|  |  |
| --- | --- |
| **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»

**Лабораторная работа №6**

**по дисциплине «Компьютерная графика»**

|  |  |
| --- | --- |
| **Тема:** Реализация и исследование построчного затравочного заполнения сплошных областей.  **Студент:**  **Группа:** ИУ7-45Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель:** Куров А. В. |  |

Москва.

2020 г.

Цель работы:

Реализация и исследование алгоритма построчного затравочного заполнения.

Необходимо обеспечить ввод произвольной многоугольной области, содержащей произвольное количество отверстий. Ввод (вершин многоугольника) производить с помощью мыши, при этом для удобства пользователя должны отображаться ребра, соединяющие вводимые вершины. Предусмотреть ввод горизонтальных и вертикальных ребер. Должен быть предусмотрен ввод затравочной точки.

Пользователь должен иметь возможность задания цвета заполнения.

Работа программы должна предусматривать два режима – с задержкой и без задержки. Режим с задержкой должен позволить проследить выполняемую последовательность действий.

(Задержку целесообразно выполнять после обработки очередной строки).

Обеспечить замер времени выполнения алгоритма (без задержки, с выводом на экран только окончательного результата).

Продемонстрировать возможность заполнения с помощью затравочного алгоритма произвольной области, ограниченной замкнутой кривой линией.

**Теоретический материал**

Алгоритмы заполнения с затравкой основываются на следующем подходе: в них предполагается, что известен хотя бы один пиксел из внутренней области фигуры, отталкиваясь от которого алгоритм находит и закрашивает все остальные пикселы, лежащие во внутренней области.

Обычно (в том числе и в нашей лабораторной работе) для определения внутренней области фигуры используют гранично-определенные области, то есть такие области, которые определены выделенными пикселами на границе (все пикселы на границы имеют такой цвет, который не может иметь ни один из пикселей, принадлежащих внутренней области). Алгоритмы, работающие с определенными таким методом областями называются *гранично-заполняющими.*

Области бывают 4-связными и 8-связными. 4-связный область – область, в которой любой пиксел можно достичь с помощью комбинации движений в 4 направлениях (вверх, вниз, влево, вправо). 8-связные области --- в 8 направлениях (добавляются диагональные). Очевидно, что алгоритм заполнения 8-связной области легко заполнит 4-связную область, но обратное – неверно.

Основная идея любого алгоритма с затравкой (в данной работе я буду рассматривать алгоритмы только на примере гранично-определенных областей, для других – подход аналогичный) заключается в следующем:

Зная, что затравочный пиксел содержится в области, которую надо закрасить мы проверяем его *соседей*: каждый соседний пиксел, не являющийся граничным или уже закрашеным мы запоминаем как затравочный (то есть принадлежавший внутренней области и подлежащий закраске). Перебрав все пиксели таким образом мы сможем найти и закрасить все пикселы, принадлежащие внутренней области.

Однако, как можно заметить, самым очевидным решением является рекурсивный алгоритм, но, как известно, рекурсия – не самый быстрый вариант. Для преобразования рекурсивных алгоритмов в итеративные часто используется такая СД, как стек (работает по принципу “первый пришел – последний ушел”). Вместо вызова в какой-то момент функции с другим объектом (как это происходит в рекурсии), мы запоминаем этот объект в стеке, чтобы обработать его на одной из следующих итераций.

Самый простой алгоритм заполнения с затравкой перебирает все пикселы “без разбора” (если так можно сказать): каждый сосед затравочного пиксела попадает в стек затравочный пикселей; как только рассмотрены все соседи текущего затравочного пиксела, из стека берется следующий затравочный пиксел (однако если он уже был обработан, то происходит просто переход к следующему в стеке пикселу) и так до тех пор, пока стек не будет пустым. Однако очевидно, что пикселы могут попадать в стек более 1 раза, так как у них имеется 4 соседа, которые, становясь затравочными, добавляют их в стек. В этом и есть главная проблема данного алгоритма: тяжело предположить, как много памяти потребуется для хранения этих затравочных пикселей.

Проблему алгоритма выше решает *построчный алгоритм заполнения с затравкой.* В алгоритме выше стек содержит много ненужной и дублирующей информации. Поэтому в построчном алгоритме было решено хранить только один затравочный пиксел для каждого непрерывного интервала на сканирующей строке. Однако и изменились действия для каждого из затравочный пикселей в стеке. Теперь при рассмотрении пиксела делаются следующие действия:

1. От текущего затравочного пиксела помечаются все расположенные в непрерывном интервале пикселы справа (идем направо и заполняем **все** подряд пикселы до тех пор, пока не встречается уже помеченный или граничный пиксел).

Запоминаем крайний *правый* пиксел.

2. Аналогично поступаем двигаясь в левом направлении. Таким образом мы закрасим весь строчный непрерывный интервал в котором находился затравочный пиксел. На данном шаге также запоминаем крайний *левый* пиксел.

