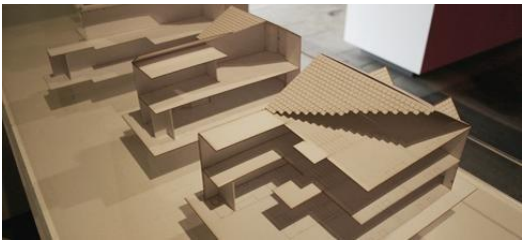
ФГБОУ ВПО «Воронежский государственный технический университет»
Кафедра компьютерных интеллектуальных технологий проектирования

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам № 4-5 по дисциплине «Программирование трехмерной графики» для студентов направления 09.03.02 «Информационные системы и технологии» всех форм обучения



Воронеж 2013

Составители: канд. техн. наук А.Н. Юров,
канд. техн. наук М.В. Паринов, ст.
преп. В.А. Рыжков
канд. техн. наук А.С. Левченко

УДК 004.9

Методические указания к лабораторным работам № 4-5 по дисциплине “Программирование” для студентов направления 09.03.02 «Информационные системы и технологии» всех форм обучения / ФГБОУ ВПО «Воронежский государственный технический университет»; сост. А.Н. Юров, М.В. Паринов, В.А. Рыжков, А.С. Левченко, Воронеж, 2013. 42 с.

Методические указания содержат практический материал по работе с графическими компонентами OpenGL и GLUT.

Предназначены для студентов 2 курса. Ил. 7. Библиогр.: 11 назв.

Рецензент д-р техн. наук, проф. А.В. Кузовкин

Ответственный за выпуск зав. кафедрой д-р техн. наук, проф. М.И. Чижов

Печатается по решению редакционно-издательского совета Воронежского государственного технического университета

ВВЕДЕНИЕ

Разработка приложений и подсистем САПР требует базовых знаний и приемов проектирования моделей, принципов, лежащих в основе описания поверхностей и кривых заданного вида. При имеющихся навыках, а также основываясь на знании какого-либо языка программирования, подготовить программную подсистему (электронную библиотеку) весьма непросто. Однако у учащихся есть возможность освоить методы и приемы программного моделирования с помощью облегченного комплекта программных решений, которые включает библиотека утилит OpenGL (glut) на языке C++. В методических указаниях приведены примеры, которые могут быть основой в последующих разработках подсистем САПР и позволят приобрести и закрепить полученные навыки на контрольных заданиях.

ЛАБОРАТОРНАЯ РАБОТА №4
МОДЕЛИРОВАНИЕ ПОВЕРХНОСТЕЙ

Цель работы: подготовить объектно-ориентированную модель приложения и выполнить графический вывод поверхности по заданным координатам.

Задачи и требования к проекту разработки:

1.Создать математическую модель некоторой поверхности, передать созданный массив в метод построения геометрических примитивов.

2.Подготовить объектно-ориентированную модель приложения.

3.Создать проект, в котором предусмотрен переход в полноэкранный режим работы с помощью клавиатуры.

Теоретические сведения

Прежде чем рассматривать вопросы, касающиеся поверхностного моделирования, добавим в проект приложения возможность отображать объекты в заданном разрешении при развертке на весь экран.

Получить доступ к режиму отображения объектов, используя не оконный вывод, а графические моды видеокарты и монитора, в glut достаточно просто. Приложение будет работать в так называемом game режиме. Для обеспечения вывода требуемых объектов на весь экран в приложение следует внести следующие строки:

```
//Режим Game
glutGameModeString("1366x768:60");
glutEnterGameMode();
```

Первая строка определяет разрешение экрана (количество точек по горизонтали и вертикали), а также указывается частота отображения. Причем частота должна соответствовать возможностям вывода дисплея ЭВМ. Таким образом, прежде чем определять тот или иной режим вывода, приложение должно запросить и получить список поддерживаемых режимов. Строка `glutEnterGameMode()` позволит перевести приложение в заданный полноэкранный режим работы. Выход из полноэкранного режима надо предусмотреть по отдельной клавише или комбинации клавиш, так как в случае с окном завершить работу приложения glut не получится (приложения glut используют бесконечный цикл обработки построений). Решение по выходу из программы может быть выполнено следующим образом:

```
void OnKeyPress (unsigned char key, int,int)
{
  if(key==27) exit(0);
}
```

В функции реализуется опрос клавиатурных кодов и если код соответствует клавише ”ESC”, производится выход в ОС.

Теперь рассмотрим примеры приложений, с помощью которых можно получить тот или иной вид поверхности. Подходов к реализации построения поверхностного контура может быть несколько и все они базируются на математических методах задания точек в пространстве по определенным правилам, с помощью ключевых зависимостей, некотором случайным набором координат, характеризующих высоты задания уровня некоторой поверхностной сетки и т.д. Кроме всего прочего, в glut можно непосредственно задать набор точек (например, воспользовавшись массивом или структурой координат ключевых точек) и соединить их между собой. Чем меньше будет дискретность между точками, тем точнее будут

