Компьютерная графика. Лабораторные работы.

1.1. Начало программирования на C#.

Для языка C# нет какой-то специальной библиотеки типов, в качестве таковой он используетобщую систему типов (CTS)платформы .NET, которая структуирует многочисленные типы (а их более 4000) в группы – пространства имен. Необходимость структурирования связана с тем, что некоторые типы, имеющие схожую функциональность, но предназначенные для разных применений, имеют одинаковые имена

Следует заметить, что многочисленные эксперты, управляющие созданием того или иного приложения, автоматически выбирают минимально необходимые пространства имен. Если, например, вы создаете консольное приложение, эксперт объявит ссылку на пространство имен System и только на него. При создании приложения для работы с графическим интерфейсом, кроме того, появятся ссылки на пространства:

System - корневое пространство имен, содержащее класс и множество низкоуровневых классов для работы с простыми типами, выполнения математических операций, сбора мусора и т. п.

System.Collections - контейнерные классы, такие как ArrayList, Queue, Stack, SortedList и т. п.

System.Data, System.Data.Common, System.Data.Oledb, System.Data.SqlClient - классы этих пространств предназначены для работы с базами данных

System.Drawing, System.Drawing.Drawing2D, System.Drawing.Printing - классы для примитивов графического интерфейса — растровых изображений, шрифтов, значков, поддержки печати

System.IO - классы, отвечающие за операции ввода-вывода

System.Net - классы, отвечающие за передачу данных по сети (запрос-ответ, создание сокетов и т. п.)

System.Security - в этом пространстве имен собраны классы, использующиеся для повышения безопасности при передаче данных (работа с разрешениями, криптография и т. п.)

System.Threading - это пространство имен для классов, которые работают с программными потоками.

System.Web - классы, используемые в веб-приложениях

System.Windows.Forms - Классы для работы с элементами интерфейса Windows — окнами, элементами управления и т. д.

System.XML - Множество классов для работы с данными в формате XML

В нашей разработке на языке C # используются лишь некоторые группы пространства имен, которые перечисленны ниже. Для обьявления ссылки на пространство имен используется оператор *using*

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
```

1.2. Написать программу для поворота отрезка прямой линии на произвольный угол относительно заданной точки

Во-первых: при повороте отрезка вокруг своей оси рекомендуем сразу учитывать угол поворота, длину нашего отрезка, начальные координаты, а так же угол поворота.

Во-вторых рисуем точку, относительно которой будем делать поворот.

Допустим мы обозначили наши начальные координаты по осям ox и oy, как x0 и y0 соответственно.

Воспользуемся формулой:

X=x0+r*Cos ((-α)*π/180°)

У=y0+ r*Cos ((-α)*π/180°)

Где α - угол поворота, r – длина отрезка, а выражение Cos ((-α)*π/180°) перевод из градусов в радианы.

Далее советуем сделать цикл, чтобы постепенно увеличивать наш угол поворота до предела.

Матрица преобразования координат точки (x,y), при повороте на угол ***a*** относительно точки (x0,y0):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x0 \\ y0 \end{bmatrix} + \begin{bmatrix} x - x0 \\ y - y0 \end{bmatrix} \cdot \begin{bmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{bmatrix}$$

(x',y') - искомые координаты.

1.3. Реализовать процедуру отсечения. В заданном окне 20-50 концентрических прямоугольников, смещенных друг относительно друга на произвольный угол и на deltaR с использование процедуры отсечения

При отсечении будем использовать массивы вершин области отсечения и массивы нормалей к каждой стороне:

```
double[] Windx=new double
double[] Windy=new double;
double[] Wnormx=new double;
double[] Wnormy=new double;
```

Так же при сжатии прямоугольников существует коэффициент компрессии, обозначим его com:

```
com=(Convert.ToDouble(Compres.Text)/100)
```

Далее опишем поворот прямоугольника относительно каждого последующего:

```
TurnRec(ll,hh,120*Math.PI/180,Windx,Windy)
```

А так же нормали к текущей стороне:

```
NormRec(Windx,Windy,Wnormx,Wnormy)
```

Следующим выражением обозначим смещение прямоугольников:

```
dl=(int)(ll*com);
dh=(int)(hh*com);
TurnRec((ll-dh),(hh-dh),120*Math.PI/180+a1,Windx1,Windy1);
```

где dl – высота прямоугольника, а dh – ширина прямоугольника

Таким образом мы можем выразить уменьшения масштаба области отсечения для вложенных прямоугольников.

```
ll=ll-dl;
hh=hh-dh;
dl=(int)(ll*com);
dh=(int)(hh*com);
```

Далее нам остается всего лишь описать поворот и уменьшение либо увеличение прямоугольников, причем поворот легко отобразить с помощью следующих выражений:

```
Windx[0]=CenX+r*Math.Cos(ang);
Windy[0]=CenY+r*Math.Sin(ang);
Windx[1]=CenX+r*Math.Cos(ang+al);
Windy[1]=CenY+r*Math.Sin(ang+al);
Windx[2]=CenX+r*Math.Cos(ang+Math.PI);
Windy[2]=CenY+r*Math.Sin(ang+Math.PI);
Windx[3]=CenX+r*Math.Cos(ang+Math.PI+al);
Windy[3]=CenY+r*Math.Sin(ang+Math.PI+al);
```

Где ang – угол поворота.

Для отсечения воспользуемся алгоритмом Кируса-Бека, алгоритм построен следующим образом:

Ni - внутренняя нормаль к i-й граничной линии окна, а P = V1 - V0 - вектор, определяющий ориентацию отсекаемого отрезка, тогда ориентация отрезка относительно i-й стороны окна определяется знаком скалярного произведения.

Искомые значения параметров t0 и t1 точек пересечения инициализируются значениями 0 и 1, соответствующими началу и концу отсекаемого отрезка.

Затем в цикле для каждой i-й стороны окна отсечения вычисляются значения скалярных произведений, входящих в

(Ni * P) * t + Ni * Q = Pi * t + Qi.

Q(t) – вектор, начинающийся в начальной точке ребра окна и заканчивающийся в некоторой точке удлиненной линии.

Если очередное Pi равно 0, то отсекаемый отрезок либо вырожден в точку, либо параллелен i-й стороне окна. При этом достаточно проанализировать знак Qi. Если Qi < 0, то отрезок вне окна и отсечение закончено иначе рассматривается следующая сторона окна.

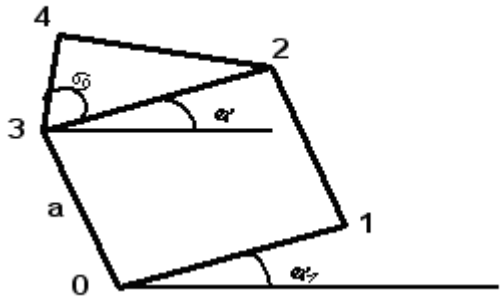
Если же Pi не равно 0, то можно вычислить значение параметра t для точки пересечения отсекаемого отрезка с i-й границей.

Если Pi < 0, это значит удлиненная линия направлена с внутренней на внешнюю стороны граничной линии, то ищутся значения параметра для конечной точки видимой части отрезка. Решение даст значение параметра t1 для конечной точки отсеченного отрезка.

Если же Pi > 0, т.е. удлиненная линия направлена с внешней на внутреннюю стороны граничной линии, то ищутся значения параметра для начальной точки видимой части отрезка. В этом случае определяется максимальное значение из всех получаемых решений.

Значения t0 и t1 используются для вычисления координат точек пересечения отрезка с окном.

1.4. Реализовать рекуррентную процедуру. (Написать вариант программы дерево Пифагора.



Каждый из прямоугольных треугольников в этом дереве имеет внутренний угол, равный 45°. Углы, задаваемые равными 45°, в общем случае будут задаваться случайным образом в пределах между (45 - *delta*)° и (45 + *delta*)°, где значение *delta* задается в качестве входного параметра вместе с параметром *n*, определяющим глубину рекурсии, с помощью которой мы сможем построить много таких «домиков», причем треугольник каждого «домика» является основанием для следующего прямоугольника. Необходимые точки пронумерованы последовательными числами 0, 1, 2, 3, 4. Координаты x_0, y_0 точки O задаются в обращении к функции. Для вычисления остальных точек вначале рассмотрим более простую ситуацию при $\varphi = 0$, то есть когда сторона 0 1 квадрата занимает горизонтальное положение.

В этом положении координаты точек определить очень просто. Они записываются в массивах *x* и *y*. Затем все конструкция поворачивается вокруг точки O на угол φ, результат поворота записывается в массивах *xx* и *yy*.

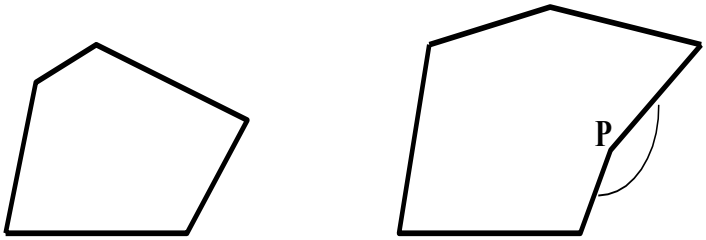
Координаты можно задать таким образом (углы `alp1 и alp указаны на рисунке`):

