**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ**

**ФГБОУ ВПО СЕВЕРО-КАВКАЗСКИЙ ГОРНО-МЕТАЛЛУРГИЧЕСКИЙ**

**ИНСТИТУТ (ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ)**

## Методические указания для выполнения лабораторных работ

**дисциплины «*Алгоритмические основы современной***

***компьютерной графики*»**

для направления подготовки

230100.62 **- «**Информатика и вычислительная техника**»**

Профиль: Автоматизированные системы обработки информации и управления

Квалификация выпускника – **бакалавр**

Форма обучения – дневная

Разработчик программы:

к.т.н., доц.

каф. «Автоматизированной

обработки информации» Соколова Е.А

Владикавказ, 2014

.

УДК 004.92

ББК 73

С 59

Составители: **к.т.н. Соколова Е. А., Караева С.А.**

Рецензент: **доц., к.т.н. Будаева Алина Алибековна**

Методические указания к лабораторным работам по курсу «Алгоритмические основы современной компьютерной графики» для студентов специальности 230100.62 – «Информатика и вычислительная техника» - Владикавказ: "Терек", 2013.- 24 с.

Методические указания предназначены для выполнения лабораторных работ по курсу "Алгоритмические основы современной компьютерной графики" для студентов специальности 230100.62 – «Информатика и вычислительная техника». Дисциплина «Алгоритмические основы современной компьютерной графики» имеет своей целью систематическое изучение основ теории и практики построения алгоритмов современной компьютерной графики.

Подготовлено кафедрой «**Автоматизированная обработка информации**»

Редактор

Компьютерная верстка

Издательство «Терек» СКГМИ (ГТУ), 2013

Подписано в печать Формат

Тираж Объем усл.п.л. Заказ №.

Подразделение оперативной полиграфии СКГМИ (ГТУ) 362021, г. Владикавказ, ул. Николаева, 44

Оглавление

[Лабораторная работа №1: Алгоритм вычерчивания отрезков 4](#_Toc373490786)

[Лабораторная работа №2: Цифровой дифференциальный анализатор 5](#_Toc373490787)

[Лабораторная работа №3: Алгоритм Брезенхема 7](#_Toc373490788)

[Лабораторная работа №3: Алгоритм Брезенхема 7](#_Toc373490789)

[Лабораторная работа №4: Общий алгоритм Брезенхема 9](#_Toc373490790)

[Лабораторная работа №5: Алгоритм Брезенхема для генерации 11](#_Toc373490791)

[Лабораторная работа №6: Простой алгоритм заполнения с затравкой 13](#_Toc373490792)

[Лабораторная работа №7: Построчный алгоритм заполнения с затравкой 15](#_Toc373490793)

[Лабораторная работа №8: Алгоритм плавающего горизонта 18](#_Toc373490794)

[Лабораторная работа №9: Алгоритм, использующий z-буфер 24](#_Toc373490795)

# Лабораторная работа №1: Алгоритм вычерчивания отрезков

**Пошаговый алгоритм:**

позиция = начало

шаг = приращение

1 **if** позиция - конец < точность **then** 4

**if** позици > конец **then** 2

**if** позиция < конец **then** 3

2 позиция = позиция - шаг

**go to** 1

3 позиция = позиция + шаг

**go to** 1

4 **finish**

**Задание**

Программная реализация алгоритма вычерчивания отрезков на любом языке программирования.

# Лабораторная работа №2: Цифровой дифференциальный анализатор

Теоретические основы:

Один из методов разложения отрезка в растр состоит в решении дифференциального уравнения, описывающего этот процесс. Для прямой линии имеем

dy / dx = const или Dy / Dx = (y2 - y1) / (x2 - x1)

Решение представляется в виде:

yi+1 = yi + Dy

yi+1 = yi + Dx (y2 - y1) / (x2 - x1) [1]

где x1, y1 и x2, y2 - концы разлагаемого отрезка и yi - начальное значение для очередного шага вдоль отрезка.

**Integer** - функция преобразования вещественного числа в целое.

**Sign** - функция, возвращающая -1, 0, 1 для отрицательного, нулевого и положительного аргумента соответственно аппроксимируем длину отрезка.

