

Лабораторная работа №4

Аффинные преобразования в пространстве

Цель работы: получение навыков использования аффинных преобразований в пространстве и создание графического приложения с использованием GDI в среде Visual Studio для визуализации простейших трёхмерных объектов.

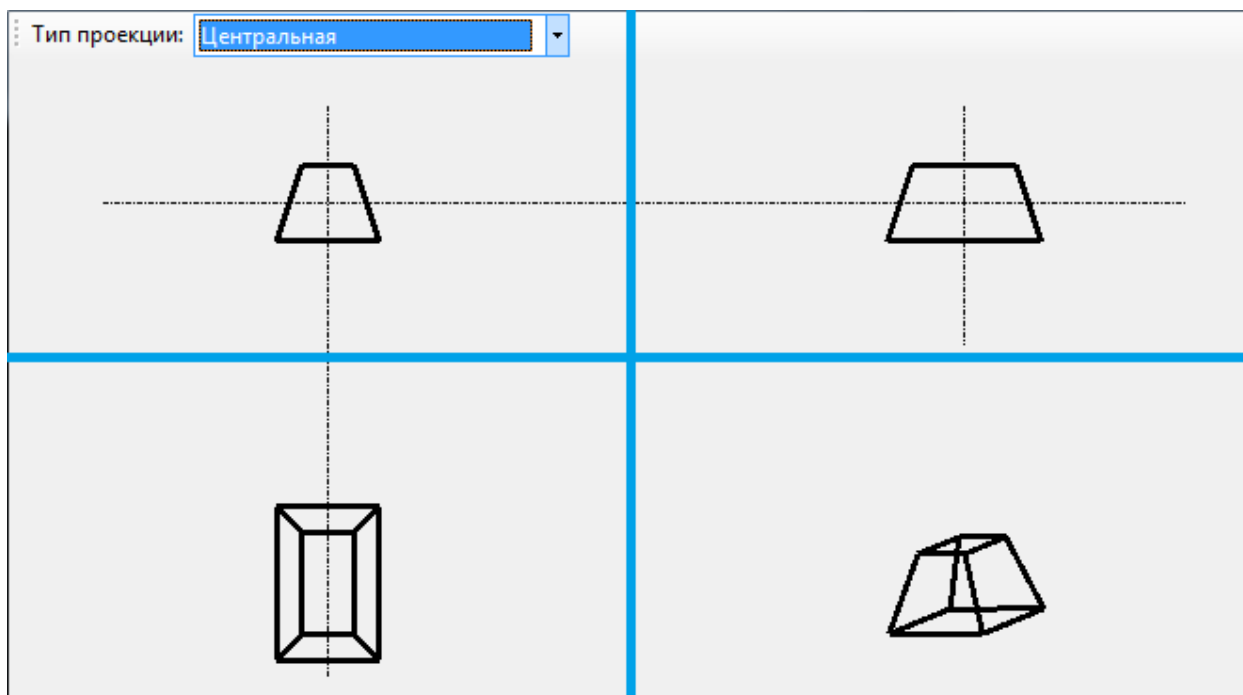
Порядок выполнения работы

1. Разработать алгоритм и составить программу для построения на экране трёхмерных изображений в соответствии с номером варианта. В качестве исходных данных взять указанные в таблице №1.

Требования к программе

1. Программа должна быть написана на языке C++.
2. Окно поделить на 4 части одинаковые части:
 - 2.1. В верхней левой части должна отображаться фронтальная проекция (вид спереди);
 - 2.2. Правая верхняя часть – профильная проекция (вид сбоку);
 - 2.3. Левая нижняя часть должна отображать вид сверху (горизонтальную проекцию);
 - 2.4. На правой нижней части должна отображаться проекция, вид которой выбирает пользователь: центральная, ортографическая, аксонометрическая: изометрическая, диметрическая, триметрическая.

Пример внешнего вида программы:



3. Предусмотреть возможность выбора вида проекции пользователем, например, с помощью выпадающего списка или клавишами.
4. Первые три проекции отобразить без перспективных искажений. Для четвёртой (справа внизу) предусмотреть возможность отдаления/приближения и поворота фигуры клавишами или с помощью мыши.
5. Предусмотреть три режима отображения фигур: только каркас (только рёбра объектов), только грани (заполненные цветом), и каркас и грани. Каркас фигуры нарисовать отрезками с использованием отсечения по методу Сазерленда-Кохена [1, стр. 89].
6. Трёхмерные объекты хранить в памяти как массив многоугольников (не массив отрезков). Все грани фигуры в программе должны быть сгенерированы процедурно.

Содержание отчёта

1. Название темы.
2. Цель работы.
3. Постановка задачи.
4. Вывод необходимых формул для построения граней фигуры.
5. Вывод необходимых формул для построения всех проекций. Указать какие матрицы используются для построения всех четырёх проекций изображений и в какой последовательности они умножаются.
6. Текст программы.
7. Результат работы программы (снимки экрана).
8. Вывод о проделанной работе

Теоретические сведения

Аффинные преобразования в пространстве

По аналогии с двумерным случаем, введём в пространстве однородные координаты. *Однородными координатами* точки (x, y, z) в трёхмерном пространстве называется четвёрка одновременно не равных нулю чисел $x_0:y_0:z_0:w$, связанных следующим соотношениями:

$$x = \frac{x_0}{w}, \quad y = \frac{y_0}{w}, \quad z = \frac{z_0}{w}, \quad w \neq 0.$$

Так же как и в двумерном случае, запишем матрицы аффинных преобразований в пространстве.

1. Матрица вращения на угол φ вокруг оси абсцисс:

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) & 0 \\ 0 & \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

2. Матрица вращения на угол ψ вокруг оси ординат:

$$R_y = \begin{pmatrix} \cos(\psi) & 0 & -\sin(\psi) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\psi) & 0 & \cos(\psi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

3. Матрица вращения на угол χ вокруг оси аппликат:

$$R_z = \begin{pmatrix} \cos(\chi) & \sin(\chi) & 0 & 0 \\ -\sin(\chi) & \cos(\chi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Обратим внимание, что матрицы вращения превращаются в единичную матрицу, если угол поворота равен нулю.

4. Матрица масштабирования (растяжения/сжатия):

$$D = \begin{pmatrix} k_x & 0 & 0 & 0 \\ 0 & k_y & 0 & 0 \\ 0 & 0 & k_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

где

$k_x > 0$ – коэффициент растяжения (сжатия) вдоль оси абсцисс;
 $k_y > 0$ – коэффициент растяжения (сжатия) вдоль оси ординат;
 $k_z > 0$ – коэффициент растяжения (сжатия) вдоль оси аппликат.

5. Матрица отражения относительно плоскости Oxy :

$$M_z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

6. Матрица отражения относительно плоскости Oxz :

$$M_y = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

7. Матрица отражения относительно плоскости Oyz :

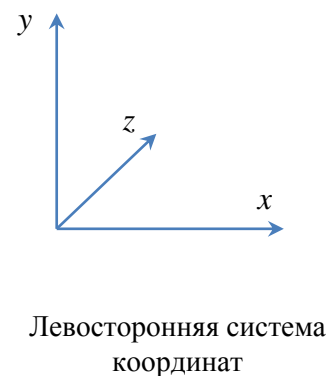
$$M_x = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

8. Матрица переноса вдоль вектора $(\Delta x, \Delta y, \Delta z)$:

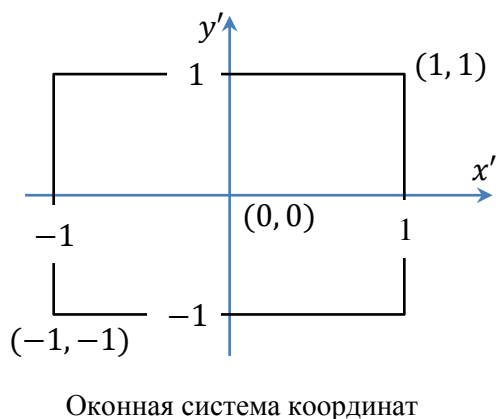
$$T = \begin{pmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Системы координат

В компьютерной графике используются различные системы координат.



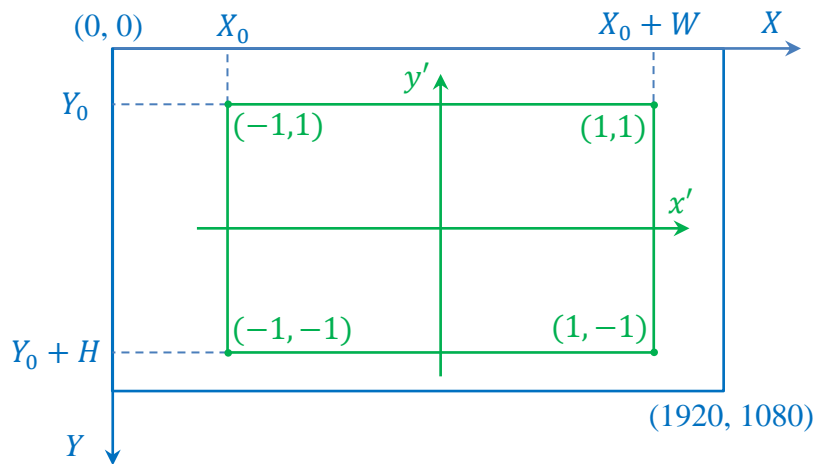
В правосторонней системе координат ось z направлена на наблюдателя, в левосторонней – от наблюдателя. Эти системы будем считать мировыми (математическими) и обозначать соответствующие координаты строчными символами x, y, z, w . Единицей измерения в мировой системе координат может быть метр, километр и др. Далее будем использовать правостороннюю систему координат.



Оконная система координат ограничена прямоугольником $-1 < x', y' < 1$. Центр экрана соответствует началу координат. Такие координаты ещё называют *нормализованными координатами*.

Окна операционной системы (Windows, Linux), а также сам экран, представляют собой матрицу пикселей. Левый верхний угол экрана (или окна) приурочен к началу координат. Ось X направлена вправо, Y – вниз. Единицей измерения в экранной системе координат является целочисленный номер пикселя (по горизонтали или вертикали). Эта система координат является двумерной и предназначена для позиционирования пикселя на экранной плоскости.

Запишем выражения для преобразования нормализованных координат в экранные. В общем случае, начало оконной системы координат может быть сдвинуто от угла экрана, как показано на рисунке.