```
X0 = x + a * Cos(alp);
Y0 = y + a * Sin(alp));
X1 = x + √2 * a * Cos(alp + PI / 4));
Y1 = y + √2 * a * Sin(alp + PI / 4)));
X2 = x + a * Cos(alp + PI / 2));
Y2 = y + a * Sin(alp + PI / 2));


xb = x3 + a * Cos(alp1) * Cos(alp + alp1));
yb = y3 + a * Cos(alp1) * Sin(alp + alp1));
Причем для отрисовки для правой ветки:
Alp = alp1 + alp;
a1 = a * Cos(alp1));
Для левой ветки:
Alp = alp - PI / 2;
a1 = a * Cos(PI/2-alp1));
```

1.5 Написать программу, генерирующую полигон произвольной формы с n-вершинами (n=100).

При генерации полигона мы не используем никах сложных алгоритмов. Если внутренние углы при всех вершинах полигона меньше 180°, то такой полигон называется *выпуклым*. Если внутренний угол при вершине P больше 180°, такую вершину будем называть *невыпуклой*. Все другие вершины на рис. 3.9 - выпуклые. Если полигон имеет хотя бы одну невыпуклую вершину, то весь такой полигон будем называть невыпуклым.



Выпуклый полигон **Невыпуклый полигон**

Если A и B - две точки на границы выпуклого полигона, то и весь отрезок AB будет принадлежать полигону. Для невыпуклых полигонов это условие может не соблюдаться. Невыпуклость полигонов является источником сложностей, это же касается переменного числа вершин полигонов. По этой причине уделим большое внимание треугольникам. Вполне очевидно, что треугольники всегда имеют фиксированное число вершин и они обязательно выпуклые. Особый интерес к ним выявляется в связи с рассмотрением произвольных полигонов, поскольку любой полигон может быть разбит на конечное число треугольников.

В качестве примера ниже приведена часть кода программы, которая переводит систему в полярные координаты, r - фиксированная по заданию.

Функция, генерирующая полигон из точки с начальными координатами (x,y) и длиной отрезка AB double aa=0,a1,r=100

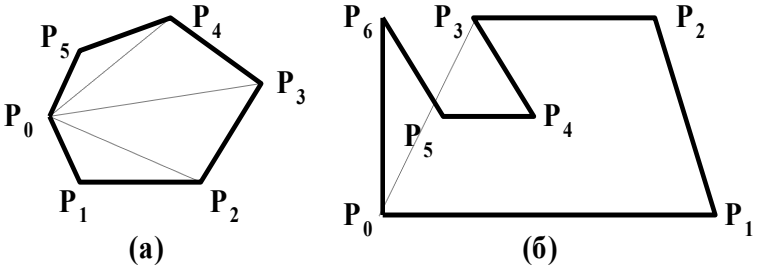
```
int[] Polx=new int[n];
int[] Poly=new int[n];
```

Переводим вполярные координаты:

```
Polx[i]=CenX+(int)(r*Math.Cos(aa+a1));
Poly[i]=CenY+(int)(r*Math.Sin(aa+a1));
```

1.6. Написать программу для разбиения полигонов на треугольники (треугольники раскрасить разным цветом).

Операция разбивки выпуклого полигона на треугольники чрезвычайно проста. Если вершины полигона пронумеровать последовательно P_0, P_1,...,P_{n-1}, а затем вычертить диагонали P_0P_2, P_0P_3, ..., P_0P_{n-2}, то этого будет достаточно. В невыпуклом полигоне, как на рис. (б), этот простой способ работать не будет, поскольку некоторые из диагоналей P_0P_2, P_0P_3, ..., P_0P_{n-2} могут выходить за пределы полигона.



(а) - диагонали внутри полигона; (б) - диагональ $P_0 P_3$ использовать нельзя.

Составим теперь программу, которая будет считывать координаты вершин полигона и выполнит разбиение полигона на треугольники. Требуется, чтобы вершины были указаны обязательно в порядке обхода против часовой стрелки. Для полигона с n вершинами сначала указывается количество вершин n, затем последовательно перечисляются n пар координат всех вершин в порядке обхода полигона против часовой стрелки. В результате будет получен чертеж полигона с вычерченными диагоналями, полностью разбивающими весь полигон на треугольники. Перед вычерчиванием диагоналей необходимо удостовериться, что все диагонали лежат полностью внутри полигона.

Предположим, что P_{i-1}, P_i , P_{i+1} обозначают три соседние вершины, причем будем считать, что $P_{-1} = P_{n-1}$ и $P_n = P_0$. В этом случае P_i будет выпуклой вершиной тогда, и только тогда, когда три вершины P_{i-1}, P_i , P_{i+1} именно в этом порядке будут обходиться в направлении против часовой стрелки и так далее.

Технически это реализуется введением целочисленного массива v_0, ..., v_{m-1}, содержащего номера вершин оставшегося полигона. Вначале задаем $m=n$ и $v_i=i$ (i=0, 1,..., n-1). Каждый раз при отсечении треугольника число m уменьшается на единицу.

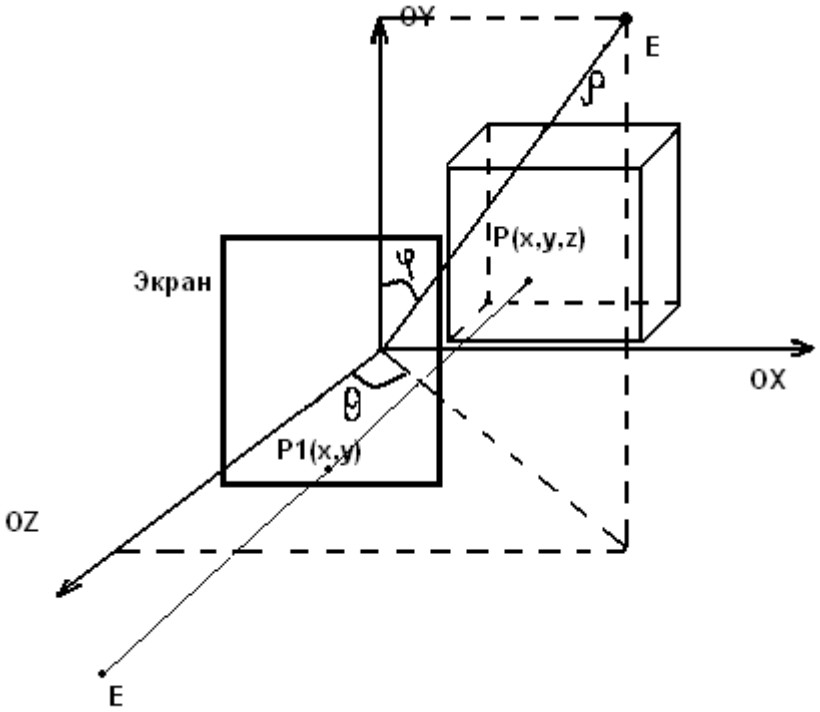
Из других проверок следует обратить внимание на:

– максимальное количество точек n, например $n \le 500$;
– минимальное и максимальное значения координат;
– ориентацию обхода точек в направлении против часовой стрелки.

Несмотря на их очевидную важность, большинство проверок здесь опущено и они оставлены для упражнений. С другой стороны, в программу включены некоторые специальные средства, которые, вообще говоря, можно опустить. Это относится к представлению диагоналей в виде штриховых линий вместо сплошных. Пусть все штрихи должны иметь одинаковую длину. Штриховая линия не должна начинаться или кончаться пробелом, в начале и в конце штрихи должны быть полной длины.

1.7 Написать программу для вычерчивания проволочной модели куба в трехмерном пространстве. Начало мировых координат выбирается в центре куба.

Пусть есть точка P(x,y,z) и необходимо построить точку P(x,y) на экране.



Существует два способа преобразования:

1. Видовое преобразование: точка P остается на месте, но система мировых координат переходит в систему видовых координат. X,Y,Z – Xe,Ye,Ze
2. Точное преобразование: точка P переходит в точку P1. Из видовых координат получаем двухмерные координаты. X,Y,Z – X,У

Xe=ρ * Sin(φ)Cos(Θ)
Ye=ρ * Sin(φ) Sin (Θ)
Ze=ρ * Cos(φ)

Финальная матрица получения видовых координат:

$$\begin{pmatrix} -\sin(\theta) & -\cos(\phi)\cdot\cos(\theta) & -\sin(\phi)\cdot\cos(\theta) & 0 \\ \cos(\theta) & -\cos(\phi)\cdot\sin(\theta) & -\sin(\phi)\cdot\sin(\theta) & 0 \\ 0 & \sin(\phi) & -\cos(\phi) & 0 \\ 0 & 0 & \rho & 1 \end{pmatrix}$$

Создание параллелепипеда:

```
xx[0]=l/2;  yy[0]=-h/2; zz[0]=-d/2;
xx[1]=l/2;  yy[1]=-h/2; zz[1]=d/2;
xx[2]=l/2;  yy[2]=h/2;  zz[2]=d/2;
xx[3]=l/2;  yy[3]=h/2;  zz[3]=-d/2;
xx[4]=-l/2; yy[4]=-h/2; zz[4]=-d/2;
xx[5]=-l/2; yy[5]=-h/2; zz[5]=d/2;
xx[6]=-l/2; yy[6]=h/2;  zz[6]=d/2;
xx[7]=-l/2; yy[7]=h/2;  zz[7]=-d/2;
```

1.8. Вычертить сложную фигуру (куб, цилиндр, пирамида).

Для наилучшего восприятия формы объекта необходимо иметь его изображение в трехмерном пространстве. Во многих случаях наглядное представление об объекте можно получить путем выполнения операций вращения и переноса, а также построения проекций. Введем однородные координаты. Точка в трехмерном пространстве [x y z] представится четырехмерным вектором [x y z 1] или [X Y Z H]. Преобразование из однородных координат описывается соотношениями

[X Y Z H]= [x y z 1] T

[x* y* z* 1] = [X/H Y/H Z/H 1]

где T - некоторая матрица преобразования.

$$T = \begin{bmatrix} a & b & c & p \\ d & e & f & g \\ h & i & j & r \\ l & m & n & s \end{bmatrix}$$

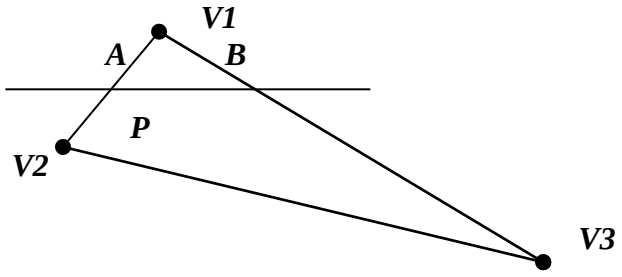
Эта матрица может быть представлена в виде 4 отдельных частей. Матрица 3x3 осуществляет линейное преобразование в виде изменения масштаба, сдвига и вращения. Матрица-строка 1x3 производит перенос, а матрица-столбец 3x1 - преобразование в перспективе. Последний скалярный элемент выполняет общее изменение масштаба. Полное преобразование, полученное путем воздействия на вектор положения матрицей 4X4 и нормализации преобразованного вектора, будем называть билинейным преобразованием. Оно обеспечивает выполнение комплекса операций сдвига, частичного изменения масштаба, вращения, отображения, переноса, а также изменения масштаба изображения в целом.

На данном этапе нам проще всего нарисовать параллелепипед, так как мы подробно рассмотрели в пункте 1.7, а дальше закрасить его.

Существуют три простейших метода закраски, дающих достаточно приемлемые результаты, опишем один из них - метод Гуро.

Этот метод обеспечивает непрерывность освещенности. Основная идея: заливка осуществляется с учетом линейной интерполяции яркости, вычисляется яркость только для вершин многоугольника.

Пусть задана плоская грань V_1 V_2 V_3, как показано на рисунке. Найдем значение освещенности в каждой ее вершине. Обозначим получившиеся значения через I_1, I_2, I_3. Рисуя грань V_1 V_2 V_3 построчно, находим значения освещенности в конце каждого горизонтального отрезка путем линейной интерполяции значений вдоль ребер. При рисовании очередного отрезка AB будем считать, интенсивность изменяется от I(A) до I(B) линейно.