**Алгоритм:**

**if** abs(x2 - x1) >= abs(y2 - y1) **then**

Длина = abs(x2 - x1)

**else**

Длина = abs(y2 - y1)

**end if**

полагаем большее из приращений Dx или Dy равными единице растра

Dx = (x2 - x1) / Длина

Dy = (y2 - y1) / Длина

округляем величины, а не отбрасываем дробную часть

использование знаковой функции делает алгоритм пригодным для всех квадрантов

x = x1 + 0.5 \* **Sign**(Dx)

y = y1 + 0.5 \* **Sign**(Dy)

начало основного цикла

i =1

**while** (i <= Длина)

**Plot** (**Integer**(x), **Integer**(y))

x = x + Dx

y = y + Dy

i = i + 1

**end while**

**finish**

**Задание**

Программная реализация цифрового дифференциального анализатора на любом языке программирования.

# Лабораторная работа №3: Алгоритм Брезенхема

|  |
| --- |
| C:\Users\ipi-cnit.sveta\Desktop\Новая папка (3)\Новая папка\image001.gif |

Рис. 1. Блок-схема алгоритма Брезенхема.

***Алгоритм Брезенхема разложения в растр отрезка для первого октанта***

Предполагается, что концы отрезка (x1,y1) и (x2,y2) не совпадают;

**Integer** - функция преобразования в целое;

x, y, Dx, Dy – целые;

е – вещественное.

инициализация переменных

x = x1

y = y1

Dx = x2 - x1

Dy = y2 - y1

Инициализация с поправкой на половину пиксела

е = Dy/Dx - 1/2

начало основного цикла

for i = 1 to Dx

plot (x,y)

while ( e => 0 )

y = y + 1

e = e - 1

end while

x = x + 1

e = e + Dy/Dx

next i

finish

**Задание**

Программно реализовать алгоритм Брезенхема для первого октанта.

# Лабораторная работа №4: Общий алгоритм Брезенхема

Теоретические основы:

Чтобы реализация алгоритма Брезенхема была полной необходимо обрабатывать отрезки во всех октантах. Модификацию легко сделать, учитывая в алгоритме номер квадранта, в котором лежит отрезок и его угловой коэффициент. Когда абсолютная величина углового коэффициента больше 1, у постоянно изменяется на единицу, а критерий ошибки Брезенхема используется для принятия решения об изменении величины x. Выбор постоянно изменяющейся (на +1 или -1) координаты зависит от квадранта.

***Обобщенный целочисленный алгоритм Брезенхема квадрантов***

Предполагается, что концы отрезка (x1,y1) и (x2,y2) не совпадают, все переменные считаются целыми

**Sign** - функция, возвращающая -1, 0, 1 для отрицательного, нулевого и положительного аргумента соответственно инициализация переменных

x = x1

y = y1

Dx = abs(x2 - x1)

Dy = abs(y2 - y1)

s1 = **Sign**(x2 - x1)

s2 = **Sign**(y2 - y1)

обмен значений Dx и Dy в зависимости от углового коэффициента наклона отрезка

**if** Dy < Dx **then**

Врем = Dx

Dx = Dy

Dy = Врем

Обмен = 1

**else**

Обмен = 0

**end if**

инициализация e с поправкой на половину пиксела

e = 2\*Dy - Dx

основной цикл

**for** i = 1 **to** Dx

**Plot**(x,y)

**while**(e =>0)

**if** Обмен = 1 **then**

x = x + s1

**else**

y = y + s2

**end if**

e = e - 2\*Dx

**end while**

**if** Обмен = 1 **then**

y = y + s2

**else**

x = x + s1

**end if**

e = e + 2\*Dy

**next i**

**finish**

**Задание**

Программно реализовать алгоритм Брезенхема для первого октанта.

Лабораторная работа №5: Алгоритм Брезенхема для генерацииокружности

Теоретические основы:

Один из наиболее эффективных и простых для понимания алгоритмов генерации окружности принадлежит Брезенхему . Если сгенерирован первый октант (от 0 до 45° против часовой стрелки), то второй октант можно получить зеркальным отражением относительно прямой у = х, что дает в совокупности первый квадрант. Первый квадрант отражается относительно прямой х = 0 для получения соответствующей части окружности во втором квадранте. Верхняя полуокружность отражается относительно прямой у = 0 для завершения построения.

|  |
| --- |
| C:\Users\ipi-cnit.sveta\Desktop\Новая папка (3)\Новая папка\image002.jpg |

Рис. 2. Блок-схема пошагового алгоритма Брезенхема для генерации окружности в первом квадранте.

**Пошаговый алгоритм Брезенхема для генерации окружности в первом квадранте**

все переменные – целые;

инициализация переменных;

xi = 0

yi = R

Di =2(1 - R)

Предел = 0

1 **Plot** (xi, yi)

**if** yi <= Предел **then 4**

Выделениеслучая 1 или 2, 4 или 5, или 3

if Di < 0 **then 2**

if Di **>** 0 **then 3**

if Di= 0 **then** 20

определение случая 1 или 2

2 d = 2Di+ 2уi - 1

if d <= 0 **then** 10

if d > 0 **then** 20

определение случая 4 или 5

3 d = 2Di+ 2хi **-** 1

**if** d ***<=*** 0 **then** 20

**if** d ***>* 0 then** 30

выполнение шагов

шаг к mH

10 хi = хi + 1

Di = Di+ 2хi + 1

**gо to 1**

шагmD

20 хi = хi + 1

yi = yi + 1

Di = Di+ 2хi - 2уi+ 2

**gо to 1**

**4 finish**

Программно реализовать алгоритм Брезенхема для генерации окружности для всех квадрантов.

# Лабораторная работа №6: Простой алгоритм заполнения с затравкой

Теоретические основы

**Простой алгоритм заполнения с затравкой и стеком**

* Поместить затравочный пиксель в стек
* Пока стек не пуст
* Извлечь пиксель из стека
* Присвоить пикселю требуемое значение
* Для каждого из соседних к текущему 4-связных пикселов проверить: является ли он граничным пикселем или не присвоено ли уже пикселю требуемое значение. Проигнорировать пиксель в любом из этих двух случаев. В противном случае поместить пиксель в стек

**Алгоритм:**

*Затравка (х, у) выдает затравочный пиксель*

**Push** - *процедура, которая помещает пиксель в стек*

**Pop** - *процедура, которая извлекает пиксель из стека*

Пиксель (х, у) = Затравка (х, у)

*Инициализируем стек*

**Push** Пиксель (х, у)

**While** (стек не пуст)

*Извлекаем пиксель из стека*

**Pop** Пиксель (х, у)

**If** Пиксель (х, у) < > Нов\_значение **then**

Пиксель (х, у) = Нов\_значение

**End if**

*Проверим, надо ли помещать соседние пиксели в стек*

**If** (Пиксель (х+1, у) < > Нов\_значение **and**

Пиксель (х+1, у) < > Гран\_значение) **Then**

**Push** Пиксель (х+1, у)

**If** (Пиксель (х, у+1) < > Нов\_значение **and**

Пиксель (х, у+1) < > Гран\_значение) **Then**

**Push** Пиксель (х, у+1)

**If** (Пиксель (х-1, у) < > Нов\_значение **and**

Пиксель (х-1, у) < > Гран\_значение) **Then**

**Push** Пиксель (х-1, у)

**If** (Пиксель (х, у-1) < > Нов\_значение **and**

Пиксель (х, у-1) < > Гран\_значение) **Then**

**Push** Пиксель (х, у-1)

**End if**

**End while**

**Задание**

Программно реализовать простой алгоритм заполнения с затравкой.

# Лабораторная работа №7: Построчный алгоритм заполнения с затравкой

Теоретические основы:

**Построчный алгоритм заполнения с затравкой**

* Затравочный пиксель на интервале извлекается из стека, содержащего затравочные пиксели.
* Интервал с затравочным пикселем заполняется влево и вправо от затравки вдоль сканирующей строки до тех пор, пока не будет найдена граница. В переменных Хлев и Хправ запоминаются крайний левый и крайний правый пиксели интервала.
* В диапазоне Хлев ? х ? Хправ проверяются строки, расположенные непосредственно над и под текущей строкой. Определяется, есть ли на них еще не заполненные пиксели. Если такие пиксели есть (т.е. не все пиксели граничные, или уже заполненные), то в указанном диапазоне крайний правый указанный пиксель в каждом интервале отмечается как затравочный и помещается в стек.

