|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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 (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 Программная инженерия**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 9 |

**Название:**

Отсечение произвольного многоугольника выпуклым отсекателем (алгоритм Сазерленда-Ходжмена)

**Дисциплина:** Компьютерная графика

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ7-43Б |  | 03.06.2020 | С.С. Кононенко |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | А.В. Куров |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2020

**1. Цель работы**

Изучение и программная реализация алгоритма Сазерленда-Ходжмена отсечения многоугольников.

**2. Техническое задание**

Необходимо обеспечить ввод отсекателя – произвольного многоугольника. Высветить его первым цветом. Также необходимо обеспечить ввод отсекаемого многоугольника (высветить вторым цветом). Должна присутствовать проверка отсекателя на выпуклость. Должен быть предусмотрен ввод вершин многоугольника в произвольных точках ребер отсекателя (включая его вершины) Ввод осуществлять с помощью мыши и нажатия других клавиш. Выполнить отсечение многоугольника, показав результат третьим цветом. Исходный многоугольник не удалять.

**3. Теоретический материал**

Основная идея алгоритма **Сазерленда – Ходжмена** состоит в том, что отсечь многоугольник относительно одной прямой или плоскости очень легко. В этом алгоритме исходный и каждый из промежуточных многоугольников отсекается последовательно относительно одной прямой.

Идея алгоритма — решение задачи последовательно: многоугольник отсекается относительно каждой грани отсекателя по отдельности и при этом результат, полученный на i-ом шаге, служит исходным для (i+1)-ого отсечения.

В процессе рассмотрения ребер отсекаемого многоугольника возможны следующие расположения (будем считать, что ребро идет от P к S):

1. Ребро полностью вне рассматриваемой границы отсекателя. Не добавляем никакую вершину.
2. Ребро пересекает границу отсекателя, S — внутри, а Р — нет. В таком случае нужно добавить в список вершин искомого отсеченного многоугольника вершину S, а также точку пересечения текущего ребра с рассматриваемой гранью.
3. Ребро полностью находится внутри отсекателя. Добавляем обе вершины.
4. Ребро пересекает границу отсекателя, S — вне отсекателя, а Р — внутри. В таком случае нужно добавить в список вершин искомого отсеченного многоугольника вершину Р, а также точку пересечения текущего ребра с рассматриваемой гранью.

Недостаток алгоритма — построение ложных ребер (ребро будет обходиться два раза).

**4. Реализация алгоритма**

**def** **get\_cut**(figure, edge, normal):

res\_figure = []

figlen = len(figure)

**if** figlen < **3**:

**return** []

pcheck = check\_vis(figure[**0**], \*edge)

**for** i **in** range(**1**, figlen + **1**):

ccheck = check\_vis(figure[i % figlen], \*edge)

**if** pcheck:

**if** ccheck:

res\_figure.append(figure[i % figlen])

**else**:

res\_figure.append(get\_intersection(

[figure[i - **1**], figure[i % figlen]], edge, normal))

**else**:

**if** ccheck:

res\_figure.append(get\_intersection(

[figure[i - **1**], figure[i % figlen]], edge, normal))

res\_figure.append(figure[i % figlen])

pcheck = ccheck

**return** res\_figure

**def** **sutherlandhodgman**(figure, cut, normals):

res\_figure = figure

**for** i, \_ **in** enumerate(cut):

edge = [cut[i], cut[(i + **1**) % len(cut)]]

res\_figure = get\_cut(res\_figure, edge, normals[i])

**if** len(res\_figure) < **3**:

**return** []

**return** res\_figure

Реализация вспомогательных функций (операции над векторами, отсекателем, нормалями):

**def** **get\_vect**(dot\_start, dot\_end):

**return** [dot\_end[**0**] - dot\_start[**0**], dot\_end[**1**] - dot\_start[**1**]]

**def** **get\_vect\_mul**(fvector, svector):

**return** fvector[**0**] \* svector[**1**] - fvector[**1**] \* svector[**0**]

**def** **get\_scalar\_mul**(fvector, svector):

**return** fvector[**0**] \* svector[**0**] + fvector[**1**] \* svector[**1**]

**def** **get\_normal**(dot\_start, dot\_end, dot\_check):

vect = get\_vect(dot\_start, dot\_end)

normal = None

**if** vect[**0**] == **0**:

normal = [**1**, **0**]

