|  |  |
| --- | --- |
| **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 № 3**

|  |  |
| --- | --- |
| **Тема** Реализация и исследование алгоритмов построения отрезков  **Студент** Пересторонин П. Г.  **Группа ИУ 7-43**  **Оценка (баллы)**  **Преподаватель Куров А. В.** |  |

Москва.

2020 г.

**Цель работы:**

Научиться выполнять построение отрезков различными алгоритмами и проанализировать их.

**Техническое задание:**

1. Рисование отдельных отрезков и сравнение их визуальных характеристик:

1. Алгоритм цифрового дифференциального анализатора;

2. Алгоритмы Брезенхема;

2.1. С дробными числами.

2.2. С целыми числами.

2.3. Со сглаживанием.

3. Алгоритм Ву;

4. Библиотечный алгоритм;

1. Исследование визуальных характеристик для отрезка, расположенного во всем спектре изменения углов;
2. Исследование временных характеристик (результат оформить в виде гистограммы);

**Теоретический материал:**

Процесс определения пикселей, наиболее эффективно аппроксимирующих отрезок, называется разложением отрезка в растр.

Общие требования к алгоритмам разложения отрезка в растр:

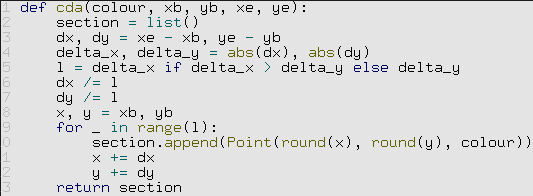
1. Отрезок должен выглядеть как отрезок прямой, начинаться и заканчиваться в заданных точках

2. Интенсивность (яркость) вдоль отрезка должна быть постоянной. Отрезки, имеющие разные углы наклона, должны быть одной интенсивности. Восприятие человека зависит не только от интенсивности свечения объекта, но и от расстояния между светящимися объектами (чтобы удовлетворить этому требованию, надо высвечивать точки с переменной интенсивностью от расстояния – потребует дополнительных вычислений, без особой нужды не используется)

3. Алгоритмы должны работать быстро.

Все алгоритмы имеют пошаговый характер – на очередном шаге высвечиваем пиксель, и производим вычисления, используемые в следующем шаге.

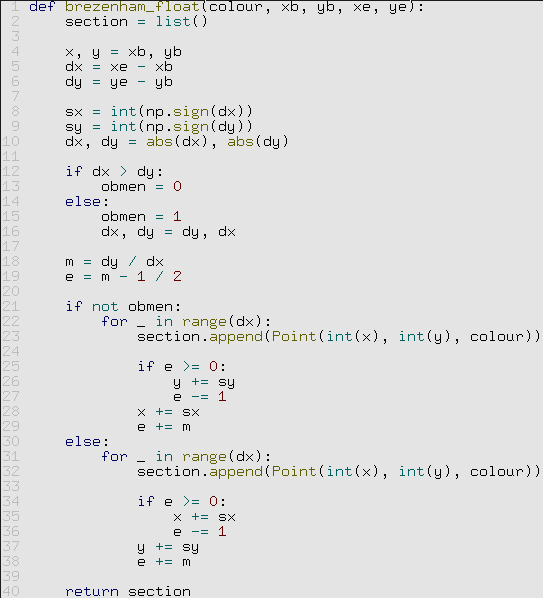
**Алгоритм цифрового дифференциального анализатора:**



Идея алгоритма в том, что мы используем подход «в лоб»: расчитываем производную (или, проще говоря, отношения приращения прямой к приращению ее аргумента) и понимаем, какое приращение имеет сама прямая относительно приращения аргумента (при чем в качестве приращения прямой берется меньшее по модулю из 2, чтобы приращение аргумента взять за единицу, а приращение функции было < 1 по модулю, и отрезок был непрерывным). Далее, прибавляя к аргументу единицу (потому что пиксел не может быть дробным, и не имеет смысла рассматривать меньшее приращение), а к значению прямой ее приращение (уi+1 = yi + Δy/Δx, где Δy/Δx — есть производная прямой) мы будем находить последующие пикселы. Очевидно, что аргумент будет считаться в целых числах, а значение прямой — в дробных (и округляться при расчете координат пиксела). Факт счета в дробных числах и округление делают этот алгоритм медленным.

**Алгоритм Брезенхема с действительными числами:**

Примечание: в своем алгоритме я вынес проверку переменной obmen за цикл, чтобы это действие не проводилось каждый раз в цикле. Это сделало код избытычным и менее читаемым, однако дало выигрыш по времени (от алгоритмов низкого уровня это требуется больше всего).