выполнены переходы между ними. Можно напрямую задать координаты точек в функции `glVertex3d(x,y,z);` и в секции `glBegin(GL_LINES); glEnd();` соединить их между собой, что приведет к созданию простейшего поверхностного построения. Следующий пример реализует подобные построения. Для поверхности определены 15 ключевых точек с координатами по x,y,z и построены линии связи между точками. Кроме того, в примере реализована обработка сообщений от клавиатуры с использованием курсорных клавиш для изменения взгляда на построенный объект.

```
    #include <GL/glut.h>
    #include <iostream>
    #include <GL/gl.h>
//класс, в котором производятся расчеты и
//вычисления позиций точек class
    base_geometry
    {
    protected:
       static int rx,ry;
    static void build()
       {
      glBegin(GL_LINES);
    glColor3d(0,1,0); //Профильный
    контур 1   glVertex3d(-5,5,0);
    glVertex3d(-3,5,5);
    glVertex3d(-3,5,5);
    glVertex3d(0,5,0);
    glVertex3d(0,5,0);
    glVertex3d(3,5,1);
    glVertex3d(3,5,1);
    glVertex3d(5,5,0); //Профильный
    контур 2   glVertex3d(-5,0,0);
    glVertex3d(-3,0,1);
    glVertex3d(-3,0,1);
    glVertex3d(0,0,0);
```

```
    glVertex3d(0,0,0);
    glVertex3d(3,0,1);
    glVertex3d(3,0,1);
    glVertex3d(5,0,0); //Профильный
    контур 3   glVertex3d(-5,-5,0);
    glVertex3d(-3,-5,1);
    glVertex3d(-3,-5,1);
    glVertex3d(0,-5,0);
    glVertex3d(0,-5,0);
    glVertex3d(3,-5,5);
    glVertex3d(3,-5,5);
    glVertex3d(5,-5,0);
    //Соединение поперечное между 2 точками №1
    glVertex3d(-5,-5,0); glVertex3d(-5,0,0);
    //Соединение поперечное между 2 точками №2
    glVertex3d(-5,0,0); glVertex3d(-5,5,0);
    //Соединение поперечное между 2 точками №3
    glVertex3d(-3,-5,1); glVertex3d(-3,0,1);
    //Соединение поперечное между 2 точками №4
    glVertex3d(-3,0,1); glVertex3d(-3,5,5);
    //Соединение поперечное между 2 точками №5
    glVertex3d(0,-5,0); glVertex3d(0,0,0);
    //Соединение поперечное между 2 точками №6
    glVertex3d(0,0,0); glVertex3d(0,5,0);
    //Соединение поперечное между 2 точками №7
    glVertex3d(3,-5,5); glVertex3d(3,0,1);
    //Соединение поперечное между 2 точками №8
    glVertex3d(3,0,1); glVertex3d(3,5,1);
    //Соединение поперечное между 2 точками №9
    glVertex3d(5,-5,0); glVertex3d(5,0,0);
    //Соединение поперечное между 2 точками №10
    glVertex3d(5,0,0); glVertex3d(5,5,0);
    glEnd();    }
    };
    int base_geometry::rx=0; int
    base_geometry::ry=0;
//Класс по заданию начальных параметров сцен и
```

```
//вызову расчетных функций class
    base:public base_geometry
    {
    public:
       static void OnReshape(int w, int h)
        {
//При делении на 0, присвоить следующее значение
    if (h==0)   h=1;
//установить область окна отображения
    glViewport(0,0,w,h);
//установить матрицу проекции
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
//используем перспективную проекцию
    gluPerspective(45,(float)w/h,0.1,100);
//возврат к матрице моделирования, чтобы была
//возможность перемещать построенный объект
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
       }
      static void OnDraw()
      {
     glClear(GL_DEPTH_BUFFER_BIT|GL_COLOR_BUFFER_
BIT);
    glLoadIdentity();
     gluLookAt( rx,ry,10,
                    0,0,0,
                    0,1,0);
//Вызов функции моделирования поверхности
    build();  glutSwapBuffers();
    }
    static void OnInit()
    {
     glEnable(GL_DEPTH_TEST);
    }
    static void OnKeyPress(int key,int,int) {
```

```
 {           switch(key) {
case        GLUT_KEY_LEFT:
rx++;           if (rx>40)
rx=40;      break;    case
GLUT_KEY_RIGHT:        rx-
-;        if (rx<-40) rx=-
40;        break;     case
GLUT_KEY_UP:         ry++;
if     (ry>30)      ry=30;
break;                case
GLUT_KEY_DOWN:        ry--
;         if (ry<-30) ry=-
30;        break;

default:
      break;
 }
}
 glutPostRedisplay();
}
static void OnExit()
{
} };
int main(int argc,char** argv)
{
 base *pointer=new base();
//начальные установки для glut
    glutInit(&argc,argv); //запрос на
    использование буфера грубины, RGBA
//режима отображения и двойной буферизации при
//выводе
    glutInitDisplayMode(GLUT_DEPTH|GLUT_RGBA|GLU
T_DOUBLE);
//размер окна вывода
    glutInitWindowSize(800,600);
//создание окна с заголовком
    glutCreateWindow("Build plato:  Up/Down-
```

```
round Y,Left/Right-round X");
//вызов функции по отображению сцены
    glutDisplayFunc(pointer->OnDraw);
//функция обработки изменений в заданном окне
    glutReshapeFunc(pointer->OnReshape);
//функция по обработке событий от специальных
//клавиш  glutSpecialFunc(pointer-
    >OnKeyPress);
//вызов метода с пользовательскими установками
    pointer->OnInit();
//производится вызов функции, когда приложение
//завершает свою работу  atexit(pointer-
    >OnExit);
//функция обеспечивает выполнение приложения в
//некотором цикле
    glutMainLoop();  return
    0;
    }
```

На рисунке 1 представлена работа приложения по моделированию некоторого контура поверхности. Видно, что поверхность однозначно определена, нет возможности детализировать объект, отсутствуют сглаживающие переходы и т.д. Все указанные недостатки можно решить с помощью аналитического расчета и задания по некоторой предопределенной структуре данных.

