МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Озерский технологический институт — филиал НИЯУ МИФИ

Кафедра прикладной математики

Вл. Пономарев

ПРАКТИКУМ

по компьютерной графике

Учебно-методическое пособие
по дисциплине «Инженерная и компьютерная графика»

Часть 3. Полигональные модели и преобразования

2017 г.

УДК 681.3.06
П 56

Вл. Пономарев. Практикум по компьютерной графике. Учебно-методическое пособие по дисциплине «Инженерная и компьютерная графика». Часть 3. Полигональные модели и преобразования. Озерск: ОТИ НИЯУ МИФИ, 2017. — 18 с.

В пособии подробно излагается, как выполнять практические работы по дисциплине «Инженерная и компьютерная графика». Работы третьей части изучения дисциплины включают в себя геометрические полигональные модели, преобразования моделей в пространстве, проекции.

В качестве основного материала при выполнении практических работ пособие предназначено для студентов, обучающихся по направлению подготовки 09.03.01 «Информатика и вычислительная техника» и специальности 09.05.01 «Применение и эксплуатация автоматизированных систем специального назначения».

Рецензенты:
1.
2.

УТВЕРЖДЕНО
Редакционно-издательским
Советом ОТИ НИЯУ МИФИ

Содержание

Общие цели занятий

В ходе практических работ данной части изучения основ компьютерной графики студенты исследуют использование структур данных для представления полигональных моделей, создают графические классы и разрабатывают функции трехмерных преобразований.

1. Работа КГ-301. Видовое преобразование

Цели:
- формирование графических классов.

Задачи:
- разработка класса пространственной точки;
- разработка класса ребра полигональной модели;
- формирование графического объекта «параллелепипед»;
- разработка функции видового преобразования.

Поддержка:

Для выполнения работы требуется подготовленный проект для среды разработки Microsoft Visual Studio, предоставляемый преподавателем.

1.1. Проект приложения

Скачайте с сайта преподавателя или иным образом получите подготовленный преподавателем проект приложения trans3d.

Установите проект в корневой каталог диска C:.

Откройте проект, убедитесь, что он компилируется и выполняется.

Примерный внешний вид окна приложения приведен на рисунке 1.