**else**:

normal = [-vect[**1**] / vect[**0**], **1**]

**if** get\_scalar\_mul(get\_vect(dot\_end, dot\_check), normal) < **0**:

**for** i **in** range(len(normal)):

normal[i] = -normal[i]

**return** normal

**def** **get\_normals**(cut):

normals = []

cutlen = len(cut)

**for** i **in** range(cutlen):

normals.append(get\_normal(cut[i], cut[(i + **1**) % cutlen], cut[(i + **2**) % cutlen]))

**return** normals

**def** **check\_cut**(cut):

**if** len(cut) < **3**:

**return** False

vect1 = get\_vect(cut[**0**], cut[**1**])

vect2 = get\_vect(cut[**1**], cut[**2**])

sign = None

**if** get\_vect\_mul(vect1, vect2) > **0**:

sign = **1**

**else**:

sign = -**1**

**for** i **in** range(**3**, len(cut)):

vecti = get\_vect(cut[i-**2**], cut[i-**1**])

vectj = get\_vect(cut[i-**1**], cut[i])

**if** sign \* get\_vect\_mul(vecti, vectj) < **0**:

**return** False

**if** sign < **0**:

cut.reverse()

**return** True

**def** **is\_in\_section**(point, section):

fvect = get\_vect(point, section[**0**])

svect = get\_vect(\*section)

**if** abs(get\_vect\_mul(fvect, svect)) <= **1e-6**:

**if** section[**0**] < point < section[**1**] **or** section[**1**] < point < section[**0**]:

**return** True

**return** False

**def** **get\_sections**(section, points):

new\_points = [\*section]

**for** point **in** points:

**if** is\_in\_section(point, section):

new\_points.append(point)

new\_points.sort()

sections = []

**for** i **in** range(len(new\_points) - **1**):

sections.append([new\_points[i], new\_points[i + **1**]])

**return** sections

**def** **check\_vis**(point, dot\_start, dot\_end):

fvect = get\_vect(dot\_start, dot\_end)

svect = get\_vect(dot\_start, point)

**if** get\_vect\_mul(fvect, svect) >= **0**:

**return** True

**return** False

**def** **get\_intersection**(section, edge, normal):

vect = get\_vect(section[**0**], section[**1**])

w\_vect = get\_vect(edge[**0**], section[**0**])

vect\_scal = get\_scalar\_mul(vect, normal)

w\_wect\_scal = get\_scalar\_mul(w\_vect, normal)

diff = [section[**1**][**0**] - section[**0**][**0**], section[**1**][**1**] - section[**0**][**1**]]