3. Рассматриваем (проверяем/сканируем) верхнюю и нижнюю строки в интервале [Хлев (Х крайнего левого пиксела, который мы запомнили на 2 шаге); Хправ (Х крайнего правого пиксела, запомненного на 1 шаге)]. Во всех неперывных интервалах не закрашенных и не граничных пикселей,

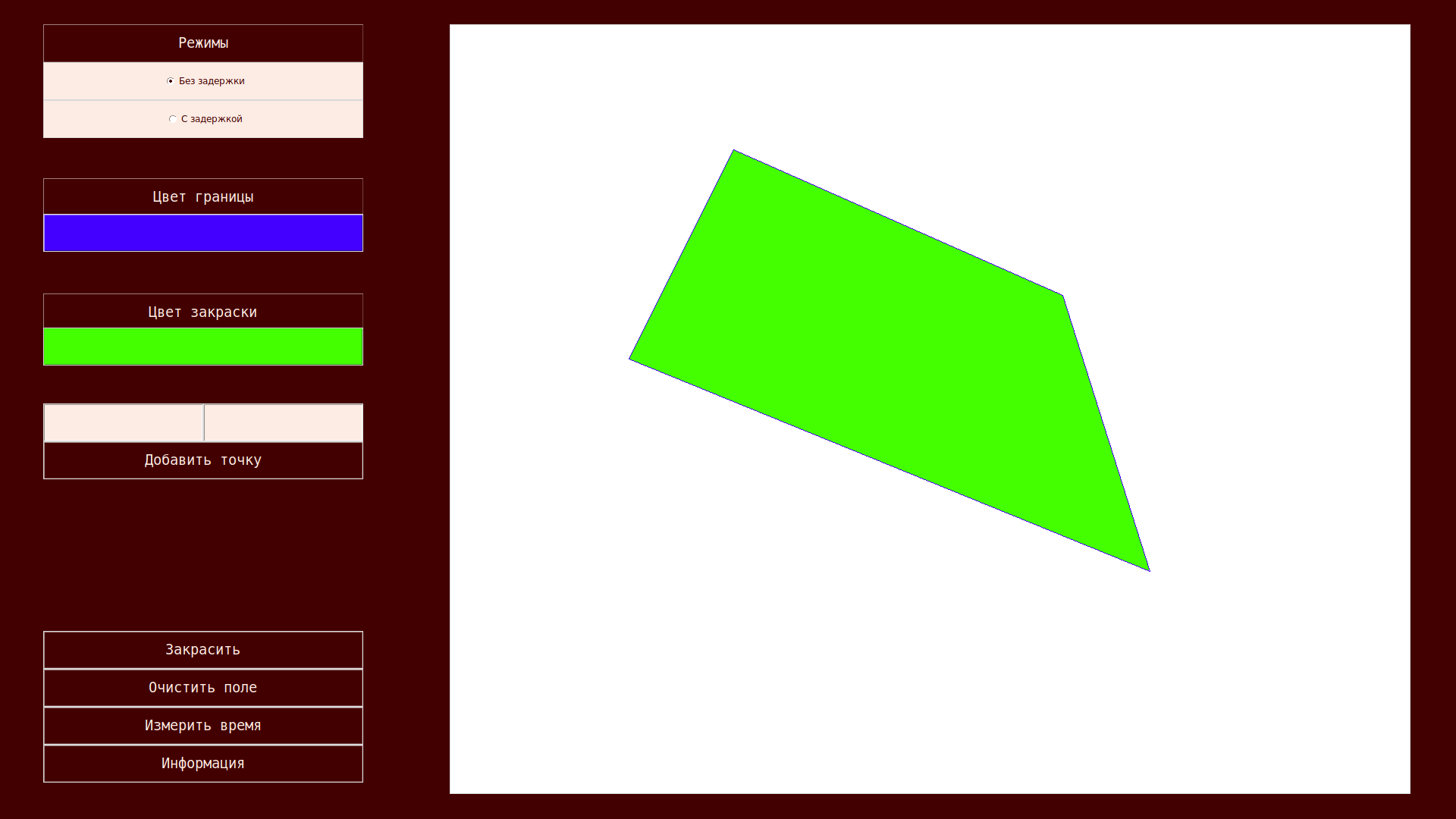
отмечаем самый правый пиксел (либо с координатой Xправ, если координата Х самого правого больше).

