Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа №8
по дисциплине «Компьютерная графика»

Тема: Реализация алгоритма отсечения отрезка произвольным выпуклым отсекателем (Алгоритм Кируса-Бека).

Студент: Блохин Дмитрий

Группа: ИУ7-42Б

Москва.
2020 г.

Цель работы:

Изучение и программная реализация алгоритма отсечения отрезка произвольным выпуклым отсекателем (Алгоритм Кируса-Бека).

Задание:

- Необходимо обеспечить ввод отсекателя – произвольного многоугольника.Высветить его первым цветом. Также необходимо обеспечить ввод нескольких (до десяти) различных отрезков (высветить их вторым цветом). Отрезки могут иметь произвольное расположение: горизонтальные, вертикальные, имеющие произвольный наклон.
- Предусмотреть ввод отрезков, параллельных границе отсекателя.
- Ввод осуществлять с помощью мыши и нажатия других клавиш.
- Выполнить отсечение отрезков, показав результат третьим цветом. Исходные отрезки не удалять.

Ход работы:

Параметрическая форма задания отрезка: P(t)=P1+(P2-P1)t, где 0<=t<=1 (накладываем ограничения, чтобы показать, что это уравнение отрезка, а не бесконечной прямой)

t <0 - для нижней и левой границ

t > 1 - для верхней и правой границ

Для распознавания видимости отдельной точки находим скалярное произведение двух векторов Ni(A-fi): один вектор - вектор внутренней нормали к очередной стороне отсекателя, второй - вектор, соединяющий произвольную точку на границе отсекателя с рассматриваемой точкой. Если оно:

1. Положительно, то точка лежит по видимую сторону отсекателя

2. Равно 0, то точка лежит на границе отсекателя

3. Отрицательно, то точка расположена по невидимую сторону отсекателя

Для нахождения точек пересечения произвольного отрезка с границами отсекателя находим скалярное произведение Nвні(P1+(P2-P1)t-fi). Если оно:

1. Положительно, то точка видима относительно текущей границы отсекателя

2. Равно 0, то лежит на текущей границе отсекателя

3. Отрицательно, то точка невидима относительно текущей границы отсекателя

*t=(-Wi*Nвні)/(Nвні * D) - находим значение параметра t, определяющее координаты точки пересечения отрезка с границами отсекателя.*

Алгоритм Кируса - Бека(алгоритма отсечения отрезка произвольным выпуклым отсекателем):

1. Ввод P1, P2, K, C(K) (массив координат вершин)

2. Проверка отсекателя на выпуклость

3. tн = 0, tв = 1 (изначально считаем, что отрезок целиком видим)

4. D=P2-P1 (директриса - вектор, определяющий направление отрезка)

5. Цикл отсечения (по i от 1 до K)

    5.1. Wi = P1 – fi

    5.2. Нахождение Nвні

    5.3. Вычисление Dскі, Wcki

    5.4. Если Dскі = 0, то если Wскі < 0, то отрезок невидим

    5.5. Вычисление t=-Wск/Dск

    5.6. Если Dскі>0, то если t>1 то отрезок невидимый, иначе tн=max(t,tн)

    5.7. Если Dск<=0, то если t<0 то отрезок невидимый, иначе tв=min(t,tв)

    5.8. Конец цикла

6. Если tн <= tв, тогда изобразить отрезок (tн, tв)

7. Конец

Реализация:

1.Код проверки на выпуклость многоугольника.

Используемые функции для реализации проверки на выпуклость многоугольника:

```
int sign(int x)
{
    if (x > 0)
        return 1;
    if (x < 0)
        return -1;
    return 0;
}

int skewProduct(const QPoint &a, const QPoint &b)
{
    return a.x() * b.y() - a.y() * b.x();
}

int direction(const QPoint &prev, const QPoint &curr, const QPoint &next)
{
    return sign(skewProduct(curr - prev, next - curr));
}
```