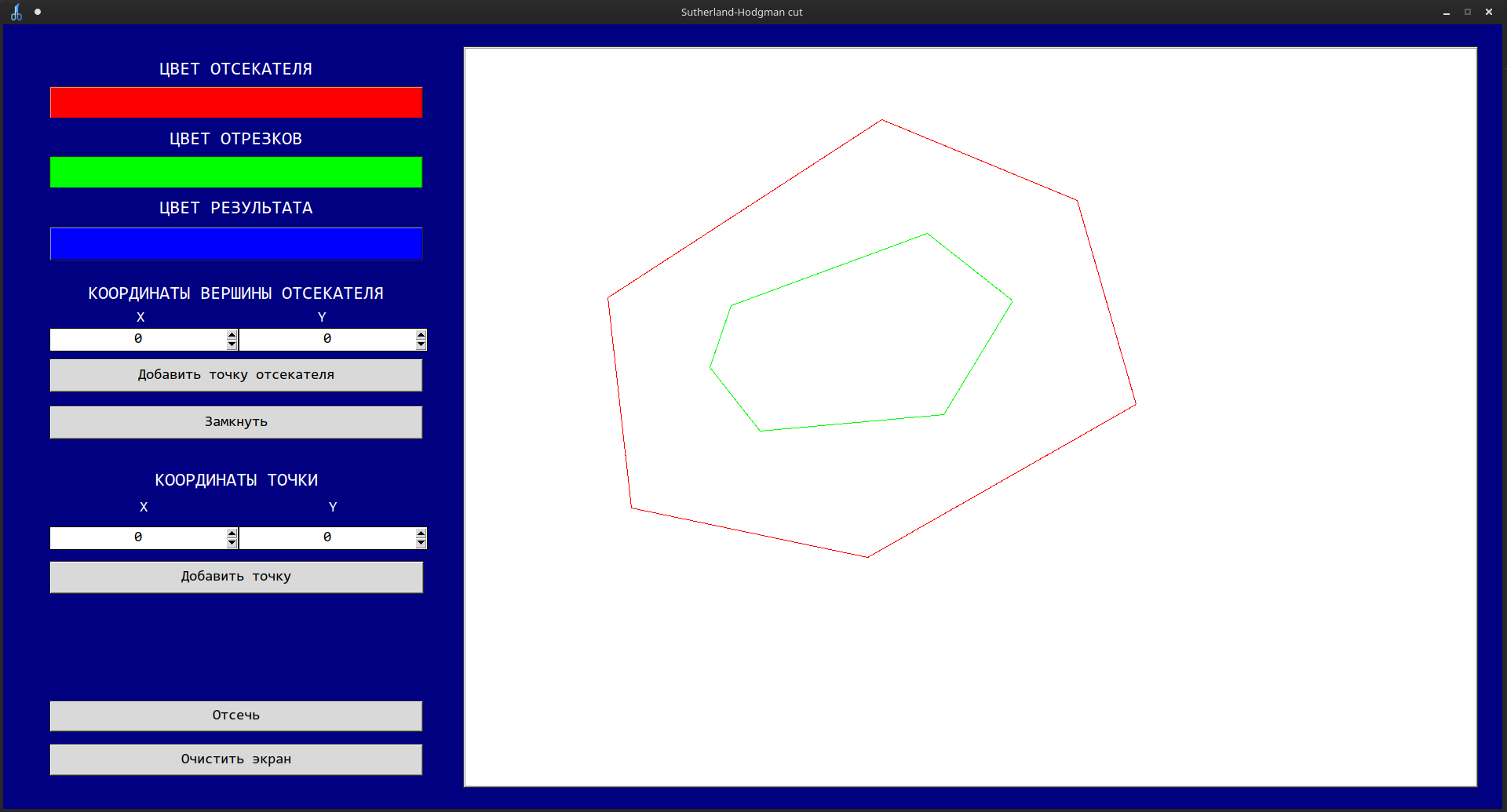
t = -w\_wect\_scal / vect\_scal

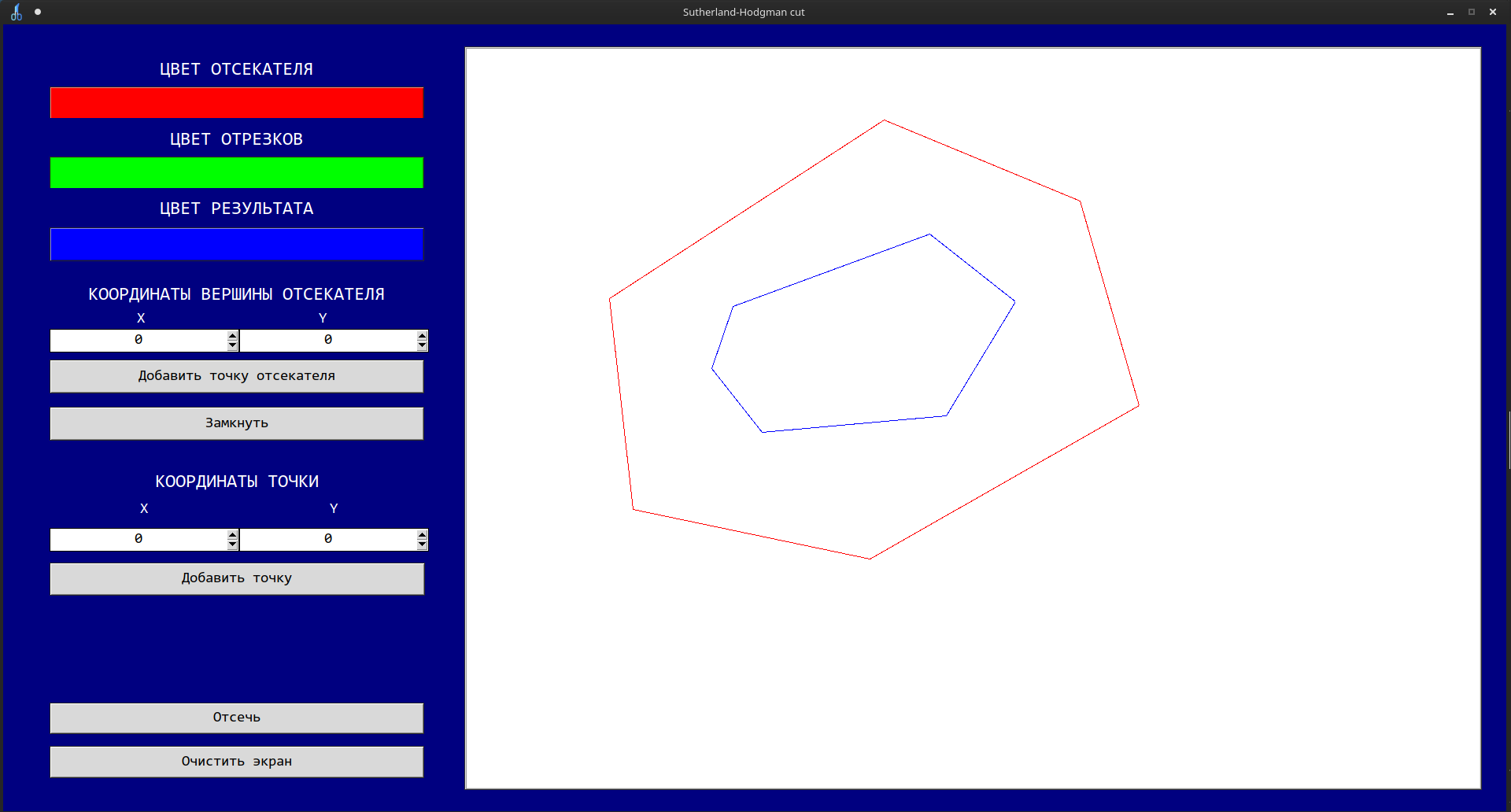
**return** [section[**0**][**0**] + diff[**0**] \* t, section[**0**][**1**] + diff[**1**] \* t]