Алгоритм Брезенхема с действительными числами помог отказаться от округления при расчете координат пиксела. Брезенхем использовал те же приращения функции (дальше буду обозначать как Δ) и аргумента (Δх = 1), но чуть иначе. Для этого Брезенхем первым делом ввел понятие «ошибки». Изначально, ошибка измерялась в пределах

-0.5 < *e*i < 0.5. На начальном этапе она была равна 0 (так как начальные координаты задаются в целых числах и точно соответсвуют координатам пиксела). Далее аргументу прибавляется единица, а ошибке — приращение функции (xi+1 = xi + 1; ei + 1 = ei + Δ). Таким образом ошибка показывает, насколько рассматриваемое целое значение пиксела (значение функции) отличается от настоящего (вещественного) значения функции и является в каждый момент времени их разностью (то есть *e*i = yидеальное - yi). Когда это значение переваливает за полпиксела (полпиксела = 0.5; получается что пиксел выше будет ближе к настоящему значению), берется следующее значение функции (пиксел выше; уi + 1 = yi + 1), а из ошибки вычитается единица (так как теперь в качестве рассматриваемого значения берется значение, на единицу большее):

*e*i + 1 = ei  - 1 = yидеальное  - уi + 1 = yидеальное - yi - 1

Таким образом каждый раз, когда ошибка выходит за пределы (e*i > 0.5*), значение прямой получает приращение 1, и нет никакого округления. Однако мы видим, что в вычислениях ошибки участвует только предыдущее значение ошибки, поэтому, чтобы ускорить работу алгоритма на ЭВМ, можно сдвинуть границы ошибки и сделать так, чтобы сравнение с границами было сравнением с нулем (а как известно для определения знака числа достаточно проверить всего один бит), поэтому так же были произведены следующие преобразования:  
*e0 = 0 ==> e0 = -0.5*

*0.5 < ei < -0.5 ==> 0 < ei < -1*

Таким образом значение ошибки сравнивается с нулем, что также может ускорить работу (если действительно сделать сравнение 1 бита)

**Алгоритм Брезенхема с целыми числами:**

Идея данного алгоритма опять же опирается на то, что ошибка вычисляется только относительно своего предыдущего значения путем прибавления Δ, при этом во всех вычислениях действительными являются только ошибка и Δ. Таким образом, сделав эти значение целочисленными, мы сможем прийти к целочисленному алгоритму. Это возможно, преобразовать следующие вычисления:

*Δ = dy / dx ==> Δ = 2dy (умножилось на 2dx)*

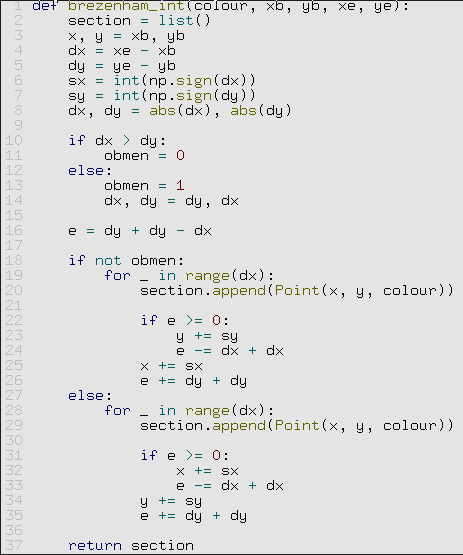
*e0 = -0.5 ==> e0 = -dx (умножилось на 2dx)*

*0 < ei < -1 ==> 0 < ei < -2dx (умножилось на 2dx)*

*ei + 1 = ei + Δ = ei + dy / dx ==> ei + 1 = ei + Δ = ei + 2dy*

*dx = xк - хн; dy = yк - yн — целые значения, потому что координаты пиксела — целые.*

Таким образом мы приходим к вычислениям в целых числах, что значительно ускоряет построение отрезка на ЭВМ.