**Алгоритм:**

Затравка (х, у) выдает затравочный пиксель

**Pop** - процедура, которая извлекает пиксель из стека

**Push** - процедура, которая помещает пиксель в стек

инициализируем стек

Push Затравка (х, у)

**While** (стек не пуст)

Извлекаем пиксель из стека и присваиваем ему новое значение

**Pop** Пиксель (х, у)

Пиксель (х, у) = Нов\_значение

Сохраняем х-координату затравочного пикселя

Время\_х = х

Заполняем интервал справа от затравки

х = х+1

**while** Пиксель (х, у) < > Гран\_значение

Пиксель (х, у) = Нов\_значение

х = х+1

**end while**

сохраняем крайний справа пиксель

Хправ = х - 1

Восстанавливаем х-координату затравки

х = Врем\_х

заполняем интервал слева от заправки

х = х - 1

**While** Пиксель (х, у) < > Гран\_значение

Пиксель (х, у) = Нов\_значение

х = х - 1

**end while**

сохраняем крайний слева пиксель

Хлев = х+1

Восстанавливаем х-координату затравки

х = Врем\_х

проверим, что строка выше не является ни границей многоугольника, ни уже полностью заполненной; если это не так, то найти затравку, начиная с левого края подинтервала сканирующей строки.

х = Хлев

у = у+1

**while** х ? Хправ

ищем заправку на строке выше

Флаг = 0

**While** (Пиксель (х, у) < > Гран\_значение **and**

Пиксель (х, у) < > Нов\_значение **and** х < Хправ

**If** Флаг = 0 **then** Флаг = 1

х = х+1

**end while**

помещаем в стек крайний правый пиксель

**if** Флаг = 1 **then**

**if** (х = Хправ **and** Пиксель (х, у) < > Гран\_значение

**and** Пиксель (х, у) < > Нов\_значение) **then**

**Push** Пиксель (х, у)

**Else**

**Push** Пиксель (х - 1, у)

**End if**

Флаг = 0

**End if**

Продолжим проверку, если интервал был прерван

Хвход = х

**While** ((Пиксель (х, у) = Гран\_значение **or** Пиксель (х, у) = Нов\_значение)

**and** х < Хправ)х = х+1 **end while**

удостоверимся, что координата пикселя увеличена

**if** х = Хвход **then** х = х+1 **end while**

проверим, что строка ниже не является ни границей многоугольника, ни уже полностью заполненной

эта часть алгоритма совершенно аналогична проверки для строки выше, за исключением того, что вместо у = у+1 надо поставить у = у - 1

**end while**

**finish**

Здесь функция **Pop** извлекает координаты х, у пикселя из стека, а функция **Push** помещает их в стек.

**Задание**

Программно реализовать построчный алгоритм заполнения с затравкой.

# Лабораторная работа №8: Алгоритм плавающего горизонта

**Алгоритм плавающего горизонта**

Гэкран - разрешение экрана в горизонтальном направлении  
Вэкран - разрешение экрана в вертикальном направлении  
Верх - массив, содержащий ординаты верхнего горизонта  
Низ - массив, содержащий ординаты нижнего горизонта  
Y - текущее значение функции у = f(x, z) при z = const  
Тфлаг - флаг видимости для текущей точки   
Пфлаг - флаг видимости для предыдущей точки, равный

0 = невидима  
1 = видима и выше верхнего горизонта  
-1 = видима и ниже нижнего горизонта

Draw - графическая команда, которая чертит видимую линию между точками, заданными их координатами  
Xmin,Xmax - минимальная и максимальная абсциссы функции  
Хшаг - приращения вдоль оси х  
Zmin Zmax - минимальная и максимальная аппликаты функции  
Zшаг - шаг между плоскостями z. = const   
Dimension Верх (Гэкран), Низ (Гэкран)  
инициализация переменных  
Xлевое = -1; Yлевое = -1; Xправое = -1; Yправое = -1  
инициализация массивов горизонтов  
Верх = 0 Низ = Вэкран  
Вычисление функции на каждой плоскости z = const, начиная с ближайшей к наблюдателю плоскости  
Zmax for z = Zmax to Zmin step - Zшаг

