**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ВТ**

отчет

**по лабораторной работе №5**

**по дисциплине «Компьютерная графика»**

Тема: **Исследование алгоритмов выявления видимости сложных сцен**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 6306 |  | Гордиенко М. Е. |
|  |  | Пустовойтова А. А. |
| Преподаватель |  | Матвеева И. В. |

Санкт-Петербург

2019

**Цель работы.**

Исследовать методы выявления видимости сложных сцен.

**Задание.**

Обеспечить реализацию раскраски поверхности, сформированной при выполнении работы по теме 3. При этом лицевая и оборотная сторона должны окрашиваться в разные цвета. Обеспечить поворот окрашенной поверхности вокруг осей *X* и *Y*.

**Теория.**

### УДАЛЕНИЕ СКРЫТЫХ ЛИНИЙ И ПОВЕРХНОСТЕЙ

**Классификация методов удаления невидимых частей**

Методы удаления невидимых частей сцены можно классифицировать:

1. По выбору удаляемых частей:  
   удаление невидимых линий, ребер, поверхностей, объемов.
2. По порядку обработки элементов сцены:  
   удаление в произвольном порядке и в порядке, определяемом процессом визуализации.
3. По системе координат:
   * алгоритмы работающие в пространстве объектов, когда каждая из N граней объекта сравнивается с остальными N-1 гранями (объем вычислений растет как N2),
   * алгоритмы работающие в пространстве изображения, когда для каждого пиксела изображения определяется какая из N граней объекта видна (при разрешении экрана M×M объем вычислений растет как M2 ×N).

**Алгоритмы удаления линий**

Применение - представление каркасных моделей. При этом не используется основное ценное качество растрового дисплея - возможность закраски поверхностей. В этой связи основная область применения - векторные устройства, но могут применяться и в растровых для ускорения процесса визуализации.

Наиболее известный ранний алгоритм - алгоритм Робертса (1963 г.). Работает с только выпуклыми телами в пространстве объектов. Каждый объект сцены представляется многогранным телом, полученным в результате пересечения плоскостей. Т.е. тело описывается списком граней, состоящих из ребер, которые в свою очередь образованы вершинами.

Вначале из описания каждого тела удаляются нелицевые плоскости, экранированные самим телом. Затем каждое из ребер сравнивается с каждым телом для определения видимости или невидимости. Т.е. объем вычислений растет как квадрат числа объектов в сцене. Наконец вычисляются новые ребра, полученные при протыкании телами друг друга.

**Алгоритм удаления поверхностей с Z-буфером**

Алгоритм предложен Эдом Кэтмулом и представляет собой обобщение буфера кадра. Обычный буфер кадра хранит коды цвета для каждого пиксела в пространстве изображения. Идея алгоритма состоит в том, чтобы для каждого пиксела дополнительно хранить еще и координату Z или глубину. При занесении очередного пиксела в буфер кадра значение его Z-координаты сравнивается с Z-координатой пиксела, который уже находится в буфере. Если Z-координата нового пиксела больше, чем координата старого, т.е. он ближе к наблюдателю, то атрибуты нового пиксела и его Z-координата заносятся в буфер, если нет, то ни чего не делается.

Этот алгоритм наиболее простой из всех алгоритмов удаления невидимых поверхностей, но требует большого объема памяти. Данные о глубине для реалистичности изображения обычно достаточно иметь с разрядностью порядка 20 бит. В этом случае при изображении нормального телевизионного размера в 768×576 пикселов для хранения Z-координат необходим объем памяти порядка 1 Мбайта. Суммарный объем памяти при 3 байтах для значений RGB составит более 2.3 Мбайта.