**5. Демонстрация работы программы**

Многоугольник внутри отсекателя.

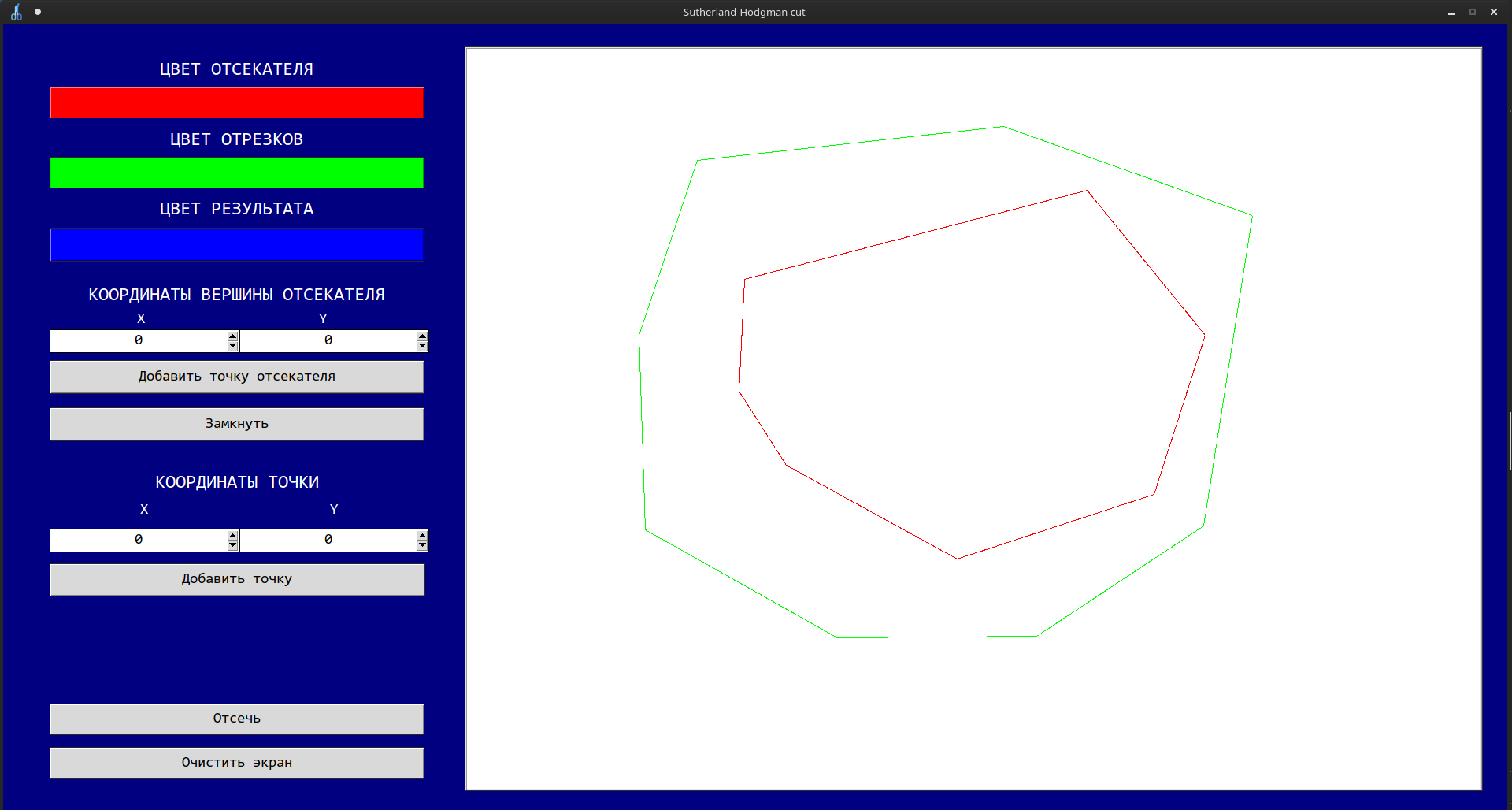
До:

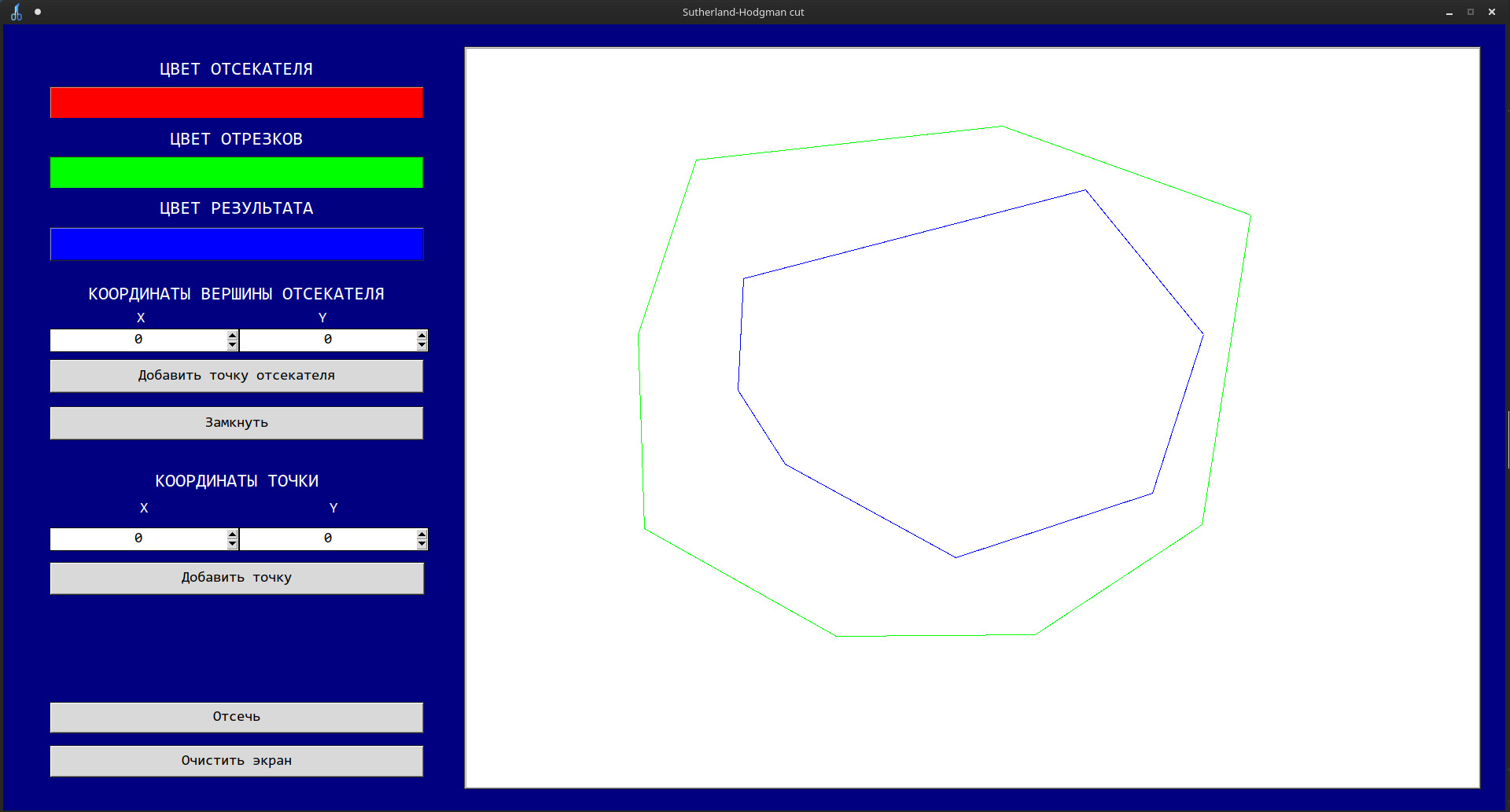
После:



Многоугольник снаружи отсекателя.

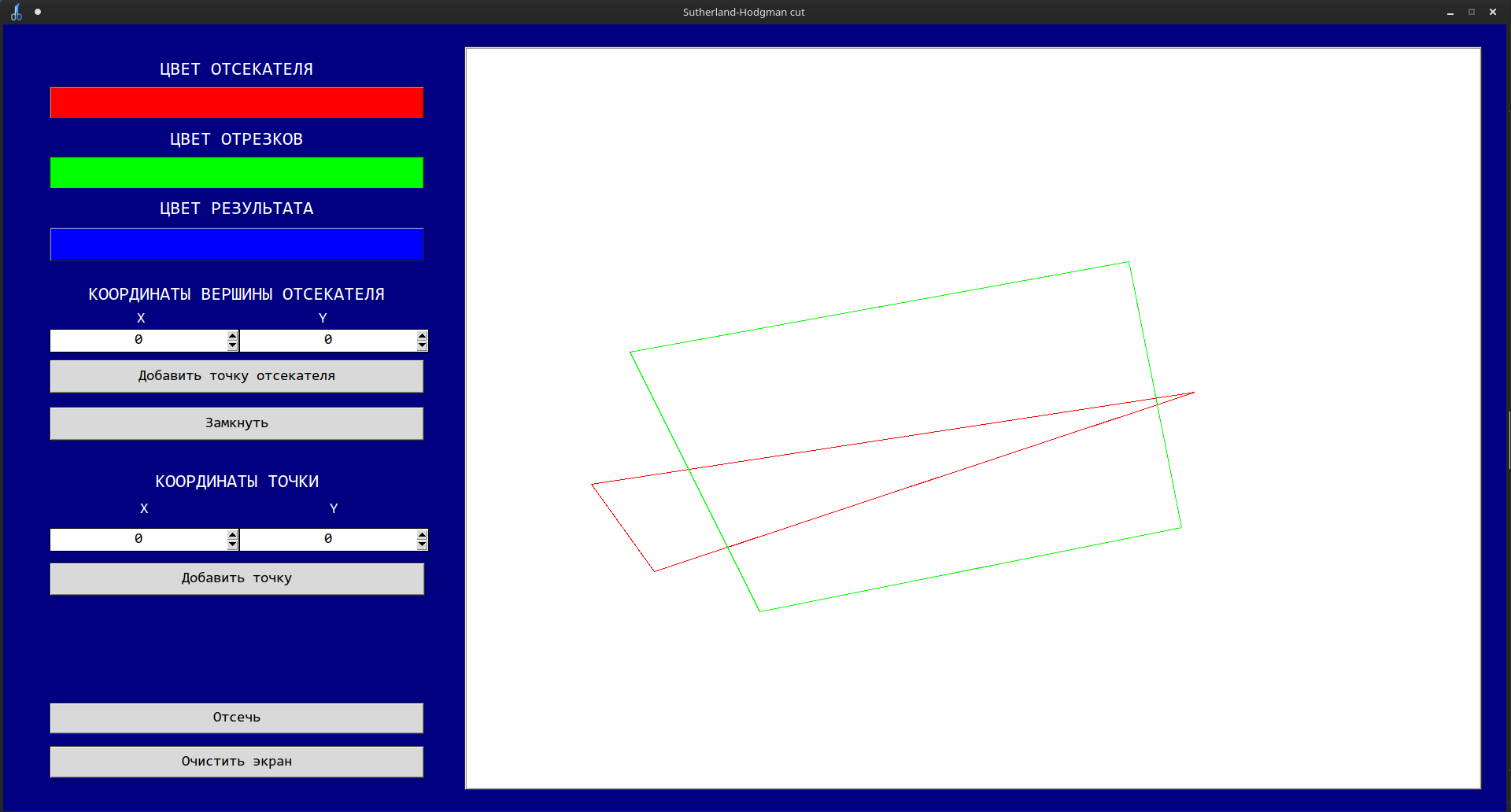
До:

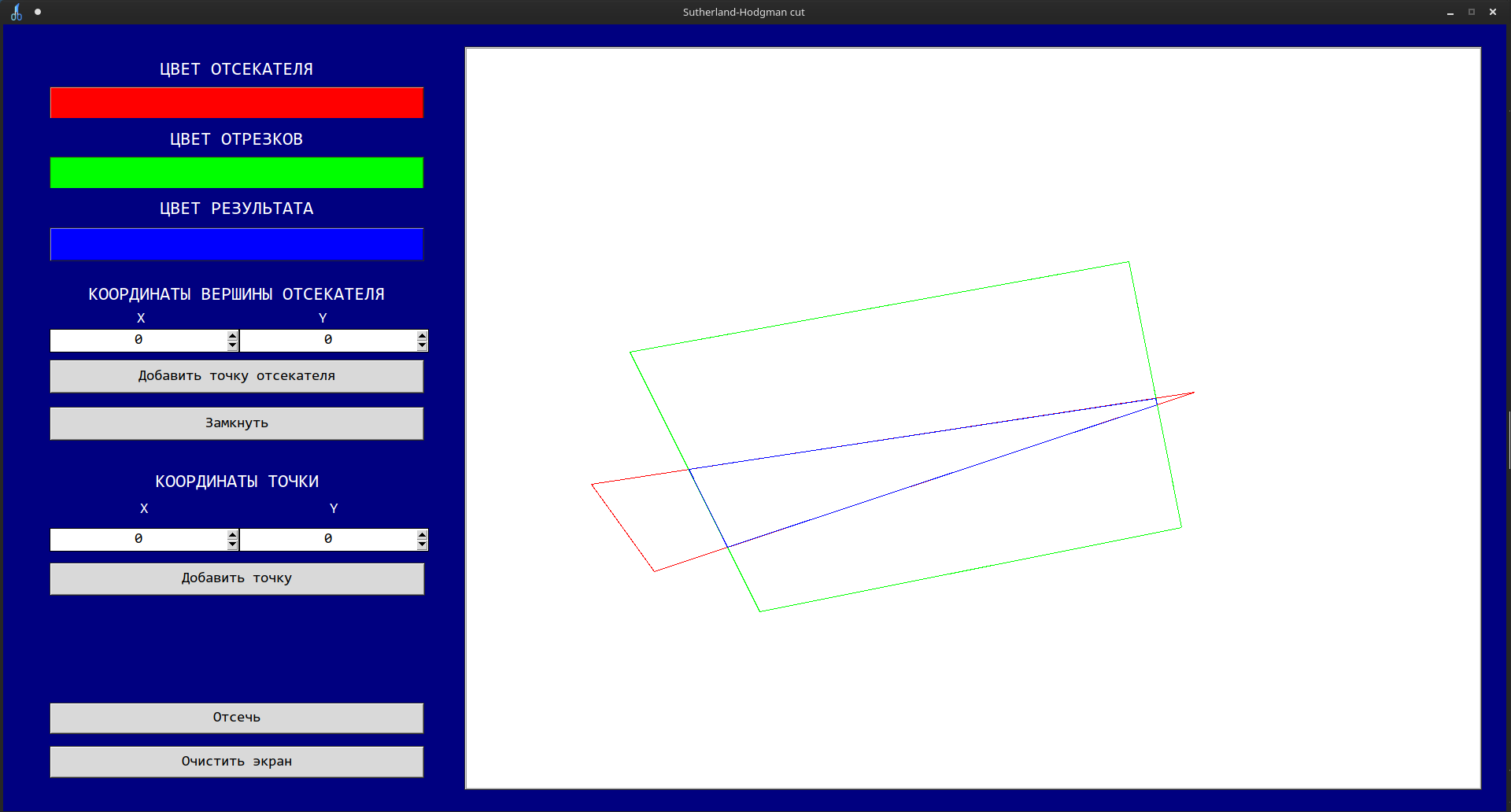
После:



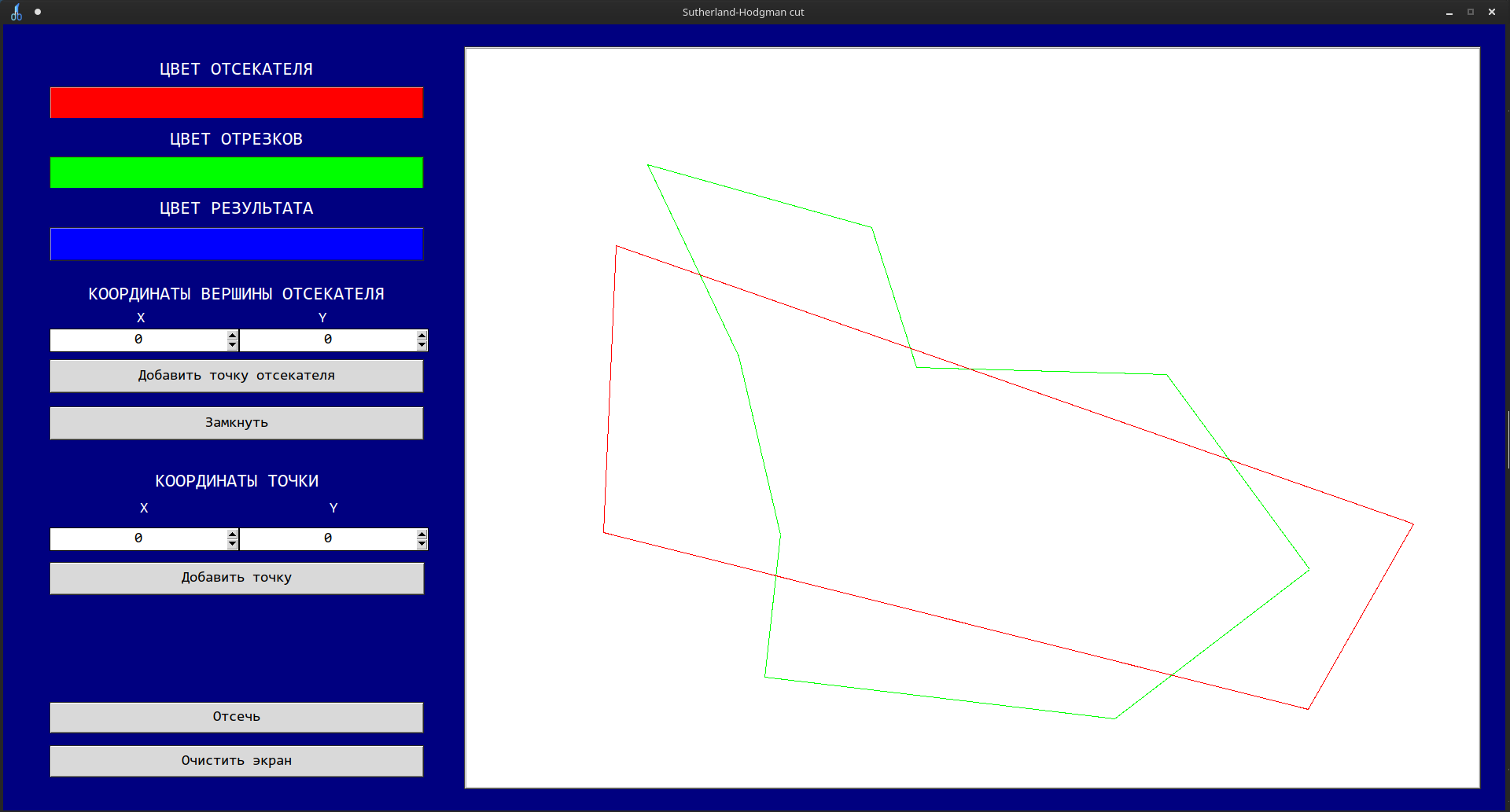
Многоугольник внутри отсекателя.

До:

После:



До:

После:

