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

Кафедра прикладной математики

Вл. Пономарев

ПРАКТИКУМ

по компьютерной графике

Учебно-методическое пособие

Часть 3. Полигональные модели и преобразования

2018 г.

УДК 681.3.06
П 56

Вл. Пономарев. Практикум по компьютерной графике. Учебно-методическое пособие. Часть 3. Полигональные модели и преобразования. Озерск: ОТИ НИЯУ МИФИ, 2018. — 20 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 исследуется использование структур данных для представления полигональных моделей, создают графические классы и разрабатывают функции трехмерных преобразований.

В этой части работ изучаются следующие темы:

1) описание трехмерной структуры точки;

2) описание структуры ребра полигонального объекта;

3) описание полигональных объектов;

4) вычисление видового преобразования;

5) вычисление простого перспективного преобразования;

6) вычисление проективных преобразований: диметрическая и изометрическая проекции, проекция Кавалье, кабинетная проекция;

7) вычисление аффинных преобразований.

На выполнение работ этой части предположительно отводится 6-8 учебных часов, с учетом того, что работы выполняются в проекте, подготовленном преподавателем.

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.

Опорные документы:
[1, с.64].

1.1. Проект приложения

Скачайте с сайта преподавателя или иным образом получите подготовленный преподавателем архив проекта приложения transf. Извлеките из архива каталог transf в корневой каталог диска C:. Убедитесь, что целевой платформой является x86 или Win32.

Откройте проект, убедитесь, что он компилируется и выполняется.

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.

Рабочие модули проекта следующие:

grass.h — модуль для определения графических классов;

trans.h — модуль для описания функций преобразований;

box.h — модуль для определения параллелепипеда;

cyl.h —модуль для определения цилиндра;

user.h — модуль для определения объекта по заданию.

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.

После реализации всех функций проекта следует сформировать матрицу координат и ребра объекта по домашнему заданию.

В модуле draw.h задаются начальные значения для вывода:

```
// выбор начальных значений
void Initials() {
    //
    // выбор графического объекта
    //
    object = O_CYL;
    object = O_USER;
    object = O_BOX;

    //
    // выбор проективного преобразования
    //
    proj = P_NONE;       // пустое
    proj = P_DIMET;      // диметрия
    proj = P_ISOMET;     // изометрия
    proj = P_CAVALIER;   // Кавалье
    proj = P_CABINET;    // кабинетная
    proj = P_VIEW;       // видовое

    //
    // выбор финального преобразования
    //
    final_proj = F_NONE;         // нет
    final_proj = F_PERSPECT;     // простая перспектива

    //
    // начальные сферические координаты точки зрения
    //
    XAngle = 30;
    ZAngle = 60;
    Plane = 800;
}
// рисует объект на устройстве вывода
void DrawObject(HDC hdc) {
    // рисуем ребра
    for (int i = 0; i < EdgesNumber; i++) {
        double z1 = EDGES[i].normal1(SCO);
        double z2 = EDGES[i].normal2(SCO);
        if (remove_invisible && z1 < EPS && z2 < EPS) continue;
        EDGES[i].draw(hdc, SCO);
    }
}
```

Переставляя строки в функции Initials, задаются объекты и преобразования, которые будут использованы при старте программы. Это помогает быстрее сформировать необходимый код.

Используется тот объект и то преобразование, которое задано последним в списке. Здесь же находится функция DrawObject, которая рисует объект. Эта функция вызывается автоматически при изменении сцены.

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 модулях, описывающих объекты, после его описания, например, в модуле box.h:

```
// описание объекта "Параллелепипед"
// размер X
#define BX 40
// размер Y
#define BY 60
// размер Z
#define BZ 80
// количество вершин
#define BOX_POINTS 8
// количество ребер
#define BOX_EDGES 12
// задание вершин и ребер
void MakeBox() {
    // координаты вершин
    // ребра
    // аффинные преобразования объекта
}
```

Сначала задаем количество вершин `BOX_POINTS` и количество ребер объекта `BOX_EDGES`. Затем в функции MakeBox задаем координаты вершин, ребра, после чего описываем аффинные преобразования.

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.

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. 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;
```

1.5. Описание класса ребра

Ребро задается двумя целыми числами, которые указывают номера вершин в матрице координат, которые этим ребром соединяются. Эти числа обозначим идентификаторами A и B. Для рисования только видимых ребер нужно задать еще две вершины, идентификаторы 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 идентификаторами 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. Функция определена в модуле transf.h. Она также разворачивает ось Y.

1.6. Описание объекта «параллелепипед»

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

Константа BOX_POINTS задает количество вершин объекта, и, соответственно, размеры матриц. Константа BOX_EDGES задает количество ребер объекта, и, соответственно, размер массива, который описывает ребра.

Дополнительно нужно задать константы BX, BY и BZ, определяющие половинные размеры параллелепипеда. Затем нужно перейти в функцию MakeBox, 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.

Например, вершина 0 может быть задана так (рисунок 1):

```
void MakeBox() {
    // координаты вершин
    WCO[0].set(-BX, -BY, -BZ);
    // ребра
    // аффинные преобразования объекта
}
```

На рисунке 1 показан пример расчетной схемы параллелепипеда.

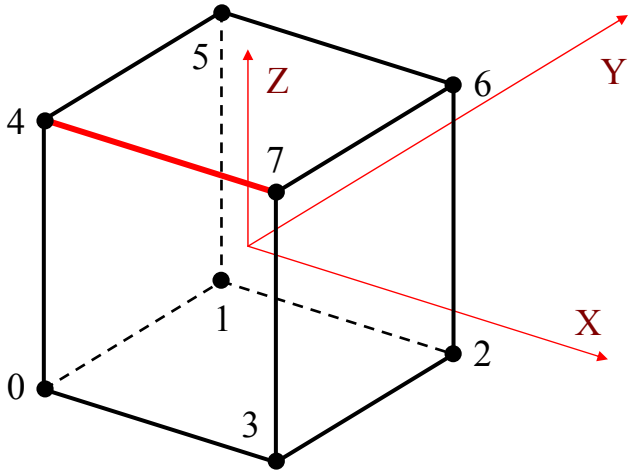


Рисунок 1 — Вершины и ребра параллелепипеда

Ребра задаются номерами двух вершин. Для вывода проволочной модели без удаления невидимых ребер порядок указания вершин не имеет значения. Однако позднее мы будем рисовать только те ребра, которые видны, поэтому вершины нужно указывать в определенном порядке.

Во-первых, нужно выбрать грань, которой ребро принадлежит. 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 1).

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:

```
void MakeBox() {
    // координаты вершин
    WCO[0].set(-BX, -BY, -BZ);
    . . .
    // ребра
    EDGES[0].set(0, 1, 2, 4);
    . . .
    // аффинные преобразования объекта
}
```

1.7. Рисование ребер

Ребра рисуются в функции DrawObject модуля draw.h:

```
// рисует объект на устройстве вывода
void DrawObject(HDC hdc) {
    // рисуем ребра
    for (int i = 0; i < EdgesNumber; i++) {
        EDGES[i].draw(hdc, SCO);
    }
}
```

Рисование заключается в формировании цикла for по всем ребрам. В цикле вызывается метод edge::draw(). Параметрами метода являются контекст устройства и матрица экранных координат.

Для вывода объекта дополнительно нужно описать пустое проективное преобразование в функции Empty модуля trans.h:

```
void Empty(matrix & M2, matrix M1, int size) {
    for (int i = 0; i < size; i++) {
        M2[i].X = M1[i].X;
        M2[i].Y = M1[i].Y;
        M2[i].Z = M1[i].Z;
    }
}
```

На рисунке 2 приведен пример изображения, которое должно получиться при запуске приложения, в котором реализован вывод ребер, при этом нужно выбрать проективное преобразование «Нет».

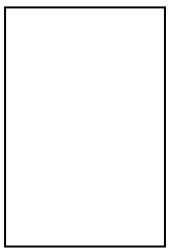


Рисунок 2 — Проволочная модель без преобразований

Мы наблюдаем изображение со стороны оси Z. Аффинных преобразований объекта не задано, центр системы координат в центре объекта.

1.8. Видовое преобразование

Описываем видовое преобразование в функции ViewPoint модуля trans.h. Параметрами видового преобразования являются исходящая матрица M2, входящая матрица M1, размер матрицы size, угол точки зрения α в плоскости XY, угол точки зрения β 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:

$x^* = y \cdot \cos\alpha - x \cdot \sin\alpha$

$y^* = z \cdot \sin\beta - x \cdot \cos\alpha \cdot \cos\beta - y \cdot \sin\alpha \cdot \cos\beta$

$z^* = d - z \cdot \cos\beta - x \cdot \cos\alpha \cdot \sin\beta - y \cdot \sin\alpha \cdot \sin\beta$

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

