|  |  |
| --- | --- |
| **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 №7**

**по дисциплине «Компьютерная графика»**

|  |  |
| --- | --- |
| **Тема: Реализация алгоритма отсечения отрезка регулярным отсекателем.**  **Студент:** Пересторонин Павел  **Группа:** ИУ7-43Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель:** Куров А. В. |  |

Москва.

2020 г.

Цель работы: изучение и программная реализация алгоритма отсечения отрезка.

Необходимо обеспечить ввод регулярного отсекателя - прямоугольника. 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.

Ввод осуществлять с помощью мыши и нажатия других клавиш.

Выполнить отсечение отрезков, показав результат третьим цветом. Исходные отрезки не удалять.

Номер по списку: 16, алгоритм: простой.

**Теоретический материал**

Отсечение - это операция удаления изображения за пределами выделенной

области, называемой окном. Чтобы выполнить данную операцию, необходимо прежде всего задать тип отсекателя.

Регулярным (стандартным) отсекателем на плоскости является прямоугольник со сторонами, параллельными координатным осям объектного пространства или экрана. Такое окно задается левым, правым, верхним и нижним двумерными ребрами. Для выполнения отсечения необходимо задать абсциссы XЛ , XП левого и правого ребер и ординаты Yн ,Yв  нижнего и верхнего ребер. Цель отсечения будет состоять в определении точек, отрезков или их частей, которые лежат внутри отсекателя.

Отрезок целиком лежит внутри окна, если обе его концевые точки лежат внутри окна. Однако обратное утверждение, к сожалению, верно не всегда. Отрезок, концевые точки которого лежат вне окна, может быть как полностью невидимым, так и частично видимым. Полностью невидимым называется отрезок, целиком лежащий вне отсекателя. Частично видимым называется отрезок, одна часть которого лежит в пределах отсекателя, а другая - вне его. Если обе концевые точки отрезка невидимы, то он будет заведомо невидимым, если они (вершины отрезка) одновременно лежат левее или правее или ниже или выше окна.

Алгоритмы отсечения часто используются на низком уровне, поэтому от них требуют высокую эффективность и скорость. В связи с этим такие алгоритмы не должны делать лишней работы: существует 2 случая, когда путем небольших проверок заведомо можно сказать, что отрезок принадлежит целиком области внутри окна или же наоборот, отрезок целиком лежит вне окна и является невидимым:

1. Если обе вершины отрезка лежат внутри окна. Проверка самая простая:

***(у > YН) и (y < YВ)и (x > XЛ) и (x < XП)***

(Должно выполняться для обеих вершин), тогда отрезок принадлежит целиком области окна, можно его выводить.

2. Если обе вершины лежат по одну сторону вне окна (левее левой границы, или правее правой, или ниже нижней, или выше верхней)

Такой случай проверяется таким образом:

***(у1 < YН) и (у2 < YН)***

***или***

***(y1 > YВ)и (y2 > YВ)***

***или***

***(x1 < XЛ) и (x2 < XЛ)***

***или***

***(x1 > XП) и (x2 > XП)***

Если данное условие истинно, то отрезок целиком лежит вне окна и дальнейших расчетов можно не делать.

Так же Д.Коэн и А.Сазерленд предложили следующий формат хранения информации о расположении вершины относительно окна.

Для определения принадлежности точки одной из девяти областей, на которые разбивается плоскость продолжениями ребер отсекателя, вводится четырехразрядный (битовый) код. Биты этого кода заполняются по следующему правилу:

T1 =1, если точка лежит левее окна, и 0 в противном случае;

T2 =1, если точка лежит правее окна, и 0 в противном случае;

T3 =1, если точка лежит ниже окна, и 0 в противном случае;

T4 =1, если точка лежит выше окна, и 0 в противном случае.

Первым считается крайний правый бит.

У меня простой алгоритм, однако для определения положения точки я использовал такой формат хранения данных (подробнее опишу в коде).

