Занятие №1.

06.09.2018.

Вводное занятие.

1. Откройте проект Win32Example. О структуре проектов Win32 можно почитать, например, здесь:

   https://msdn.microsoft.com/ru-ru/library/bb384843.aspx

2. В коде файла Win32Example.cpp найдите следующие процедуры:
   - WinMain – главная процедура приложения, о ней будет рассказано на лекции 08.09.2018;
   - WndProc – оконная процедура, которая содержит обработчики системных сообщений;
   - TestDrawing и DrawDiagonal – процедуры, осуществляющие непосредственную отрисовку.

3. В оконной процедуре содержатся обработчики следующих системных сообщений:
   - WM_PAINT – сообщение, отправляемое приложению тогда, когда требуется перерисовка его окна (именно в обработчике этого сообщения и вызывается процедура TestDrawing);
   - WM_DESTROY – сообщение, отправляемое приложению при его закрытии.

   Полный список системных сообщений можно найти, например, здесь:

   https://autohotkey.com/docs/misc/SendMessageList.htm

4. Перейдите к процедуре TestDrawing. Протестируйте разные функции отрисовки линий и графических примитивов. Обратите внимание, что все графические процедуры вызываются для графического контекста dc.

   При желании попробуйте поменять цвет линий:

   ```
   COLORREF PenColor=RGB(255,0,0);
   HPEN pen = (HPEN) SelectObject( dc, CreatePen(PS_SOLID,1,PenColor) );
   ... // Графические процедуры
   DeleteObject( SelectObject(dc, pen) );
   ```

   Дополнительная информация по функциям отрисовки:
   https://msdn.microsoft.com/ru-ru/library/windows/desktop/dd145031.aspx

   Дополнительная информация по стилям отрисовки:
   https://msdn.microsoft.com/ru-ru/library/windows/desktop/dd183523.aspx

5. В обработчике сообщения WM_PAINT замените вызов процедуры TestDrawing на DrawDiagonal (не забыв объявить её прототип). Запустите приложение и определите, в каких случаях (перетаскивание окна, изменение его размеров, свёртывание/развёртывание окна и т.д.) диагональ перерисовывается, а в каких – нет. Обработчик какого сообщения требуется добавить, чтобы диагональ перерисовывалась всегда?

6. Добавьте в оконную процедуру обработчик нажатия левой кнопки мыши (сообщение WM_LBUTTONDOWN). При нажатии левой кнопки мыши выполняйте, например, следующее действие: отрисовку/удаление другой диагонали. Для этого, например, можно объявить глобальную переменную логического типа, отвечающую за отрисовку другой диагонали, а в процедуре DrawDiagonal производить эту отрисовку в зависимости от значения этой переменной.

Занятие №2.

13.09.2018.

Лабораторная работа №0.

1. Откройте проект FigureViewer.
2. Изучите структуру проекта:
   - Figure.h – модуль, содержащий абстрактный класс Figure;
   - Sight.h – модуль, содержащий класс Sight – неабстрактный потомок класса Figure;
   - FigureViewer.cpp – главный модуль.
3. Откройте главный модуль FigureViewer.cpp. Обратите внимание, что главном модуле объявляется только одна глобальная переменная – создаётся объект класса Sight, а все дальнейшие действия осуществляются посредством обращения к методам, реализованным в этом классе.

   Изучите обработчики системных сообщений, представленные в оконной процедуре:
   - WM_PAINT – очистка рабочей области окна и отрисовка фигуры;
   - WM_KEYDOWN – управление фигурой с клавиатуры.

     Справка по системному сообщению WM_KEYDOWN здесь: https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms646280.aspx

     Коды различных клавиш: https://msdn.microsoft.com/ru-ru/library/windows/desktop/dd375731.aspx
   - WM_RBUTTONDOWN, WM_LBUTTONDOWN, WM_MOUSEMOVE, WM_LBUTTONUP – управление фигурой при помощи мыши.