инициализация предыдущих значений по х и y:  
Хпред и Yпред  
Хпред = Xmin   
Yпред = f(Xmin, z)  
если используется видовое преобразование, то его нужно применить к Хпред, Yпред, z в данной точке обработка левого бокового ребра   
call Обрребра (Хпред, Yпред, Хлев, Yлев; Верх, Низ)  
call Видимость (Хпред, Yпред, Верх, Низ; Пфлаг)  
для каждой точки на кривой, лежащей в плоскости z = const  
for х = Ymin to Xmax step Хшаг

y = f(x, z)   
если используется видовое преобразование, то его нужно применить к данной точке проверка видимости текущей точки и заполнение соответствующего массива горизонта  
call Видимость (х, y, Верх, Низ; Тфлаг)  
If Тфлаг = Пфлаг then

if Тфлаг = 1 или Тфлаг = -1 then

Draw (Хпред, Yпред, х, у)  
call Горизонт (Хпред, Yпред, х, у; Верх, Низ)

else   
end If  
если видимость изменилась, то вычисляется пересечение и заполняется массив горизонта   
else

if Тфлаг = 0 then

if Пфлаг = 1 then

call Пересечение(Хпред,Yпрел, х, у, Верх; Xi, Yi)

else

call Пересечение(Хпред, Yпред, х, у, Низ; Xi, Yi)

end If

Draw(Хпред, Yпред, Xi, Yi)   
call Горизонт(Хпред, Yпред, Xi, Yi; Верх, Низ)   
else

if Тфлаг = 1 then

if Пфлаг = 0 then

call Пересечение(Хпред, Yпред, х, у, Верх; Xi, Yi)  
Draw(Xi, Yi, х, у)   
call Горизонт(Xi, Yi, х, у; Верх, Низ)

else

call Пересечение(Хпред, Yпред, х, у, Низ; Xi, Yi)  
Draw(Хпред, Yпред, Xi, Yi)  
call Горизонт(Хпред, Yпред, Xi, Yi; Верх, Низ)   
call Пересечение(Хпред, Yпред, х, у, Верх; Xi, Yi)   
Draw(Xi, Yi, х, у)   
call Горизонт(Xi, Yi, х, у; Верх, Низ)

end if

else

if Пфлаг = 0 then

call Пересечение(Хпред, Yпред, х, у, Верх; Xi, Yi)  
Draw (Xi, Yi, х, у)  
call Горизонт(Xi, Yi, х, у; Верх, Низ)

else

call Пересечение(Хпред, Yпред, х, у. Верх; Xi, Yi) Draw(Хпред, Yпред, Xi, Yi) call Горизонт(Хпред, Yпред, Xi, Yi; Верх,Низ) call Пересечение(Хпред, Yпред, x, y, Низ; Xi, Yi) Draw (Xi, Yi, x, y) call Горизонт (Xi, Yi, x, у; Верх, Низ)

end if

end if

end if

end if

Вновь инициализировать Пфлаг, Хпред, Yпред   
Пфлаг = Тфлаг   
Хпред = x   
Yпред = у

next х  
Обработка правого концевого ребра   
call Обрребра(х, у, Хправ, Yправ; Верх, Низ)

next z  
finish

подпрограмма обработки бокового ребра  
subroutine Обрребра(х, у, Хребра, Yребра; Верх, Низ)

если Хребра = -1, то встречена первая кривая и ребро не создается  
If Хребра = -1 then 1  
call Горизонт(Хребра, Yребра, х, у; Верх, Низ)  
1: Хребра = х  
Yребра = у  
return

подпрограмма определения видимости точки  
subroutine Видимость(х, у. Верх, Низ; Тфлаг)  
видимость точки определяется по отношению к верхнему и нижнему плавающим горизонтам. Если точка лежит на самом горизонте, то она считается видимой,   
Tфлаг = 0, если точка невидима  
= 1, если она видима и выше верхнего горизонта   
= -1, если она видима и ниже нижнего горизонта х считается целой

If у < Верх (х) and у > Низ (х) then Тфлаг = 0  
If у >= Верх (х) then Тфлаг = 1  
if у <= Низ (х) then Тфлаг = -1  
return

подпрограмма заполнения массивов плавающих горизонтов   
subroutine Горизонт(XI, YI, X2, Y2; Верх, Низ)  
Эта подпрограмма использует линейную интерполяцию для заполнения массивов горизонтов между X1 и X2  
Мах(а, b) определяет большее из а и b  
Мin(а, b) определяет меньшее из а и b

проверка вертикальности наклона   
If (X2 - X1) = 0 then

Верх (X2) = Мах (Верх (X2), Y2)   
Низ (X2) = Мin (Низ (X2), Y2)

else

Наклон = (Y2 - Y1)/(Х2 - X1)   
for х = X1 to X2 step 1  
y = Наклон \* (х - X1) + Y1   
Верх (х) = Мах(Верх (х), у)  
Низ (х) = Мin(Низ (х), у)   
next х

end If   
return

подпрограмма вычисления пересечения текущей кривой с горизонтом   
subroutine Пересечение(X1, Y1, X2, Y2), Массив; Xi, Yi)   
Эта процедура вычисляет пересечение двух отрезков прямых.