**Исходный код. Примеры работы программы.**

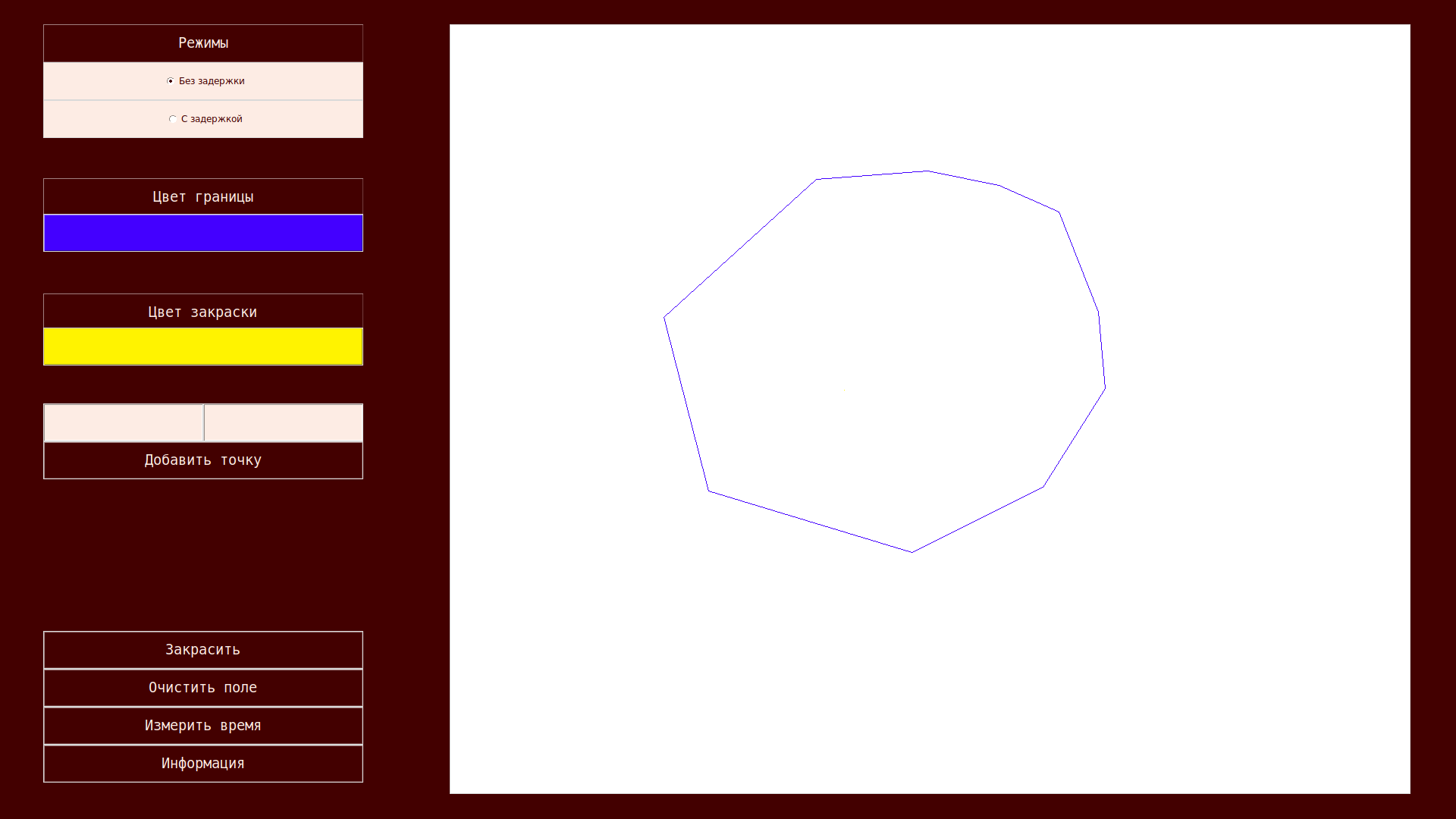


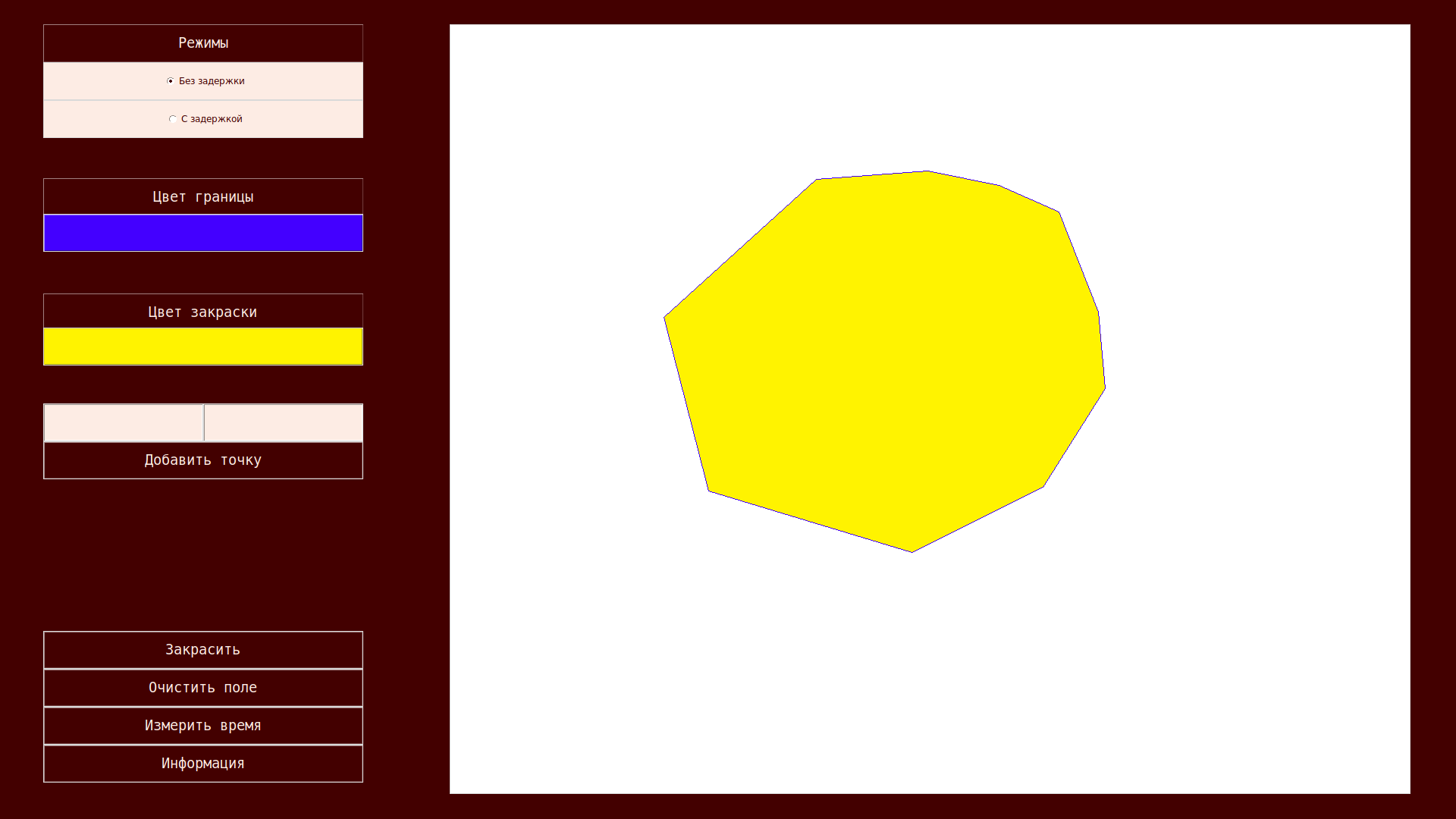
Код (стек изначально содержить только затравочный пиксел):