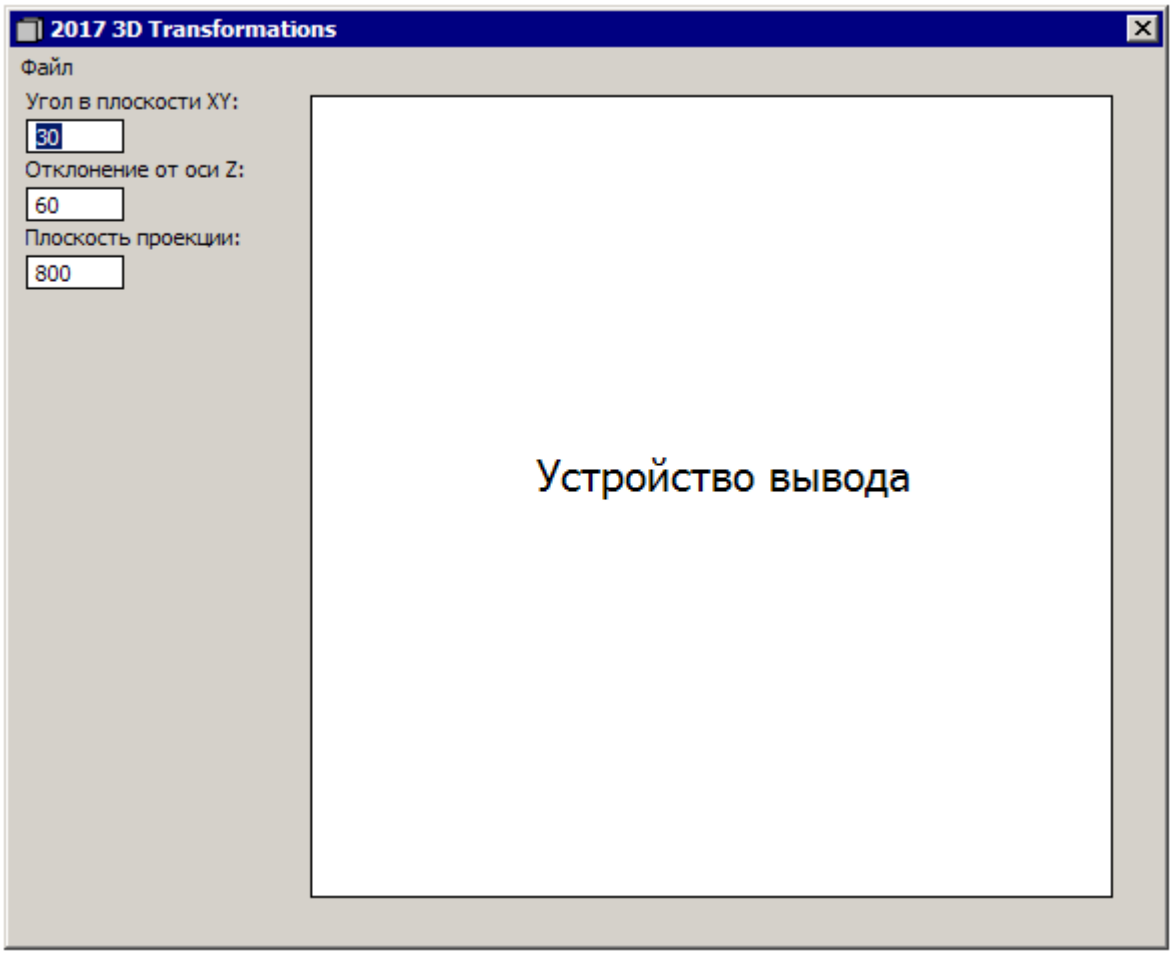

Рисунок 1 — Окно приложения

В окне приложения есть три поля для ввода сферических координат точки зрения и область для вывода изображения — устройство вывода.

Для изменения углов точки зрения используются клавиши со стрелками «Вниз», «Вверх», «Влево» и «Вправо». Удержание этих клавиш непрерывно изменяет угол на небольшую величину, что позволяет непрерывно изменять точку зрения, «вращать» объект.

Изменение положения плоскости проекции производится редко, но если это значение изменено, для отображения результата необходимо нажать клавишу со стрелкой «Вверх» и далее, используя клавиши со стрелками, изменить углы наблюдения.

В проект включены следующие модули, используемые в работе:

grass.h — модуль для определения графических классов;

trans.h — модуль для описания функций преобразований;

scene.h — модуль для определения графических объектов, формирования сцены, выполнения преобразований и рисования объектов на устройстве вывода.

1.2. Общее описание работы

Сначала нужно разработать класс трехмерной точки и класс, описывающий ребро полигональной модели.

Далее необходимо сформировать матрицу координат объекта «параллелепипед», «цилиндр» или «конус» и описать ребра этого объекта.

Затем необходимо описать видовое преобразование и вывод ребер объекта в устройство вывода.

В результате этих действий на устройстве вывода должно получиться трехмерное изображение объекта.

В заключение описывается простое перспективное преобразование и удаление невидимых ребер.

В программе есть следующие функции в модуле scene.h:

Box() — функция, в которой задаются матрица координат и ребра стандартного объекта «параллелепипед».

Cylinder() — функция, в которой задаются матрица координат и ребра стандартного объекта «цилиндр».

Cone() — функция, в которой задаются матрица координат и ребра стандартного объекта «конус».

User() — функция, в которой задаются матрица координат и ребра объекта, определяемого домашним заданием.

Scene() — функция, в которой устанавливаются начальные значения точки зрения, выбирается полигональный объект и выполняются аффинные преобразования объекта.

DrawView() — функция, в которой выполняется последовательность проективного и финального преобразований, а также рисование ребер объекта на устройстве вывода.

1.3. Порядок преобразований

Сформированный объект подвергается нескольким преобразованиям в следующей последовательности:

1) аффинные преобразования, такие, как перенос, поворот, отражение, сдвиг или произвольное преобразование;

2) проективное преобразование, такое, как диметрическая и изометрическая проекции, проекция Кавалье и кабинетная, прямоугольная проекция, видовое преобразование;

3) финальное преобразование, которое может отсутствовать или быть перспективным, обычно простым перспективным преобразованием.

Аффинные преобразования описываются в функции Scene(), а проективные преобразования и финальное преобразование описываются в функции DrawView() модуля scene.h.