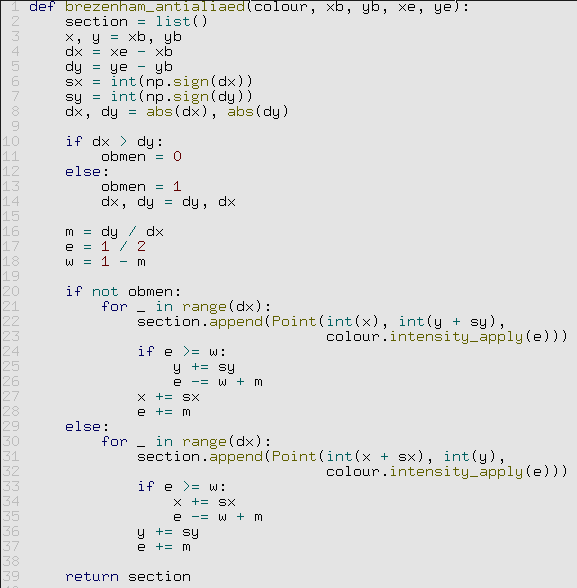
******

**Алгоритм Брезенхема построения отрезка с устранением ступенчатости:**

Данный алгоритм чаще всего используется при отображении ребер многоугольника, который закрашивается. Идея состоит в сглаживании резких переходов от ступени к ступени. Сглаживание основывается на том, что каждый пиксель высвечивается со своим уровнем интенсивности. Уровень выбирается пропорционально площади части пикселя, находящейся под «идеальной» прямой. При рассмотрении площади пиксел считается квадратом со стороной 1, прямая делит пиксел на 2 части, таким образом площадь под прямой будет иметь значение 1 < S < 0, что можно расценивать как «доля» интенсивности (1 — максимальная интенсивность, 0 — минимальная, 0.5 — средняя и т. д.). Приращение площади можно связать с приращением функции (что то же: с приращением ошибки). Таким образом начальная площадь = 0.5 (прямая проходит через центр квадрата), а следующее значение находится как:

Si + 1 = Si + m / 2, где m = dy / dx = приращение функции (прямой).

Как только значение площади становится больше единицы (получается мы захватываем еще долю пикселя выше) ==> вычитаем единицу и рассматриваем площадь пикселя выше.



**Алгоритм Ву:**

Алгоритм Ву — один из алгоритмов построения сглаженных прямых двойной толщины.

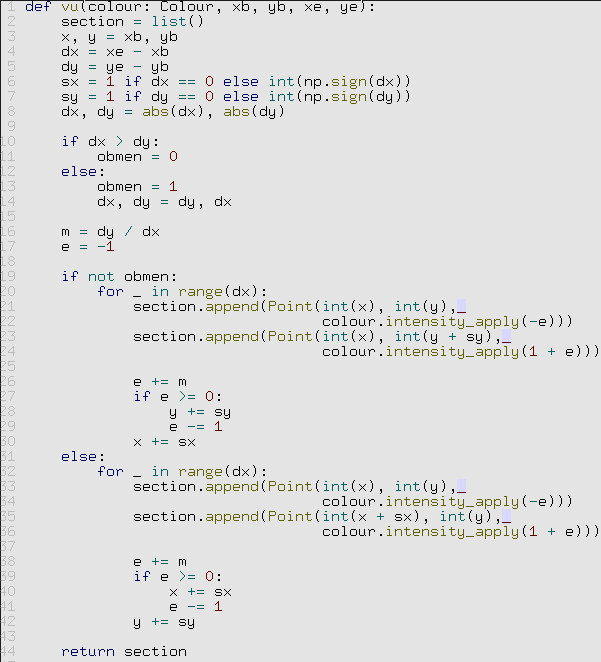
Идея алгоритма основана на 4 принципах:

1. Отрезок высвечивается толщиной в 2 пикселя.

