РЕКОМЕНДАЦИИ ДЛЯ ВЫПОЛНЕНИЯ ЗАДАНИЯ 7

Функциональность, заложенная при выполнении задания 4 должна сохраниться в настоящем проекте. Поэтому, для начала создадим копию проекта созданного в четвертом задании и будем поэтапно вносить изменения в этот проект.

Будем использовать структуры данных, реализованные при выполнении задания 6, поэтому следует заменить содержимое файлов `Transform.cpp` и `Transform.h` на содержимое соответствующих файлов из проекта задания 6.

1 Построение графического изображения, заданного набором многоугольников

Рассмотрим вариант построения изображения, состоящего из набора многоугольников.

1.1 Формат входного файла

Предполагаем, что во входном файле могут встречаться (кроме пустых строк и строк с комментариями) строки команд следующего вида:

`frame Vcx Vcy Vx Vy` — команда выделения кадра. Если команда встречается в файле более одного раза, то каждый последующий экземпляр команды отменяет действие предыдущей;

`polygon n x1 y1 ... xn yn` — задается многоугольник с `n` вершинами, в котором числа `xi` или `yi` — координаты `i`-ой вершины многоугольника в порядке обхода вершин по часовой (или против часовой) стрелки.

Например файл

```
frame 0 0 250 250

# 1-й многоугольник с 5-ю вершинами
polygon 5 10 20 20 60 100 50 30 10 15 15
```

```
# 2-й многоугольник с 3-мя вершинами
polygon 3 100 100 200 200 90 150
```

содержит данные о двух многоугольниках, где первый состоит из пяти вершин, а второй — из трех.

1.2 Структуры данных

Для каждого многоугольника нам необходимо хранить список его вершин (в порядке их обхода). В файле `Transform.h` опишем тип данных `polygon` для многоугольника как экземпляр вектора точек:

```
typedef System::Collections::Generic::List<point> polygon;
```

В файле `Form1.h` опишем список загруженных в приложение многоугольников как параметр класса.

```
private: System::Collections::Generic::List<polygon^> polygons;
```

Параметр `polygons` — список указателей на многоугольники.

В процедуре, выполняющейся при загрузке формы (`Form1_Load`), добавим первоначальную очистку списка `polygons`.

```
polygons.Clear();
```

1.3 Процедура отрисовки изображения

Внесем изменения в обработчик события *Paint*.

В этой процедуре вместо цикла по элементам списка `lines` организуем цикл по элементам списка `polygons`:

grdPolygons

```
for (int i = 0; i < polygons.Count; i++) {
    ...
}
```

В этом цикле для многоугольника с номером `i` (его обозначаем через `p`) необходимо во вложенном цикле изобразить каждое из его ребер. Будем считать, что точки многоугольника пронумерованы от `0` до `p->Count - 1`. Тогда пусть первое ребро начинается в точке с номером `p->Count - 1` и заканчивается в точке с номером `0`, 2-е ребро начинается в точке с номером `0` и заканчивается в точке с номером `1`, третье ребро заканчивается в точке с номером `2` и т. д.

Сначала извлечем из списка начальную точку первого ребра.

```
polygon^ p = polygons[i];
point a, b, c;
vec A, B, A1, B1;
point2vec(p[p->Count - 1], A);
timesMatVec(T, A, A1);
vec2point(A1, a);
```

Теперь пройдемся в цикле по конечным точкам каждого из ребер и изобразим эти ребра.

```
for (int j = 0; j < p->Count; j++){
     point2vec(p[j], B);
     timesMatVec(T, B, B1);
     vec2point(B1, b);
     g ->DrawLine(blackPen, a.x, a.y, b.x, b.y);
     a = b;
}
```

1.4 Чтение файла

Чтение данных из входного файла организуем тем же образом, как делали это при выполнении задания 6.

В отличие от задания 6, во входном файле могут встречаться команды только 2-х типов. Обработка команды `frame` остается неизменной.

Выполнение команды `polygon` заключается в следующем. Сначала считываем из потока число вершин многоугольника:

```
int numpoint;
s >> numpoint;
```

После этого опишем новый многоугольник и в цикле прочитаем точки его вершин:

```
polygon^ P = gcnew polygon(0);
for (int i = 0; i<numpoint; i++) {
    point p;
    s >> p.x >>  p.y;
    P->Add(p);
}
```

Сформированный многоугольник добавим в список `polygons`:

```
polygons.Add(P);
```

На данном этапе проект готов. При запуске проекта могут считываться файлы с описанием многоугольников и изображение многоугольников выводится в окне. Можно проект запустить и проверить его работу.

2 Реализация алгоритма отсечения отрезков

Добавьте в проект файлы `PClip.cpp` и `PClip.h` с описанием процедур, реализующих алгоритм отсечения отрезков, соответствующий вашему варианту.

Процедурой в `PClip.cpp`, отсекающей многоугольник относительно окна видимости, должна быть процедура:

```
polygon^ Pclip (polygon^ P, point Pmin, point Pmax)
```

где `P` — отсекаемый многоугольник, `Pmin` и `Pmax` — точки соответственно с минимальными и максимальными значениями координат окна отсечения.

Подключите описанный код к проекту (подключите использование библиотеки `Pclip.h` с помощью `#include` в основном файле проекта).

В файле `Form1.h` внесите изменения в процедуру отрисовки формы (`Form1_Paint`): перед тем как нарисовать многоугольник, произведите для него запуск процедуры отсечения и выведите на экран только лишь результат отсечения.

Чтобы продемонстрировать побочные эффекты алгоритма отсечения, атрибуты пера для отрисовки границ окна видимости и пера ребер многоугольников должны отличаться. Кроме того, границы окна видимости следует выводить перед вычерчиванием многоугольников.