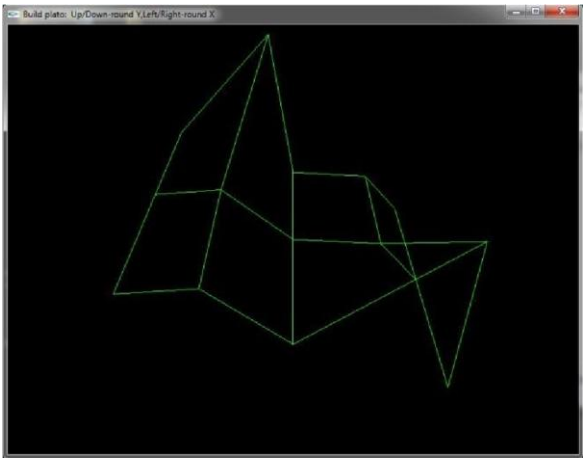


Рис. 1. Простейшее представление поверхности

В следующем примере показано, как произвести построение поверхности, которая задана группой точек (базовые 16), а от них получить направляющие для детализации объекта. Детализация поверхности определяется клавишами '+“ и „-„„ повороты по осям x и y, клавишами 'a' и 'd', 'w“ и “x“. Однако если клавиатурная раскладка будет работать в кириллическом режиме или режим Caps Lock активен, вращение объекта получено не будет. Код подробно прокомментирован, а на рисунке 2 представлен результат работы glut-приложения.

```
    #include <GL/glut.h>
    #include <GL/freeglut.h>
    #include <iostream>
    #include <GL/gl.h>
//структура определяет набор координат для
//построения поверхности
    struct Point {
    float x;  float y;
    float z;
    };
```

```
//группа точек, которые определяют поверхность
//(некоторую выпуклость) Point
    Points[4][4] = {
     {
           { 10,0,10 },
           {  5,0,10 },
           { -5,0,10 },
           {-10,0,10 }
     },
     {
           { 10,0,5 },
           {  5,6,5 },
           { -5,6,5 },
           {-10,0,5 }
     },
     {
           { 10,0,-5 },
           {  5,6,-5 },
           { -5,6,-5 },
           {-10,0,-5 }
     },
     {
           { 10,0,-10 },
           {  5,0,-10 },
           { -5,0,-10 },
           {-10,0,-10 }
     }
    };

//класс, в котором производятся расчеты и
//вычисления позиций точек class
    surface
    {
    protected:
//уровень детализации и изменения частоты
```

```
//отрисовки static unsigned
    int size;
//вращение объекта вдоль горизонтальной оси под
//некоторым углом
    static int roundx;
//вращение объекта вдоль вертикальной оси под
//некоторым углом static
    int roundy;
//Так как структурой определено 16 основных
//точек, можно получить 4 кривые
//произвольного поверхностного представления.
//Функция является основной, так
//как выполняет задачу для каждого ряда из
//четырех точек в u-направлении получить
//4 новые точки. Новые точки сформируют кривую в
//v-направлении, что позволит вычислить
//оставшиеся точки static Point
    CalculateU(float t,int row) {
//последняя точка
    Point p;
//инвертирование значения t (замена на обратное)
    float it = 1.0f-t;
//вычисление связующих функций по построению
//поверхности  float b0 =
    t*t*t;  float b1 =
    2*t*t*it;  float b2 =
    2*t*it*it;  float b3 =
    it*it;

//сложение результирующих точек и соответствующим
//им связующих функций
     p.x = b0*Points[row][0].x +
    b1*Points[row][1].x +
    b2*Points[row][2].x +
    b3*Points[row][3].x ;
```

```
     p.y = b0*Points[row][0].y +
    b1*Points[row][1].y +
    b2*Points[row][2].y +
    b3*Points[row][3].y ;

     p.z = b0*Points[row][0].z +
            b1*Points[row][1].z +
    b2*Points[row][2].z +
    b3*Points[row][3].z ;
     return p;
    }
    //CalculateV()
//Для получения 4 точек в U направлении,
//необходимо использовать те точки, которые были
//получены путем расчета кривых в V направлении.
//Функция использует промежуточные точки и
//создает конечные для построения заданной
//поверхности static Point CalculateV(float
    t,Point* pnts)
{
     Point p;
//Изменение значения t на обратное
    float it = 1.0f-t;
//вычисление связующих функций по построению
//поверхности  float b0 =
    t*t*t;  float b1 =
    3*t*t*it;  float b2 =
    3*t*it*it;  float b3 =
    it*it*it;
//сложение результирующих точек и соответствующим
//им связующих функций
     p.x = b0*pnts[0].x +
    b1*pnts[1].x +     b2*pnts[2].x +
       b3*pnts[3].x ;
```

```
     p.y = b0*pnts[0].y +
    b1*pnts[1].y +     b2*pnts[2].y +
       b3*pnts[3].y ;

     p.z = b0*pnts[0].z +
    b1*pnts[1].z +     b2*pnts[2].z +
       b3*pnts[3].z ; return p;
    }
    //Calculate()
//Вычисление точек по u и v параметрам в
//диапазоне от 0,0 до 1,1.
    static Point Calculate(float u,float v) {
//необходимо оценить 4 кривые в u направлении.
//при этом точки будут сохранены во временном
//массиве
     Point temp[4];
//вычисление каждой точки в конечной v кривой
    temp[0] = CalculateU(u,0);  temp[1] =
    CalculateU(u,1);  temp[2] =
    CalculateU(u,2);  temp[3] =
    CalculateU(u,3);
    //после получения 4 точек, можно производить
//вычисления в v направлении.
//таким образом формируются финальные точки
    return CalculateV(v,temp);
    } public:
    surface()
        {
        } };
    unsigned int surface::size=4;
    int surface::roundx=12; int
    surface::roundy=15;
//Класс по заданию начальных параметров сцен и
//вызову расчетных функций class
    base:public surface
    {
    public:
```

```
        static void OnReshape(int w, int h)
         {
//При делении на 0, присвоить следующее значение
    if (h==0)   h=1;
//установить область окна отображения
glViewport(0,0,w,h);
//установить матрицу проекции
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
//используем перспективную проекцию
    gluPerspective(45,(float)w/h,0.1,100);
//возврат к матрице моделирования, чтобы была
//возможность перемещать построенный объект
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
       }
      static void OnDraw() {
//очистка буфера глубины сцены
    glClear(GL_DEPTH_BUFFER_BIT|GL_COLOR_BUFFER_
BIT);
//обнулить предыдущие значения при построении
    glLoadIdentity();
// установить позицию камеры просмотра
    gluLookAt(       roundx,roundy,22,
    //просмотр с позиции смотрящего
                     0,0,0,
    //целевые точки
                     0,1,0);
    //в направлении
    glColor3f(1,1,0);  //Назначить
    размер точек  glPointSize(5);
    glBegin(GL_POINTS);
//использовать параметрическое значение
    for(unsigned int i=0;i!=size;++i) {
//вычисление параметрического u значения
     float u = (float)i/(size-1);
     for(unsigned int j=0;j!=size;++j) {
```

```
// вычисление параметрического u значения
    float v = (float)j/(size-1);
     //расчет точек  на поверхности
                Point p = Calculate(u,v);
                //отрисовка точки
           glVertex3f(p.x,p.y,p.z);
           }
     }
     glEnd();
//так как изображение уже подготовлено и
//находится в скрытой (обратной) части буфера
//необходимо переместить его в видимую часть(в
//фронтальную), чтобы отобразить картинку
    glutSwapBuffers();
    }
    static void OnInit()
    {
     glEnable(GL_DEPTH_TEST);
    }
    static      void OnKeyPress(unsigned  char
key,int,int) {  switch(key) {
//увеличиваем значение по числу объектов в точках
    case '+':
          ++size;
          break;
//уменьшаем значение по числу объектов в точках
    case '-':
          --size;
//но минимальное значение может быть не менее 3.
     if (size<3)
                size=3;
     break;  case 'a':
          roundx++;
          if (roundx>45) roundx=45;
    break;     case 'd':
    roundx--;
```

```
            if (roundx<-45) roundx=-45;;
    break;     case 'w':
    roundy++;
            if (roundy>45) roundy=45;
    break;
        case 'x':
    roundy--;
            if (roundy<-45) roundy=-45;;
    break;

    default:
           break;
     }
//glut запрос по перестроению изображения на
//экране
    glutPostRedisplay();
    }
    static void OnExit()
    {
    } };
    int main(int argc,char** argv)
    {
     base *pointer=new base();
//начальные установки для glut
    glutInit(&argc,argv);
//запрос на использование буфера глубины, RGBA
//режима отображения и двойной буферизации при
//выводе
    glutInitDisplayMode(GLUT_DEPTH|GLUT_RGBA|GLU
T_DOUBLE);
//размер окна вывода
    glutInitWindowSize(800,600);
//создание окна с заголовком
    glutCreateWindow("Surface: +/- size, a/d-
round X,w/x-round Y");
//вызов функции по отображению сцены
    glutDisplayFunc(pointer->OnDraw);
```

```
//функция обработки изменений в заданном окне
    glutReshapeFunc(pointer->OnReshape);
    //функция по обработке событий от клавиатуры
    glutKeyboardFunc(pointer->OnKeyPress);
//вызов метода с пользовательскими установками
     pointer->OnInit();
//производится вызов функции, когда приложение
//завершает свою работу  atexit(pointer-
    >OnExit);
//функция обеспечивает выполнение приложения в
//некотором цикле
    glutMainLoop();  return
    0;
    }
```