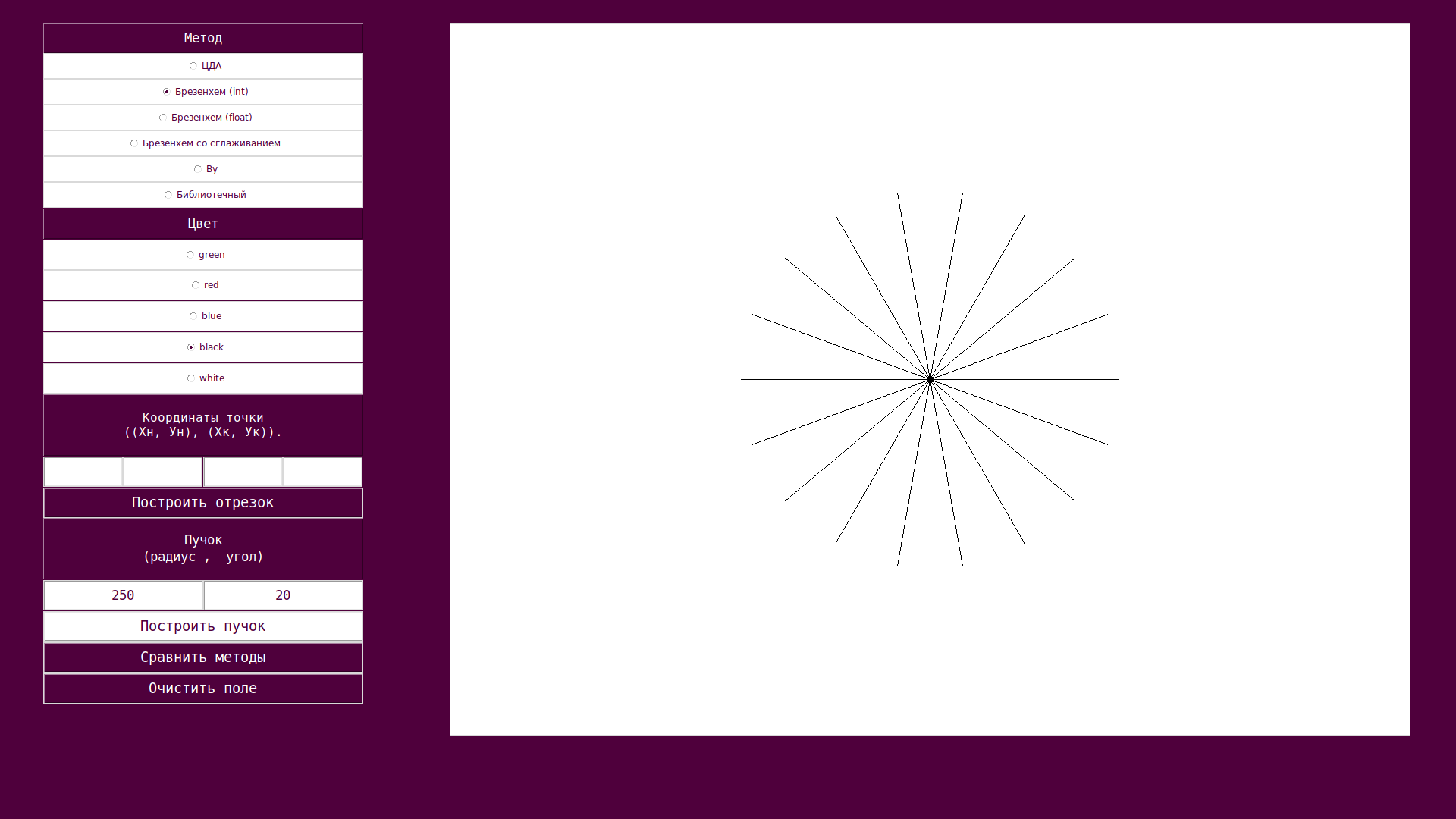
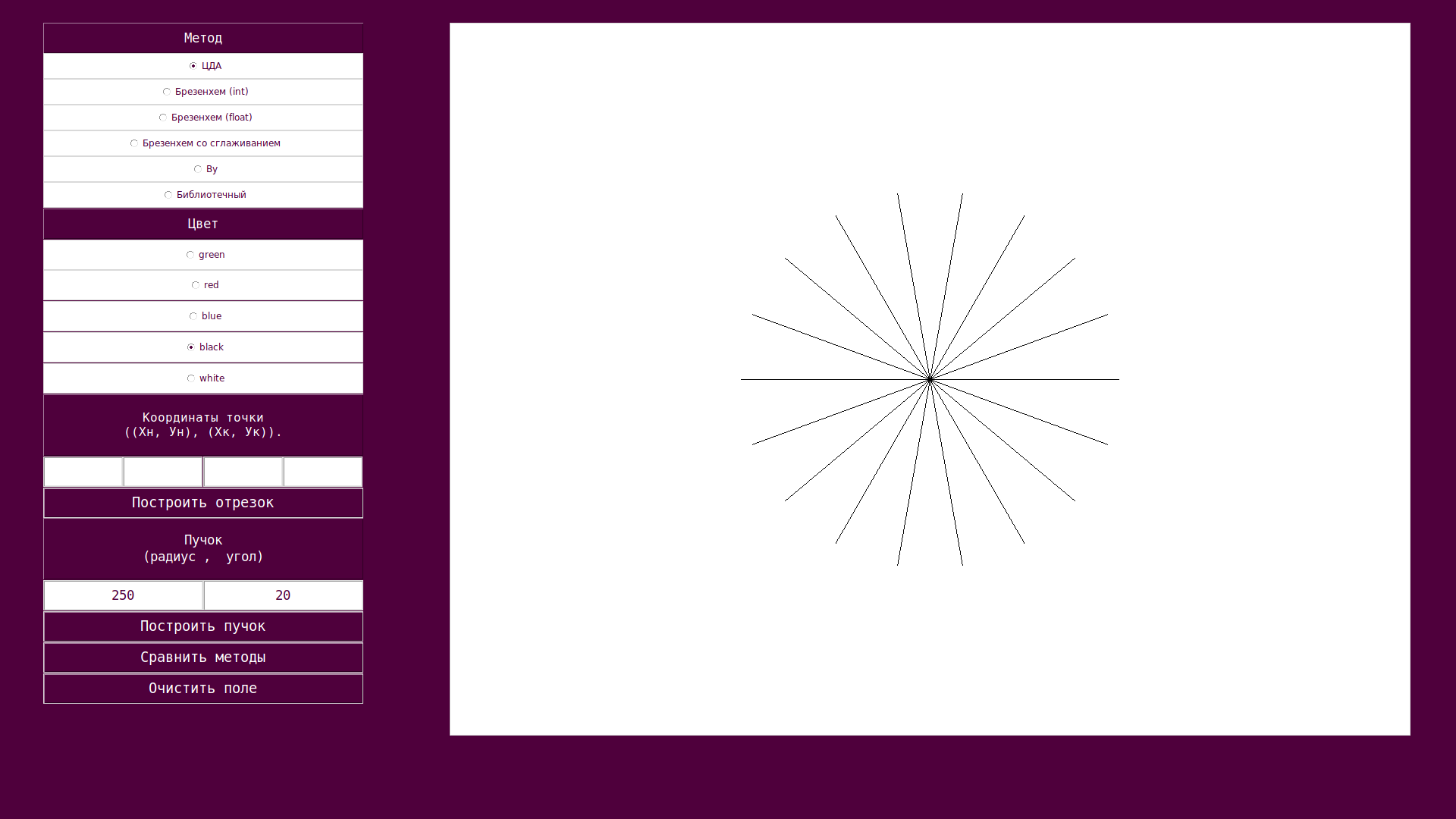
2. Суммарная интенсивность 2 используемых пикселей на каждом шаге постоянна (*Iconst = I1 + I2).*

