

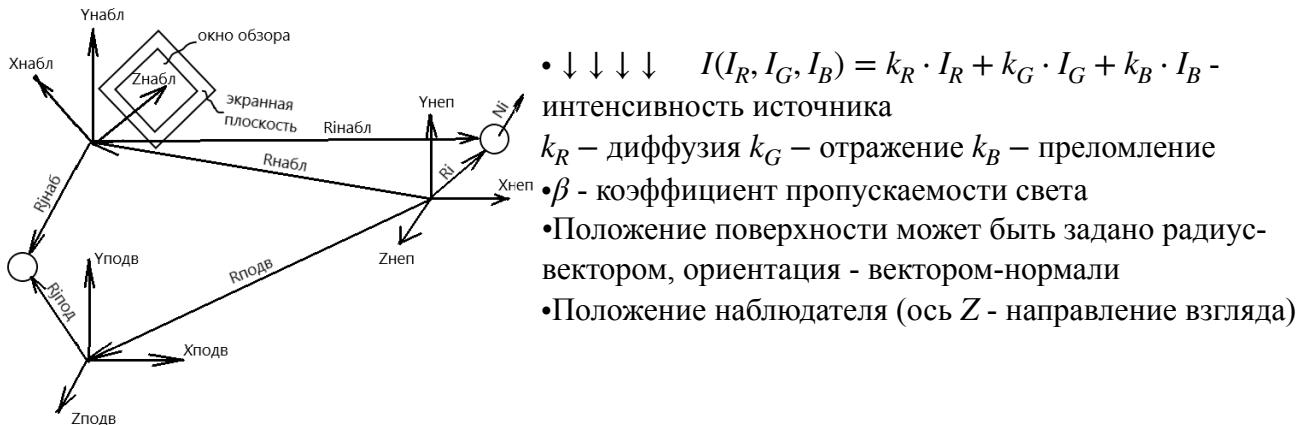
**ОТВЕТЫ К ЭКЗАМЕНУ ПО КУРСУ "КОМПЬЮТЕРНАЯ ГРАФИКА"**  
**(2018/19 уч. год)**

1.	Задача синтеза сложного динамического изображения. Этапы синтеза изображения. Последовательность и основное содержание.	3
2.	Преобразования на плоскости. Вывод расчетных соотношений. Матрицы преобразований.	5
3.	Построение плоских кривых. Выбор шага изменения аргумента. Алгоритм построения эллипса и окружности по методу средней точки.	7
4.	Требования, предъявляемые к алгоритмам вычерчивания отрезков. Пошаговый алгоритм разложения отрезка в растр. Разложение в растр по методу цифрового дифференциального анализатора.	9
5.	Алгоритмы Брезенхема разложения отрезков в растр. Простой алгоритм Брезенхема. Целочисленный алгоритм Брезенхема. Общий алгоритм Брезенхема.	10
6.	Основные расчетные соотношения и алгоритм Брезенхема для генерации окружности.	11
7.	Растровая развертка сплошных областей. Алгоритм с упорядоченным списком ребер.	13
8.	Заполнение многоугольников. Алгоритмы заполнения по ребрам, с перегородкой, со списком ребер и флагом.	15
9.	Алгоритм заполнения с затравкой, простой алгоритм заполнения с затравкой.	17
10.	Алгоритмы заполнения с затравкой. Построчный алгоритм заполнения с затравкой.	18
11.	Основы методов устранения ступенчатости. Алгоритм Брезенхема с устранением ступенчатости. Алгоритм By.	20
12.	Двумерное отсечение. Простой алгоритм отсечения отрезка.	22
13.	Отсечение. Алгоритм Сазерленда-Коэна отсечения отрезка.	24
14.	Отсечение Алгоритм разбиения средней точкой при отсечении отрезка.	25
15.	Отсечение. Алгоритм Кируса-Бека отсечения отрезка.	26
16.	Внутреннее и внешнее отсечение. Определение выпуклости многоугольника; определение нормали; разбиение невыпуклых многоугольников. Триангуляция многоугольников.	27
17.	Отсечение многоугольников. Алгоритм Сазерленда-Ходжмена.	28
18.	Отсечение многоугольников невыпуклыми областями. Алгоритм Вейлера-Азерттона.	30
19.	Модели трехмерных объектов. Требования, предъявляемые к моделям.	32
20.	Операции преобразования в трехмерном пространстве. Матрицы преобразований.	33
21.	Трехмерное отсечение. Виды отсекателей. Вычисление кодов концов отрезка для каждого типа отсекателей. Алгоритм отсечения отрезков средней точкой.	34
22.	Отсечение отрезков в трехмерном пространстве. Трехмерный алгоритм Кируса-Бека.	36
23.	Определение факта выпуклости трехмерных тел. Разбиение тела на выпуклые многогранники.	37
24.	Алгоритм плавающего горизонта.	38
25.	Задача удаления невидимых линий и поверхностей. Ее значение в машинной графике. Классификация алгоритмов по способу выбора системы координат (объектное пространство, пространство изображений).	39
26.	Алгоритм Робертса. Основные этапы и математические основы каждого этапа.	40
27.	Алгоритм Робертса. Формирование матрицы тела. Удаление нелицевых граней.	41
28.	Алгоритм Робертса. Удаление отрезков, экранируемых другими телами.	42
29.	Удаление невидимых линий и поверхностей в пространстве изображений. Алгоритм Варнока (разбиение окнами): последовательность действий и основные принципы.	43
30.	Типы многоугольников, анализируемых в алгоритме Варнока. Методы их идентификации.	45
31.	Алгоритм Вейлера-Азерттона удаления невидимых линий и поверхностей.	47
32.	Алгоритм, использующий Z-буфер.	48
33.	Алгоритм, использующий список приоритетов.	49

34. Алгоритм построчного сканирования, использующий Z-буфер. Интервальные методы построчного сканирования (основные предпосылки).	50
35. Алгоритм определения видимых поверхностей путем трассировки лучей.	52
36. Построение реалистических изображений. Физические и психологические факторы, учитываемые при создании реалистичных изображений. Простая модель освещения.	54
37. Построение реалистических изображений. Метод Гуро закраски поверхностей (получение слаженного изображения).	55
38. Построение реалистических изображений. Закраска Фонга (улучшение аппроксимации кривизны поверхности).	56
39. Определение нормали к поверхности и вектора отражения ( 4 способа) в алгоритмах построения реалистических изображений.	57
40. Построение теней при создании реалистических изображений. Учет теней в алгоритмах удаления невидимых поверхностей.	59
41. Учет прозрачности в модели освещения. Учет прозрачности в алгоритмах удаления невидимых поверхностей.	61
42. Учет фактуры при создании реалистических изображений.	62
43. Глобальная модель освещения с трассировкой лучей.	63
44. Алгоритм трассировки лучей с использованием глобальной модели освещения	64
45. Определение направления преломленного луча.	66

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

**Машинная графика** – совокупность методов и средств при образовании информации в графическую форму и из графической формы с помощью ЭВМ.



Методы → математические + алгоритмические

Средства → технические + программные

## Для поверхности:

1. уравнение поверхности
2. цвет
3. оптические свойства (коэффициенты зеркального отражения, диффузионного отражения, пропускания, преломления)  
Изображение строим в картинной плоскости (расположена перпендикулярно взгляду), на ней задаётся окно обзора (ось z через центр окна обзора)  
 $f = 30-50$  Гц – частота генерации изображения
4. Источник света (положение в пространстве, цвет, интенсивность)
5. Для динамических объектов – уравнения воздействия, по которым можно рассчитать положение в интересующий момент времени
6. Характеристики окружающей среды (цвет фона)
7. Системы координат
8. Положение картинной плоскости, размер окна обзора

## Этапы синтеза изображения

### Этап 1

Разработка трехмерной математической модели объектов визуальной обстановки

### Этап 2

Определение:

- направления линии визирования
- положения картинной плоскости
- размеров окна обзора
- значений управляемых сигналов

### Этап 3

Формирование управляемых операторов, определяющих пространственное перемещение объектов обстановки

### Этап 4

Преобразование модели синтезирующей обстановки к системе координат, связанной с наблюдателем

Этап 5

Отсечение объектов обстановки в пределах пирамиды видимости

Этап 6

Вычисление двумерных перспективных проекций объектов визуализации на картинную плоскость

Этап 7

Исключение невидимых элементов синтезируемой обстановки, закрашивание, затенение видимых участков

Этап 8

Вывод полученного изображения на поверхность растрового дисплея

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

Все изменения рисунков можно выполнить с помощью трех базовых операций:

- 1) переноса (перемещения) изображения;
- 2) масштабирования (увеличения или уменьшения размеров) изображения;
- 3) поворота изображения (употребляют также термины вращение, изменение ориентации).

Для реализации перечисленных операций используется аппарат линейных преобразований.

**Линейное преобразование на плоскости** - это такое отображение плоскости в себя, при котором прямая переходит в прямую. Произвольная точка с координатами (X,Y) переходит в результате линейного преобразования в точку с координатами (X1,Y1) в соответствии с выражениями:

$$\begin{aligned} X_1 &= A*X + B*Y + C \\ Y_1 &= D*X + E*Y + F \end{aligned} \quad M = \begin{pmatrix} A & D & 0 \\ B & E & 0 \\ C & F & 1 \end{pmatrix}$$

A,B,C,D,E,F - коэффициенты преобразования, однозначно его определяющие.

**В матричной форме:**

$(X_1, Y_1, 1) = (X, Y, 1) * M$ , где через M обозначается матрица преобразования.

Для двух последовательно выполняемых линейных преобразований можно записать следующее выражение

$$(X_2, Y_2, 1) = (X_1, Y_1, 1) * M_2 = (X, Y, 1) * M_1 * M_2 = (X, Y, 1) * M,$$

где X,Y - координаты исходной точки;

X1,Y1 - координаты точки после первого преобразования;

X2,Y2 - координаты точки после второго преобразования;

M1,M2,M - матрицы преобразований.

Поскольку в общем случае операция умножения матриц не является коммутативной, то в общем случае и два последовательных линейных преобразования некоммутативны.

Если определитель матрицы преобразования отличен от нуля, то такое преобразование будет являться аффинным.

Преобразования, при которых сохраняется плоскость вырождается в прямую или точку, параллельность прямых сохраняется и существует обратное преобразование называются **аффинскими**.

Аффинное преобразование может быть представлено суперпозицией трех преобразований: переноса, масштабирования, поворота.

**Перенос** изображения заключается в перемещении отображеного объекта из одного места экрана в другое место.

2 параметра: dx и dy

$$\begin{aligned} x_1 &= x + dx \\ y_1 &= y + dy \end{aligned} \quad M_{\text{пер}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{pmatrix}$$

### Масштабирование

Размер рисунка можно изменить, если умножить все расстояния между точками на некоторую постоянную величину (коэффициент масштабирования).

4 параметра: Центр масштабирования  $M(x_M, y_M)$ , коэффициенты масштабирования  $k_x, k_y$  1) 1)

Перенос, совмещающей центр масштабирования с началом координат

2) Масштабировании

3) Обратный перенос

$$k_x > 1 \quad 0 < k_x < 1$$

$k_y > 1$  - удаляется от центра

$0 < k_y < 1$  - приближается к центру

$$k_x = -1$$

$k_y = -1$  - отражение (центральная симметрия)

$$k_x = 1 \quad k_y = -1$$

$k_x = -1 \quad k_y = 1$  - осевая симметрия

$k_x = k_y$  - однородное масштабирование     $k_x \neq k_y$  - неоднородное масштабирование

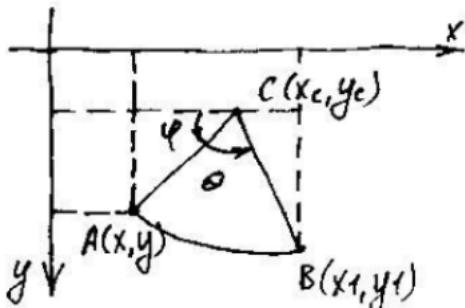
$$\begin{aligned}x_1 - x_M &= k_x(x - x_M) \\x_1 &= k_x \cdot x + (1 - k_x) \cdot x_M \\y_1 &= k_y \cdot y + (1 - k_y) \cdot y_M\end{aligned}$$

$$M_{\text{масшт}} = \begin{pmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

### Поворот

3 параметра: Центр поворота  $C(x_c, y_c)$ , угол  $\theta$

- 1) Перенос, совмещающей центр поворота с началом координат
- 2) Поворот
- 3) Обратный перенос



$$M_{\text{повор}} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{aligned}x_1 &= x_c + R\cos(180 - (\varphi + \theta)) = x_c - R\cos(\varphi + \theta) = \\&= x_c - R\cos(\varphi)\cos(\theta) + R\sin(\varphi)\sin(\theta) = \\&= x_c - (x_c - x)\cos(\theta) + (y - y_c)\sin(\theta) = \\&= x_c + (x - x_c)\cos(\theta) + (y - y_c)\sin(\theta) \\y_1 &= y_c + R\sin(180 - (\varphi + \theta)) = y_c + R\sin(\varphi + \theta) = \\&= y_c + R\sin(\varphi)\cos(\theta) + R\cos(\varphi)\sin(\theta) = \\&= y_c + (y - y_c)\cos(\theta) + (x_c - x)\sin(\theta) = \\&= y_c - (x - x_c)\sin(\theta) + (y - y_c)\cos(\theta)\end{aligned}$$

### Коммутативные преобразования

Преобр1	Преобр2
Перенос	Перенос
Масштаб	Масштаб
Поворот	Поворот
Масштаб однородное	Поворот

**Аддитивные и мультипликативные операции** – типа сложение и умножение. Аддитивные: перенос-перенос и поворот-поворот. Чтобы найти итоговую матрицу такого поворота, достаточно сложить аргументы. Мультипликативные: масштаб-масштаб, перемножить аргументы.

$$M_{\text{пер-пер}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx_1 + dx_2 & dy_1 + dy_2 & 1 \end{pmatrix} \quad M_{\text{масштаб-масштаб}} = \begin{pmatrix} kx_1 \cdot kx_2 & 0 & 0 \\ 0 & ky_1 \cdot ky_2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

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

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

#### Выбор шага

При достаточно большом радиусе, 2 соседние точки должны быть выбраны так, чтобы величина угла в радианах была не менее  $\frac{1}{R}$

#### Алгоритм средней точки для построения эллипса

$$\frac{(x - x_C)^2}{a^2} + \frac{(y - y_C)^2}{b^2} = 1 \text{ - уравнение эллипса}$$

$$f_{\text{пробная функция}} = b^2 x^2 + a^2 y^2 - a^2 b^2 = \begin{cases} < 0, \text{ точка } M \text{ расположена внутри эллипса} \\ = 0, \text{ точка } M \text{ расположена на дуге эллипса} \\ > 0, \text{ точка } M \text{ расположена вне эллипса} \end{cases}$$

$$df_{\text{начальная}} = b^2 - a^2 b + \frac{1}{4} a^2$$

$$1 \text{ интервал } \left( \frac{dx}{dy} > -1 \right)$$

.  $x_{i-1}, y_{i-1}$

$$\cdot M_{\text{ср}} \left( x_{i-1} + 1, y_{i-1} - \frac{1}{2} \right)$$

$$\cdot B(x_{i-1} + 1, y_{i-1} - 1)$$

Случай 1 дуга эллипса проходит между точкой A и точкой M  $\rightarrow$  надо выбирать точку A (точка M располагается внутри эллипса)

Случай 2 дуга эллипса проходит между точкой M и точкой B  $\rightarrow$  надо выбирать точку B (точка M располагается вне эллипса)

$$f_{\text{пробная функция}} = b^2(x_{i-1} + 1)^2 + a^2 \left( y_{i-1} - \frac{1}{2} \right)^2 - a^2 b^2$$

$$\text{Пока } x \leq \frac{a^2}{\sqrt{a^2 + b^2}}$$

$$x = x + 1$$

Необходимо корректировать значение пробной функции для следующего шага

Если  $f_{\text{пр}} > 0$ , то  $y = y - 1$  и  $df = df - 2a^2 \cdot y_{i-1}$

$$df = df + 2b^2 x_{i-1} + b^2$$

Между интервалов необходимо еще скорректировать пробную функцию

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

$$2 \text{ интервал } \left( \frac{dx}{dy} < -1 \right)$$

.  $x_{i-1}, y_{i_1}$

$$\cdot A(x_{i-1}, y_{i_1} - 1) \quad \cdot M_{\text{ср}} \left( x_{i-1} + \frac{1}{2}, y_{i-1} - 1 \right) \quad \cdot B(x_{i-1} + 1, y_{i-1} - 1)$$

Случай 1 дуга эллипса проходит между точкой A и точкой M  $\rightarrow$  надо выбирать точку A (средняя точка расположена вне эллипса)

Случай 2 дуга эллипса проходит между точкой  $M$  и точкой  $B \rightarrow$  надо выбирать точку  $B$  (средняя точка расположена внутри эллипса)

$$f_{\text{пробная функция}} = b^2 \left( x_{i-1} + \frac{1}{2} \right)^2 + a^2(y_{i-1} - 1)^2 - a^2b^2$$

Пока  $y \geq 0$

$$y = y - 1$$

Необходимо корректировать значение пробной функции для следующего шага

Если  $f_{\text{пр}} < 0$ , то  $x = x + 1$  и  $df = df + 2b^2 \cdot x_{i-1}$

$$df = df - 2a^2y_{i-1} + a^2$$

#### **Алгоритм средней точки для построения окружности**

1. Значение пробной функции  $p = \frac{5}{4} - r$

2.  $x = 0, y = r$

3. Высветить пиксели  $(x; y)$  и  $(y; x)$  (с учетом смещения центра для 4 сторон)

4. Пока  $x < y$

4.1  $x = x + 1$

4.2 Если  $p < 0$ , то  $p = p + 2x + 1$ , иначе  $y = y - 1$  и  $p = p + 2(x - y) + 1$

4.3 Высветить пиксели  $(x; y)$  и  $(y; x)$

## **4. Требования, предъявляемые к алгоритмам вычерчивания отрезков. Пошаговый алгоритм разложения отрезка в растр. Разложение в растр по методу цифрового дифференциального анализатора.**

Растр - совокупность точек.

### **Требования:**

- отрезок начинается и заканчивается в определенных точках
- интенсивность отрезка не должна зависеть от длины и угла наклона
- алгоритмы должны работать быстро

Первое требование в силу дискретной природы растрового дисплея выполнено всегда быть не может. Можно лишь добиться того, что визуально (человеческим глазом) отрезок будет восприниматься прямым. Решение этой задачи может достигаться путем увеличения разрешающей способности экрана дисплея и применения методов устранения ступенчатости.

Второму требованию удовлетворяют также только горизонтальные, вертикальные и наклоненные под углом в 45° отрезки. Однако вертикальные и горизонтальные отрезки по сравнению с отрезками, расположенными под 45°, будут выглядеть ярче, так как расстояние между соседними пикселями у них меньше, чем у наклонных отрезков. Обеспечение постоянной яркости вдоль отрезка требует вычисления очередного пикселя яркостью, зависящей от расстояния между пикселями, вычисление которого производится с использованием операций извлечения квадратного корня и умножения. Использование этих операций существенно замедляет работу алгоритма, поэтому второе требование остается, как правило, невыполненным.

Удовлетворение третьего требования достигается путем сведения к минимуму вычислительных операций, использования операций над целочисленными данными, а также реализацией алгоритмов на аппаратном или микропрограммном уровне.

**Расположение (вычерчивание) отрезков в растр** — процесс определения пикселей наилучшим образом аппроксимирующий (представляющий) заданный отрезок.

3 очевидных случая: горизонтальная линия, вертикальная и под углом 45°

В остальных случаях на каждом шаге необходимо выбирать из двух пикселей.

### **Алгоритм цифрового дифференциального анализатора**

1. Ввод  $(x_H, y_H), (x_K, y_K)$

2. Анализ отрезка на вырожденность

3. Вычислить  $dx = x_K - x_H$     $dy = y_K - y_H$

4. Если  $|dx| > |dy|$ , то  $l = |dx|$ , иначе  $l = |dy|$

5. Вычислить  $s = \frac{dx}{l}$ ,  $sy = \frac{dy}{l}$

6.  $x = x_H, y = y_H$

7. Цикл построения отрезка (от 1 до  $l + 1$ )

    7.1. Высветить точку с координатами  $T(E(x), E(y))$ ,  $E$  – округление

    7.2.  $x = x + sx, y = y + sy$

    7.3 Конец цикла

8. Конец

Алгоритм работает медленнее других алгоритмов из-за операции округления. В рассматриваемых алгоритме большее из приращений ( $\Delta X$  или  $\Delta Y$ ) выбирается в качестве единицы раstra, а приращение вдоль другой координатной оси подлежит определению. Если же поступить по-другому (меньшее из приращений взять равным единице), то отрезок на экране может получиться "дырявым", то есть состоящим из отдельных точек, не расположенных вплотную друг к другу.

## **5. Алгоритмы Брезенхема разложения отрезков в растр. Простой алгоритм Брезенхема. Целочисленный алгоритм Брезенхема. Общий алгоритм Брезенхема.**

**Ошибка (e)** — расстояние от действительного (идеального) отрезка до ближайшего пикселя (точки раstra).

Поскольку при реализации алгоритма на ЭВМ удобнее анализировать не само значение ошибки, а ее знак, то истинное значение ошибки смещается на -0,5.

**Простой алгоритм** — позволяющий построить отрезок в 1-й октанте

**Общий алгоритм** — для построения произвольного отрезка

### **Простой алгоритм Брезенхема**

1. Ввод исходных данных  $X_h, Y_h, X_k, Y_k$
2. Проверка вырожденности отрезка. Если отрезок вырожденный, то высвечивается точка и осуществляется переход к п.8
3. Вычисление приращений  $dX=X_k-X_h$  и  $dY=Y_k-Y_h$ .
4. Вычисление модуля тангенса угла наклона отрезка:  $m=dY/dX$
5. Инициализация начального значения ошибки:  $f=m-0,5$
6. Инициализация начальных значений координат текущего пикселя:  $X=X_h, Y=Y_h$
7. Цикл от  $i=1$  до  $i=dX+1$  с шагом 1:
  - 7.1 Высвечивание точки с координатами  $(X, Y)$
  - 7.2 Если  $f < 0$ , то  $Y++, f++$
  - 7.3 Вычисление ошибки  $f=f+m$
  - 7.4  $X++$

### **Общий алгоритм Брезенхема**

1. Ввод  $(x_h, y_h), (x_k, y_k)$
2. Проверка вырожденности, если отрезок вырожден, то выветить точку и переход к п. 11
3.  $x = x_h, y = y_h$
4.  $dx = x_k - x_h, dy = y_k - y_h$
5.  $sx = \text{sign}(dx), sy = \text{sign}(dy)$
6.  $dx = |dx|, dy = |dy|$
7. Если  $dx > dy$ , то  $swap = 1, swap(dx, dy)$
8.  $m = \frac{dy}{dx}$  (тангенс угла наклона)
9.  $e = m - \frac{1}{2}$  //  $e = 2dy - dx$
10. Цикл вычерчивания отрезка (по  $i$  от 1 до  $dx + 1$ )
  - 10.1 Высвечивание  $(x; y)$
  - 10.2 Если  $e \geq 0$ , то если  $swap = 0$ , то  $y = y + sy$ , иначе  $x = x + sx;$   
 $e = e - 1$  //  $e = e - 2dx$
  - 10.3 Если  $swap = 0$ , то  $x = x + sx$ , иначе  $y = y + sy$
  - 10.4  $e = e + m$  //  $e = e + 2dy$
  - 10.5 Конец цикла
11. Конец

С целой ошибкой алгоритм работает быстрее (в комментариях)

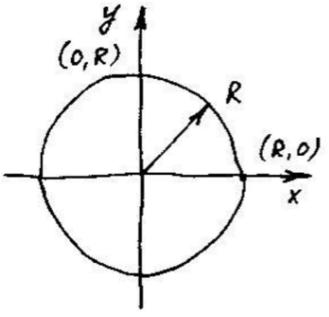
## 6. Основные расчетные соотношения и алгоритм Брезенхема для генерации окружности.

$(x - x_0)^2 + (y - y_0)^2 = R^2$  - уравнение окружности

$$x^2 + y^2 = R^2$$

$$y = \sqrt{R^2 - x^2}$$

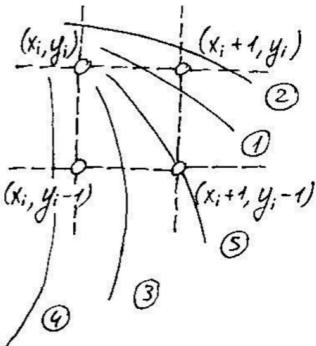
$$y = f(x)$$



Дуга от  $(0, R)$  к  $(R, 0)$   
Рассмотрим построение  $\frac{1}{4}$  части дуги окружности (1-я четверть)

$$\Delta i = (x_i + 1)^2 + (y_i - 1)^2 - R^2$$

$$\Delta i = \begin{cases} < 0, \text{диагональный пиксель расположен внутри окружности, надо выбирать горизонтальный или диагональный пиксель} \\ 0, \text{диагональный пиксель расположен на дуге окружности (он и выбирается)} \\ > 0, \text{диагональный пиксель расположен вне окружности, надо выбирать вертикальный или диагональный пиксель} \end{cases}$$



### 1. $\Delta i < 0$

$$\delta_1 = |(x_i + 1)^2 + y_i^2 - R^2| - |(x_i + 1)^2 + (y_i - 1)^2 - R^2|$$

$$\delta_1 = \begin{cases} < 0, \text{расстояние до горизонтального пикселя меньше, чем до диагонального, выбирается горизонтальный} \\ 0, \text{расстояния одинаковые, можно выбрать любой} \\ > 0, \text{расстояние до горизонтального пикселя больше, чем до диагонального, выбирается диагональный} \end{cases}$$

**Случай 1** - между горизонтальным и диагональным пикселям

$$\delta_1 = (x_i + 1)^2 + y_i^2 - R^2 + (x_i + 1)^2 + (y_i - 1)^2 - R^2 = 2[(x_i + 1)^2 + (y_i - 1)^2 - R^2] + 2y_i - 1 = 2(\Delta i + y_i) - 1$$

**Случай 2** - выше горизонтального пикселя

$$\delta_1 = -(x_i + 1)^2 - y_i^2 + R^2 + (x_i + 1)^2 - R^2 = 1 - 2y_i < 0 \quad y_i \geq 1$$

### 2. $\Delta i > 0$

$$\delta_2 = |(x_i + 1)^2 + (y_i - 1)^2 - R^2| - |x_i^2 + (y_i - 1)^2 - R^2|$$

$$\delta_2 = \begin{cases} < 0, \text{расстояние до вертикального пикселя больше, чем до диагонального, выбирается диагональный} \\ 0, \text{расстояния одинаковые, можно выбрать любой} \\ > 0, \text{расстояние до диагонального пикселя больше, чем до вертикального, выбирается вертикальный} \end{cases}$$

**Случай 3** - между диагональным и нижним пикселям

$$\delta_2 = (x_i + 1)^2 + (y_i - 1)^2 - R^2 + x_i^2 + (y_i - 1)^2 - R^2 = 2[(x_i + 1)^2 + (y_i - 1)^2 - R^2] - 2x_i - 1 = 2(\Delta i - x_i) - 1$$

**Случай 4** - ниже нижнего пикселя

Для этого случая выбор пикселя очевиден, но надо проверить корректность знака  $\delta_2$

$$\delta_2 = |(x_i + 1)^2 + (y_i - 1)^2 - R^2| - |x_i^2 + (y_i - 1)^2 - R^2| = (x_i + 1)^2 + (y_i - 1)^2 - R^2 - x_i^2 - (y_i - 1)^2 + R^2 = w x_i + 1 > 0 \quad x_i > 0$$

**Случай 5** - проходит по диагональному пикселию

$\Delta i = 0$  выбор диагонального пикселя очевиден, надо проверить корректность знаков  $\delta_1, \delta_2$

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

$$\delta_2 = |(x_i + 1)^2 + (y_i - 1)^2 - R^2| - |x_i^2 + (y_i - 1)^2 - R^2| < 0$$

### Горизонтальный шаг

$$x_{i+1} = x_i + 1$$

$$y_{i+1} = y_i$$

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

### Диагональный шаг

$$x_{i+1} = x_i + 1$$

$$y_{i+1} = y_i - 1$$

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

### Вертикальный шаг

$$x_{i+1} = x_i$$

$$y_{i+1} = y_i - 1$$

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

Чтобы построить полную окружность, достаточно сгенерировать ее одну восьмую часть.

Остальные части получаются затем путем симметричного отражения относительно определенной прямой. Так, отражая одну восьмую часть, построенную в первом октанте для углов в диапазоне 0-45°, относительно прямой с уравнением Y=X, получим одну четвертую часть, лежащую в первом квадранте. Отразив эту четверть относительно прямой X=0, получим одну вторую часть, лежащую выше оси абсцисс, наконец, отразив эту полуокружность относительно прямой Y=0, получим полную окружность.

### Алгоритм Брезенхема для генерации окружности в первом квадранте:

1. Ввод исходных данных R (радиус окружности) и при необходимости Xc, Yc (координаты центра окружности).

2. Задание начальных значений текущих координат пикселя X=0, Y=R, параметра D=2(1-R), установка конечного значения ординаты пикселя Yk=0.

3. Высвечивание текущего пикселя (X, Y).

4. Проверка окончания работы: если Y < Yk, 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;

если D > 0, то переход к п.7.

6. Вычисление параметра D1=2D+2Y-1 и анализ полученного значения:

если D1 < 0, то переход к п.8;

если D1 > 0, то переход к п.9.

7. Вычисление параметра D2=2D-2X-1 и анализ полученного значения:

если D2 < 0, то переход к п.9;

если D2 > 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 и 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 (вертикальный шаг): Y=Y-1; D=D-2Y+1.

Переход к п.3.

11. Конец.

## **7. Растворная развертка сплошных областей. Алгоритм с упорядоченным списком ребер.**

**Растворная развертка** – генерация областей на основе простых описаний рёбер или вершин (закраска). Две категории методов – растворная развертка и затравочное заполнение. При разработке более эффективных методов учитывается тот факт, что соседние пиксели скорее всего имеют одинаковые характеристики, за исключением пикселов, лежащих на границе. Это свойство принято называть свойством пространственной когерентности.

При определении точек пересечения сканирующей строки с ребрами многоугольника необязательно, чтобы они сразу оказались отсортированными в порядке возрастания координаты  $x$ . Поэтому найденные точки пересечения необходимо отсортировать в порядке возрастания координаты  $x$ . Отсортированные таким образом точки образуют пары. Для каждого интервала пикселов, задаваемого парой пересечений, используется цвет (интенсивность) заполняемого многоугольника. Для интервалов между парами пересечений и для двух крайних интервалов (от начала строки до первого пересечения и от последнего пересечения до конца строки) используется фоновый цвет (интенсивность).

Проблемы:

- 1) Каждая точка горизонтального ребра может рассматриваться как точка пересечения со сканирующей строкой. Это приведет к неправильному результату. Поэтому горизонтальные ребра игнорируются.
- 2) При пересечении сканирующей строки с ребрами многоугольника получим нечетное количество пересечений, и разбиение пикселов на пары даст неверный результат. Точку пересечения в вершине многоугольника следует учитывать два раза, если она является точкой локального экстремума, и учитывать один раз, если она не является точкой локального экстремума.

Определить локальный экстремум многоугольника в рассматриваемой вершине можно с помощью проверки ординат концевых точек двух ребер, соединенных в вершине. Если у обоих концов координаты  $y$  больше, чем у вершины, значит, вершина является точкой локального минимума. Если координаты  $y$  меньше, то вершина является точкой локального максимума.

### Простой вариант алгоритма с упорядоченным списком ребер

1. Найти все точки пересечения сканирующих строк ( $y = \text{const}$ ) с ребрами многоугольника и сохранять их в массив  $(x_1y_1)(x_2y_2)(x_3y_3)\dots(x_ny_n)$
2. Упорядочить элементы массива по невозрастанию координаты  $y$   $(x_1y_1)(x_ky_1)\dots(x_my_2)\dots(x_py_q)$  ( $y_1 \geq y_2 \geq \dots \geq y_q$ )
3. Упорядочить все элементы массива с одинаковыми значениями  $y$  по неубыванию значения координаты  $x$   
 $(x_1y_1)(x_2y_1)(x_5y_1)\dots(x_ky_1)x \leq x_2 \leq x_3 \leq \dots \leq x_k$   
 $(x_ly_2)(x_{l+1}y_2)(x_{l+2}y_2)\dots(x_ny_2)x \leq x_2 \leq x_3 \leq \dots \leq x_k$
4. Точки пересечения, расположенные на одной сканирующей строке (имеющие одинаковое значение координаты  $y$ ), разбить на пары и закрасить все пиксели, расположенные внутри интервала, ограниченного очередной парой пикселей

### БОЛЕЕ ЭФФЕКТИВНЫЕ АЛГОРИТМЫ С УПОРЯДОЧЕННЫМ СПИСКОМ РЕБЕР

В предыдущем алгоритме формируется большой список точек пересечения, который затем необходимо полностью отсортировать. Эффективность алгоритма существенно возрастает, если повысить эффективность сортировки. Эта задача решается путем разделения сортировки по координате  $y$  и сортировки в строке по возрастанию координаты  $x$  на основе групповой сортировки по  $y$ . Алгоритм в этом случае выглядит следующим образом:

## 1. Подготовка исходных данных.

- Определение для каждого ребра многоугольника точек пересечения со сканирующими строками. Для решения этой задачи используется алгоритм Брезенхема или ЦДА. Горизонтальные ребра не рассматриваются.
- Размещение координат x найденных точек пересечения в группе, соответствующей у координате сканирующей строки.
- Сортировка для каждой y-группы координат x точек пересечения в порядке возрастания, т.е.  $x_1$  предшествует  $x_2$ , если  $x_1 \leq x_2$ .

## 2. Преобразование данных в растровую форму.

- Выделение для каждой сканирующей строки из списка координат x точек пересечения пар точек пересечений. Закраска пикселов, имеющих x-координаты, лежащие в интервале  $x_1 \leq x \leq x_2$ .

## Алгоритм со списком активных ребер

### 1. Подготовка исходных данных.

- Определение для каждого ребра многоугольника наивысшей сканирующей строки, пересекаемой ребром.
- Занесение ребра многоугольника в y-группу, соответствующую этой сканирующей строки.
- Сохранение в связном списке следующих значений: начального значения координаты x точек пересечения;  $\Delta y$  - числа сканирующих строк, пересекаемых ребром многоугольника;  $\Delta x$  - шага изменения координаты x при переходе от одной сканирующей строки к следующей строке.

### 2. Преобразование этих данных в растровую форму.

- Проверка для каждой сканирующей строки y-группы на наличие новых ребер. Добавление новых ребер в САР.
- Сортировка x-координат точек пересечения из САР в порядке возрастания, т.е.  $x_1$  предшествует  $x_2$ , если  $x_1 \leq x_2$ .
- Выделение пар точек пересечений в отсортированном списке.
- Закрашивание пикселов на очередной сканирующей строке со значениями x-координаты, лежащей в интервале  $x_1 \leq x \leq x_2$ .
- Уменьшение на единицу количества пересекаемых строк для всех ребер из САР:  $\Delta y = \Delta y - 1$ . Исключение ребра из САР при  $\Delta y < 0$ . Вычисление нового значения координаты x точки пересечения со сканирующей строкой  $x = x + \Delta x$ .
- Переход к следующей сканирующей строке.

## **8. Заполнение многоугольников. Алгоритмы заполнения по ребрам, с перегородкой, со списком ребер и флагом.**

### **Алгоритм заполнения по ребрам (простой и неэффективный)**

Цвет фона

Цвет закраски

Цвет фона = Цвет закраски

Цвет закраски = Цвет фона

Дополнить все пиксели сканирующей строки, расположенные правее точки пересечения ребра многоугольника с этой сканирующей строкой

Ребра многоугольника можно обрабатывать в произвольном порядке

Смена цвета: многократно

Каждый пиксель может изменять свой цвет многократно

Количество раз определяется количеством ребер, расположенных левее этого пикселя.

Количество обрабатываемых пикселей - много лишних вне многоугольника.

### **Алгоритм заполнения с перегородкой**

Для сокращения числа обрабатываемых пикселов используется "перегородка". Рекомендуется проводить перегородку через одну из вершин многоугольника.

Для каждой сканирующей строки, пересекающей ребро многоугольника, дополнить пиксели, расположенные правее точки пересечения, но левее перегородки, если пересечение расположено левее перегородки, дополнить пиксели, расположенные левее точки пересечения, но правее перегородки, если пересечение расположено правее перегородки.

Недостатком алгоритма заполнения с перегородкой все же остается неоднократная обработка части пикселов.

Для того чтобы избавиться от этого, разработан модифицированный алгоритм заполнения сплошной области со списком ребер и флагом.

### **Алгоритм со списком ребер и флагом**

Во-первых, обрисовывается контур, ограничивающий область, в результате чего на каждой строке сканирования определяются пары ограничивающих пикселей; а во-вторых, активизируются пиксели, расположенные между вычисленными на предыдущем шаге ограничивающими пикселями.

Пересечение находится с помощью анализа цвета пикселя

1. Ввод исходных данных: цвет границы, цвет заполнения, количество вершин, координаты вершин многоугольника

2. Высвечивание (очерчивание) границ многоугольника

(Флаг - признак расположения точки внутри или вне многоугольника

$$flag = \begin{cases} true, & \text{если внутри} \\ false, & \text{если снаружи} \end{cases}$$

3. Заполняем многоугольник

Цикл по всем сканирующим строкам, пересекающим многоугольник

(по  $y$  от  $y_{max}$  до  $y_{min}$ )

$flag = false$

Цикл анализа пикселей текущей строки

(по  $x$  от  $x_{min}$  до  $x_{max}$ )

$if y(x, y) = \text{цвет границы}$

$flag = \overline{flag}$

$if flag = true$

Цвет( $x, y$ ) = цвет закраски

*else*

Цвет(x, y) = цвет фона

Конец цикла по x

Конец цикла по y

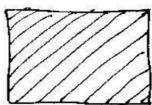
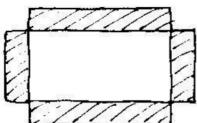
При использовании этого алгоритма следует учитывать, что ребра многоугольника, образующие очень малый угол, могут сливаться, поэтому при обрисовке контура, начиная со второго ребра, надо проверять, не совпадает ли точка нового ребра с уже высвеченным пикселом.

Алгоритм является эффективным, потому что каждый пиксель изменяет свой цвет один раз и информацию о цвете пикселя мы считываем один раз.

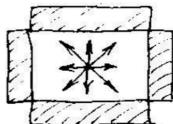
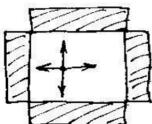
## 9. Алгоритм заполнения с затравкой, простой алгоритм заполнения с затравкой.

**Затравка (затравочный пиксель)** - точка в заполненной области

*Границно-определённые области: Внутренне-определенные*



*Области бывают 4-связные и 8-связные:*



Необходимо:

Задать область и координаты точки внутри области. Рассматриваются пиксели, соседние с данным. Используется стек.

### Простой алгоритм заполнения с затравкой

1. Ввод исходных данных: цвет границы, цвет заполнения, координаты затравки  $(x, y)$ , информация о границе
2. Очерчивание (высвечивание) границ(ы) области
3. Запись затравочного пикселя в стек
4. Пока стек не пуст, выполнять следующие действия:
  - 4.1 Извлечь затравочный пиксель из стека
  - 4.2 Закрасить извлеченный пиксель: Цвет( $x, y$ ) = Цвет закраски (Если Цвет( $x, y$ )  $\neq$  Цвет закраски)
  - 4.3 Анализ цвета четырех соседних пикселей  
Если Цвет( $x, y$ )  $\neq$  Цвет закраски и Цвет( $x, y$ )  $\neq$  Цвет границ, то поместить пиксель( $x, y$ ) в стек.

Достоинство: простой алгоритм

Недостатки: каждый затравочный пиксель помещается в стек, некоторые пиксели могут помещаться в стек по два-три раза

## **10. Алгоритмы заполнения с затравкой. Построчный алгоритм заполнения с затравкой.**

Повышение эффективности связано с уменьшением объема используемой памяти

В стек помещается один пиксель для непрерывного интервала пикселей

**Непрерывный интервал пикселей** - группа примыкающих друг к другу еще не закрашенных и не граничных пикселей, ограниченная закрашенными или граничными пикселями.

Алгоритм применим для выпуклых, невыпуклых областей, областей, содержащих отверстия.

Однако во внешней, примыкающей к данной гранично-определенной области, не должно быть пикселов с цветом заполнения.

1) закраска пикселей той же строки, где затравочный

2) поиск новых затравочных пикселей в смежных с текущей строкой строк

1. Ввод исходных данных: информация о внешней и внутренних границах, цвет границы, цвет заполнения, затравочный пиксель  $3(x, y)$

2. Очерчивание (высвечивание) границ области

3. Занести затравочный пикселя в стек

4. Пока стек не пуст, выполнить действия:

    4.1 Извлечь затравочный пиксель из стека

    4.2 Заполнить (закрасить) затравочный пиксель и пиксели текущей строки ( $y$ ), двигаясь влево и вправо от затравочного пикселя до встречи с граничными пикселями (границы не закрашивать!)

    4.3 Запомнить левую ( $x_{\text{Л}}$ ) и правую ( $x_{\text{Р}}$ ) границы интервала

    4.4 Поиск новых затравочных пикселей в интервале  $(x_{\text{Л}}) \leq x \leq (x_{\text{Р}})$  на двух соседних строках ( $y + 1, y - 1$ ) (в качестве затравочного в стек помещается самый правый не закрашенный и не граничный пиксель из интервала таких пикселей)

    4.5 Занесение в стек найденных затравочных пикселей

Занесение пикселя ( $x, y$ ) в стек

while (стек не пуст)

    Извлекаем пиксель ( $x, y$ ) из стека и закрашиваем

    Закрашиваем пиксели до правой границы  $x_{\text{пр}} = x - 1$

    Закрашиваем пиксели до левой границы  $x_{\text{Л}} = x + 1$

$y = y + 1$

$x = x_{\text{Л}}$

//Проверка в интервале  $x_{\text{Л}} \leq x \leq x_{\text{пр}}$  вышележащей строки с целью определения ее принадлежности границе многоугольника или ее полного заполнения и поиск затравки, начиная с левого края в случае ее незаполненности:

    if ( $x \leq x_{\text{пр}}$ )

$fl = 0$  // Флаг нахождения затравки

        while (Цвет( $x, y$ )  $\neq$  цвет границы & Цвет( $x, y$ )  $\neq$  цвет закраски &  $x \leq x_{\text{пр}}$ )

$fl = 1$  // наличие

$x = x + 1$

        //помещение в стек крайнего правого незаполненного пикселя

        if  $fl = 1$

            if ( $x = x_{\text{пр}}$  & Цвет( $x, y$ )  $\neq$  цвет границы & Цвет( $x, y$ )  $\neq$  цвет закраски)

                Занесение пикселя ( $x, y$ ) в стек

            else

                Занесение пикселя ( $x - 1, y$ ) в стек

$fl = 0$

// Поиск нового интервала в случае прерывания интервала пикселов уже заполненными или граничными пикселями

$xt = x$

while(Цвет( $x, y$ ) = цвет границы || Цвет( $x, y$ ) = цвет закраски &  $x < x_{\text{пр}}$ )

$x = x + 1$

if ( $x = xt$ )

$x = x + 1$

//тоже самое для нижней строки

$x = x_{\text{д}}; y = y - 2$

## 11. Основы методов устранения ступенчатости. Алгоритм Брезенхема с устранением ступенчатости. Алгоритм Ву.

Основная причина появления лестничного эффекта заключается в том, что отрезки имеют непрерывную природу тогда как растровое устройство дискретно. Принципиально устраниить ступенчатость (лестничный эффект) невозможно. Однако применением специальных методов можно добиться того, что визуально ступеньки будут слабо заметны или практически незаметны. Существует два метода устранения искажений изображения:

1. увеличить частоту выборки, что достигается с помощью увеличения размещения раstra => учитываются более мелкие детали
  2. трактовать пиксел не как точку, а как конечную область (высвечивать пиксел с разной интенсивностью пропорциональной площади части пикселя, находящейся под отрезком).
- При наличии нескольких оттенков цвета внешний вид отрезка улучшается путем размытия его краев.

### Алгоритм Брезенхема

Интенсивность пикселя устанавливается пропорционально площади части пикселя, находящейся под отрезком

1. Ввод исходных данных  $x_H, y_H, x_K, y_K$  (координаты концов отрезка),  $I$  - количество уровней интенсивности
2. Проверка вырожденности отрезка. Если отрезок вырожден, то высвечивание отдельного пикселя и переход к п.13
3. Вычисление приращений  $dx = x_K - x_H$  и  $dy = y_K - y_H$
4. Вычисление шага изменения каждой координаты:  $sx = \text{sign}(dx)$ ,  $sy = \text{sign}(dy)$
5. Вычисление модулей приращения координат:  $dx = |dx|$ ,  $dy = |dy|$
6. Вычисление модуля тангенса угла наклона  $m = \frac{dy}{dx}$
7. Анализ вычисленного значения  $m$  и обмен местами  $dx$  и  $dy$  при  $m > 1$ 
  - если  $m > 1$ , то выполнить  $\text{swap}(dx, dy)$ ;  $m = \frac{1}{m}$ ;  $fl = 1$
  - если  $m < 1$ , то  $fl = 0$
8. Инициализация начального значения ошибки  $f = \frac{I}{2}$
9. Инициализация начальных значений координат текущего пикселя:  $x = x_H, y = y_H$
10. Вычисление скорректированного значения тангенса угла наклона  $m = m \cdot I$  и коэффициента  $W = I - m$  - максимальное значение отклонения
11. Высвечивание пикселя с координатами  $(x; y)$  интенсивностью  $E(f)$
12. Цикл от  $i = 1$  до  $i = dx$  с шагом 1
  - Если  $f < W$ , то
    - a) если  $fl = 0$ , то  $x = x + sx$
    - b) если  $fl = 1$ , то  $y = y + sy$
    - c)  $f = f + m$
  - Если  $f > W$ , то  $x = x + sx, y = y + sy, f = f - W$
  - Высвечивание пикселя с координатами  $(x; y)$  интенсивностью  $E(f)$
13. Конец

$$\begin{aligned} \text{tg} \alpha &= m \quad \text{и} \quad y_{im} = m \\ S &= S_D + S_A = y_i \cdot 1 + \frac{m \cdot 1}{2} = y_i + \frac{m}{2} \end{aligned}$$

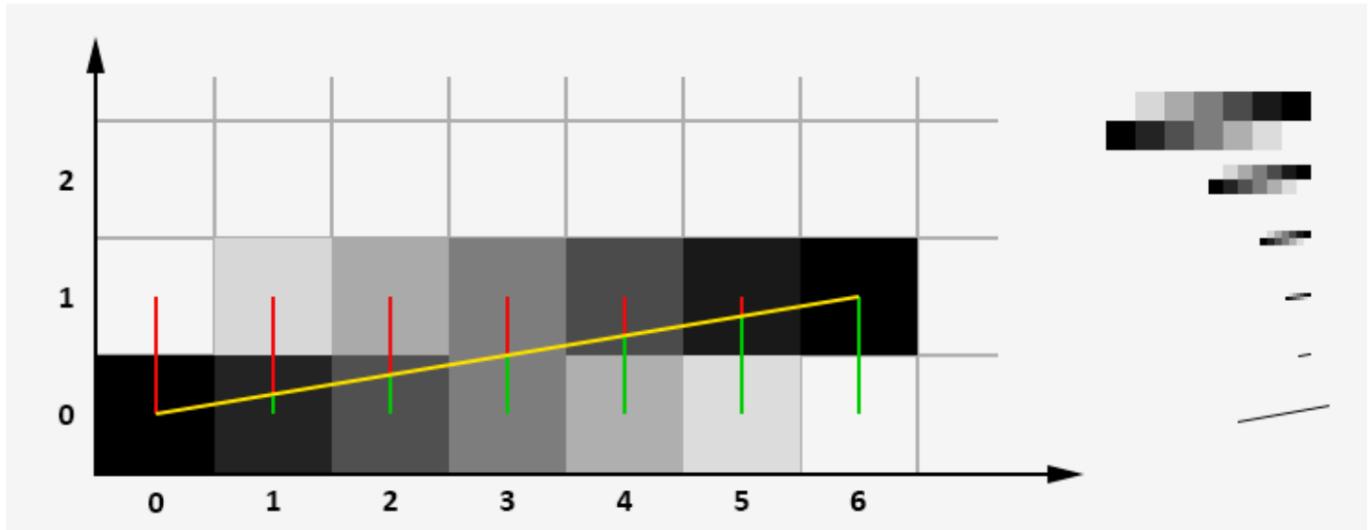
$$S = S_D + S_A = y_i \cdot 1 + \frac{m \cdot 1}{2} = y_i + \frac{m}{2}$$

$$S_{i+1} = y_i \cdot 1 + m \cdot 1 + \frac{m \cdot 1}{2} = S_i + m$$

$$S_{i+1} = y_i \cdot 1 + m \cdot 1 + \frac{m \cdot 1}{2} - 1 \cdot 1 = S_i + m - 1 = S_i - (1 - m)$$

## Алгоритм Ву

На каждом шаге ведётся расчёт для двух ближайших к прямой пикселей, и они закрашиваются с разной интенсивностью, в зависимости от удалённости. Точное пересечение середины пикселя даёт 100% интенсивности, если пиксель находится на расстоянии в 0.9 пикселя, то интенсивность будет 10%. Иными словами, сто процентов интенсивности делится между пикселями, которые ограничивают векторную линию с двух сторон.



## 12. Двумерное отсечение. Простой алгоритм отсечения отрезка.

**Отсечение** - удаление части изображения, находящейся за пределами заданной (выделенной) области.

Отсечение

1) Двухмерное

2) Трехмерное

Отсекатели

1) Регулярные (прямоугольник со сторонами параллельными координатным осям)

2) Нерегулярные

а. Выпуклые произвольные многоугольники

б. Невыпуклые произвольные многоугольники

**Видимый объект** - объект, целиком располагающийся внутри отсечения

**Невидимый объект** - объект, целиком располагающийся вне отсечения

**Частично видимый объект** - объект, часть которого располагается внутри отсечения, а часть вне отсечения

1001	1000	1010
0001	0000	0010
0101	0100	0110

$$T_1 = \begin{cases} 1, & \text{если } x < x_{\text{Л}} \\ 0, & \text{иначе} \end{cases}$$

$$T_2 = \begin{cases} 1, & \text{если } x < x_{\text{пр}} \\ 0, & \text{иначе} \end{cases}$$

$$T_3 = \begin{cases} 1, & \text{если } y < y_{\text{Н}} \\ 0, & \text{иначе} \end{cases}$$

$$T_4 = \begin{cases} 1, & \text{если } y < y_{\text{В}} \\ 0, & \text{иначе} \end{cases}$$

### Простой алгоритм отсечения

Искать точки пересечения необходимо только в том случае, когда мы уверены в том, что они есть

$$P_{\text{тек}} = P_1$$

$P_{\text{тек}} \cdot x < x_{\text{Л}}$  - пересечение ищется

$y_{\text{Л}} \leq y_{\text{пер}} \leq y_{\text{В}}$  - корректное  $\rightarrow$  конец второй точки пересечения  $P_{\text{тек}} = P_2$

$P_{\text{тек}} \cdot x < x_{\text{пр}}$  - пересечение не ищется

1. Ввод данных:  $x_{\text{Л}}, x_{\text{П}}, y_{\text{Н}}, y_{\text{В}}, P_1, P_2$
2. Инициализация признака видимости  $Fl = 1$  (отрезок видимый)
3. Инициализация тангенса угла наклона  $m = 10^{30}$  (предполагается, что отрезок вертикальный)
4. Вычисление кодов концов отрезка  $T_1, T_2$ , их сумм  $S_1, S_2$
5. Проверка полной видимости отрезка: если  $(S_1 = 0) \wedge (S_2 = 0)$ , то  $R_1 = P_1, R_2 = P_2$ , переход к п. 21
6. Проверка полной невидимости отрезка:  
выполнение  $p = \sum_{i=1}^4 T_{1i}T_{2i}$ , если  $p \neq 0$ , то  $Fl = -1$  переход к п. 21 //отрезок невидим
7. Проверка видимости первой вершины: если  $S_1 = 0$ , то  $R_1 = P_1, P_{\text{тек}} = P_2, i = 2$  (номер шага отсечения), переход к п. 13
8. Проверка видимости второй вершины: если  $S_2 = 0$ , то  $R_1 = P_2, P_{\text{тек}} = P_1, i = 2$ , переход к п. 13
9. Инициализация счетчика количества отсечений  $i = 0$
10.  $i = i + 1$
11. Если  $i > 2$ , то переход к п. 21 // Проверка завершения процедуры отсечения

12.  $P_{\text{тек}} = P_i$
13. Если  $P_2 \cdot x - P_1 \cdot x = 0$ , то переход к п. 18 // отрезок вертикальный, не может быть пересечения с левой и правой границами отсекателя
14. Находим тангенс угла  $m = \frac{P_2 \cdot y - P_1 \cdot y}{P_2 \cdot x - P_1 \cdot x}$
15. Если  $P_{\text{тек}} \cdot x < x_{\text{Л}}$ , то  $y_{\text{пер}} = m(x_{\text{Л}} - P_{\text{тек}} \cdot x) + P_{\text{тек}} \cdot y$ , проверка корректности пересечения:  
если  $y_{\text{Н}} \leq y_{\text{пер}} \leq y_{\text{В}}$ , то  $R_i \cdot x = x_{\text{Л}}, R_i \cdot y = y_{\text{пер}}$ , переходим к п. 10
16. Если  $P_{\text{тек}} \cdot x > x_{\text{пр}}$ , то  $y_{\text{пер}} = m(x_{\text{пр}} - P_{\text{тек}} \cdot x) + P_{\text{тек}} \cdot y$ , проверка корректности пересечения: если  $y_{\text{Н}} \leq y_{\text{пер}} \leq y_{\text{В}}$ , то  $R_i \cdot x = x_{\text{пр}}, R_i \cdot y = y_{\text{пер}}$ , переход к п. 10
17. Если  $m=0$ , переходим к п. 10
18. Если  $P_{\text{тек}} \cdot y < y_{\text{Н}}$ , то  $x_{\text{пер}} = \frac{1}{m}(y_{\text{Н}} - P_{\text{тек}} \cdot y) + P_{\text{тек}} \cdot x$ , проверка корректности пересечения:  
если  $x_{\text{Л}} \leq x_{\text{пер}} \leq x_{\text{пр}}$ , то  $R_i \cdot x = x_{\text{пер}}, R_i \cdot y = y_{\text{Н}}$ , переходим к п. 10
19. Если  $P_{\text{тек}} \cdot y > y_{\text{В}}$ , то  $x_{\text{пер}} = \frac{1}{m}(y_{\text{В}} + P_{\text{тек}} \cdot y) + P_{\text{тек}} \cdot x$ , проверка корректности пересечения:  
если  $x_{\text{Л}} \leq x_{\text{пер}} \leq x_{\text{пр}}$ , то  $R_i \cdot x = x_{\text{пер}}, R_i \cdot y = y_{\text{В}}$ , переход к п. 10
20.  $Fl = -1$
21. Если  $Fl = 1$ , то визуализация отрезка  $R_1R_2$
22. Конец

## 13. Отсечение. Алгоритм Сазерленда-Коэна отсечения отрезка.

Алгоритм Сазерленда-Коэна, как и в предыдущем случае, предусматривает нахождение точек пересечения отрезка со сторонами окна прямоугольной формы. Однако здесь не производится проверка корректности найденных точек пересечения. Найденная точка пересечения разбивает отрезок на две части: полностью невидимую относительно очередной стороны отсекателя и видимую часть. 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.

$$Fl = \begin{cases} -1, & \text{отрезок вертикальный} \\ 1, & \text{отрезок горизонтальный} \\ 0, & \text{общего положения} \end{cases} \quad wid = \begin{cases} 1, & \text{полностью видимый отрезок} \\ -1, & \text{полностью невидимый отрезок} \\ 0, & \text{частично видимый} \end{cases}$$

$O(x_{\text{Л}}, x_{\text{пр}}, y_{\text{Н}}, y_{\text{В}})$

1. Ввод множества  $O(x_{\text{Л}}, x_{\text{пр}}, y_{\text{Н}}, y_{\text{В}}), P_1, P_2$

2.  $Fl = -1$  предполагаем, что отрезок вертикален

3. Если  $P_2.x - P_1.x \neq 0$ , то вычисляем тангенс угла наклона  $m = \frac{P_2.y - P_1.y}{P_2.x - P_1.x}$ , если  $m = 0$ , то

$Fl = 1$ , иначе  $Fl = 0$

4. Цикл отсечения отрезка по всем границам отсечения (по  $i$  от 1 до 4)

4.1 Определение признака видимости  $wid$  и кодов концов  $T_1, T_2$  функцией( $O, P_1, P_2$ )

// На основе  $T_1, T_2, S_1, S_2, P$

4.2 Если  $wid = -1$ , то переход к п. 6, иначе если  $wid = 1$ , то переход к п. 5

4.3 Если  $T_{1i} = T_{2i}$ , то переход к п. 4.9н

4.4 Если  $T_{1i} = 0$ , то swap( $P_1, P_2$ )

4.5 Если  $Fl = -1$ , то переход к п. 4.8

4.6 Если  $i \leq 2$ , то  $P_1.y = m(O_i - P_1.x) + P_1.y; P_1.x = O_i$ , переход к п. 4.9

4.7  $P_1.x = \frac{1}{m}(O_i - P_1.y) + P_1.x$

4.8  $P_1.y = O_i$

4.9 Конец цикла

5. Визуализация отрезка  $P_1P_2$

6. Конец

## 14. Отсечение Алгоритм разбиения средней точкой при отсечении отрезка.

Если известно, что корень находится на отрезке ab, то делим пополам и смотрим на знак, и уменьшаем интервал поиска

$$P_{\text{cp}} = \frac{P_1 + P_2}{2} \quad |P_1 - P_2| \leq \epsilon$$

Этот подход оправдан, так как нахождение середины работает быстрее (деление на 2 это сдвиг). Алгоритм предусматривает вычисление кодов концевых точек отрезков объектов синтезируемой сцены и проверок, определяющих полную видимость отрезков (коды обоих концов отрезка равны 0), и полную невидимость отрезков (побитное логическое произведение кодов концевых точек не равно нулю). Те же отрезки, о которых нельзя судить столь однозначно, разбиваются 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. Затем определяется видимость вычисленной точки. Таким образом, процесс определения пересечения сводится к двоичному поиску, столь эффективно реализуемому аппаратно.