Массив содержит информацию о соответствующем горизонте  
Sign - функция, принимающая значения -1,0, 1, если знак ее аргумента <0, =0, >0

проверка бесконечности наклона   
If (X2 - X1) = 0 then

Xi = X2  
Yi = Массив (X2)

else

вычисление пересечения  
обход начинается с самой левой используемой точки пересечение считается обнаруженным, когда изменяется знак разности значений у   
Наклон = (Y2 - Y1)/(X2 - X1)   
Ysign = Sign(Y1 + Наклон - Массив (X1 + 1))   
Csign = Ysign   
Yi= Y1 + Наклон   
Xi = X1 + 1  
while(Csign = Ysign)

Yi = Yi + Наклон  
Xi = Xi + 1  
Csign = Sign(Yi - Массив (Xi))

end while выбирается ближайшее целое число   
If |Yi - Наклон - Массив (Xi - 1)| <= |Yi - Массив (Xi)| then

Yi = Yi - Наклон  
Xi = Xi - 1

end if

end if

return

**Задание**

Программно реализовать алгоритм плавающего горизонта.

# Лабораторная работа №9: Алгоритм, использующий z-буфер

**Алгоритм z-буфера**

Заполнить буфер кадра фоновым значением интенсивности или цвета.  
Заполнить z -буфер минимальным значением z.  
Преобразовать каждый многоугольник в растровую форму в произвольном порядке.  
Для каждого Пиксел(x,y) в многоугольнике вычислить его глубину z(x,y).  
Сравнить глубину z(х,у) со значением Zбуфер(х,у), хранящимся в z-буфере в этой же позиции.  
Если z(х, у) > Zбуфер (х,у), то записать атрибут этого многоугольника (интенсивность, цвет и т. п.) в буфер кадра и заменить Zбуфер(х,у) на z(х,у).  
В противном случае никаких действий не производить.

В качестве предварительного шага там, где это целесообразно, применяется удаление нелицевых граней .

**Задание**

Программно реализовать алгоритм z-буфера.

**Лабораторная работа №10: Алгоритм трассировки лучей для простых непрозрачных поверхностей**

Алгоритм трассировки лучей для простых непрозрачных поверхностей можно представить следующим образом:

Создать список объектов, содержащий, по меньшей мере, следующую информацию:

Полное описание объекта: тип, поверхность, характеристики и т.д.

Описание сферической оболочки: центр и радиус.

Флаг прямоугольной оболочки. Если этот флаг поднят,

то будет выполнен габаритный тест с прямоугольной оболочкой,

если же опущен, то тест выполняться не будет. Заметим, что габаритный тест необходим не для всех объектов, например для сферы он не нужен.

Описание прямоугольной оболочки: xmin,xmax, ymin,ymax, zmin,zmax

Для каждого трассируемого луча:

Выполнить для каждого объекта трехмерный тест со сферической оболочкой в исходной системе координат. Если луч пересекает эту сферу, то занести объект в список активных объектов.

Если список активных объектов пуст, то изобразить данный пиксел с фоновым значением интенсивности и продолжить работу. В противном случае, перенести и повернуть луч так, чтобы он совместился с осью z. Запомнить это комбинированное преобразование.

Для каждого объекта из списка активных объектов:

Если флаг прямоугольной оболочки поднят, преобразовать, используя комбинированное преобразование, эту оболочку в систему координат, в которой находится луч, и выполнить соответствующий тест. Если пересечения с лучом нет, то перейти к следующему объекту. В противном случае преобразовать, используя комбинированное преобразование, объект в систему координат, в которой находится луч, и определить его пересечения с лучом, если они существуют. Занести все пересечения в список пересечений.

Если список пересечений пуст, то изобразить данный пиксел с фоновым значением интенсивности.

В противном случае определить zmax для списка пересечений.

Вычислить преобразование, обратное комбинированному преобразованию.

Используя это обратное преобразование, определить точку пересечения в исходной системе координат.

Изобразить данный пиксел, используя атрибуты пересеченного объекта и соответствующую модель освещенности.

**Задание**

Программно реализовать алгоритм трассировки лучей.