3. Сглаживание осуществляется за счет перераспределения интенсивности Iconst между 2 пикселями.

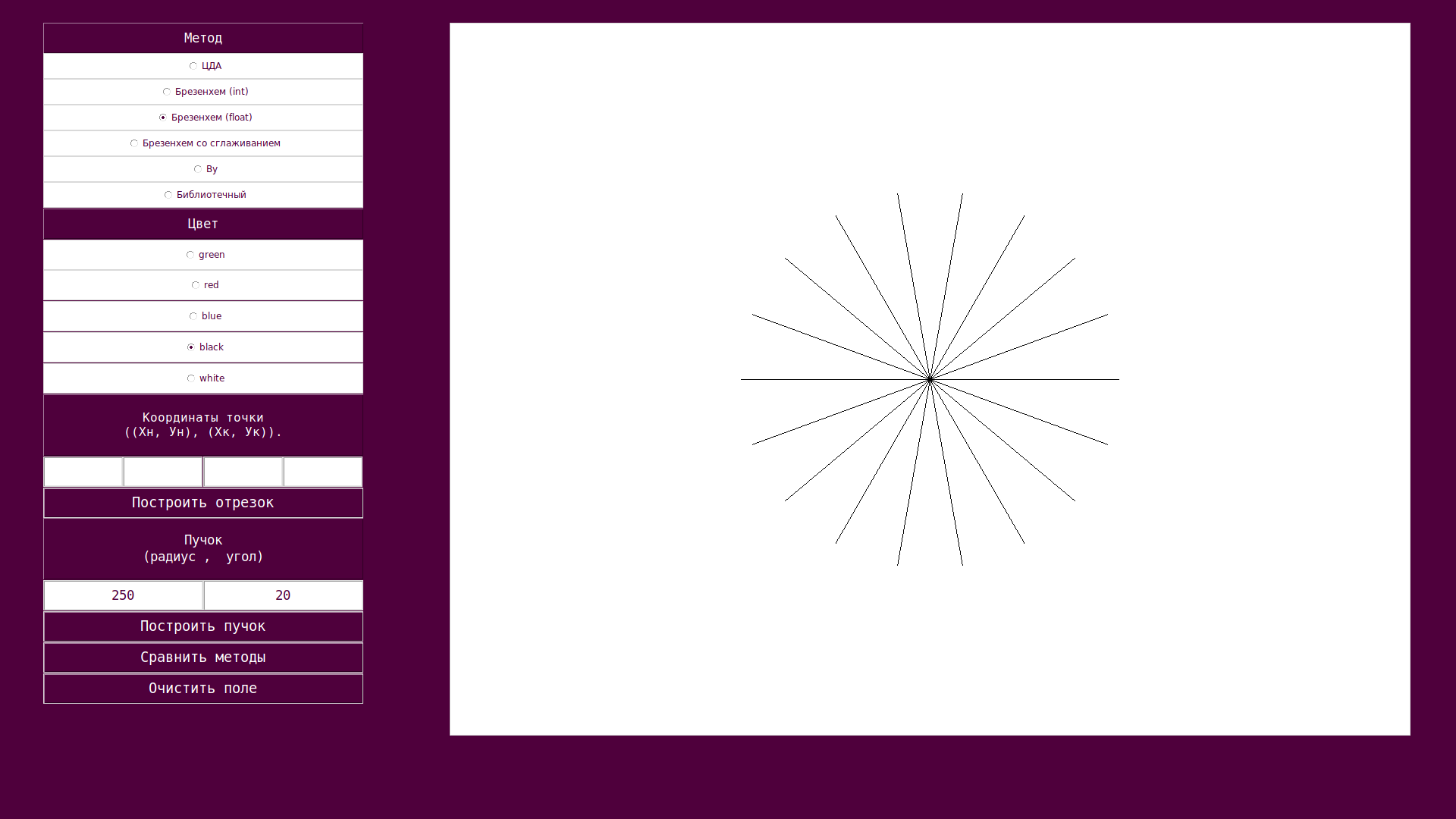
4. Интенсивность пиксела выбирается (задается) в зависимосит от расстояния между пикселем и точкой, расположенной на идеальном отрезке: больше его интенсивность.