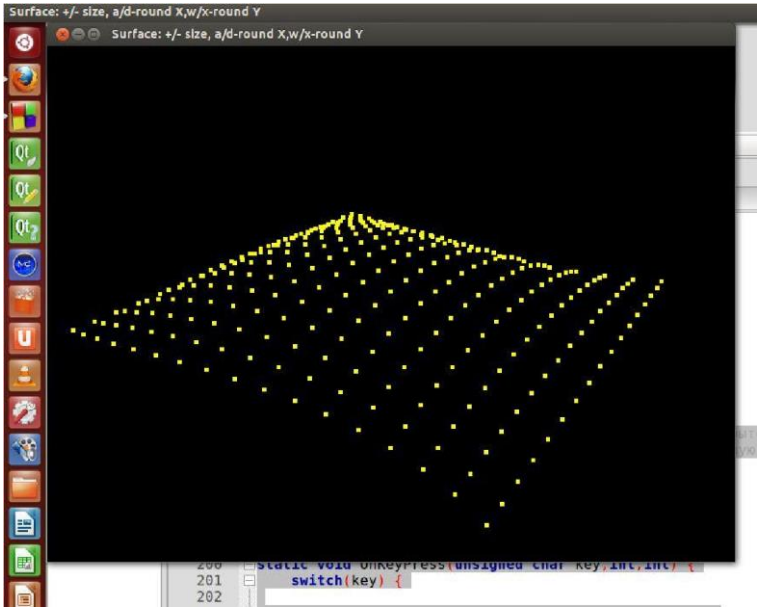


Рис. 2. Моделирование поверхности расчетными точками

На рисунке 3 показан вид поверхности по функции, зависящей от значений двух переменных. Листинг программы, приведенный по тексту, содержит уже известные команды и

позволяет подготовить вывод на экран несколько поверхностей, которые определены пользователем в зависимости от нажатия функциональных клавиш (F1-F5). Поворот по осям, как и в рассматриваемых примерах, выполнен с помощью курсорных клавиш. Степень кривизны поверхности определяется клавишами PgUp-PgDown.

Приложение подготовлено по материалам сайта www.3d how2.org.ua

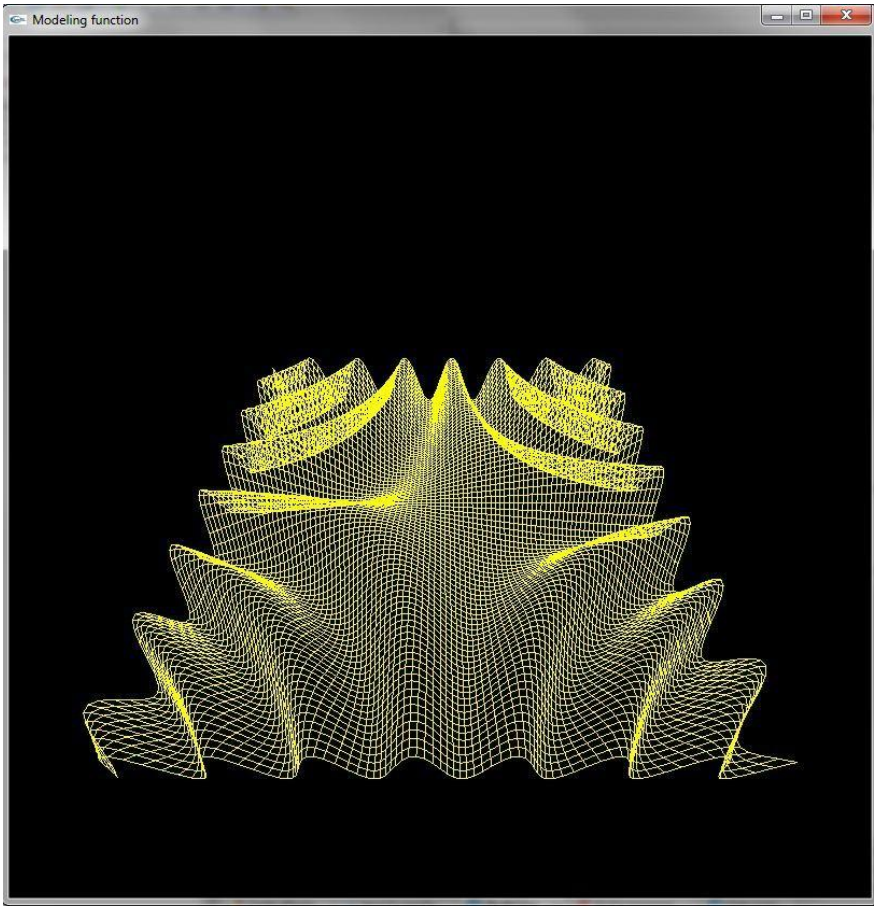


Рис. 3. Моделирование поверхностей с использованием математических функций

```
#include <GL/glut.h>
#include <stdlib.h>
#include <cmath>

class modeling
{
static int rx,ry;
static int curvature;
static int view;
public:
//Задана функция вида static inline float
func(float x, float y)
{
    switch(view)
    {
case 1:
//Некоторая синусоидальная функция двух
//переменных return sin(x * y *
0.0001)*curvature;       break;
case 2:
//Построение функции тангенса, зависящей от x и y
return tan(x * y * 0.0001)*curvature;
break;     case 3:
//Некоторая логарифмическая зависимость
return log(x * y * 0.0001)*curvature;
break;     case 4:
//Гиперболический параболоид return ((x*x/2-
y*y/4)*0.00001)*curvature;       break;
case 5:   return exp(x * y/3 *
0.0001)*curvature;       break;

     default:
 return sin(x * y * 0.0001)*curvature;
break;
```

```
    } }
static void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
glPushMatrix();
glTranslatef(0, 0, -800);
glRotatef(-30, 1, 0, 0);
glColor3d(1,1,0);  gluLookAt(
rx,ry,150,
                      0,0,0,
                 0,1,0);
    for (float x = -480; x < 480; x += 10) {
glBegin(GL_LINE_STRIP);
    for (float y = -480; y < 480; y += 10)
        {
            glVertex3f(x, y, func(x, y));
        }
glEnd();
    }
    for (float y = -480; y < 480; y += 10){
glBegin(GL_LINE_STRIP);
    for (float x = -480; x < 480; x += 10)
        {
            glVertex3f(x, y, func(x, y));
        }
glEnd();
    }
glPopMatrix();
glutSwapBuffers();
    }
static void OnReshape(int w, int h)
    {
         glClearColor(0, 0, 0, 1.0);
     if (h==0)
          h=1;
     glMatrixMode(GL_PROJECTION);
```

```
      glLoadIdentity();

glFrustum(-100, 100, -100, 100, 100, 2000);

      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
   }
static void OnKeyPress(int key,int,int) {
      {
        switch(key)  {    case
GLUT_KEY_LEFT:          rx++;
if      (rx>400)      rx=400;
break;                    case
GLUT_KEY_RIGHT:         rx--;
if     (rx<-400)     rx=-400;
break;     case GLUT_KEY_UP:
ry++;             if  (ry>400)
ry=400;       break;     case
GLUT_KEY_DOWN:          ry--;
if     (ry<-400)     ry=-400;
break;                    case
GLUT_KEY_PAGE_DOWN:
      curvature--;
break;     case
GLUT_KEY_PAGE_UP:
      curvature++;
break;     case
GLUT_KEY_F1:
      view=1;
break;     case
GLUT_KEY_F2:
      view=2;
break;     case
GLUT_KEY_F3:
      view=3;
break;     case
GLUT_KEY_F4:
```

```
      view=4;
break;     case
GLUT_KEY_F5:
      view=5;
      break;
     default:
break;
      }
      }
     glutPostRedisplay();
}
};
//Параметры по умолчанию
int modeling::rx=0; int
modeling::ry=0; int
modeling::curvature=0;
int modeling::view=1;
int main(int argc, char **argv)
{
 modeling *pnt;
pnt=new modeling();
glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
glutInitWindowSize(800, 800);
glutInitWindowPosition(20, 86);
glutCreateWindow("Modeling function");
glutReshapeFunc(pnt->OnReshape);
glutDisplayFunc(pnt->display);
glutSpecialFunc(pnt->OnKeyPress);
glutMainLoop();
}
```