Интерфейс:

**

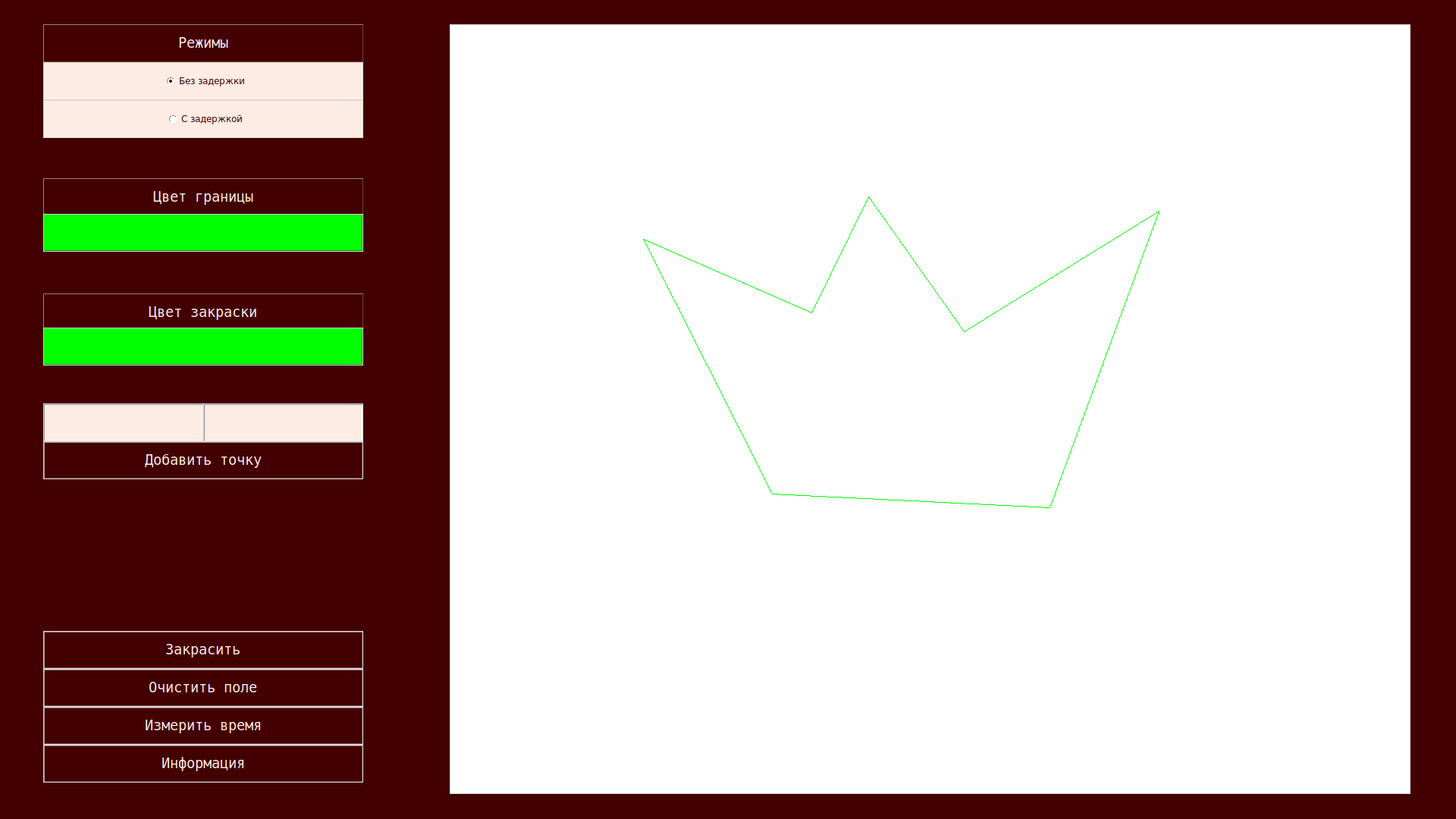
Заполнение произвольного выпуклого многоугольника:

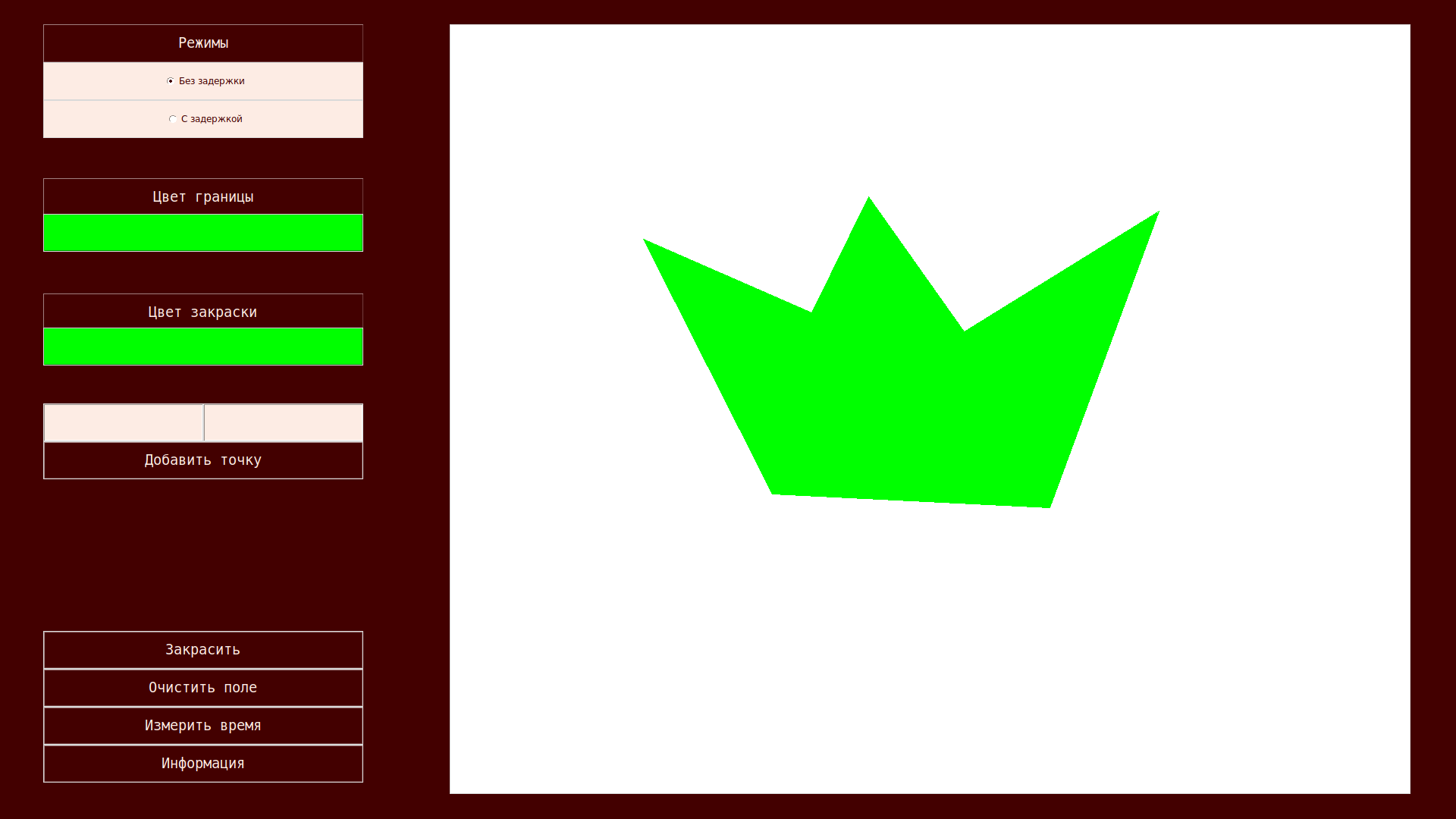




Время заполнения: 0.133424 секунды.