Суть простого алгоритма заключается в том, что для оставшихся отрезков (не лежащих внутри области и не являющиеся полностью невидимыми) находятся точки (или точка) пересечения с окном и отсечение внешней части (подробнее в коде)

**Исходный код программы.**

# Маски для установления битов в 4битовом коде (бит устанавливается в единицу по правилам, описаным выше (когда вершина вне границ окна относительно одной из 4 сторон отсекателя)) (вызывается из функции cut\_section)

MASK\_LEFT = **0**b0001

MASK\_RIGHT = **0**b0010

MASK\_BOTTOM = **0**b0100

MASK\_TOP = **0**b1000

# Функция установления битов. Если нужно установить бит (выполнение условия из теор. Части, то происходит битовая дизьюнкция и бит становится единичным)

**def** **set\_bits**(point, rect\_sides):

# Изначально все биты сброшены

bits = **0**b0000

**if** point[**0**] < rect\_sides[cfg.LEFT]:

bits |= MASK\_LEFT

**if** point[**0**] > rect\_sides[cfg.RIGHT]:

bits |= MASK\_RIGHT

**if** point[**1**] < rect\_sides[cfg.BOTTOM]:

bits |= MASK\_BOTTOM

**if** point[**1**] > rect\_sides[cfg.TOP]:

bits |= MASK\_TOP

**return** bits

# Функция для нахождения границы отсечения отрезка в случае вертикального расположения (вызывается из функции cut\_section)

# Параметры:

# p – массив из 2 точек (вершины отрезка)

# index – 0 или 1 – индекс рассматриваемой вершины

# rect – массив из 4 элементов, содержащий границы окна

**def** **find\_vertical**(p, index, rect):

# если вершина выше верхней границы отсекателя, то верхняя часть (

# пересечения с верхней границей отсекателя) отсекается. Новый *у* = верхней # # границе

**if** p[index][**1**] > rect[cfg.TOP]:

**return** [p[index][**0**], rect[cfg.TOP]]

# иначе: аналогичный анализ, только относительно нижней границы

**elif** p[index][**1**] < rect[cfg.BOTTOM]:

**return** [p[index][**0**], rect[cfg.BOTTOM]]

# иначе: рассматриваемая вершина внутри границ, оставляем, как есть

# (стоит отметить, что в эту функцию не попадают отрезки, находящиеся левее или правее области окна)

**else**:

**return** p[index]

# Главная функция.

# Параметры:

# rect – Массив из 4 элементов, являющихся границами окна (по обращению будет ясно,

# какая граница рассматривается (пример: rect[cfg.LEFT] – левая граница))

# p – массив из 2 точек – вершины отрезка (как массив массивов: [[x1, y1], [x2, y2]]

**def** **cut\_section**(rect, p):

# Выставляем биты в коде (то есть находим положение точек относительно окна)

# s[0] – для 1ой точки, s[1] – для 2ой

s = list()

**for** i **in** range(**2**):

s.append(set\_bits(p[i], rect))

# Полностью видимый отрезок (оба кода = 0, получается, обе точки внутри границ)

# рисуем отрезок целиком и выходим из функции.

**if** s[**0**] == **0** **and** s[**1**] == **0**:

draw\_section(p[**0**][**0**], p[**0**][**1**], p[**1**][**0**], p[**1**][**1**], res\_color)

**return**

# Полностью невидимый отрезок (выходим из функции)

# В данном случае используется тот факт, что если у 2 вершин установлен один и

# тот же бит в коде, то они лежат, например, “левее левой границы” и никак не

# могут пересекать окно)

**if** s[**0**] & s[**1**]:

**return**

# cur\_index - содержит индекс текущей обрабатываемой вершины

cur\_index = **0**

# массив для записи вершин,получающихся после отсечения.

res = list()

# Проверка, не находится ли одна из точек внутри окна (первая проверка для

# точки с индексом 0, вторая - с индексом 1). Если вторая точка внутри области

# поставим ее на первое место и работаем с другой (смена мест нужна,

# чтобы в начале была обработанная точка, а за ней - нет)

**if** s[**0**] == **0**:

cur\_index = **1**

res.append(p[**0**])

**elif** s[**1**] == **0**:

res.append(p[**1**])

cur\_index = **1**

# Смена мест, описанная выше (необработанная точка на 2 месте), не забываем

# также поменять коды местами

p.reverse()

s.reverse()

