

1. Задача синтеза сложного динамического изображения. Этапы синтеза изображения. Последовательность и основное содержание.

Синтез: вход, выход, объекты, решаемые задачи. Постановка задачи

Синтез: Вход-формальное описание, Выход-визуальное представление, Объекты-искусственно созданные изображения, Решаемые задачи-построение, генерация и преобразование изображения.

Задача синтеза — создание визуального представления объектов, имеющих формальное описание.

Этапы синтеза

1. Разработка трехмерной математической модели синтезируемой визуальной обстановки.
2. Определение направления линии визирования (прямая линия зрения, направленная от глаза наблюдателя на какую-нибудь отдаленную точку), положения картинной плоскости, размеров окна обзора, значений управляющих сигналов.
3. Формирование операторов, осуществляющих пространственное перемещение моделируемых динамических объектов визуализации.
4. Преобразование модели, синтезируемой в пространстве, к системе координат, связанным с наблюдателем.
5. Отсечение объектов визуального пространства по границам пирамиды видимости.
6. Вычисление двумерных перспективных проекций синтезируемых объектов видимости на картинную плоскость.
7. Исключение невидимых элементов синтезируемого пространства при заданном положении наблюдателя, закрашивание и затенение видимых элементов объектов визуализации.
8. Вывод полученного изображения на поверхность растрового дисплея.

Параметры поверхности

1. Уравнение поверхности (x, y, z поверхности).
2. Цвет.
3. Оптические свойства – коэффициенты: зеркального отражения K_{iz} , диф. отражения K_{id} , преломления (если прозрачная) K_{ipr}

Параметры системы

1. Система координат.
2. Положение картинной плоскости, размер окна обзора (x, y, z наблюдателя, последняя ось - направление взгляда).
3. Информация об источниках освещения: местоположение источника xi, yi, zi , цвет, интенсивность $I(I_r, I_g, I_b)$
4. Информация о среде: коэффициент поглощения Спогл
5. Частота изменения кадров $f_{min}=30\text{Гц}$ (вообще 24, 30 и 60), период $T=1/f$
 - Для моментов времени отстающих друг от друга на величину T , должна быть обеспечена возможность вычисления текущих координат и положения объекта.

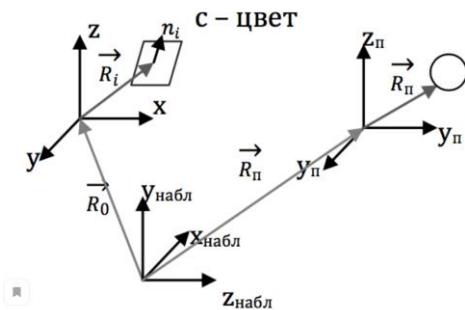
t1...t2...t3...>t

Анализ: Вход- визуальное представление, Выход- формальное описание, Объекты-изображения, созданные ЭВМ, выделенные из фото и слайдов, Решаемые задачи-распознавание образов, структур, анализ, поиск взаимосвязей.

Обработка изображения: Вход- визуальное представление, Выход-визуальное представление, Объекты-сканируемое изображение, Решаемые задачи- повышение качества изображения (повышение контрастности, подавление шума)

Постановка задачи:

неподвижная система подвижная система



Уровни алгоритмов

1. Алгоритмы **верхнего уровня**. Удаление невидимых линий и поверхностей, создание растрового изображения.
2. Алгоритмы **среднего уровня**. Построение плоских изображений.
3. Алгоритмы **низкого уровня** (базовые). Разложение простейших объектов в растр.

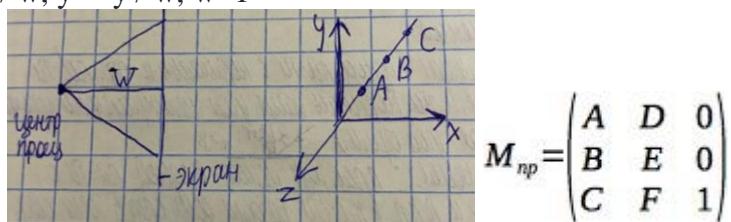
2. Преобразования на плоскости. Вывод расчетных соотношений. Матрицы преобразований.

Скалярная, матричная форма, в однородных координатах.

Линейное преобразование точки $A(x, y) \rightarrow B(x_1, y_1)$ может быть записано

- в скалярной форме формулами $x_1 = Ax + By + C$, $y_1 = Dx + Ey + F$
- В матричной форме $(x \ y) \begin{pmatrix} a & b \\ c & d \end{pmatrix} = (xa + yc, xb + yc)$
- В однородных координатах

Однородные координаты – представление точки в n-мерном пространстве задается при помощи n+1 координат. Для двумерного случая - (x, y, w). W - масштабный множитель (для плоского случая w = 1-плоскость экрана). Нормализация однородных координат: $x' = x / w$, $y' = y / w$, $w=1$



$$(x_1, y_1, 1) = (x, y, 1) * M_{np}$$

При этом такое преобразование некоммутативно $(x, y, 1) * M_{pr1} * M_{pr2} \neq (x, y, 1) * M_{pr2} * M_{pr1}$ (в общем случае, и если такое перемножение возможно вообще)

Аффинные преобразования

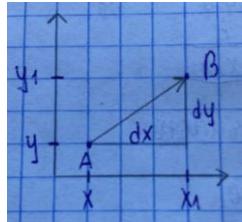
При которых плоскость не вырождается в прямую или в точку, сохраняется параллельность прямых и существует обратное преобразование.

- Операция преобразования, в общем случае, не коммутативна.
- Если определитель матрицы преобразований отличен от нуля, то такое преобразование будет аффинным (потому что тогда существует обратная матрица-обратное преобразование)
 - Представляются совокупностью преобразований: переноса, масштабирования, поворота.

$$A(x, y) \rightarrow B(x_1, y_1)$$

Перенос

- Это изменение местоположения изображения.
- 2 параметра – dx, dy ($|dx| \geq 0.5, |dy| \geq 0.5$, так как работаем с целыми числами)
- Вывод:



- В скалярной форме

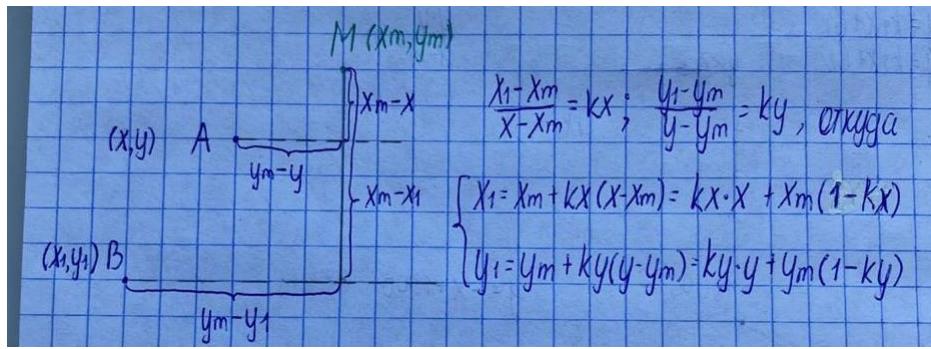
$$\begin{aligned} x_1 &= x + dx \\ y_1 &= y + dy \end{aligned}$$

• Матрица преобразований

$$M_{np} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{pmatrix}$$

Масштабирование

- Это изменение размеров и пропорций изображения
- 4 параметра – координаты центра масштабирования (x_m, y_m) и коэффициенты масштабирования (k_x, k_y – любые действительные значения)
- Вывод:



- В скалярной форме

$$x_1 = kx * x + (1 - kx) * x_m$$

$$y_1 = ky * y + (1 - ky) * y_m$$

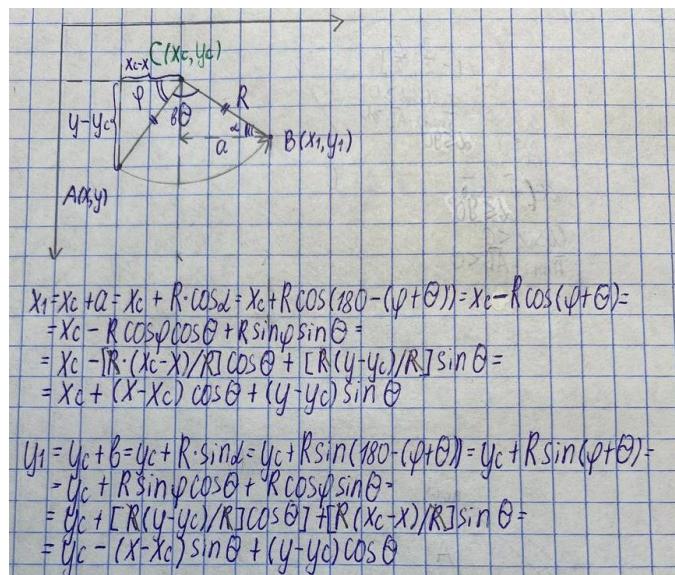
- Матрица преобразований

$$M_{np} = \begin{pmatrix} kx & 0 & 0 \\ 0 & ky & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- $Kx == ky$ – однородное масштабирование, иначе – неоднородное
- $(kx \text{ или } ky) > 1$ – изображение удаляется от центра и увеличивается.
- $0 < (kx \text{ или } ky) < 1$ – изображение приближается к центру и уменьшается.
- $kx \text{ или } ky < 0$ – отражение:
 - относительно ОY: $kx = -1, ky = 1$
 - относительно ОX: $kx = 1, ky = -1$
 - относительно начала координат: $kx = -1, ky = -1$

Поворот

- это операция, заключающаяся в изменении ориентации изображения
- 3 параметра – координаты центра поворота (x_c, y_c) и угол поворота θ
- Вывод:



- В скалярной форме:

$$x_1 = x_c + (x - x_c) * \cos\theta + (y - y_c) * \sin\theta$$

$$y_1 = y_c - (x - x_c) * \sin\theta + (y - y_c) * \cos\theta$$

- Матрица преобразований

$$M_{np} = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- За положительное выбирается направление против часовой стрелки (в нормальной системе)

Коммутативные

Коммутативность – это независимость результата преобразований от порядка, в котором они происходят.

Коммутативные операции: Перенос-перенос, Масштабирование – масштабирование, Поворот-поворот, Однородное масштабирование – поворот. Остальные комбинации - не коммутативны.

Аддитивные

Это значит, что матрица итогового преобразования может быть получена сложением

- Перенос (матрица переноса * матрица переноса)

$$M_{nep-nep} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx_1+dx_2 & dy_1+dy_2 & 1 \end{pmatrix}$$

- Поворот (матрица поворота * матрица поворота)

$$M_{nos-nos} = \begin{pmatrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Мультипликативные

Это значит, что матрица итогового преобразования может быть получена умножением

- Масштабирование (матрица масштабирования * матрица масштабирования)

$$M_{masch-masch} = \begin{pmatrix} kx_1*kx_2 & 0 & 0 \\ 0 & ky_1*ky_2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

3. Требования, предъявляемые к алгоритмам вычерчивания отрезков.

Пошаговый алгоритм разложения отрезка в растр. Разложение в растр по методу цифрового дифференциального анализатора.

Разложение отрезка в растр

Это процесс определения пикселей, наилучшим образом аппроксимирующих отрезок. Простейшие случаи: вертикальный – x фиксирован, горизонтальный – y фиксирован, под 45-на каждом шаге смещаем x, y

Требования к алгоритмам

1. Отрезок должен выглядеть как отрезок прямой. В общем случае не может быть выполнено, так как экран дискретен. Можно создать лишь иллюзию, например, методами устранения ступенчатости.
2. Отрезок должен заканчиваться и начинаться в заданных точках.
3. Яркость или интенсивность не должна зависеть от длины отрезка и угла его наклона, должна быть постоянной вдоль всего отрезка. В общем случае не может быть выполнено, так как экран дискретен: например, вертикальные и горизонтальные отрезки будут выглядеть более яркими, нежели отрезок в 45 градусов, в связи с меньшим расстоянием между пикселями.
4. Алгоритмы должны работать быстро. Для этого стараются избегать операций в вещественных числах, а также заменять умножение и деление сложением и вычитанием

Пошаговый алгоритм разложения отрезка в растр.

Базовые алгоритмы могут использовать

- численные методы - используется уравнение отрезка (или кривой)
- инкрементные (пошаговые) методы - на каждом шаге вычисляется направление элементарного движения от текущей точки к следующей.

Алгоритмы разложения геометрических объектов в растр имеют пошаговый характер:

1. Выбор текущего пикселя
2. Вычисления, результат которых используется на следующем шаге

При этом удается использовать целочисленную арифметику и избежать выполнения операций умножения и деления.

Метод цифрового дифференциального анализатора.

- Прямая линия на плоскости описывается уравнением $Ax + By + C=0$; $m=dy / dx = \text{const}$ – тангенс угла наклона
- Координаты следующей точки вычисляются системой:
 $x(i+1)=x(i)+ \Delta x$, $y(i+1)=y(i)+ \Delta y$
- Приращение, равное (по модулю) единице, выбирается для той координаты, приращение которой больше по модулю.

если $|m| \leq 1$, то берем $\Delta x=1$, а $\Delta y = (dy / dx) * \Delta x = (y_k - y_n) / (x_k - x_n)$
И тогда получаем систему: $x(i+1)=x(i)+ 1$; $y(i+1) = y(i) + (y_k - y_n) / (x_k - x_n)$
если $|m| > 1$, то наоборот

Алгоритм

1. Ввод (x_n , y_n), (x_k , y_k)
2. Анализ отрезка на вырожденность: если вырожден, то переход к пункту 7
3. Если $|x_k-x_n| \geq |y_k-y_n|$, то $l=|x_k-x_n|$, иначе $l=|y_k-y_n|$
4. $dx = (x_k-x_n) / l$, $dy = (y_k-y_n) / l$
5. $x = x_n$, $y = y_n$ (x, y вещественные)
6. Цикл построения отрезка (по i от 1 до $l+1$)
 - 6.1. Высветить точку (округление(x), округление(y))
 - 6.2. $x = x + dx$, $y = y + dy$
 - 6.3. Конец цикла
7. Конец.

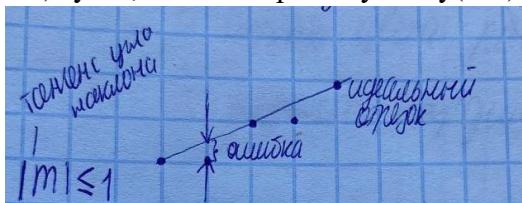
Оценка: Алгоритм медленный из-за округления.

4. Алгоритмы Брезенхема разложения отрезков в растр. Простой алгоритм Брезенхема. Целочисленный алгоритм Брезенхема. Общий алгоритм Брезенхема.

Понятие ошибки

Ошибка - расстояние между точкой раstra (пикселом) и точкой на идеальном отрезке при аппроксимации отрезка на текущем шаге. Лежит в основе алгоритма Брезенхема.

При $0 < m \leq 1$: $e_i = y_{id} - y_i$. Если $e_i \geq 0.5$, то выбирается верхняя ордината $y(i+1) = y_i + 1$, иначе – предыдущая $y(i+1) = y_i$



При этом на текущем шаге вычисляется ошибка для следующего шага: $e(i+1) = y_{id}(i+1) - y_i$
 $y_{id}(i+1) = y_i + m$

Если $y_{id}(i+1) = y_i$, то ранее вычисленное значение ошибки не изменится, но если выбрать верхний пиксель $y_{id}(i+1) = y_i + 1$, то изменится: $e(i+1) = y_{id}(i+1) - y_i = e(i) + m - 1 = e(i) - 1$

Поэтому делаем так: предварительно $e = e + m$, и на следующем шаге, если выбран верхний пиксель, то корректируем: $e = e - 1$

Чтобы ускорить вычисления, изначально ошибку уменьшают: $e_{нач} = e_{нач} - 0.5 = m - 0.5$ и тогда в цикле достаточно сравнивать ошибку с 0 (и для сравнения не требуется вычитание)

В зависимости от полученного значения ошибки выбирается пиксель с той же ординатой (при $e < 0$) или пиксель с ординатой, на единицу большей, чем у предыдущего пикселя (при $e > 0$).

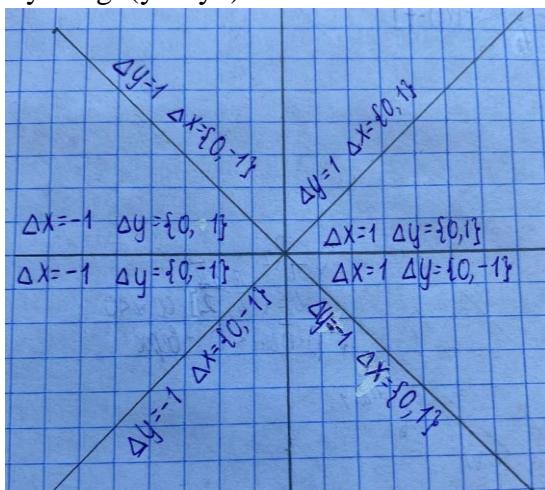
Приращения

Как и в предыдущем алгоритме, большее по модулю из приращений принимается равным шагу раstra, то есть единице, причем знак приращения совпадает со знаком разности конечной и начальной координат отрезка:

$$\text{Sign} = \{-1, 0, 1\}$$

$$\Delta x = \text{sign}(x_k - x_n)$$

$$\Delta y = \text{sign}(y_k - y_n)$$



Простой алгоритм Брезенхема

предназначен для построения отрезков, расположенных в первом октанте. В этом случае на каждом шаге координата X пикселя всегда получает единичное приращение, а координата Y изменяется либо на ноль, либо на единицу в зависимости от расстояния между действительным положением отрезка и ближайшей точкой раstra, аппроксимирующей на данном шаге отрезок.

1. Ввод исходных данных (X_h, Y_h), (X_k, Y_k)
2. Проверка на вырожденность и принадлежность 1 октанту
3. $X = X_h ; Y = Y_h$
4. $dx = X_k - X_h; dy = Y_k - Y_h$
5. $m = dy / dx; e = m - 0.5$
6. Основной цикл (по i от 1 до $dx + 1$)
 - 6.1. Высвечивание (x, y)
 - 6.2. Пока $e \geq 0$, то
 - 6.2.1. $y = y + 1$
 - 6.2.2. $e = e - 1$
 - 6.3. $e = e + m$
 - 6.4. $x = x + 1$
7. Конец

Общий алгоритм Брезенхема

Подходит для любого октанта, благодаря тому, что большее по модулю из приращений принимается равным шагу раstra, то есть единице, причем знак приращения совпадает со знаком разности конечной и начальной координат отрезка:

$\Delta x = \text{sign}(x_k - x_h)$ если $|X_k - X_h| \geq |Y_k - Y_h|$

$\Delta y = \text{sign}(y_k - y_h)$ иначе

Вторая координата смещается или не смещается в зависимости от ошибки на текущем шаге.

Введем обмен признак обмена местами координат x и y: если $|m| \leq 1$, то $\text{обмен}=0$ ($y(x)$), иначе $\text{обмен}=1$ ($x(y)$)

1. Ввод исходных данных (X_h, Y_h), (X_k, Y_k)
2. Проверка на вырожденность
3. $dx = X_k - X_h; dy = Y_k - Y_h$
4. $Sx = \text{sign}(dx); Sy = \text{sign}(dy)$
5. $dx = |dx|; dy = |dy|$
6. Если $dx > dy$, то обмен = 0, иначе обмен = 1; $t = dx; dx = dy; dy = t$
7. $m = dy / dx; e = m - 0.5$ [для целочисленного $e = 2dy - dx$]
8. $X = X_h ; Y = Y_h$
9. Цикл построения отрезка (по i от 1 до $dx + 1$)
 - 9.1. Высвечивание точки(x, y)
 - 9.2. Пока $e \geq 0$:
 - 9.2.1. если обмен = 0, то $y = y + sy$, иначе $x = x + sx$
 - 9.2.2. $e = e - 1$ [для целочисленного $e = e - 2dx$]
 - 9.3. Если обмен = 0, то $x = x + sx$, иначе $y = y + sy$
 - 9.4. $e = e + m$ [для целочисленного $e = e + 2dy$]
10. Конец

Здесь e и m - действительные

Целочисленный алгоритм Брезенхема

Приведенный алгоритм легко преобразуется к целочисленному варианту:

Для этого выражение при инициализации ошибки (п.7) запишем в виде: $e = dY/dX - 1/2$ и, умножив обе части этого равенства на $2dX$, получим: $2dX e = 2dY - dX$.

Обозначив $e' = 2dXe$ окончательно получим: $e' = 2dY - dX$

Тогда подсчет нового значения ошибки будет производиться по формулам: $e' = e' - 2dX$ в п. 9.2.2 и $E' = e' + 2dY$ в п. 9.4

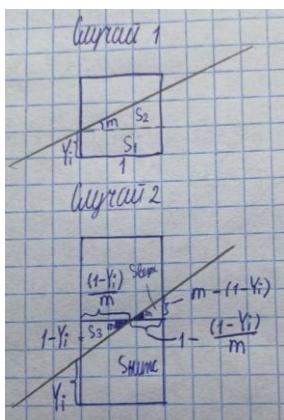
5. Основы методов устранения ступенчатости. Алгоритм Брезенхема с устранением ступенчатости. Алгоритм By.

Пиксель рассматривать не как математическую точку, а как некоторую конечную область. Чтобы сделать ступеньки практически незаметными, необходимо размыть края. Для этого нужно высвечивать отрезок с изменением интенсивности

Алгоритм Брезенхема

Интенсивность пропорциональна площади части пикселя, находящейся под отрезком.

Отрезок, тангенс угла наклона m которого лежит в диапазоне $0 < m < 1$ (все отрезки приводятся к такому виду), может при данном значении абсциссы пересечь один или два пикселя.