При этом используются три матрицы координат:

WCO — мировые координаты. В этой матрице задаются вершины объекта и выполняются аффинные преобразования.

ECO — видовые координаты. В этой матрице находятся координаты вершин объекта после проективного преобразования.

SCO — экранные координаты. В этой матрице находятся координаты вершин объекта после финального преобразования. Эти координаты используются непосредственно для рисования (рисунок 2).

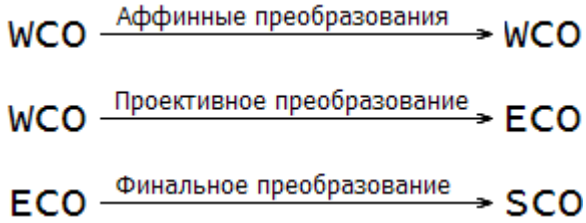

Рисунок 2 — Последовательность преобразований

Необходимо следить за правильным использованием матриц.

В функциях преобразований параметр M2, — это матрица, в которую записываются координаты после преобразования, параметр M1, — это матрица исходных координат, параметр size, — это размер матриц.

Рисование объекта на устройстве вывода всегда должно выполняться с использованием матрицы экранных координат.

Аффинные преобразования могут отсутствовать.

При этом они просто не описываются.

Если финальное преобразование не требуется, используется преобразование Empty, которое просто копирует одну матрицу в другую.

Для любого преобразования, кроме видового, финальное преобразование всегда пустое. Для видового преобразования финальным является либо пустое преобразование, либо перспектива.

1.4. Выбор графического объекта

Выбор графического объекта в проекте производится при помощи условной трансляции. Для этого определены следующие символы:

GO_USER — символ объекта пользователя (по заданию);

GO_BOX — символ объекта «параллелепипед»;

GO_CYL — символ объекта «цилиндр»;

GO_CONE — символ объекта «конус».

В начале модуля scene.h следует определить символ GO (*graphical object*), задав ему значение одного из перечисленных выше символов.

В модуле все определения есть:

```
//#define GO GO_USER
#define GO GO_BOX
//#define GO GO_CYL
//#define GO GO_CONE
```

В этом примере выбран объект «параллелепипед».

Нельзя определять символ GO дважды!

Только одна строка должна быть раскомментированной.

1.5. Выбор проективного и финального преобразований

Проектирование преобразование определяется значением переменной proj, которая имеет тип перечисления projections. Аналогичным образом финальное преобразование определяется значением переменной final, которая имеет тип перечисления finals.

Выбор преобразований производится в начале функции Scene:

```
// формирование сцены
void Scene() {
    // выбор проективного преобразования
    proj = P_DIMET;      // диметрия
    proj = P_ISOMET;     // изометрия
    proj = P_CAVALIER;   // Кавалье
    proj = P_CABINET;    // кабинетная
    proj = P_VIEW;       // видовое
    // выбор финального преобразования
    final = F_NONE;      // нет
    final = F_PERSPECT; // простая перспектива
```

В приведенном примере выбрано видовое проективное преобразование и перспективное финальное преобразование. Выбор других преобразований осуществляется перестановкой строк.

Выбор финального преобразования имеет значение только для видового проективного преобразования.

Для других проективных преобразований выполняется пустое преобразование.

1.6. Описание класса точки

Переходим в модуль grass.h и описываем класс пространственной, трехмерной точки.

Первоначально в модуле есть только следующее описание класса:

```
// класс "точка"
struct point {
    // координаты
    // конструктор по умолчанию
    // задает координаты
};
```

Следом за комментарием «координаты» нужно описать элементы данных класса, коими являются координаты X, Y и Z вещественного типа двойной точности.

Конструктор по умолчанию задает начальные нулевые значения координат X, Y и Z. Описываем конструктор после комментария «конструктор по умолчанию».

Метод set() с тремя параметрами x, y, и z задает новые значения координат. Метод ничего не возвращает. Описываем метод после комментария «задает координаты».

После описания класса точки в модуле описывается тип matrix:

```
// тип ссылки на матрицу
typedef point * matrix;
```

1.7. Описание класса ребра

Ребро задается двумя целыми числами, которые указывают номера вершин в матрице координат, которые этим ребром соединяются. Эти числа можно обозначить идентификаторами A и B. Чтобы рисовать только видимые ребра, нужно задать еще две вершины, которые будем обозначать идентификаторами C и D. Назначение этих двух вершин будет разъяснено позднее.