     Предполагается, что при нажатии правой клавиши мыши фигура перемещается в ту точку рабочей области, где расположен курсор. А при помощи левой клавиши мыши осуществляется перетаскивание фигуры.
4. Откройте модуль Figure.h. Изучите поля класса Figure:
   - size – размер фигуры;
   - offsetX, offsetY – смещение фигуры относительно левого верхнего угла рабочей области окна;

- isDragging – логическая переменная, принимающая значение «истина» во время перетаскивания фигуры при помощи левой клавиши мыши;
- previousX, previousY – координаты предыдущего положения фигуры, используемые во время её перетаскивания.

5. Найдите в модуле Figure.h абстрактные методы Draw и InnerPoint. Откройте модуль Sight.h и разберитесь с деталями реализации этих методов в классе Sight.
6. Реализуйте все необходимые методы класса Figure:
   - Clear – очистка рабочей области экрана;
   - MoveTo – перемещение фигуры в точку с координатами (X, Y); проверьте, правильно ли реагирует фигура на нажатие правой кнопки мыши;
   - Move – перемещение фигуры на X пикселей вправо и Y пикселей вниз;
   - StartDragging, Drag, StopDragging – управление перетаскиванием фигуры.
7. Реализуйте возможность управления размером фигуры, например, при помощи колёсика мыши (WM_MOUSEWHEEL).

   Учтите, что для некоторых системных сообщений (в том числе и для сообщения WM_MOUSEWHEEL) функции GET_X_LPARAM(lParam), GET_Y_LPARAM(lParam) возвращают координаты курсора не в рабочей области окна, а на всём экране. Для перехода от глобальных экранных координат к локальным координатам в рабочей области окна используется функция ScreenToClient:

```
POINT P;
P.x = GET_X_LPARAM(lParam);
P.y = GET_Y_LPARAM(lParam);
ScreenToClient(hWnd, &P);
sight.StartDragging(P.x, P.y);
```

8. Создайте по аналогии с Sight ещё одного неабстрактного наследника для класса Figure.

Занятие №3.

20.09.2018.

Лабораторная работа №0.

Варианты дополнительных заданий:

1. Реализуйте возможность управления цветом объекта. Для этого, например, добавьте в класс Figure поле Color и метод SetColor.
2. Производите масштабирование фигуры не при простом вращении колёсика мыши, а если при этом зажата клавиша <CTRL>.
3. При масштабировании объекта проверяйте, наведён ли на него курсор мыши.

Учтите, что для некоторых системных сообщений (в том числе и для сообщения WM_MOUSEWHEEL) функции GET_X_LPARAM(lParam), GET_Y_LPARAM(lParam) возвращают координаты курсора не в рабочей области окна, а на всём экране. Для перехода от глобальных экранных координат к локальным координатам в рабочей области окна используется функция ScreenToClient:

```
POINT P;
P.x = GET_X_LPARAM(lParam);
P.y = GET_Y_LPARAM(lParam);
ScreenToClient(hWnd, &P);
if ( sight.InnerPoint(P.x, P.y) ) ...
```

4. Добейтесь того, чтобы при манипуляциях (перетаскивании, масштабировании и т.д.) объект не выходил бы за пределы рабочей области окна.
5. Реализуйте проверку на внутреннюю точку для невыпуклой фигуры.
6. Реализуйте возможность взаимодействия двух (или более) объектов:
   - проверка пересечения двух объектов;
   - проверка нахождения центральной точки одного объекта внутри другого (проверка, «наведён ли прицел»);
   - отталкивание одного объекта другим и т.д.
7. Создайте массив объектов и переключайте управление с одного объекта на другой при помощи клавиши <TAB>. Объект, находящийся в фокусе, выделяйте другим цветом.

Занятие №3.

27.09.2018.

Лабораторная работа №0.

Завершение выполнения лабораторной работы №0.

Занятие №5.

04.10.2018.

Лабораторная работа №1.