С помощью операций вращения можно смоделировать некоторый контур и получить объект glut. В следующем примере показано, как использовать функцию glRotate(m, bool_x, bool_y, bool_z) для создания грубой

модели поверхности. Параметры указанной функции следующие: mугол поворота, bool_x=1, если требуется вращение вокруг оси x, 0-вращение по оси x не задано, bool_y и bool_z задаются аналогичным образом.

```
      #include <GL/glut.h>
      #include <iostream>
      #include <GL/gl.h>
//класс, в котором производятся расчеты и
//вычисления позиций точек
      class base_geometry
      {
      protected:
         static int rx,ry,size; static
      void build()
      {
      glClear(GL_DEPTH_BUFFER_BIT|GL_COLOR_BUFFER_
BIT);      glLoadIdentity();
      glTranslatef(0, 0, -800);

      gluLookAt( 0,ry,20,
                              0,0,0,
                        0,1,0);
      for (int m=0;m<10*size;m++)
      { glRotatef(m,0,1,0);
      {
      glBegin(GL_LINE_STRIP);
      glColor3d(1,1,1);
      glVertex3d(1*size,1*size,0);
      glVertex3d(2*size,1*size,0);
      glVertex3d(2*size,2*size,0);
      glVertex3d(1*size,2*size,0);
      glVertex3d(1*size,1*size,0); glEnd();
      } }
         glutSwapBuffers();
      } };
```

```
      int base_geometry::rx=0; int
      base_geometry::ry=0; int
      base_geometry::size=90;

//Класс по заданию начальных параметров сцен и
//вызову расчетных функций class
      base:public base_geometry
      {
      public:
      static
      void
      OnReshape
      (int w,
      int h)
          {
//При делении на 0, присвоить следующее значение
      if (h==0)             h=1;
      //установить область окна отображения
           glViewport(0,0,w,h);
      //установить матрицу проекции
           glMatrixMode(GL_PROJECTION);
           glLoadIdentity();
           glFrustum(-100, 100, -100, 100, 100,
2000);
//возврат к матрице моделирования, чтобы была
//возможность перемещать построенный объект
      glMatrixMode(GL_MODELVIEW);
           glLoadIdentity();
         }
      static void OnInit()
      {
           glEnable(GL_DEPTH_TEST);
      }
      static void OnKeyPress(int key,int,int) {
           {
```

```
            switch(key) {
      case GLUT_KEY_UP:
            ry++;
            if (ry>360) ry=360;
      break;     case
      GLUT_KEY_DOWN:
            ry--;
            if (ry<-360) ry=-360;
      break;     case
      GLUT_KEY_HOME:
            size++;
            if (size>500) size=500;
      break;     case GLUT_KEY_END:
            size--;        if
      (size<1) size=1;
      break;     default:
                 break;
           }
           }
           glutPostRedisplay();
      }
      static void OnExit()
      {
      } };
      int main(int argc,char** argv)
      {
           base *pointer=new base();
//начальные установки для glut
      glutInit(&argc,argv);
//запрос на использование буфера грубины, RGBA
//режима отображения и двойной буферизации при
//выводе   glutInitDisplayMode(GLUT_DEPTH|GLUT_RG
BA|GLUT_DOUBLE);
//размер окна вывода
      glutInitWindowSize(800,600);
```

```
//создание окна с заголовком
     glutCreateWindow("Rotation");
//вызов функции по отображению сцены
     glutDisplayFunc(pointer->build);
//функция обработки изменений в заданном окне
     glutReshapeFunc(pointer->OnReshape);
//функция по обработке событий от специальных
//клавиш   glutSpecialFunc(pointer->OnKeyPress);
//вызов метода с пользовательскими установками
     pointer->OnInit();
//производится вызов функции, когда приложение
//завершает свою работу    atexit(pointer-
     >OnExit);
//функция обеспечивает выполнение приложения в
//некотором цикле
          glutMainLoop();
          return 0;
     }
```

Результат работы программы показан на рисунке 4.

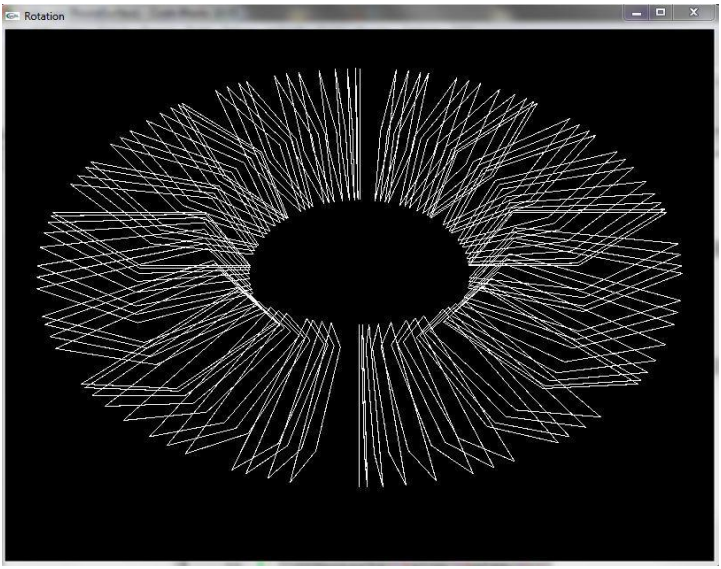


Рис. 4. Вращение контура для построения поверхности

Задания на самостоятельную работу:

1. Для листинга, где производится построение поверхности с помощью опорных точек, изменить приложение так, чтобы поверхность была определена связующими линиями.
2. Добавить функцию, которая бы определяла кривизну поверхности.
3. Для просмотра поверхности с разных ракурсов подключить манипулятор мышь, используя при этом функцию обратного вызова glutPassiveMotionFunc(). Параметры по использованию указанной функции взять из документации.
4. Спроектировать собственную объектноориентированную модель приложения с геометрическим способом задания поверхности объекта.

ЛАБОРАТОРНАЯ РАБОТА №5
ТВЕРДОТЕЛЬНОЕ МОДЕЛИРОВАНИЕ

Цель работы: подготовить объектно-ориентированную модель приложения, построить твердотельную модель согласно заданию.

Задачи и требования к проекту разработки:

1. Изучить приемы построения твердотельных моделей с помощью функций glut и OpenGL.
2. Подготовить объектно-ориентированную модель приложения.
3. Построить программно модель, используя известные приемы проектирования тел в glut.