1. Ввод данных  $P_1, P_2, x_h, x_{\text{пр}}, y_h, y_b, \epsilon$
2.  $i = 1$
3. Вычисление кодов отрезка, их сумм ( $T_1, T_2, S_1, S_2$ )
4. Проверка полной видимости: если  $(S_1 = 0) \wedge (S_2 = 0)$ , то переход к п. 17
5. Вычисление произведения кодов концов отрезка, проверка полной невидимости: если  $p \neq 0$ , то переход к п. 18
6.  $T = P_1$
7. Если  $i > 2$ , то вычислить произведение кодов концов ( $P$ ); если  $p \neq 0$ , то переход к п. 18, иначе переход к п. 17
8. Если  $S_2 = 0$ , то  $Res_i = P_2; P_1 = P_2; P_2 = T; i = i + 1$ , переход к п. 3
9. Если  $|P_1 - P_2| \leq \epsilon$ , то  $Res_i = P_1; P_1 = P_2; P_2 = T; i = i + 1$ , переход к п. 3
10.  $P_{\text{cp}} = \frac{P_1 + P_2}{2}$
11.  $P_{\text{тек}} = P_1$
12.  $P_1 = P_{\text{cp}}$
13. Вычисление кодов для вершин  $P_1$
14. Вычисление логического произведения кодов концов отрезка  $P$
15. Если  $p \neq 0$ , то  $P_1 = P_{\text{тек}}, P_2 = P_{\text{cp}}$
16. Переход к п. 9
17. Визуализация отрезка  $P_1P_2$
18. Конец

## 15. Отсечение. Алгоритм Кируса-Бека отсечения отрезка.

### Определение выпуклости (невыпуклости) многоугольника

Рассматриваем стороны многоугольника как векторы и в каждой вершине вычисляем векторное произведение векторов смежных с этой вершиной, далее анализируем знаки полученных векторных произведений. В результате перемножения векторов получается вектор, «знак вектора» это знак проекции (оси Z).

**1 случай**, если знаки всех векторных произведений равны 0, то многоугольник вырожденный

**2 случай**, если знаки всех векторных произведений неположительные, то многоугольник выпуклый, а внутренние нормали ориентированы вправо от направления обхода

**3 случай**, если знаки всех векторных произведений положительные, то многоугольник выпуклый, а внутренние нормали ориентированы влево от направления обхода

**4 случай**, если среди знаков векторных произведений есть как положительные так и отрицательные, то многоугольник невыпуклый

$$P(t) = P_1 + (P_2 - P_1)t, \quad 0 \leq t \leq 1$$

$$\overrightarrow{n_{BH}} \cdot \overrightarrow{BA} = \begin{cases} > 0, & \text{то т. A видима} \\ < 0, & \text{то т. A невидима} \\ = 0, & \text{то т. A расположена на границе отсекателя} \end{cases}$$

$$\overrightarrow{n_{BH}} \cdot \overrightarrow{D} = \begin{cases} > 0, & \text{точка пересечения расположена ближе к началу} \\ < 0, & \text{точка пересечения расположена ближе к концу} \\ = 0, & \text{параллельны} \end{cases}$$

1. Ввод данных  $P_1, P_2, N, C(N)$

2.  $t_H = 0, t_B = 1, D = \overrightarrow{P_2 - P_1}$  - вектор ориентации

3. Цикл отсечения отрезка по всем границам отсекателя (по  $i$  от 1 до  $N$ )

3.1 Нахождение вектора внутренней нормали  $n_{BH_i}$

3.2 Нахождение вектора  $\vec{W}_i = \overrightarrow{P_1 - f_i}$  - от начала отрезка до граничной точки

3.3 Вычисление скалярного произведения  $D_{CK} = \overrightarrow{n_{BH_i}} \cdot \overrightarrow{D}; \vec{W}_{CK} = \vec{W}_i \cdot \overrightarrow{n_{BH_i}}$

3.4 Если  $D_{CK} = 0$ , то если  $W_{CK} \geq 0$ , то переход на п. 3.7, иначе переход к п. 5

3.5  $t = -\frac{W_{CK}}{D_{CK}}$

3.6 Если  $D_{CK} > 0$ , то если  $t > 1$ , то переход к п. 5, иначе  $t_H = \max(t_H, t)$ ,

иначе если  $t < 0$ , то переход к п. 5, иначе  $t_B = \min(t_B, t)$

3.7 Конец цикла

4. Если  $t_H \leq t_B$ , то изображение отрезка  $P(t_H) \div P(t_B)$

5. Конец

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

Удаление части изображения, находящейся за пределами заданной (выделенной) области, называется отсечением (**внутреннее отсечение**)

**Стирание** - удаление части изображения, находящейся в пределах заданной области (**внешнее отсечение**)

### **Определение выпуклости (невыпуклости) многоугольника**

Рассматриваем стороны многоугольника как векторы и в каждой вершине вычисляем векторное произведение векторов смежных с этой вершиной, далее анализируем знаки полученных векторных произведений. В результате перемножения векторов получается вектор, «знак вектора» это знак проекции.

**1 случай**, если знаки всех векторных произведений равны 0, то многоугольник вырожденный

**2 случай**, если знаки всех векторных произведений неположительные, то многоугольник выпуклый, а внутренние нормали ориентированы вправо от направления обхода

**3 случай**, если знаки всех векторных произведений положительные, то многоугольник выпуклый, а внутренние нормали ориентированы влево от направления обхода

**4 случай**, если среди знаков векторных произведений есть как положительные так и отрицательные, то многоугольник невыпуклый

### **Определение вектора внутренней нормали**

$$a_x \cdot n_x + a_y \cdot n_y = 0$$

Если нормали ориентированы вправо(направление обхода = -1):  $n_x = a_y, n_y = -a_x$

Если нормали ориентированы влево(направление обхода = 1):  $n_x = -a_y, n_y = a_x$

$$\vec{n} \cdot \overrightarrow{C_{i+j}C_i} = \begin{cases} > 0, \text{ то нормаль внутренняя} \\ < 0, \text{ то нормаль внешняя} \\ \text{умножить на -1 параметр нормали} \end{cases}$$

Вершины многоугольника пронумерованы против часовой стрелки.

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.

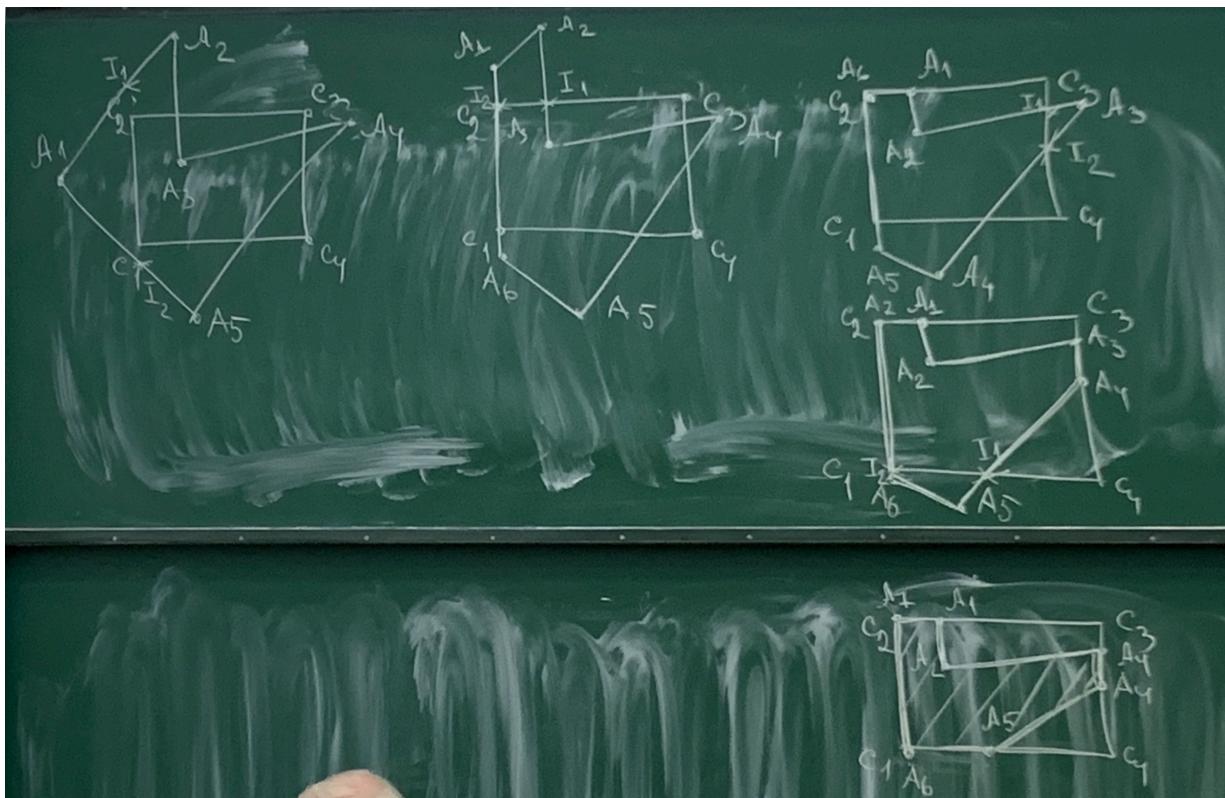
Если ординаты всех  $(i+2)$ -ых вершин равны нулю, то многоугольник вырожден, т.е. является отрезком.

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  $x_{i+1}$ . 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.

## 17. Отсечение многоугольников. Алгоритм Сазерленда-Ходжмена.

Случаи:

- 1) Полнотью невидимое ребро. В результат ничего не заносим (0 точек).
- 2) Ребро входит в область отсекателя (частично видимое). Найти пересечение ребра с границей отсекателя, в результат занести точку пересечения и конечную вершину (2 точки).
- 3) Ребро полностью видимо. В результат заносим обе вершины, но предыдущая точка была обработана и занесена на предыдущем шаге (ребре) (1 точка).
- 4) Ребро выходит из отсекателя наружу (частично видимое). Найти точку пересечения и занести ее в результат (начальная точка была занесена на предыдущем шаге) (1 точка).



### Алгоритм

1. Ввод данных  $N_A$ ,  $A(N_A)$ ,  $N_C$ ,  $C(N_C)$  (продублировать т.  $C_1$ , записав ее в конец,  $N_C = N_C + 1$ ) + проверка на выпуклость
2. Отсечение многоугольника по всем границам отсекателя (по  $i$  от 1 до  $N_C$ )

$$2.1 N_B = 0, B = 0$$

- 2.2 Цикл по всем вершинам (ребрам) отсекаемого многоугольника (по  $j$  от 1 до  $N_A$ )
  - 2.2.1 Если  $j = 1$ , то  $F = A_j$ , переход к п. 2.2.4
  - 2.2.2 Определение наличия пересечения ребра  $SA_j$  с прямой  $C_iC_{i+1}$   
( $pr = 1$ , если пересечение есть)
  - 2.2.3 Если  $pr = 1$ , то нахождение точки пересечения ребра  $SA_j$  и прямой  $C_iC_{i+1}$ .  
Занесение  $I$  (точки пересечения) в результат:  $N_B = N_B + 1$ ,  $B(N_B) = I$
  - 2.2.4  $S = A_j$
  - 2.2.5 Определение видимости точки  $S$  относительно прямой  $C_iC_{i+1}$   
( $wid = 1$ , если видима)
  - 2.2.6 Если  $wid = 1$ , то занесение ее в результат:  $N_B = N_B + 1$ ,  $B(N_B) = S$
  - 2.2.7 Конец цикла по  $j$

- 2.3 Если  $N_B = 0$ , то переход к п. 4 (многоугольник невидим относительно грани, значит и вовсе невидим)

- 2.4 Определение факта пересечения ребра  $SF$  с прямой  $C_iC_{i+1}$  ( $pe = 1$ , если пересечение есть)

2.5 Если  $pe = 1$ , то нахождение точки пересечения  $I$  и занесения в результат:

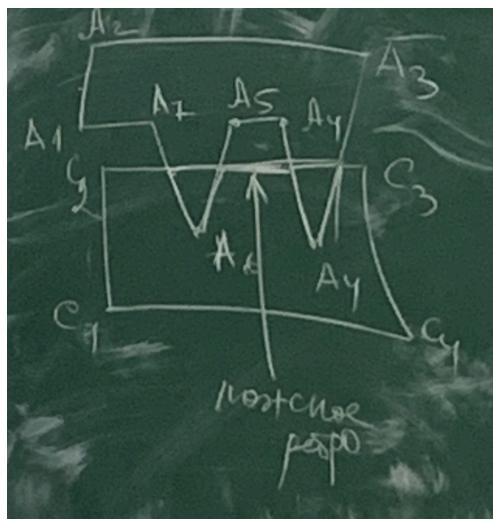
$$N_B = N_B + 1, \quad N(N_B) = I$$

$2.6 N_A = N_B$ ,  $A = B$  (копирование промежуточного результата в исходный массив)

## 2.7 Конец цикла по $i$

### 3. Визуализация многоугольника A

#### 4. Конец



Недостаток этого алгоритма заключается в том, что в определенных ситуациях появляются *ложные (лишние) ребра*. В том случае, когда в результате отсечения получается два или более многоугольника, ложные ребра и есть ребра, соединяющие эти многоугольники.

Когда мы рассматривали отсечение не рассматривали случай нескольких многоугольников, результат записывается в один массив, то есть многоугольники рассматриваются как одно целое и при визуализации соединяются последовательно.

**Ложное ребро** это участок ребра отсекателя. Оно рисуется дважды при обходе (первый при переходе от одного многоугольника к другому, второй при замыкании фигуры)

## **18. Отсечение многоугольников невыпуклыми областями. Алгоритм Вейлера-Азертони.**

Вейлер и Азертон предложили весьма мощный алгоритм, позволяющий отсекать невыпуклые многоугольники, содержащие внутренние отверстия, многоугольником невыпуклой формы, также содержащим внутренние отверстия. Позволяет находить как внутренние, так и внешние многоугольники

Границы многоугольников, получаемых в результате отсечения, **совпадают либо с границами исходного многоугольника, либо с участками границ отсекателя**. Никаких других новых границ не возникает.

Для реализации отсечения следует описать каждую границу отсекаемого многоугольника и отсекателя в виде **циклического списка** их вершин. При этом **внешние границы многоугольников должны обходиться по часовой стрелке, а внутренние границы - против часовой стрелки**.

Для нахождения внутренних многоугольников движение начинаем с очередной точки входа, причём просмотр начинается со списка отсекаемого многоугольника. Просматриваемые вершины заносятся в список вершин результирующего многоугольника. Чтобы находить внешние многоугольники, движение надо начинать с очередной точки выхода. Списки границ отсекателя надо просматривать в обратном направлении.

В ходе реализации алгоритма необходимо находить точки пересечения ребер отсекаемого многоугольника и отсекателя и точки пересечения надо разделить на две группы: точки входа и точки выхода.

**Точка входа** - точка пересечения, в которой ребро отсекаемого многоугольника входит внутрь отсекателя

**Точка выхода** - точка пересечения, в которой ребро отсекаемого многоугольника выходит из отсекателя наружу.

Реализуя вышеописанную последовательность действий, получим правильный результат только в том случае, если границы отсекаемого и отсекающего многоугольников пересекаются.

Если же нет пересечения границ, то необходимо установить соответствие между полученной границей (она будет внешней или внутренней) и границей самого многоугольника или отсекателя, которая будет являться результирующей внутренней или внешней границей.

Границы многоугольника, лежащие внутри отсекателя, будут являться границами результирующего внутреннего многоугольника. Границы, лежащие вне отсекателя, будут являться границами получаемых внешних многоугольников.

Границы отсекателя, попавшие внутрь многоугольника, образуют в нем отверстия. В итоге границы отсекающего многоугольника должны быть помещены в оба списка принадлежности: внутренний и внешний.

1. Ввод количества вершин внешней границы отсекаемого многоугольника и их координат.
2. Ввод количества отверстий в отсекаемом многоугольнике, по каждому отверстию - количества вершин и их координат.
3. Ввод количества вершин внешней границы отсекателя и их координат.
4. Ввод количества отверстий в отсекателе, по каждому отверстию - количества вершин и их координат.
5. Формирование кольцевых двунаправленных списков вершин по всем границам отсекаемого многоугольника.
6. Формирование кольцевых двунаправленных списков вершин по всем границам отсекателя.
7. Вычисление координат всех точек пересечения отсекаемого многоугольника и отсекателя.
8. Добавление всех найденных точек пересечения на соответствующие места в списки вершин многоугольников.

9. Определение типа каждой точки пересечения и формирование двух списков - точек входа и точек выхода.
10. Проведение собственно отсечения. Организация для этого цикла по всем точкам входа.
11. Нахождение в списке вершин отсекаемого многоугольника очередной точки входа.
12. Просмотр списка вершин отсекаемого многоугольника до нахождения точки пересечения.  
Копирование просмотренных вершин в список внутренней принадлежности.
13. Переход к списку вершин отсекателя и поиск в нем одноименной точки пересечения.
14. Просмотр списка вершин отсекателя до нахождения точки пересечения. Копирование просмотренных вершин в список внутренней принадлежности.
15. Сравнение найденной точки пересечения с первой точкой: если эти точки не совпадают, то переход к п. 13, иначе к п. 17.
16. Определение необходимости формирования второй границы отсеченного многоугольника: если не все границы отсекаемого многоугольника имеют пересечение с границами отсекателя, то необходим поиск другой границы, иначе переход к п. 19.
17. Конец цикла (выбор следующей точки входа из списка, если он не пуст и переход к п. 12, иначе выход из цикла).
18. Конец.

Следует отметить, что определение точек пересечения ребер отсекаемого многоугольника и отсекателя требует определенной аккуратности. В частности, надо четко различать собственно пересечение ребер от их касания. Случай, когда вершина или ребро отсекаемого многоугольника касаются ребра отсекателя (вершина отсекаемого многоугольника лежит на ребре отсекателя или ребро отсекаемого многоугольника совпадает со стороной отсекателя), не является пересечением. При реализации алгоритма следует проверять только такие точки пересечения, которые совпадают с вершинами ребер отсекаемого многоугольника.

Можно предложить следующий подход к определению характера точки пересечения. Следует проверить принадлежность точки, ближайшей к точке пересечения и расположенной по направлению вектора стороны отсекаемого многоугольника. Если эта точка принадлежит ребру (а не его продолжению), то ее следует считать точкой пересечения, в противном случае она является точкой касания.

## **19. Модели трехмерных объектов. Требования, предъявляемые к моделям.**

### **Модели объектов**

- \* Каркасная ("проволочная") (не дает представления о наличии отверстий)
- \* Поверхностная (не дает представления о том, с какой стороны материал, а с какой пустота)
- \* Объемная

### **Требования**

1. Модель не должна противоречить исходному объекту
2. Модель должна допускать возможность конструирования тела целиком
3. Модель должна допускать вычисление геометрических характеристик тела
4. Модель должна позволять проводить расчеты: кинематические, сопротивления

### **Свойства**

1. Однородность (тело заполнено изнутри)
2. Конечность (каждое тело занимает конечный объём)
3. Жёсткость (сплошное тело сохраняет форму независимо от положения)

### **Требования к ПО**

1. Согласованность операций (любые операции над сплошными телами должны приводить к сплошным телам)
2. Возможность описания (любое тело должно представляться в машинном виде)
3. Непротиворечивость информации (любая точка принадлежит 1 телу, для любой точки можно сказать, принадлежит ли она телу)
4. Компактность модели
5. Открытость модели (разрабатываемая модель должна иметь применении при работе с разными алгоритмами)

## 20. Операции преобразования в трехмерном пространстве. Матрицы преобразований.

Матрица преобразований в трехмерном пространстве в однородных координатах будет иметь размерность 4\*4. Координаты точки (X, Y, Z) заменяются четверткой (WX, WY, WZ, W), W ≠ 0.

Каждая точка пространства может быть задана четверткой одновременно не равных нулю чисел, эта четвертка определена однозначно до общего множителя. Такой подход позволяет в матричной форме описать операции преобразования.

Любое аффинное преобразование в трехмерном пространстве может быть представлено в виде суперпозиции переносов, масштабирований, поворотов.