# Основной цикл программы, cur\_index – индекс рассматриваемой точки (точек

# всего 2 с индексами 0 и 1, поэтому идем до 2)

**while** cur\_index < **2**:

# Если х совпадают, то прямая вертикальная (рассматриваем отдельно, так

# как для вертикальной прямой невозможно в общем случае найти наклон

# (знаменатель = 0))

**if** p[**0**][**0**] == p[**1**][**0**]:

res.append(find\_vertical(p, cur\_index, rect))

cur\_index += **1**

**continue**

# Нахождение наклона прямой

m = (p[**1**][**1**] - p[**0**][**1**]) / (p[**1**][**0**] - p[**0**][**0**])

# Нахождение пересечения с левой границей (если вершина находится левее

# левой границы)

# Здесь используется код вершины: если установлен бит, связанный с

# нахождением левее левой границы окна, то находим пересечение с левой

# границей

**if** s[cur\_index] & MASK\_LEFT:

y = round(m \* (rect[cfg.LEFT] - p[cur\_index][**0**]) + p[cur\_index][**1**])

# Проверяем: если произошло пересечение с границей окна – отсекаем

# и переходим к рассмотрению след. Вершины, если произошло пересечение

# продолжением – анализируем дальше.

**if** y <= rect[cfg.TOP] **and** y >= rect[cfg.BOTTOM]:

res.append([rect[cfg.LEFT], y])

cur\_index += **1**

**continue**

# Нахождение пересечения с правой границей (аналогично операции выше)

**elif** s[cur\_index] & MASK\_RIGHT:

y = round(m \* (rect[cfg.RIGHT] - p[cur\_index][**0**]) + p[cur\_index][**1**])

**if** y <= rect[cfg.TOP] **and** y >= rect[cfg.BOTTOM]:

res.append([rect[cfg.RIGHT], y])

cur\_index += **1**

**continue**

# Если прямая горизонтальна, пересечения с верхней и нижней границей быть # не может (заканчиваем обработку текущей вершины)

**if** m == **0**:

cur\_index += **1**

**continue**

# Нахождение пересечений с верхней и нижней границами

# С верхней (аналогично случаю с левой и правой границей, однако

# рассматривается формула с выраженным х (потому что у берется из величины

# верхней границы))

**if** s[cur\_index] & MASK\_TOP:

x = round((rect[cfg.TOP] - p[cur\_index][**1**]) / m + p[cur\_index][**0**])

# Проверка: пересечение с границей окна или же с продолжением

**if** x <= rect[cfg.RIGHT] **and** x >= rect[cfg.LEFT]:

res.append([x, rect[cfg.TOP]])

cur\_index += **1**

**continue**

# С нижней (аналогично операции выше, только для нижней границы окна)

**elif** s[cur\_index] & MASK\_BOTTOM:

x = round((rect[cfg.BOTTOM] - p[cur\_index][**1**]) / m + p[cur\_index][**0**])

**if** x <= rect[cfg.RIGHT] **and** x >= rect[cfg.LEFT]:

res.append([x, rect[cfg.BOTTOM]])

cur\_index += **1**

**continue**

cur\_index += **1**

# Если отсеченный отрезок найден (то есть изначальный отрезок не являлся

# полностью невидимым), то чертим его

**if** res:

draw\_section(res[**0**][**0**], res[**0**][**1**], res[**1**][**0**], res[**1**][**1**], res\_color)

Конец кода.

**Интерфейс и примеры работы.**

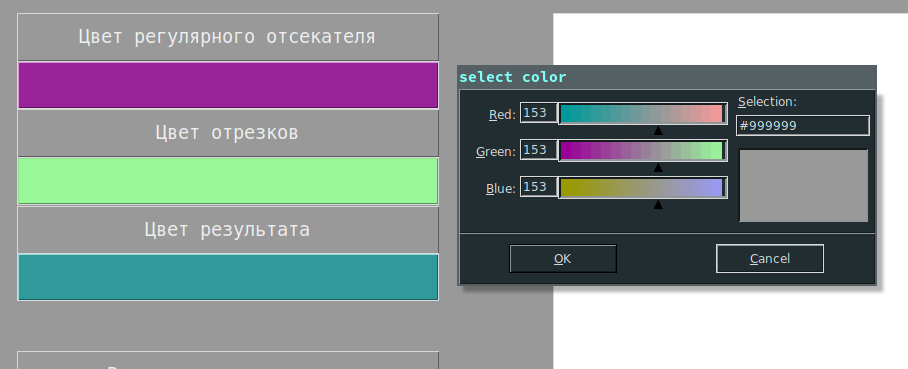
Интерфейс:



Интерфейс предусматривает 2 формата ввода: через поля (координаты) и через выбор мышью точки на плоскости.