```
for(i=0;i<=rad;i+=(float).5)
                        {
for(j=-(float)(Math.Sqrt(rad*rad-i*i));
j<=(float)(Math.Sqrt(rad*rad-i*i));j+=(float).5)
                              {
k=(float)(Math.Sqrt(rad*rad-i*i-j*j+.001));
coso=(S[0]*i+S[1]*j+S[2]*k)/(Math.Sqrt(S[0]*S[0]+S[1]*S[1]+S[2]*S[2])*Math.Sq
rt(i*i+j*j+k*k));
V=Math.Abs(255*Math.Pow(coso,3));
```

```
k=-(float)(Math.Sqrt(rad*rad-i*i-j*j+.001));
coso=(S[0]*i+S[1]*j+S[2]*k)/(Math.Sqrt(S[0]*S[0]+S[1]*S[1]+S[2]*S[2])*Math.Sq
rt(i*i+j*j+k*k));
V=Math.Abs(255*Math.Pow(coso,3));
```

1.9 Написать программу для моделирования освещения объектов по Фонгу.

Закон Фонга (закон зеркального отражения):

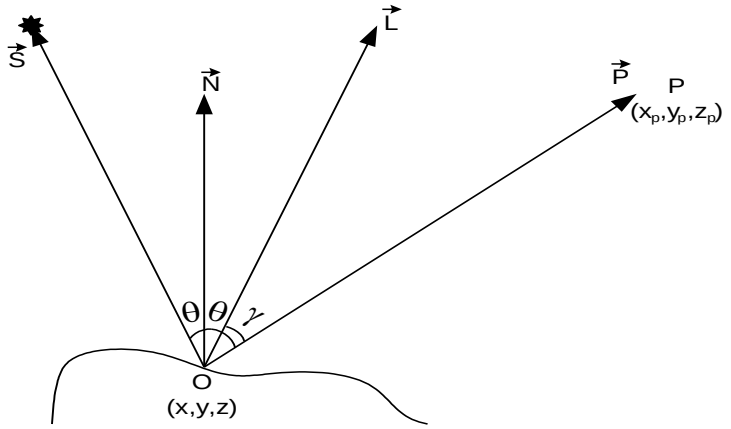


Рис.3.1.4

N - нормаль к поверхности в точке (x,y,z);
S - падающий луч от источника S;
L - отраженный луч света;
P - направление на наблюдателя P (x_P,y_P,z_P);
θ – угол падения и отражения;
γ – угол между отраженным лучом и направлением на наблюдателя.
Формула для определения зеркальной составляющей V, где n – степень зеркальности поверхности, $n \in [1;200]$:

$$V_\phi = E * I * \cos^n(\gamma)$$

Чем больше n тем больше зеркальные свойства поверхности:

$$\cos(\gamma) = \frac{\vec{L} \cdot P}{|L| \cdot |P|}$$

P = [x_P-x_0,y_P-y_0,z_P-z_0]
S = [x_S-x_0,y_S-y_0,z_S-z_0]
L = 2 * (N * S) * N-S
|L| = |N| = |S| = 1
Вектора N,S,L нормированные и лежат в одной плоскости, пусть I = const, тогда
$V=V_{max} * (E (1-\varepsilon- \varepsilon_\phi) * \eta + \varepsilon) + E_\phi * \varepsilon_\phi * \mu$
ε – доля рассеянного света,
ε_Φ – доля отраженного света, $\varepsilon_\phi \in [0;1]$; $0 \le \varepsilon + \varepsilon_\phi \le 1$,
μ – фотометрический коэффициент зеркального отражения,
η – фотометрический коэффициент диффузионного отражения.

$\mu = |\cos^{(n)}\gamma|$ - прожектор