Первоначально в модуле имеется следующее описание класса ребра:

```
// класс "ребро"
struct edge {
    // номера вершин ребра
    // конструктор по умолчанию
    // задает вершины
    void set(int a, int b, int c = 0, int d = 0) {
    }
    // рисует ребро на устройстве hdc
    void draw(HDC hdc, matrix M) {
    }
};
```

После комментария «номера вершин ребра» описываем четыре элемента данных целого типа с названиями A, B, C и D.

После комментария «конструктор по умолчанию» описываем конструктор, задающий нулевые значения всем элементам данных.

Метод set() задает новые значения элементов данных.

Обратим внимание, что параметры c и d описаны как необязательные, однако их тоже нужно использовать для задания значений.

Метод draw() рисует ребро на указанном параметром hdc устройстве.

Для рисования следует использовать функцию DrawEdge, параметрами которой являются контекст устройства и четыре координаты.

1.8. Определение объекта «параллелепипед»

Сначала нужно подсчитать количество вершин объекта и количество ребер. Подсчитанные значения необходимо установить в определениях констант POINTS и EDGES.

Константа POINTS задает количество вершин объекта, и, соответственно, размер матриц. Константа EDGES задает количество ребер объекта, и, соответственно, размер массива edges, который описывает ребра.

В тексте модуля определение констант POINTS и EDGES встречаются несколько раз. Нужно выбрать тот блок, который соответствует объекту «параллелепипед», и значению символа GO, равному GO_BOX:

```
#elif GO == GO_BOX
    // количество точек параллелепипеда
    #define POINTS 8
    // количество ребер параллелепипеда
    #define EDGES 12
#elif GO == GO_CYL
```

Дополнительно нужно задать константы BX, BY и BZ, определяющие половинные размеры параллелепипеда. Эти константы задаются перед функцией Box():

```
    // половинные размеры
    #define BX 40
    #define BY 60
    #define BZ 80
```

Затем нужно перейти в функцию Box(), и задать координаты вершин и вершины ребер.

Центр системы координат находится в центре параллелепипеда, поэтому используются половинные размеры. Вершины можно задавать в произвольном порядке, но лучше систематично, сначала вершины нижней грани, затем вершины верхней грани, в одном и том же порядке.

На рисунке 3 показан пример задания точек.

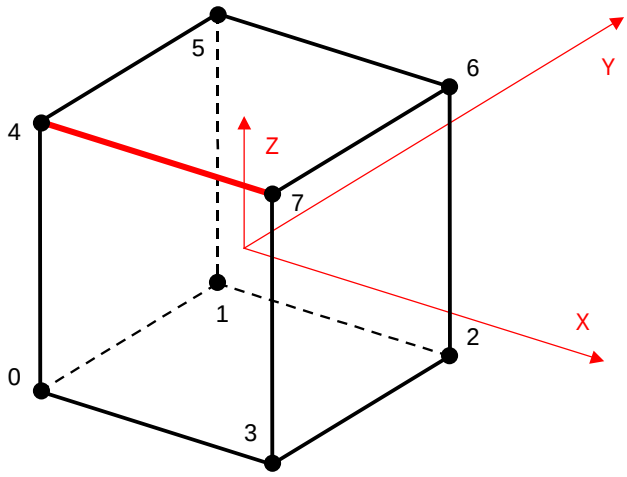

Рисунок 3 — Вершины и ребра параллелепипеда

Например, вершина 0 может быть задана следующим образом:

```
// параллелепипед
void Box() {
    // координаты вершин
    WCO[0].set(-BX, -BY, -BZ);
    // ребра
}
```

Далее нужно описать ребра.

Как было сказано, ребра задаются номерами двух вершин.

Для вывода проволочной модели без удаления невидимых ребер порядок указания вершин не имеет никакого значения. Однако в дальнейшем мы попробуем нарисовать только те ребра, которые видны. А для этого вершины нужно указывать в определенном порядке.

Во-первых, нужно выбрать грань, которой ребро принадлежит. А ребро параллелепипеда принадлежит двум граням одновременно.

Далее нужно смотреть на грань с ее внешней стороны. Тогда нужно задать две точки так, чтобы они располагались в порядке обхода вершин против часовой стрелки. При этом сразу нужно задать третью точку, которая находится в том же порядке на этой грани, и четвертую точку, которая находится на второй грани в противоположном порядке.