Время работы алгоритма не зависит от сложности сцены. Многоугольники, составляющие сцену, могут обрабатываться в произвольном порядке. Для сокращения затрат времени нелицевые многоугольники могут быть удалены. По сути дела алгоритм с Z-буфером - некоторая модификация уже рассмотренного алгоритма заливки многоугольника. Если используется построчный алгоритм заливки, то легко сделать пошаговое вычисление Z-координаты очередного пиксела, дополнительно храня Z-координаты его вершин и вычисляя приращение dz Z-координаты при перемещении вдоль X на dx, равное 1. Если известно уравнение плоскости, в которой лежит обрабатываемый многоугольник, то можно обойтись без хранения Z-координат вершин. Пусть уравнение плоскости имеет вид:

|  |  |
| --- | --- |
| |  | | --- | | A·x + B·y + C·z + D = 0. | |

Тогда при C не равном нулю

|  |  |
| --- | --- |
| |  | | --- | | z = -(A·x + B·y + D)/C | |

Найдем приращение Z-координаты пиксела при шаге по X на dx, помня, что Y очередной обрабатываемой строки - константа.

|  |  |
| --- | --- |
| |  | | --- | | dz = -(A·(x+dx) + D)/C + (A·x + D)/C = -A·dx/C | |

но dx = 1, поэтому

|  |  |
| --- | --- |
| |  | | --- | | dz = -A/C. | |

Основной недостаток алгоритма с Z-буфером - дополнительные затраты памяти. Для их уменьшения можно разбивать изображение на несколько прямоугольников или полос. В пределе можно использовать Z-буфер в виде одной строки. Понятно, что это приведет к увеличению времени, так как каждый прямоугольник будет обрабатываться столько раз, на сколько областей разбито пространство изображения. Уменьшение затрат времени в этом случае может быть обеспечено предварительной сортировкой многоугольников на плоскости.

Другие недостатки алгоритма с Z-буфером заключаются в том, что так как пикселы в буфер заносятся в произвольном порядке, то возникают трудности с реализацией эффектов прозрачности или просвечивания и устранением лестничного эффекта с использованием предфильтрации, когда каждый пиксел экрана трактуется как точка конечного размера и его атрибуты устанавливаются в зависимости от того какая часть пиксела изображения попадает в пиксел экрана. Но другой подход к устранению лестничного эффекта, основанный на постфильтрации - усреднении значений пиксела с использованием изображения с большим разрешением реализуется сравнительно просто за счет увеличения расхода памяти (и времени). В этом случае используются два метода. Первый состоит в том, что увеличивается разрешение только кадрового буфера, хранящего атрибуты пикселов, а разрешение Z-буфера делается совпадающим с размерами пространства изображения. Глубина изображения вычисляется только для центра группы усредняемых пикселов. Это метод неприменим, когда расстояние до наблюдателя имитируется изменением интенсивности пикселов. Во втором методе и кадровый и Z буфера имеют увеличенное разрешение и усредняются атрибуты пиксела, так и его глубина.

Общая схема алгоритма с Z-буфером:

**Построчный алгоритм с Z-буфером**

Рассмотрим теперь алгоритм с Z-буфером размером в одну строку, который представляет собой обобщение алгоритма построчной заливки многоугольника, представленный в процедурах V\_FP0 и V\_FP1 в приложениях. Модификация должна учесть то, что для каждой строки сканирования теперь может обрабатываться не один многоугольник.

Общая схема такого алгоритма следующая:

1. Подготовка данных.  
   Для каждого многоугольника определить максимальную Y-координату.  
   Занести многоугольник в группу многоугольников, соответствующую данной Y-координате.
2. Собственно заливка.

**Алгоритм разбиения области Варнока**

Алгоритм работает в пространстве изображения и анализирует область на экране дисплея (окно) на наличие в них видимых элементов. Если в окне нет изображения, то оно просто закрашивается фоном. Если же в окне имеется элемент, то проверяется достаточно ли он прост для визуализации. Если объект сложный, то окно разбивается на более мелкие, для каждого из которых выполняется тест на отсутствие и/или простоту изображения. Рекурсивный процесс разбиения может продолжаться до тех пор пока не будет достигнут предел разрешения экрана.