$\mu = \frac{|\cos(n)_\gamma|}{r2} * r_{min}^2$ - точечный источник света

$\eta = \frac{\cos(\theta)}{\cos(\theta)}$ - прожектор

$\eta = \frac{}{r^2} * r_{min}^2$ - точечный источник света

E_ϕ – характеристика источника света, в общем случае $E_\phi \neq E$, если угол $\gamma > 90°$, то не надо учитывать зеркальную составляющую.

1.10 Написать программу для отображения сложного объекта в динамике (вращение, приближение, удаление, освещенность).

Так как освещенность по Фонгу мы рассмотрели выше в пункте 1.8, рисование сложной фигуры в трехмерном пространстве и ее закраску в пункте 1.7, то для выполнения этого задания достаточно лишь рассмотреть вращение, то есть поворот относительно некоторого угла и приближение или удаление.

Сначала рассмотрим поворот вокруг направляющего вектора на некоторый угол:

```
aX=Math.Cos(aXY*Math.PI/180);
aY=Math.Cos(aXZ*Math.PI/180);
aZ=Math.Cos(aYZ*Math.PI/180);
Ot=Ott*Math.PI/180; - некоторый угол поворота
```

Поворот на угол:

```
Buf[0]=xx[i]*(Math.Cos(Ot)+Math.Pow(aX,2)*(1-
Math.Cos(Ot)))+yy[i]*(aZ*Math.Sin(Ot)+aX*aY*(1-Math.Cos(Ot)))
                                        +zz[i]*(-aY*Math.Sin(Ot)+aX*aZ*(1-Math.Cos(Ot)));
Buf[1]=xx[i]*(-aY*Math.Sin(Ot)+aY*aX*(1-
Math.Cos(Ot)))+yy[i]*(Math.Cos(Ot)+Math.Pow(aY,2)*(1-Math.Cos(Ot)))
                                        +zz[i]*(aX*Math.Sin(Ot)+aY*aZ*(1-Math.Cos(Ot)));
Buf[2]=xx[i]*(aY*Math.Sin(Ot)+aZ*aX*(1-
Math.Cos(Ot)))+yy[i]*(aX*Math.Sin(Ot)+aZ*aY*(1-Math.Cos(Ot)))
                                        +zz[i]*(Math.Cos(Ot)+Math.Pow(aZ,2)*(1-Math.Cos(Ot)));
```

Затем нужно рассчитать интенсивность света в каждой точке и отобразить эти точки. В большинстве случаев объекты задаются набором плоских выпуклых граней, как мы делали при закраске параллелепипеда в пункте 1.8.