### Перенос

$$x_1 = x + dx \quad M_{\text{пер}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{pmatrix}$$

$$y_1 = y + dy$$

$$z_1 = z + dz$$

### Масштабирование

$$x_1 = k_x \cdot x + (1 - k_x) \cdot x_M \quad M_{\text{масшт}} = \begin{pmatrix} k_x & 0 & 0 & 0 \\ 0 & k_y & 0 & 0 \\ 0 & 0 & k_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$y_1 = k_y \cdot y + (1 - k_y) \cdot y_M$$

$$z_1 = k_z \cdot z + (1 - k_z) \cdot z_M$$

### Поворот

$$M_{\text{повор}_x} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad x_1 = x$$

$$y_1 = y_c + (y - y_c)\cos(\theta) - (z - z_c)\sin(\theta)$$

$$z_1 = z_c + (z - z_c)\cos(\theta) + (y - y_c)\sin(\theta)$$

$$M_{\text{повор}_y} = \begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad x_1 = x_c + (x - x_c)\cos(\theta) + (z - z_c)\sin(\theta)$$

$$y_1 = y$$

$$z_1 = z_c + (z - z_c)\cos(\theta) - (x - x_c)\sin(\theta)$$

$$M_{\text{повор}_z} = \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 & 0 \\ -\sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad x_1 = x_c + (x - x_c)\cos(\theta) - (y - y_c)\sin(\theta)$$

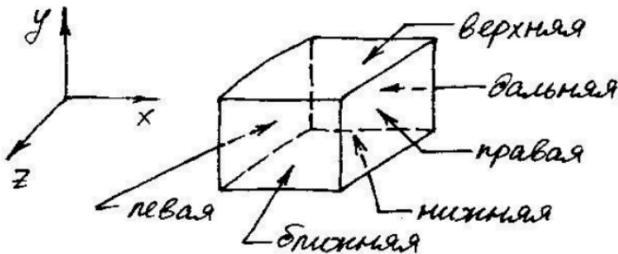
$$y_1 = y_c + (y - y_c)\cos(\theta) + (x - x_c)\sin(\theta)$$

$$z_1 = z$$

## 21. Трехмерное отсечение. Виды отсекателей. Вычисление кодов концов отрезка для каждого типа отсекателей. Алгоритм отсечения отрезков средней точкой.

### Виды отсекателей

#### 1. Прямоугольный параллелепипед

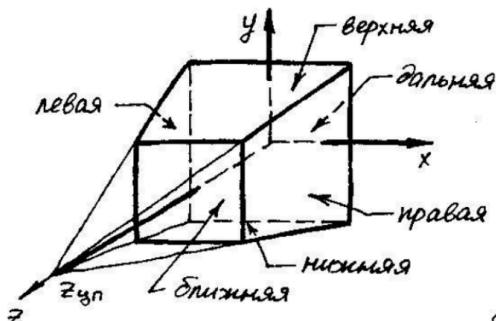


Как и в двумерном отсечении, здесь используется битовый код видимости вершины (обобщенный).  
 $T_{i4} = 1$ , если  $Z > Z_6$ , иначе  $T_{i4} = 0$   
 $T_{i5} = 1$ , если  $Z < Z_D$ , иначе  $T_{i5} = 0$

Трёхмерный алгоритм аналогичен двумерному, но к координатам добавляется третья компонента.  
Код видимости задаётся 6 битами.

Если отсекатель – усечённая пирамида, то для составления кода используются пробные функции

#### 2. Усеченная пирамида видимости



$Z_{\text{ CPP }}$  - центр проекции

Уравнение плоскости, проходящей через правую грань

$$x = \frac{(z - z_{\text{ CPP }}) \cdot x_{\text{ пр }}}{z_D - z_{\text{ CPP }}} \text{, где } x = z\alpha_1 + \alpha_2$$

$$\alpha_1 = \frac{x_{\text{ пр }}}{z_D - z_{\text{ CPP }}} \text{ и } \alpha_2 = -\alpha_1 \cdot z_{\text{ пр }}$$

$$f_{\Pi} = x - z\alpha_1 - \alpha_2 \begin{cases} > 0, & \text{если точка справа от плоскости} \\ = 0, & \text{если точка на плоскости} \\ < 0, & \text{если точка слева от плоскости} \end{cases}$$

$$f_{\Pi} = x - z\beta_1 - \beta_2 \begin{cases} > 0, & \text{если точка справа от плоскости} \\ = 0, & \text{если точка на плоскости} \\ < 0, & \text{если точка слева от плоскости} \end{cases}$$

$$\beta_1 = \frac{x_{\Pi}}{z_D - z_{\text{ CPP }}} \text{ и } \beta_2 = -\beta_1 \cdot z_{\text{ пр }}$$

$$f_B = y - z\gamma_1 - \gamma_2 \begin{cases} > 0, & \text{если точка выше плоскости} \\ = 0, & \text{если точка на плоскости} \\ < 0, & \text{если точка ниже плоскости} \end{cases}$$

$$\gamma_1 = \frac{y_B}{z_D - z_{\text{ CPP }}} \text{ и } \gamma_2 = -\gamma_1 \cdot z_{\text{ CPP }}$$

$$f_H = y - z\delta_1 - \delta_2 \begin{cases} > 0, & \text{если точка выше плоскости} \\ = 0, & \text{если точка на плоскости} \\ < 0, & \text{если точка ниже плоскости} \end{cases}$$

$$\delta_1 = \frac{y_H}{z_D - z_{\text{ CPP }}} \text{ и } \delta_2 = -\delta_1 \cdot z_{\text{ CPP }}$$

$$f_6 = z - z_6 \begin{cases} > 0, \text{ если точка ближе плоскости} \\ = 0, \text{ если точка на плоскости} \\ < 0, \text{ если точка дальше плоскости} \end{cases}$$

$$f_D = z - z_D \begin{cases} > 0, \text{ если точка ближе плоскости} \\ = 0, \text{ если точка на плоскости} \\ < 0, \text{ если точка дальше плоскости} \end{cases}$$

## **22. Отсечение отрезков в трехмерном пространстве. Трехмерный алгоритм Кируса-Бека.**

Также легко обобщается на трехмерный случай и алгоритм Кируса-Бека.

- Отсекатель в этом случае может представлять собой произвольный выпуклый многогранник.
- Координаты точек и векторов должны иметь три компоненты в соответствии с количеством измерений.
- В качестве точек, лежащих на грани отсекателя - параллелепипеда, удобно выбирать точки, лежащие на концах главной диагонали. В качестве пробных точек также удобно выбирать вершины пирамиды, в которых сходятся три грани пирамиды.
- Для отсекателя - параллелепипеда легко определяются и внутренние нормали. Не вызывает больших трудностей выбор пробных точек и вычисление нормалей и в случае с усеченной пирамидой. Определение нормалей к ближней и дальней граням очевидно, а определение нормалей к остальным четырем граням возможно на основании вычисления векторного произведения векторов, построенных на ребрах граней, начинающихся в одной из вершин грани.
- Понятие «граница (или сторона, или ребро) отсекающего многоугольника», используемое в двумерном алгоритме, в трехмерном алгоритме трансформируется в понятие «граница (или грань) отсекающего многогранника».
- Бесконечные прямые, проведенные через ребра многоугольника, заменяются бесконечными плоскостями, несущими грани многогранника.
- Подпрограмма расчета скалярного произведения векторов реализуется по аналогии с тем, как это сделано в двумерном алгоритме, но векторы задаются тремя компонентами (по осям x, y и z).

## **23. Определение факта выпуклости трехмерных тел. Разбиение тела на выпуклые многогранники.**

1. Перенести тело таким образом, чтобы одна из вершин грани оказалась в начале координат.
2. Повернуть тело вокруг начала координат так, чтобы одно из двух смежных выбранной вершине ребер грани совпало с одной из координатных осей.
3. Повернуть тело вокруг выбранной оси координат так, чтобы выбранная грань легла на координатную плоскость.
4. Для всех вершин тела, не принадлежащих выбранной грани, определить знаки координаты, которая перпендикулярна этой грани.
5. Провести анализ полученных знаков, руководствуясь следующим правилом:
  - \* если знаки для всех вершин совпадают или равны нулю, то тело является выпуклым относительно выбранной грани; если тело выпукло относительно всех своих граней, то оно является в целом выпуклым, в противном случае оно невыпукло;
  - \* если знаки для всех вершин не совпадают, то тело невыпукло относительно очередной грани, следовательно, оно невыпукло в целом;
  - \* если для всех вершин значения координаты, перпендикулярной выбранной грани, равны нулю, то тело вырождено, т.е. это плоский многоугольник.
6. Вектор внутренней нормали к выбранной грани, заданный в преобразованной системе координат, имеет все нулевые компоненты, кроме той, которая перпендикулярна рассматриваемой грани. Знак этой компоненты для грани, относительно которой тело выпукло, совпадает с ранее найденным в п.4 знаком.  
Для определения значения внутренней нормали в исходной системе координат необходимо применить к ней обратное преобразование поворотов.
7. Если тело оказалось невыпуклым относительно рассматриваемой грани, то его следует разрезать плоскостью, несущей выбранную грань.
8. Для каждого из вновь полученных тел повторить описанную процедуру до тех пор, пока все тела не станут выпуклыми

## 24. Алгоритм плавающего горизонта.

Алгоритм плавающего горизонта чаще всего используется для удаления невидимых линий трехмерного представления функций, описывающих поверхность в виде  $F(x, y, z) = 0$ .

Главная идея алгоритма заключается в сведении трехмерной задачи к двумерной путем пересечения исходной поверхности последовательностью параллельных секущих плоскостей, имеющих постоянные значения координат  $x, y$  или  $z$ .

**Для каждой секущей плоскости  $z = const$  выполнить следующие действия:**

1. Обработать левое боковое ребро. Если очередная точка кривой является первой и кривая первая, то запомнить эту точку, если очередная точка является первой, а кривая не первая, то соединить точку  $P$  с точкой  $S$ . Запомнить точку  $P$  в качестве точки  $S$ .

2. Для каждой точки, расположенной на кривой, полученной в текущей секущей плоскости выполнить следующие операции:

2.1 Определить видимость точки: точка видима, если она расположена выше верхнего горизонта ( $y_{тек}(x_i) > y_{max}(x_i)$ ) или ниже нижнего горизонта ( $y_{тек}(x_i) < y_{min}(x_i)$ )

2.2 Проверить изменение видимости текущей кривой на рассматриваемом интервале.

В случае изменения видимости кривой найти точку пересечения кривой с соответствующим горизонтом. Точка пересечения ищется как пересечение двух отрезков

2.3 Изобразить полностью участок (сегмент) текущей кривой, если очередная точка видима и видимость изменилась

2.4 Если очередная точка невидима и видимость не изменилась, то ничего не изображается

2.5 Если текущая точка видима и видимость изменилась, то изобразить участок кривой от точки пересечения до текущей точки

2.6 Если текущая точка невидима и видимость изменилась, то изобразить участок кривой от предыдущей точки до точки пересечения

2.7 Обработать правое боковое ребро по аналогии. Изменить в случае необходимости (если точка видима) значение одного из горизонтов.

Может быть неправильно выбран шаг дискретизации

Для хранения максимальных значений у при каждом значении  $x$  используется массив, длина которого равна числу различных точек (разрешению) по оси  $x$  в пространстве изображения. Значения, хранящиеся в этом массиве, представляют собой текущие значения "горизонта".

Поэтому по мере рисования каждой очередной кривой этот горизонт "всплывает". Фактически этот алгоритм удаления невидимых линий работает каждый раз с одной линией.

Алгоритм работает очень хорошо до тех пор, пока какая-нибудь очередная кривая не окажется ниже самой первой из кривых. Нижняя сторона поверхности делается видимой, если модифицировать этот алгоритм, включив в него нижний горизонт, который опускается вниз по ходу работы алгоритма. Это реализуется при помощи второго массива, длина которого равна числу различных точек по оси  $x$  в пространстве изображения. Этот массив содержит наименьшие значения  $u$  для каждого значения  $x$ .

## **25. Задача удаления невидимых линий и поверхностей. Ее значение в машинной графике. Классификация алгоритмов по способу выбора системы координат (объектное пространство, пространство изображений).**

Задача удаления невидимых линий и поверхностей является одной из наиболее сложных в машинной графике. Осложняется тем, что необходимо учитывать положение наблюдателя. Алгоритмы удаления невидимых линий и поверхностей служат для определения линий ребер, поверхностей или объемов, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства.

Единого и наилучшего способа решения задачи нет. Возникло множество алгоритмов, многие ориентированы на специализированные приложения. Все алгоритмы включают в себя сортировку. Главная – сортировка по расстоянию от точки наблюдения до тела-поверхности-ребра-точки. Так же есть сортировка по загораживанию тел друг другом. Эффективность алгоритмов в большой степени зависит от эффективности алгоритма сортировки.

Существует связь между скоростью работы алгоритма и детальностью его результата – ни один алгоритм не может достигнуть хороших показателей одновременно. Алгоритмы делятся на:

1. Работающие в объектном пространстве (мировая ск, высокая точность). Сложность  $\sim N^2$
2. Работающие в пространстве изображений (ск связана с дисплеем, точность ограничена разрешающей способностью дисплея). Сложность  $\sim N^*M$

$N$  – количество объектов,  $M$  – количество пикселей.  $N^2 < N^*M$ , но алгоритмы, работающие в пространстве изображений могут быть эффективнее в силу свойства когерентности при растровой реализации (близко расположенные пиксели чаще обладают одинаковыми свойствами).

## 26. Алгоритм Робертса. Основные этапы и математические основы каждого этапа.

1. Удаление невидимых линий

2. Работает только с выпуклыми телами

Сортировка тел по мере их удалённости от наблюдателя. Сложность  $\sim N^2$

### 0 Этап подготовки исходных данных

- Определение коэффициентов уравнения плоскости каждой грани, проверка правильности знака уравнения и формирование матрицы объекта визуализации.
- Проведение видового преобразования матрицы объекта, вычисление прямоугольной охватывающей оболочки объекта.

1 Этап удаления невидимых ребер, экранируемых самим телом (если тело единственное, то работа завершается)

- Определение нелицевых граней, удаление их из списка граней и соответствующих ребер - из списка ребер.
- Проверка видимости полученных отрезков по отношению ко всем объектам сцены в соответствии с этапами

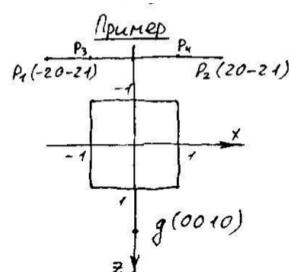
2 Этап удаления невидимых ребер, экранируемых другими телами сцены

- Определение списка других объектов синтезируемой сцены, которые могут быть экранированы данным объектом визуализации на основании сравнений охватывающих оболочек объектов.

3 Этап удаления невидимых ребер, получаемых в случае взаимного проникновения тела (в этом случае появляются новые ребра)

- Формирование списка проникновений на основании сравнений охватывающих оболочек объектов.
- Определение невидимых отрезков или участков отрезков.
- Проверка видимости полученных отрезков по отношению ко всем объектам сцены в соответствии с этапами
- Формирование списка возможных отрезков, соединяющих точки проникновения, для пар объектов, связанных отношением проникновения.

4 Этап визуализация изображения.



$$[V] = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$\bar{d} = (4000); \quad S = (-20-21)$$

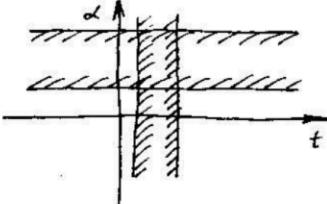
$$P = \bar{S} \cdot [V] = (-1311-13)$$

$$w = \bar{g} \cdot [V] = (00001-1)$$

$$q = \bar{d} \cdot [V] = (4-40000)$$

$$h_j = p_j + t q_j + \alpha w_j > 0 \Rightarrow$$

$$\Rightarrow \begin{cases} -1 + 4t > 0 \\ 3 - 4t > 0 \\ 1 > 0 \\ 1 > 0 \\ -1 + \alpha > 0 \\ 3 - \alpha > 0 \end{cases}$$



$$P_3 = P\left(\frac{1}{4}\right) = (-20-21) + (4000)\frac{1}{4} = (-10-21)$$

$$P_4 = P\left(\frac{3}{4}\right) = (-20-21) + (4000)\frac{3}{4} = (10-21)$$

$P_3 P_4$  - невидимая часть.

## 27. Алгоритм Робертса. Формирование матрицы тела. Удаление нелицевых граней.

### Формирование матрицы тела

Для каждого тела(представляет собой набор многогранников) сформировать матрицу

$$[V] = \begin{bmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \\ c_1 & c_2 & \dots & c_n \\ d_1 & d_2 & \dots & d_n \end{bmatrix} 4 \times n - \text{кол-во граней}$$

Каждая плоскость описывается уравнением:

$$ax + by + cz + d = 0$$

Матрица тела должна быть сформирована таким образом, чтобы каждая точка тела располагалась по положительную сторону от грани тела. Если это не так, то соответствующий столбец матрицы умножить на  $-1$

$$[P] = [S][V], p_j > 0, j = \overline{1, n}$$

$$[P] = [a \ b \ c \ d] \quad [S] = [x \ y \ z \ 1] - \text{координаты точки}$$

Уравнение нормали для плоскости:

$$\vec{n} = a\vec{i} + b\vec{j} + c\vec{k}$$

$[T]$  – матрица преобразований

$[B]$  – матрица координат вершин тела

$[BT]$  – матрица координат вершин преобразованного тела

$[VT]$  – матрица преобразований тела

$$[B][V] = [D]$$

$$[BT][VT] = [D]$$

$$[B][V] = [BT][VT]$$

$$[V] = [T][VT]$$

$$[T]^{-1}[V] = [VT]$$

### Удаление нелицевых граней

Для определения невидимых граней тела надо вектор взгляда умножить на матрицу тела. На полученном векторе надо найти отрицательные компоненты, они соответствуют невидимым граням тела. Найдя невидимые грани мы найдем невидимые ребра.

$[E] = [0 \ 0 \ -1 \ 0]$  - вектор наблюдения. Предполагается, что наблюдатель находится в бесконечности на положительной полуоси  $Z$  и смотрит в начало координат.

## 28. Алгоритм Робертса. Удаление отрезков, экранируемых другими телами.

Отсортировать все тела по максимальному z тела. Наиболее удалённое тело – более экранируемое.  
 $g = [0 \ 0 \ 1 \ 0]$  - вектор наблюдения

Отрезок:

$$P(t) = P_1 + (P_2 - P_1)t, 0 \leq t \leq 1 \quad \vec{d} = \overrightarrow{P_2 - P_1}$$

Существуют ли значения  $t$ , при которых данный отрезок невидим?

Для этого соединим произвольную точку  $t$ , заданную текущим значением параметра, с точкой наблюдения. Если эта прямая не пересекает ни одного многогранника, ни одного объекта, то точка видима. Если же прямая пересекает какой-либо многогранник, то исследуемая точка экранируется этим многогранником. Прямую от точки  $P(t)$  до точки наблюдения представим также в параметрической форме:

$Q(\alpha, t) = P_1 + (P_2 - P_1)t + \alpha g, \alpha \geq 0$ , так как объекты, экранирующие  $P(t)$  могут находиться, только в той части данной плоскости, которая заключена между исследуемым отрезком  $P(t)$  и точкой наблюдения.

Скалярное произведение любой точки, расположенной внутри объекта и матрицы объекта, положительно. Точка, находящаяся внутри объекта, невидима. Следовательно, для проверки на экранирование вектор текущей точки отрезка умножают поочередно на матрицу каждого объекта и определяют положительное решение, соответствующее прохождению отрезка внутри объекта:

$$[H] = [Q][V] \quad [P] = [P_1][V] \quad [Q] = [d][V] \quad [W] = [g][V]$$

Значения  $t$  и  $\alpha$ , для которых все значения вектора  $H$  положительны, соответствуют невидимой части отрезка.

Тогда:

$h_j = p_j + q_j t + \alpha w_j > 0, j = \overline{1, n}$  где  $j$  - номер столбца в матрице объекта;  $w$  - вектор третьей строки матрицы объекта.

Последние условия должны выполняться для всех плоскостей, ограничивающих объект, т.е. для всех значений  $j$ . Граница между видимой и невидимой частями отрезка определяется условием  $H_j = 0$ .

Находят все значения  $t$  и  $\alpha$ , при которых изменяется значение видимости отрезка или части отрезка (число возможных решений при  $n$  плоскостях равно  $\frac{n(n-1)}{2}$ ). Затем каждое решение подставляем во все остальные неравенства системы.

Так определяют минимальное значение параметра  $t$  среди максимальных  $t_{min_{max}}$  и максимальное среди минимальных  $t_{max_{min}}$ .

Отрезок невидим:  $t_{max_{min}} < t < t_{min_{max}}$

Затруднительно определить полностью невидимые ребра, но можно простым способом определить полностью видимые. Обе вершины должны располагаться перед гранью тела или на грани и эта грань должна быть видима:  $wj \leq 0, pj \leq 0, pj + qj \leq 0$ .

**В случае прорыва** объектов сцены ищутся решения на границе  $\alpha = 0$ , и вычисляют отрезки, которые образуются при прорывании объектами друг друга. Эти отрезки проверяют на экранирование всеми прочими объектами сцены. Видимые отрезки образуют структуру прорыва.

## **29. Удаление невидимых линий и поверхностей в пространстве изображений. Алгоритм Варнока (разбиение окнами): последовательность действий и основные принципы.**

Мы должны дать ответ на вопрос что нужно изобразить на экране, есть окно для вывода изображения, делим окно на подокна. Основная идея состоит в том, что рассматривается окно. Если его очевидно нарисовать, то рисуем, иначе делим на подокна.

Одной единой версии этого алгоритма не существует.

Выделяют простейшую версию алгоритма и более сложную.

**Простейший алгоритм:** окно делится на подокна всякий раз, если это окно не пусто. Окно является пустым, если все многоугольники сцены являются внешними по отношению к этому окну. Пределом деления окна является окно размером с один пиксель.

В более сложных версиях, мы пытаемся для окон большего размера выбрать что отрисовывать.

### 1 случай

Все многоугольники внешние → окно закрасить цветом фона

(тест лучом для определения внешних многоугольников, четное количество пересечений)

### 2 случай

Внутренние многоугольники

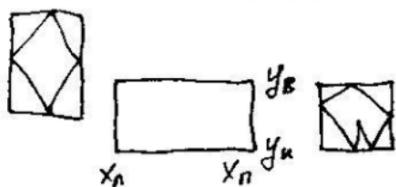
$(x_{\min} \geq x_{\Pi}) \text{ and } (x_{\max} \leq x_{\text{пр}}) \text{ and } (y_{\min} \leq y_{\Pi}) \text{ and } (y_{\max} \geq y_{\text{в}})$

Один внутренний многоугольник

1. Закрасить окно цветом фона
2. Выполнить растровую развертку единственного многоугольника

### 3 случай

Внешние многоугольники



$(x_{\max} < x_{\Pi}) \text{ or } (x_{\min} > x_{\Pi}) \text{ or } (y_{\max} < y_{\Pi}) \text{ or } (y_{\min} > y_{\Pi})$

Если все многоугольники внешние, то окно закрашивается цветом фона

### 4 случай

Пересекающие многоугольники

$$f_{\text{пр}} \phi = ax + by + cz + d$$

Если для всех вершин окна знак пробной функции одинаковый, то эта сторона многоугольника пересекать окно не может

- 1) Выполнить алгоритм отсечения многоугольника по границам окна
- 2) Закрасить окно цветом фона
- 3) Выполнить растровую развертку внутреннего многоугольника (результата отсечения)

### 5 случай

Охватывающие многоугольники

Окно надо закрасить цветом охватывающего многоугольника

### 6 случай

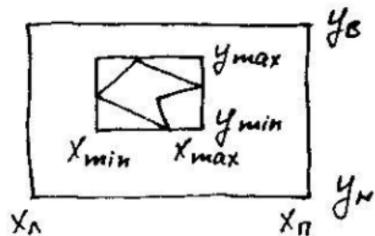
С окном связано несколько многоугольников разного типа, но есть один охватывающий, расположенный ближе других к наблюдателю, то закрашиваем все цветом охватывающего многоугольника

#### Рекомендуемая последовательность

1. Проводится простейший габаритный тест с прямоугольной оболочкой, определяется как можно большее количество пустых окон и окон с единственным внутренним многоугольником
2. Выполнение теста с целью определения окон, пересекаемых единственным многоугольником
3. Тест с целью распознавания внешних и охватывающих многоугольников. Получаем новые пустые окна и окна, охватываемые пустым многоугольником
4. Можно проводить разбиение окна на подокна или проводить ещё тест на обнаружение охватывающего многоугольника, лежащего ближе к наблюдателю

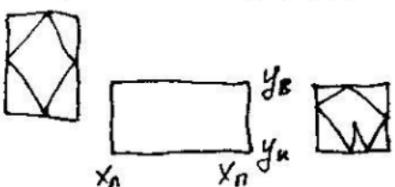
## 30. Типы многоугольников, анализируемых в алгоритме Варнока. Методы их идентификации.

### Внутренние многоугольники



$(x_{\min} \geq x_{\Pi}) \text{ and } (x_{\max} \leq x_{\text{пр}}) \text{ and } (y_{\min} \leq y_H) \text{ and } (y_{\max} \geq y_B)$  - многоугольник полностью виден

### Внешние многоугольник

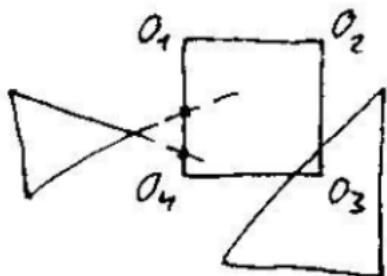


$(x_{\max} < x_{\Pi}) \text{ or } (x_{\min} > x_{\Pi}) \text{ or } (y_{\max} < y_H) \text{ or } (y_{\min} > y_B)$  - многоугольник невидим  
Но такой тест не распознает случай, когда многоугольник "обтекает" вершину

### Пересекающие многоугольники

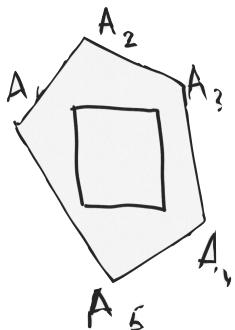
Составить уравнения для каждой прямой, проходящей через ребро многоугольника:

$$f_{\text{пр}} \phi = Ax + By + Cz + D$$



Если для всех вершин окна знак пробной функции одинаковый, то эта сторона многоугольника пересекать окно не может. Если знаки различаются, то нужно проверить наличие точек пересечения прямой с окном, если такие есть, то многоугольник пересекает окно.  
Если ни одно из ребер не пересекает окно, то он либо внешний, либо охватывающий.

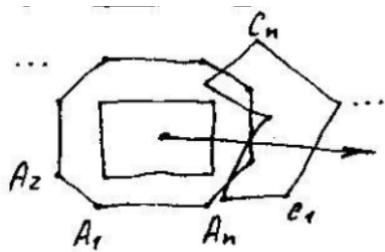
### Охватывающий многоугольник



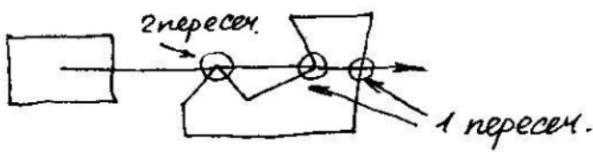
### Тест с бесконечным лучом (внешний или охватывающий)

Кол-во пересечений луча с ребрами четное  $\Rightarrow$  многоугольник внешний

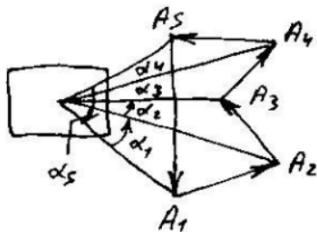
Кол-во пересечений луча с ребрами нечетное  $\Rightarrow$  многоугольник охватывающий



Частный случай:



Тест на вычисление угла (внешний или охватывающий)



$$\sum_{i=1}^n \alpha_i = 0 \Rightarrow \text{многоугольник внешний}$$

$$\sum_{i=1}^n \alpha_i = \pm 360^\circ \cdot m \Rightarrow \text{многоугольник охватывающий},$$

m - кол-во обхватов

Сумму считать с учетом направления

### **31. Алгоритм Вейлера-Азертона удаления невидимых линий и поверхностей.**

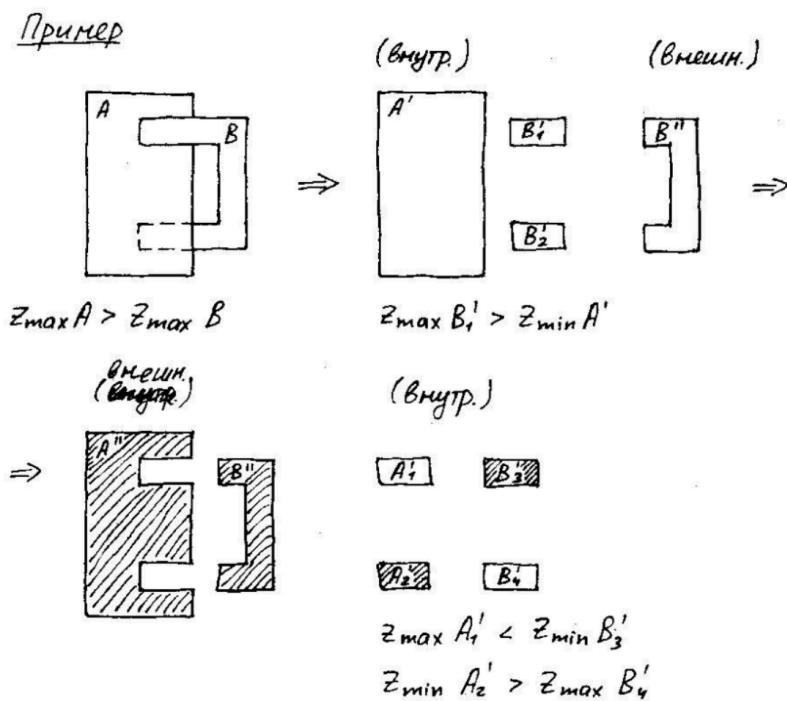
Алгоритм работает в объектном пространстве. В алгоритме Варнока необходимо выполнить много делений окна на подокна. В качестве отсекателя в этом алгоритме будет выбираться один из многоугольников сцены.

Алгоритм решения задачи удаления целиком базируется на алгоритме отсечения тех же авторов. Работа будет вестись с проекциями многоугольников. Алгоритм отсечения позволяет найти как внутренние так и внешние отсечения, что важно.

- Сортировка многоугольников по глубине ( $z_{\max}$ )
  - Отсечение многоугольников по границам ближайшего к наблюдателю многоугольника.  
Отсекаться будут все многоугольники в этом списке, включая первый. Формируется 2 списка – внутренний (для каждого отсекаемого многоугольника то часть, которая оказывается внутри отсекателя) и внешний (оставшаяся часть))
  - Удаление многоугольников, экранируемых ближайшим к наблюдателю многоугольником
  - Устранение всех неопределенностей с помощью рекурсивного подразбиения многоугольников.  
После образования списка внутренних и списках внешних многоугольников, в списке внутренних проверяем, чтобы  $z_{\min}$  Отсекаемого (для всех вершин) был больше  $z_{\max}$  отсеченного, если это не так, то меняем их роли (отсекатель  $\rightarrow$  отсекаемый, отсекаемый  $\rightarrow$  отсекатель) и продолжаем проверку.

Этот алгоритм справляется с циклическим перекрыванием многоугольников (часть многоугольника лежит позади другого, а часть спереди)

Если многоугольники прорываются, то придется один из многоугольников разбить на два линией пересечения этих многоугольников



## 32. Алгоритм, использующий Z-буфер.

Для него требуется буфер кадра, в котором запоминаются значения яркости, а также Z-буфер (буфер глубины), куда можно помещать информацию о координате z для каждого пикселя. Вначале в Z-буфер заносятся минимально возможные значения z, а буфер кадра заполняется значениями пикселя, описывающими фон. Затем каждый многоугольник преобразуется в растровую форму и записывается в буфер кадра, при этом, однако, не производится начального упорядочения.

Точки  $Z(x, y) > Z_{\max}(x, y)$ , смотрим точки пересечения, ищем ближайший многоугольник

1. Всем элементам буфера кадра присвоить фоновое значение
2. Инициализировать Z-буфер минимальным значением глубин
3. Выполнить растровую развертку каждого многоугольника сцены
  - 3.1 Для каждого пикселя, связанного с многоугольником, вычислить его глубину  $z(x, y)$
  - 3.2 Сравнить глубину многоугольника со значением, хранимы в Z-буфере  
если  $z(x, y) > z_{\text{буф}}$ , то  $z_{\text{буф}}(x, y) = z(x, y)$
  - Цвет( $x, y$ ) = Цвет текущего многоугольника (заносим в буфер кадра)
4. Результат (Содержимое буфера-кадра) вывести на экран

+ Простота, сцена может быть любой сложности  
+ Элементы сцены не нужно сортировать

- Раньше этот алгоритм не использовался из-за недостатка памяти  
- Трудоемкость устранения лестничного эффекта  
- Трудоемкость реализации эффектов прозрачности

$Ax + By + Cz + D = 0$  - уравнение плоскости

$z = -\frac{Ax + By + D}{C}$ , при  $C \neq 0$ , если  $C = 0$ , то плоскость многоугольника параллельна оси Z, что для наблюдателя вырождается в прямую.

### **33. Алгоритм, использующий список приоритетов.**

**Алгоритм использующий список приоритетов (алгоритм художника)**

(т. е. с самых дальних начинаний)

В основе алгоритма способ изображения сцены от дальних объектов к ближним.

Прежде чем изображать текущий надо ответить на вопрос будет ли он экранировать следующие.

- 1) Сортировка многоугольников по глубине  $z_{min}$ (по возрастанию)
- 2) Построить самый дальний многоугольник А, если он не экранирует другие другие многоугольники ( $z_{max}(A) < z_{min}(B)$ )
- 3) Проверить, экранирует ли многоугольник А многоугольник В при  $z_{max}(A) > z_{min}(B)$ . Если на один из тестов даётся положительный ответ, А надо рисовать(А не экранирует В). Если все отрицательные, то А и В меняются местами, позиция В приоритетнее, тесты повторяются снова. Если вновь все отрицательные(многоугольники циклически перекрываются), то А разрезается плоскостью, несущей В, исходный многоугольник удаляется из списка, а его части заносятся в список. Тесты повторяются снова.
  - 1) Верно ли, что прямоугольники объемлющие оболочки прямоугольников А и В не перекрываются по X? по Y?
  - 2) Верно ли , что многоугольник А целиком расположен по ту сторону плоскости, несущей многоугольник В, которая находится дальше от наблюдателя?
$$Ax + By + Cz + D = 0$$
$$f_{\text{пр } \phi} = Ax + By + Cz + D$$
A, B, C, D - коэффициенты уравнения плоскости, несущей многоугольник А. Координаты каждой вершины многоугольника В подставляются в пробную функцию. Если знаки пробной функции для всех вершин совпадают, то В целиком лежит по одну сторону от А. Для того, чтобы определить по какую сторону лежит многоугольник, необходимо сравнить получившийся знак со знаком пробной функции точки, положение которой уже известно
  - 3) Верно ли, что многоугольник В целиком расположен по ту сторону плоскости несущей многоугольник А, которая ближе расположена к наблюдателю?
  - 4) Верно ли, что проекции многоугольников А и В не пересекаются?

Для пункта 5 можно использовать алгоритм Варнока

## **34. Алгоритм построчного сканирования, использующий Z-буфер. Интервальные методы построчного сканирования (основные предпосылки).**

Алгоритмы Варнока, Z-буфера и строящего список приоритетов обрабатывают элементы сцены в порядке, который не связан с процессом визуализации. Алгоритмы построчного сканирования обрабатывают сцену в порядке прохождения сканирующей строки.

По сравнению с обычным алгоритмом, алгоритм построчного сканирования использует значительно меньше памяти(ширину экрана для каждого буфера, а не ширину \* высоту).

### **Алгоритм**

#### **1. Подготовка информации**

- a) Для каждого многоугольника определить самую верхнюю сканирующую строку, которую он пересекает.
- b) Занести многоугольник в группу у, соответствующую этой сканирующей строке
- c) Запомнить для каждого многоугольника: Ду – число строк, пересекающих этот многоугольник, список рёбер многоугольника, коэффициенты А, В, С, Д уравнения плоскости многоугольника, визуальные атрибуты многоугольника

#### **2. Решение задачи удаления невидимых поверхностей**

- a) Инициализировать буфер кадра дисплея
- b) Для каждой сканирующей строки

- Инициализировать буфер кадра размером с одну сканирующую строку, заполнив его фоновым цветом
- Инициализировать Z-буфер размером с одну сканирующую строку значением Zmin
- Проверить необходимость добавления в список активных многоугольников (САМ) новых многоугольников (текущий у <= Y группы)
- Если было добавление многоугольника в САМ, то добавить в САР соответствующие рёбра новых многоугольников
- Если произведено удаление какого-либо элемента из пары рёбер САР, то проверить необходимость удаления всего многоугольника из САМ. Если он остаётся, то проверить необходимость удаления другого ребра из этой пары – если его удалять не нужно, то доукомплектовать пару (добавив недостающее левое или правое ребро)

- c) В САР должна храниться следующая информация:

- 1) Хл – пересечение левого ребра с текущей сканирующей строкой
- 2) ΔХл – приращение Хл в интервале между соседними сканирующими строками
- 3) ΔYл – число сканирующих строк, пересекаемых левым ребром
- 4) Xп, ΔXп, ΔYп
- 5) ΔZx=-A/C для C≠0 (иначе ΔZx=0) – приращение по Z вдоль сканирующей строке
- 6) ΔZy=-B/C для C≠0 (иначе ΔZy=0) – приращение по Z в интервале между соседними сканирующими строками

- d) Для каждой пары ребёр многоугольника из САР выполнить:

- 1) Извлечь
- 2) Инициализировать Z значением Zл
- 3) Для каждого пикселя Xл≤X≤Xп вычислить Z(x,y=const). Z1=Zл, ..., Zk=Zk-1+ΔZx
- 4) Если Z>Zбуф(X), то Zбуф(X)=Z и занести атрибуты многоугольника в буфер кадра

- e) Записать буфер кадра сканирующей строки в буфер кадра дисплея

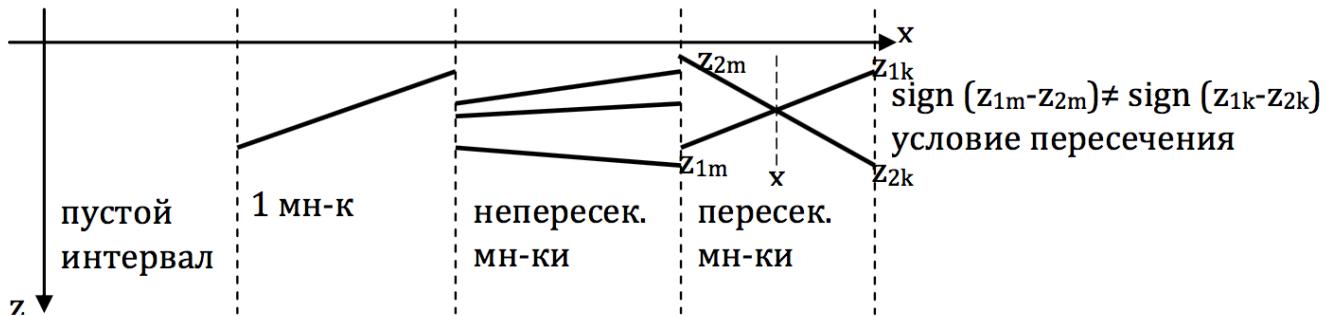
- f) Скорректировать САР

- 1) ΔYл--, ΔYп--
- 2) Xл=Xл+ΔХл, Xп=Xп+ΔХп
- 3) Zл=Zл+ΔZxΔX+ΔZy

- 4) Если  $\Delta Y_l < 0$  или  $\Delta Y_p < 0$ , то удалить соответствующее ребро из списка, пометив положение обоих рёбер в списке и породивший их многоугольник
- g) Скорректировать CAM
- 1)  $\Delta Y--$
  - 2) Если  $\Delta Y < 0$ , то удалить многоугольник из CAM

### Интервальные методы построчного сканирования

В алгоритме построчного сканирования с использованием Z-буфера глубина многоугольника вычисляется для каждого пикселя на сканирующей строке. Количество вычислений можно сократить, если использовать понятие интервалов. Решение задачи удаления невидимых поверхностей сводится к выбору видимых отрезков в каждом интервале, полученном путём деления сканирующей строки проекциями точек пересечения ребёр



1. Занести в буфер кадра цвет фона
2. Занести в буфер атрибуты многоугольника, соответствующие этому отрезку
3. Занести в буфер атрибуты многоугольника, соответствующие отрезку с MAX Z
4. Занести в буфер кадра параметры многоугольника максимального по z посередине интервала
5.  $\text{sign}(z_{1m}-z_{2m}) \neq \text{sign}(z_{1k}-z_{2k})$  – разбить интервал точкой пересечения, перейти к 4

## **35. Алгоритм определения видимых поверхностей путем трассировки лучей.**

Основная идея этого метода заключается в том, что наблюдатель видит объект благодаря световым лучам, испускаемым некоторым источником, которые падают на объект. Свет достигает наблюдателя, если он отражается от поверхности, преломляется или проходит через нее.