В качестве примера зададим ребро 4-7 (рисунок 3).

Ребро принадлежит граням 0-3-7-4 и 4-5-6-7. Предположим, что первой гранью является грань 0-3-7-4. Тогда, если смотреть на грань сверху, в порядке против часовой стрелки вершины 4 и 7 встречаются в последовательности сначала 7, затем 4. Третьей вершиной этой грани может любая из оставшихся.

Для второй грани в порядке обхода вершин уже по часовой стрелке, вершины 4 и 7 встречаются в последовательности 7-4, и третьей вершиной для этой грани будет любая из оставшихся.

Можно рассчитать вершины следующим образом.

Выбираем любую последовательность вершин грани, например, 4-7.

Далее выясняем, для какой грани при обходе вершин против часовой стрелки выполняется последовательность 4-7. Для нашего примера это грань 4-5-6-7. Тогда третьей вершиной ребра будет любая из оставшихся вершин этой грани, а четвертой вершиной будет любая из оставшихся вершин другой грани.

Задать ребро 4-7 можно при помощи следующего примера кода:

```
edges[0].set(4, 7, 6, 3);
```

1.9. Рисование ребер

В конце функции scene.DrawView() отведено место для рисования объекта, то есть для вывода его ребер:

```
void DrawView(HDC hdc) {
    . . .
    //
    // рисуем ребра
    //
}
```

Рисование заключается в формировании цикла for по всем ребрам.

В цикле вызывается метод edge::draw().

Параметрами метода являются контекст устройства hdc и матрица SCO.

На рисунке 4 приведен пример изображения, которое должно при этом получиться при запуске приложения, в котором реализован вывод ребер.

Рисунок 4 — Проволочная модель без преобразований

Мы наблюдаем изображение со стороны оси Z.

1.10. Видовое преобразование

Теперь нам нужно описать видовое преобразование.

Параметрами видового преобразования являются исходящая матрица M2, входящая матрица M1, размер матрицы size, угол точки зрения a в плоскости XY, угол точки зрения b отклонения от оси Z, расстояние до точки зрения и картинной плоскости d.

Координаты видового преобразования вычисляются по формулам, приведенным в учебно-методическом пособии [1].

Требуется левостороннее видовое преобразование, которое не разворачивает ось Y.

Надо помнить, что углы передаются в функцию в градусах, и их нужно преобразовать в радианы следующим образом (угол a):

```
a *= PI180;
```

Далее нужно вычислить синусы и косинусы углов a и b, обозначая их переменными sina, cosa, sinb, cosb.

Вычисления по расчетным формулам выполняются в цикле.

Лучше вычислять координаты в локальные переменные x, y, z, а потом присвоить их элементам выходной матрицы.

Если все сделано верно, при запуске приложения мы увидим изображение, показанное на рисунке 5.

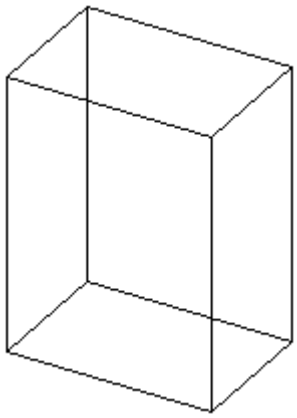

Рисунок 5 — Видовое преобразование

1.11. Простое перспективное преобразование

Полученное изображение кажется немного искаженным потому, что в нем нет перспективы. Поэтому далее нужно описать простое перспективное преобразование.

Дополнительным параметром функции простого перспективного преобразования SimplePerspect является расстояние до плоскости проецирования d, которое у нас совпадает с расстоянием до точки зрения Plane.

Вычисление координат перспективы производится по формулам:

$$x = d \cdot X / Z$$
$$y = d \cdot Y / Z$$
$$z = Z$$

Программируем это преобразование. Результат применения простого перспективного преобразования показан на рисунке 6.

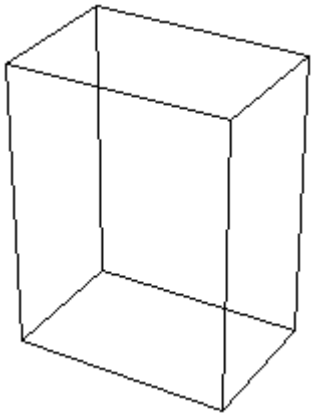

Рисунок 6 — Простая перспектива

Величина перспективы зависит от расстояния до плоскости проекции, которое можно изменять в небольших пределах, чтобы исследовать зависимость.