Реализация проверки на выпуклость многоугольника

```
int MainWindow::checkConvex(QVector<QPoint> &clipper_vertices, int k)
{
    int f = 1;
    QPoint prev_vertex = clipper_vertices.back();
    QPoint curr_vertex = clipper_vertices[0];
    QPoint next_vertex = clipper_vertices[1];

    int prev_direction = direction(prev_vertex, curr_vertex, next_vertex);
    int curr_direction = 0;
    for (int i = 1; i < k && f; ++i)
    {
        prev_vertex = curr_vertex;
        curr_vertex = next_vertex;
        next_vertex = clipper_vertices[(i + 1) % clipper_vertices.size()];
        curr_direction = direction(prev_vertex, curr_vertex, next_vertex);
        if (curr_direction != prev_direction)
            f = 0;
       prev_direction = curr_direction;
    }
    return  f * curr_direction;
}
```

2. Реализация алгоритма Кируса-Бека.

Используемые функции для реализации алгоритма Кируса - Бека:

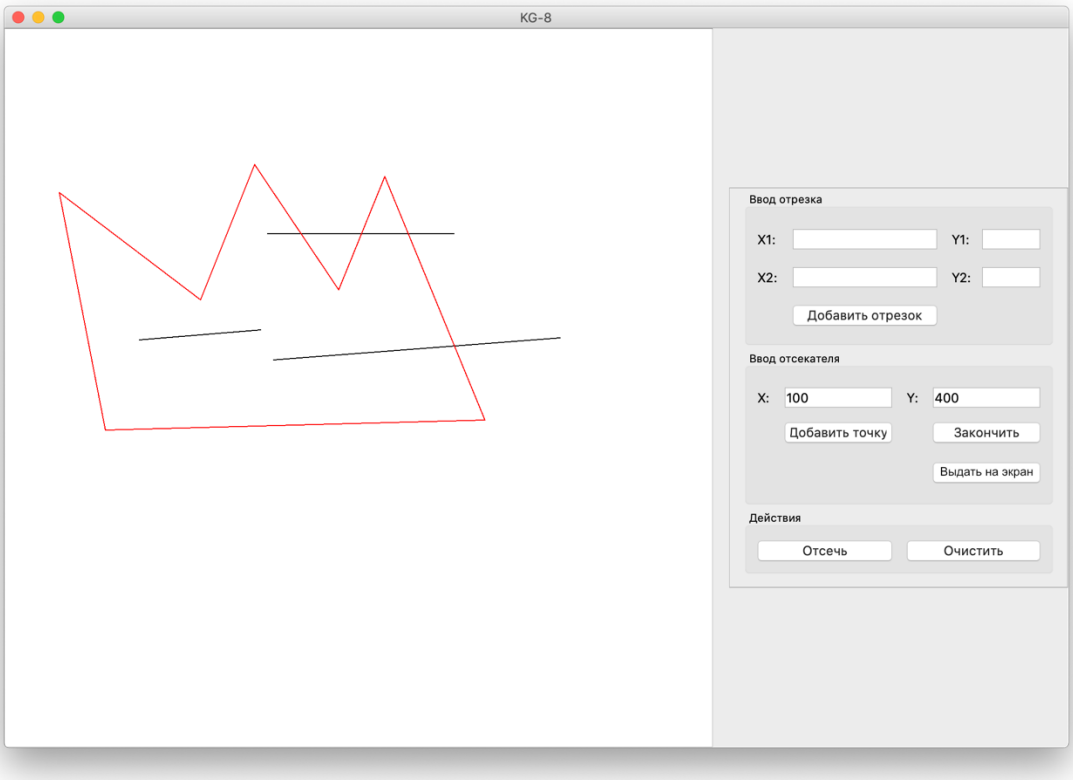
```
double scalar_product(const QPointF &a, const QPointF &b)
{
    return a.x() * b.x() + a.y() * b.y();
}

QPointF normal(const QPointF &point)
{
    return {-point.y(), point.x()};
}
```