Теоретические сведения

Как уже отмечалось в предыдущих работах и в предложенных листингах программ, все трехмерные объекты были построены с использованием следующих основных атрибутов-примитивов OpenGL: GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP, GL_NRIANGLES, GL_TRIANGLE_STRIP, GL_QUAD_STRIPS и т.д. При увеличении размера или в случае сложной геометрической структуры управление всеми отдельными элементами (гранями и ребрами) тел может быть непростым, следовательно, применяют иные подходы к проектированию, например собирают объекты из более примитивных или простых [1]. Разделяя объект на меньшие элементы, можно облегчить тем самым процесс кодирования и отображения объектов. Кроме всего, базовые блоки могут быть использованы многократно в разрабатываемом приложении. В более сложных приложениях моделирования возможность повторного использования крайне важна. Есть множество элементов, которые различаются лишь габаритами, а контурная геометрия в пределах серии полностью идентична. Типичными представителями могут выступать элементы приспособлений, определенные в рамках государственного стандарта (ГОСТа), например: втулки, кондукторные плиты, тройники, фланцы и т.д. В следующем примере показаны приемы моделирования тел посредством описания всей геометрии для куба и цилиндра. Листинг приложения содержит подробные комментарии по выполняемым функциям. Кроме всего, в примере предложен способ по заданию материала объектам и системе освещения объектов, механизм которого учащимся предлагается изучить самостоятельно.

```
    #include <GL/glut.h>
    #include <stdio.h>
    #include <math.h>

    static unsigned int g_dwDisplayListNum;
    static float           g_fSpinX   = 0.0f,
                           g_fSpinZ   = 0.0f;
    static int             g_iMousePosX = 0;
    static int             g_iMousePosY = 0;
    static char            g_bWireframe = 0;

    //Класс по созданию твердотельных объектов
    class Object3D
    {
    protected:

    //Метод по созданию куба
    //Первый параметр-продольное разбиение куба
    //Второй параметр-поперечное разбиение куба
    //Третий параметр-разбиение боковой грани
    //Четвертый-размер объекта
    static unsigned int generateCube(unsigned int
iWsections=7,
                                     unsigned
int iHsections=5,
                int                  unsigned
iDsections=3,
                                     float
fDimension=3 ) {
     unsigned int iDisplayListNum;
    float fHalfDimension;  float
    fWInc,fHInc,fDInc;  unsigned
    int   i,j;

    //задать параметры абсолютной величиной
```

```
//(сделать положительными) во избежание ошибок
    fDimension      =      fabs(fDimension);
    //вычисление ряда необходимых значений
    fHalfDimension = 0.5f*fDimension;  fHInc
    = (float)fDimension/iHsections;  fWInc  =
    (float)fDimension/iWsections;   fDInc   =
    (float)fDimension/iDsections;
    //определить      количество      делений
    if(iWsections < 1)
     {
           iWsections = 1;
     }
     if(iHsections < 1)
     {
           iHsections = 1;
     }
     if(iDsections < 1)
     {
           iDsections = 1;
     }
//создать список значений и начать запись в
//список ряда полученных из них
    iDisplayListNum = glGenLists(1);
    glNewList(iDisplayListNum,GL_COMPILE);

    //верхняя и нижняя грань куба
           for(i=0;i<iWsections;i++)
           {
                glBegin(GL_QUAD_STRIP);
     glNormal3f(0.0f,1.0f,0.0f);
                     for(j=0;j<=iDsections;j++)
                     {
                          glVertex3f(
(i+1)*fWInc - fHalfDimension, fHalfDimension,
j*fDInc - fHalfDimension );
                          glVertex3f(
```

```
i*fWInc - fHalfDimension, fHalfDimension, j*fDInc
- fHalfDimension );
                        }
                  glEnd();
                  glBegin(GL_QUAD_STRIP);
                        glNormal3f(0.0f,-
1.0f,0.0f);
                        for(j=0;j<=iDsections;j++)
                        {
                              glVertex3f( i*fWInc -
fHalfDimension, -fHalfDimension, j*fDInc -
fHalfDimension);
                              glVertex3f(
(i+1)*fWInc - fHalfDimension, -fHalfDimension,
j*fDInc - fHalfDimension );
                        }
                  glEnd();
            }
      //левая и правая грани куба
            for(i=0;i<iHsections;i++)
            {
                  glBegin(GL_QUAD_STRIP);
      glNormal3f(1.0f,0.0f,0.0f);
                        for(j=0;j<=iDsections;j++)
                        {
                              glVertex3f(
fHalfDimension,     i*fHInc - fHalfDimension,
j*fDInc - fHalfDimension );
                              glVertex3f(
fHalfDimension, (i+1)*fHInc - fHalfDimension,
j*fDInc - fHalfDimension );
                        }
                  glEnd();
                  glBegin(GL_QUAD_STRIP);
```

```
                    glNormal3f(-
1.0f,0.0f,0.0f);
                    for(j=0;j<=iDsections;j++)
                    {
                         glVertex3f(          -
fHalfDimension, (i+1)*fHInc - fHalfDimension,
j*fDInc - fHalfDimension );
                         glVertex3f(          -
fHalfDimension,     i*fHInc - fHalfDimension,
j*fDInc - fHalfDimension );
                    }
               glEnd();
          }
    //ближайшая и удаленная грани кубы
          for(i=0;i<iWsections;i++)
          {
               glBegin(GL_QUAD_STRIP);
                  glNormal3f(0.0f,0.0f,1.0f);

                    for(j=0;j<=iHsections;j++)
                    {
                         glVertex3f(
i*fWInc  - fHalfDimension, j*fHInc -
fHalfDimension, fHalfDimension );
                         glVertex3f(
(i+1)*fWInc  - fHalfDimension, j*fHInc -
fHalfDimension, fHalfDimension );
                    }
               glEnd();
               glBegin(GL_QUAD_STRIP);
                    glNormal3f(0.0f,0.0f,-
1.0f);

                    for(j=0;j<=iHsections;j++)
```

```
                    {
                            glVertex3f(
(i+1)*fWInc - fHalfDimension, j*fHInc -
fHalfDimension, -fHalfDimension );
                            glVertex3f(
i*fWInc     -     fHalfDimension,     j*fHInc      -
fHalfDimension, -fHalfDimension );
                    }

                glEnd();
            }
     glEndList();
     return iDisplayListNum;
    }
    //Метод построения цилиндра
    //параметр 1-количество поперечных разбиений
    //параметр 2-количество продольных разбиений
    //параметр 3-высота цилиндра //параметр
    4-радиус цилиндра
    static unsigned int generateCylinder(unsigned
                                          float
fHeight=3   ,
                                          float
int iVsections=15,

     unsigned int iHsections=15,
fRadius=2   ) {
     unsigned int iDisplayListNum;
    float fHalfHeight;  float