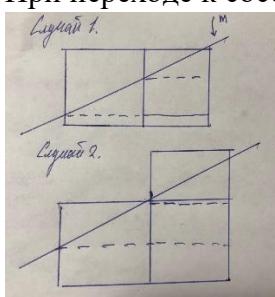
$$1) S = S_1 + S_2 = Y_i * 1 + 1 * m / 2 = Y_i + m / 2$$

$$2) S_{\text{ниж}} = 1 - S_3 = 1 - (1 - Y_i)^2 / (2m), S_{\text{верх}} = (m - 1 + Y_i)^2 / (2m), S_{\text{общ}} = Y_i + m / 2$$

Формулы одинаковые

Ошибка в данном алгоритме – та самая площадь $Y_i + m / 2$.

При переходе к соседнему пиксели:



- Если ордината не увеличивается, то площадь увеличивается на величину площади прямоугольника со сторонами 1 и m : $S(i+1) = S_i + m$, $e = e + m$.
- Если увеличивается на единицу, то вычисленная доля площади пикселя будет содержать и площадь нижнего пикселя (1), через который отрезок не проходит: $S(i+1) = S_i + m - 1 = S(i+1) - 1$, $e = e - 1$. Площадь не может быть отрицательной величиной

$(S=S-(1-m))$, то необходимо скорректировать ошибку, прибавив к ней величину $W=1-m$ – будет использоваться в качестве порогового, е сравниваем с ней

- Начальное значение ошибки $e=m-1/2 + (1-m)=1/2$. Первый пиксель отрезка будет высвечиваться интенсивностью, равной половине максимальной.
- Значение ошибки лежит в диапазоне $(0, 1)$

Если умножить тангенс угла наклона m , коэффициент W , ошибку e на I_{count} (количество уровней интенсивности), то будем работать не с дробной от максимальной, а с истинной интенсивностью

Алгоритм

1. Ввод исходных данных X_n, Y_n, X_k, Y_k, I - количество уровней интенсивности.
2. Если вырожден, то высвечивание отдельного пикселя и переход к п.13.
3. Вычисление приращений $dX=X_k-X_n$ и $dY=Y_k-Y_n$.
4. Вычисление шага изменения каждой координаты: $SX=\text{sign}(dX), SY=\text{sign}(dY)$.
5. Вычисление модулей приращения координат: $dX=|dX|, dY=|dY|$
6. Если $dX >= dY$, то $f1=0$, иначе ($f1=1, p=dX; dX=dY; dY=p$) ($f1$ - флаг, определяющий факт обмена местами координат)
7. $m=dY/dX$.
8. $f=I/2$
9. $X=X_n, Y=Y_n$.
10. Вычисление скорректированного значения тангенса угла наклона $m=mI$ и коэффициента $W=I-m$.
11. Цикл по i 1 до dX :
 - 11.1. Высвечивание пикселя (X, Y) интенсивностью $E(f)$.
 - 11.2. Если $f > W$, то [(если $f1=1$, то $X=X+SX$, иначе $(Y=Y+SY); f=f-1]$
 - 11.3. Если $f1=0$, то $X=X+SX$, иначе $(Y=Y+SY)$
 - 11.4. $f=f+m$
12. Конец.

Этот алгоритм лучше всего подходит для очерчивания многоугольников

Алгоритм ВУ (китайский алгоритм)

- Отрезок рисуется толщиной в 2 пикселя
- Суммарная интенсивность 2 высвечиваемых пикселов – постоянная $I=I1+I2=\text{const}$
- Интенсивность высвечивания пикселя зависит от его расстояния до точки на идеальном отрезке (чем ближе, тем больше)
 - Сглаживание получается за счёт перераспределения интенсивности I_{const} между 2 пикселями (на каждом шаге)

Горизонтальные и вертикальные линии не требуют дополнительного сглаживания.

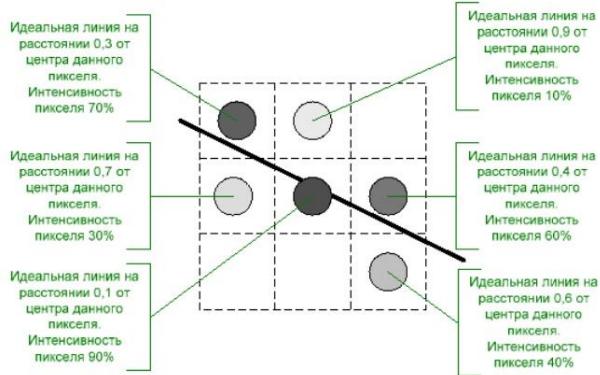
$$D1+d2=1$$

$d1 = y(i1) - [y(i1)]$ – дробная часть координаты-расстояние от нижнего пикселя $y(i1)$, до точки на идеальном отрезке.

$$d2 = 1 - d1$$

$$I(i1) = I_{const} * d2$$

$$I(i2) = I_{const} * d1$$



гусеница

6. Построение плоских кривых. Выбор шага изменения аргумента. Алгоритм построения эллипса и окружности по методу средней точки.

Что строим

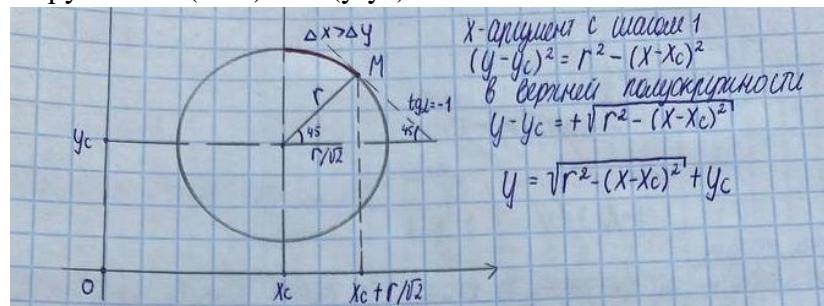
Достаточно построить эллипс только в 1 координатной четверти, а затем отзеркалить на остальные (окружность – даже 1/8)

Для эллипса построение ведется от верхней $(x_c, y_c + b)$ до правой точки $(x_c + a, y_c)$. Вычисляется точка M на эллипсе в первой координатной четверти, в которой тангенс угла наклона касательной к эллипсу $\operatorname{tg} \alpha = -1$. Пока не дошли до этой точки, $dx > dy$ и поэтому x-аргумент с шагом 1, y-вычисляется, затем-наоборот.

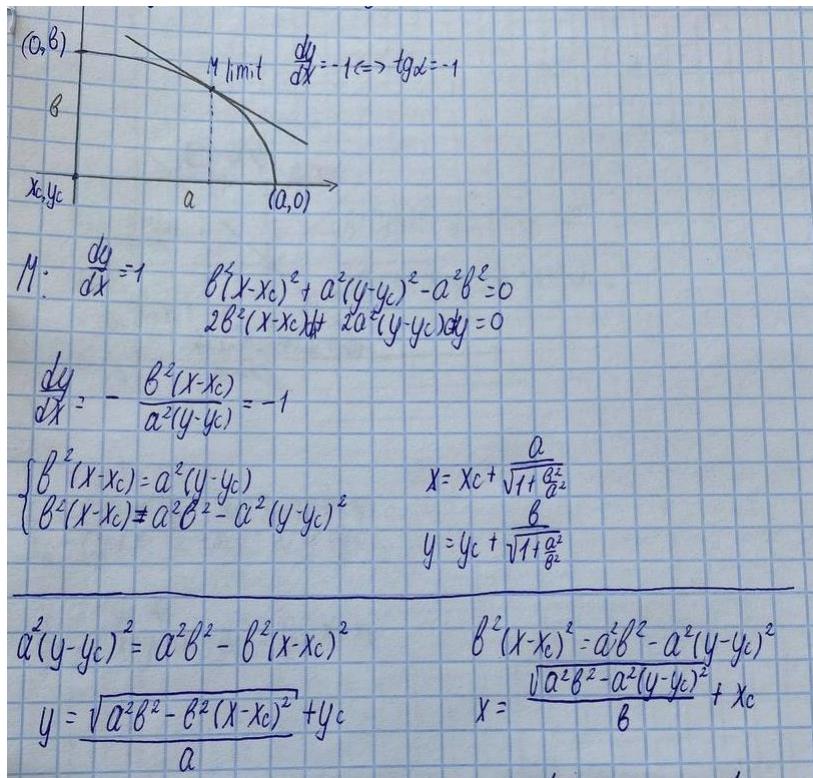
Для окружности построение ведется от верхней точки $(x_c, y_c + r)$ до точки M $(x_c + r/\sqrt{2}, y_c + r/\sqrt{2})$

Канонические уравнения

$$\text{Окружность } (x-x_c)^2 + (y-y_c)^2 = r^2$$



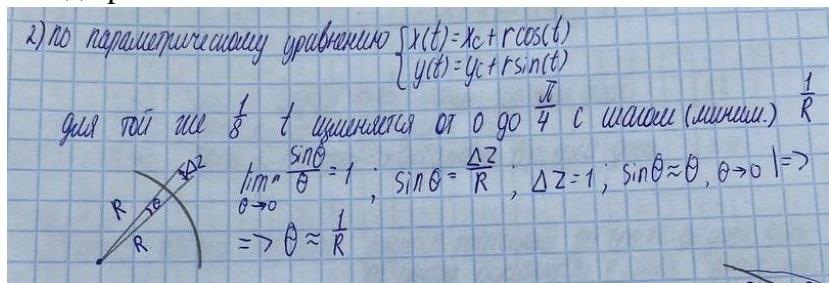
$$\text{Эллипс } (x-x_c)^2/a^2 + (y-y_c)^2/b^2 = 1$$



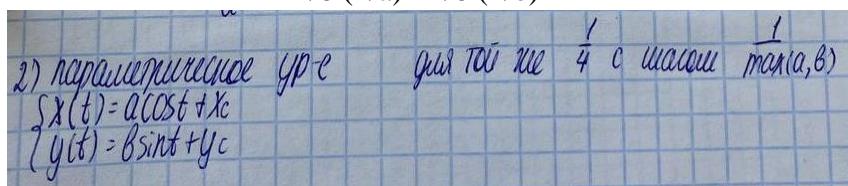
Параметрическое уравнение

Окружность

$\pi/4$ до $\pi/8$ с шагом $-1/R$



Эллипс. ТУТ ТОЖЕ $1/8$ (1/a) + $1/8$ (1/b)



Чем плохо так строить? Много вычислений

Как определять точку M

Переход происходит, когда $dy/dx = -1$. Продифференцировав $b^2 x^2 + a^2 y^2 - a^2 b^2 = 0$, получим:

$$\begin{aligned} d(b^2 X^2 + a^2 Y^2 - a^2 b^2)/dX &= 0 \\ 2b^2 X + 2a^2 Y(dY/dX) &= 0 \\ dY/dX &= -b^2 X/a^2 Y \end{aligned}$$

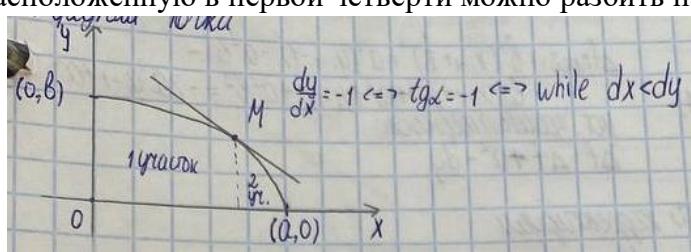
Следовательно, в точке, где $dY/dX = -1$, выполняется равенство $b^2 X = a^2 Y$ или $2b^2 X = 2a^2 Y$. Последняя запись более удобна при записи алгоритма, так как значения $2b^2 X$ и $2a^2 Y$ вычисляются по ходу выполнения алгоритма и их можно

использовать для определения точки, в которой выполняется равенство $dy/dx = -1$.

$$\begin{aligned}
 M: \frac{dy}{dx} = 1 & \quad b^2(x-x_c)^2 + a^2(y-y_c)^2 - a^2b^2 = 0 \\
 & 2b^2(x-x_c) + 2a^2(y-y_c)dy = 0 \\
 \frac{dy}{dx} = -\frac{b^2(x-x_c)}{a^2(y-y_c)} & = -1 \\
 \begin{cases} b^2(x-x_c) = a^2(y-y_c) \\ b^2(x-x_c) \neq a^2b^2 - a^2(y-y_c)^2 \end{cases} & \begin{aligned} x &= x_c + \frac{a}{\sqrt{1+\frac{b^2}{a^2}}} \\ y &= y_c + \frac{b}{\sqrt{1+\frac{a^2}{b^2}}} \end{aligned}
 \end{aligned}$$

Метод средней точки для построения эллипса

Условливаемся, что центр эллипса находится в точке $(0,0)$ (потом можно просто перенести). Строим $\frac{1}{4}$ часть эллипса, а затем отзеркаливаем. При этом часть эллипса, расположенную в первой четверти можно разбить на 2 участка:



Название средней точки связано с тем, что на каждом этапе выбора пикселя анализируется средняя точка между двумя пикселями, из которых мы выбираем.

- На первом участке выбор идет между двумя вертикальными пикселями.
- На втором участке выбор идет между двумя горизонтальными пикселями.

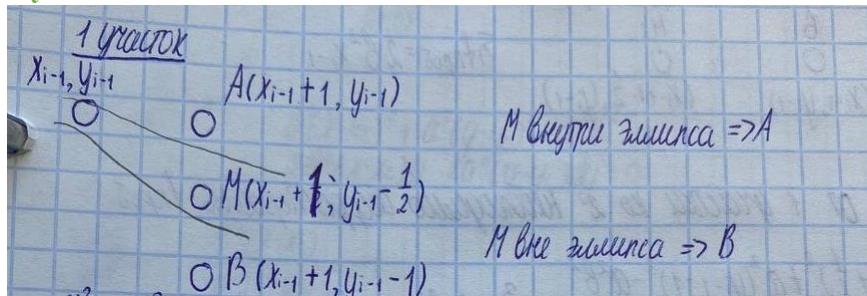
Определяется, находится средняя точка внутри или вне эллипса, и в зависимости от этого делается выбор ближайшего к идеальному эллипса пикселя (если на эллипсе, то можно выбрать любую)

Каноническое уравнение эллипса с центром в начале координат можно переписать в виде: $b^2 x^2 + a^2 y^2 - a^2 b^2 = 0$

Введем пробную функцию, которая равна 0, если точка принадлежит описываемой фигуре $f_{prob}(x, y) = b^2 x^2 + a^2 y^2 - a^2 b^2$, иначе

- $f(x, y) < 0$ - точка внутри эллипса.
- $f(x, y) > 0$ - точка вне эллипса.

1 участок



Значение пробной функции для средней точки:

$$f_{prob} = b^2(X_{i-1}+1)^2 + a^2(Y_{i-1}-0,5)^2 - a^2 b^2$$

Полученное значение требует затратных вычислений, поэтому можно свести его получение к рекуррентному процессу. Координаты предыдущей средней точки: $(X(i-1), Y(i-1) - 0,5)$ (если на предыдущем шаге выбран верхний пиксель A.)

$$df = f_{npi} - f_{npi-1} = [b^2(X_{i-1}+1)^2 + a^2(Y_{i-1}-0,5)^2 - a^2 b^2] - [b^2(X_{i-1})^2 + a^2(Y_{i-1}-0,5)^2 - a^2 b^2] = b^2(2X_{i-1}+1) = 2b^2 X_{i-1} + b^2 \quad (3.3.9)$$

Операцию умножения можно не выполнять, а ограничиться только выполнением операций сложения, если вести вычисления по следующей схеме:

$dx = dx + bd$; $df = df + b2 + dx$, где $b2 = b * b$, $bd = 2b2$. Значения $b2$, bd вычисляются один раз в начале работы алгоритма.

Если пиксель A оказался ближе к эллипсу ($f_{npi} < 0$), то вычисленное значение f_{npi} можно использовать на следующем шаге в качестве f_{npi-1} . Если же к эллипсу ближе расположен пиксель B, то полученное значение пробной функции надо скорректировать так, чтобы f_{npi} , которое будет рассматриваться как f_{npi-1} на следующем шаге, было вычислено относительно точки $(X_{i-1}, Y_{i-1} - 0,5)$. В этом случае получим:

$$df = [b^2(X_{i-1})^2 + a^2(Y_{i-1}-0,5)^2 - a^2 b^2] - [b^2(X_{i-1})^2 + a^2(Y_{i-1}+0,5)^2 - a^2 b^2] = -2a^2 Y_{i-1} \quad (3.3.10)$$

2 участок



- Если М расположена вне эллипса, то А расположена ближе к эллипсу.
- Если М расположена внутри эллипса, то В расположена ближе к эллипсу.

$$df = f_{npi} - f_{npi-1} = [b^2(X_{i-1}+0,5)^2 + a^2(Y_{i-1}-1)^2 - a^2 b^2] - [b^2(X_{i-1}+0,5)^2 + a^2(Y_{i-1})^2 - a^2 b^2] = -a^2(2Y_{i-1}-1) = -2a^2 Y_{i-1} + a^2 \quad (3.3.12)$$

Устанавливается закономерность $df = df + a^2/2 - dy$

Если пиксель A оказался ближе к эллипсу ($f_{npi} > 0$), то вычисленное значение f_{npi} можно использовать на следующем шаге в качестве f_{npi-1} . Если же к эллипсу ближе расположен пиксель B, то полученное значение пробной функции надо скорректировать так, чтобы f_{npi} , которое будет рассматриваться как f_{npi-1} на следующем шаге, было вычислено относительно точки $(X_{i-1}+0,5, Y_{i-1})$. В этом случае получим:

$$df = [b^2(X_{i-1}+0,5)^2 + a^2(Y_{i-1})^2 - a^2 b^2] - [b^2(X_{i-1}-0,5)^2 + a^2(Y_{i-1})^2 - a^2 b^2] = 2b^2 X_{i-1} \quad (3.3.13)$$

Переход

При переходе от 1 участка ко второму, необходимо скорректировать fпроб:

$$df_{np} = b^2(x_{i-1+\frac{1}{2}})^2 + a^2(y_{i-1-\frac{1}{2}})^2 - a^2b^2$$

$$- b^2(x_{i-1+\frac{1}{2}})^2 + a^2(y_{i-1-\frac{1}{2}})^2 + a^2b^2 = \frac{3}{4}(a^2 - b^2) - (b^2x_{i-1} + a^2y_{i-1})$$

Поскольку в алгоритме ранее вычисляются значения $2b^2 X_{i-1}$ и $2a^2 Y_{i-1}$, то полученное выражение лучше записать следующим образом:

$$df = 3(a^2 - b^2)/4 - (2b^2 X_{i-1} + 2a^2 Y_{i-1})/2$$

Начальные значения

В заключение остается вычислить только начальное значение f_{np} . Первая точка имеет координаты $(0, b)$, вторая точка – $(1, b)$ или $(1, b-1)$. Следовательно, средняя точка имеет координаты $(1, b-0,5)$. Поэтому

$$f_{np} = b^2 1^2 + a^2(b-0,5)^2 - a^2 b^2 = b^2 - a^2 b + 0,25 a^2$$

Чтобы складывать, а не умножать:

$$Dx=db^2x^2=2b^2x$$

$$Dy=da^2y^2=2a^2y$$

Алгоритм

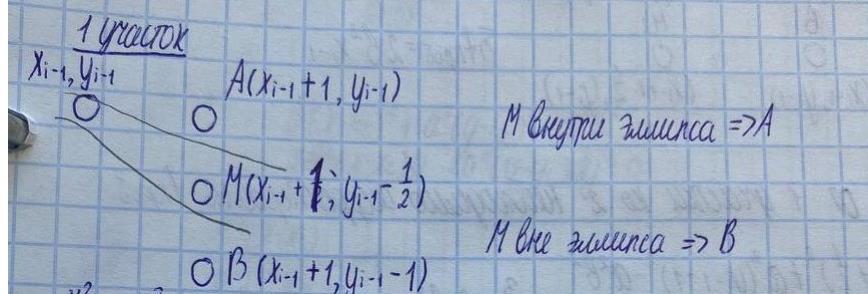
1. Ввод исходных данных: a , b , x_c , y_c
2. Вычисление полезных констант: $b2 = b * b$; $bd = 2 * b2$; $a2 = a * a$; $ad = 2 * a2$;
3. Инициализация начальными значениями: $x = 0$, $y = b$, $f = b2 - a2 * b + 0,25 * a2$;
 $dx = bd * x$; $dy = ad * y$;
4. Цикл для первого участка: пока $dx < dy$ делать
 - 4.1. Высвечиваем эту точку (x, y) , смещенную на x_c , y_c , а также 3 ее зеркальных отражения
 - 4.2. $x = x + 1$
 - 4.3. $dx = dx + bd$
 - 4.4. Если $f > 0$ (средняя точка вне эллипса – выбираем нижнюю и корректируем пробную функцию), то $(y=y-1; dy = dy-ad; f=f-dy)$
 - 4.5. Вычисляем новую пробную функцию $f=f+dx + b2$
5. Переходим от 1 ко 2 участку, корректируя пробную функцию $f = f + 3 / 4 * (a2 - b2) - (a2 * y + b2 * x)$
6. Цикл для второго участка: пока $y >= 0$ делать
 - 6.1. Высвечиваем эту точку (x, y) , смещенную на x_c , y_c , а также 3 ее зеркальных отражения
 - 6.2. $y = y-1$
 - 6.3. $dy=dy-ad$
 - 6.4. Если $f <= 0$ (средняя точка внутри эллипса – выбираем правую и корректируем пробную функцию), то $(x=x+1; dx = dx+bd; f=f+dx)$
 - 6.5. Вычисляем новую пробную функцию $f=f - dy + a2$
7. Конец

Метод средней точки для построения окружности

Это частный случай эллипса. Достаточно построить лишь первый участок (то есть 1/8 окружности)

Каноническое уравнение окружности с центром в начале координат можно переписать в виде: $x^2 + y^2 - r^2 = 0$

Тогда $f_{\text{проб}}(x, y) = x^2 + y^2 - r^2$ – разница квадратов расстояний до точки (x, y) и до точки идеальной окружности



Координаты предыдущей средней точки: $(X(i-1), Y(i-1) - 0,5)$ (если на предыдущем шаге выбран верхний пиксель А.)

$$df = f_{\text{проб}} - f_{\text{проб}, i-1} = [(x_{i-1}+1)^2 + (y_{i-1} - \frac{1}{2})^2 - r^2] - [(x_{i-1})^2 + (y_{i-1} + \frac{1}{2})^2 - r^2] = \\ = 2x_{i-1} + 1$$

корректируем
если же были выбраны нижний пиксель, то на ~~право~~

$$df = \dots = [(x_{i-1})^2 + (y_{i-1} + \frac{1}{2})^2 - r^2] - [(x_{i-1})^2 + (y_{i-1} + \frac{1}{2})^2 - r^2] = \\ = -2y_{i-1}$$

Начальные значения: первая точка имеет координаты $(0, r)$, вторая $(1, r)$ или $(1, r-0,5)$. $f_{\text{проб}}(x, y) = 1 + (r-0,5)^2 - r^2 = 5/4 - r$

Чтобы складывать, а не умножать:

$$Dx = db^2 x^2 - 2b^2 x$$

$$Dy = da^2 y^2 - 2a^2 y$$

1. Ввод исходных данных: r , x_c , y_c
2. Вычисление полезных констант: $r2 = r * r$; $rd = 2 * r2$;
3. Инициализация начальными значениями: $x = 0$, $y = r$, $f = 5/4 - r$;
4. Цикл для первого участка: пока $x <= y$ делать
 - 4.1. Высвечиваем эту точку (x, y) , смещенную на x_c , y_c , а также 7 ее зеркальных отражений
 - 4.2. $x = x + 1$
 - 4.3. Если $f > 0$ (средняя точка вне окружности – выбираем нижнюю и корректируем пробную функцию), то $(y=y-1; f=f-2*y)$
 - 4.4. Вычисляем новую пробную функцию $f=f+2*x + 1$
5. Конец

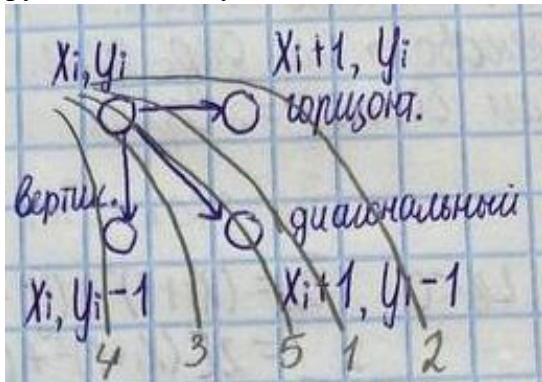
7. Основные расчетные соотношения и алгоритм Брезенхема для генерации окружности.