Можно выделить 4 случая взаимного расположения окна и многоугольника (рис. [1](http://ermak.cs.nstu.ru/kg_rivs/graf08.htm#r11_01)):  
многоугольник целиком вне окна,  
многоугольник целиком внутри окна,  
многоугольник пересекает окно,  
многоугольник охватывает окно.

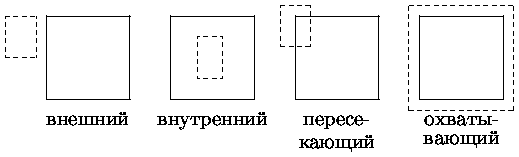


Рис. 1: Соотношения между окном экрана (сплошная рамка) и многоугольником (штриховая рамка)

В четырех случаях можно сразу принять решение о правилах закраски области экрана:

В любых других случаях процесс разбиения окна продолжается. Легко видеть, что при растре 1024×1024 и делении стороны окна пополам требуется не более 10 разбиений. Если достигнуто максимальное разбиение, но не обнаружено ни одного из приведенных выше четырех случаев, то для точки с центром в полученном минимальном окне (размером в пиксел) вычисляются глубины оставшихся многоугольников и закраску определяет многоугольник, наиболее близкий к наблюдателю. При этом для устранения лестничного эффекта можно выполнить дополнительные разбиения и закрасить пиксел с учетом всех многоугольников, видимых в минимальном окне.

Первые три случая идентифицируются легко. Последний же случай фактически сводится к поиску охватывающего многоугольника, перекрывающего все остальные многоугольники, связанные с окном. Проверка на такой многоугольник может быть выполнена следующим образом: в угловых точках окна вычисляются Z-координаты для всех многоугольников, связанных с окном. Если все четыре такие Z-координаты охватывающего многоугольника ближе к наблюдателю, чем все остальные, то окно закрашивается цветом соответствующего охватывающего многоугольника. Если же нет, то мы имеем сложный случай и разбиение следует продолжить.

Очевидно, что после разбиения окна охватывающие и внешние многоугольники наследуются от исходного окна. Поэтому необходимо проверять лишь внутренние и пересекающие многоугольники.

Из изложенного ясно, что важной частью алгоритма является определение расположения многоугольника относительно окна.

Проверка на то что многоугольник внешний или внутренний относительно окна для случая прямоугольных окон легко реализуется использованием прямоугольной оболочки многоугольника и сравнением координат. Для внутреннего многоугольника должны одновременно выполняться условия:

|  |  |
| --- | --- |
| |  | | --- | | Xmin  Wл    и    Xmax  Wп    и    Ymin  Wн    и    Ymax  Wв, | |

|  |  |  |  |
| --- | --- | --- | --- |
| здесь | Xmin,Xmax,Ymin,Ymax | - | ребра оболочки |
|  | Wл, Wп, Wн, Wв | - | ребра окна |

Для внешнего многоугольника достаточно выполнение любого из следующих условий:

|  |  |
| --- | --- |
| |  | | --- | | Xmin > Xп,    Xmax < Wл,    Ymin > Wв,     Ymax < Wн | |

Таким способом внешний многоугольник, охватывающий угол окна не будет идентифицирован как внешний (см. рис. [2](http://ermak.cs.nstu.ru/kg_rivs/graf08.htm#r11_02)).

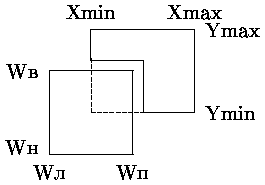


Рис. 2: Ошибочное определение внешнего многоугольника как пересекающего при использовании прямоугольной оболочки

Проверка на пересечение окна многоугольником может быть выполнена проверкой на расположение всех вершин окна по одну сторону от прямой, на которой расположено ребро многоугольника. Пусть ребро многоугольника задано точками P1(x1,y1,z1) и P2(x2,y2,z2), а очередная вершина окна задается точкой P3(x3,y3,z3). Векторное произведение вектора **P1P3** на вектор **P1P2**, равное (x3-x1)(y2-y1) - (y3-y1)(x2-x1) будет меньше 0, равно 0 или больше 0, если вершина лежит слева, на или справа от прямой P1P2. Если знаки различны, то окно и многоугольник пересекаются. Если же все знаки одинаковы, то окно лежит по одну сторону от ребра, т.е. многоугольник может быть либо внешним, либо охватывающим.