Пусть окно имеет ширину W и высоту H пикселей. На графиках видно, что координаты соответствуют друг другу так:

$$\begin{aligned} X = X_0 &\rightarrow x' = -1, \\ X = X_0 + W &\rightarrow x' = 1, \\ Y = Y_0 &\rightarrow y' = 1, \\ Y = Y_0 + H &\rightarrow y' = -1. \end{aligned}$$

Поскольку данное преобразование является линейным, оно рассчитывается исходя из простых пропорций:

$$\frac{X - X_0}{W} = \frac{x' + 1}{2}, \quad \frac{Y - Y_0}{H} = \frac{1 - y'}{2}.$$

Хоть это и не имеет чаще всего смысла, но для координаты Z тоже вводят некоторые границы Z_0, Z_1 и соответствующую пропорцию:

$$\frac{Z - Z_0}{Z_1 - Z_0} = \frac{z' + 1}{2}.$$

Из представленных пропорций выразим экранные координаты (X, Y, Z) через нормализованные координаты (x', y', z') :

$$\begin{aligned} X &= x' \frac{W}{2} + \frac{W}{2} + X_0, \\ Y &= -y' \frac{H}{2} + \frac{H}{2} + Y_0, \\ Z &= z' \frac{Z_1 - Z_0}{2} + \frac{Z_1 + Z_0}{2}. \end{aligned}$$

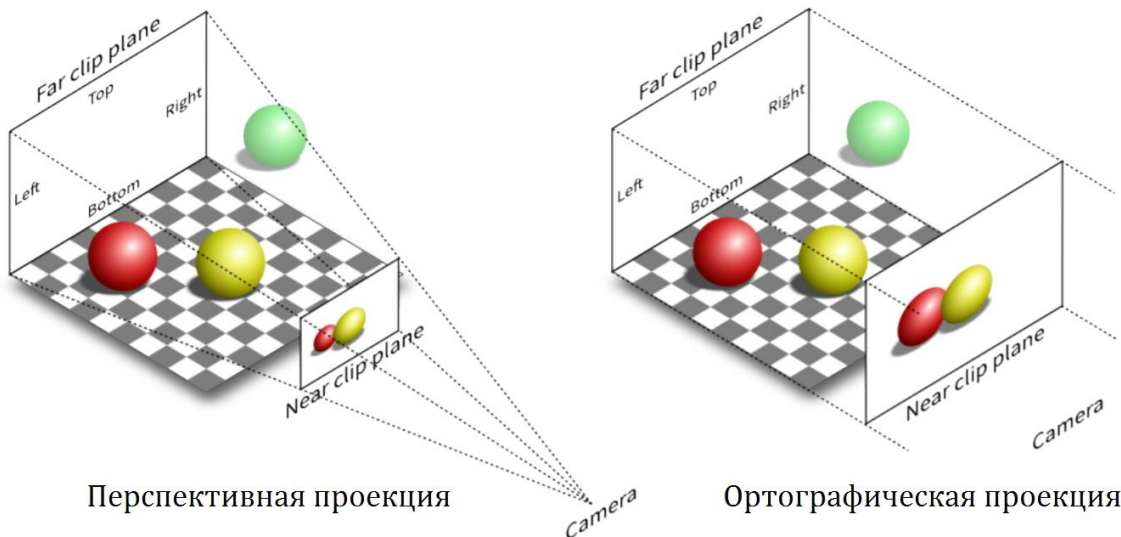
Полученные выражения можно переписать в матричном виде:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{W}{2} & 0 & 0 & X_0 + \frac{W}{2} \\ 0 & -\frac{H}{2} & 0 & Y_0 + \frac{H}{2} \\ 0 & 0 & \frac{Z_1 - Z_0}{2} & \frac{Z_1 + Z_0}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}.$$

Построенную матрицу будем называть *матрицей перехода к экранным координатам* и обозначать как $P_{viewport}$.

Проектирование

В трёхмерной графике рассматривают ортографические и перспективные проекции.



Перспективная проекция

Ортографическая проекция

Для получения двумерного изображения трёхмерного объекта на экране необходимо после аффинных преобразований выполнить ещё одну операцию, называемую *проектированием* (или *проецированием*). В геометрическом смысле, проекция – изображение трёхмерной фигуры на картинной (экранной) плоскости. Операция проектирования изображена на рис. 1. Результатом проектирования является набор точек, лежащих на экранной плоскости, которая обычно совпадает с плоскостью $z = 0$.

Для получения проекции необходимо сопоставить каждой вершине трёхмерной фигуры точку на экранной плоскости. Существует несколько видов проекций. Самым простым видом проектирования является *ортографическое* проектирование. Подобные проекции используются в черчении и инженерной графике. Для получения ортографической проекции трёхмерной фигуры необходимо провести из каждой её вершины перпендикуляр к картинной плоскости. В результате этого координата z обратится в ноль, а координаты x, y останутся неизменными. В самом простом случае это равносильно умножению однородных координат точки на следующую матрицу:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix}.$$

В общем случае ортографическое проектирование выполняют с целью нормализации координат точки, т.е. приведения их к диапазону $[-1, 1]$. Построим в трёхмерном

пространстве параллелепипед (рис. 1) с гранями, расположенными перпендикулярно координатным осям.

Грани параллелепипеда лежат на 6 плоскостях отсечения:

- $x = x_1$ – левая (left) плоскость отсечения;
- $x = x_2$ – правая (right) плоскость отсечения;
- $y = y_1$ – нижняя (bottom) плоскость отсечения;
- $y = y_2$ – верхняя (top) плоскость отсечения;
- $z = -z_1$ – ближняя (near) плоскость отсечения;
- $z = -z_2$ – дальняя (far) плоскость отсечения.

Можно также сказать, что

z_1 – расстояние от наблюдателя до ближней плоскости отсечения,

z_2 – расстояние от наблюдателя до дальней плоскости отсечения.

Экранная плоскость будет совпадать с плоскостью $z = 0$.

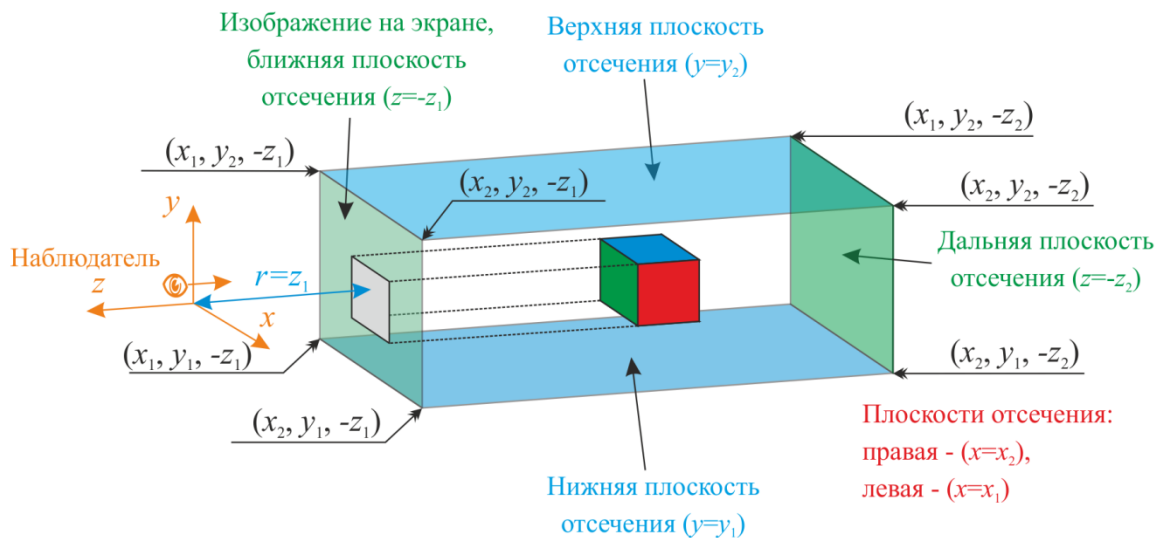


Рис. 1. Ортогографическая проекция кубика

Пусть точка (x, y, z) находится внутри параллелепипеда и движется от левой плоскости отсечения до правой. Необходимо нормализовать координату x таким образом, чтобы она при движении линейно увеличивалась от -1 до 1. Т.е., если точка (x, y, z) находится на левой плоскости, то её нормализованная координата $x' = -1$, а если на правой, то $x' = 1$. Это достигается простым преобразованием:

$$x' = -1 + 2 \frac{x - x_1}{x_2 - x_1}.$$

Для оси ординат данное преобразование записывается аналогично, достаточно лишь заменить все координаты x на координаты y :

$$y' = -1 + 2 \frac{y - y_1}{y_2 - y_1}.$$

Для оси аппликат это выражение отличается знаком координаты z . Требуется, чтобы при удалении точки внутрь экрана её нормализованная координата z' увеличивалась, поэтому

$$z' = -1 + 2 \frac{-z - z_1}{z_2 - z_1}.$$

Тогда на ближней плоскости отсечения $z = -z_1$ координата z нормализуется к значению -1, на дальней – к 1. Значение z' также называют *глубиной* точки.

Далее приведём значение x' к общему знаменателю $(x_2 - x_1)$:

$$x' = -1 + 2 \frac{x - x_1}{x_2 - x_1} = \frac{x_1 - x_2 + 2(x - x_1)}{x_2 - x_1} = \frac{2x}{x_2 - x_1} - \frac{x_2 + x_1}{x_2 - x_1}.$$

Запишем подобные выражения для осей y, z :

$$y' = \frac{2y}{y_2 - y_1} - \frac{y_2 + y_1}{y_2 - y_1},$$

$$z' = \frac{-2z}{z_2 - z_1} - \frac{z_2 + z_1}{z_2 - z_1}.$$

Теперь нетрудно записать эти выражения в матричном виде:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{x_2 - x_1} & 0 & 0 & -\frac{x_2 + x_1}{x_2 - x_1} \\ 0 & \frac{2}{y_2 - y_1} & 0 & -\frac{y_2 + y_1}{y_2 - y_1} \\ 0 & 0 & \frac{-2}{z_2 - z_1} & -\frac{z_2 + z_1}{z_2 - z_1} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}.$$

Построенную матрицу называют *матрицей ортогографического проектирования*. Обозначим её как P_{ortho} .

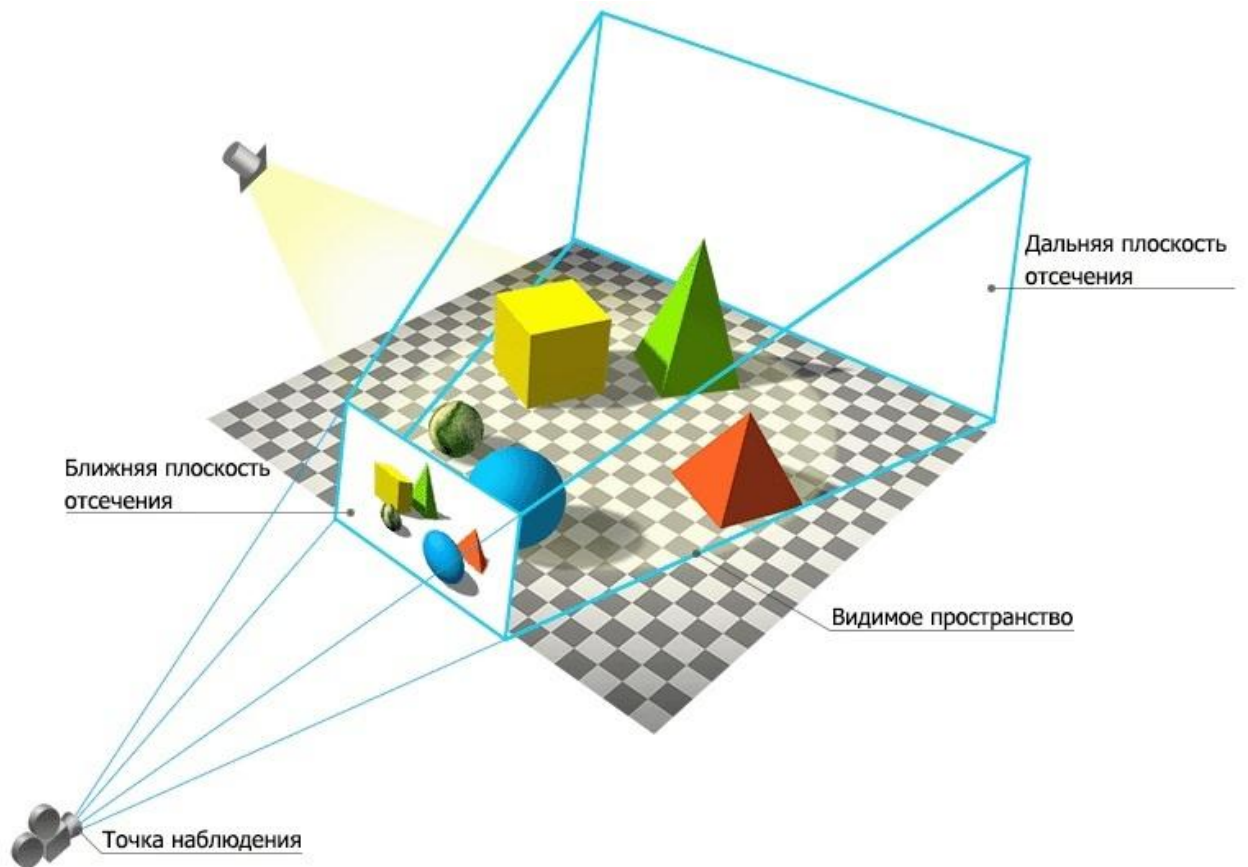
Таким образом, матрица P_{ortho} преобразует мировые координаты объектов в оконные координаты и нормализует их к диапазону $[-1, 1]$. После её использования пользователь будет видеть на экране только точки, находящиеся внутри ограничивающего параллелепипеда:

$$x_1 \leq x \leq x_2, \quad y_1 \leq y \leq y_2, \quad z_1 \leq z \leq z_2.$$

Естественным образом пользователь не увидит те точки, для которых $x \notin [x_1, x_2]$ или $y \notin [y_1, y_2]$, потому что оконная система координат ограничена прямоугольником $-1 \leq x, y \leq 1$ ([рассматривалось ранее](#)). Т.е. они будут за границами экрана.

Координата z здесь не обнуляется, а также преобразуется к диапазону $[-1, 1]$. В буфере глубины точки, не попадающие в этот диапазон, будут отброшены (не будут нарисованы). Ось z направлена на наблюдателя, поэтому элемент в 3 строке и 3 столбце матрицы P_{ortho} имеет отрицательный знак.

Ортогографические проекции обладают одним существенным недостатком – наблюдатель не может оценить расстояние до объектов (глубину). Если перемещать объект вдоль оси z , то его ортогографическая проекция на экранную плоскость не изменится. В связи с этим, в компьютерной графике (игры, САД-системы и др.) чаще применяется *центральное (перспективное)* проектирование. При таком проектировании при удалении от наблюдателя проекции объектов на экранную плоскость будут уменьшаться и появится глубина.



Рассмотрим перспективное проектирование. Камера (точка схода лучей) находится в начале координат и направлена против оси z . Пусть (x, y, z) – точка, которая находится внутри усечённой пирамиды (рисунок), образованной шестью плоскостями:

1. Левая (left) плоскость отсечения,
2. Правая (right) плоскость отсечения,
3. Нижняя (bottom) плоскость отсечения,
4. Верхняя (top) плоскость отсечения,
5. Ближняя (near) плоскость отсечения,
6. Дальняя (far) плоскость отсечения.

Исходными данными для перспективного проектирования являются значения:

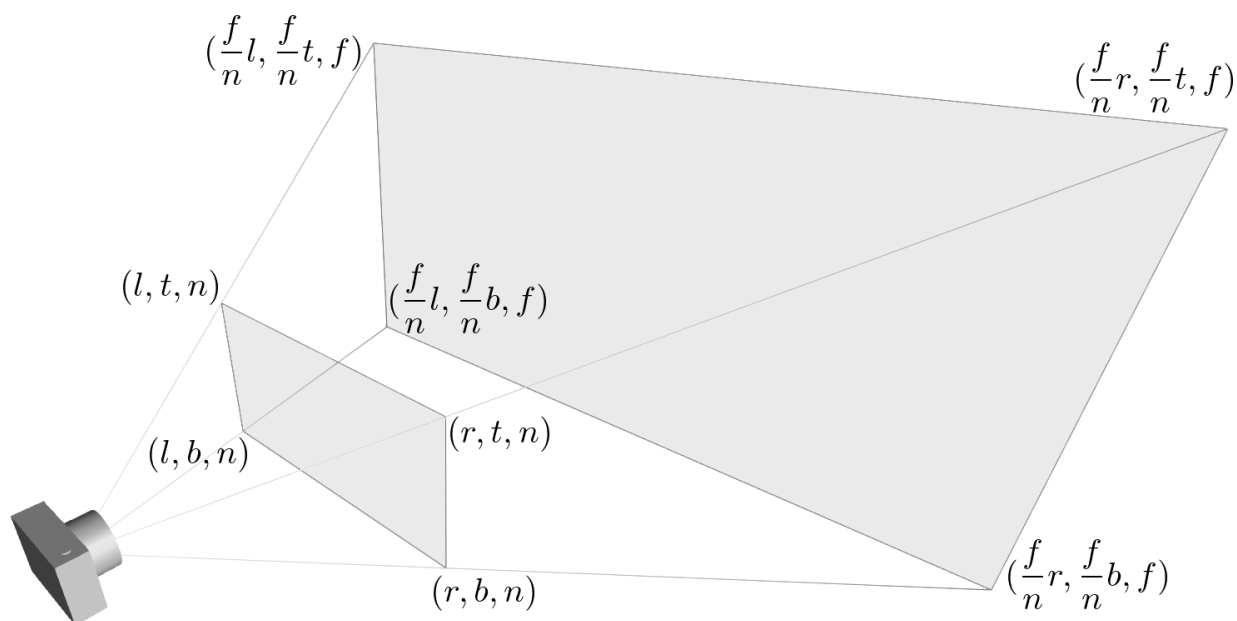
уравнение ближней плоскости отсечения: $z = -n$,

уравнение дальней плоскости отсечения: $z = -f$,

границы прямоугольника $x = l$, $x = r$, $y = b$, $y = t$ на ближней плоскости отсечения $z = n$, которые определяют видимую область. Чем больше будут стороны данного прямоугольника, тем шире будет видимая область.

Все лучи сходятся в начало координат, значит, чтобы получить координаты противоположных точек на дальней плоскости отсечения, нужно умножить координаты вершин ближнего прямоугольника на f/n .

(На картинке третья координата z должна быть со знаком минус).



Перспективное проектирование заключается в том, чтобы нормализовать все три координаты точки (x, y, z) к диапазону $[-1, 1]$. К примеру, если точка находится на ближней плоскости отсечения, то её координата z должна быть нормализована к значению -1 ; если на дальней – то к значению 1 . Аналогично должны быть нормализованы координаты x, y . Если $x = l$, то координате x должна быть сопоставлена -1 ; если $x = r$, то 1 . Запишем для наглядности ещё некоторые соответствия:

$$(l, b, -n) \rightarrow (-1, -1, -1),$$

$$(r, t, -n) \rightarrow (1, 1, -1),$$

$$\left(\frac{f}{n}l, \frac{f}{n}b, -f\right) \rightarrow (-1, 1, 1),$$

$$\left(\frac{f}{n}r, \frac{f}{n}t, -f\right) \rightarrow (1, 1, 1).$$

Запишем уравнение левой плоскости отсечения. На ней находятся три точки – $(l, b, -n)$, $(l, t, -n)$, $(0, 0, 0)$. Уравнение плоскости, проходящей через эти три точки имеет вид:

$$x + \frac{l}{n}z = 0.$$

Аналогичным образом можно сразу записать уравнения всех остальных плоскостей отсечения:

$$\text{правая} - x + \frac{r}{n}z = 0,$$

$$\text{нижняя} - y + \frac{b}{n}z = 0,$$

$$\text{верхняя} - y + \frac{t}{n}z = 0.$$

Проведём через наблюдаемую точку (x, y, z) горизонтальную прямую (параллельную оси абсцисс). Она пересечёт левую плоскость отсечения в точке $\left(-\frac{l}{n}z, y, z\right)$, правую – в точке $\left(-\frac{r}{n}z, y, z\right)$. Значит, координата x нормализуется от нуля до единицы следующим образом:

$$x_l = -\frac{l}{n}z, \quad x_r = -\frac{r}{n}z,$$

$$\frac{x - x_l}{x_r - x_l} = \frac{x - \left(-\frac{l}{n}z\right)}{-\frac{r}{n}z - \left(-\frac{l}{n}z\right)} = \frac{xn + lz}{-rz + lz}.$$

Нормализация координаты x от -1 до 1 запишется так:

$$x' = -1 + 2 \frac{xn + lz}{-rz + lz} = \frac{rz - lz + 2xn + 2lz}{-rz + lz} = \frac{z(r + l) + 2xn}{-z(r - l)}.$$

Теперь проведём через наблюдаемую точку (x, y, z) вертикальную прямую. Нормализованная координата y выводится аналогично, достаточно заменить в последнем выражении x на y , l на b , r на t :

$$y' = \frac{z(t + b) + 2yn}{-z(t - b)}.$$

Координату z нормализуют иначе. На неё накладывается дополнительное требование – значение z' должно быстро стремиться к минус бесконечности при приближении точки (x, y, z) к началу координат, т.е. к наблюдателю. Её принято рассчитывать по формуле:

$$z' = 1 + 2 \frac{n(z + f)}{z(f - n)} = \frac{z(f - n) + 2n(z + f)}{z(f - n)} = \frac{z(f + n) + 2nz}{z(f - n)}.$$

Значение z' называют *псевдоглубиной*, потому что её используют в тестах глубины. Можно проверить, что при $z = -n$ выполняется условие: $z' = -1$. При $z = -f$ псевдоглубина z' равна 1 . Если $z' \notin [-1, 1]$, то точка (x, y, z) будет отсечена и не попадёт в буфер глубины.

Рассмотренные выражения для x', y', z' можно объединить одним матричным преобразованием:

$$\begin{pmatrix} x^* \\ y^* \\ z^* \\ w^* \end{pmatrix} = P_{proj} \times \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix},$$

$$x' = \frac{x^*}{w^*}, \quad y' = \frac{y^*}{w^*}, \quad z' = \frac{z^*}{w^*}.$$

Матрица перспективного проектирования P_{proj} неудобна (неудобна примерно также как колготки через голову надевать) тем, что при её использовании у трёхмерных объектов после проектирования не сохраняются соотношения сторон. К тому же нет гарантии, что при делении компонента w^* не будет равна нулю.