Что строим

Приведем расчеты для построения $\frac{1}{4}$ части окружности (в первой четверти), полагая, что ее центр в точке $(0, 0)$ (потом можно просто перенести) от точки $(0, R)$ до $(R, 0)$. Вообще можно строить для $\frac{1}{8}$ части. Тогда если строим от точки $(0, R)$ до $(R/\sqrt{2}, R/\sqrt{2})$, $R/\sqrt{2}$, то выбираем только из горизонтального или диагонального пикселов, если от $(R/\sqrt{2}, R/\sqrt{2})$, $R/\sqrt{2}$ до $(R, 0)$, то только из вертикального и диагонального пикселов.

Варианты

Возможны 3 варианта выбора следующего пикселя и 5 вариантов прохождения дуги окружности между ними:



Рассмотрим разницу квадратов расстояний от центра окружности до диагонального пикселя и до точки на идеальной окружности: $\Delta i = (x_i + 1)^2 + (y_i - 1)^2 - R^2$, которая

- <0 , если диагональный пиксель внутри окружности \rightarrow выбираем горизонтальный/диагональный (1,2)
- $=0$, если диагональный пиксель на окружности \rightarrow выбираем диагональный (5)
- >0 , если диагональный пиксель вне окружности \rightarrow выбираем вертикальный/диагональный (3, 4)

$\Delta i < 0$ - горизонтальный/диагональный

Критерий выбора горизонтальный/диагональный – разность L_g (разность квадратов расстояний от центра окружности до горизонтального пикселя и до точки на идеальной окружности) и L_d (разность квадратов расстояний от центра окружности до диагонального пикселя и до точки на идеальной окружности):

$$\delta 1 = L_g - L_d = |(x_i + 1)^2 + (y_i)^2 - R^2| - |(x_i + 1)^2 + (y_i - 1)^2 - R^2|, \text{ которая}$$

- <0 , если ошибка для диагонального больше \rightarrow выбираем горизонтальный
- $=0$, если ошибка одинакова \rightarrow выбираем любой (горизонтальный для определенности)
- >0 , если ошибка для горизонтального больше \rightarrow выбираем диагональный

Для сокращения вычислений, упростим $\delta 1$:

1. Случай $L_g > 0, L_d < 0$:

$$\begin{aligned} \delta 1 &= (x_i + 1)^2 + (y_i)^2 - R^2 + (x_i + 1)^2 + (y_i - 1)^2 - R^2 \\ &= 2 * [(x_i + 1)^2 + (y_i - 1)^2 - R^2] + 2 * y_i - 1 = 2 * (\Delta i + y_i) - 1 \end{aligned}$$

2. Случай $L_g < 0, L_d < 0$:

$$\begin{aligned} \delta 1 &= -(x_i + 1)^2 - (y_i)^2 + R^2 + (x_i + 1)^2 + (y_i - 1)^2 - R^2 = 1 - 2 * y_i < 0 \\ &\text{выбираем горизонтальный. Не диагональный в силу монотонности убывания} \end{aligned}$$

Δi > 0 - вертикальный/диагональный

Критерий выбора горизонтальный/диагональный – разность L_d (разность квадратов расстояний от центра окружности до диагонального пикселя и до точки на идеальной окружности) и L_v (разность квадратов расстояний от центра окружности до вертикального пикселя и до точки на идеальной окружности):

$$\delta_2 = L_d - L_v = |(x_i + 1)^2 + (y_i - 1)^2 - R^2| - |(x_i)^2 + (y_i - 1)^2 - R^2|, \text{ которая}$$

- <0 , если ошибка для вертикального больше \rightarrow выбираем диагональный
- $=0$, если ошибка одинакова \rightarrow выбираем любой (диагональный для определенности)
- >0 , если ошибка для диагонального больше \rightarrow выбираем вертикальный

Для сокращения вычислений, упростим δ_2 :

3. Случай $L_d > 0, L_v < 0$:

$$\begin{aligned} \delta_1 &= (x_i + 1)^2 + (y_i - 1)^2 - R^2 + (x_i)^2 + (y_i - 1)^2 - R^2 \\ &= 2 * [(x_i + 1)^2 + (y_i - 1)^2 - R^2] - 2 * x_i - 1 = 2 * (\Delta_i - x_i) - 1 \end{aligned}$$

4. Случай $L_d > 0, L_v > 0$:

$$\delta_1 = (x_i + 1)^2 + (y_i - 1)^2 - R^2 - (x_i)^2 - (y_i - 1)^2 + R^2 = 1 + 2 * x_i > 0 -$$

выбираем вертикальный

Формулы

Формулы для вычисления координат очередного пикселя и следующего критерия для каждого из 3 вариантов

1. При переходе к горизонтальному:

$$x_{i+1} = x_i + 1, y_{i+1} = y_i$$

$$\Delta_{i+1} = (x_{i+1} + 1)^2 + (y_{i+1} - 1)^2 - R^2 = (x_i + 2)^2 + (y_i - 1)^2 - R^2 = (x_i + 1)^2 + (y_i - 1)^2 - R^2 + 2x_i + 3 = \Delta_i + 2x_{i+1} + 1$$

2. При переходе к диагональному:

$$x_{i+1} = x_i + 1, y_{i+1} = y_i - 1$$

$$\Delta_{i+1} = (x_{i+1} + 1)^2 + (y_{i+1} - 1)^2 - R^2 = (x_i + 2)^2 + (y_i - 2)^2 - R^2 = (x_i + 1)^2 + (y_i - 1)^2 - R^2 + 2x_i + 3 - 2y_i + 3 = \Delta_i + 2(x_{i+1} - y_{i+1} + 1)$$

3. При переходе к вертикальному:

$$x_{i+1} = x_i, y_{i+1} = y_i - 1$$

$$\Delta_{i+1} = (x_{i+1} + 1)^2 + (y_{i+1} - 1)^2 - R^2 = (x_i + 1)^2 + (y_i - 2)^2 - R^2 = (x_i + 1)^2 + (y_i - 1)^2 - R^2 - 2y_i + 3 = \Delta_i - 2y_{i+1} + 1$$

Начальные значения

$$x = 0, y = R, \Delta = (0 + 1)^2 + (R - 1)^2 - R^2 = 2(1 - R)$$

Алгоритм

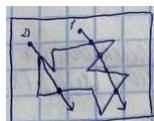
1. Ввод исходных данных R (радиус окружности) и при необходимости X_c, Y_c (координаты центра окружности)
2. Задание начальных значений текущих координат пикселя $X=0, Y=R$, параметра $D=2(1-R)$, установка конечного значения ординаты пикселя $Y_k=0$

3. Высвечивание текущего пикселя (X, Y)
4. Проверка окончания работы: если $Y < Y_k$, то переход к п.11
5. Анализ значения параметра D : если $D < 0$, то переход к п.6; если $D = 0$, то переход к п.9, иначе переход к п.7
6. Вычисление параметра $D_1 = 2D + 2Y - 1$ и анализ полученного значения: если $D_1 < 0$, то переход к п.8; Если $D_1 > 0$, то переход к п.9
7. Вычисление параметра $D_2 = 2D - 2X - 1$ и анализ полученного значения; если $D_2 < 0$, то переход к п.9, если $D_2 > 0$, то переход к п.10
8. Вычисление новых значений X и D (горизонтальный шаг): $X = X + 1$; $D = D + 2X + 1$. Переход к п.3
9. Вычисление новых значений X, Y, D (диагональный шаг): $X = X + 1$; $Y = Y - 1$; $D = D + 2(X - Y + 1)$. Переход к п.3
10. Вычисление новых значений Y и D (вертикальный шаг): $Y = Y - 1$; $D = D - 2Y + 1$. Переход к п.3
11. Конец

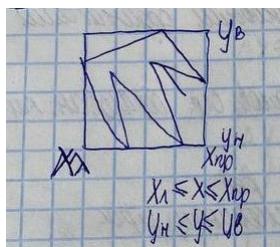
8. Растворная развертка сплошных областей. Алгоритм с упорядоченным списком ребер.

Общее

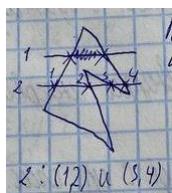
1. Это генерация сплошных областей на основе описаний рёбер или вершин.
2. Алгоритмы заполнения справляются с произвольными многоугольниками (выпуклые и нет, с отверстиями).
3. Простой способ определить, где точка – луч из этой точки: четное количество пересечений – вне, нечетное – внутри (проходит через вершину – частный случай). Если идти горизонтальным лучом $y = \text{const}$ по всем пикселям – миллионы операций.



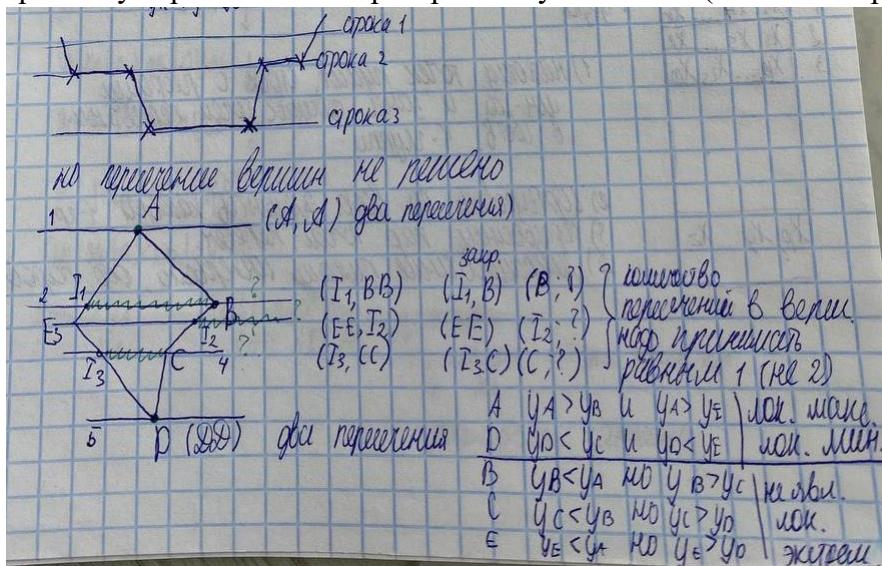
4. Две группы алгоритмом заполнения – растровые и затравочные
В методах растровой развертки пытаются определить в порядке сканирования строк, лежит ли точка внутри многоугольника. Сканирующая строка – совокупность пикселей, имеющих одну и ту же определенную координату y .
5. Многоугольник расширяют



6. Принадлежность точек строки меняется только в точках пересечения строки с ребрами



7. В каждом алгоритме нужно рассматривать решение двух проблем: прохождение сканирующих строк точно через вершины многоугольников и проблему горизонтальных рёбер многоугольников (их как бы пропускаем).



8. УПОРЯДОЧИВАЮТСЯ РЁБРА, А НЕ ТОЧКИ ПЕРЕСЕЧЕНИЯ!!!

9. Процесс реализации можно разбить на две части: 1. Сортировка данных. 2. Преобразование отсортированных данных в растровую форму.

Простой алгоритм с упорядоченным списком ребер

1. Найти все точки пересечения всех сканирующих строк с ребрами многоугольника, сохранить в списке
2. Упорядочить массив пересечений по убыванию значений координаты y .
3. Упорядочить массив пересечений с одинаковыми y по возрастанию x .
4. Разбить найденные точки пересечений с одинаковыми ординатами на пары.
На этом этапе нужно обработать экстремумы. В случае прохождение очередной сканирующей строки через вершину многоугольника в массиве будут найдены точки с одинаковым значением абсциссы. Если рассматриваемая точка - экстремум, оставляем обе точки, если нет - оставляем в одном экземпляре.
5. Закрасить пиксели, расположенные внутри полученных интервалов

Алгоритм не эффективный из-за кучи сортировок. Их будет столько же, сколько сканирующих строк.

Первые 3 пункта – подготовка, оставшиеся – перевод в растровую форму

Более эффективный вариант.

Можно первую сортировку выполнять одновременно с нахождением точек пересечения.

Память, отведенную для хранения точек пересечения, разбиваем на столько участков, сколько сканирующих строк, пересекающих ребра многоугольника. Эти участки памяти можно назвать Y-группами.

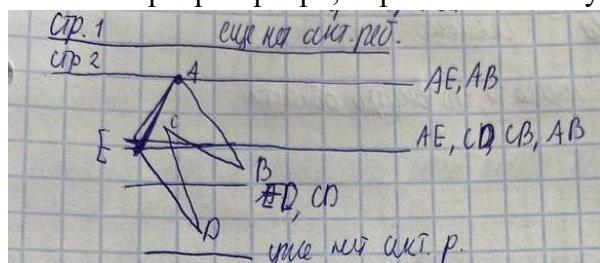


1. нахождение точек пересечения строк с ребрами и занесение абсцисс пересечения в соответствующую у-группу
2. сортировка по возрастанию элементов каждой у-группы
3. образование пар
4. закраска

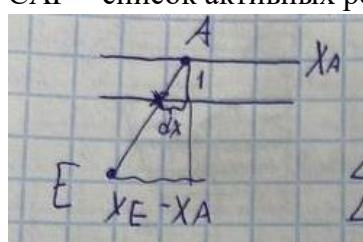
плюсы – процесс развертки начинается до окончания сортировки, легче добавлять и изменять информацию, минус - Проблемы с памятью

РЕШЕНИЕ ВСЕХ ПРОБЛЕМ: динамический список.

Активное ребро - ребро, пересекаемое текущей сканирующей строкой.

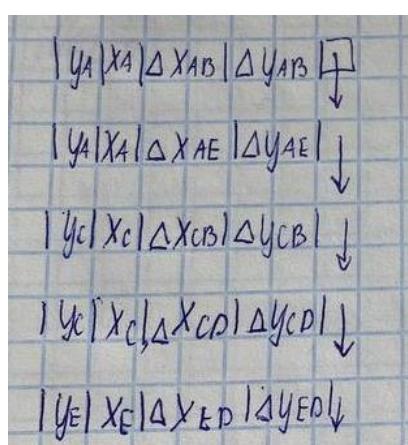


CAP – список активных ребер



$Dy = |y_A - y_E|$ - количество сканирующих строк, которые пересекают рассматриваемое ребро (чтобы определить в каком случае нужно исключать ребро: на каждом шаге $dy = dy - 1$, как только $dy < 0$ - исключаем ребро (оно обработано))

$Dx = (xE - xA) / |y_A - y_E|$ - котангенс угла наклона. На каждом шаге $x = x + dx$



При формировании списка мы должны сначала упорядочить ребра (по убыванию наибольшего значения координаты у ребра).

Если ребро активно, переходим к рассмотрению следующего элемента списка, пока не встретим ребро, которое не является активным на данной сканирующей строке.

Для активных ребер определяем абсциссы точки пресечения с очередной сканирующей строкой, она хранится во 2ом поле элемента списка.

Если ребро перестает быть активным, то удаляем элемент списка.

Опустошение списка => весь многоугольник закрашен.

1. Для каждого ребра: в $y = \max(y_n, y_k)$, $x = \text{соответствующий}(x_n, x_k)$, $dy = |y_n - y_k|$, $dx = (\text{соотв}(x_n - x_k)) / dy$, $\text{next} = \text{NULL}$
2. Сортируем рёбра в порядке убывания поля y
3. В CAP первое ребро, строка = $\text{edges}[0].n$, позиция = 1
4. пока строка > 0:
 - 4.1 если CAP пуст, то завершаем
 - 4.2 для всех рёбер после текущей позиции: если поле строки == y , (вносим ребро в CAP, позиция = $j + 1$), иначе – к 4.3
 - 4.3. Из CAP берем все текущие координаты x , сортируем, пары, высвечиваем
 - 4.4. строка -= 1, корректируем все рёбра ($dy -= 1$, $x -= dx$)
 - 4.5. удаляем из CAP рёбра, для которых поле dy обнулилось

Оценка алгоритма

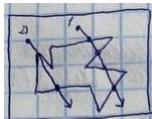
+ -меньше памяти, точки пересечения вычисляются в пошаговом режиме. - По быстродействию – непонятно, вроде плохо. плюсы первого это минусы второго

1. Получение информации о цвете пикселя – ни разу, не интересует
 2. Изменение цвета пикселя – для каждого по разу
 3. Количество обрабатываемых пикселей – только те, что внутри области
- Принадлежность пикселей внутренней области многоугольника определяем путем нахождения точек пересечения сканирующих строк с ребрами многоугольника.

9. Заполнение многоугольников. Алгоритмы заполнения по ребрам, с перегородкой, со списком ребер и флагом.

Общее

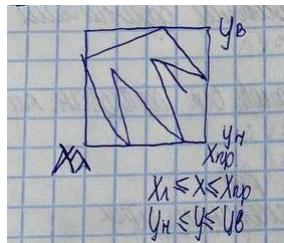
1. Растровая развертка (сплошных областей) – генерация сплошных областей на основе простых описаний описаний рёбер или вершин.
2. Алгоритмы заполнения справляются с произвольными многоугольниками (выпуклые и нет, с отверстиями).
3. Простой способ определить, где точка – луч из этой точки: четное количество пересечений – вне, нечетное – внутри (проходит через вершину – частный случай). Если идти горизонтальным лучом $y=\text{const}$ по всем пикселям – миллионы операций.



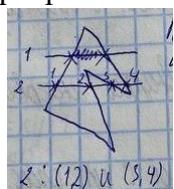
4. Две группы алгоритмом заполнения – растровые и затравочные

В методах растровой развёртки пытаются определить в порядке сканирования строк, лежит ли точка внутри многоугольника или контура. Эти алгоритмы обычно идут сверху вниз по заданному многоугольнику (то есть от y_{min} до y_{max}). Сканирующая строка - совокупность пикселей, имеющих одну и ту же определенную координату y .