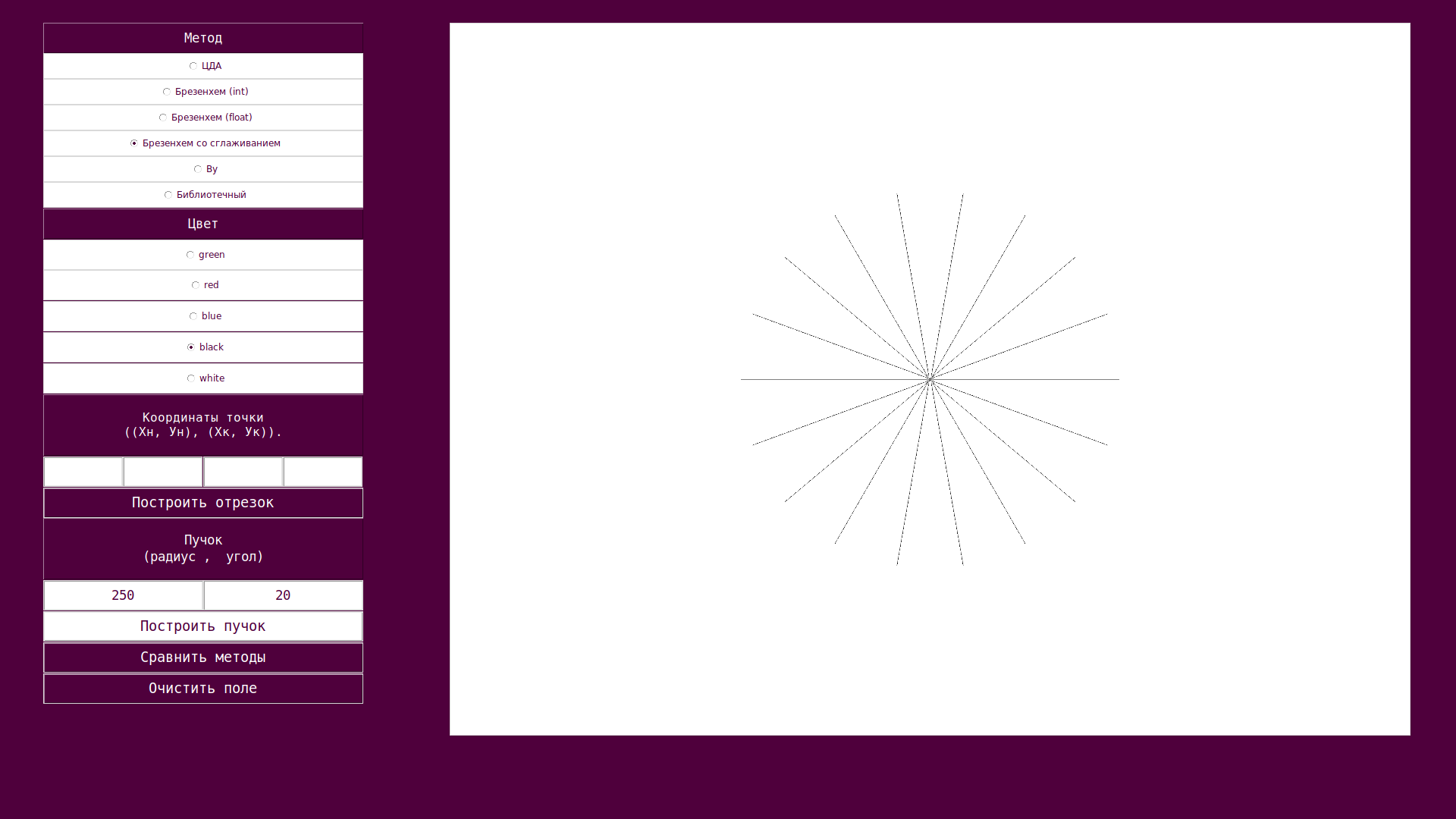


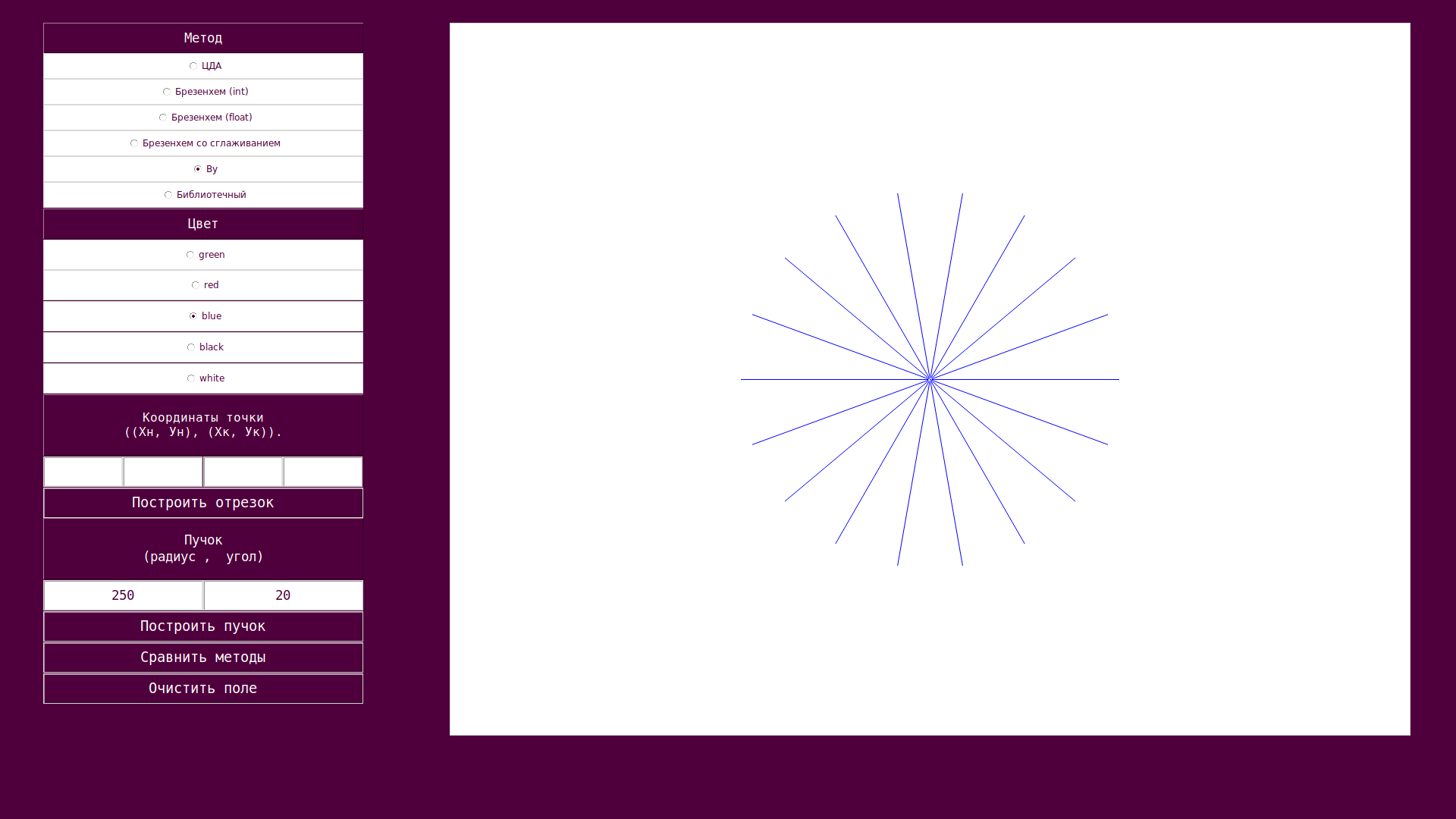
**Вывод пучков для всех алгоритмов (метод указан на скриншотах интерфейса, идет по порядку, указанному там же)**