Растреризация треугольника в пространстве

Для отображения трёхмерных объектов на экране в нужном порядке необходимо также уметь определять видимость граней. Нетрудно догадаться, что грани, которые перекрываются другими гранями, должно быть либо отброшены при растреризации, либо нарисованы раньше. Существуют несколько способов отсекаания невидимых граней. Рассмотрим самый простой из них, который в графических процессорах реализуется на аппаратном уровне и называется *метод z-буфера*. Идея использования z-буфера проста:

1. Каждому пикселю на экране сопоставляется число z , которое называется глубиной пикселя. Данное значение инициализируется значением $+\infty$. Все значения z для каждого пикселя образуют матрицу или буфер глубины размером, равным размеру экрана (например 1920x1080 для разрешения Full HD).
2. При растреризации очередного треугольника для каждого его пикселя определяется расстояние z' от точки на треугольнике до экранной плоскости. Если $z' < z$, то в z-буфере глубина z текущего пикселя перезаписывается значением z' , а в буфере кадра обновляется цвет соответствующего пикселя на цвет треугольника. После растреризации всех треугольников в буфер глубины будут записаны минимальные расстояния от пикселей до треугольников.

Ввиду своей простоты метод z-буфера реализуется аппаратно и позволяет очень быстро отсекаать невидимые грани. Причём даже не имеет значения, в каком порядке растреризуются эти грани. Но у него есть существенный недостаток – он не годится для растреризации полупрозрачных объектов.

Нам не обязательно объявлять отдельную матрицу для хранения z-буфера. Добавим в структуру ячейки буфера кадра, которая хранит цвет, ещё одно вещественное поле Z и инициализируем его значением INFINITY (бесконечность).

```
// Пиксель в буфере кадра
typedef struct tagPIXEL
{
    unsigned char RED;           // Компонента красного цвета
    unsigned char GREEN;        // Компонента зелёного цвета
    unsigned char BLUE;         // Компонента синего цвета
    float Z;                    // Глубина пикселя
    tagPIXEL() : RED(0), GREEN(0), BLUE(0), Z(INFINITY) { }
} PIXEL;
```

Добавим в функцию растреризации во внутренний цикл вычисление барицентрических координат h_0 , h_1 , h_2 текущего пикселя (x, y) :

```
float S = (y1 - y2) * (x0 - x2) + (x2 - x1) * (y0 - y2); // Площадь треугольника
float h0 = ((y1 - y2) * (x - x2) + (x2 - x1) * (y - y2)) / S;
float h1 = ((y2 - y0) * (x - x2) + (x0 - x2) * (y - y2)) / S;
float h2 = ((y0 - y1) * (x - x1) + (x1 - x0) * (y - y1)) / S;
```

Тогда координата z точки (текущего пикселя) на треугольнике вычисляется в результате интерполяции координат z трёх вершин треугольника:

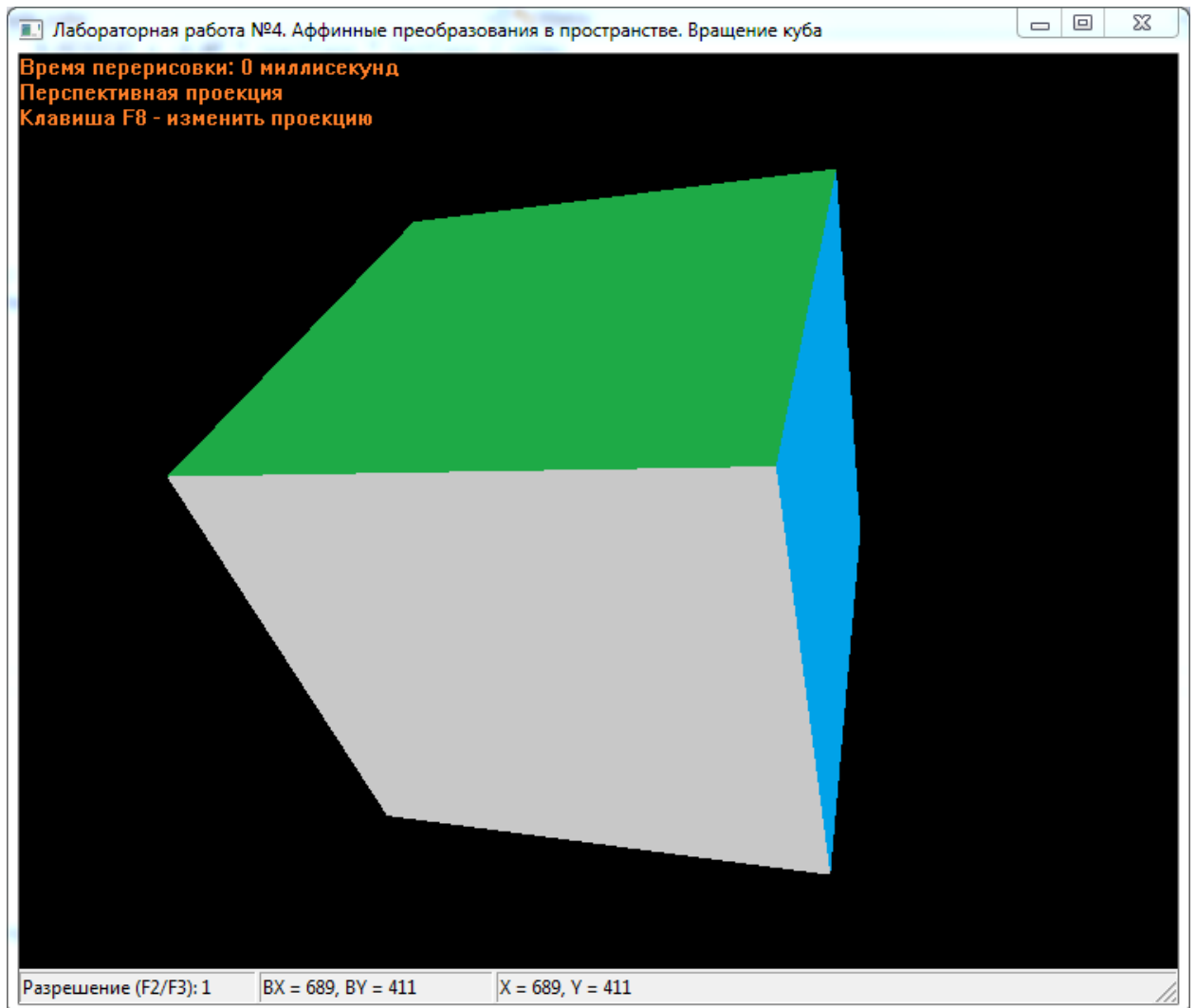
```
float Z = h0 * z0 + h1 * z1 + h2 * z2; // Глубина пикселя
```

Если $Z < -1$ или $Z > 1$, то пиксель не записывается в буфер глубины в соответствии с требованиями перспективного проектирования. Если точка (x, y, z) не треугольнике ближе уже записанной в буфере кадра, то перезаписываем цвет и глубину:

```
// Определение глубины точки в экранной системе координат
PIXEL* pixel = pixels + (size_t)Y * width + X; // Вычислим адрес пикселя (Y, X) в
матрице пикселей pixels
// pixel->Z - глубина пикселя, которая уже записана в буфер кадра
// Если текущий пиксель находится ближе того пикселя, который уже записан в буфере
кадра
if (Z > -1 && Z < 1 && Z < pixel->Z)
{ // то обновляем пиксель в буфере кадра
    pixel->RED = color.RED;
```

```
pixel->GREEN = color.GREEN;  
pixel->BLUE = color.BLUE;  
pixel->Z = Z;  
}
```

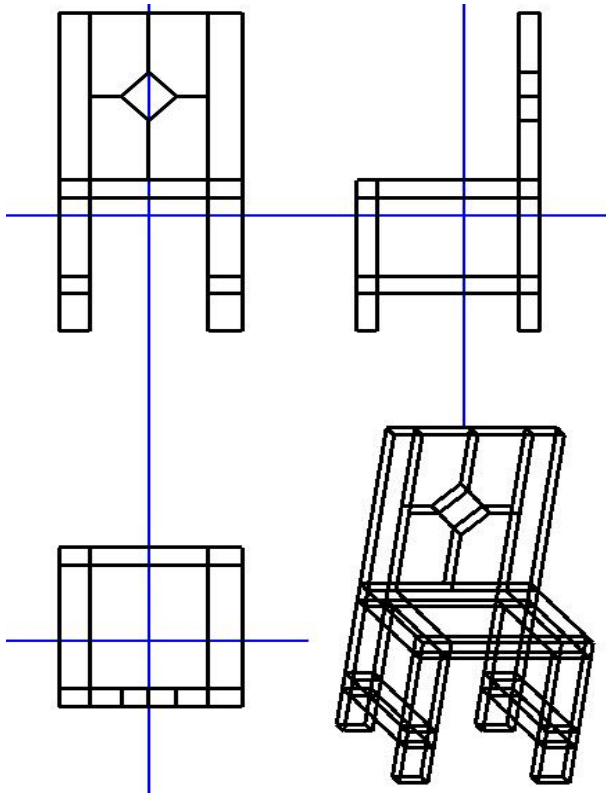
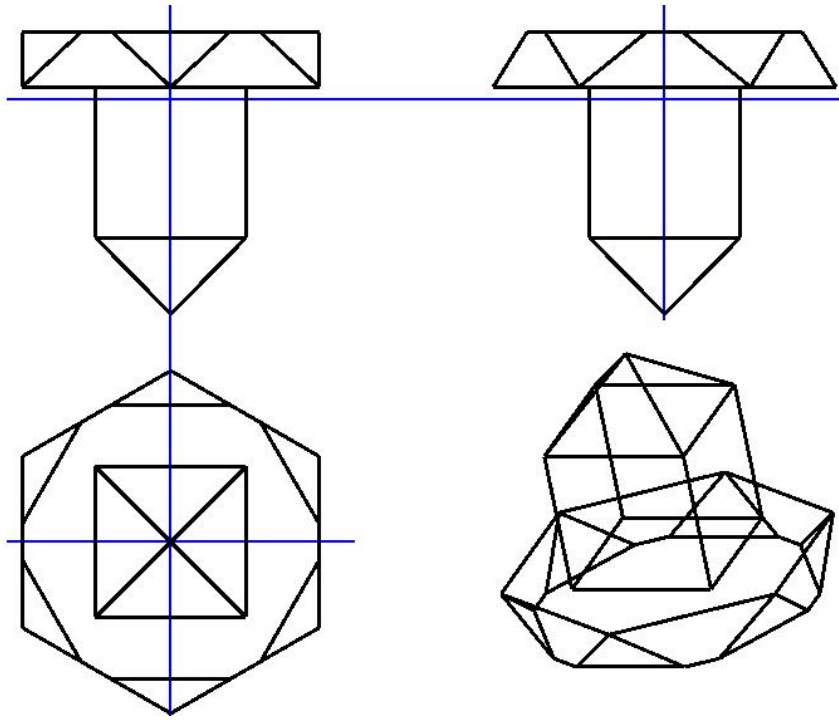
Пример рисования куба смотреть в приложенном проекте.