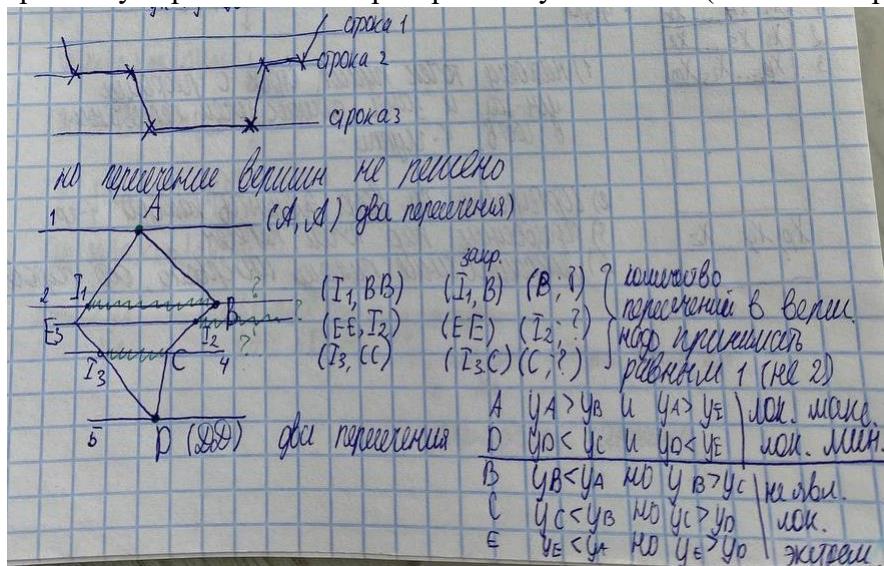
5. Многоугольник расширяют



6. Принадлежность точек строки меняется только в точках пересечения строки с ребрами



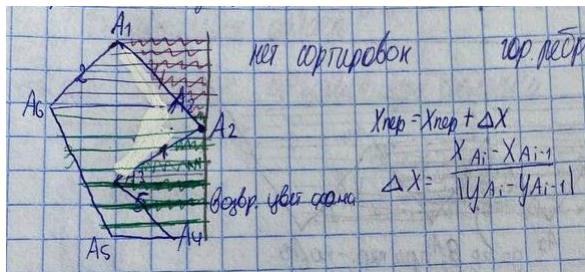
7. В каждом алгоритме нужно рассматривать решение двух проблем: прохождение сканирующих строк точно через вершины многоугольников и проблему горизонтальных рёбер многоугольников (их как бы пропускаем).



Алгоритм по ребрам

- цвет фона = НЕ(цвет заполнения) и наоборот
- Для каждой сканирующей строки, пересекающей ребро многоугольника, дополнить все пиксели, расположенные правее точки пересечения. Будем заполнять сканирующей строкой до вертикальной прямой, проходящей через самую правую вершину многоугольника.

Плюс алгоритма – не нужно создавать и сортировать списки различных данных. Можно ребра в произвольном порядке обрабатывать.



1. находим абсциссу крайней правой вершины
2. В цикле по ребрам
 - 2.1. Горизонтальные ребра игнорируются if $y_1 == y_2$: continue
 - 2.2. if $y_1 > y_2$: $y_1, y_2 = y_2, y_1; x_1, x_2 = x_2, x_1$
 - 2.3. $cur_y = y_1$, $end_y = y_2$ (последняя игнорируется), $dx = (x_2 - x_1) / (y_2 - y_1)$, $start_x = x_1 + 1$
 - 2.4. в цикле по сканирующим строкам while $cur_y < end_y$:
 - 2.4.1. $x = \text{round}(start_x)$
 - 2.4.2. while $x < xm$: Инвертировать (x, cur_y), $x += 1$
 - 2.4.3. $cur_y += 1$, $start_x += dx$

Оценка эффективности

1. Получение информации о цвете пикселя – для каждого правее и не раз (столько, сколько ребро расположено левее)
 2. Изменение цвета пикселя – для каждого правее и не раз
 3. Количество обрабатываемых пикселей – и внутри, и вне
- Неэффективный, но простой

Алгоритм заполнения с перегородкой

Дополнить пиксели сканирующей строки, расположенные правее точки пересечения с ребром многоугольника, но левее перегородки, если пересечение расположено левее перегородки. (и наоборот)

Перегородка – воображаемая вертикальная прямая, которую мы проводим с целью уменьшения количества раз изменения цвета каждого пикселя. Перегородку надо стремиться проводить таким образом, чтобы количество пикселей слева и справа было примерно одинаковым. Получить перегородку внутри многоугольника можно, усреднив координаты всех вершин или усреднив максимальное и минимальное.

+сокращение числа обрабатываемых пикселей для пикселей, расположенных правее перегородки. – все еще неоднократная обработка и даже тех, что вне

Алгоритм заполнения со списком ребер и флагом

До этого – геометрическое определение точек пересечения (признаки – координаты, длина, ...), теперь – графический (цвет, интенсивность, ...).

Флаг – признак расположения пикселя внутри (истина) или вне (ложь) многоугольника.

Алгоритм можно разделить на 2 шага выполнения:

1. Очерчивание границ закрашиваемой области. (должен быть задан ее цвет)
2. Заполнение области

Анализируя цвет границы, мы будем определять точки пересечения сканирующей строки с ребрами. Используем оболочку для многоугольника

Псевдокод

1. Очерчивание границ многоугольника.
2. Цикл по всем строкам, пересекающим ребра многоугольника (по y от y_{\min} до y_{\max})
 - 2.1. Флаг = ложь

2.2. Цикл по всем пикселям строки (по x от x_{min} до x_{max})

2.2.1. Если цвет $(x, y) = \text{цвет границы}$, то инвертировать значение флага

2.2.2. Если флаг истина, то цвет $(x, y) = \text{цвет закраски}$, иначе цвет $(x, y) = \text{цвет фона}$

2.3. Конец цикла по x.

3. Конец цикла по y.

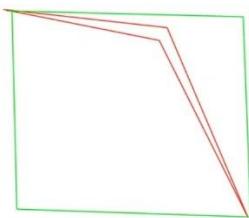
Оценка эффективности

1. Получение информации о цвете пикселя – только 1

2. Изменение цвета пикселя – только 1

3. Количество обрабатываемых пикселей – и внутри, и вне

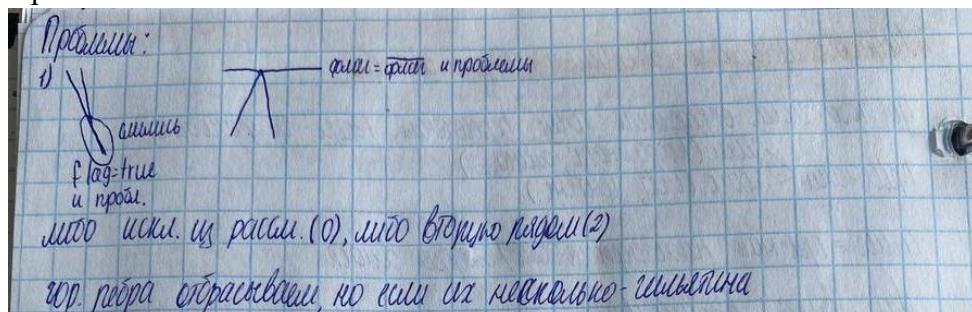
Но это будет не всегда эффективно. Пример:



Алгоритм считается таким же быстрым, как САР. Считается что при аппаратной реализации будем самым быстрым.

А как найти точку пересечения (по Курову)? использовать рекуретное соотношение $x = x + dx$, где dx - котангенс угла

Проблемы



10. Алгоритм заполнения с затравкой, простой алгоритм заполнения с затравкой.

Общее

В алгоритмах затравочного заполнения сплошных областей предполагается, что известна определенная точка (затравка) внутри области и необходимо определить точки, соседние с затравочной и расположенные внутри области. Если соседняя точка расположена не внутри, значит обнаружена граница области, если же точка находится внутри, то она становится новой затравочной точкой и поиск продолжается рекурсивно.

Известны два алгоритма заполнения с затравкой: простой алгоритм заполнения с затравкой; построчный алгоритм заполнения с затравкой. Данные алгоритмы применимы к **гранично-определенным областям**, то есть таким, что все пиксели на границе данных областей имеют выделенное значение или цвет, но ни один пиксель из внутренней части таких областей не может иметь это выделенное значение. (Пиксели, расположенные на границе **внутренне-определенной** области, могут иметь разные цвета, за исключением цвета самой области.)

Границно-определенные области могут быть 4- или 8-связными. Любой пиксель в 4-связной области становится доступен с помощью движений только в четырех направлениях: вверх, вниз, направо, налево; для 8-связной же области к любому пикселю можно подойти, используя комбинацию движений в 2-х горизонтальных, в 2-х вертикальных и 4-х диагональных направлениях. *Алгоритм заполнения 8-связной области заполнит и 4-связную область (обратное неверно).*

Отметим также то, что граница четырехсвязной области является восьмисвязной, а граница восьмисвязной области – четырехсвязной.

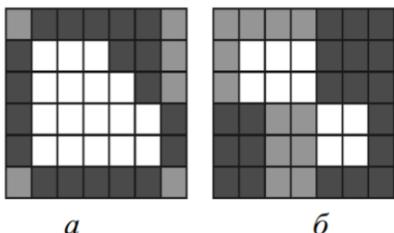


Рис.8.2. Четырехсвязная границно-определенная (а) и восьмисвязная внутренне-определенная (б) области

Область – любая (даже кривая, с отверстиями)

Простой алгоритм заполнения с затравкой (базовый алгоритм разработан Смитом) легко реализовать, используя понятие стека с порядком обслуживания "первым пришел, последним обслужен", то есть, когда новое значение помещается в стек, то все остальные опускаются вниз на один уровень, а когда значение извлекается из стека, все остальные поднимаются на один уровень. Такой стек еще называется стеком прямого действия.

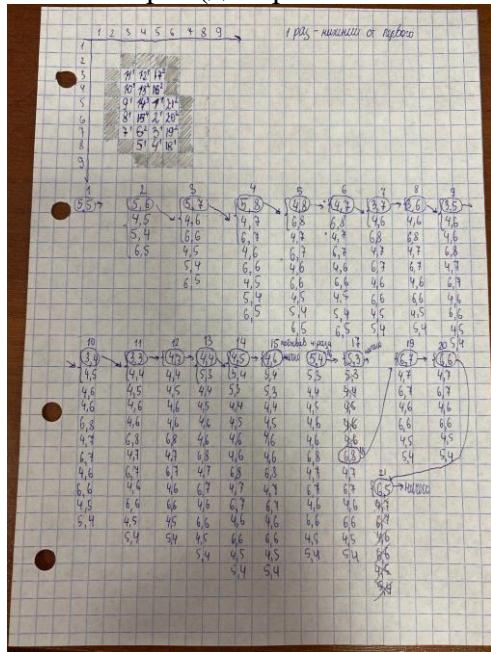
Проанализировать - сравнить цвет с цветом границы и проверить, не закрашен ли рассматриваемый пиксель. Если пиксель не является граничным, и не закрашен, то нужно рассматривать его как новый затравочный пиксель.

Алгоритм для 4-связной

1. Ввод исходных данных (информация о границах области, цвет границы, цвет заполнения, координаты затравки). Если граница не очерчена-очертить
2. Занесение в стек затравочного пикселя.
3. Пока стек не пуст, то:
 - 3.1. Затравочный пиксель (x, y) извлекается из стека
 - 3.2. Закрасить пиксель (если время закраски > времени получения информации, то можно сначала проанализировать его цвет, и если он не закрашен, закрасить) (получается, если пиксель уже закрашен, то он побывал в стеке, и его соседи уже проанализированы и занесены в стек, их вновь заносить не надо)
 - 3.3. Проводится анализ: надо ли помещать соседние пиксели в стек. Для этого, каждый из соседних к данному 4-связанных пикселя [(x+1, y), (x,y+1), (x-1, y), (x, y-1)] проверяется на два условия: не является ли он граничным или не присвоено ли уже ему требуемое значение вот так:
 - 3.3.1. если (пиксель != граничное зн-е и пиксель != требуемое зн-е), тогда пиксель помещается в стек;

В приведенном алгоритме 4-связанные пиксели проверяются, начиная с правого от текущего, в направлении против часовой стрелки.

- все неграничные, не закрашенные пиксели области помещаются в стек – большой расход памяти
- пиксель рассматривается несколько раз и заносится в стек несколько раз (до 4 раз включительно), поэтому хранятся там тоже лишняя память



11. Алгоритмы заполнения с затравкой. Построчный алгоритм заполнения с затравкой.

Общее

В алгоритмах затравочного заполнения сплошных областей предполагается, что известна определенная точка (затравка) внутри области и необходимо определить точки, соседние с затравочной и расположенные внутри области. Если соседняя точка расположена не внутри, значит обнаружена граница области, если же точка находится внутри, то она становится новой затравочной точкой и поиск продолжается рекурсивно.

Известны два алгоритма заполнения с затравкой: простой алгоритм заполнения с затравкой; построчный алгоритм заполнения с затравкой. Данные алгоритмы применимы к **гранично-определенным областям**, то есть таким, что все пиксели на границе данных областей имеют выделенное значение или цвет, но ни один пиксель из внутренней части таких областей не может иметь это выделенное значение. (Пиксели, расположенные на границе **внутренне-определенной** области, могут иметь разные цвета, за исключением цвета самой области.) Границно-определенные области могут быть 4- или 8-связными. Любой пиксель в 4-связной области становится доступен с помощью движений только в четырех направлениях: вверх, вниз, направо, налево; для 8-связной же области к любому пикселю можно подойти, используя комбинацию движений в 2-х горизонтальных, в 2-х вертикальных и 4-х диагональных направлениях. **Алгоритм заполнения 8-связной области заполнит и 4-связную область (обратное неверно).**

Отметим также то, что граница четырехсвязной области является восьмисвязной, а граница восьмисвязной области – четырехсвязной.

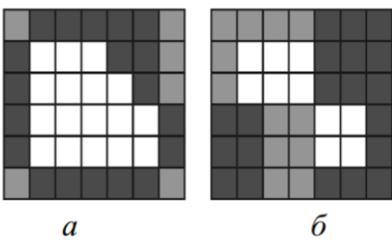


Рис.8.2. Четырехсвязная гранично-определенная (а) и восьмисвязная внутренне-определенная (б) области

Область – любая (даже кривая, с отверстиями) но для алгоритма **во внешней, примыкающей к данной гранично-определенной области, не должно быть пикселей с цветом заполнения.**

Построчный алгоритм

Улучшить простой алгоритм можно, если в стек заносить только один пиксель для **непрерывного интервала пикселей** (группа примыкающих друг к другу не закрашенных пикселей, ограниченная граничными или закрашенными пикселями).

В заданном интервале в качестве затравочного выбирается **самый правый** пиксель из возможных.

Каждый пиксель просматривается до 3 раз (по 2 – в верхней и нижней)

В алгоритме **2 шага**

1. заполнение пиксели текущей строки (влево и вправо от затравочного).

Расширение границ происходит на 1 этапе

2. поиск новых затравочных пикселей (на двух соседних строках по отношению к текущей) в интервале $x_l \leq x \leq x_{pr}$

Острый момент – область толщиной в 1 пикс.

Алгоритм

1. Ввод (границы, цвет границы, цвет заполнения, координаты затравки).

Очерчивание границы, если еще не

2. Занесение в стек затравочного пикселя.

3. Пока стек не пуст, то:

- 3.1. Извлечь затравку (x, y)

3.2. Заполнить интервал с затравочным пикселем вправо и влево от затравочной точки, пока не будет найдена граница слева и справа, запоминаем границы:

3.2.1. Вправо: $x_{cur}=x$; пока (цвет (x_{cur}, y) != цвет границы) делать (цвет (x_{cur}, y) = цвет закраски; $x_{cur}+=1$);

3.2.2. запоминаем крайний правый пиксель: $x_r=x_{cur}-1$.

3.2.3. Влево: $x_{cur}=x-1$; пока (цвет (x_{cur}, y) != цвет границы) делать (цвет (x_{cur}, y) = цвет закраски; $x_{cur}-=1$);

3.2.4. запоминаем крайний левый пиксель: $x_l=x_{cur}+1$.

3.3. Поиск новых затравочных пикселей в интервале $x_l \leq x \leq x_r$ на двух соседних строках $y+1, y-1$ (поиск первого и в случае его нахождения – самого правого) и занесение в стек. Если интервал прерывается граничными или закрашенными пикселями, то необходимо найти новый интервал не закрашенных пикселей и найти в нем самый правый затравочный пиксель.

3.3.1. Страна «ниже» текущей в экранной системе

$$x = x_l, y = y+1$$

3.3.2. В цикле пока ($x \leq x_r$) ищем затравочные пиксели:

3.3.2.1. Flag=0 (флаг переходения затравки)

3.3.2.2. Пока (цвет (x, y) != цвет границы) и (цвет (x, y) != цвет заполнения) и (x <= xr), то (если flag=0, то flag=1) и x=x+1 (<= решает проблему области толщиной в 1 пиксель)

3.3.2.3. помещаем в стек крайний правый затравочный пиксель: Если (flag=1) (это произойдет (для первого захода), если рассматриваемая строка в начале интервала нуждается в заполнении, но еще не заполнена), то

3.3.2.3.1. Если (цвет (x, y) != цвет границы) и (цвет (x, y) != цвет заполнения) и (x == xr), то занесение в стек пикселя (x, y) (это произойдет, если весь интервал $x_l \leq x < x_r$ на данной строке не нуждается в заполнении, но еще не заполнен)

3.3.2.3.2. Иначе – занесение в стек (x-1, y) (встретили границу или уже заполненную часть или чуть перешагнули)

3.3.2.4. Поиск нового интервала в случае прерывания текущего интервала (произойдет, если $x < x_r$) (Далее проверка не является ли строка ниже текущей границей многоугольника, или уже полностью заполненной продолжается в том случае, если интервал был прерван)

3.3.2.4.1. $x_n = x$ (запоминаем абсциссу текущего пикселя), flag=0

3.3.2.4.2. Пока ((цвет (x, y) == цвет границы) или (цвет (x, y) == цвет заполнения)) и ($x < x_r$), то $x = x + 1$

3.3.2.4.3. убедимся, что координата абсциссы пикселя увеличилась (чтобы не зациклился): Если $x == x_n$, то $x = x + 1$, повторение всех действий для нового полуинтервала – продолжение цикла 3.3.2

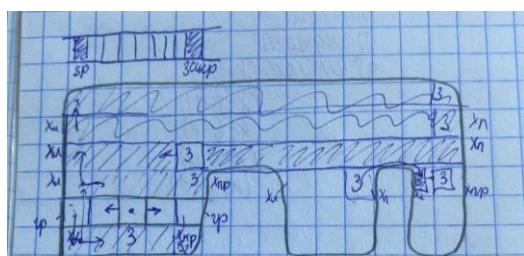
Аналогично пункту 3.3.1 осуществляем проверку строки ниже текущей, изменения лишь касаются координаты y, ей присваивается значение: $y = y - 2$.

Эффективность

- Цвет пикселя меняется всего лишь один раз.
- Обрабатываются пиксели внутри области и граничные пиксели. В среднем, пиксель будет обработан 3 раза: при проходе по его строке, и при поиске затравочных пикселей

Алгоритм лучше, чем по ребрам, по ребрам с перегородкой (так как там нужно несколько раз анализировать и менять каждый пиксель + снаружи)

Позволяет закрашивать и более сложные фигуры, нежели многоугольники.



12. Двумерное отсечение. Простой алгоритм отсечения отрезка.

Общее

1. Отсечение - это операция удаления изображения за пределами выделенной области, называемой отсекателем. **Стирание** – внутри.
2. Виды **отсечений** – **Двумерное Трехмерное**. Отсечение в пространстве либо сводится к отсечению на плоскости, либо сами плоские алгоритмы без труда распространяются на трехмерный случай.
3. Чтобы выполнить данную операцию, необходимо задать тип отсекателя.

Виды отсекателей (двумерных):

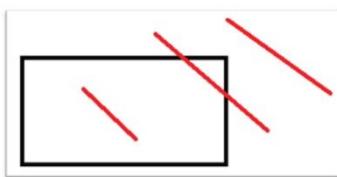
- Регулярные(стандартные) - прямоугольник со сторонами, параллельными осям координат

- Нерегулярные (произвольные) – многоугольник (может иметь отверстия)
 - Выпуклый
 - Невыпуклый

4. Должны быть также известны геометрические характеристики изображенных объектов. **В результате отсечения** должны получиться геометрические характеристики объектов, остающихся в пределах окна отсечения.

Отсечение отрезков регулярным отсекателем

1. Для выполнения отсечения **необходимо задать** абсциссы X_l , X_p левого и правого ребер и ординаты Y_h , Y_b нижнего и верхнего ребер.
2. Точки, лежащие **целиком внутри окна**, удовлетворяют условию $(X_l \leq X \leq X_p) \wedge (Y_b \leq Y \leq Y_h)$, где (X, Y) - координаты точки.
3. Отрезок является **полностью видимым** (лев), если целиком лежит внутри отсекателя, если обе его концевые точки лежат внутри окна. Иначе - **Полностью невидимым** (прав) или **частично видимым** (центр). 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.



Простой алгоритм определения видимости отрезка

1. 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

$T_1 = 1$, если точка лежит левее окна ($x < x_l$), и 0 в противном случае;
 $T_2 = 1$, если точка лежит правее окна ($x > x_p$), и 0 в противном случае;
 $T_3 = 1$, если точка лежит ниже окна ($y < y_b$), и 0 в противном случае;
 $T_4 = 1$, если точка лежит выше окна ($y > y_h$), и 0 в противном случае.

Первым считается крайний правый бит.

в координатах экрана в условиях для T_3 и T_4 знаки строгого неравенства поменяются между собой местами

1001	1000	1010
0001	0000	0010
0101	0100	0110

2. Определяется **поразрядная сумма кода** точки: $S = \text{сумма}(\text{по } i \text{ от 1 до 4})T_i$

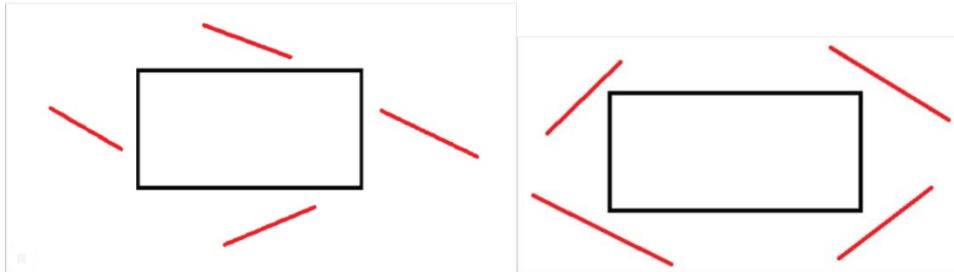
$S = 0$ - точка видима

Тогда **отрезок будет полностью видимым**, если поразрядные суммы кодов концов этого отрезка равны нулю:

$(S1=0 \text{ и } S2 = 0)$, $S1 = \text{сумма}(\text{по } I \text{ от 1 до 4})T_{1i}$, $S2 = \text{сумма}(\text{по } I \text{ от 1 до 4})T_{2i}$

3. Если отрезок не является полностью видимым, то он может являться либо частично видимым, либо полностью невидимым.