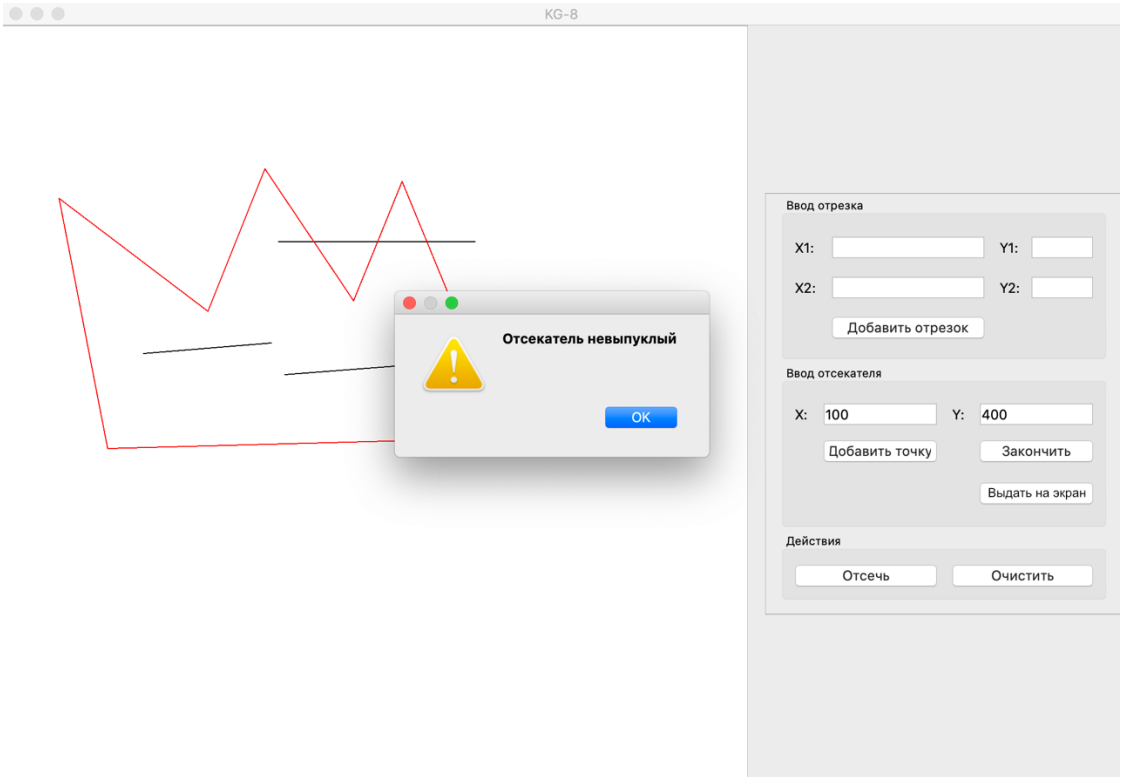
Реализация алгоритма Кируса-Бека:

```
void MainWindow::KirusBek(const QLine &line, int direction, const QVector<QLine> &edges, int k, QPainter &painter)
{
  double tb = 0;
  double te = 1;
  QPointF D = line.p2() - line.p1();

  for (int i = 0; i < k; i++)
  {
      QPointF W = line.p1() - edges[i].p1();

      //Вычисление вектора внутренней нормали к очередной i-ой стороне окна отсечения
      QPointF n = normal(direction * (edges[i].p2() - edges[i].p1()));

      double Dsc = scalar_product(D, n);
      double Wsc = scalar_product(W, n);

      if (fabs(Dsc) <= EPS) // Dsc == 0
      {
          if (Wsc < 0)
              return; // отрезок невидим
      }
      else
      {
          double t = -Wsc / Dsc;
          // поиск верхнего и нижнего предела t
          if (Dsc > 0)
          {
              if (t > 1)
                  return;
              else
                  tb = max(t, tb);
          }
          else
          {
              if (t < 0)
                  return;
              else
                  te = min(t, te);
          }
      }
  }
  // проверка фактической видимости отсеченного отрезка
  if (tb <= te)
    painter.drawLine(line.p1() + D * te, line.p1() + D * tb);
}
```

Демонстрация работы программы:

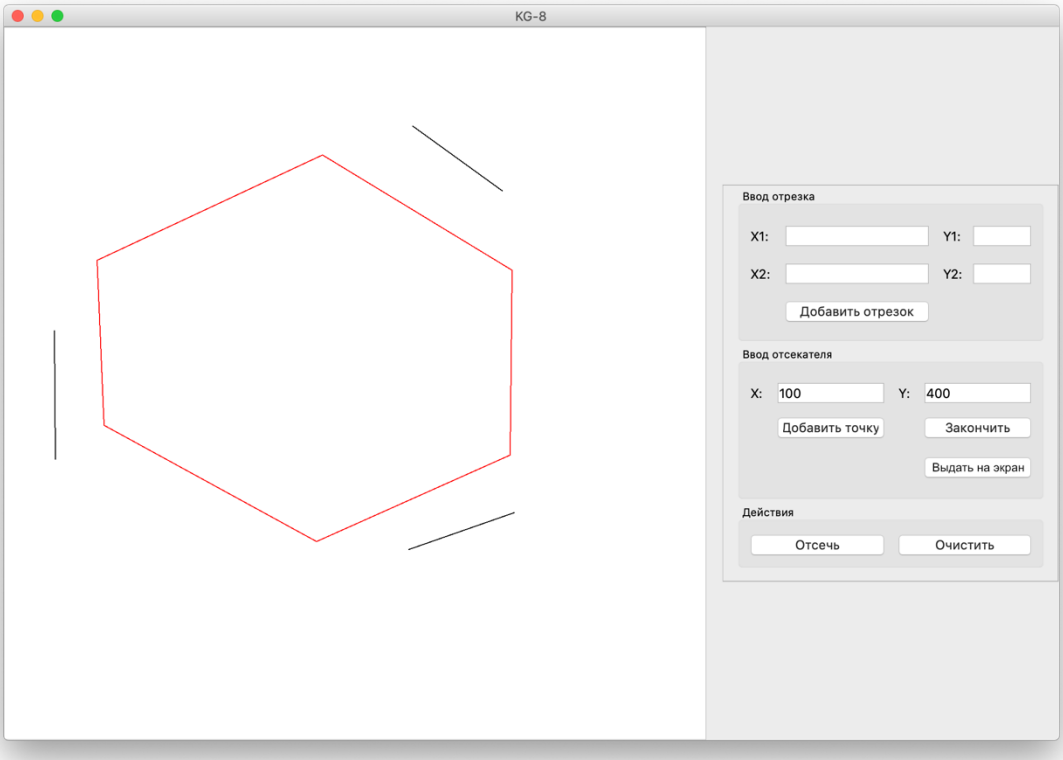
Проверка выпуклости многоугольника:



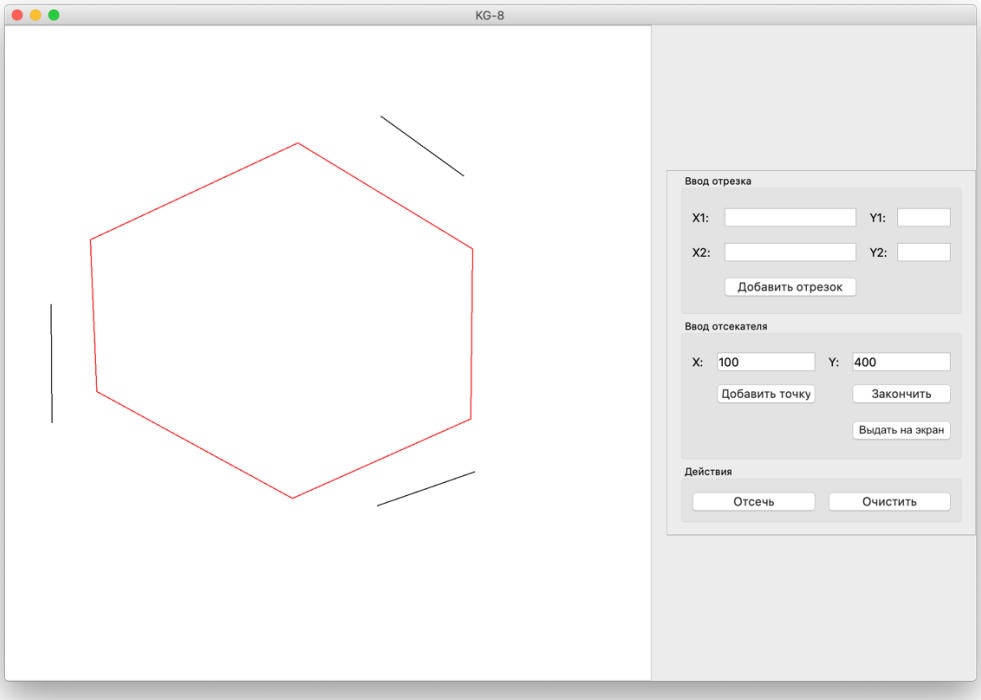


Пример 1

До

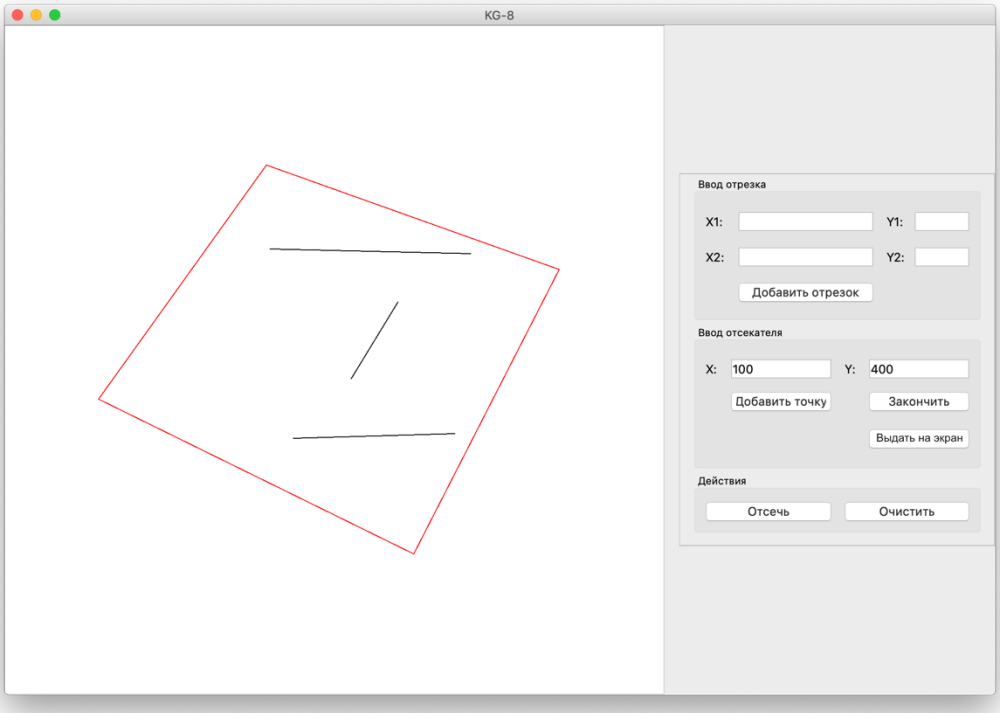


После