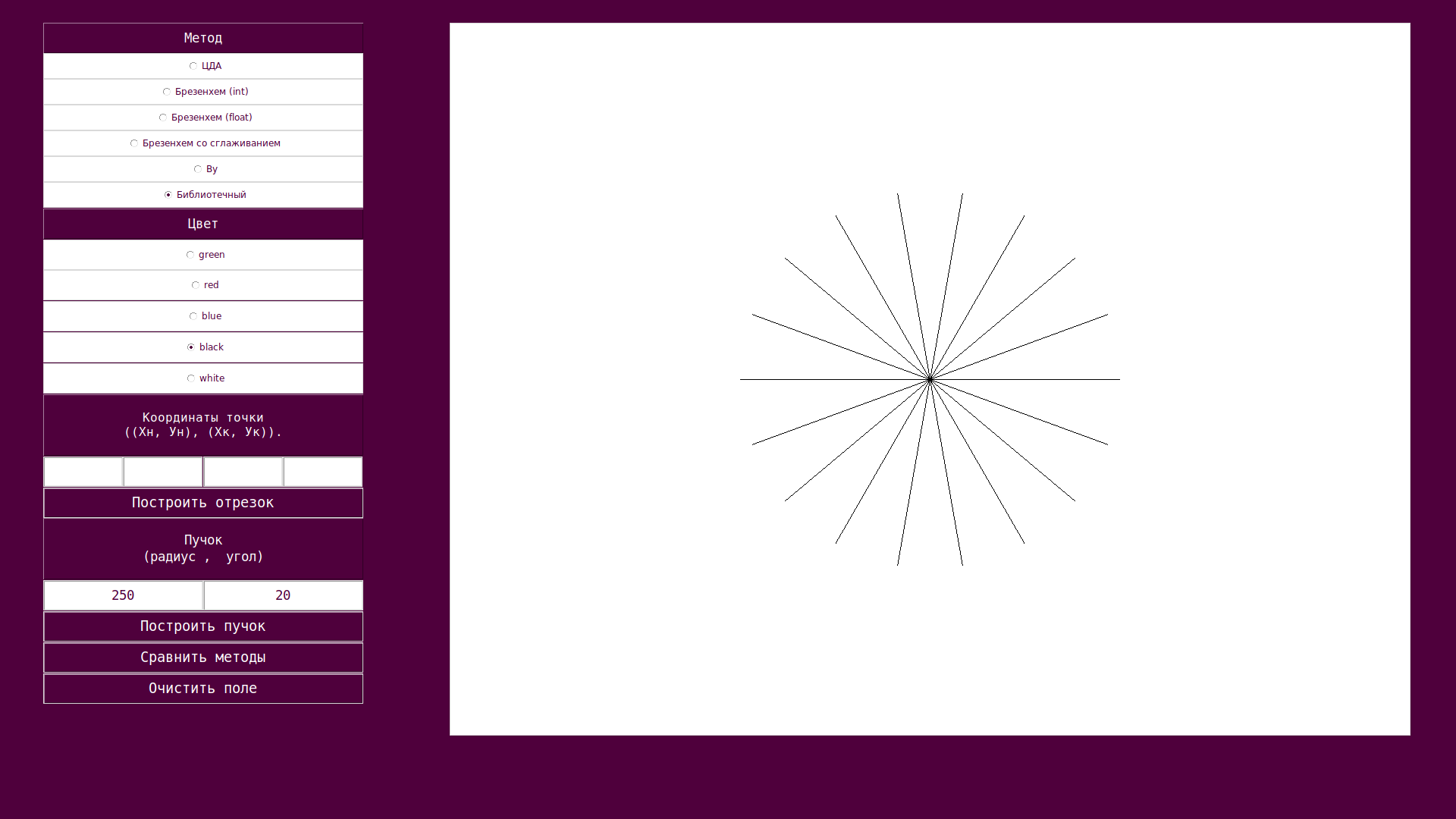
**1.ЦДА.**

**2. Брезенхем (int).**

**3. Брезенхем (вещественные числа).**

**4. Брезенхем со сглаживанием.**

**5. Ву.**

**6. Библиотечный алгоритм.**

**Выводы по результатам рисования.**

1. Скорость.

Если нас интересует скорость построения отрезков в первую очередь, то алгоритмы располагаются в следующем порядке (про преимущества можно почитать выше):

1. Брезенхем с целыми числами.

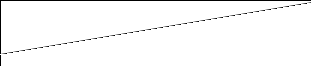
2. Брезенхем с вещественными числами.

3. ЦДА.

4. Брезенхем со сглаживанием (время тратится на расчет интенсивности, поэтому этот алгоритм предназначен немного для других целей)

**Дополнение.**

**1. Брезенхем со сглаживанием.**

Из рисунка видно, что сглаживание работает только на одну часть отрезка. Таким образом (как было указано выше) обычно сглаживают многоугольники с внешней стороны (в случае отрезка за внешнюю сторону бралась бы верхняя часть отрезка).

**2. Сравнение времен.**

Сравнение проводилось при построении пучка из 18 отрезков длиной в 250 пикселей.