Ниже приведены два вида полностью невидимых отрезков:



Для вида полностью невидимых отрезков, предоставленного на втором рисунке, с помощью простого анализа кодов концов отрезков определить полную невидимость не получается.

Введенные коды концов отрезка можно использовать для определения полностью невидимых отрезков на 1 рисунке: если обе вершины расположены по невидимую сторону одной из границ отсекателя, то в одинаковых разрядах у них 1.

Вводится **побитное логическое произведение** кодов концов отрезка

$P = \text{сумма}(\text{по } I \text{ от 1 до 4})T_{1i}*T_{2i}$

Отрезок является полностью невидимым и его можно отбросить, если побитное логическое произведение кодов концов отрезка не равно нулю:

$P \neq 0$ – полностью невидим

4. При этом мы также имеем возможность определить частичную видимость следующим простым тестом:

$(S1 = 0 \& S2 \neq 0) \mid (S1 \neq 0 \& S2 = 0)$ – заведомо частично видим

Для таких отрезков от нас потребуется найти вторую вершину видимой части.

5. Алгоритм:

1. Вычисление кодов первой и второй вершин отрезка T1, T2.
2. Вычисление сумм кодов вершин $S_1 = \sum_{i=1}^4 T1_i$, $S_2 = \sum_{i=1}^4 T2_i$
3. Определение полной видимости отрезка: если $(S_1=0) \& (S_2=0)$, то pr = 1 и переход к п. 6.
4. Вычисление попарного логического произведения кодов концов отрезка: $p = \sum_{i=1}^4 T1_i T2_i$.
5. Определение полной невидимости отрезка: если $p \neq 0$, то pr = -1, иначе pr = 0.
6. Конец.

Пересечение отрезка с ребром отсекателя

1. Нужно решить систему двух уравнений (задающие положение отрезка и ребра отсекателя на плоскости) с двумя неизвестными. Отрезок задается $(y-y_1)/(x-x_1)=m$)

$$y = m(x - x_1) + y_1 \text{ — для невертикального отрезка}$$

В этих формулах m — это тангенс угла наклона отрезка.

$$x = \frac{(y - y_1)}{m} + x_1 \text{ — для негоризонтального отрезка}$$

2. Для нахождения точки пересечения с *вертикальными* ребрами необходимо найти *только ординату* точки пересечения, с *горизонтальными* — *только абсциссу* точки пересечения. Для этого известная ордината подставляется в уравнение отрезка, откуда и находится искомая абсцисса.
3. **В итоге координаты искомых точек пересечения отрезка с границами отсекателя находятся из следующих выражений:**

$$Y = m(X_l - X_1) + Y_1, m \neq \infty \text{ (не вертикальный)} \quad (\text{с левой границей})$$

$$Y = m(X_p - X_1) + Y_1, m \neq \infty \text{ (не вертикальный)} \quad (\text{с правой границей})$$

$$X = (1/m)(Y_n - Y_1) + X_1, m \neq 0 \text{ (не горизонтальный)} \quad (\text{с нижней границей})$$

$$X = (1/m)(Y_v - Y_1) + X_1, m \neq 0 \text{ (не горизонтальный)} \quad (\text{с верхней границей}).$$

4. **Указанные ограничения на значение тангенса угла наклона отрезка выполняются**, т.к. в алгоритмах предварительно анализируется характер отрезка (вертикальный, горизонтальный или общего положения) и в дальнейшем для вертикальных отрезков не ищутся точки пересечения с боковыми ребрами, а для горизонтальных отрезков не ищутся точки пересечения с нижним и верхним ребрами отсекателя.

Простой алгоритм отсечения отрезка

На основе сформированных кодов концов отрезка проверяется полная видимость, невидимость отрезка или видимость одной из вершин отрезка. Если отрезок не является полностью видимым/невидимым, то далее ищутся точки пересечения отрезка с границами отсекателя:

- анализируется возможность пересечения отрезка с очередной границей отсекателя.
- Если пересечение возможно, то находятся координаты точки пересечения,
- проводится анализ ее корректности. **Под корректностью понимается принадлежность найденной точки ребру отсекателя, а не его продолжению.**
- В случае корректности найденного пересечения точка заносится в результат и ищется вторая точка пересечения отрезка с границами отсекателя, иначе ищется пересечение со следующей стороной отсекателя.

После нахождения двух точек пересечения визуализируется полученный результат. Если не было найдено ни одного корректного пересечения, то отрезок является невидимым.

Прямая может пересечь выпуклый многоугольник только в 2 точках (или 0)

Алгоритм

1. Ввод координат отсекателя (X_l, X_n, Y_h, Y_b), координат концов отрезка $P_1(X_1, Y_1), P_2(X_2, Y_2)$.
2. Инициализация признака видимости отрезка $wid=1$ ($wid = 1$ - отрезок видимый; $wid = -1$ - отрезок невидимый).
3. Вычисление кодов концов отрезка $T1, T2$.
4. Вычисление сумм кодов концов отрезка $S_1 = \sum_{i=1}^4 T1_i, S_2 = \sum_{i=1}^4 T2_i$.
5. Проверка полной видимости отрезка:
если $(S_1=0) \& (S_2=0)$, то: (занесение в результат координат концов отрезка $R_1=P_1, R_2=P_2$ и переход к п. 31 (визуализация))
6. Вычисление логического произведения кодов концов отрезка $PL = \sum_{i=1}^4 T1_i T2_i$.
7. Проверка тривиальной полной невидимости отрезка:
если $PL \neq 0$, то ($wid = -1$ и переход к п. 31(нет визуализации, выход))
8. Задание начального значения тангенса угла наклона отрезка $m=10^{30}$ (большое число, вначале предполагается, что отрезок вертикальный).
9. Проверка видимости первого конца отрезка:
если $S_1=0$ (первый конец видим), то:
 $R_1=P_1$ (занесение этой вершины в результат),
 $Q=P_2$ (занесение координат другой вершины в рабочую переменную Q),
 $i=2$ (номер шага отсечения),
переход к п.15.
10. Проверка видимости второго конца отрезка (аналогично):
если $S_2=0$, то: $R_1=P_2, Q=P_1, i=2$ (номер шага отсечения), переход к п.15.
11. Установка начального значения шага отсечения $i=0$.
12. Вычисление текущего номера шага отсечения $i=i+1$.
13. Проверка завершения процедуры отсечения: если $i>2$, то переход к п.31.
(возможная визуализация)
14. Занесение в рабочую переменную Q координат i -ой вершины $Q=P_i$.
С вертикальными

15. Определение расположения отрезка: если $X_2=X_1$ (отрезок вертикальный), то переход к п.23 (не может быть пересечения с левой и правой границами отсекателя).
16. Вычисление тангенса угла наклона отрезка $m=(Y_2-Y_1) / (X_2-X_1)$.
17. Проверка возможности пересечения с левой границей отсекателя: если $Q_x > X_{\text{л}}$ (пересечения нет), то переход к п.20.
18. Вычисление ординаты точки пересечения отрезка с левой границей отсекателя: $Y_p = m(X_{\text{л}} - Q_x) + Q_y$.
19. Проверка корректности найденного пересечения: если $(Y_p \geq Y_h) \& (Y_p \leq Y_v)$ (пересечение корректное), то: $R_{i,x} = X_{\text{л}}$, $R_{i,y} = Y_p$ (занесение полученных координат в результат), переход к п.12.
20. Проверка возможности пересечения отрезка с правой границей отсекателя: если $Q_x < X_{\text{п}}$, то переход к п.23.
21. $Y_p = m(X_{\text{п}} - Q_x) + Q_y$.
22. если $(Y_p \geq Y_h) \& (Y_p \leq Y_v)$, то $R_{i,x} = X_{\text{п}}$, $R_{i,y} = Y_p$, переход к п.12.

С горизонтальными

23. Проверка горизонтальности отрезка: если $m=0$, то переход к п.12. (не может быть пересечения с нижней и верхней границами отсекателя).
24. Проверка возможности пересечения с верхней границей отсекателя: если $Q_y < Y_v$ (пересечения нет), то переход к п.27.
25. Вычисление абсциссы точки пересечения с верхней границей: $X_p = (Y_v - Q_y) / m + Q_x$.
26. Проверка корректности найденного пересечения: если $(X_p \geq X_{\text{л}}) \& (X_p \leq X_{\text{п}})$ (пересечение корректно), то: $R_{i,x} = X_p$; $R_{i,y} = Y_v$ (занесение полученных координат в результат); переход к п. 12.
27. Проверка возможности пересечения с нижней границей отсекателя: если $Q_y > Y_h$ (пересечения нет), то переход к п. 30 (вершина невидима и ни одно пересечение не является корректным, следовательно, весь отрезок невидим).
28. Вычисление абсциссы точки пересечения с нижней границей: $X_p = (Y_h - Q_y) / m + Q_x$.
29. Проверка корректности найденного пересечения: $(X_p \geq X_{\text{л}}) \& (X_p \leq X_{\text{п}})$ (пересечение корректно), то: $R_{i,x} = X_p$; $R_{i,y} = Y_h$; переход к п. 12.
30. Установка признака видимости $\text{wid} = -1$ (отрезок невидим полностью, так как ни одно пересечение не оказалось корректным, а прямая может пересечь выпуклый многоугольник только в 2 точках (или 0))
31. Проверка признака видимости: если $\text{wid}=1$, то вычерчивание отрезка R_1R_2 .
32. Конец.

В алгоритме СК:

- 1) не производится проверка корректности найденных точек пересечения.
- 2) каждый раз после нахождения точки пересечения (которая разбивает отрезок на полностью невидимую и видимую часть относительно очередной стороны отсекателя) отбрасывается полностью невидимая часть отрезка.

13. Отсечение. Алгоритм Сазерленда Коэна отсечения отрезка.

Алгоритм Сазерленда Коэна отсечения отрезка.

Отличия:

- 1) не производится проверка корректности найденных точек пересечения.

2) точка пересечения разбивает отрезок на видимую и невидимую части и каждый раз после нахождения точки пересечения отбрасывается полностью невидимая часть отрезка.

Невидимой будет та часть отрезка, которая заключена от вершины отрезка, невидимой относительно текущей стороны окна, до точки пересечения. Этот факт используется в алгоритме для **определения части отрезка, подлежащей отсечению**. Это связано с тем, что точка, попадающая на ребро отсекателя, имеет нулевой код (она считается видимой), поэтому попарное логическое произведение кодов концов будет равно нулю, и с его помощью не получится определить ту часть отрезка, которая невидима

При этом предполагается, что **невидимой относительно каждого ребра должна быть первая вершина отрезка**. Поэтому в случае необходимости вершины меняются местами. Следует отметить, что одновременно обе вершины не могут быть невидимыми относительно одного текущего ребра отсекателя, так как этот факт позволил бы на предварительном этапе отбросить отрезок как тривиально невидимый.

В алгоритме предусматривается предварительный анализ.

Для удобства работы данные, задающие отсекатель, заносятся в массив "окно" $O(4)$, в котором $O_1=X_l$, $O_2=X_n$, $O_3=Y_n$, $O_4=Y_b$. **Отсечение производится в определенном порядке:** левой, правой, нижней, верхней границами отсекателя. Такое задание исходных данных позволяет для горизонтального отрезка не проводить третий и четвертый этапы, а для вертикального отрезка - первый и второй этапы.

В алгоритме используется признак (**флаг**) **Fl, определяющий расположение отрезка:** $Fl=-1$ - отрезок вертикальный, $Fl=0$ - общего положения, $Fl=1$ – горизонтальный и признак **wid, определяющий видимость отрезка:** $wid=-1$ -полностью невидимый, $wid=0$ может быть частично видимый, $wid=1$ -видимый

Алгоритм

1. Ввод координат отсекателя $(X_l(O_1), X_n(O_2), Y_n(O_3), Y_b(O_4))$ и координат концов отрезка $P_1(X_1, Y_1), P_2(X_2, Y_2)$.
2. Установка флага расположения отрезка:
 $Fl=0$. Если $(P_{2,x}-P_{1,x}=0)$, то ($Fl=-1$), иначе $(m=(P_{2,y}-P_{1,y})/(P_{2,x}-P_{1,x}))$, если $(m=0)$, то ($Fl=1$)
3. Начало цикла (по i от 1 до 4) отсечения отрезка по всем четырем сторонам отсекателя.
 - 3.1. Обращение к подпрограмме (была выше) определения видимости отрезка P_1P_2 , которая возвращает признак **wid**
 - 3.2. Анализ полученного признака видимости:
если $wid = -1$, то переход к п. 5; (выход из цикла, конец)
если $wid=1$, то переход к п. 4. (выход из цикла, изображение, конец)
 - 3.3. Проверка видимости обеих вершин отрезка относительно текущей i -ой стороны окна: если $T_{1i}=T_{2i}(=0$, иначе был бы полностью невидим), то переход к п.3.6 – следующий шаг цикла
 - 3.4. Проверка видимости первой вершины: если $T_{1i}=0$ (вершина видима), то обмен местами вершин: $R=P_1; P_1=P_2; P_2=R$.
 - 3.5. Нахождение точки пересечения с текущей стороной:
если $Fl \neq -1$ (не вертикальный), то

{Если $i < 3$ (лев/прав), то ($P1.y = m(Oi - P1.x) + P1.y$; $P1.x = Oi$, переход на следующий шаг выполнения цикла), иначе(верх/низ и вот тут нет проверки на горизонтальность)
($P1.x = (O_i - P1.y)/m + P1.x$, $P1.y = O_i$)}

Иначе $P1.y = O_i$ (и вот тут нет проверки на шаг)

3.6. Конец цикла по i

4. Вычерчивание отрезка P_1P_2 .

5. Конец.

Отсечение - это операция удаления изображения за пределами выделенной области, называемой окном (отсекателем). **Стирание** – внутри.

Виды отсечений – Двумерное/Трехмерное

Отсечение в пространстве либо сводится к отсечению на плоскости, либо сами плоские алгоритмы без труда распространяются на трехмерный случай.

Чтобы выполнить данную операцию, необходимо задать тип отсекателя.

Виды отсекателей (двумерных):

- Регулярные(стандартные) - прямоугольник со сторонами, параллельными осям координат
- Нерегулярные (произвольные) – многоугольник (может иметь отверстия)
 - Выпуклый
 - Невыпуклый

Должны быть также известны геометрические характеристики изображенных объектов.

В результате отсечения должны получиться геометрические характеристики объектов, остающихся в пределах окна отсечения. **Цель отсечения** в определении точек, отрезков или их частей, которые лежат внутри отсекателя.

Отсечение используется

- в алгоритмах устранения ступенчатости
- при создании трехмерных и реалистических изображений для удаления невидимых наблюдателем линий и поверхностей (алгоритм Варнока, алгоритм Вейлера-Азертона)
- для учета теней
- формирования фактуры.

Отсечение отрезков регулярным отсекателем

Для выполнения отсечения **необходимо задать** абсциссы X_l , X_p левого и правого ребер и ординаты Y_h , Y_b нижнего и верхнего ребер.

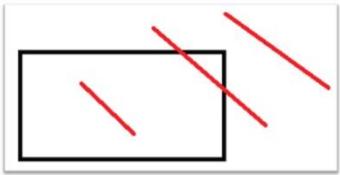
Точки, лежащие **целиком внутри окна**, удовлетворяют условию ($X_l \leq X \leq X_p \wedge (Y_h \leq Y \leq Y_b)$), где (X, Y) - координаты точки. Считается, что точки, лежащие **на границе окна, принадлежат внутренней области окна**.

Отрезок является **полностью видимым** (лев), если целиком лежит внутри отсекателя, если обе его концевые точки лежат внутри окна. Однако обратное утверждение верно не всегда. Отрезок, концевые точки которого лежат вне окна, может быть, как полностью невидимым, так и частично видимым.

Полностью невидимым (прав) называется отрезок, целиком лежащий вне отсекателя.

Частично видимым (центр) 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 выше окна.



Простой алгоритм определения видимости отрезка

Тест определяет полностью видимый отрезок как такой, у которого ни одна из координат концов отрезка не находится вне отсекателя. Если же оба конца отрезка лежат целиком слева, справа, снизу или сверху от окна, то он является невидимым. Остающиеся отрезки требуют дальнейшего исследования для установления факта их частичной видимости или полной невидимости. Собственно, для идентификации именно таких отрезков и требуются специальные алгоритмы отсечения.

Каждый раз при обработке очередного отрезка проверять выполнения данных неравенств не очень удобно, поэтому **Д.Коэн и А.Сазерленд** предложили формализовать описанную процедуру.

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

$T_1=1$, если точка лежит левее окна ($x < x_{\text{л}}$), и 0 в противном случае;

$T_2=1$, если точка лежит правее окна ($x > x_{\text{п}}$), и 0 в противном случае;

$T_3=1$, если точка лежит ниже окна ($y < y_{\text{н}}$), и 0 в противном случае;

$T_4=1$, если точка лежит выше окна ($y > y_{\text{в}}$), и 0 в противном случае.

Первым считается крайний правый бит.

в координатах экрана в условиях для T_3 и T_4 знаки строгого неравенства поменяются между собой местами

1	0	0	1
0	0	1	0
0	1	0	0

Определяется **поразрядная сумма кода** точки: $S = \text{сумма}(\text{по } i \text{ от 1 до 4})T_i$

(при представлении в виде двоичного числа – просто T)

$S = 0$ - точка видима (все биты ее кода нулевые)

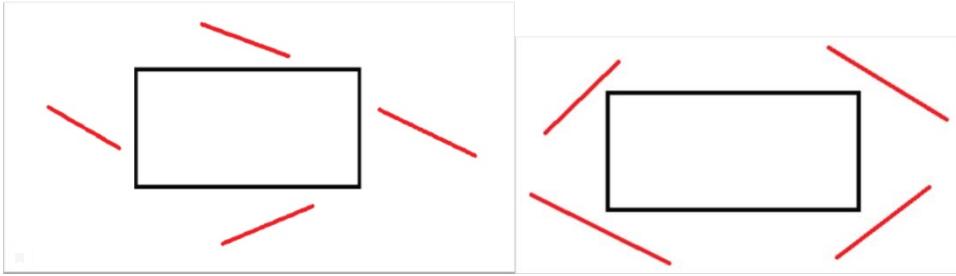
$S \neq 0$ - точка невидима.

Тогда **отрезок будет полностью видимым**, если поразрядные суммы кодов концов этого отрезка равны нулю:

$(S_1=0 \text{ и } S_2=0)$, $S_1 = \text{сумма}(\text{по } I \text{ от 1 до 4})T_{1i}$, $S_2 = \text{сумма}(\text{по } I \text{ от 1 до 4})T_{2i}$

Если отрезок не является полностью видимым, то он может являться либо частично видимым, либо полностью невидимым.

Ниже приведены два вида полностью невидимых отрезков:



Для вида полностью невидимых отрезков, предоставленного на втором рисунке, с помощью простого анализа кодов концов отрезков определить полную невидимость не получается.

Введенные коды концов отрезка можно использовать для определения полностью невидимых отрезков на 1 рисунке: если обе вершины расположены по невидимую сторону одной из границ отсекателя, то в одинаковых разрядах у них 1.

Вводится **побитное логическое произведение** кодов концов отрезка

$$P = \text{сумма(по } I \text{ от 1 до 4)} T_{1i} * T_{2i}$$

(при представлении в виде двоичного числа – $T_1 \& T_2$, где & - поразрядное логические «и»)

Отрезок является полностью невидимым и его можно отбросить, если побитное логическое произведение кодов концов отрезка не равно нулю:

$P \neq 0$ – полностью невидим

В противном случае ($P = 0$) - может быть целиком или частично видимым или даже целиком невидимым

Алгоритм:

7. Вычисление кодов первой и второй вершин отрезка T_1, T_2 .

8. Вычисление сумм кодов вершин $S_1 = \sum_{i=1}^4 T_{1i}, S_2 =$

9. Определение полной видимости отрезка: если $(S_1=0) \& (S_2=0)$, то $pr=1$ и переход к п. 6.

10. Вычисление попарного логического произведения кодов концов отрезка:

$$p = \sum_{i=1}^4 T_{1i} T_{2i} .$$

11. Определение полной невидимости отрезка: если $p \neq 0$, то $pr=-1$, иначе $pr=0$.

12. Конец.

При этом мы также имеем возможность определить частичную видимость следующим простым тестом:

$(S_1 = 0 \& S_2 \neq 0) | (S_1 \neq 0 \& S_2 = 0)$ – заведомо частично видим

Для таких отрезков от нас потребуется найти вторую вершину видимой части.

Пересечение отрезка с ребром отсекателя

ищется как решение системы двух уравнений с двумя неизвестными. Уравнениями системы являются уравнения, задающие положение отрезка и ребра отсекателя на плоскости. Решение такой системы - координаты точки пересечения этих двух отрезков. В

алгоритмах отсечения в силу большего удобства решения конкретной задачи используется параметрический способ задания отрезка в виде $(y-y_1)/(x-x_1)=m$

$$y = m(x - x_1) + y_1 \text{ — для невертикального отрезка}$$

В этих формулах m — это тангенс угла наклона отрезка.

$$x = \frac{(y - y_1)}{m} + x_1 \text{ — для негоризонтального отрезка}$$

Для нахождения точки пересечения с *вертикальными* ребрами необходимо найти *только ординату* точки пересечения, с *горизонтальными* — *только абсциссу* точки пересечения. Для этого известная ордината подставляется в уравнение отрезка, откуда и находится искомая абсцисса.