Вернемся к примеру [2](http://ermak.cs.nstu.ru/kg_rivs/graf08.htm#r11_02). Такой многоугольник рассмотренными тестами не был идентифицирован ни как внутренний ни как пересекающий. Т.е. он может быть либо внешним, либо охватывающим. Для завершающей классификации может использоваться тест с подсчетом угла, рассматривавшийся ранее для определения нахождения точки внутри/вне многоугольника. В этом тесте вычисляется суммарный угол, на который повернется луч, исходящий из некоторой точки окна (обычно центра), при последовательном обходе вершин многоугольника.

Если суммарный угол равен 0, то многоугольник - внешний. Если же угол равен N×360, то многоугольник охватывает окно N раз. Простейшая иллюстрация этого теста приведена на рис. [3](http://ermak.cs.nstu.ru/kg_rivs/graf08.htm#r11_03).

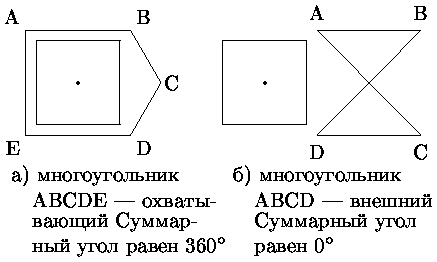


Рис. 3: Тест на охватывающий/внешний многоугольник

**Построчный алгоритм Уоткинса**

В алгоритмах построчного сканирования результирующее изображение генерируется построчно причем, подобно ранее рассмотренному алгоритму построчной заливки многоугольника, используется связность соседних растровых строк изображения. Отличие состоит в том, что учитываются все, а не один многоугольник.

Алгоритм работает в пространстве изображения с окном высотой в одну строку и шириной в экран, тем самым трехмерная задача сводится к двумерной.

Последовательность шагов алгоритма:  
построение списка ребер,  
построение списка многоугольников,

**Алгоритм трассировки лучей**

При рассмотрении этого алгоритма предполагается, что наблюдатель находится на положительной полуоси Z, а экран дисплея перпендикулярен оси Z и располагается между объектом и наблюдателем.

Удаление невидимых (скрытых) поверхностей в алгоритме трассировки лучей выполняется следующим образом:  
сцена преобразуется в пространство изображения,

Ясно, что наиболее важная часть алгоритма - процедура определения пересечения, которая в принципе выполняется Rx×Ry×N раз (здесь Rx,Ry - разрешение дисплея по X и Y, соответственно, а N - количество многоугольников в сцене).

Очевидно, что повышение эффективности может достигаться сокращением времени вычисления пересечений и избавлением от ненужных вычислений. Последнее обеспечивается использованием геометрически простой оболочки, объемлющей объект - если луч не пересекает оболочку, то не нужно вычислять пересечения с ним многоугольников, составляющих исследуемый объект.

При использовании прямоугольной оболочки определяется преобразование, совмещающее луч с осью Z. Оболочка подвергается этому преобразованию, а затем попарно сравниваются знаки Xmin с Xmax и Ymin с Ymax. Если они различны, то есть пересечение луча с оболочкой (см. рис. [4](http://ermak.cs.nstu.ru/kg_rivs/graf08.htm#r11_04))