Используется подход, в котором отслеживаются (трассируются) лучи в обратном направлении, т.е. от наблюдателя к объекту. Алгоритм работает в пространстве изображения, наблюдатель находится в бесконечности на положительной полуоси  $Z$ .

В качестве объемлющей оболочки удобно взять сферу или прямоугольный параллелепипед.

Для сферы:

Уравнения луча:

$$x(t) = x_1 + (x_2 - x_1)t;$$

$$y(t) = y_1 + (y_2 - y_1)t;$$

$$z(t) = z_1 + (z_2 - z_1)t;$$

Минимальное расстояние от луча до центра сферы

$$l^2 = (x_1 + (x_2 - x_1)t - x_0)^2 + (y_1 + (y_2 - y_1)t - y_0)^2 + (z_1 + (z_2 - z_1)t - z_0)^2,$$

$$\frac{dl^2}{dt} = 2(a + (x_2 - x_1)t)(x_2 - x_1) + 2(b + (y_2 - y_1)t)(y_2 - y_1) + 2(c + (z_2 - z_1)t)(z_2 - z_1) = 0,$$

$$t_{\min} = - \frac{a(x_2 - x_1) + b(y_2 - y_1) + c(z_2 - z_1)}{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}.$$

Если  $l^2(t_{\min}) > R^2 \Rightarrow$  пересечения нет

Для прямоугольного параллелепипеда:

Выполнить преобразование, совмещающие луч с осью  $Z$ :

$$(x_{\min} < 0) \wedge (x_{\max} > 0) \wedge (y_{\min} < 0) \wedge (y_{\max} > 0) \Rightarrow \text{пересечение есть}$$

Нахождение пересечения с поверхностью второго порядка:

$$Q(x, y, z) = a_1x^2 + a_2y^2 + a_3z^2 + b_1xy + b_2xz + b_3yz + c_1x + c_2y + c_3z + d = 0.$$

Применяя преобразование, совмещающие луч с осью  $Z$ :

$$x = y = 0, z = -2c'_3 \pm \frac{\sqrt{c'^2_3 - 4a'_3d}}{2a'_3}$$

Если  $(c'^2_3 - 4a'_3d) < 0$ , то пересечения нет.

**Алгоритм трассировки лучей:**

1. Подготовка исходных данных.

Создание списка объектов со следующей информацией:

- тип объекта
- коэффициенты
- параметры объемлющей оболочки

2. Выполнение следующих действий для каждого трассируемого луча:

- Выполнение теста со сферической оболочкой для каждой поверхности в исходной системе координат. Если пересечение найдено, то соответствующее тело занести в список активных тел.
- Если список пуст, выставить текущих пиксель цветом фона. Иначе найти преобразование, совмещающее луч с осью  $Z$ (поворот и перенос).

3. Выполнение следующих операций для каждого объекта из списка активных объектов:

- Если используется прямоугольная объемлющая оболочка, то выполнить преобразование оболочки в новую систему координат и выполнить тест.

- Если пересечение луча с оболочкой установлено, то выполнить преобразование тела и найти пересечение, занести его в список пересечений. Иначе перейти к следующему объекту.
- Если список пересечений пуст, высветить цвет фона
- Иначе выбрать пересечение с максимальной глубиной  $z_{max}$  и закрасить цветом ближайшей поверхности, либо учитывая модель освещения
- Если учитывается освещение, то либо преобразовать источник освещения, либо выполнить обратное преобразование для самой точки.

## **36. Построение реалистических изображений. Физические и психологические факторы, учитываемые при создании реалистичных изображений. Простая модель освещения.**

**При построении реалистичного изображения необходимо:**

- 1) Учитывать оптические свойства поверхностей
- 2) Воспроизводить рисунок на поверхности
- 3) Воспроизводить неровности
- 4) Учитывать, что поверхности отбрасывают тени
- 5) Учитывать восприятие окружающего мира человеческим глазом.

**Простая модель освещения.** Объект можно увидеть, если он отражает или пропускает свет, если же объект поглощает весь падающий свет, то он будет невидим.

**Виды отражения:**

- 1) Диффузное

При диффузном считается, что тело поглощает падающую энергию. Диффузно отраженная энергия распространяется равномерно в пространстве (во всех направлениях). Диффузное отражение происходит, когда свет как бы проникает под поверхность объекта, поглощается, а затем вновь испускается.

$I = I_l k_d \cos \theta, 0 \leq \theta \leq \frac{\pi}{2}$ , где  $I$  – интенсивность отраженного света;  $I_l$  – интенсивность точечного источника;  $k_d$  – коэффициент диффузного отражения  $0 \leq k_d \leq 1$ ;  $\theta$  - угол между направлением света и нормалью к поверхности.

Но помимо обычного источника есть еще и рассеивающие, следовательно формула будет выглядеть так:  $I = I_p k_p + I_l k_d \cos \theta, 0 \leq \theta \leq \frac{\pi}{2}$ ,  $I_p$  – интенсивность рассеянного света,  $k_p$  - коэффициент диффузного отражения рассеянного света.

Так как интенсивность зависит от расстояния пройденного лучем можно формулу записать в следующем виде, учитывая что интенсивность пропорциональна не квадрату расстояние, а линейному.  $I = I_p k_p + \frac{I_l k_d \cos \theta}{d + K}$ ,  $d$  – расстояние,  $K$  – константа, чтобы избежать деления на ноль(подбирается в зависимости от эксперимента).

- 2) Зеркальное

Зеркальное отражение направленное, угол отражения от идеальной зеркальной поверхности(гладкой) равен углу падения. Благодаря зеркальному отражению на блестящих предметах появляются блики.

Интенсивность зеркально отраженного света зависит от угла падения, длины волны и свойств вещества:

$I_s = I_l w(\theta, \lambda) \cos^n(\alpha)$ , где  $w(\theta, \lambda)$  – функция отражения, представляющая отношение зеркально отраженного света к падающему(угол падения, длина), а  $n$  – степень гладкости поверхности

Учитывая рассеивание и диффузное отражение

$$I = I_p k_p + \frac{I_l (k_d \cos \theta + w(\theta, \lambda) \cos^n(\alpha))}{d + K}$$

Функцию  $w(\theta, \lambda)$  заменяют константой  $k_z$ , которая выбирается из эстетических соображений или определяется экспериментально. В итоге  $I = I_p k_p + \frac{I_l (k_d \cos \theta + k_z \cos^n(\alpha))}{d + K}$ .

Если имеется несколько источников света, то интенсивности от отдельных источников

$$\text{суммируются } I = I_p k_p + \sum_{j=1}^m \frac{I_j (k_d \cos \theta_j + k_z \cos^{n_j}(\alpha_j))}{d + K}, \text{ где } m \text{ – количество источников света.}$$

## 37. Построение реалистических изображений. Метод Гуро закраски поверхностей (получение сглаженного изображения).

Существуют три основных способа закраски объектов:

- \* простая закраска,
- \* закраска по Гуро(интерполяцией интенсивности),
- \* закраска по Фонгу(интерполяция векторов нормалей).

В простой закраске каждая грань закрашивается одним уровнем интенсивности. При ее использовании предполагается что:

- 1) Источник света расположен в бесконечности, поэтому угол между падающим лучом и нормалью постоянен на всей грани.
- 2) Наблюдатель находится в бесконечности, поэтому угол между вектором наблюдения и отражения постоянен на всей полигональной грани.
- 3) Закрашиваемая грань является реально существующей, а не получена в результате аппроксимации.

Использование этого типа закраски приводит к тому, что переход от одной грани к другой очень заметен.

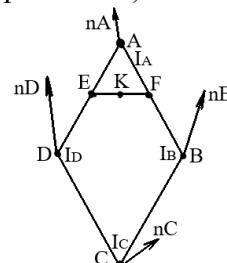
Для сглаживания используют метод Гуро.

- 1) Вычисляются нормали к поверхности.
- 2) Определяются нормали в вершинах путем усреднения нормалей по всем полигональным граням, которым принадлежит рассматриваемая вершина.
- 3) Вычисляются значения интенсивностей в вершинах многоугольника.
- 4) Осуществляется закраска многоугольника с использованием билинейной интерполяции интенсивностей. Сначала вычисляются значения интенсивностей вдоль каждого ребра (первая линейная интерполяция), а затем вычисляются значения интенсивностей вдоль сканирующей строки между ребрами многоугольника (вторая линейная интерполяция).

$$I_E = (1 - v)I_A + vI_D, \text{ где } v = \frac{AE}{AD},$$

$$I_F = (1 - w)I_A + wI_B, \text{ где } w = \frac{AF}{AB},$$

$$I_K = (1 - t)I_E + tI_F, \text{ где } t = \frac{EK}{EF}$$



Инкрементальное вычисление для сканирующей строки:

$$I_{K_1} = (1 - t_1)I_E + t_1I_F, \quad I_{K_2} = (1 - t_2)I_E + t_2I_F, \quad I_{K_2} = I_{K_1} + I_F(v_2 - v_1) + I_E(v_1 - v_2) = I_{K_1} + (I_F - I_E)(t_2 - t_1) = I_{K_1} + \Delta I \Delta t$$

Этот тип закраски хорошо сочетается с диффузным отражением.

Недостаток: усреднение нормалей вследствие чего поверхность может казаться плоской. Если нужно сохранить переход то усреднение не делается. Иначе вводятся дополнительные многоугольники.

## 38. Построение реалистических изображений. Закраска Фонга (улучшение аппроксимации кривизны поверхности).

Для построения реалистичности изображений необходимо учитывать закраску поверхностей. В зависимости от положения объекта наблюдателя и источника освещения, объект может закрашиваться разными уровнями интенсивности. Каждая грань тела закрашивается одним уровнем интенсивности, если:

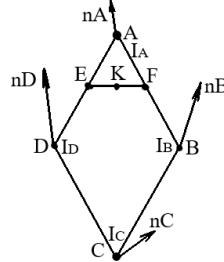
- 1) Источник света находится в бесконечности.
- 2) Наблюдатель находится в бесконечности.
- 3) Закрашиваемый многоугольник реально существует, а не получен в результате аппроксимации поверхности. Если сферическая поверхность аппроксимирована полигонами, то каждый будет закрашен разными цветами, и мы увидим ребра, а их быть не должно.

Необходимо улучшать изображение. Фонг предложил сглаживание изображения путем билинейной интерполяции нормали. Метод Фонга требует больших вычислительных затрат в отличии от метода Гуро. Сначала вычисляются нормали к поверхностям, затем на основе этих нормалей вычисляются значения нормалей в вершинах многоугольников. Далее выполняется аппроксимация вдоль ребер.

$$\vec{n}_E = \vec{n}_A(1 - V) + V \vec{n}_D; \quad V = \frac{AE}{AD}$$

$$\vec{n}_F = \vec{n}_A(1 - W) + W \vec{n}_B; \quad W = \frac{AF}{AB}$$

$$\vec{n}_K = \vec{E}(1 - t) + t \vec{n}_F; \quad t = \frac{EK}{EF}$$



Значение нормали, вычисленное описанным способом, используется для вычисления интенсивности пикселя. Закраска по методу Фонга достигается лучшая локальная аппроксимация кривизны поверхности и, следовательно, получается более реалистическое изображение, в частности, правдоподобнее выглядят зеркальные блики. Так же есть ситуация при которой может быть резкий скачок интенсивности в точке с при интерполяции

## 39. Определение нормали к поверхности и вектора отражения (4 способа) в алгоритмах построения реалистических изображений.

**Определение нормали к поверхности.** (нужны только внешние нормали)

1) Для того, чтобы вычислить нормаль к поверхности, можно векторно умножить 2 вектора, являющихся касательными к поверхности. Эти векторы можно определить через частные производные функции, задающей исходную поверхность, по направлениям:  $\frac{\partial Q(x, y)}{\partial x}, \frac{\partial Q(x, y)}{\partial y}$ .

Во многих алгоритмах удаления невидимых линий и поверхностей используются только ребра и вершины, поэтому при объединении их с моделью освещения необходимо знать приближенное значение нормали на ребрах и в вершинах. Пусть заданы уравнения плоскостей, несущих грани модели, тогда нормаль к их общей вершине равна среднему значению нормалей ко всем этим плоскостям:

$$\vec{n} = \vec{n}_1 + \vec{n}_2 + \vec{n}_3 = (a_1 + a_2 + a_3)\vec{i} + (b_1 + b_2 + b_3)\vec{j} + (c_1 + c_2 + c_3)\vec{k},$$

где  $a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3$ - коэффициенты уравнений плоскостей. (если требуется найти только направление нормали, то делить результат на количество граней не нужно). Нормаль к вершине также можно определить, усредняя векторные произведения всех ребер, пересекающихся в вершине:  $\vec{n} = [\vec{V}_1, \vec{V}_2] + [\vec{V}_2, \vec{V}_3] + [\vec{V}_3, \vec{V}_1]$ .

**Определение вектора отражения** (вектор нормали, вектор падения и вектор отражения лежат в одной плоскости, угол падения равен углу отражения):

1) Этот случай удобен, когда источник света расположен на оси z и когда он единственный. Начало системы координат переносится в точку поверхности, тогда проекции нормали и вектора отражения на плоскость xy лежат на одной прямой. (Все действия с единичными векторами)

$$nx^2 + ny^2 + nz^2 = 1, Rx^2 + Ry^2 + Rz^2 = 1.$$

Из второго рисунка по подобию треугольников:

$$\frac{nx}{Ry} = \frac{Rx}{Ry}, \frac{nx^2}{ny^2} = \frac{Rx^2}{Ry^2}.$$

Из третьего рисунка:

$$nz = \cos(\theta), Rz = \cos(2*\theta) = 2*\cos^2(\theta) - 1, Rz = 2*nz^2 - 1,$$

$$Rx^2 + Ry^2 = 1 - Rz^2 = 1 - (2*nz^2 - 1)^2 = 1 - 4*nz^4 + 4*nz^2 - 1 = 4*nz^2(1 - nz^2) = 4*nz^2(nz^2 + ny^2)$$

$$Rx^2 \left( 1 + \frac{Ry^2}{Rx^2} \right) = Rx^2 \left( 1 + \frac{ny^2}{nx^2} \right) = 4*nz^2(nx^2 + ny^2),$$

$$\frac{Rx^2}{nx^2}(nx^2 + ny^2) = 4*nz^2(nx^2 + ny^2) \rightarrow Rx^2 = 4nz^2*nx^2 \rightarrow Rx = 2nx*nz,$$

$$Ry = \frac{ny*Ry}{nx} = \frac{ny*2nx*nz}{nx} = 2*ny*nz$$

2) Этот способ удобен в том случае, если в сцене несколько источников освещения. Совмещение вектора нормали с осью z (единичные векторы):

$$\vec{L}_Z = \vec{R}_Z, \vec{L}_X = -\vec{R}_X, \vec{L}_Y = -\vec{R}_Y, \text{ где вектор } L - \text{вектор падения.}$$

3) Условие того, что векторы падения, нормали и отражения лежат в одной плоскости, можно записать так:  $[\vec{L}, \vec{n}] = [\vec{n}, \vec{R}]$ . Составляем определитель для первого и второго векторных произведений через знак равенства. Раскрываем определитель:

$$\vec{i}(L_Y \cdot n_Z - n_Y \cdot L_Z) + \vec{j}(L_Z \cdot n_X - n_Z \cdot L_X) + \vec{k}(n_Y \cdot L_X - n_X \cdot L_Y) =$$

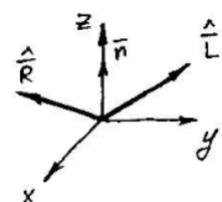
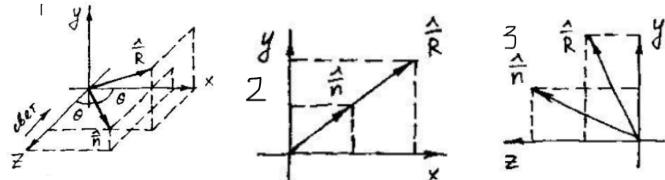
$$\vec{i}(n_Y \cdot R_Z - R_Y \cdot n_Z) + \vec{j}(n_Z \cdot R_X - R_Z \cdot n_X) + \vec{k}(n_X \cdot R_Y - R_X \cdot n_Y)$$

Получаем систему из 3х уравнений с 3-мя неизвестными. Система ЛЗ, поэтому нужно добавить еще одно уравнение, используя скалярное произведение:  $\vec{L} \cdot \vec{n} = \vec{n} \cdot \vec{R}$ ,

$$Lx*nx + Ly*ny + Lz*nz = nx*Rx + ny*Ry + nz*Rz.$$

4) Аналитический метод

$$\cos(\theta) = (-\vec{L}, \vec{n}) - \text{скалярное произведение}, \cos(\theta) = (\vec{n}, \vec{R})$$



Представим вектор R как линейную комбинацию векторов L и n:  $\vec{R} = \alpha \vec{L} + \beta \vec{n}$ ,  
 $(\alpha \vec{L} + \beta \vec{n}) = \cos(\Theta)$ ,  $\alpha(\vec{L}, \vec{n}) + \beta(\vec{n}, \vec{n}) = -\alpha \cos(\Theta) + \beta = \cos(\Theta)$ ,  $\beta = (1 + \alpha) \cdot \cos(\Theta)$ ,  
 $(\alpha \vec{L} + \beta \vec{n})^2 = \alpha^2(\vec{L}, \vec{L}) + 2 \cdot \alpha \beta (\vec{L}, \vec{n}) + \beta^2(\vec{n}, \vec{n}) = 1$ ,  $\alpha^2 - 2\alpha \beta \cos(\Theta) + \beta^2 = 1$ ,  
 $\alpha^2 - 2\alpha * \cos^2(\Theta) * (1 + \alpha) + (1 + \alpha)^2 * \cos^2(\Theta) = 1$ ,  $\alpha^2 + (1 + \alpha) * \cos^2(\Theta) * (1 + \alpha - 2\alpha) = 1$ ,  
 $\alpha^2 + (1 + \alpha) * \cos(\Theta) * (1 - \alpha) = 1$ ,  $\alpha^2 + \cos^2(\Theta) * (1 - \alpha^2) = 1$ ,  $\alpha^2 + \cos^2(\Theta) - \alpha^2 * \cos^2(\Theta) = 1$ ,  
 $\alpha^2(1 - \cos^2(\Theta)) = 1 - \cos^2(\Theta)$ ,  $\alpha^2 = 1 \rightarrow \alpha = \pm 1$

Векторы n и R сонаправлены, поэтому  $\alpha = 1$ ,  $\beta = 2 \cdot \cos(\Theta)$

## **40. Построение теней при создании реалистических изображений. Учет теней в алгоритмах удаления невидимых поверхностей.**

Тень не зависит от положения наблюдателя. Если направление вектора наблюдения совпадает с вектором падения света, то тень не видна.