При вводе с помощью мышки отсекатель задается 2 углами: левый нижний (для компьютера, для нас – верхний) и правый верхний (для нас – нижний). Положение углов выбирается нажатием правой кнопки мыши.

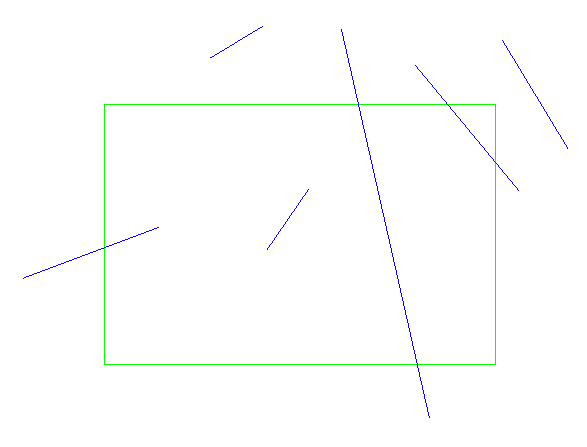
Отрезки вводятся с помощью левой кнопки мыши. Первое нажатие – начало отрезка, второе – конец. Можно ввести начало координатами, а конец выбрать мышкой, и наоборот. Цвета могут быть любые, используется палитра:



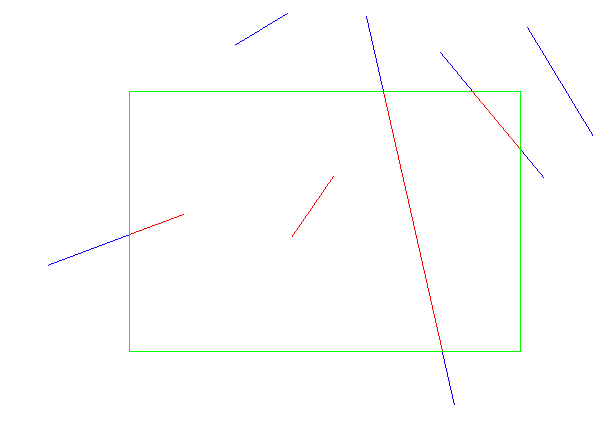
Примеры работы:

Общие случаи:

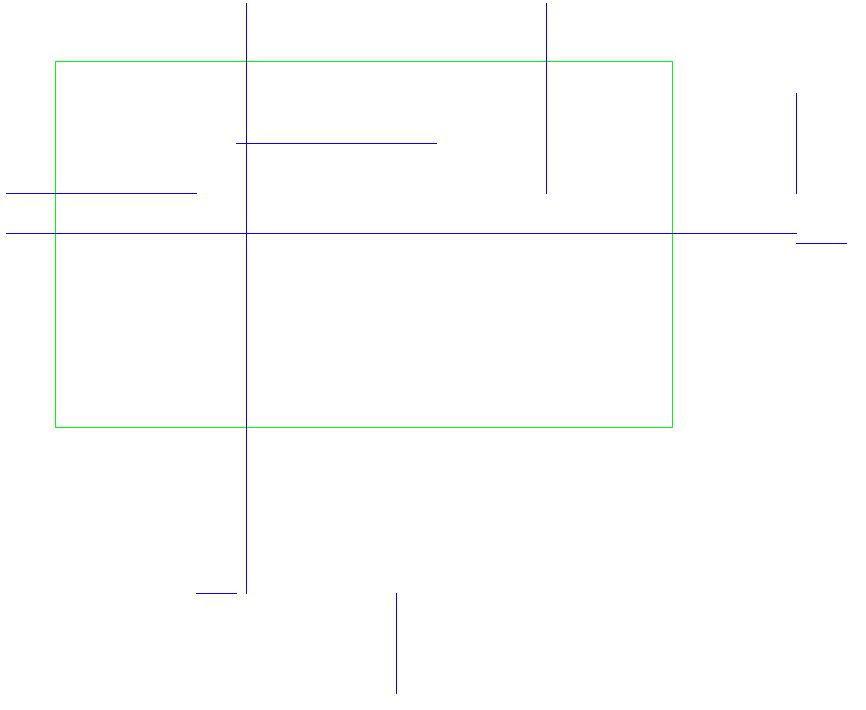
До:

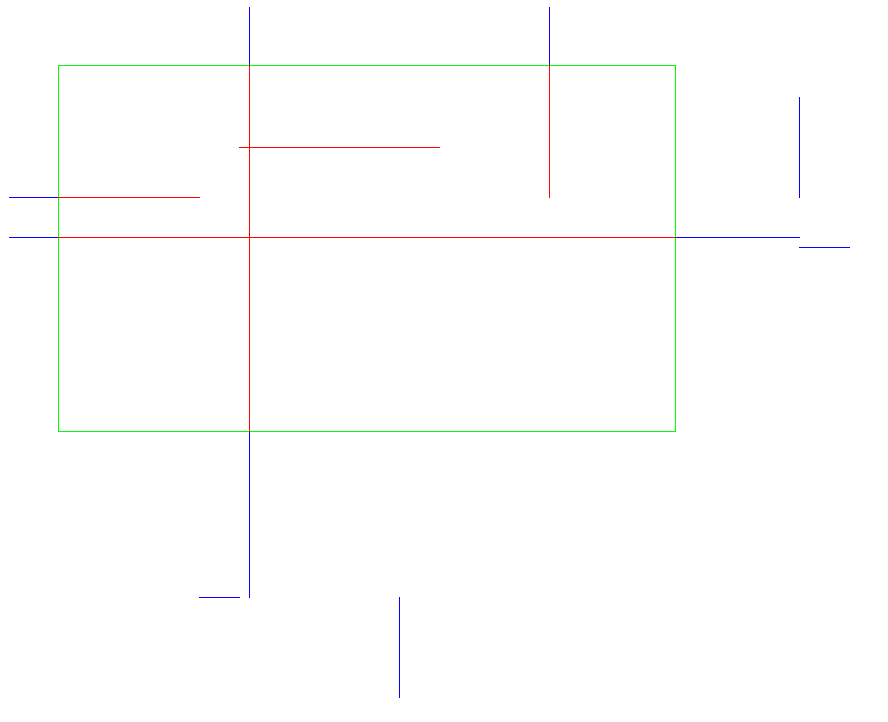


После:



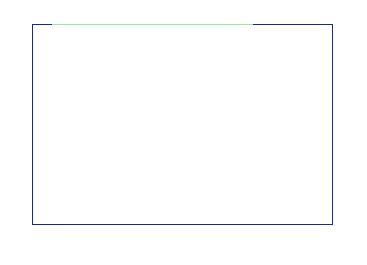
Горизонтальные и вертикальные прямые:

До:

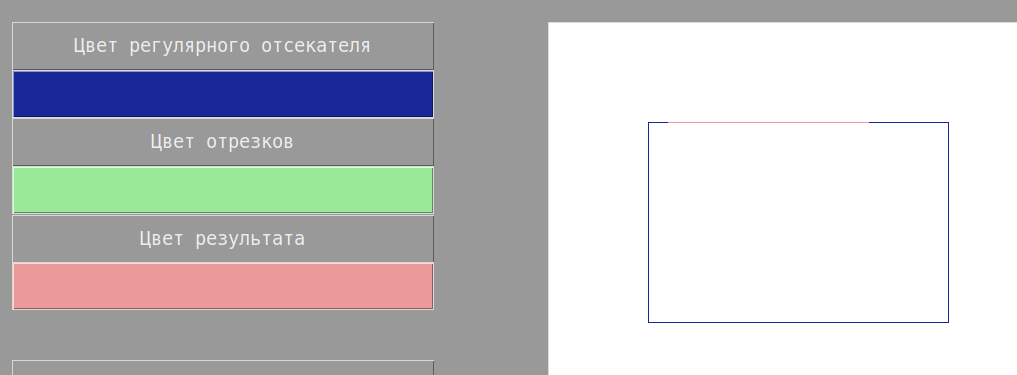
После:

Отрезок проходит вдоль границы отсекателя:

До:



После:



(телесный – цвет результата)