1. Откройте проект Plot2DViewer. Для нечётных вариантов – первый проект, для чётных – второй.
2. Ознакомьтесь с файлом Data.h, в котором заданы исходные значения параметров, определяющих связь между мировыми и экранными координатами, а также примеры функций, графики которых будут построены.
3. Ознакомьтесь с главным модулем Plot2DViewer.cpp. Найдите в оконной процедуре объявление объекта класса Scene2D, а также обработчики системных сообщений WM_PAINT и WM_SIZE.
4. Откройте модуль Camera2D.h. Изучите защищённые (protected) поля класса Camera2D. Учтите, что горизонтальное и вертикальное разрешения W и H не в

точности равны координатам правой нижней точки окна, а отличаются от них на единицу. Реализуйте защищённые методы класса Camera2D, отвечающие за работу с мировыми и экранными координатами.

5. Реализуйте публичные методы SetResolution и Clear класса Camera2D. Какие действия необходимо выполнять при изменении размеров окна? Какие поля нуждаются в обновлении? Используйте эти поля и при реализации метода Clear, отвечающего за очистку рабочей области экрана.
6. Найдите в классе Camera2D скрытые поля posX и posY, отвечающие за расположение графического курсора в мировых координатах. Реализуйте методы MoveTo и LineTo, отвечающие за управление этим графическим курсором. Используя эти методы, реализуйте ещё один – Axes, отвечающий за построение координатных осей. При этом можно ограничиться простой отрисовкой двух прямых линий, а можно и добавить (по желанию) засечки, сетку, метки и т.д. Запустите проект и убедитесь, что координатные оси отрисовываются верно.
7. Откройте файл Scene2D.h. В данной заготовке класс Scene2D наследуется от класса Camera2D. Вообще говоря, использовать наследование не обязательно: можно было бы, например, просто добавлять объект класса Camera2D в класс Scene2D. Но тогда потребуются методы-геттеры для защищённых полей класса Camera2D. Если же использовать наследование, то защищённые поля класса Camera2D будут доступны в классе-потомке Scene2D и могут быть использованы при построении графика функции.
8. Реализуйте метод Plot класса Scene2D, в который передаётся функция f, график которой необходимо построить. Для построения используйте количество точек, равное горизонтальному разрешению рабочей области окна. При этом вам пригодится величина, выражающая ширину пикселя в единицах масштаба мировой системы координат. Эту величину можно вычислять непосредственно в методе Plot, а можно и добавить отдельный метод в класс Camera2D.

Занятие №6.

11.10.2018.

Лабораторная работа №1.

1. Реализуйте возможность перетаскивания при помощи мыши графика функции (вместе с координатной системой). Для этого может понадобиться добавить в класс Camera2D скрытое поле логического типа, принимающее значение «истина» во время перетаскивания графика при помощи левой клавиши мыши, а также методы, вызываемые при срабатывании сообщений WM_LBUTTONDOWN, WM_MOUSEMOVE, WM_LBUTTONUP.
2. Реализуйте возможность масштабирования при помощи колёсика мыши графика функции (вместе с координатной системой) с сохранением позиции той точки изображения, на которую наведён курсор мыши.
3. Добавьте в метод, вызываемый при изменении размеров окна, код, обеспечивающий согласование масштабов по координатным осям.

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.

Занятие №13.

29.11.2018.

Лабораторная работа №3.

1. Создайте модуль Model3D.h, в котором реализовывается класс Model3D. Для отладки сначала работайте в проекте MatrixExample, а окончательную версию перенесите в основной проект.
2. Добавьте в класс Model3D следующие скрытые поля:
   - Matrix<> Vertices;
   - Matrix<> InitialVertices;
   - Matrix<> ProjectedVertices;
   - Matrix<int> Faces;
   - Matrix<int> Edges.
3. Добавьте в класс Model3D следующие конструкторы и методы:
   - Model3D() – конструктор по умолчанию;
   - Model3D(const Matrix<>, const Matrix<int>) – конструктор создания модели по заданным карте вершин и карте граней;
   - Model3D(string, string) – конструктор создания модели по именам файлов, в которых лежат карта вершин и карта граней;
   - void SetEdges() – создание карты рёбер по заданной карте граней;
   - Matrix<> GetVertices(), Matrix<int> GetFaces(), Matrix<int> GetEdges() – получение карты вершин, карты граней и карты рёбер;
   - double GetVertexX(int), double GetVertexY(int) и double GetVertexZ(int) – получение координат вершины модели с заданным номером;
   - void Apply(Matrix<>) – применение к модели аффинного преобразования, заданного своей матрицей;
   - void Project(Matrix<>) – проецирование модели.
4. Протестируйте созданный класс. Перенесите реализованный модуль в основной проект.