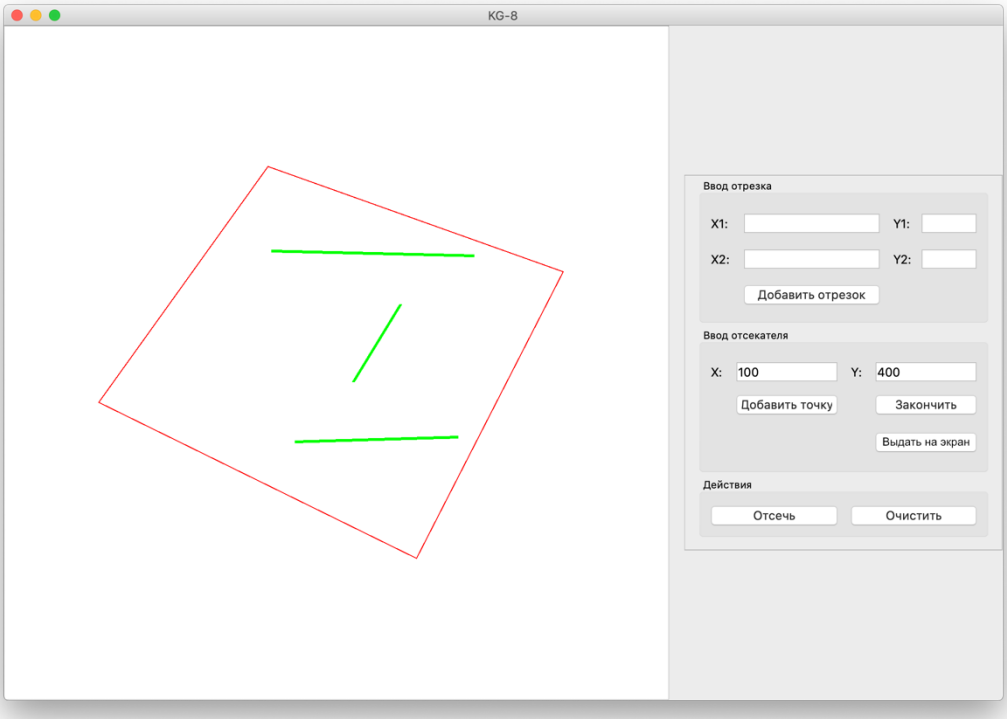


Пример 2

До

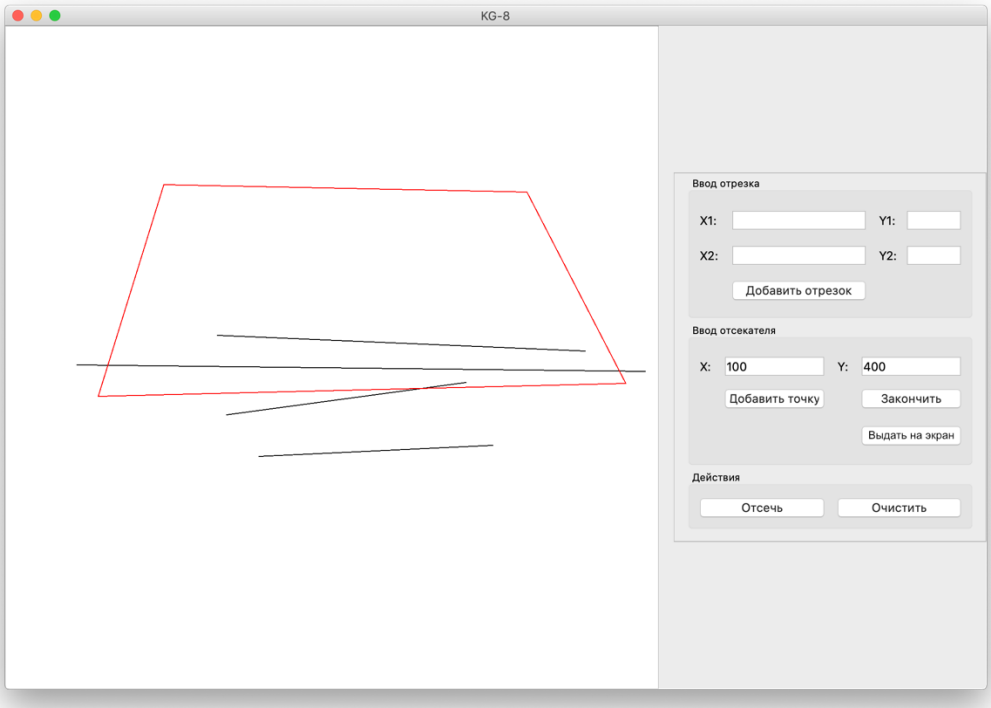


После