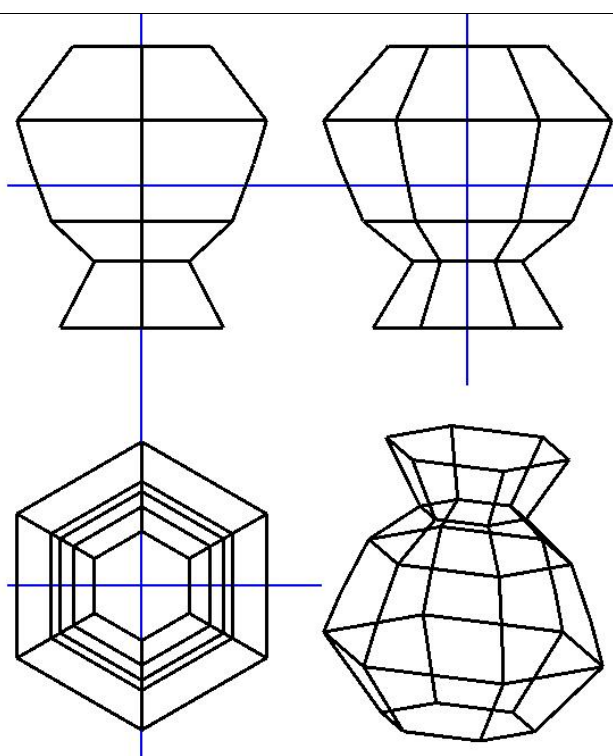
Список литературы

1. Шикин Е.В., Боресков А.В. Компьютерная графика. Динамика, реалистические изображения. М.: ДИАЛОГ МИФИ, 1996. 288 с.

Таблица 1

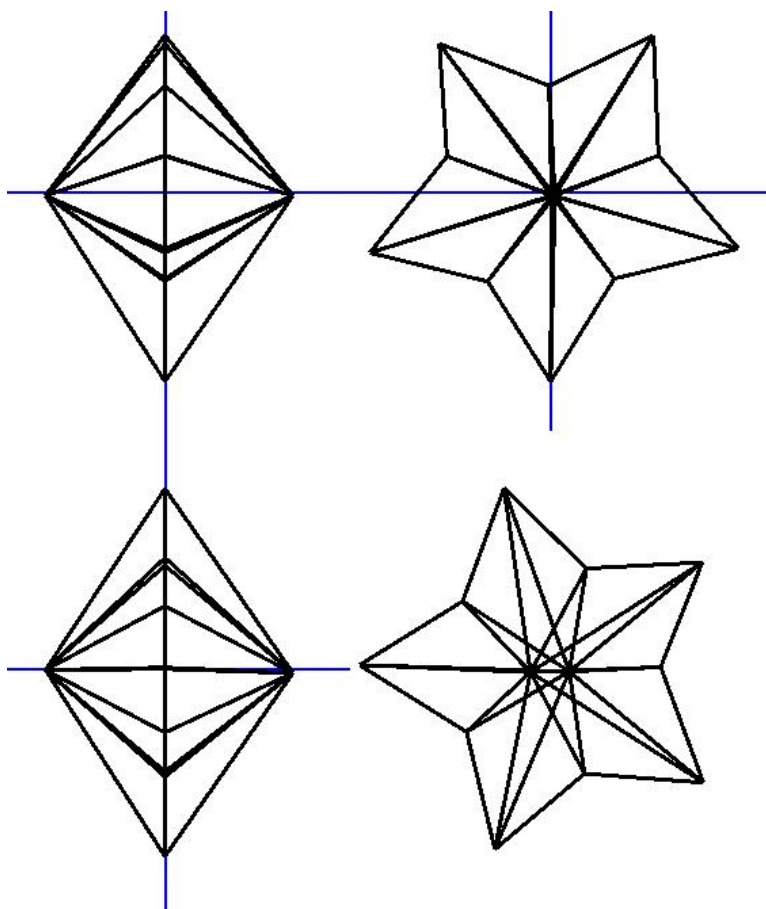
№	Рисунок	Входные данные
1		Количество перегородок между ножками стула.
2		n-угольник стоящий на шестиугольнике.

3

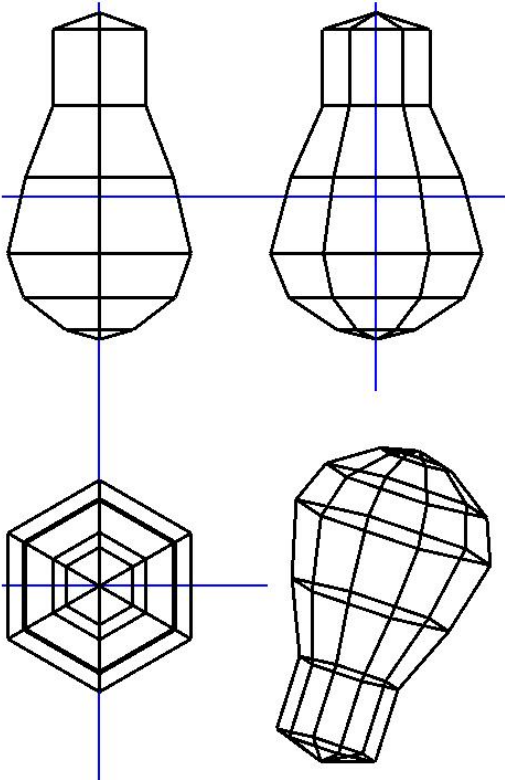
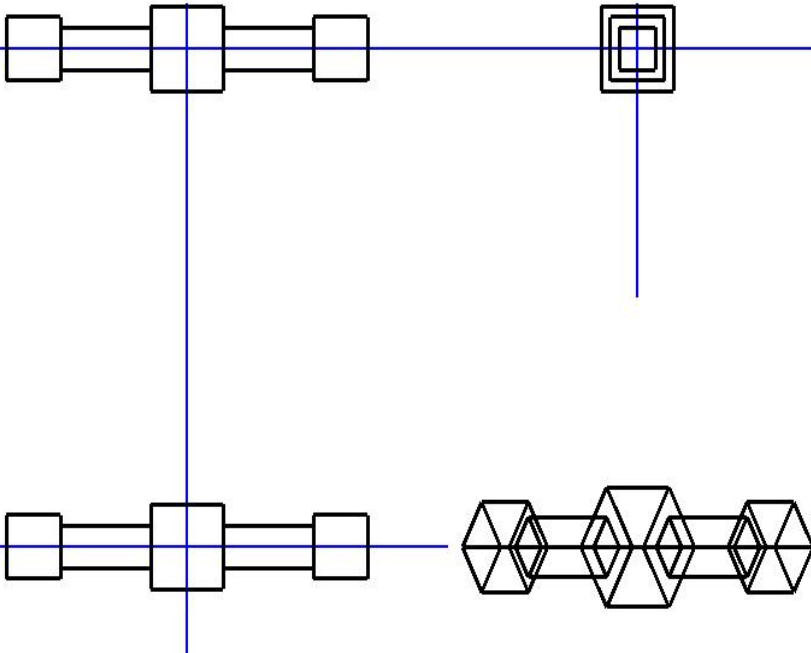


Количество n -
угольников
между
основанием и
горлышком и
количество
углов (фигура
получится
более гладкая).

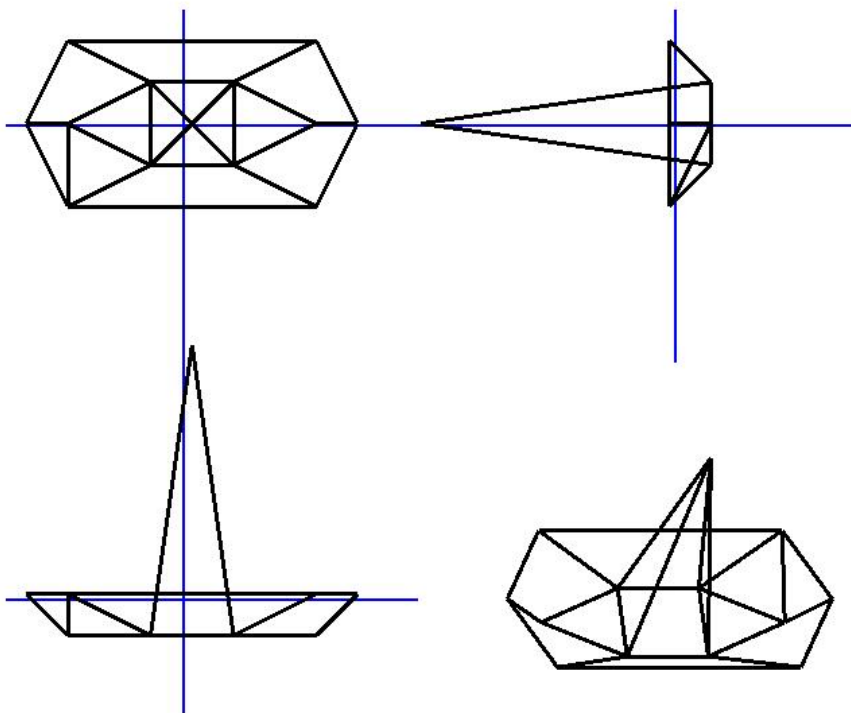
4



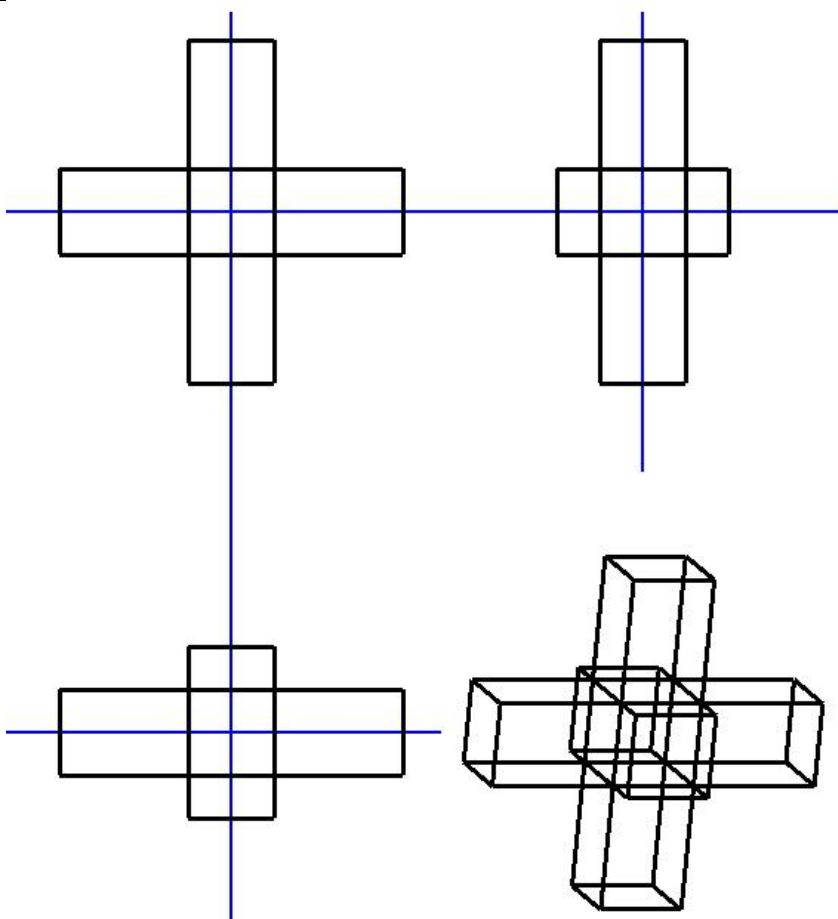
Ширина звезды
и радиус
описанной
окружности.