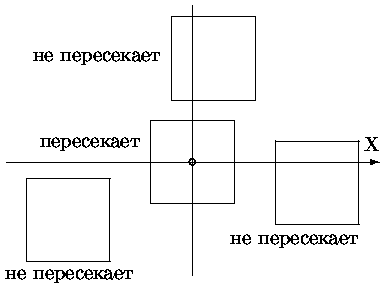


Рис. 4: Определение пересечения луча и оболочки

При использовании сферической оболочки для определения пересечения луча со сферой достаточно сосчитать расстояние от луча до центра сферы. Если оно больше радиуса, то пересечения нет. Параметрическое уравнение луча, проходящего через две точки P1(x1,y1,z1) и P2(x2,y2,z2), имеет вид:

|  |  |
| --- | --- |
| |  | | --- | | P(t) = P1 + (P2 - P1)×t | |

Минимальное расстояние от точки центра сферы P0(x0,y0,z0) до луча равно:

|  |  |
| --- | --- |
| |  | | --- | | d2 = (x-x0)2 + (y-y0)2 + (z-z0)2 | |

Этому соответствует значение t:

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | t = - | (x2-x1)·(x1-x0) + (y2-y1)·(y1-y0) + (z2-z1)·(z1-z0)  (x2-x1)2 + (y2-y1)2 + (z2-z1)2 | . | |

Если d2 > R2, то луч не пересекает объекты, заключенные в оболочку.

Дальнейшее сокращение расчетов пересечений основывается на использовании групп пространственно связанных объектов. Каждая такая группа окружается общей оболочкой. Получается иерархическая последовательность оболочек, вложенная в общую оболочку для всей сцены. Если луч не пересекает какую-либо оболочку, то из рассмотрения исключаются все оболочки, вложенные в нее и, следовательно, объекты. Если же луч пересекает некоторую оболочку, то рекурсивно анализируются все оболочки вложенные в нее.

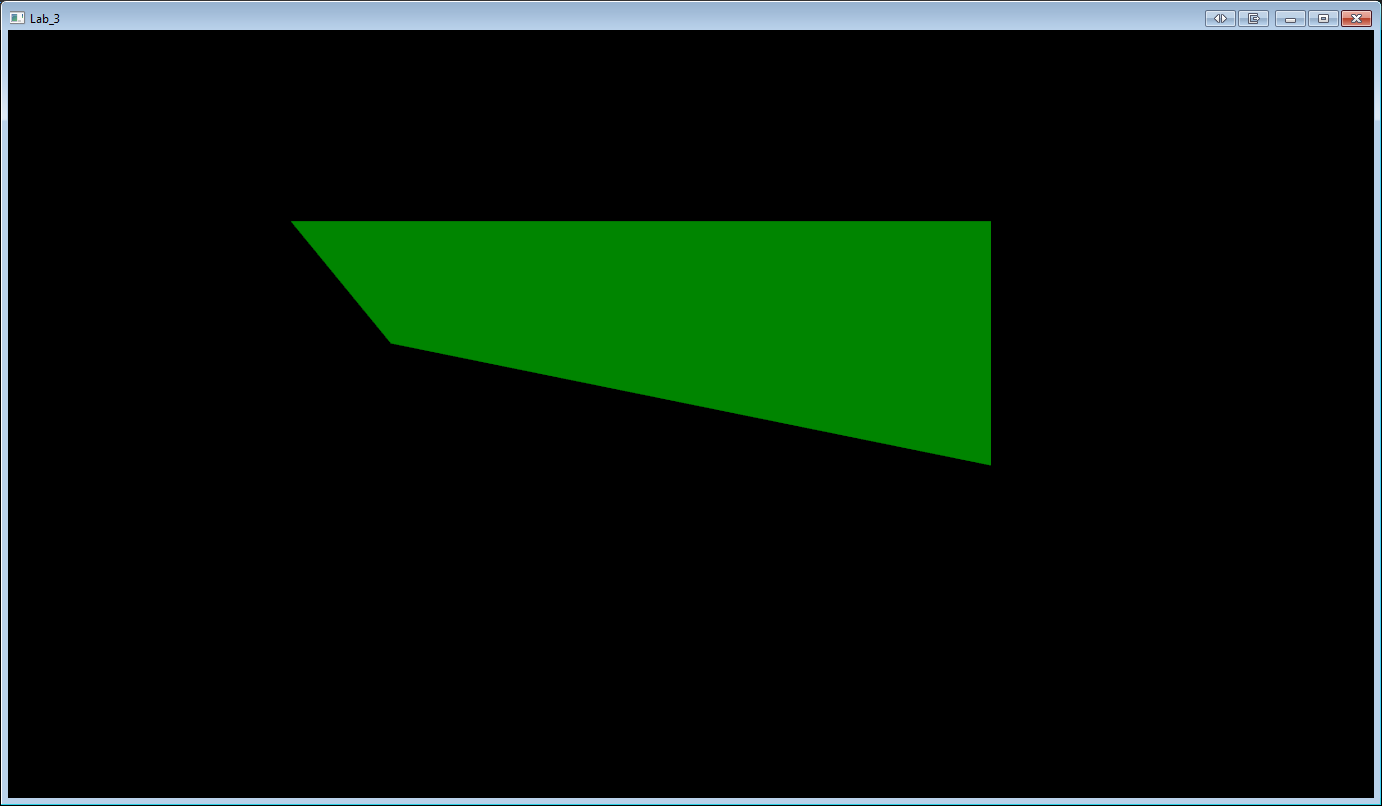
Наряду с вложенными оболочками для сокращения расчетов пересечений используется отложенное вычисление пересечений с объектами. Если обнаруживается, что объект пересекается лучом, то он заносится в специальный список пересеченных. После завершения обработки всех объектов сцены объекты, попавшие в список пересеченных упорядочиваются по глубине. Заведомо невидимые отбрасываются а для оставшихся выполняется расчет пересечений и отображается точка пересечения наиболее близкая к наблюдателю.