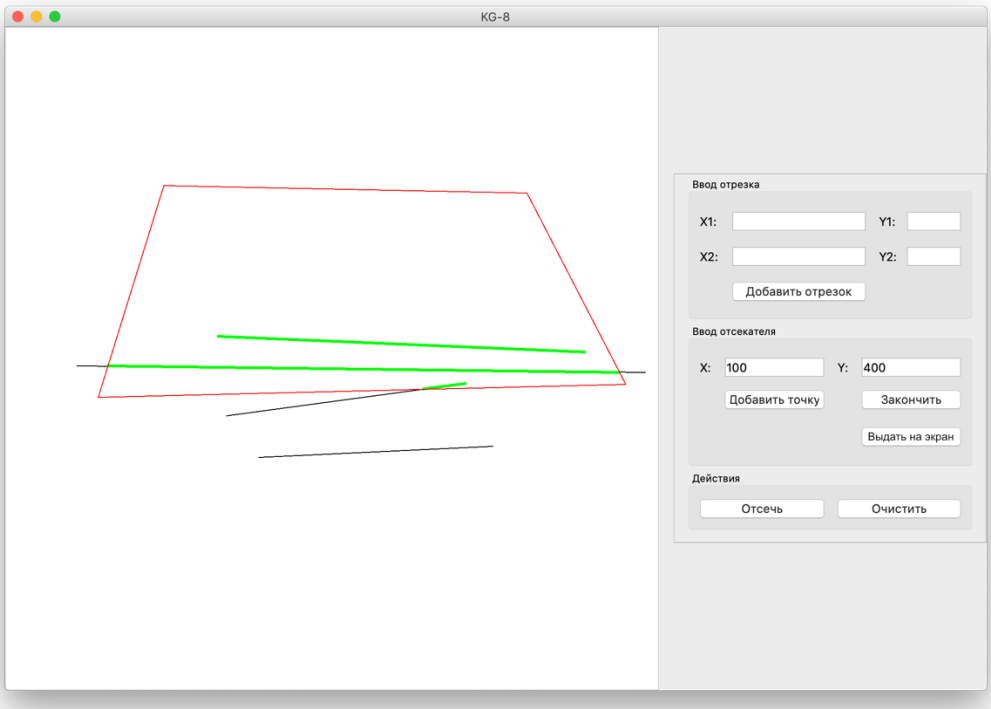


Пример 3

До

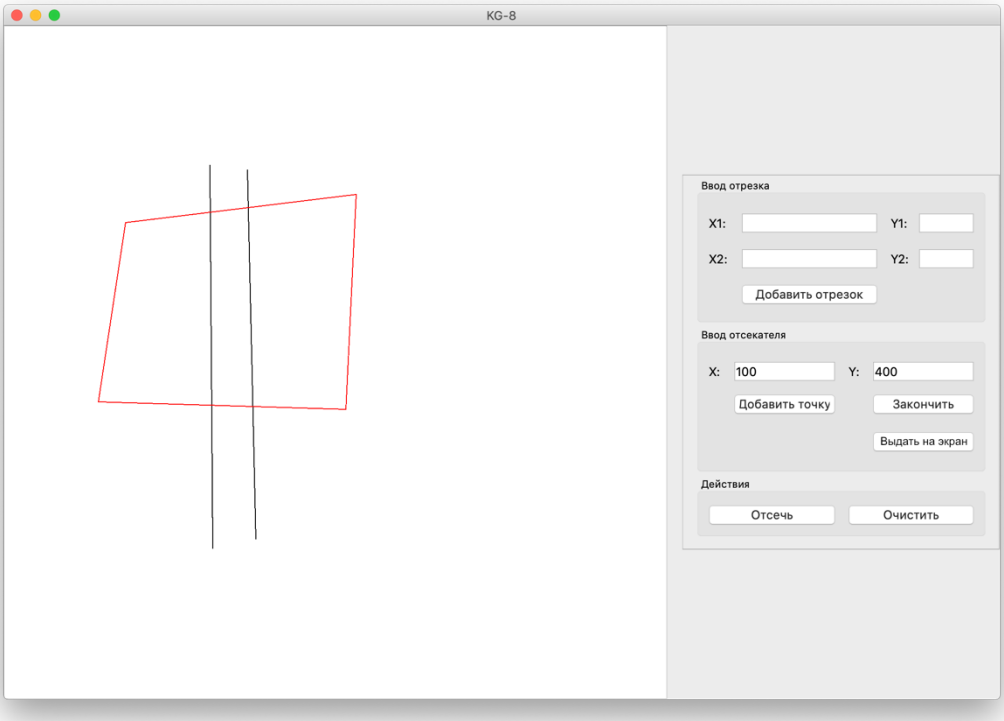


После



Пример 4

До



После