```
a *= PI180;
```

Далее нужно вычислить синусы и косинусы углов α и β, обозначая их переменными sinA, cosA, sinB, cosB.

```
double sinA = sin(a);
```

Вычисления по расчетным формулам выполняются в цикле.

Обычная ошибка в расчетах видового преобразования заключается в том, что вычисляется, например, координата *x*, и затем новая координата используется для расчета координаты *y*, что приводит к искажению формулы и объекта. Если расчеты верны, при запуске приложения мы увидим изображение, показанное на рисунке 3.

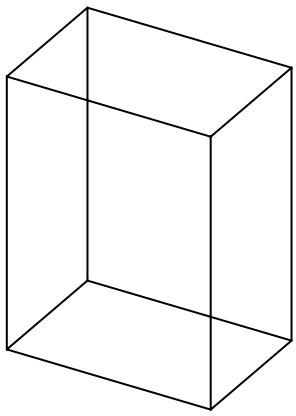


Рисунок 3 — Видовое преобразование

Нужно выбрать проективное преобразование «Точка зрения».

Использованы следующие параметры точки зрения:

XAngle = 30° (угол α), ZAngle = 60° (угол β), Plane = 800 (*d*).

1.9. Простое перспективное преобразование

Полученное изображение кажется немного искаженным потому, что в нем нет перспективы. Поэтому далее нужно описать простое перспективное преобразование в функции SimplePerspect модуля trans.h. Дополнительным параметром этой функции является расстояние до плоскости проецирования *d*, которое 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 преобразования показан на рисунке 4, выбрано финальное преобразование «Перспектива».

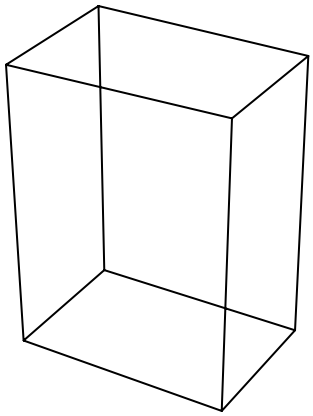


Рисунок 4 — Простая перспектива

Здесь использованы те же параметры точки зрения, что и раньше. Величина перспективы зависит от расстояния до плоскости проекции, которое можно изменять в небольших пределах для исследования зависимости.

1.10. 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:

$Nz = (x_B - x_A)\cdot(y_C - y_A) - (y_B - y_A)\cdot(x_C - x_A)$

Здесь *Nz* — координата *z* вектора нормали *N*, x_A — координата *x* вершины A, x_B — 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 *Nz* 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 *Nz* 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 DrawObject, 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 вычисления верны, при запуске приложения увидим объект без невидимых ребер (рисунок 5) при включенном флажке «Скрыть ребра».