**Тень состоит из двух частей:**

\* полная тень — центральная, темная, резко очерченная часть;

\* полутень — окружает полную, более светлая часть тени.

Различают **собственные тени** — когда объект препятствует попаданию света на некоторые свои грани, и **проекционные тени** — когда один объект препятствует попаданию света на поверхность другого.

**Видимые тени** — те, которые видны со стороны наблюдателя.

**Процесс построения тени:**

1. удалить невидимые поверхности для положения каждого источника;

2. удалить невидимые поверхности для положения наблюдателя.

**Построение собственной тени:**

Удаление нелицевых граней, если точку наблюдения совместить с источником света (алгоритм Робертса).

**Построение проекционной тени:**

Построить проекции всех нелицевых граней на сцену, если точку наблюдения совместить с источником света. Точки пересечения проецируемой грани со всеми другими плоскостями образуют многоугольники, которые помечают как теневые и заносят в список.

**Интервальный метод построчного сканирования:**

1) Для каждого источника и многоугольника определить собственные тени и проекционные тени. Они записываются в виде двоичной матрицы – строки – многоугольники, отбрасывающие тень, столбцы – затеняемые многоугольники

2) Обработка сцены относительно положения наблюдателя:

1. Определить видимые отрезки на интервале

2. С помощью списка теневых многоугольников определить, падает ли тень на многоугольник, которые создаёт видимый отрезок на данном интервале

а) нет теневых многоугольников – изобразить отрезок

б) есть теневые многоугольники, но они не пересекают и не покрывают данный интервал – изобразить видимый отрезок

с) интервал полностью покрывается хотя бы одним многоугольником – интенсивность изображаемого отрезка определяется с учётом интенсивностей этих многоугольников и самого отрезка

д) хотя бы один теневой многоугольник частично покрывает интервал – рекурсивно разбивать интервал и работать с подинтервалами.

**Алгоритм, использующий Z-буфер:**

1) Строится сцена, где точка наблюдения совпадает с источником. Запомнить значения Z в отдельном теневом Z-буфере.

2) Строится сцена из точки наблюдателя. При обработке каждой поверхности глубина каждого пикселя сравнивается с глубиной Z<sub>буф</sub> наблюдателя. Если > (видима), то x'uz из вида наблюдателя преобразуются в x' u' z' на виде из источника. Для того, чтобы проверить, видимо ли значение z' из положения источника, она сравнивается со значением теневого z-буфера при x' u'. Если оно видимо, то оно отображается в буфер кадра x, у без изменений. Если нет, то точка находится в тени

и изображается согласно правилу расчета интенсивности с учетом затемнения, а значение в z-буфере наблюдателя заменяется на  $z'$ .

**Алгоритм Вейлера-Азертона (этапы):**

- 1) С помощью алгоритма удаления невидимых поверхностей выделяются освещенные, видимые из положения источника, грани (хранятся именно они).
- 2) Освещенные многоугольники помечаются и преобразуются к исходной ориентации, где приписываются к своим прототипам (путем присвоения своего номера каждому многоугольнику или его части).
- 3) Объединенные данные о многоугольниках обрабатываются со стороны наблюдателя. Если какая-либо область не освещена, то применяется соответствующее правило расчета интенсивности с учетом затемнения. (для нескольких источников света будет добавляться несколько наборов освещенных граней)

**Алгоритм трассировки лучей (этапы):**

- 1) Трассировкой лучей от точки наблюдения через плоскость проекции определяются видимые точки сцены.
- 2) Трассировка лучей от видимой точки до источника света. Если на пути луча в сцене встретился какой-то объект, то свет от источника не попадает в данную точку и она оказывается в тени.

## **41. Учет прозрачности в модели освещения. Учет прозрачности в алгоритмах удаления невидимых поверхностей.**

**Зеркальное (направленное)** – свойственно прозрачным телам. Если смотреть на объект сквозь такое вещество, то за исключением контурных линий криволинейных поверхностей, искажения происходить не будет.

**Диффузное (рассеянное)** – вещества кажутся полупрозрачными или матовыми.

Чтобы включить преломление в модель освещения нужно при построении видимых поверхностей учитывать не только падающий, но и отражённый и пропущенный свет. Эффективнее это делается с помощью глобальной модели освещения в сочетании с алгоритмом трассировки лучей для выделения видимых поверхностей. В более простых реализациях эффект преломления не рассматривается, не принимается во внимание как путь, пройденным лучом в среде, влияет на его интенсивность. Простое пропускание света можно встроить в любой алгоритм, кроме алгоритма с Z-буфером. Прозрачные поверхности помечаются и если видимая грань прозрачна, то в буфер кадра записывается линейная комбинация двух ближайших поверхностей. При этом интенсивность:  $I = t \cdot I_1 + (1 - t)I_2$ ,  $0 \leq t \leq 1$ . Где  $I_1$ - видимая поверхность,  $I_2$ - поверхность, расположенная непосредственно за ней,  $t$  – коэффициент прозрачности первой поверхности. Если вторая поверхность тоже прозрачна, то алгоритм применяется рекурсивно, пока не встретиться непрозрачная поверхность или фон.

**Алгоритм с z-буфером (нужны отдельные буфера прозрачности, интенсивности, весовых коэффициентов):**

- 1) Для каждого многоугольника
- 2) Если прозрачен – занести в список прозрачных
- 3) Если нет и  $Z > Z_{\text{буф}}$ , то записать в буфер кадра для непрозрачных многоугольников и скорректировать  $Z_{\text{буф}}$
- 4) Для каждого многоугольника из списка прозрачных
  - 5) Если  $Z > Z_{\text{буф}}$ , то прибавить его коэффициент прозрачности к значению в буфере весовых коэффициентов прозрачности, прибавить его интенсивность к значению в буфере интенсивности прозрачности по правилу:  $I_{bn} = I_{bo} \cdot t_{bo} + I_c \cdot t_c$ , ( $bn$  – новое,  $bo$  – старое,  $c$  – для многоугольника). Таким образом получается взвешенная сумма интенсивностей всех прозрачных многоугольников, находящихся перед ближайшим непрозрачным.
  - 6) Объединим буфера интенсивности для прозрачных и непрозрачных многоугольников :  $I_{fb} = t_{bo} \cdot I_{bo} + (1 - t_{bo})I_{fbo}$  ( $I_{fb}$  – окончательная интенсивность в буфере кадра,  $I_{fbo}$  – старое значение интенсивности в этом буфере).

## 42. Учет фактуры при создании реалистических изображений.

Под фактурой в компьютерной графике понимается детализация строения поверхности.

**Рассматривают 2 вида детализации:**

- 1) На гладкую поверхность наносят рисунок, в этом случае поверхность остается гладкой.
- 2) Во втором случае ставится задача создания неровностей на поверхности, т.е создание шероховатой поверхности.

**Существующие способы моделирования текстуры можно подразделить на:**

- 1) Проективные текстуры
- 2)Процедурные текстуры.

В первом случае задается рисунок, в некоторой системе координат и ведется поиск функции отображения координат из исходной системы координат в нужную. Находим точки отображения, анализируем, В первом случае для точек плоского рисунка находим соответствующие им координаты точек на поверхности, во втором случае для точек поверхности находим соответствующие им точки на плоскости и анализируем, принадлежит ли полученная точка рисунку или нет. Если принадлежит, то исходную точку поверхности закрашиваем, в противном случае – нет.

Нанесение на поверхность рисунка можно рассматривать как детализацию цветом. Для ее реализации используются многоугольники детализации. Они лежат в той же плоскости что и основные, но помечаются так, чтобы в алгоритмах удаления невидимых поверхностей могли присвоить им более высокие приоритеты.

Во втором случае необходимо найти функцию  $C(x, y, z)$ , определяющую для каждой точки поверхности цвет таким образом, чтобы он соответствовал цвету моделируемого материала (наносимого рисунка). Такие текстуры называют процедурными. Например для дерева можно использовать следующую функцию

$$Q(x, y) = C_1 + (C_2 - C_1)f(\sqrt{x^2 + y^2}), \text{ где } C_1, C_2 \text{ – некоторые цвета, соответствующие светлым и темным кольцам структуры дерева; } f(t) \text{ – некоторая неотрицательная периодическая функция.}$$

Пусть  $Q(x, y)$  – уравнение поверхности, т.е. функция Q позволяет для каждой точки поверхности определить ее третью координату Z. Нормаль в точке поверхности может определяться векторным произведением этих производных  $Q'_x, Q'_y$ :  $\vec{n} = Q'_x(x, y) \times Q'_y(x, y)$ ; Пусть P(X, Y) – функция возмущения, тогда радиус-вектор точки на новой поверхности будет определяться

$$Q_b = Q(x, y) + P(x, y) \cdot \frac{\vec{n}}{|n|}, \text{ тогда } Q'_{bx} = Q'_x(x, y) + P'_x(x, y) \cdot \frac{\vec{n}}{|n|} + P(x, y) \cdot \frac{\vec{n}}{|n|}' x, \text{ а}$$

$$Q'_{by} = Q'_y(x, y) + P'_y(x, y) \cdot \frac{\vec{n}}{|n|} + P(x, y) \cdot \frac{\vec{n}}{|n|}' y \rightarrow \vec{n}_b = Q'_x \times Q'_y + \frac{P'_x(\vec{n}) \times Q'_y}{|n|}$$

## **43. Глобальная модель освещения с трассировкой лучей.**

Модель освещения предназначена для расчета интенсивности отраженного к наблюдателю света в каждой точке изображения. Разделяют локальную и глобальную модели освещения.

**Локальная модель** учитывает только свет, падающий от источника, и ориентацию поверхности:  $I = I_p k_p + I_{\text{и}} k_{\text{д}} \cos(\Theta) + I_{\text{и}} k_3 \cos^n(\alpha)$ .

**Глобальная модель** учитывает также свет, отраженный от других объектов или пропущенный сквозь них:  $I = I_p k_p + I_{\text{и}} k_{\text{д}} \cos(\Theta) + I_{\text{и}} k_3 \cos^n(\alpha) + k_3 I_3 \cos(\varphi) + k_{\text{пр}} I_{\text{пр}} \cos(\Psi)$ .

При трассировке отслеживают траекторию луча, который после столкновения с поверхностью частично поглощается, частично диффузно рассеивается, а остальная энергия луча идет на зеркальное отражение и преломление. Если источник света видим из точки пересечения, то сначала вычисляют вклад этого источника в освещенность рассматриваемой точки, используя модель отражения. Затем трассируемый луч разделяют на составляющие отражения и преломления. Эти две составляющие в дальнейшем будут анализироваться, как и начальные приведенные лучи. При столкновении такого вторичного луча с поверхностью могут возникнуть новые вторичные лучи. Трассировку организуют как рекурсивную процедуру, которая сама себя вызывает, как только выясняется, что анализируемый луч отражается или преломляется.

Процедура выполняется до тех пор, пока не останется пересечений либо пока интенсивность не станет слишком мала, чтобы ее учитывать. При таком алгоритме строится дерево, правые ветви которого это преломленные лучи, левые – отраженные, а узлы соответствуют точкам пересечения с поверхностями. Для расчёта интенсивности в точке нужно пройти дерево в обратном порядке. Таким образом, в отличие от простой модели освещения, глобальная модель дополнитель но учитывает зеркальное отражение от других поверхностей, попадающее в рассматриваемую точку, а также преломление, т.е. интенсивность, приносимую преломленным лучом. Кроме того, в расчетах надо учитывать ослабление пропущенного и зеркально отраженного света с расстоянием между точками пересечения.

## 44. Алгоритм трассировки лучей с использованием глобальной модели освещения

Основная идея этого метода заключается в том, что наблюдатель видит объект благодаря световым лучам, испускаемым некоторым источником, которые падают на объект. Свет достигает наблюдателя, если он отражается от поверхности, преломляется или проходит через нее. Используется подход, в котором отслеживаются (трассируются) лучи в обратном направлении, т.е. от наблюдателя к объекту. Алгоритм работает в пространстве изображения, наблюдатель находится в бесконечности на положительной полуоси  $Z$ .

В качестве объемлющей оболочки удобно взять сферу или прямоугольный параллелепипед.

Для сферы:

Уравнения луча:

$$x(t) = x_1 + (x_2 - x_1)t;$$

$$y(t) = y_1 + (y_2 - y_1)t;$$

$$z(t) = z_1 + (z_2 - z_1)t;$$

Минимальное расстояние от луча до центра сферы

$$l^2 = (x_1 + (x_2 - x_1)t - x_0)^2 + (y_1 + (y_2 - y_1)t - y_0)^2 + (z_1 + (z_2 - z_1)t - z_0)^2,$$

$$\frac{dl^2}{dt} = 2(a + (x_2 - x_1)t)(x_2 - x_1) + 2(b + (y_2 - y_1)t)(y_2 - y_1) + 2(c + (z_2 - z_1)t)(z_2 - z_1) = 0,$$

$$t_{\min} = - \frac{a(x_2 - x_1) + b(y_2 - y_1) + c(z_2 - z_1)}{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}.$$

Если  $l^2(t_{\min}) > R^2 \Rightarrow$  пересечения нет

Для прямоугольного параллелепипеда:

Выполнить преобразование, совмещающие луч с осью  $Z$ :

$$(x_{\min} < 0) \wedge (x_{\max} > 0) \wedge (y_{\min} < 0) \wedge (y_{\max} > 0) \Rightarrow \text{пересечение есть}$$

Нахождение пересечения с поверхностью второго порядка:

$$Q(x, y, z) = a_1x^2 + a_2y^2 + a_3z^2 + b_1xy + b_2xz + b_3yz + c_1x + c_2y + c_3z + d = 0.$$

Применяя преобразование, совмещающие луч с осью  $Z$ :

$$x = y = 0, z = -2c_3' \pm \sqrt{\frac{c_3'^2 - 4a_3'd}{2a_3'}}$$

Если  $(c_3'^2 - 4a_3'd) < 0$ , то пересечения нет.

**Алгоритм трассировки лучей:**

1. Подготовка исходных данных.

Создание списка объектов со следующей информацией:

- тип объекта
- коэффициенты
- параметры объемлющей оболочки

2. Выполнение следующих действий для каждого трассируемого луча:

- Выполнение теста со сферической оболочкой для каждой поверхности в исходной системе координат. Если пересечение найдено, то соответствующее тело занести в список активных тел.
- Если список пуст, выставить текущих пиксель цветом фона. Иначе найти преобразование, совмещающее луч с осью  $Z$ (поворот и перенос).

3. Выполнение следующих операций для каждого объекта из списка активных объектов:

- Если используется прямоугольная объемлющая оболочка, то выполнить преобразование оболочки в новую систему координат и выполнить тест.

- Если пересечение луча с оболочкой установлено, то выполнить преобразование тела и найти пересечение, занести его в список пересечений. Иначе перейти к следующему объекту.
- Если список пересечений пуст, высветить цвет фона
- Иначе выбрать пересечение с максимальной глубиной  $z_{max}$  и закрасить цветом ближайшей поверхности, либо учитывая модель освещения
- определение точки пересечения в исходной системе координат с использованием обратного преобразования;
- Затем вычислить интенсивность точки с учетом глобальной модели освещения и высветить пиксель с полученной интенсивностью.

Глобальная модель учитывает свет, падающий от источника, ориентацию поверхности, свет, отраженный от других объектов или пропущенный сквозь них:

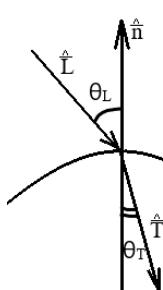
$$I = I_p k_p + I_i k_d \cos(\Theta) + I_i k_s \cos^n(\alpha) + k_3 I_3 \cos(\varphi) + k_{\text{пр}} I_{\text{пр}} \cos(\Psi).$$

При трассировке отслеживают траекторию луча, который после столкновения с поверхностью частично поглощается, частично диффузно рассеивается, а остальная энергия луча идет на зеркальное отражение и преломление.

## 45. Определение направления преломленного луча.

В более сложных моделях освещения учитываются пропускающие свойства поверхностей, т.е. прозрачность этих поверхностей, что приводит к наличию преломленного луча.

Закон преломления: падающий и преломленной лучи, а также вектор нормали лежат в одной плоскости, а углы падения и преломления связаны следующей формулой:  $\eta_1 \sin \theta_1 = \eta_2 \sin \theta_2$ .



Вектор преломления представляет собой линейную комбинацию вектора падающего света и вектора нормали:

$$\hat{T} = \alpha \hat{L} + \beta \hat{n}, \quad (-\hat{L}; \hat{n}) = \cos(\theta_L), \quad (-\hat{n}; \hat{T}) = \cos(\theta_T).$$

$$(\alpha \hat{L} + \beta \hat{n}; -\hat{n}) = \cos(\theta_T), \quad \alpha (\hat{L}; -\hat{n}) + \beta (\hat{n}; -\hat{n}) = \cos(\theta_T)$$

$$\alpha \cos \theta_L - \beta = \cos \theta_T, \quad \beta = \alpha \cos \theta_L - \cos \theta_T$$

Вычислим скалярное произведение единичного вектора преломления самого на себя:

$$\hat{T}^2 = (\alpha \hat{L} + \beta \hat{n})^2 = \alpha^2 (\hat{L}; \hat{L}) + 2\alpha\beta (\hat{L}; \hat{n}) + \beta^2 (\hat{n}; \hat{n}) = 1$$

$$\alpha^2 + 2\alpha\beta (\hat{L}; \hat{n}) + \beta^2 = 1, \quad \alpha^2 - 2\alpha\beta \cos \theta_L + \beta^2 = 1$$

Подставим  $\beta$ :

$$\alpha^2 - 2\alpha(\alpha \cos \theta_L - \cos \theta_T) \cos \theta_L + (\alpha \cos \theta_L - \cos \theta_T)^2 = 1$$

$$\alpha^2 - (\alpha \cos \theta_L - \cos \theta_T)(2\alpha \cos \theta_L - \cos \theta_L + \cos \theta_T)^2 = 1$$

$$\alpha^2 - (\alpha^2 \cos^2 \theta_L - \cos^2 \theta_T) = 1$$

Выразим угол преломления:

$$\eta_1^2 (1 - \cos^2(\theta_L)) = \eta_2^2 (1 - \cos^2(\theta_T)),$$

$$\eta^2 (1 - \cos^2(\theta_L)) = 1 - \cos^2(\theta_T),$$

$$\cos^2(\theta_T) = 1 - \eta^2 (1 - \cos^2(\theta_L)).$$

Подставим в уравнение:  $\alpha^2 - (\alpha^2 \cos^2(\theta_L) - 1 + \eta^2 (1 - \cos^2(\theta_L))) = 1, \quad \alpha^2 = \eta^2$ , получаем  $\alpha = \eta$   
(второй корень  $\alpha = -\eta$  не подходит по физическому смыслу)

$$\beta = \eta \cos \theta_L - \sqrt{1 - \eta^2 (1 - \cos^2 \theta_L)}$$

Под корнем может быть отрицательное значение, это будет означать полное внутреннее отражение, т.е. отсутствие преломленного луча при переходе из оптически более плотной среды в оптически менее плотную среду.