Дополнительное сокращение объема вычислений может достигаться отбрасыванием нелицевых граней, учетов связности строк растрового разложения и т.д.

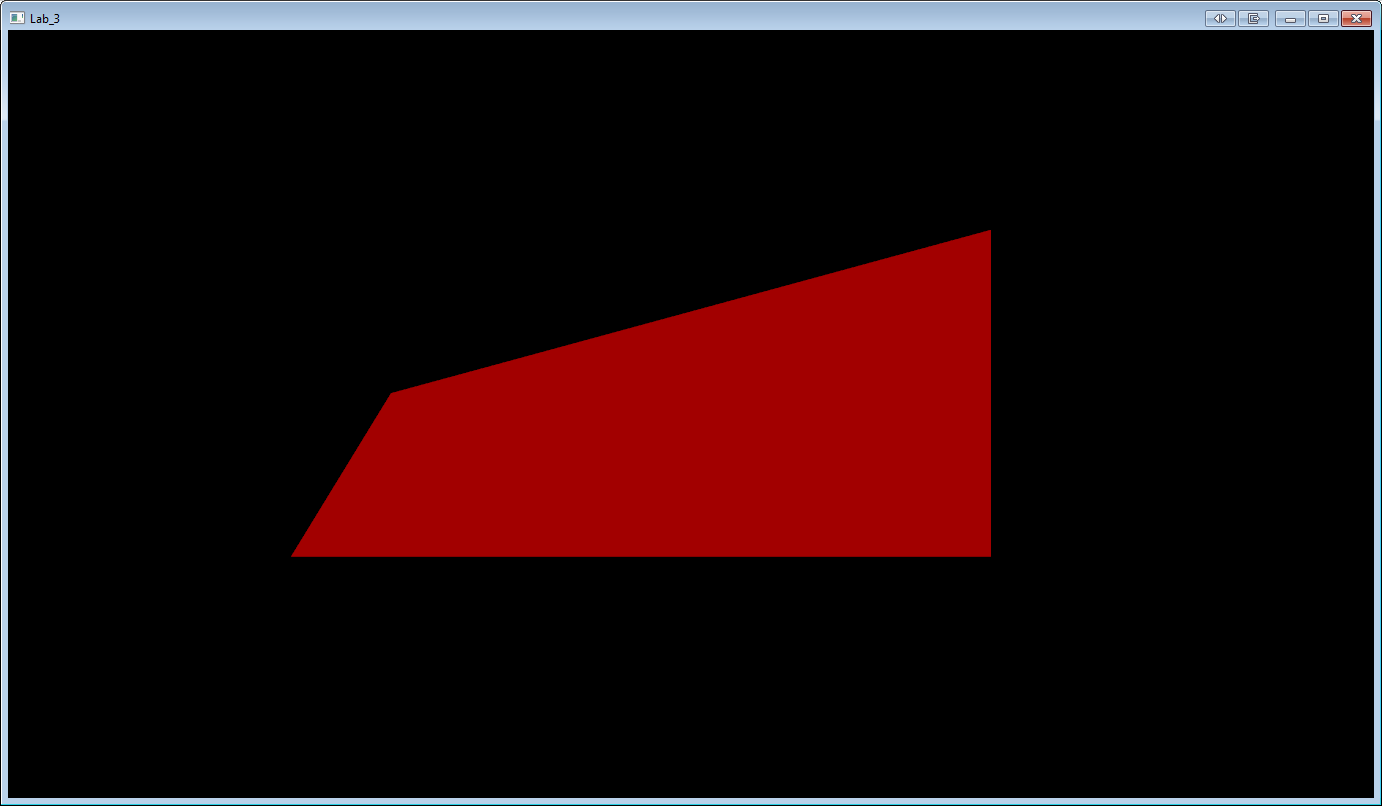
Для сокращения времени вычислений собственно пересечений предложено достаточно много алгоритмов, упрощающих вычисления для определенной формы задания поверхностей.