1.12. Удаление невидимых ребер

Удаление невидимых ребер или их частей является далеко не простой задачей. Однако в частных случаях, когда выводится один простой (выпуклый) объект, эта задача может быть решена легко.

Ребро видно, если грань, которой оно принадлежит, видна.

Определить видимость грани можно при помощи вектора нормали грани. Вектор нормали грани, — это вектор, перпендикулярный грани и направленный во внешнюю от объекта сторону. Вычислить вектор нормали можно по известным формулам.

Нас интересует не весь вектор нормали, а только его координата z.

Если эта координата положительна, грань видна.

Координата z вектора нормали вычисляется по трем точкам плоскости грани, иначе говоря, по любым трем вершинам грани.

Обозначим эти вершины буквами A, B и C.

Для правильного вычисления требуется, чтобы точки A, B и C были такими, что если смотреть на грань со стороны вектора нормали (с внешней, наблюдаемой стороны грани), то точки должны встречаться при обходе их против часовой стрелки в порядке A, B, C.

Координата z вектора нормали вычисляется по следующей формуле:

```
z = (XB - XA) * (YC - YA) - (YB - YA) * (XC - XA)
```

Здесь XA — координата X вершины A, XB — координата X вершины B, и так далее.

Нужно также учитывать, что грань может быть не видна, а ребро видимо, поскольку оно принадлежат сразу двум граням. Поэтому координату z вектора нормали нужно вычислять для двух граней, первой и второй.

Обозначим первой гранью ту, для которой порядок обхода вершин противоположен часовой стрелке, соответственно, другая грань вторая.

Для первой грани координата z вектора нормали вычисляется по приведенной формуле.

Для второй грани вершина A заменяется вершиной B, вершина B заменяется вершиной A, а вершина C заменяется вершиной D.

Для вычисления координат z нормалей первой и второй граней в классе edge есть два метода: normal1 и normal2.

Опишем в этих методах вычисление координаты z вектора нормали.

Далее переходим в метод DrawView, в котором выводятся ребра.

Вычисляем в цикле вывода ребер два значения:

z1 — координата z вектора нормали первой грани;

z2 — координата z вектора нормали второй грани;

Если одновременно z1 и z2 меньше константы EPS, считаем, что обе грани не видны, и в этом случае ребро не рисуем (например, переходим к следующей итерации при помощи оператора continue).

Если все сделано правильно, при запуске приложения увидим объект без невидимых ребер (рисунок 7).

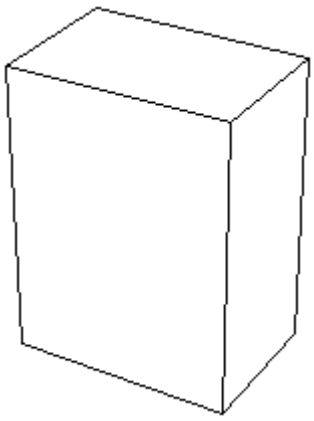

Рисунок 7 — Удалены невидимые ребра

2. Работа КГ-302. Проективные преобразования

Цели:
- разработка проективных преобразований;
- разработка аффинных преобразований.

Задачи:
- разработка функций диметрического, изометрического, свободного и кабинетного преобразований;
- проектирование объекта по заданию;
- разработка функций аффинных преобразований.

2.1. Проективные преобразования

Диметрическое преобразование описывается в функции Dimetric().
Изометрическое преобразование описывается в функции Isometric().
Проекция Кавалье описывается в функции Cavalier().
Кабинетная проекция описывается в функции Cabinet().
Функции находятся в модуле trans.h.
Расчетные формулы приведены в учебно-методическом пособии [1].
На рисунке 8 приведены исследуемые проекции для параллелепипеда.