5		<p>Количество п-угольников между цоколем и верхушкой лампочки и количество углов (фигура получится более гладкая).</p>
6		<p>Изменять расстояние между кубиками при движении колесика мыши</p>

7

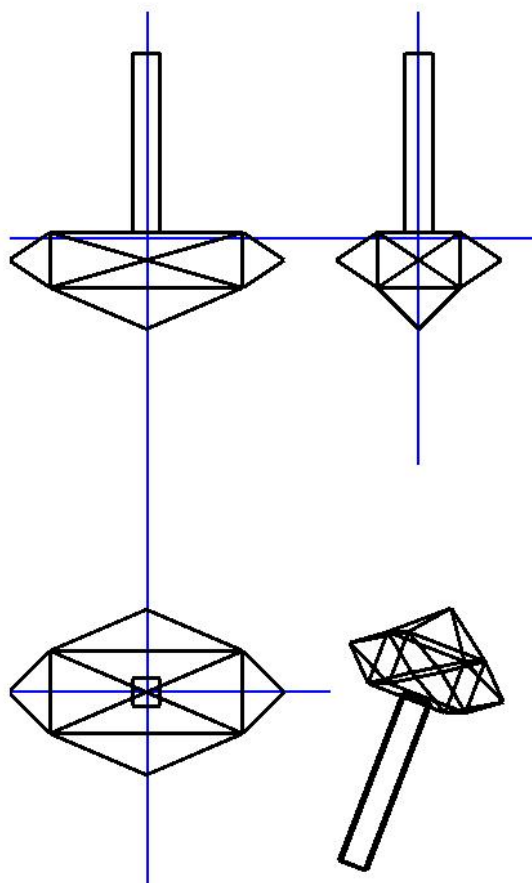


Количество
углов (в идеале
получится
круглая)



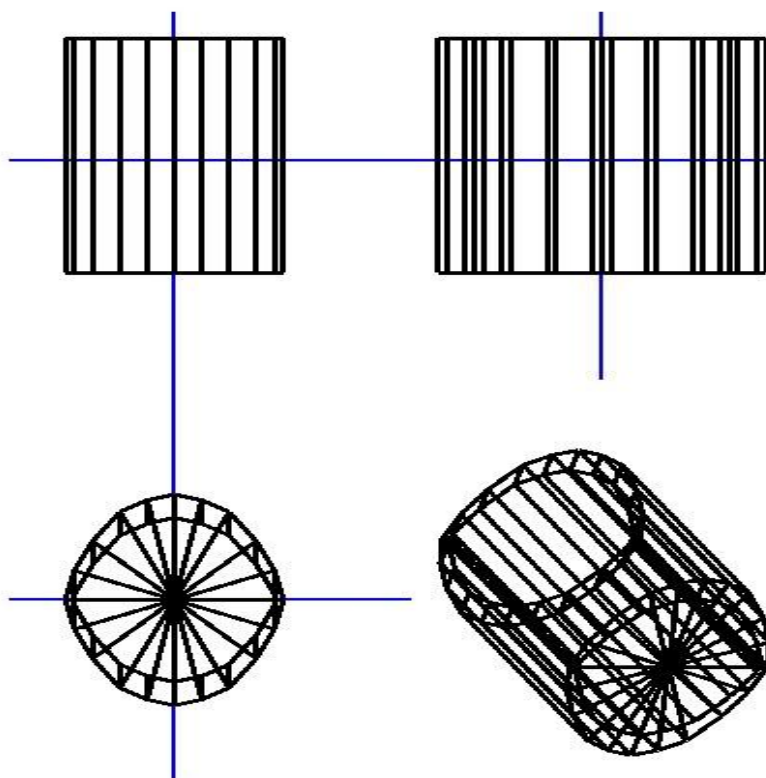
Изменять
количество
углов
выступающей
фигуры

8

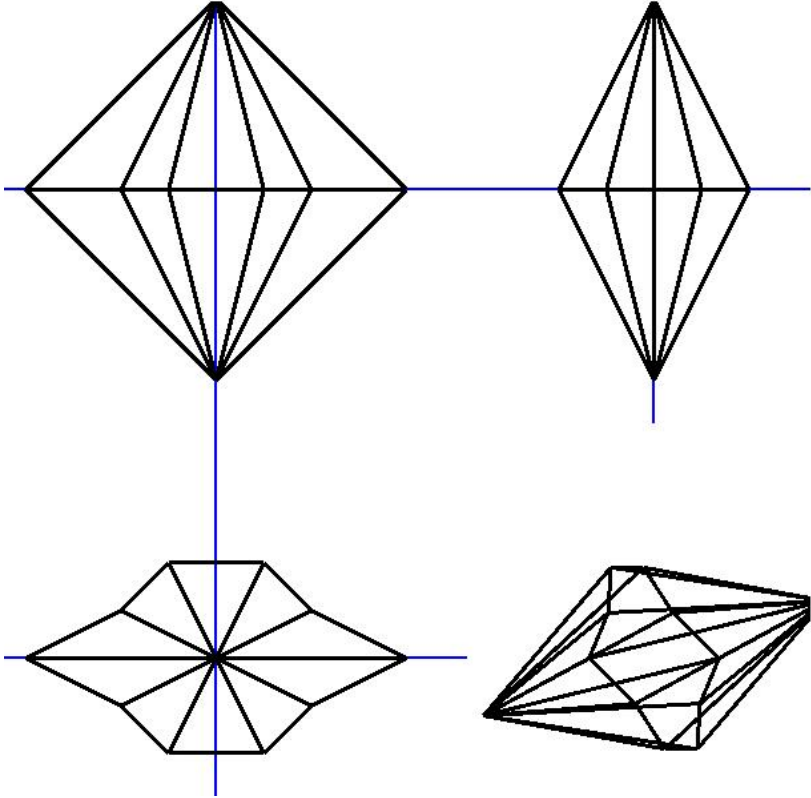
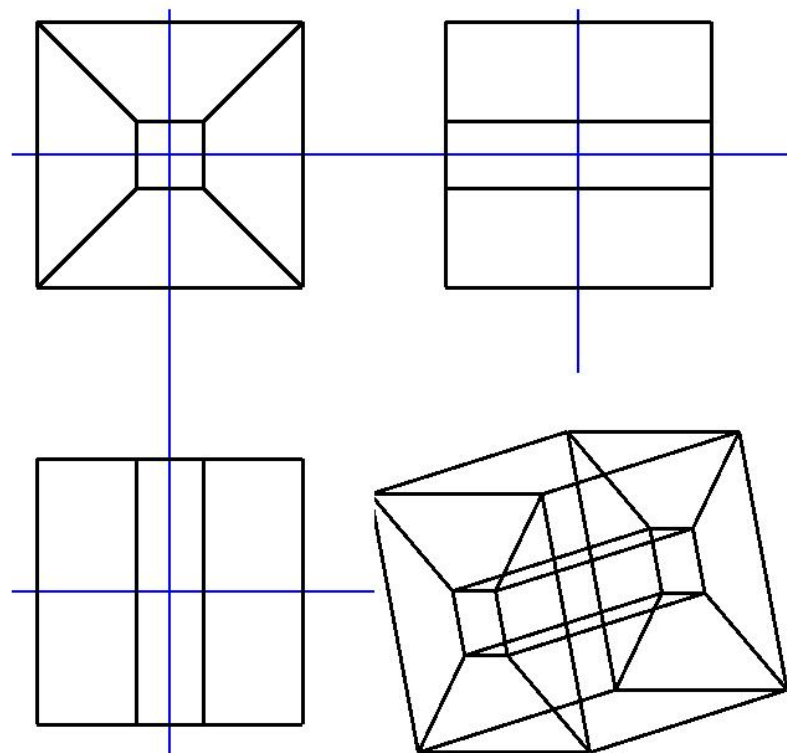


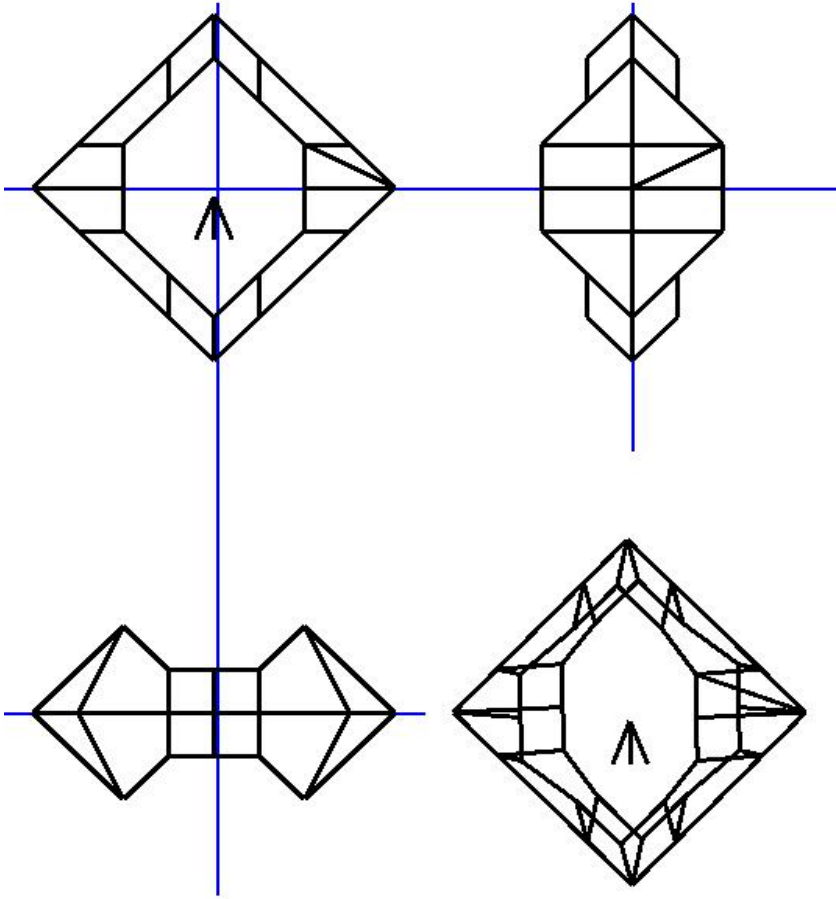
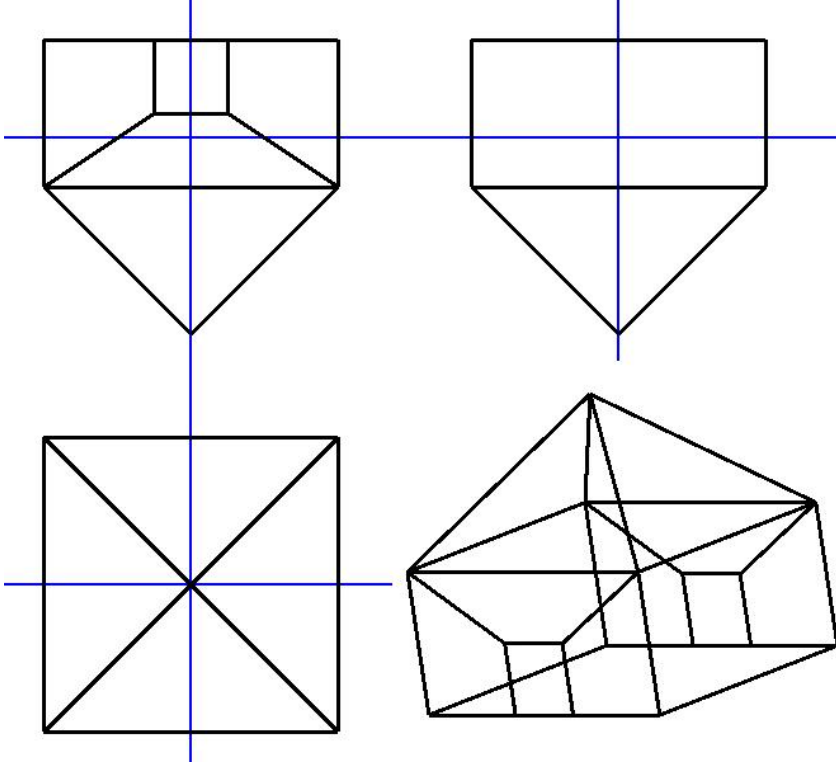
Изменять угол
пирамид при
движении
колесика мыши

9

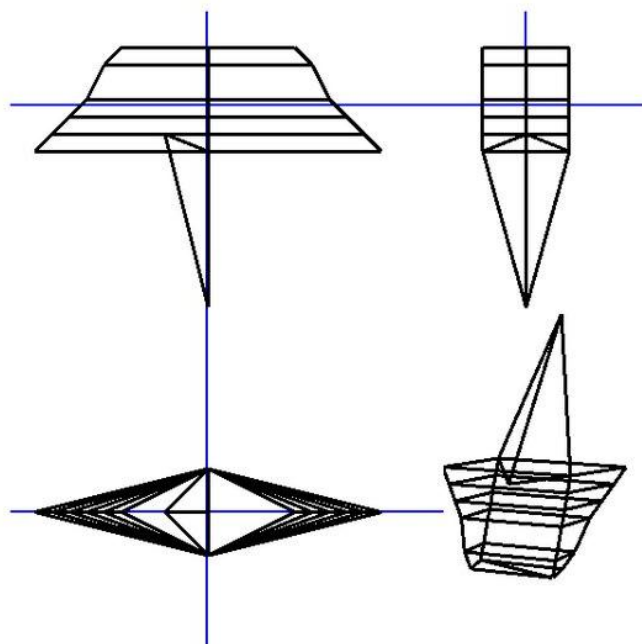


Изменять
количество
граней при
движении
колесика мыши
(делать стакан
более круглым)

10		<p>Изменять количество граней</p>
11		<p>Изменять количество углов во внутренней фигуре</p>
12		<p>Изменять направление стрелочки при движении</p>

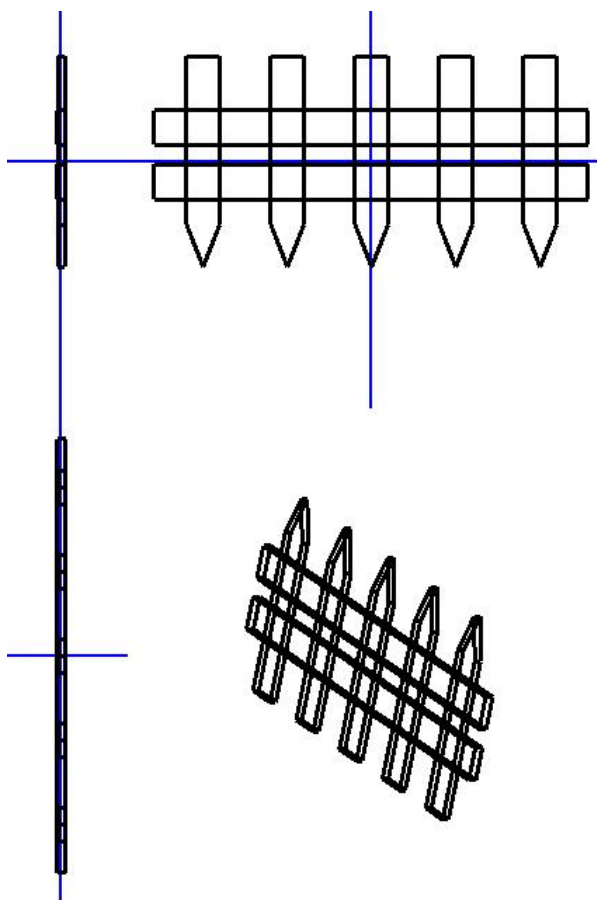
		<p>КОЛЕСИКОМ МЫШИ</p>
13		<p>Изменять количество углов во внутренней фигуре (сейчас – это дверь)</p>

14



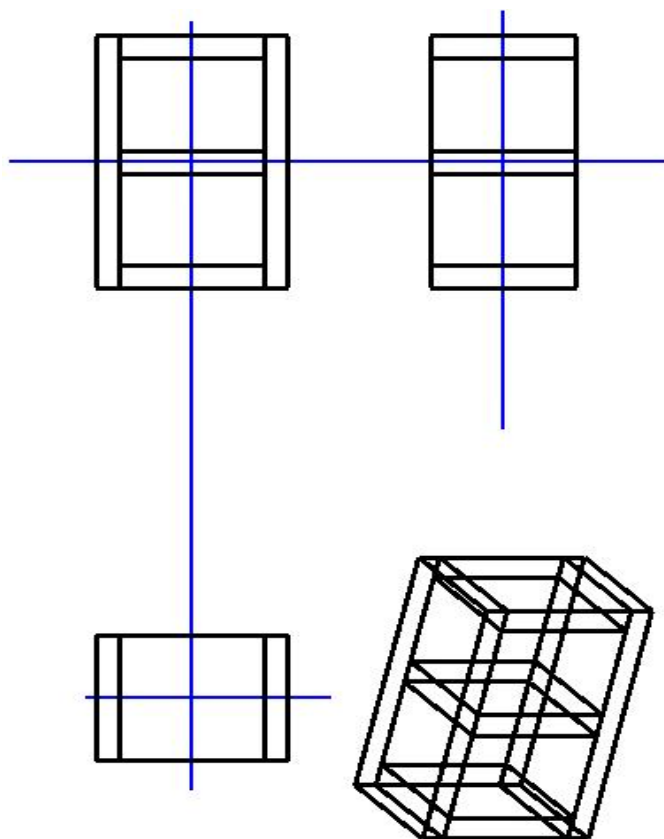
Изменять количество граней в парусе при движении колесика мыши (делать парус более овальным)