В итоге координаты искомых точек пересечения отрезка с границами отсекателя находятся из следующих выражений:

$$Y = m(X_{л}-X_1) + Y_1, m \neq \infty \text{ (не вертикальный) (с левой границей)}$$

$$Y = m(X_{п}-X_1) + Y_1, m \neq \infty \text{ (не вертикальный) (с правой границей)}$$

$$X = (1/m)(Y_{н}-Y_1) + X_1, m \neq 0 \text{ (не горизонтальный) (с нижней границей)}$$

$$X = (1/m)(Y_{в}-Y_1) + X_1, m \neq 0 \text{ (не горизонтальный) (с верхней границей).}$$

Где (X_1, Y_1) и (X_2, Y_2) — координаты двух точек, через которые проходит отрезок $m=(Y_2 - Y_1)/(X_2 - X_1)$ — тангенс угла наклона отрезка,

Указанные ограничения на значение тангенса угла наклона отрезка выполняются, т.к. в алгоритмах предварительно анализируется характер отрезка (вертикальный, горизонтальный или общего положения) и в дальнейшем для вертикальных отрезков не ищутся точки пересечения с боковыми ребрами, а для горизонтальных отрезков не ищутся точки пересечения с нижним и верхним ребрами отсекателя.

14. Отсечение Алгоритм разбиения средней точкой при отсечении отрезка.

Алгоритм

1. Ввод $(X_{л}, X_{п}, Y_{н}, Y_{в}), P_1(X_1, Y_1), P_2(X_2, Y_2)$, точности ϵ

2. Вычисление кодов концевых точек T_1 и T_2 , вычисление сумм кодов концов

$$S_1 = \sum_{i=1}^4 T_{1_i}, S_2 = \sum_{i=1}^4 T_{2_i}$$

3. если $(S_1==0)$ and $(S_2==0)$, то переход к п. 8.визуализация

4. Вычисление побитного логического произведения

5. Если $P!=0$, то переход к п. 9. Конец

6. Запоминание вершины $R=P_1$, $tr=T_1$

7. Поиск видимой вершины: цикл по $_$ in range(2)

7.1. если $S_2==0$, то занесение P_2 в результат, переход к 7.4,

7.2. Поиск точки пересечения. Цикл пока $|P_1 - P_2| > \epsilon$:

$$7.2.1. P_{ср.} = (P_1 + P_2)/2$$

7.2.2. Тср

7.2.3. Проверка видимости отрезка $[P_{ср.}, P_2]$. Если отрезок полностью невидим, то $P_2=P_{ср.}$ (отрезаем его), иначе $P_1=P_{ср.}$ (приближаемся к точке пересечения с другой стороны)

7.3. Если отрезок не является полностью невидимым $t1 \& t2 == 0$, то занесение в результат вершины P_2 , иначе весь отрезок объявляется невидимым, к пункту 9

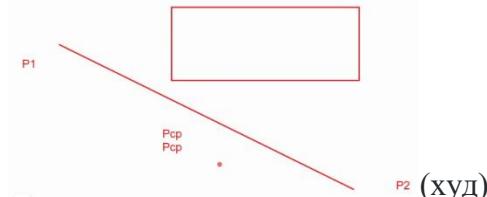
7.4. Обмен местами вершин $P_1=P_2, t1=t2, P_2=R, t2=tr$ и переход к пункту 7 для поиска второй вершины

8. Визуализация отрезка
9. Конец

(описание не очень соответствует)

1. В предыдущих алгоритмах координаты точки пересечения отрезка с ребром отсекателя определялись непосредственно путем решения соответствующих уравнений (аналитически). В данном алгоритме непосредственное вычисление координат отсутствует, **точка пересечения определяется с использованием двоичного поиска (численным методом)**.
2. **Поиск точки пересечения означает** поиск наиболее далеко расположенной от вершины P_1 еще видимой точки отрезка.
3. Еще одно отличие – **необходимо задавать точность epsilon**. Каждое разбиение средней точкой дает грубую оценку искомых пересечений, поэтому **разбиения производятся до тех пор**, пока не будет обнаружено пересечение с заранее заданной точностью. Предельная точность при работе с целыми числами $|P_1 - P_2| \leq e$, $e = \sqrt{2}$
4. **Деление отрезка пополам:** $P_{cp} = (P_1 + P_2) / 2$

Анализ, остальные разбивают на две равные части. К каждой из половин отрезков применяют те же проверки до тех пор, пока не будет обнаружено пересечение со стороной окна отсечения, либо пока он не выродится в точку (худший случай). Затем определяется видимость вычисленной точки.



полный анализ каждой концевой точки отрезка состоит из следующих трех этапов:

- проверка на видимость, в положительном случае она принимается за наиболее удаленную видимую точку и анализ завершен,
- проверка на полную невидимость отрезка, в положительном случае отрезок не рисуется, анализ завершен,
- грубая оценка наиболее удаленной видимой точки путем деления исследуемого отрезка средней точкой. Применение вышеизложенных проверок и оценок к двум полученным частям отрезка.

Если часть отрезка является полностью невидимой, то средняя точка даёт верхнюю оценку для наиболее удаленной видимой точки. Если же средняя точка видима, то она даёт нижнюю оценку для наиболее удаленной видимой точки.

Оценка эффективности

Сам по себе алгоритм простой. Но весь процесс итерационный, и он может затянуться.

Программная реализация данного алгоритма медленнее, чем алгоритма Сазерленда-Коэна; аппаратная же, за счет того, что сложение и деление на 2 очень быстры, намного эффективнее.

Отсечение - это операция удаления изображения за пределами выделенной области, называемой окном (отсекателем). **Стирание** – внутри.

Виды отсечений - Двумерное Трехмерное

Отсечение в пространстве либо сводится к отсечению на плоскости, либо сами плоские алгоритмы без труда распространяются на трехмерный случай.

Чтобы выполнить данную операцию, необходимо задать тип отсекателя.

Виды отсекателей (двумерных):

- Регулярные(стандартные) - прямоугольник со сторонами, параллельными осям координат
- Нерегулярные (произвольные) – многоугольник (может иметь отверстия)
 - Выпуклый
 - Невыпуклый

Должны быть также известны геометрические характеристики изображенных объектов.

В результате отсечения должны получиться геометрические характеристики объектов, остающихся в пределах окна отсечения. **Цель отсечения** в определении точек, отрезков или их частей, которые лежат внутри отсекателя.

Отсечение используется

- в алгоритмах устранения ступенчатости
- при создании трехмерных и реалистических изображений для удаления невидимых наблюдателем линий и поверхностей (алгоритм Варнока, алгоритм Вейлера-Азертонса)
- для учета теней
- формирования фактуры.

Отсечение отрезков регулярным отсекателем

Для выполнения отсечения **необходимо задать** абсциссы X_l , X_p левого и правого ребер и ординаты Y_h , Y_b нижнего и верхнего ребер.

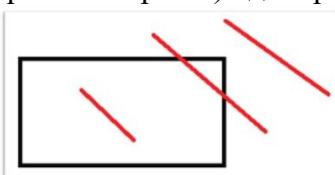
Точки, лежащие **целиком внутри окна**, удовлетворяют условию ($X_l \leq X \leq X_p \wedge Y_b \leq Y \leq Y_h$), где (X, Y) - координаты точки. Считается, что точки, лежащие **на границе окна, принадлежат внутренней области окна**.

Отрезок является **полностью видимым** (лев), если целиком лежит внутри отсекателя, если обе его концевые точки лежат внутри окна. Однако обратное утверждение верно не всегда. Отрезок, концевые точки которого лежат вне окна, может быть, как полностью невидимым, так и частично видимым.

Полностью невидимым (прав) называется отрезок, целиком лежащий вне отсекателя.

Частично видимым (центр) 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.



Простой алгоритм определения видимости отрезка

Тест определяет полностью видимый отрезок как такой, у которого ни одна из координат концов отрезка не находится вне отсекателя. Если же оба конца отрезка лежат целиком слева, справа, снизу или сверху от окна, то он является невидимым. Остающиеся отрезки требуют дальнейшего исследования для установления факта их частичной видимости или полной невидимости. Собственно, для идентификации именно таких отрезков и требуются специальные алгоритмы отсечения.

Каждый раз при обработке очередного отрезка проверять выполнения данных неравенств не очень удобно, поэтому **Д.Коэн и А.Сазерленд** предложили формализовать описанную процедуру.

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

$T_1=1$, если точка лежит левее окна ($x < x_{\text{л}}$), и 0 в противном случае;

$T_2=1$, если точка лежит правее окна ($x > x_{\text{п}}$), и 0 в противном случае;

$T_3=1$, если точка лежит ниже окна ($y < y_{\text{н}}$), и 0 в противном случае;

$T_4=1$, если точка лежит выше окна ($y > y_{\text{в}}$), и 0 в противном случае.

Первым считается крайний правый бит.

в координатах экрана в условиях для T_3 и T_4 знаки строгого неравенства поменяются между собой местами

1001	1000	1010
0001	0000	0010
0101	0100	0110

Определяется **поразрядная сумма кода** точки: $S = \text{сумма}(\text{по } i \text{ от 1 до 4})T_i$

(при представлении в виде двоичного числа – просто T)

$S = 0$ - точка видима (все биты ее кода нулевые)

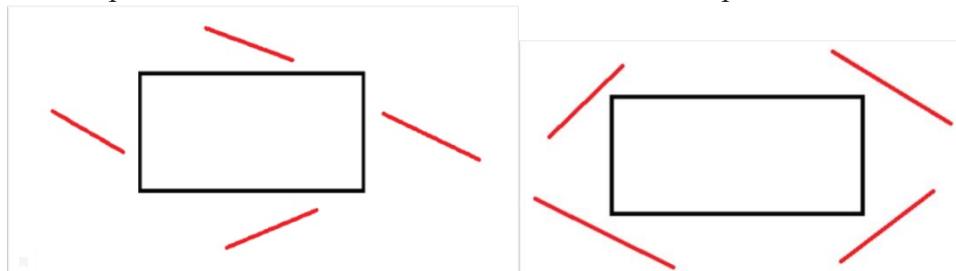
$S \neq 0$ - точка невидима.

Тогда **отрезок будет полностью видимым**, если поразрядные суммы кодов концов этого отрезка равны нулю:

$(S1=0 \text{ и } S2 = 0)$, $S1 = \text{сумма}(\text{по } I \text{ от 1 до 4})T1i$, $S2 = \text{сумма}(\text{по } I \text{ от 1 до 4})T2i$

Если отрезок не является полностью видимым, то он может являться либо частично видимым, либо полностью невидимым.

Ниже приведены два вида полностью невидимых отрезков:



Для вида полностью невидимых отрезков, предоставленного на втором рисунке, с помощью простого анализа кодов концов отрезков определить полную невидимость не получается.

Введенные коды концов отрезка можно использовать для определения полностью невидимых отрезков на 1 рисунке: если обе вершины расположены по невидимую сторону одной из границ отсекателя, то в одинаковых разрядах у них 1.

Вводится **побитное логическое произведение** кодов концов отрезка

$P = \text{сумма(по } I \text{ от 1 до 4)} T1_i * T2_i$

(при представлении в виде двоичного числа – $T1 \& T2$, где $\&$ - поразрядное логические «и»)

Отрезок является полностью невидимым и его можно отбросить, если побитное логическое произведение кодов концов отрезка не равно нулю:

$P \neq 0$ – полностью невидим

В противном случае ($P = 0$) - может быть целиком или частично видимым или даже целиком невидимым

Алгоритм:

13. Вычисление кодов первой и второй вершин отрезка $T1, T2$.

14. Вычисление сумм кодов вершин $S_1 = \sum_{i=1}^4 T1_i, S_2 =$

15. Определение полной видимости отрезка: если $(S_1=0) \& (S_2=0)$, то $pr = 1$ и переход к п. 6.

16. Вычисление попарного логического произведения кодов концов отрезка:

$$p = \sum_{i=1}^4 T1_i T2_i.$$

17. Определение полной невидимости отрезка: если $p \neq 0$, то $pr = -1$, иначе $pr = 0$.

18. Конец.

При этом мы также имеем возможность определить частичную видимость следующим простым тестом:

$(S1 = 0 \& S2 \neq 0) | (S1 \neq 0 \& S2 = 0)$ – заведомо частично видим

Для таких отрезков от нас потребуется найти вторую вершину видимой части.

15. Отсечение. Алгоритм Кируса Бека отсечения отрезка.

1. В отличие от рассмотренных выпуклый отсекатель.
2. В этом алгоритме **для определения пересечений** используется вектор внутренней нормали и параметрическая форма (векторное уравнение) задания отрезка. $P(t) = P_1 + (P_2 - P_1)t; 0 \leq t \leq 1$, где t - параметр.
3. Значения параметра t для точек пересечения отрезка с границами регулярного отсекателя определяются из следующих соотношений:

$$t = \frac{X_a - P_{1,x}}{P_{2,x} - P_{1,x}} \quad 0 \leq t \leq 1 \quad (\text{для левой границы})$$

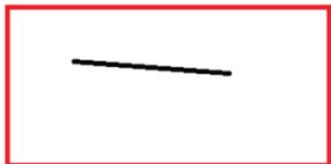
$$t = \frac{X_b - P_{1,x}}{P_{2,x} - P_{1,x}} \quad 0 \leq t \leq 1 \quad (\text{для правой границы})$$

$$t = \frac{Y_a - P_{1,y}}{P_{2,y} - P_{1,y}} \quad 0 \leq t \leq 1 \quad (\text{для нижней границы})$$

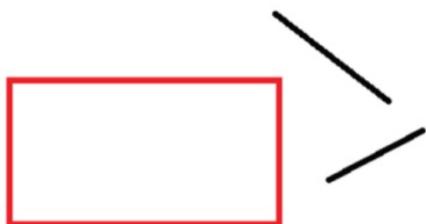
$$t = \frac{Y_b - P_{1,y}}{P_{2,y} - P_{1,y}} \quad 0 \leq t \leq 1 \quad (\text{для верхней границы})$$

Если в результате вычислений получаются значения параметра t , выходящие за пределы интервала $0 \leq t \leq 1$, то такие решения отвергаются, так как они соответствуют точкам, лежащим вне рассматриваемого отрезка.

4. Как для полностью видимого, так и для полностью невидимого отрезка значения параметра t будут лежать за пределами допустимого интервала. Поэтому не удается получить простой критерий идентификации полностью видимых и полностью невидимых отрезков. Требуется специальный для выпуклого отсекателя метод (придется пройти полный цикл).



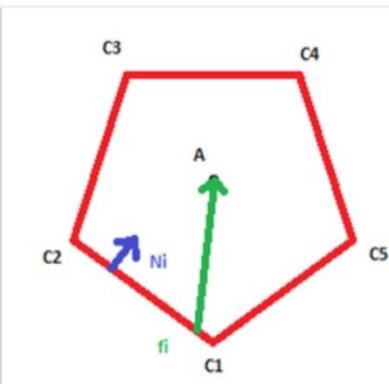
$T < 0$ для верхней и левой, $T > 1$ для нижней и правой



5. Как же распознается полностью видимый? Мы изначально положим $t_h=0$, $t_b=1$ и к концу алгоритма ничего не изменится. А для полностью невидимого получатся некорректные значения или частный случай.

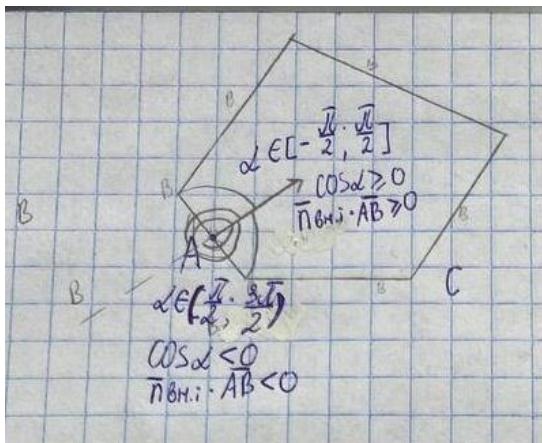
Точка по видимую или по невидимую сторону от очередной границы?

Рассмотрим ребро $C_1 C_2$. Построим внутреннюю нормаль N_i к рассматриваемой стороне из произвольной её точки. В качестве второго вектора построим вектор, начинающийся в произвольной точке f_i (удобно брать C_1) рассматриваемого ребра и заканчивающийся в заданной точке.



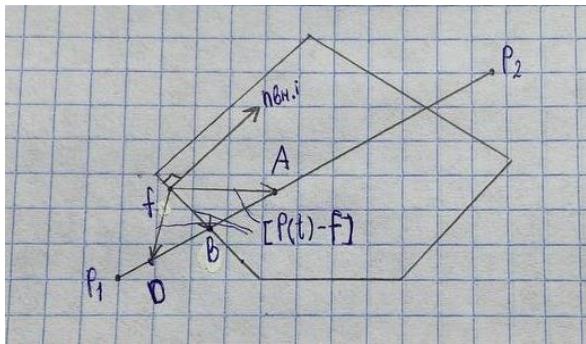
Рассмотрим скалярное произведение двух заданных векторов $N_i(A-f_i)$

- Если $N_i(A-f_i) > 0$, то точка видима относительно рассматриваемой стороны, так как косинус угла между данными векторами в этом случае положителен, то есть лежит в диапазоне от -90° до 90° , не включая эти конечные значения.
- Если $N_i(A-f_i) = 0$, то точка лежит на границе отсекателя.
- Если $N_i(A-f_i) < 0$, то точка расположена по невидимую сторону границы отсекателя.



Точки пересечения отрезка с границами отсекателя

Скалярное произведение внутренней нормали к i -му ребру окна отсечения на вектор, начинающийся в точке, лежащей на i -ом ребре окна отсечения, и заканчивающийся в любой точке исследуемого отрезка, $n_{bi} [P(t) - f_i]$, где $i=1,2,3,..m$; m - число сторон окна отсечения =0, если точка на границе



Таким образом, уравнение для определения точки пересечения отрезка с i ребром: $n_{bi} [P_1 + (P_2 - P_1) t - f_i] = 0$, или $n_{bi} [P_1 - f_i] + n_{bi} [P_2 - P_1] t = 0$.

$D = P_2 - P_1$ - вектор ориентации отрезка (директриса)

$Wi = P_1 - f_i$ – коэффициент, пропорционален расстоянию от начала отрезка до исследуемой граничной точки

Тогда условие: $t (n_{bi} D) + Wi n_{bi} = 0$, откуда $t = - (Wi n_{bi}) / (D n_{bi})$, $i=1,2..m$.

Знаменатель $D n_{bi}$ принимает нулевое значение в двух случаях:

1. $D=0$ ($P_2=P_1$) - вырождение отрезка в точку. Эта точка лежит: числитель $Wi n_{bi} < 0$ - вне, $=0$ – на границе, >0 - внутри

2. $D n_{bi}=0$ - отрезок параллелен ребру отсекателя.

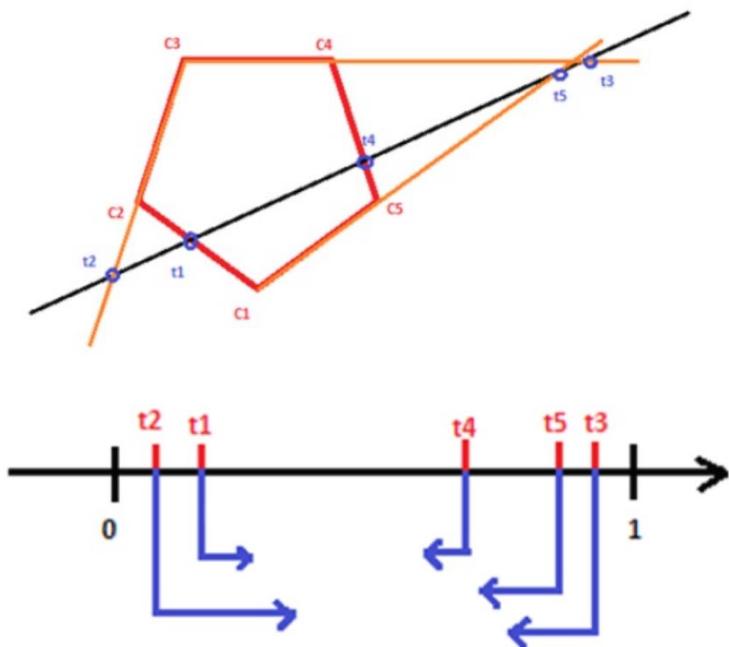
Для определения того, по какую сторону параллельный границе отрезок находится, достаточно проверить на видимость произвольную точку отрезка. Знак скалярного произведения, стоящего в числителе, $W_i * N_{vn}$ позволяет определить положение первой вершины отрезка относительно границы отсекателя. Если $W_i * N_{vn} \geq 0$, то отрезок является видимым для текущей рассматриваемой стороны отсекателя. Если же $W_{sk} < 0$, отрезок является полностью невидимым.

Выбор точек пересечения

Используя последнее уравнение, можно определить значения параметра t , при которых происходят пересечения исследуемого отрезка с ребрами окна отсечения. Если значения t

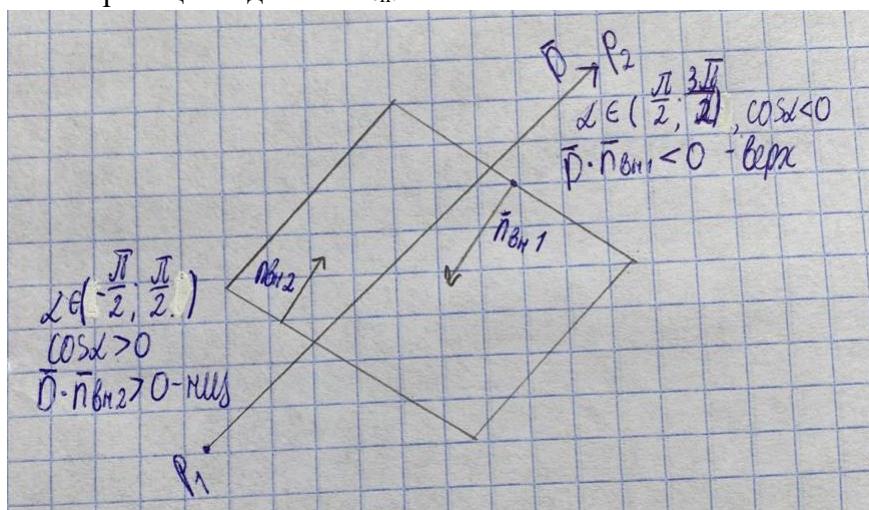
не принадлежат интервалу $0 \leq t \leq 1$, то их не рассматривают, поскольку они соответствуют точкам, лежащим вне исходного отрезка.