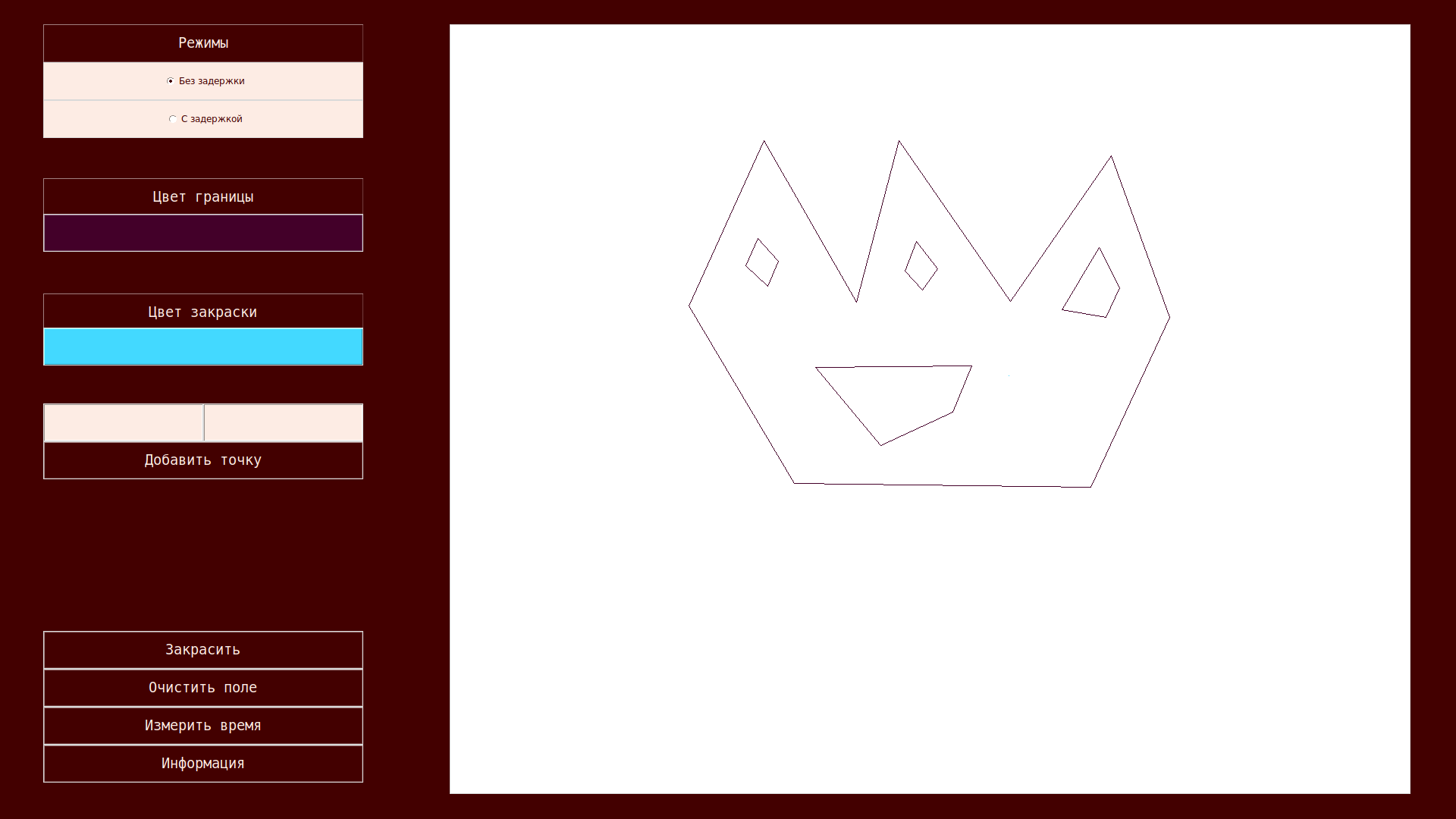
Заполнение произвольного невыпуклого многоугольника:

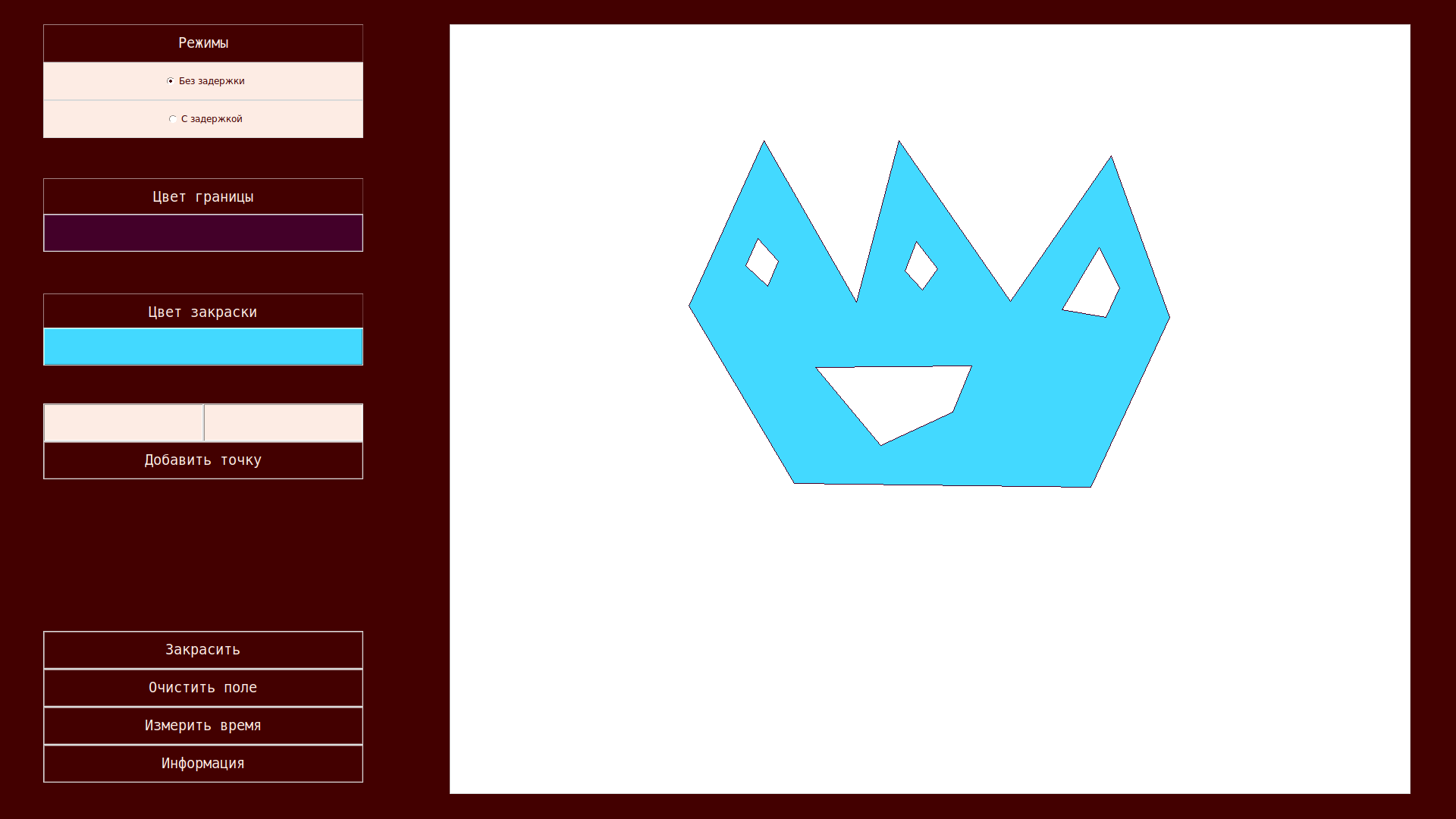




Время заполнения: 0.153424 секунды.

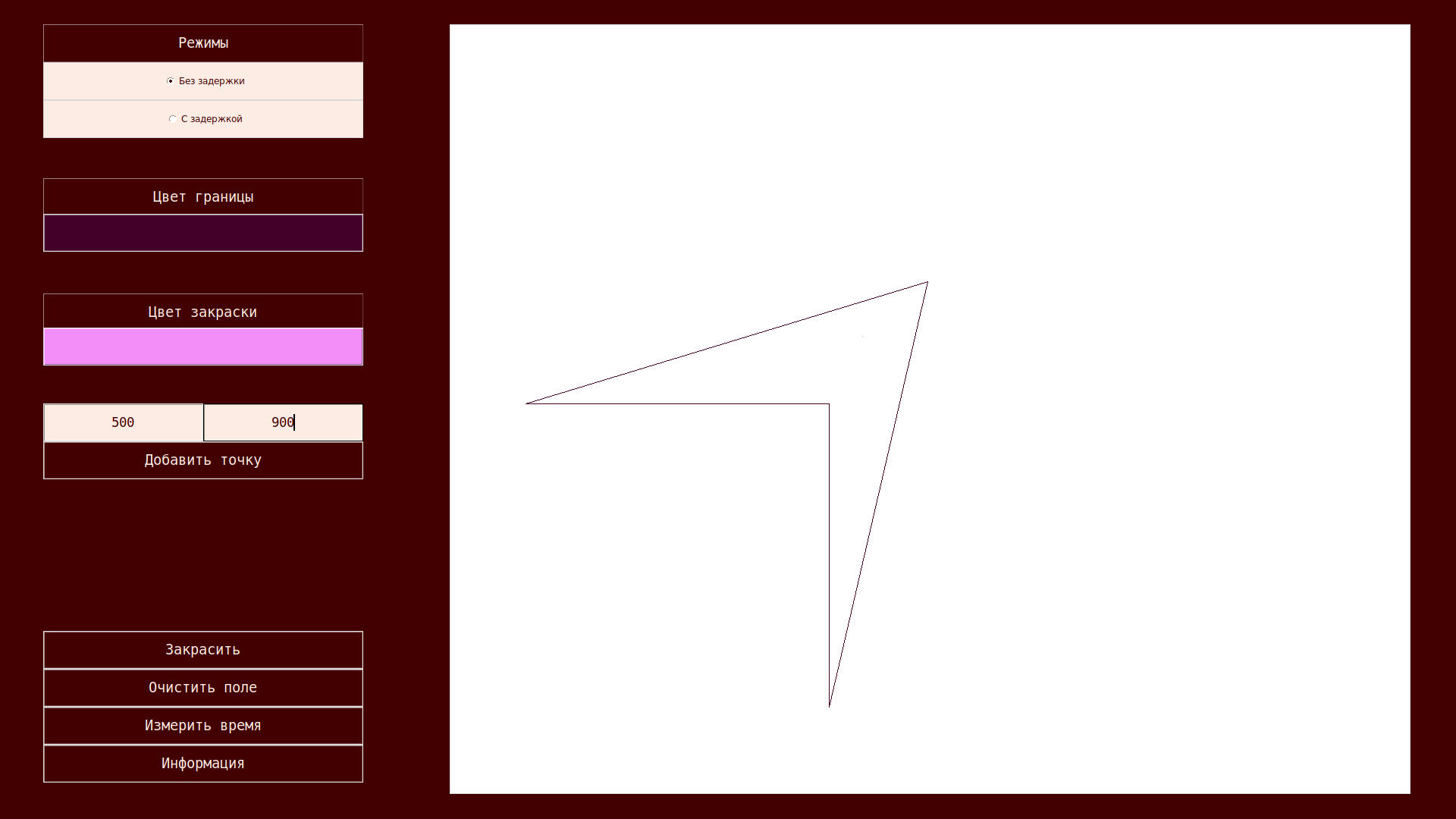
Заполнение произвольного многоугольника с отверстиями:

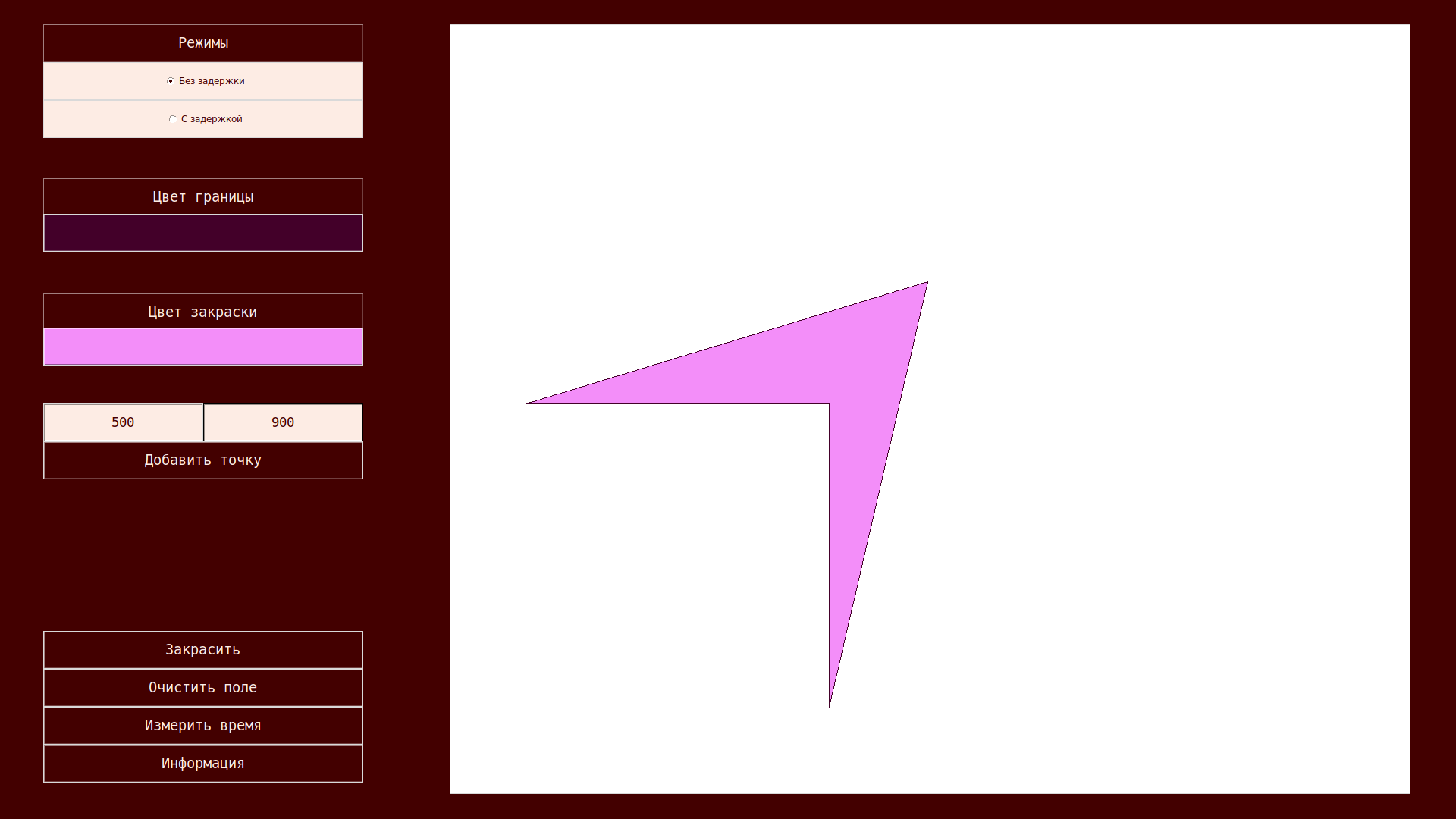




Время заполнения: 0.152019 секунды.

Заполнение многоугольника с вертикальными и горизонтальными прямыми:





Время заполнения: 0.1160235 секунды.

Заполнение произвольного многоугольника с задержкой (стрелка указывает расположение затравочного пиксела):