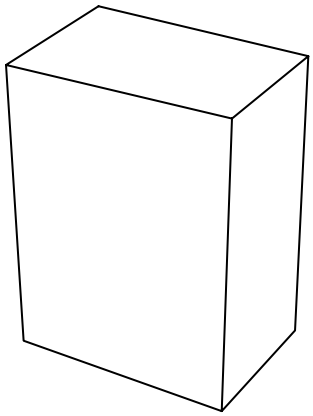


Рисунок 5 — Удалены невидимые ребра

Здесь использованы те же параметры точки зрения, что и раньше.

1.11. Описание объекта «цилиндр»

Переходим к описанию цилиндра, модуль cyl.h.

Эта задача сложнее, и связано это с тем, что число боковых прямоугольных граней объекта является переменной величиной, заданной константой CN. Если CN равно трем, получим трехгранную призму. Построим развертку боковых граней (рисунок 6, грань 2 нарисована дважды).

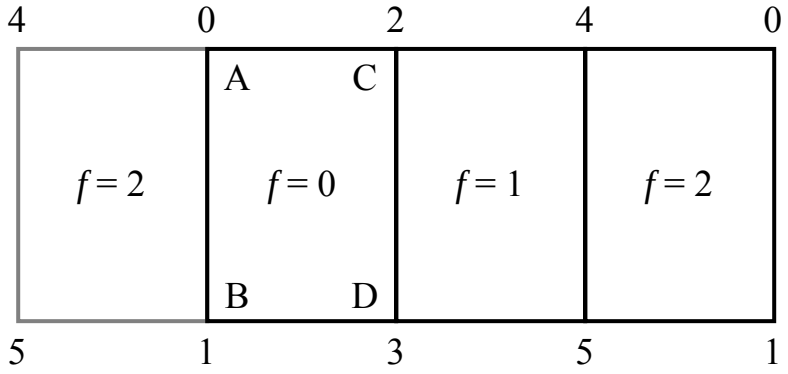
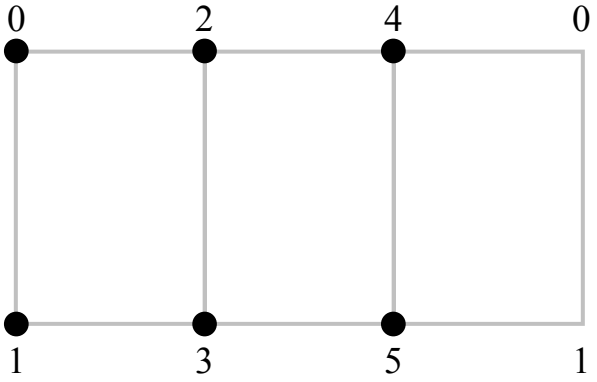
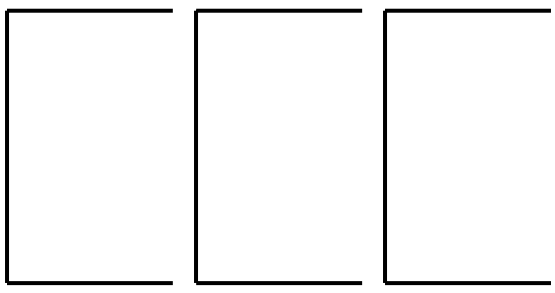


Рисунок 6 — Развертка трехгранной призмы

Здесь грани обозначены *f* (face) и пронумерованы от нуля.
Очевидно, что число вершин CYL_POINTS равно (CN+CN):



Очевидно, что число ребер CYL_EDGES равно (CN+CN+CN):



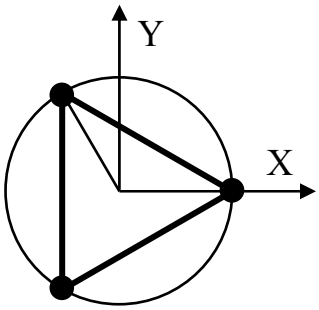
Будем вычислять координаты и ребра в цикле, в каждой итерации *f* которого задаются точки и ребра грани с условным номером *f*.

Обозначим номера вершин грани в итерации *f* буквами A, B, C и D. Считаем, что это идентификаторы целочисленных переменных. Тогда:

Итерация *f*	A	B	C	D
0	$2f = 0$	A + 1 = 1	A + 2 = 2	A + 3 = 3
1	$2f = 2$	A + 1 = 3	A + 2 = 4	A + 3 = 5
2	$2f = 4$	A + 1 = 5	A + 2 = 6	A + 3 = 7

Видим, что в последней итерации вершины C и D получают неправильные номера 6 и 7. Но если в итерации использовать эти значения по модулю CYL_POINTS, то 6 преобразуется в 0, а 7 в 1.

Перейдем к координатам:



Полный угол окружности равен 2π, угол между точками θ = 2π / CN. Константа PI в проекте определена. Угол ω для грани *f* вычисляется как *f*·θ. Тогда координаты *x* и *y* вычисляются по формулам:

$x = radius \cdot \cos\omega$

$y = radius \cdot \sin\omega$