Занятие №14.

06.12.2018.

Лабораторная работа №3.

1. Добавьте в проект модуль модуль Vector.h (либо Vector3D.h), в котором реализовывается класс Vector (Vector3D). Для этого класса, помимо основных арифметических операций, реализуйте методы, вычисляющие скалярное и векторное произведения.
2. Добавьте в проект модуль Camera3D.h, в котором реализовывается класс Camera3D, который является наследником класса Camera2D.
3. Добавьте в класс Camera3D следующие скрытые поля:
    - векторы Ov, T, N;
    - число D;
    - матрицы WorldToView, ViewToProject, WorldToProject.
4. Добавьте в класс Camera3D следующие методы:
    - Camera3D() – конструктор по умолчанию, в котором вызываются методы-сеттеры для полей Ov, T, N, D. В качестве значений по умолчанию можно взять, например, Ov(0; 0; 0), T(0; 1; 0), N(0; 0; 1), D=16.
    - методы-сеттеры для полей Ov, T, N, D;
    - UpdateCamera() – обновление матриц перехода.
5. Добавьте в проект модуль Scene3D.h, в котором реализовывается класс Scene3D, который является наследником класса Camera3D. Добавьте в главный модуль объект класса Scene3D (вместо объекта класса Scene2D). Добавьте в класс Scene3D объект model класса Model3D. Предусмотрите инициализацию этого объекта, включающую в том числе и проецирование модели. Добавьте в класс Scene3D метод Render, осуществляющий отрисовку объекта model.
6. Дополнительное задание: добавьте в основной проект возможность контроля камеры либо переключения камер. Например:
    - при горизонтальном движении мыши можно осуществлять поворот вектора N вокруг вектора T, а при вертикальном движении мыши – поворот вектора N в плоскости векторов N и T;
    - при вращении колёсика мыши можно изменять D;
    - при нажатии некоторых клавиш (например, клавиш нампада) можно переключать камеры.

    Во всех случаях в обработчиках соответствующих событий сначала вызываются методы-сеттеры для полей Ov, T, N, D, после чего вызывается метод UpdateCamera.

Занятие №15.

13.12.2018.

Лабораторная работа №3.

1. Реализуйте в модуле AffineTransform.h функции, возвращающие матрицы базовых аффинных преобразований в трёхмерном пространстве:
    - Identity3D(), либо Identity(int n), либо Identity(bool 3D) – тождественное аффинное преобразование;
    - Translation(x, y, z) – параллельный перенос на вектор с координатами (x, y, z);
    - RotationX(t), RotationY(t), RotationZ(t) – повороты вокруг координатных осей на угол t против часовой стрелки;
    - RotationX(c, s), RotationY(c, s), RotationZ(c, s) – повороты вокруг координатных осей на угол, косинус и синус которого пропорциональны c и s соответственно;
    - Scaling(kx, ky, kz) – 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.
2. Добавьте в оконную процедуру главного модуля обработчики нажатий на различные клавиши, в которых осуществляется применение базовых аффинных преобразований к модели посредством вызова методов Apply и Project класса Model3D. Как вариант, можно реализовать ещё один метод Apply в классе Scene3D, в котором осуществлялись бы вызовы методов Apply и Project класса Model3D.

Занятие №16.

20.12.2018.

Лабораторная работа №3.

Завершение выполнения лабораторной работы №3.