    fHeightInc;  float fAngle;
    unsigned int   i,j;
    //установить абсолютные значения для всех
//параметров во избежание ошибок
    fHeight = fabs(fHeight);  fRadius
    = fabs(fRadius);
```

```
    //вычисление ряда полезных значений
    fHalfHeight = 0.5f*fHeight;
     fHeightInc  = (float)fHeight/iHsections;
    //Задание числа вертикальных секций
    if(iVsections < 5)
     {
          iVsections = 5;
     }
    //создать список значений и начать запись в
список ряда полученных из них  iDisplayListNum =
glGenLists(1);
glNewList(iDisplayListNum,GL_COMPILE);
    //получение верхнего основания цилиндра
вращением треугольников    glBegin(
GL_TRIANGLE_FAN );
     glNormal3f(0.0f,1.0f,0.0f);

     glVertex3f(0.0f,fHalfHeight,0.0f);
          for(i=0;i<=iVsections;i++)
               {
                    fAngle     =      (float)i*(-
M_PI*2)/iVsections;

      glVertex3f(fRadius*cos(fAngle),fHalfHeight,f
Radius*sin(fAngle));
               }
          glEnd();
    //получение нижнего основания цилиндра
//вращением треугольников       glBegin(
    GL_TRIANGLE_FAN );
     glNormal3f(0.0f,-1.0f,0.0f);
     glVertex3f(0.0f,-
fHalfHeight,0.0f);
               for(i=0;i<=iVsections;i++)
               {
                    fAngle                    =
```

```
(float)i*(M_PI*2)/iVsections;


     glVertex3f(fRadius*cos(fAngle),fHalfHeight,f
Radius*sin(fAngle));
                }
           glEnd();
    //отрисовать число секций, которые были
//получены как сетчатые плитки
     for(i=0;i<iHsections;i++)
          {
               float fCurrMin = -fHalfHeight +
fHeightInc*i,
                       fCurrMax = fCurrMin +
fHeightInc;
               glBegin( GL_QUAD_STRIP);
                    for(j=0;j<=iVsections;j++)
                    {
                         float ca,sa,sc;
                         fAngle                =
(float)j*(M_PI*2)/iVsections;
                         ca                    =
fRadius*cos(fAngle);
                         sa                    =
fRadius*sin(fAngle);
                           sc = 1.0f/(sqrt(ca*ca
+sa*sa));

     glNormal3f(ca,0.0f,sa);

     glVertex3f(ca,fCurrMin,sa);

     glVertex3f(ca,fCurrMax,sa);
                    }
```

```
                glEnd();
            }
     glEndList();
     return iDisplayListNum;
    }
    };
    //В классе реализованы глобальные настройки
//под GLUT приложение class
    base_GLUT:public Object3D
    { private:
    static void init( void )
    {
     glClearColor(0.0f,0.0f,0.0f,0.0f);
     glShadeModel(GL_SMOOTH);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);
    glCullFace(GL_BACK);
    //Настройки освещения
     {
          float       fpAmbient[]              =
{0.15f,0.15f,0.15f,0.0f};
          float       fpDiffuse[]              =
{0.8f,0.8f,0.8f,0.0f};
          float          fpSpecular[]          =
{0.8f,0.8f,0.8f,0.0f};
          float          fpPosition[]          =
{1.0f,16.0f,10.0f,1.0f};


     glLightfv(GL_LIGHT0,GL_AMBIENT,fpAmbient);

     glLightfv(GL_LIGHT0,GL_DIFFUSE,fpDiffuse);

     glLightfv(GL_LIGHT0,GL_SPECULAR,fpSpecular);

     glLightfv(GL_LIGHT0,GL_POSITION,fpPosition);
```

```
          glEnable(GL_LIGHTING);
      glEnable(GL_LIGHT0);
      }

    //Настройки свойств материала
      {
           float       fpAmbient[]               =
{0.3f,0.3f,0.3f,0.0f};
           float       fpDiffuse[]               =
{0.7f,0.7f,0.7f,0.0f};
           float          fpSpecular[]           =
{1.0f,1.0f,1.0f,0.0f};
      glMaterialfv(GL_FRONT_AND_BACK,GL_AMBIENT,fp
Ambient);

      glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,fp
Diffuse);

      glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,f
pSpecular);
           glMaterialf
(GL_FRONT_AND_BACK,GL_SHININESS,128);
      }
    //Вызов функции по созданию цилиндра
    g_dwDisplayListNum = generateCube(5,5,5,5);
    }
    static void display( void )
    {
      glClear(        GL_DEPTH_BUFFER_BIT        |
GL_COLOR_BUFFER_BIT );
    glLoadIdentity();  gluLookAt
    ( 0.0f,0.0f,10.0f,
                     0.0f,0.0f,0.0f,
      0.0f,1.0f,0.0f );
```

```
     glRotatef( g_fSpinZ, 1,0,0);
    glRotatef( -g_fSpinX,0,0,1);
    //установить режим вывода-проволочный или
//твердотельный
     glPolygonMode(             GL_FRONT_AND_BACK,
(g_bWireframe) ? GL_LINE : GL_FILL );
     (g_bWireframe)? glDisable(GL_CULL_FACE):
glEnable(GL_CULL_FACE);
     (g_bWireframe)? glDisable(GL_LIGHTING) :
glEnable(GL_LIGHTING);
    //вычертить геометрическую модель
    glPushMatrix();
        glCallList( g_dwDisplayListNum );
    glPopMatrix();  glutSwapBuffers();
    }
    static void reshape( int w, int h )
    {
     glViewport(0,0,w,h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
     gluPerspective(52.0f,(float)w/h,1,100);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    }
    static void mouseClick( int button, int
state, int posX, int posY )
    {
    //захват позиции курсора при нажатой клавише
//мыши для последующих действий
    if( state == GLUT_DOWN )
     {     g_iMousePosX = posX;
     g_iMousePosY = posY;
     }
     glutPostRedisplay();
    }
    static void mouseMotion( int x,int y)
```

```
    {
    //изменение значения угла вращения, если мышь
//была перемещена  g_fSpinX += (float)(x-
    g_iMousePosX)*0.1f;  g_fSpinZ +=
    (float)(y-g_iMousePosY)*0.1f;
    g_iMousePosX = x;  g_iMousePosY = y;
    glutPostRedisplay();
    }
    //Реакция на нажатие клавиш
    static void keyboard(unsigned char key, int
posX, int posY )
    {
     switch( key )
     {
    //'Проволочная' геометрия объектов case
     'w' : case 'W' :
           g_bWireframe = !g_bWireframe;
     break;
    //Вызов метода построения куба  case
    'C' : case 'c' :
    glDeleteLists(g_dwDisplayListNum,1);
           generateCube();
           break;
    //Вызов метода проектирования цилиндра
    case 'V' : case 'v' :
    glDeleteLists(g_dwDisplayListNum,1);
     generateCylinder();        break;

    default:
           break;
     }
     glutPostRedisplay();
    } public:
    //Конструктор класса. Вызываются методы
//обработки
```

```
    //событий от мыши, клавиатуры, изменения
//размеров окна и т.д.
     base_GLUT()
     {
          glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMouseFunc(mouseClick);
    glutMotionFunc(mouseMotion);
    glutKeyboardFunc(keyboard);
    init();
     } };
    int main( int argc, char** argv )
    {
          glutInit(&argc, argv);
          glutInitDisplayMode   ( GLUT_DOUBLE |
    GLUT_RGB    |
                                    GLUT_DEPTH  );
    glutInitWindowSize    ( 800 , 600 );
    glutInitWindowPosition( 100 , 100 );
    glutCreateWindow( "Press 'c'-cube; 'v'-
cilinder; 'w'-GridMode" );      base_GLUT
    *pointer=new base_GLUT();
    glutMainLoop();   delete pointer;
    }
```