Для верхних точек координата *z* равна половине высоты, для нижних точек — отрицательному значению половины высоты.

Зададим в том же цикле ребра. В каждой итерации задается три ребра. Обозначим номера ребер как R_1, R_2, R_3. Эти значения вычисляются от значения *f* по формулам $R_1 = 3 \cdot f$, $R_2 = R_1 + 1$, $R_3 = R_2 + 2$ (рисунок 7).

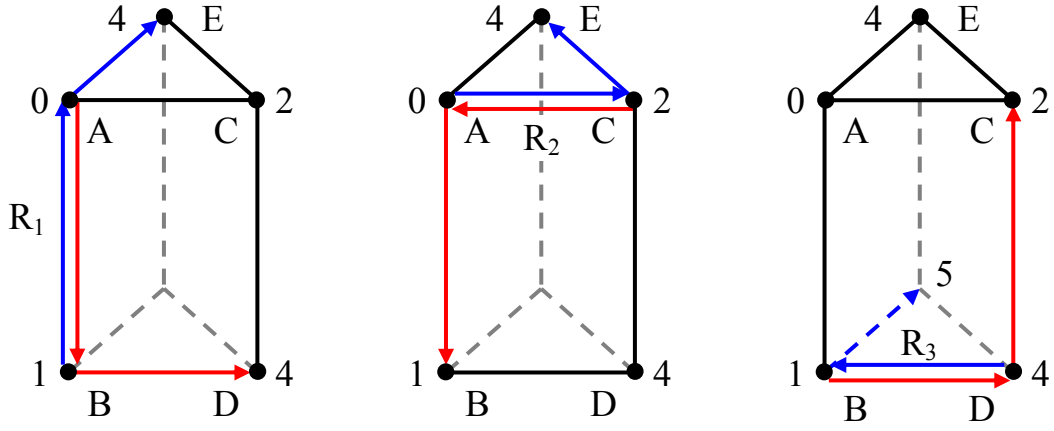


Рисунок 7 — Расчет вершин ребер

Для каждого ребра нужно задать четыре вершины.

Будем рассматривать итерацию $f = 0$.

Для ребра R_1 первая грань задается вершинами ABD, а вторая грань задается вершинами BAE, E — дополнительная точка. Следовательно, в конструктор ребра передаем вершины A, B, D и E. Красными стрелками на рисунке 7 обозначено направление обхода вершин первой грани, а синими стрелками — второй грани.

Для ребра R_2 первая грань задается вершинами CAB, вторая — ACE.

Для ребра R_3 первая грань задается вершинами BDC, вторая — DB5. Учитывая, что точка 5 — это точка E минус единица, задание третьего ребра также сводится к точке E.

Как вычислить номер точки E. Для нулевой грани он равен числу вершин минус два. Для других граней он на две единицы меньше номера вершины A.

Программируем цикл вычислений. Убедимся, что треугольная призма рисуется, невидимые ребра удаляются.

Если вы не уверены, что это полигональное представление цилиндра, задайте число боковых граней сначала 31, затем 111.

1.12. Описание объекта по заданию

Заключительная часть этой работы отведена для описания объекта по заданию. Возможные варианты объектов следующие.

Простые варианты:

1. Тетраэдр.
2. Октаэдр.
3. Додекаэдр.
4. Икосаэдр.
5. Одна из букв Z, Y, K, X, W, F, M, Ш, Я, Ъ.

Сложные варианты:

1. Конус (простой вариант).
2. Два конуса, совмещенные основаниями (чуть сложнее).
3. Сфера (средней сложности).
4. Полусфера (средней сложности).
5. Тор (сложный вариант).
6. Парашют (неполный телесный угол сферы).

Вариант выбирается по своим возможностям, при этом все варианты должны быть разными.

