**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра вычислительной техники**

**Отчет**

**по лабораторной работе №4**

**по дисциплине «Компьютерная графика»**

**Тема: "** **Исследование алгоритмов отсечения отрезков и многоугольников окнами различного вида ”**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6307 |  | Лазарев С. О. |
| Преподаватель |  | Матвеева И. В. |

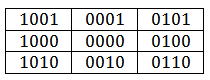
Санкт-Петербург

2019

**Задание:**

Обеспечить реализацию алгоритма отсечения массива произвольных от-резков заданным прямоугольным окном с использование алгоритма Ко-эна-Сазерленда. Вначале следует вывести на экран сгенерированные от-резки полностью, а затем другим цветом или яркостью те, которые полно-стью или частично попадают в область окна.

**Теоретическая часть:**

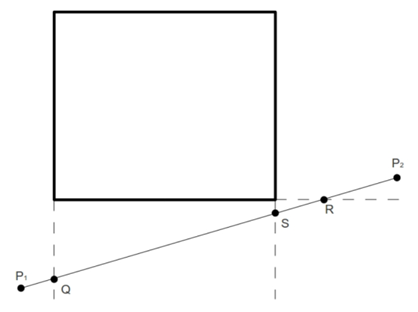
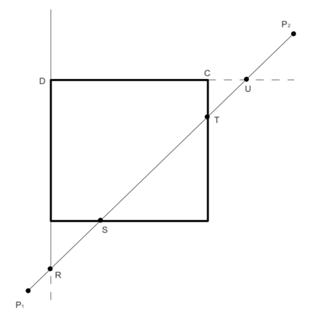
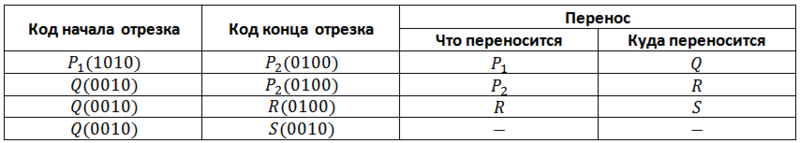
Суть алгоритма заключается в том, что концам отрезка присваивается четырёхбитный код: b0,b1,b2,b3. Этот четырёхбитный код содержит информацию о положении точки относительно области вывода. На практике возможны 9 комбинаций:  
  
Поясним эти коды:

b0=0, если x≥xmin;b0=1, если x<xmin;b1=0, если x≤xmax;b1=1, если x>xmax;b2=0, если y≥ymin;b2=1, если y<ymin;b3=0, если y≤ymax;b3=1, если y>ymax;  
После того, как коды получены, возможны следующие варианты:

Коды содержат только 0, а значит отрезок целиком лежит внутри окна и должен быть отрисован целиком;

Коды содержат единичный бит в одной и той же позиции, а значит, отрезок лежит за пределами окна и не будет отрисован;

Во всех остальных случаях в окне лежит только часть отрезка, и это значит, что есть необходимость в отсечении.

Первые два случая легко проверяются с помощью побитовых логических операций. Наибольший интерес представляет именно третий случай. Рассмотрим его на небольшом конкретном примере.  
  
  
  
Если код любого из концов содержит единичный бит, то либо P1, либо P2 перемещается из-за пределов окна к одной из его границ (или к её продолжению). Т.е. точка P1 перемещается в точку R, а P2 в точку U. Для новых точек мы вновь вычисляем четырёхбитные коды. В нашем случае концы отрезка по-прежнему лежат вне окна, т.е. нам понадобится ещё одно перемещение: точка R переместиться в точку S, а точка U переместиться в точку T. Получается, что процесс отсечения является итеративным. Так как на каждом шаге уменьшается расстояние между концами отрезка, мы можем утверждать, что алгоритм сойдётся. В итоге будет получен отрезок (в нашем случае (S;T), который и надо отобразить).  
Рассмотрим работу этого алгоритма на ещё одном примере.  
  
  
  
Расположение отрезка (P1;P2) не соответствует ни условиям полной видимости, ни условиям полной невидимости, поэтому в этом случае тоже прибегнуть к операции переноса точек.  
  
После выполненных переносов коды концов отрезка удовлетворяют второму условию, т.е. отрезок целиком не будет выводиться на экран.

**Иллюстрация работы приложения:**

Приложение написано на языке Dart, фреймворк Flutter. Тестирование проводилось на устройстве iPhone X.

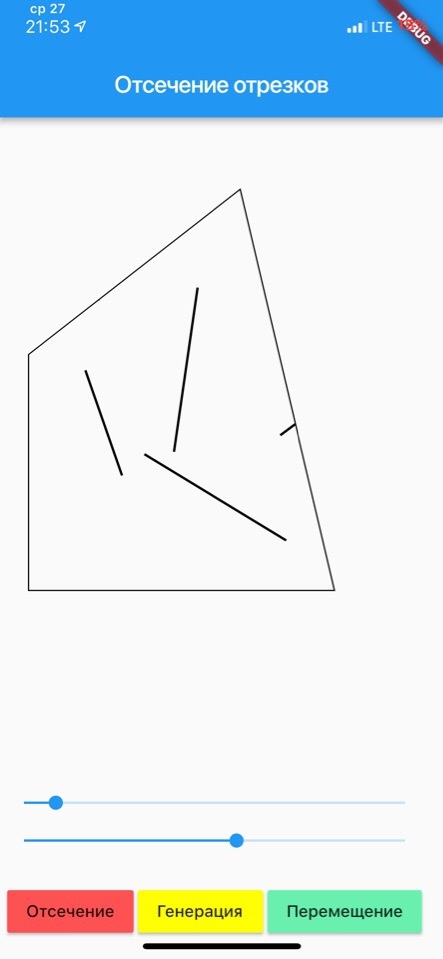


Рис. 1. Иллюстрация работы приложения

**Вывод:**

Я научился отсекать отрезки окном. Получил навык работы с изометрической проекцией.

**Код:**

class ShapesPainter extends CustomPainter {

@override

void paint(Canvas canvas, Size size) {

var path = new Path();

if (yes) {

path.moveTo(100 + x, 320 - y);

path.lineTo(100 + x, 380 - y + upperY);

path.lineTo(280 + x + upperX, 380 - y + upperY);

path.lineTo(280 + x, 320 - y - upperY);

path.lineTo(100 + x, 320 - y);

}

for (int i = 0; i < points.length; i++) {

path.moveTo(points[i][0].dx, points[i][0].dy);

path.lineTo(points[i][1].dx, points[i][1].dy);

}

var paint = new Paint()

..color = Colors.black

..style = PaintingStyle.stroke

..strokeWidth = 2.0;

canvas.drawPath(path, paint);

path.close();

}

@override

bool shouldRepaint(CustomPainter oldDelegate) => true;

}

class MyClipper extends CustomClipper<Path> {

@override

Path getClip(Size size) {

var path = new Path();

path.moveTo(100 + x, 320 - y);

path.lineTo(100 + x, 380 - y + upperY);

path.lineTo(280 + x + upperX, 380 - y + upperY);

path.lineTo(280 + x, 320 - y - upperY);

path.lineTo(100 + x, 320 - y);

return path;

}

@override

bool shouldReclip(CustomClipper<Path> oldClipper) => true;

}