Несмотря на то, что отрезок может пересечь замкнутую выпуклую область не более, чем в двух точках, при решении уравнения можно получить более двух решений в интервале $0 \leq t \leq 1$. Полученные решения следует разбить на две группы: верхнюю и нижнюю, в зависимости от близости найденной точки пересечения к началу или концу отрезка.



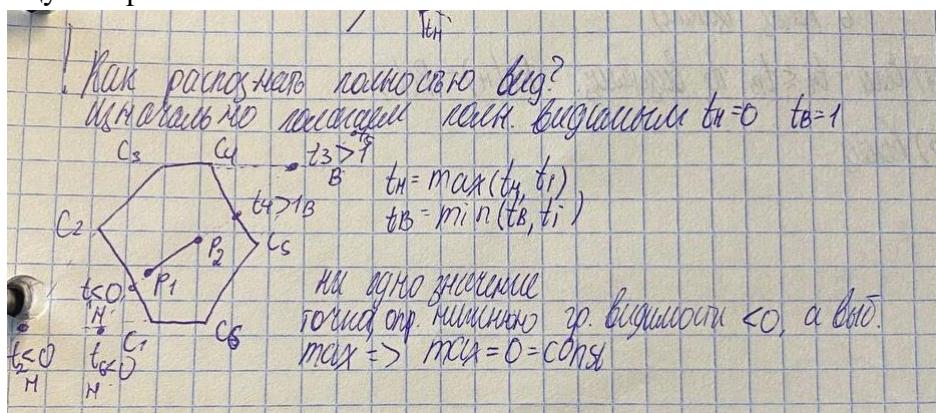
Чтобы отрезок был видимым относительно всего отсекателя, он должен быть видим относительно всех ребер отсекателя одновременно. Концам видимой части отрезка будут соответствовать два значения параметра t , одно из которых является максимальным значением из нижней группы $t_{\max\min}$, а второе - минимальным из верхней группы $t_{\min\max}$.

Найденное значение параметра t для очередной точки пересечения рассматривают в качестве возможного верхнего предела t_b , если знаменатель $D_{nBi} < 0$; в случае же, когда знаменатель положителен, значение параметра t определяет точку, которую относят к нижней границе видимости t_h .

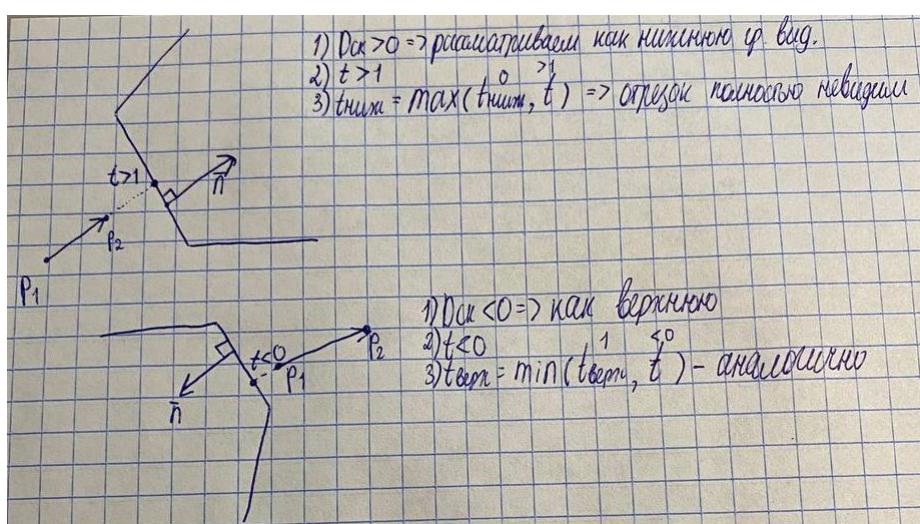


Распознавание

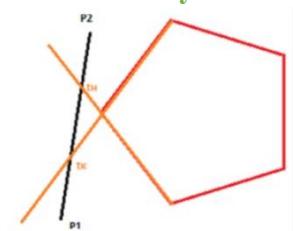
Как же распознается полностью видимый? Мы изначально положим $t_h=0$, $t_b=1$ и к концу алгоритма ничего не изменится.



А для полностью невидимого получатся некорректные значения или частный случай.



Частный случай



получим две точки, соответствующие параметрам t_h и t_k , определяющие начало и конец видимой части отрезка. То есть формально мы получаем два значения параметра, расположенных в интервале от нуля до единицы. Если не выполнить проверку того, что начало видимой части отрезка располагается до его конца, то будет получен неверный результат. То есть будет внесена следующая проверка:

$$t_h \leq t_k$$

Алгоритм

1. Ввод исходных данных: координат концевых точек отрезка $P_1(P_{1.x}, P_{1.y})$, $P_2(P_{2.x}, P_{2.y})$, числа сторон n окна отсечения и координат его вершин (массив C).

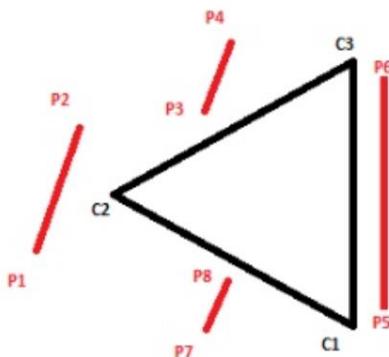
2. Проверка отсекателя на выпуклость. Если отсекатель невыпуклый, то вывести ошибку и перейти к пункту 7.
 3. Вычисление вектора ориентации отрезка $D = P_2 - P_1$.
 4. Инициализация пределов значений параметра t при условии, что отрезок полностью видимый, то есть $t_h = 0, t_b = 1$.
 5. Начало цикла по всем сторонам отсекающего окна ($i = 1..n$).
 - 5.1. Вычисление вектора внутренней нормали к очередной i -ой стороне окна отсечения - n_{bi}
 - 5.2. Вычисление вектора $W_i = P_1 - f_i (=C_i)$
 - 5.3. Вычисление скалярных произведений $W_{ci} = W_i \cdot n_{bi}, D_{ci} = D \cdot n_{bi}$.
 - 5.4. Если $D_{ci} = 0$ (вырождение отрезка в точку или его параллельность стороне отсекателя) то:
 - {если $W_{ci} < 0$, то отрезок (точка) невидим и переход к п. 7;
 - Иначе видим и переход к следующему шагу цикла}
- Иначе Вычисление параметра $t = - (W_{ci} / D_{ci})$
- 5.5. если $D_{ci} > 0$ то, поиск нижней границы параметра t :
 - если $t > 1$, то переход к п. 7 (отрезок невидим).
 - иначе $t_h = \max(t, t_b)$
 - иначе поиск верхней границы параметра t :
 - если $t < 0$, то переход к п. 7 (отрезок невидим)
 - иначе $t_b = \min(t, t_b)$
 - 5.6. Конец цикла по сторонам отсекателя.
6. Проверка фактической видимости отсеченного отрезка. Если $t_h \leq t_b$, то визуализация отрезка в интервале от $P(t_h)$ до $P(t_b)$.
 7. Конец алгоритма.

Аналогично отсекается отрезок и внешней областью отсекающего окна. Для этого определяются, а затем вычерчиваются части отрезка, лежащие вне окна отсечения. При внутреннем отсечении методом Кируса-Бека получаются значения параметра t (t_h, t_b), определяющие видимую часть отрезка (лежащую в пределах окна). Если же взять отрезки, для которых значения этого параметра лежат в интервалах $(0, t_h)$ и $(t_b, 1)$, то мы выполним фактически внешнее отсечение

Вопросы

Возьмите простейший отсекатель — треугольник (н показан у вас на рис.). Расположите невидимые отрезки, невидимость которых определяется всеми возможными способами и объясните определение невидимости.

Ответ:



Для картинки приведённой выше будем иметь:

Для отрезка P5-P6 на этапе поиска точек пересечения с гранью отсекателя C1-C3 будем иметь скалярное произведение $DNi = 0$, где D — директриса рассматриваемого отрезка, а Ni — внутренняя нормаль стороны C1-C3, причём отрезок не вырожден в точку (так как D \neq 0), а следовательно, он параллелен стороне многоугольного отсекателя. Для определения того, по какую сторону такой отрезок находится, достаточно проверить на видимость произвольную точку, выбранную на отрезке. Поэтому мы проверяем, то, какой знак у скалярного произведения $WiNi$, причём $Wi = P5 - fi$, где fi — произвольная точка, выбранная на рассматриваемой стороне отсекателя, а $P5$ — вершина рассматриваемого отрезка. Так как в указанном случае $WiNi < 0$, значит отрезок находится по невидимую сторону, а, значит, он полностью невидимый (так как видимый отрезок должен быть видим для каждой стороны отсекателя).

Отрезок P1-P2 является частным случаем. Для него будут найдены t_h и t_k , но такие, что $t_h > t_k$, что по выходу из цикла по сторонам отсекателя в алгоритме укажет на факт того, что отрезок визуализировать не надо, так как он полностью невидимый.

Для отрезка P3-P4 будет вычислен параметр $t < 0$ для стороны C1-C2, что укажет на факт того, что отрезок невидим, так как конечный предел параметра отрицателен и пересечение имеет место не для самого отрезка, а его продолжения за вершину P3.

Для отрезка P7-P8 также будет вычислен параметр t , но этот $t > 1$ для стороны C1-C2, что укажет на факт того, что отрезок невидим, так как начальный предел параметра превышает единицу и пересечение с отсекателем имеет место не для самого отрезка, а его продолжения за вершину P8.

Объясните, как распознаются полностью видимые отрезки.

Ответ: при начале работы алгоритма мы делаем предположение, что отрезок полностью видимый, то есть $t_h = 0$, а $t_k = 1$. Для каждого найденного пересечения мы будем находить параметры при $D_{Sk} > 0$: $t < t_h$; И при $D_{Sk} <= 0$: $t > t_k$, и поэтому при выходе из цикла по всем сторонам отсекателя изначально заданные параметры t_h и t_k не изменятся. А так как эти параметры не изменятся, будет высвечен отрезок полностью, что будет означать его полную видимость.

Как определяли нужное направление нормали?

Ответ: нужное направление нормали я определял с помощью скалярного произведения найденной нормали со следующим за обрабатываемым ребром многоугольника (причём общая точка двух рёбер для следующего ребра является точкой начала вектора). Данное произведение всегда должно быть больше нуля для внутренней нормали, так как алгоритм работает только с выпуклыми многоугольниками. Следовательно, если значение вычисленного произведения было меньше нуля, то я брал обратный найденному вектор нормали.

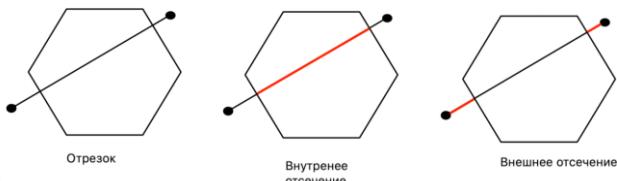
16. Внутреннее и внешнее отсечение. Определение выпуклости многоугольника; определение нормали; разбиение невыпуклых многоугольников. Триангуляция многоугольников.

Отсечение - это операция удаления изображения за пределами выделенной области, называемой окном (отсекателем). **Стирание** – внутри.

При внутреннем отсечении методом Кируса-Бека (отсечение отрезка окном выпуклой формы) получаются значения параметра t (t_h, t_b), определяющие видимую часть отрезка (лежащую в пределах окна). Если же взять отрезки, для которых значения этого параметра лежат в интервалах $(0, t_h)$ и $(t_b, 1)$, то мы выполним фактически внешнее отсечение

В Алгоритм Вейлера Азертоне (Отсечение многоугольников невыпуклыми областями) нужно сменить начало (внутреннее – с точки входного типа, внешнее – с выходного) и направление просмотра контуров отсекателя (внутреннее – начало-конец, внешнее – наоборот)

Внешнее отсечение используется при отсечении окном невыпуклой формы. Для выполнения отсечения окном невыпуклой формы следует преобразовать исходный невыпуклый многоугольник в выпуклый (это достигается путем соединения новым ребром вершин, соседних с той вершиной многоугольника, в которой нарушается выпуклость многоугольника). Затем выполняется внутреннее отсечение новым (дополненным) многоугольником. На следующем шаге отсеченный отрезок подвергается внешнему отсечению по границам дополняющего многоугольника. В итоге получаем отрезок, отсеченный невыпуклым окном.



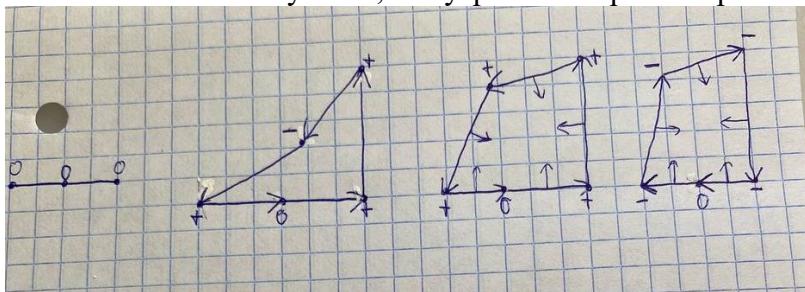
Способ определения выпуклости многоугольника

Существует ряд способов проверки многоугольника на выпуклость. Один из них основан на вычислении векторных произведений его смежных сторон (можно двигаться по/против часовой стрелки).

Векторное произведение дает вектор, перпендикулярный в данном случае плоскости oxy , под знаком подразумевается знак проекции на ось z

Правило определения выпуклости многоугольника следующее.

- Если знаки всех векторных произведений смежных сторон многоугольника равны нулю, то многоугольник вырождается в отрезок.
- Если есть, как положительные так и отрицательные знаки, то многоугольник невыпуклый.
- Если все знаки векторных произведений смежных сторон неотрицательные, то отсекающий многоугольник выпуклый, а внутренние нормали ориентированы влево от обхода. Если же все знаки неположительны, то многоугольник также является выпуклым, а внутренние нормали ориентированы вправо от его контура.



Способ определения внутренней нормали к заданной стороне многоугольника

Пусть A - некоторый вектор, нормаль к которому мы ищем, а вектор N - искомая нормаль. Запишем формулу для вычисления скалярного произведения двух этих векторов: $Ax*Nx+Ay*Ny=0$, так как вектор перпендикулярен нормали

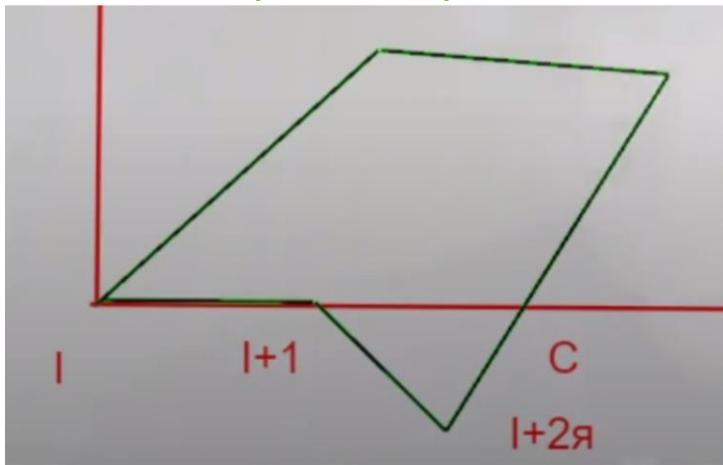
Так как нам требуется найти направление нормали, можем задать одну из проекций нормали = 1 (в моём случае, это Nx): $Ax + Ay \cdot Ny = 0$.

Если $Ay = 0$ (вектор параллелен оси ox), то вектор нормали $\{0, 1\}$, иначе $Ny = -Ax/Ay$

Найденную нормаль надо проверить, является ли она внутренней или внешней. Для этой проверки можно воспользоваться следующим фактом. Если вектор стороны многоугольника образован как разность векторов пары смежных его вершин A_{i-1} и A_i и если скалярное произведение нормали n и вектора от A_{i-1} до A_{i+1} положительно (если направление обхода дает положительный знак векторных произведений смежных сторон), то n - внутренняя нормаль. В противном случае n - внешняя нормаль. Для получения внутренней нормали полученное значение n надо умножить на -1 .

Еще один способ проверки на внутренность – аналогично проверить скалярное произведение найденной нормали и вектора следующей стороны, при условии, что 3 точки не лежат на одной прямой

Разбиение невыпуклых многоугольников



Можно предложить и третий способ определения факта выпуклости многоугольника с одновременным определением вектора внутренней нормали и разбиения исследуемого многоугольника на выпуклые многоугольники в случае его невыпуклости. Этот способ основан на использовании переносов и поворотов исследуемого многоугольника. Предполагается, что вершины многоугольника пронумерованы в направлении против часовой стрелки. Рассматриваемый алгоритм может быть представлен в следующем виде:

1. Осуществить перенос многоугольника таким образом, чтобы очередная i -ая вершина совпала с началом координат.
2. Осуществить поворот многоугольника относительно начала координат таким образом, чтобы следующая $(i+1)$ -ая вершина оказалась на положительной полуоси абсцисс x .
3. Определить знак ординаты у всех $(i+2)$ -ых вершин. Если знаки ординат у всех $(i+2)$ -ых вершин неотрицательные, то многоугольник выпуклый относительно очередной стороны, соединяющей i -ую и $(i+1)$ -ую вершины.

В повернутой системе координат внутренняя нормаль к ребру, соединяющему i -ую и $(i+1)$ -ую вершины, имеет нулевую абсциссу и ординату, знак которой совпадает со знаком ординаты $(i+2)$ -ой вершины. Для определения ориентации внутренней нормали исходного многоугольника следует выполнить обратные повороты.

Если ордината $(i+2)$ -ой вершины равна нулю, то i -ая, $(i+1)$ -ая и $(i+2)$ -ая вершины лежат на одной прямой, т.е. ребра, соединяющие i -ую и $(i+1)$ -ую вершины,

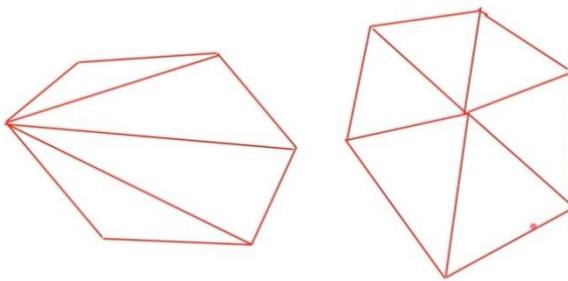
а также $(i+1)$ -ую и $(i+2)$ -ую вершины коллинеарны. Если ординаты всех $(i+2)$ -ых вершин равны нулю, то многоугольник вырожден, т.е. является отрезком.

Если знак ординаты $(i+2)$ -ой вершины отрицателен, то многоугольник невыпуклый, его следует разбить на части. Многоугольник разрезается вдоль положительной полуоси x . Для этого ищутся стороны многоугольника, пересекающие ось абсцисс, и для них находится ближайшая к началу координат точка пересечения с этой осью, абсцисса которой больше значения x_{i+1} ($x > x_{i+1}$). Далее формируются два многоугольника: первый многоугольник образован вершинами, начиная с $(i+1)$ -ой и кончая найденной точкой пересечения (многоугольник лежит целиком ниже оси абсцисс), второй многоугольник образован всеми вершинами исходного многоугольника, не вошедшими в первый многоугольник, и точкой пересечения. Полученные многоугольники могут быть невыпуклыми, поэтому описанная процедура применяется к ним до тех пор, пока все многоугольники, получаемые в процессе разбиения, не станут выпуклыми.

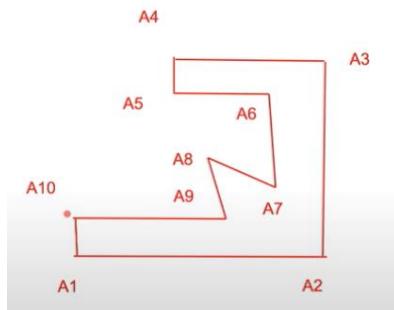
Триангуляция

Триангуляция - разбиение геометрического объекта на треугольники (прим. на самом деле на симплексы - n -мерные обобщения треугольника)

Как разбить выпуклый многоугольник - понятно. Нужно либо любую вершину соединить со всеми остальными (слева). Но тогда некоторые треугольники будут сильно "вытянутыми" (Получаем $n - 2$ треугольника, где n - количество сторон). Второй вариант - выбрать произвольную точку внутри фигуры (например, центр) и соединить её со всеми вершинами (количество полученных треугольников = количеству сторон).



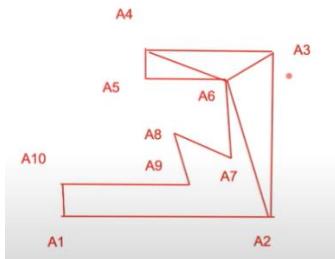
1. Разбить невыпуклый многоугольник на выпуклые.