Результат работы приложения в “проволочных” моделях представлен на рисунках 5 и 6.

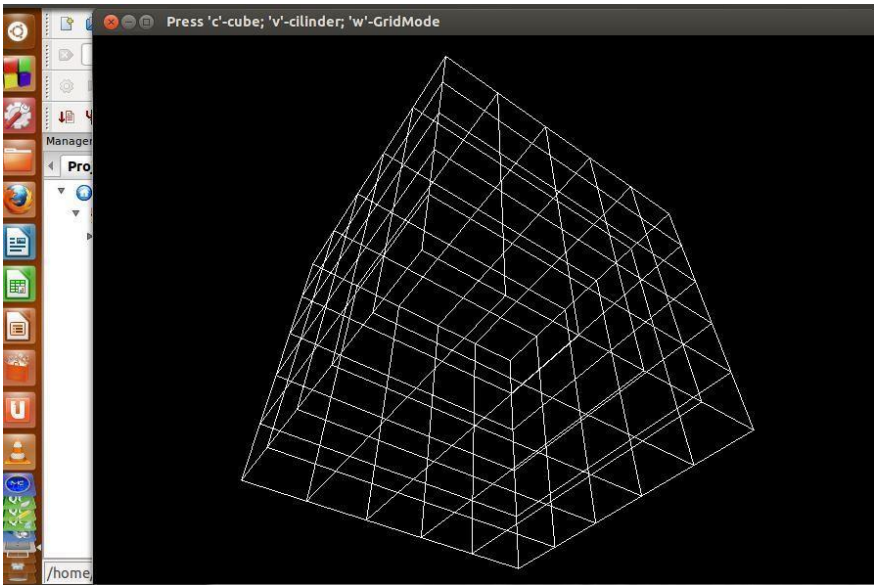


Рис. 5. Построение параметрической модели куба

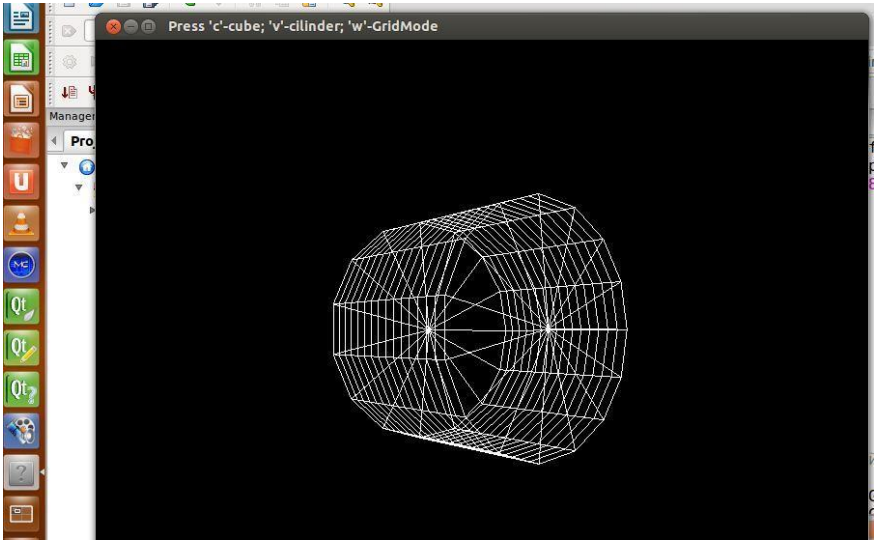


Рис. 6. Построение параметрической модели цилиндра

Задания на самостоятельную работу:

1. Подготовить на основе приведенного листинга программы проект построения твердотельных моделей, представленных на рисунке 7.
2. Добавить в проект второй источник освещения и возможность изменения свойств материала.

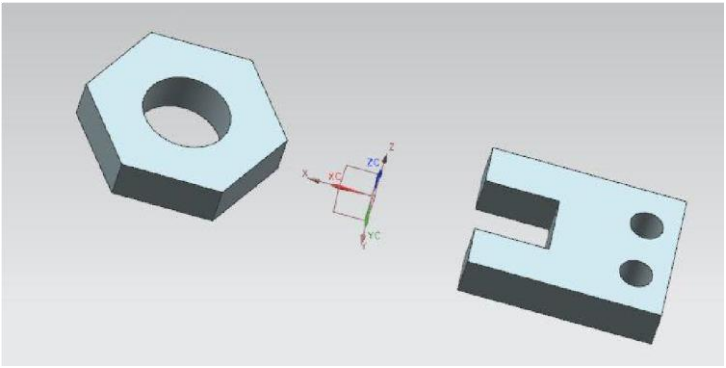


Рис. 7. Элементы проектирования

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Райт-мл Р.С. OpenGL. Суперкнига, 3-е изд. / Р.С. Райтмл, Б.Липчак -М.: ООО ”И.Д. Вильямс”, 2006. – 1040 с.
2. Роджерс Д. Математические основы машинной графики/ Д. Роджерс, Дж. Адамс- М.: Мир, 2001. – 604 с.
3. Баяковский Ю.М. Графическая библиотека OpenGL. Учебно-методическое пособие / Ю.М. Баяковский, А.В. Игнатенко, А.И. Фролов –М.: Изд. отдел факультета Вычислительной Математики и Кибернетики МГУ им. Ломоносова, 2003.-132 с.
4. Довбуш Г.Ф. Visual C++ на примерах / Г.Ф. Довбуш, А.Д. Хомоненко. – СПб.: БХВ-Петербург, 2007. – 528 с.

5. Мюссер Д.Р. C++ и STL: справочное руководство / Д.Р. Мюссер, Ж.Дж. Дердж, А. Сейни. 2-е изд. - М.: ООО ”И.Д. Вильямс”, 2010. – 430 с.

6. Прата С. Язык программирования C++. Лекции и упражнения / С. Прата. 5-е изд. – М.: ООО ”И.Д. Вильямс”, 2007. – 1184 с.

7. Страуструп Б Язык программирования C++ /Б.
Страуструп. - М.: Бином, 2011. – 1136 с.

8. Шилдт Г. C++ Базовый курс / Г. Шилдт. 3-е изд. – М.: ООО ”И.Д. Вильямс”, 2010. – 624 с.

9. Коплиен Дж. Программирование на C++ / Дж. Коплиен. – СПб.: Питер, 2005. – 480 с.

10. Roberge J. A laboratory course in C++ structures. 2ed./ J. Roberge, S. Brandl, D. Whittington. Jones and Bartlett, 2003. -411 p.

11. London J. Modeling Derivatives in C++ / London J. Wiley, 2005. -841p.

СОДЕРЖАНИЕ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам № 4-5 по дисциплине «Программирование трехмерной графики» для студентов направления 09.03.02 «Информационные системы и технологии» всех форм обучения

Составители:
Юров Алексей Николаевич
Паринов Максим Викторович
Рыжков Владимир Анатольевич
Левченко Александр Сергеевич

В авторской редакции

Компьютерный набор А.Н. Юрова

Подписано к изданию 20.01.2013.
Уч.-изд. л. 2,6. «С»

ФГБОУ ВПО «Воронежский государственный технический университет»
394026 Воронеж, Московский просп., 14