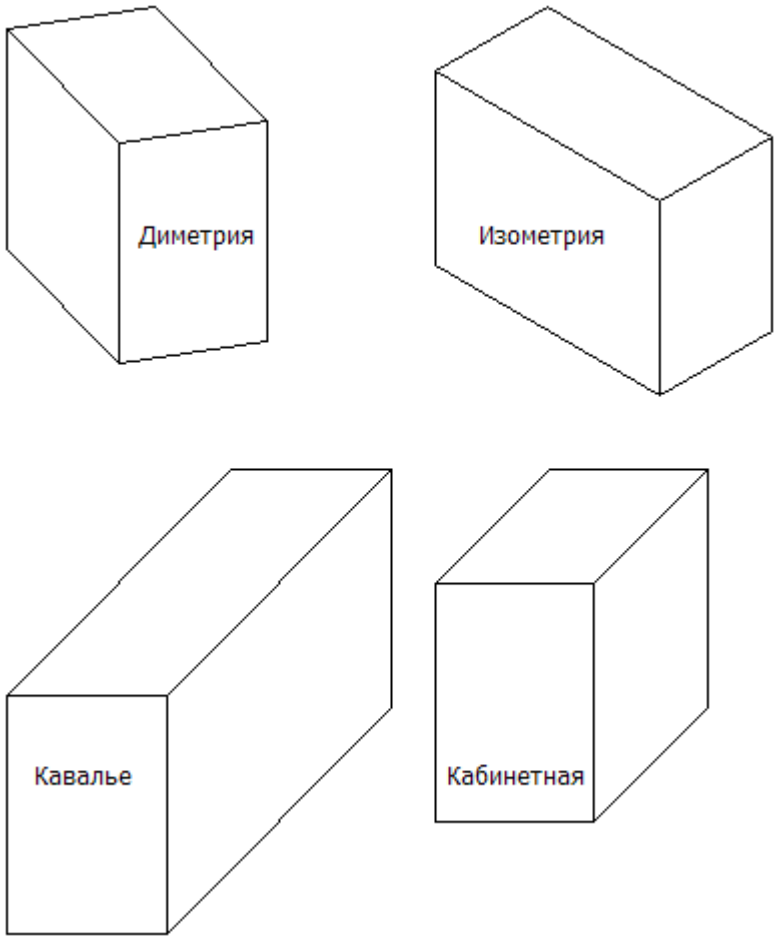

Рисунок 8 — Проективные преобразования

2.2. Разработка объекта по заданию

Выберите графический объект GO_USER.

Объект по заданию описывается в функции User(). Как правило, это одна буква латинского или кириллического алфавита.

Задайте константы размеров буквы UX, UY, UZ перед функцией User().

Учитывайте, что UX — это размер по ширине или ширина полосы буквы, UY — размер по высоте, UZ — размер по глубине (толщина буквы).

Определите количество вершин и ребер объекта, задайте их константами POINTS и EDGES в соответствующей секции модуля.

Задайте вершины и ребра.

Задавайте объект так, чтобы начало системы координат находилось в левом нижнем углу задней стороны объекта.

После того, как объект будет правильно рисоваться, переходите к описанию аффинных преобразований.

Вам надлежит показать следующие аффинные преобразования:

- перенос объекта одновременно вдоль всех осей, функция translation();
- вращение вокруг оси X, функция rotationX();
- вращение вокруг оси Y, функция rotationY();
- вращение вокруг оси Z, функция rotationZ().

Аффинные преобразования описываются в соответствующей секции функции scene():

```
    //
    // выбирается функция формирования графического объекта
    //
    // здесь же:
    //
    // аффинные преобразования
    //
#if GO == GO_USER
    User();
    // место для аффинных преобразований
#elif GO == GO_BOX
    Box();
#elif GO == GO_CYL
    Cylinder();
#elif GO == GO_CONE
    Cone();
#endif
```

3. Рекомендуемая литература

1. Пономарев В.В. Машинная графика. Учебное пособие. Ред. 2. Озерск: ОТИ МИФИ, 2006. — 72 с. ил.

2. Л. Аммерал. Принципы программирования в машинной графике. Пер. с англ. – М.: "Сол Систем", 1992. – 224 с.: ил.

3. Роджерс Д., Адамс Дж. Математические основы машинной графики: Пер. с англ. / Пер. Ю.П. Кулябичев, В.Г. Иваненко; ред. Ю.И. Топчеев. — М.: Машиностроение, 1980. — 240 с., ил.

4. Е.В. Шикин, А.В. Боресков. Компьютерная графика. Динамика, реалистические изображения. М.: "Диалог-МИФИ", 1995. — 288 с.