**Описание реализации.**

Данная работа совмещена с работой №3, соответственно основные алгоритмы и интерфейс в ней тот же.



Единственное отличие состоит в методе раскраски различных сторон поверхности в разные цвета.



**Приложение.**

**Код программы.**

Метод вычисления вектора нормали плоскости.

void FourPointLinearPlane::setNorm() {

Point a(points(0, 1) - points(0, 0)),

b(points(1, 0) - points(0, 0));

float x = a.GetPosition().y \* b.GetPosition().z - a.GetPosition().z \* b.GetPosition().y,

y = a.GetPosition().z \* b.GetPosition().x - a.GetPosition().x \* b.GetPosition().z,

z = a.GetPosition().x \* b.GetPosition().y - a.GetPosition().y \* b.GetPosition().x;

norm = sf::Vector3f(x, y, z);

}

Метод отрисовки плоскости (в зависимости от косинуса угла между вектором нормали плоскости и вектором взгляда плоскость окрашивается в различные цвета):

void FourPointLinearPlane::Draw(sf::RenderWindow\* window, sf::Color color, sf::Color color2) {

sf::ConvexShape shape;

shape.setPointCount(4);

float cos = norm.z / (sqrt(pow(norm.x, 2) + pow(norm.y, 2) + pow(norm.z, 2)));

if (cos < 0)

shape.setFillColor(sf::Color(color.r - (color.r \* (1 - abs(cos)) / 1.25f), color.g - (color.g \* (1 - abs(cos)) / 1.25f), color.b - (color.b \* (1 - abs(cos)) / 1.25f)));

else shape.setFillColor(sf::Color(color2.r - (color2.r \* (1 - abs(cos)) / 1.25f), color2.g - (color2.g \* (1 - abs(cos)) / 1.25f), color2.b - (color2.b \* (1 - abs(cos)) / 1.25f)));

shape.setPoint(0, points(0, 0).Isometric(window->getSize().x, window->getSize().y));

shape.setPoint(1, points(0, 1).Isometric(window->getSize().x, window->getSize().y));

shape.setPoint(2, points(1, 1).Isometric(window->getSize().x, window->getSize().y));

shape.setPoint(3, points(1, 0).Isometric(window->getSize().x, window->getSize().y));

window->draw(shape);