2. Работа КГ-302. Проективные преобразования

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

Задачи:
- разработка функций диметрического, изометрического, свободного и кабинетного преобразований;
- разработка функций аффинных преобразований.

Опорные документы:
[1, с.26]

2.1. Диметрическая и изометрическая проекции

Все функции преобразований описываются в модуле trans.h.
Диметрическое преобразование описывается в функции Dimetric.
Формулы диметрического преобразования:

$x^* = 0{,}935414x + 0{,}353553z$

$y^* = 0{,}133630x + 0{,}925820y - 0{,}353553z$

$z^* = 0{,}866025z - 0{,}327326x + 0{,}377964y$

Параллелепипед в этой проекции изображен на рисунке 8 слева.
Изометрическое преобразование описывается в функции Isometric.
Формулы изометрического преобразования:

$x^* = 0{,}707106(x + z)$

$y^* = 0{,}408248(x - z) + 0{,}816496y$

$z^* = 0{,}577350(z - x + y)$

Параллелепипед в этой проекции изображен на рисунке 8 справа.

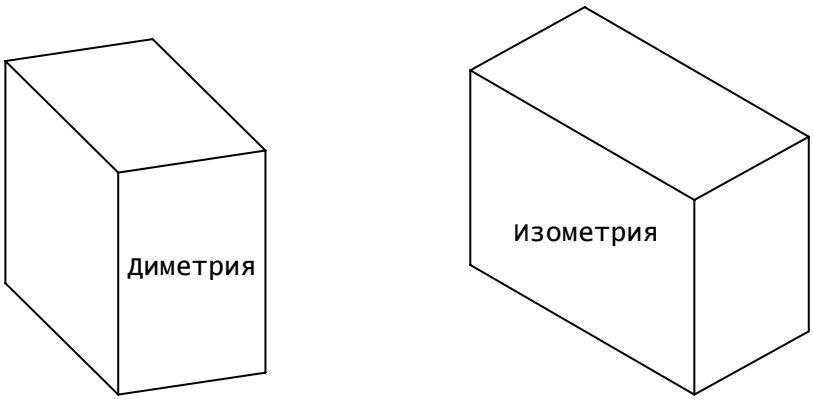


Рисунок 8 — Диметрическая и изометрическая проекции

2.2. Проекции Кавалье и кабинетная

Проекция Кавалье описывается в функции Cavalier, а кабинетная проекция — в функции Cabinet.

Формулы преобразования следующие:

$x^* = x - a \cdot z$

$y^* = y - a \cdot z$

$z^* = z$

Здесь a равно косинусу сорока пяти градусов для проекции Кавалье, и половина этого значения для кабинетной проекции.

Параллелепипед в этих проекциях изображен на рисунке 9.

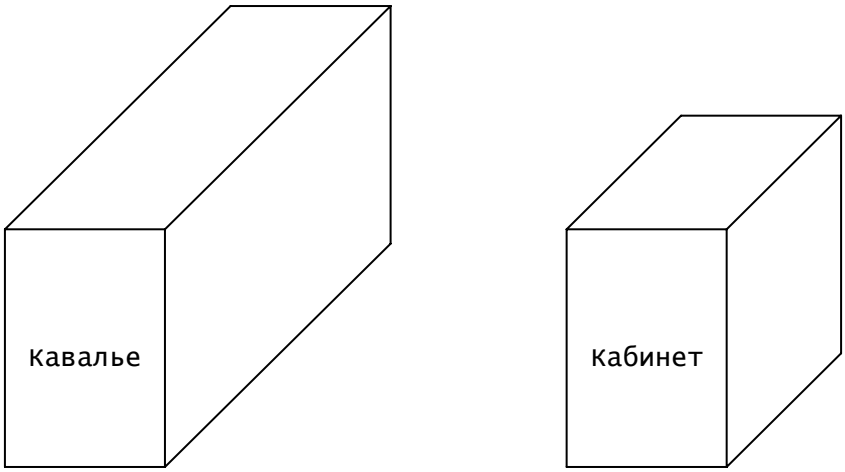


Рисунок 9 — Проекции Кавалье и кабинетная

2.3. Аффинные преобразования

Описываем следующие преобразования:

- перенос (функция Translate),
- вращение вокруг оси X (функция RotX),
- вращение вокруг оси Y (функция RotY),
- вращение вокруг оси Z (функция RotZ).

Формулы преобразований приведены в пособии [1].

Каждое из преобразований тестируем, используя объект «Цилиндр», наблюдая при этом объект в различных проекциях.

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.