотрите инициализацию этого объекта (либо в конструкторе класса Scene2D, либо из главного модуля).
3. Добавьте в класс Scene2D метод Render, осуществляющий отрисовку объекта model.
4. Добавьте в оконную процедуру главного модуля Plot2DViewer.cpp обработчики нажатий на различные клавиши, в которых осуществляется применение базовых аффинных преобразований к модели посредством вызова метода Apply класса Model2D. Так, например, в обработчик нажатия клавиши «стрелка вправо» может быть добавлен следующий код:

```
scene.model.Apply( Transfer(0.5, 0) );
```

Занятие №12.

22.11.2018.

Лабораторная работа №2.

Завершение выполнения лабораторной работы №2.