15



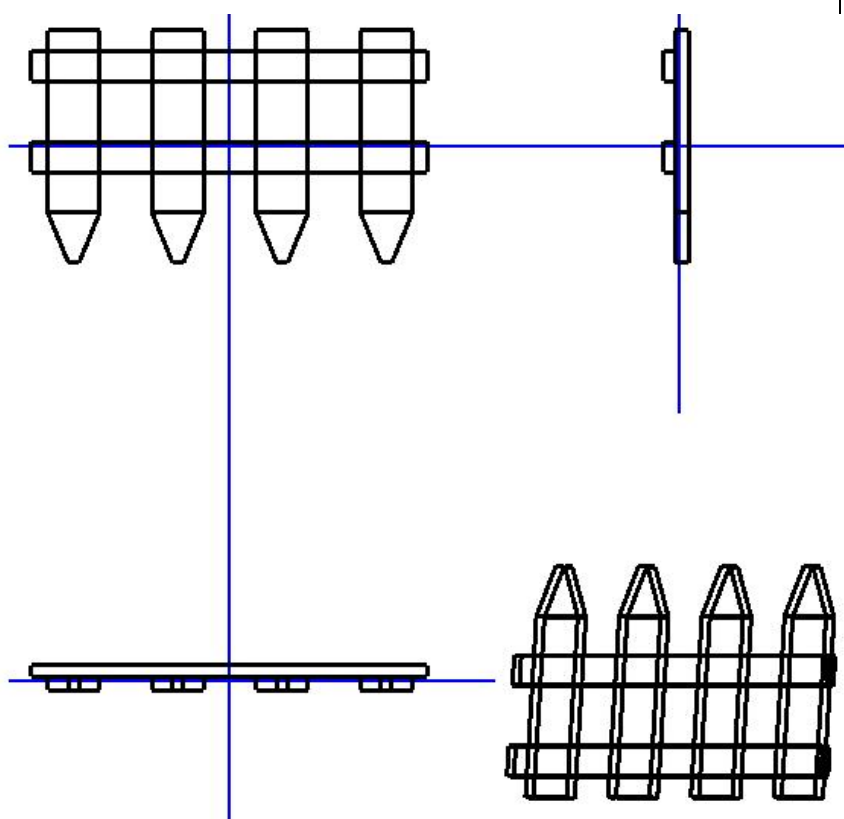
Количество шпал между рельсами

16



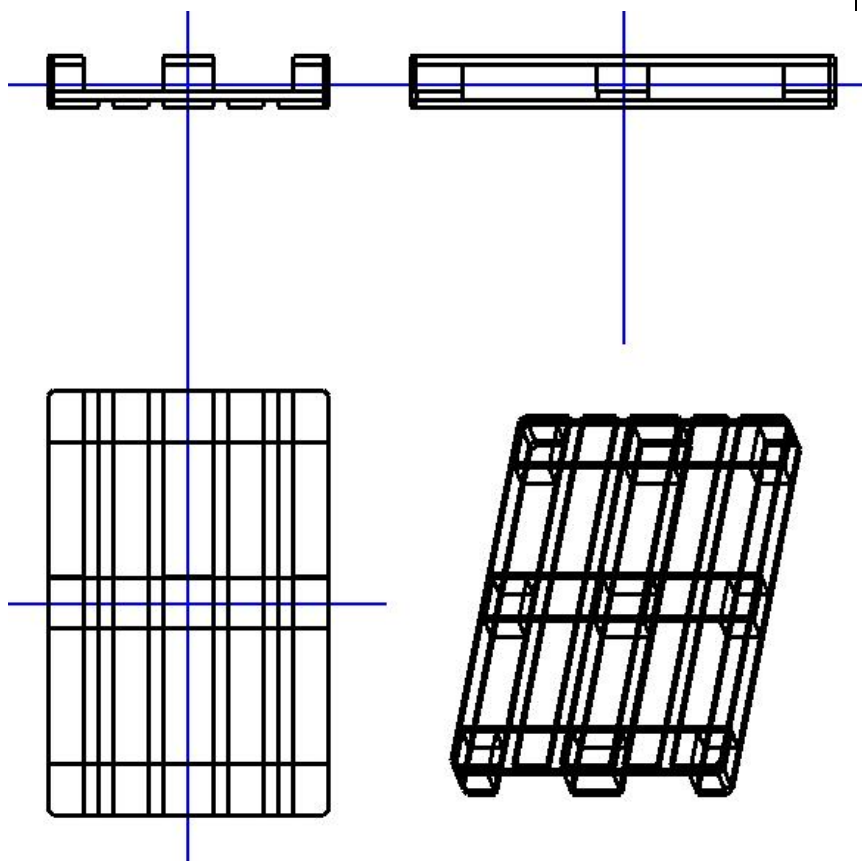
Ширина
средней
перегородки
фигуры

17



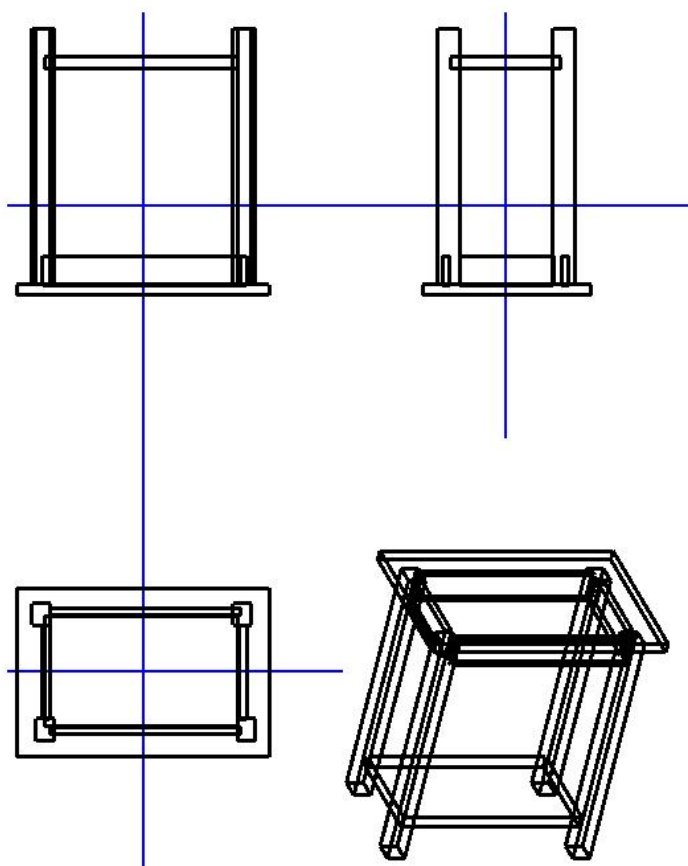
Количество
вертикальных
досок в заборе

18



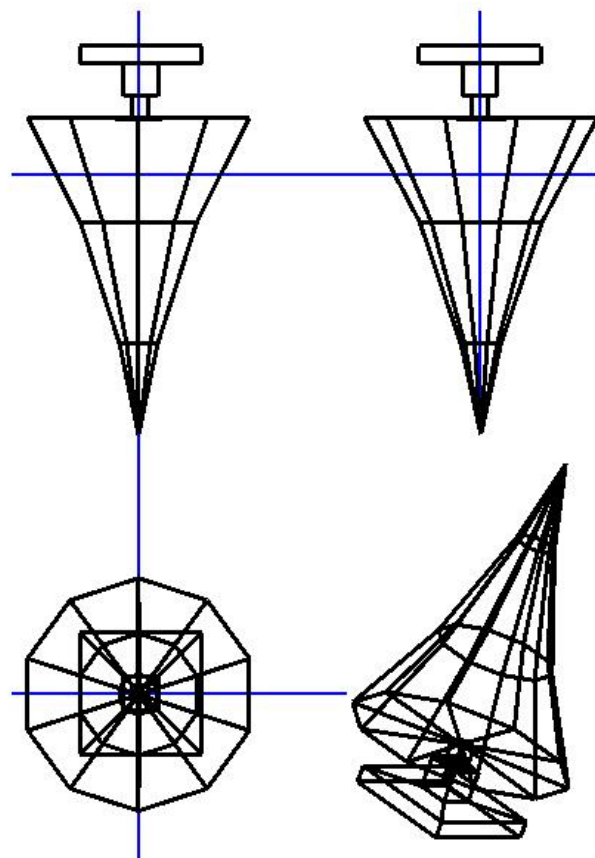
Ширина досок в
поддоне

19



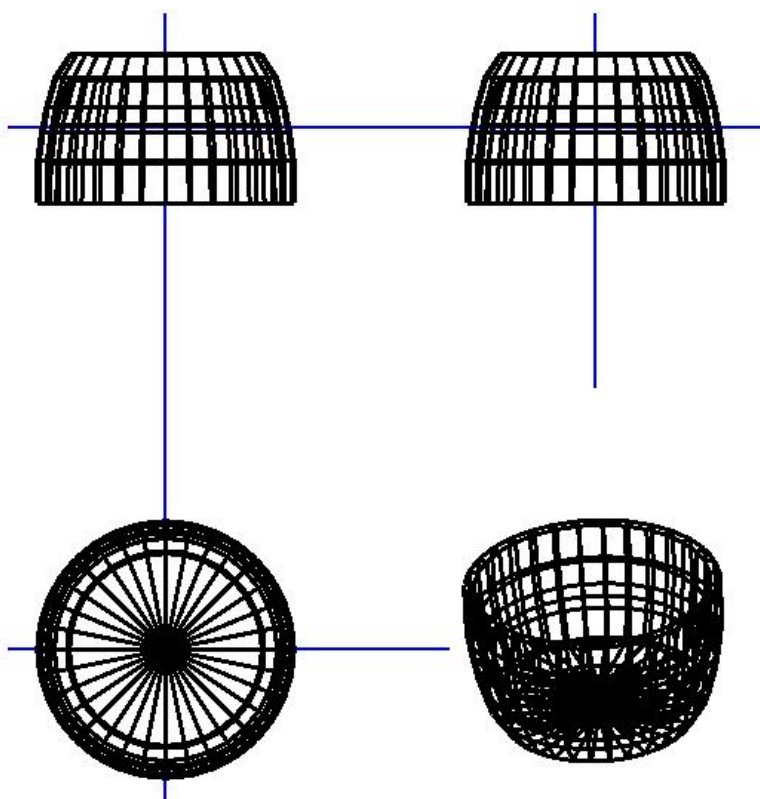
Ширина ножек
стола

20



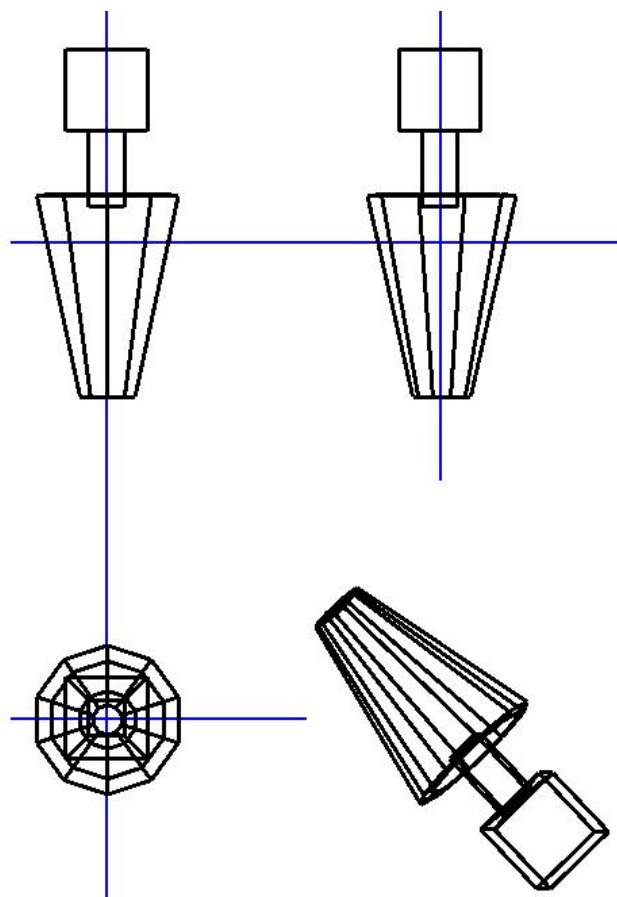
Количество п-
угольников
между
основанием и
макушкой и
количество
углов (фигура
получится
более гладкая).

21



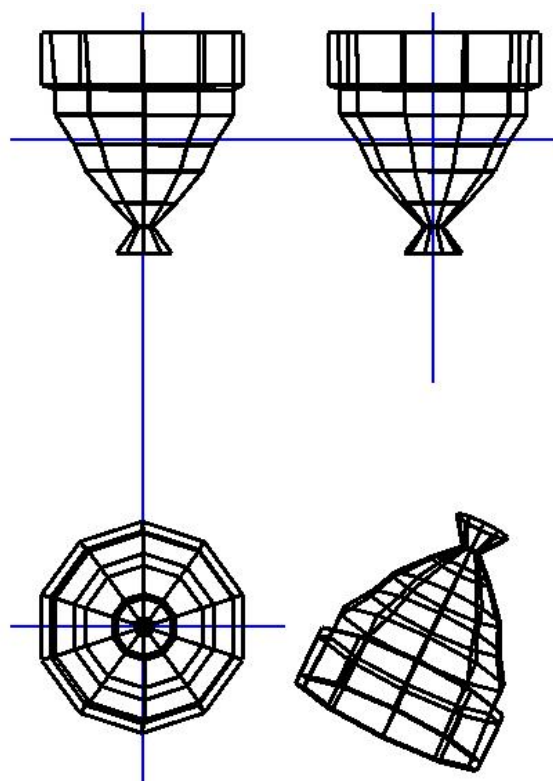
Количество п-
угольников
между дном и
горлышком и
количество
углов (фигура
получится
более гладкая).

22

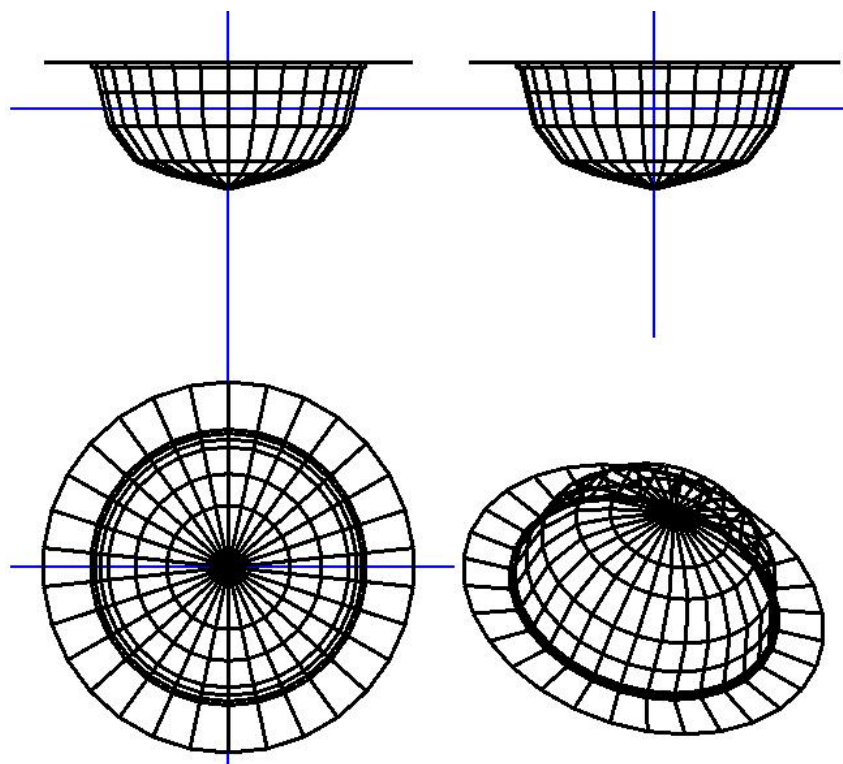


Количество п-
угольников
между
основанием и
верхней частью
торшера и
количество
углов (фигура
получится
более гладкая).

23



Количество п-
угольников
между
основанием и
помпоном и
количество
углов (фигура
получится
более гладкая).



Количество n -
угольников
между
основанием и
макушкой и
количество
углов (фигура
получится
более гладкая).