Для начала надо найти такую вершину, чтобы диагональ, соединяющая данную вершину с какой-либо другой вершиной содержалась бы только внутри многоугольника или проходила только через ребра из тех вершин, которые она соединяет. В геометрии доказано, что в качестве такой вершины надо брать невыпуклую вершину. Один из основных способов определить невыпуклую вершину - векторное произведение. В данном примере вершины пронумерованы против часовой стрелки, значит в данном случае выпуклыми будут те вершины, где произведение будет положительным.

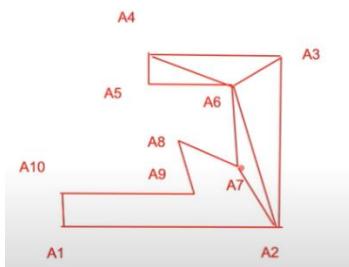
Перебираем все вершины, начиная с A1, тогда вершина A6 окажется невыпуклой, выбираем эту вершину в качестве "базовой"

Ищем такую диагональ: пробуем соединить с первой, и не получается - выходит за пределы, а вот со второй соединить уже можно, потом с вершиной A3, потом еще можно и с A4 соединить



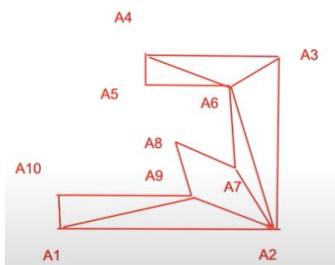
Вернемся к первому многоугольнику, и выясним, что он все еще невыпуклый. Смотрим по порядку и выясняем, что вершина A7 невыпуклая

Пробуем соединить с чем-нибудь. С вершиной A1 опять не можем, зато можем с A2

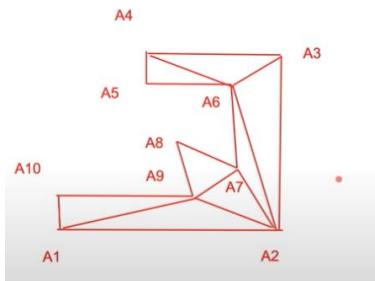


Возвращаемся к первому многоугольнику. Он опять невыпуклый. Теперь невыпуклая вершина A9.

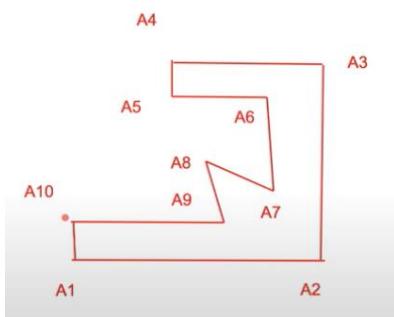
Соединяем ее с вершиной A1



Остается только четырехугольник A2-A7-A8-A9, соединим любые две противоположные вершины.

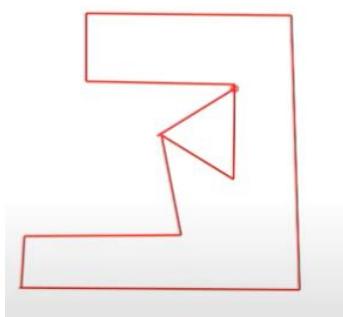


2. Дополнить невыпуклый многоугольник до выпуклого.

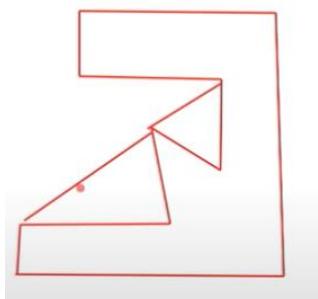


Тут нужно найти такую вершину, для которой соединение предыдущей и последующей вершины даст дополняющее ребро, которое будет располагаться вне многоугольника. Красивого способа нет, поэтому перебираем все вершины с начала.

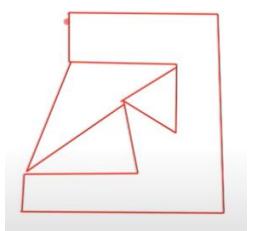
Первую взять не можем, тк диагональ между вершиной A10 и A2 будет лежать внутри. Вторую тоже взять не можем, тк диагональ будет частично лежать внутри многоугольника. ... А вот вершины A6 и A8 соединить уже можно



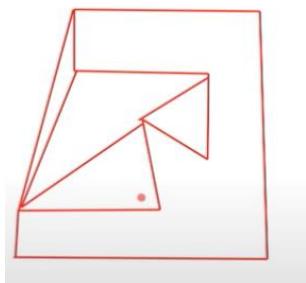
Потом соединяем вершины A8 и A10



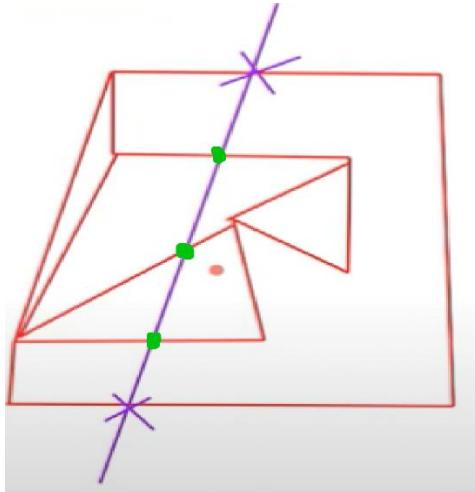
Потом соединяем вершины A10 и A5



И на последок A4 и A10



Далее нужно выполнить отсечение сначала по изначальной фигуре, а потом по треугольникам. Они должны быть разные. Т.е если, как на примере, внешнее по основному многоугольнику, то по полученным треугольникам - внутреннее.



у него на картинке прям совсем плохо было, я другими цветами обозначил

17. Отсечение многоугольников. Алгоритм Сазерленда Ходжмена

Общее

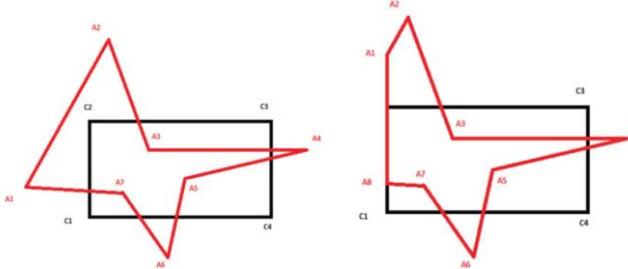
Почему нельзя воспользоваться 3 изученными алгоритмами отсечения отрезков: потому что многоугольник – часть плоскости, ограниченная замкнутой ломаной линией (а не просто линия). И если осекать, как отрезки, то можно получить незамкнутую линию. А для решения задачи требуется найти замкнутую ломанную линию, которая будет состоять не только из участков рёбер заданного исходного многоугольника, но и будет содержать в себе в качестве результата участки рёбер отсекателя.



Алгоритм Сазерленда и Ходжмена:

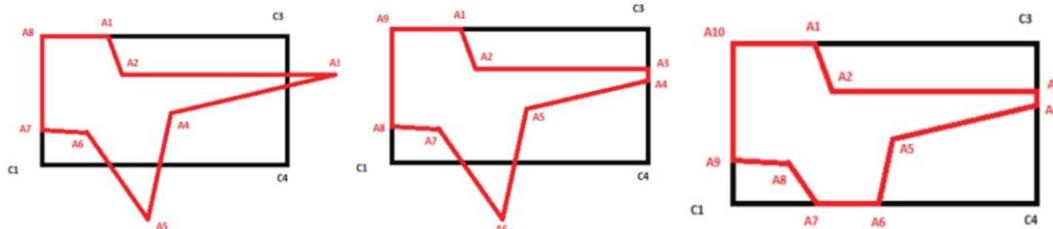
- Отсекатель – выпуклый, отсекаемый – произвольный
- Последовательно выполнять отсечение многоугольника каждой границей отсекателя и результат, полученный на очередном шаге, является исходным многоугольником для выполнения отсечения следующей границей отсекателя.
- Необходимо уметь определять видимость вершин многоугольника и находить точки пересечения его рёбер с границами отсекателя (!!прямой, проходящей через очередную границу отсекателя)
- Полностью невидимый: если в результате получен пустой список. На каждом ребре в конце можно смотреть очередной полученный список и выходить, если он пустой. Видимый – ничего не изменится

Пример



1) C1-C2.

- A1 невидима - не заносим
- A2 видима. Видимость A1 и A2 разная, значит ребро многоугольника пересекает рассматриваемую границу. Определяем точку пересечения текущего ребра с линией, проходящей через границу и заносим найденную точку и A2 в результат.
- A3, A4, A5, A6, A7 - видимы, заносим.
- A7-A1 – разная видимость. Определяем точку пересечения, заносим



Определение видимости точки

- 1) С помощью скалярного произведения: внутренней нормали и вектора от вершины ребра до точки

1) $f(S) \cdot n =$

$\begin{cases} >0 - \text{видим} \\ <0 - \text{невидим} \\ =0 - \text{на границе} \end{cases}$

- 2) С помощью пробной функции

из уравнения прямой, проходящей через ребро отсекателя: $f_{\text{проб}} = Ax + By + C$.
Подставляем координаты исследуемой и известной точек (отсекатель выпуклый, поэтому можно взять вершину отсекателя, не принадлежащую этому ребру). Если совпадают – видима.

- 3) С помощью векторного произведения: вектор ребра и вектор от вершины до точки

3) Исп. векторного произв.

$C_i \cdot S =$

$\begin{cases} >0 - \text{видим} \\ <0 - \text{невидим} \\ =0 - \text{на границе} \end{cases}$

Определение точки пересечения

параметрическая форма задания:

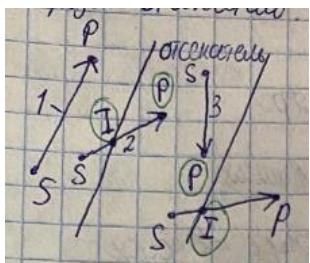
$P(t) = P_1 + (P_2 - P_1)t, 0 \leq t \leq 1$ – ребро отсекаемого многоугольника (отрезок)

$Q(s) = Q_1 + (Q_2 - Q_1)s$ – граница отсекателя (прямая)

$P(t) = Q(s)$ – система 2 уравнений с 2 неизвестными, решаем и получаем значение параметра соответствующей точки пересечения. После этого вычисляются x и y координаты точки пересечения.

Как здесь не допустить деления на 0 (когда прямые не пересекаются): видимость концов отрезка должна быть разной

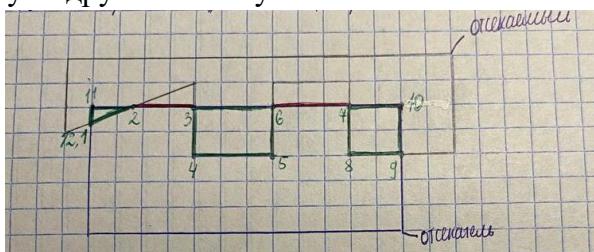
Взаимное положение стороны отсекаемого и внутренней области отсекателя



- 1) обе невидимы: в результат ничего не заносится.
- 2) начальная невидима, а конечная видима (отрезок входит внутрь отсекателя): найти точку пересечения, занести в результат найденную и конечную точки.
- 3) обе видимы: но начальная вершина текущего ребра является конечной вершиной предыдущего ребра – она будет занесена на предыдущем шаге, заносим только конечную
- 4) начальная видима, конечная – невидима (выходит наружу из отсекателя): начальная занесена на предыдущем шаге, а на текущем шаге определить точку пересечения и занести её.

Недостаток алгоритма – «ложные» ребра

Ребра, которых быть не должно, появляются, так как подразумевалось, что на выходе алгоритма получается 1 массив и все вершины соединяются последовательно. Такие рёбра возникают в том случае, когда в результате отсечения получается несколько не связанных друг с другом многоугольников.



10-11 -> 10-7, 7-6, 6-3, 3-2, 2-11

Как их определить?

- 1) часть ребра отсекателя (но это может быть и нормально)
- 2) геометрический – есть полностью невидимые ребра (необходимо, но недостаточно)
- 3) графический способ – ребро проходит дважды. Идея – разбить каждое ребро результата на под-отрезки с помощью других вершин результата. В полученных ребрах если ребро встречается четное число раз-ложное, иначе-норм

Алгоритм:

Для удобства в конец отсекателя-первая и вторая, отсекаемого – первая

1. Ввод: np, p – количество и вершины отсекаемого, nc, c-отсекателя
2. Проверка и подготовка отсекателя (выпуклый) и отсекаемого(мн-к)
3. Цикл по ребрам отсекателя (по I от 0 до nc-1):
 - 3.1. Обнуляем количество вершин результата nq=0, q=[]
 - 3.2. Подготовка признака видимости предыдущей wid_prev= видимость(p[0])
 - 3.3. Цикл по ребрам отсекаемого (по j от 1 до np). Вершина j рассматривается как кон.
 - 3.3.1. Вычисление признака видимости текущей wid_cur= видимость(p[j])
 - 3.3.2. Если текущее ребро отсекаемого пересекается с прямой через ребро отсекателя (если wid_prev != wid_cur, то заносим точку пересечения: (найти dot, nq+=1, q.append(dot))
 - 3.3.3. Если wid_cur, то заносим (nq+=1, q.append(p[j]))
 - 3.3.4. wid_prev= wid_cur
 - 3.4. Если невидим относительно ребра (nq==0), то полностью невидим (переход к концу)
 - 3.5. Готовим исходный отсекаемый для следующего шага (q.append(q[0]); np,p = nq, q)
4. Удалить ложные ребра
5. Конец

Три положения, когда отсекатель и отсекаемый не пересекаются:

1. Рядом – относительно одной из границ невидим и конец
2. Отсекаемый внутри – все точки всегда видимы и заносятся в результат
3. Отсекаемый снаружи – после отсечения останутся точки-вершины отсекателя

Эти случаи здесь решаются сами, а в следующем алгоритме - проблема

18. Отсечение многоугольников невыпуклыми областями. Алгоритм Вейлера Азертона

Общее

- Отсекатель и отсекаемое могут быть невыпуклым и содержать отверстия.
- Отсечение как внутреннее, так и внешнее
- Работа с двунаправленными циклическими списками.
 - В результате отсечения не должно появляться новых границ. Ребра отсеченного многоугольника совпадают либо с участками ребер исходного многоугольника, либо с участками ребер отсекателя.
 - Внешние контуры отсекаемого и отсекателя обходят по часовой стрелке, а внутренние - против (можно и наоборот - против-по), чтобы внутренняя часть находилась справа от направления обхода и при пересечении делается поворот направо (при обратном ходе - налево).

На этапе ввода исходных данных необходимо сформировать двунаправленные кольцевые списки для каждого контура каждого многоугольника (2: внешние + все внутренние отверстия)

Затем надо найти все точки пересечения многоугольника и отсекателя и добавить их на соответствующие места в ранее сформированные списки границ (попадает в 2 списка).

Целесообразно установить двунаправленную связь между элементами списков-одинаковыми точками пересечения (то есть 3 указателя в элементе: предыдущий, следующий и, если точка пересечения, соответствующий элемент в другом списке)

Все точки пересечения необходимо разбить на точки входа (точка пересечения, в которой ребро отсекаемого многоугольника входит внутрь отсекателя) и точки выхода (выходит наружу). Количество точек отсечения должно быть четным!

Пример

- Отсекаемый многоугольник

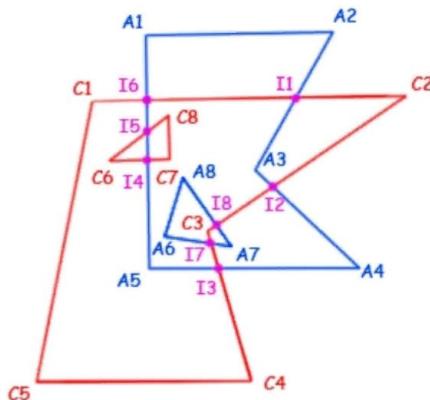
A1...A5 - Вершины внешней границы (по часовой)

A6...A8 - Вершины внутренней границы (против часовой)

- Отсекатель

C1...C5 - Вершины внешней границы (по часовой)

C6...C8 - Вершины внутренней границы (против часовой)



I1...I6 - точки пересечения внешней границы отсекаемого многоугольника с отсекателем
I7, I8 - аналогично для внутренней границы

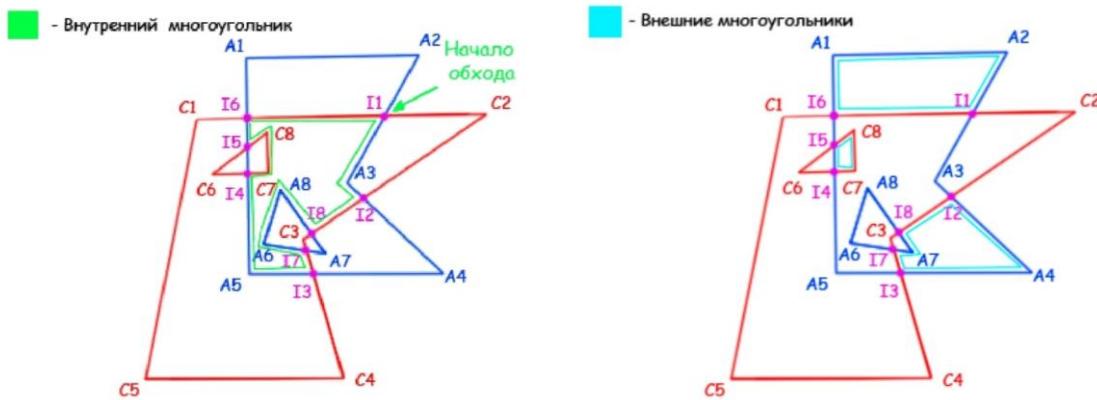
Точки пересечения: Точки входа: I1 I3 I5 I8, Точки выхода: I2 I4 I6 I7

Списки для контуров с учетом пересечений (Для наглядности первая повторно):

- Отсекаемый: Внешняя (A1 A2 I1 A3 I2 A4 I3 A5 I4 I5 I6 A1), Внутренняя (A6 I7 A7 I8 A8 A6)
- Отсекатель: Внешняя (C1 I6 I1 C2 I2 I8 C3 I7 I3 C4 C5 C1), Внутренняя (C6 I4 C7 C8 I5 C6)

Приступаем к отсечению. Начинаем со списка контуров отсекаемого многоугольника.

Для внутреннего: начать просмотр вершин с точки входного типа и просматривать списки от начала к концу, для внешнего – с выходного, границы отсекателя от конца к началу. В точке пересечения перейти от одного у другому. Все просмотренные вершины мы копируем в список вершин результирующего многоугольника, пока не замкнется. И так пока есть точки пересечений

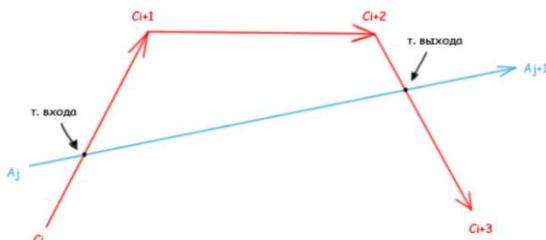


Внутренне: I1 A3 I2 -> (переход) -> I8 -> (поворачиваем направо, I8 соответствует пересечению ребра, относящегося к внутренней границе.) -> A8 A6 I7 -> I3 -> A5 I4 -> C7 C8 I5 I6 I1. У нас получился один внутренний многоугольник. Других нет.

Внешне: (I2 A4 I3 I7 A7 I8 I2), (I4 I5 C8 C7 I4), (I6 A1 A2 I1 I6)

Распознавание точек входа и точек выхода

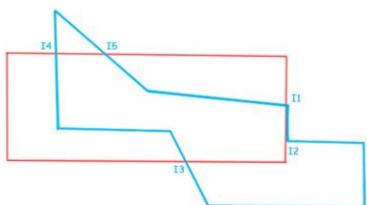
Если векторное произведение [вектора ребра отсекаемого] на [вектор ребра отсекателя] положительно, значит точка входа, отрицательно - выхода.



$A_j A_{j+1} * C_i C_{i+1} > 0$ – входа, $A_j A_{j+1} * C_{i+2} C_{i+3} < 0$ – выхода.

Частные случаи

1. Ребра отсекаемого расположены на ребрах отсекателя.



Вершины будут являться не просто вершинами, а точками пересечения.

Искали точки пересечения, обнаружили, что их нечетное количество. Значит, одна из точек - точка касания (I11), которую мы исключаем, а другая-пересечения(I2), оставляем.

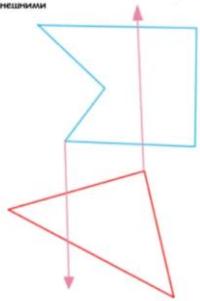
Как распознать, какая из точек I11 I2 является точкой пересечения, а какая - точкой касания: если точка пересечения совпадает с началом ребра, то это пересечение, если с концом - то это касание.

2. Границы отсекаемого многоугольника и отсекателя не пересекаются.

Предыдущий (Сазерлента-Ходжмена)правлялся сам, тут-проблема.

Надо определить, как многоугольники расположены относительно друг друга. Надо лучше спускать из точки одного многоугольника и подсчитывать количество пересечений с ребрами другого многоугольника.

1-ый случай. Отсекатель и отсекаемый многоугольник являются внешними



- У обоих четное - отсекаемый является внешним.
- Из отсекателя – четное, из отсекаемого – нечетное: отсекатель является объемлющим многоугольником для отсекаемого.
- Из отсекателя – нечетное, из отсекаемого – четное: наоборот

3. Совсем неприятный случай. Вершина отсекаемого на ребре отсекателя.
проблема тут примерно та же, что и в 1 случае: непонятно, какая точка пересечения, какая касания. Надо обработать много случаев при нахождении пересечений (до 4 пересечений при пересечении с углом отсекателя угла многоугольника). Решение – рисовать область на пиксель шире

