Московский авиационный институт
(Национальный исследовательский университет)
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Лабораторная работа № 2

по курсу «Компьютерная графика»

Тема: Каркасная визуализация выпуклого многогранника

Студент: Сахарин Н. А.
Группа: 308
Преподаватель: Чернышов Л.Н.

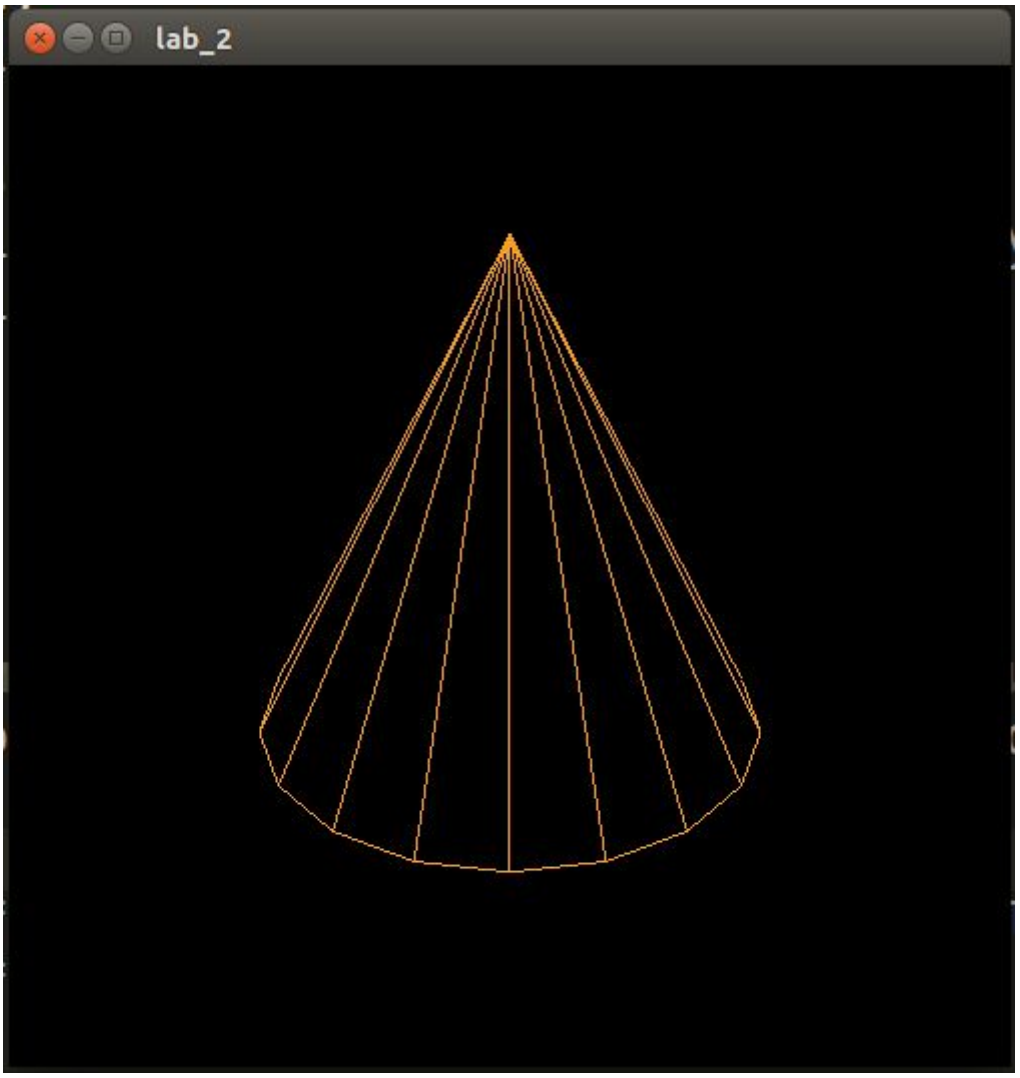
Москва, 2017

1. Постановка задачи

Разработать формат представления 16-гранной правильной пирамиды и процедуру её каркасной отрисовки в ортографической и изометрической проекциях. Обеспечить удаление невидимых линий и возможность пространственных поворотов и масштабирования многогранника. Обеспечить автоматическое центрирование и изменение размеров изображения при изменении размеров окна.

2. Решение задачи

Данная лабораторная работа выполнена на языке программирования C++ с использованием библиотеки OpenGL.



Оцентровка окна и задание осей координат производились встроенными в

библиотеку функциями. Построение 3D модели многогранника состояло из отрисовки граней по вычисленным координатам вершин. Удаление невидимых граней производилось алгоритмом Робертса.
Поворот многогранника осуществлялся посредством умножения матрицы проекции модели на матрицу поворота вдоль оси X или Y. Также была необходимость пересчитать координаты точки из которой шло наблюдение за объектом, поскольку её координаты двигались вместе с графиком и фактически точка обзора стояла на месте относительно фигуры.
Масштабирование заключалось лишь в умножении координат вершин на константу.

3. Листинг программы

```
/*
Сахарин Никита 8О-308Б
Каркасная визуализация выпуклого многогранника.
(16-гранной правильной пирамиды)
Удаление невидимых линий.
*/

#include <GL/glut.h>
#include <math.h>
#include <stdbool.h>

typedef struct {
    float x;
    float y;
    float z;
} vector;

vector v[17];
float xRot = 0, yRot = 0;
float scale = 1;
float GL_PI = 3.1415926535897932;
vector obs_pnt, aux_pnt;

void KeyBoard ( unsigned char key, int x, int y ) {
    /* Изменение масштаба рисунка */
```

```
    if ( key == '=' ) {
        if ( scale < 1.7 )
            scale += 0.1;
    }
    else if ( key == '-' ) {
        if ( scale > 0.3 )
            scale -= 0.1;
    }

    glutPostRedisplay (); // Обновляем окно
}

void SKeyBoard ( int key, int x, int y ) {
    /* Поворот рисунка при нажатии на стрелочки */

    if ( key == GLUT_KEY_UP )
        xRot += 2.0f * GL_PI / 180;

    if ( key == GLUT_KEY_DOWN )
        xRot += -2.0f * GL_PI / 180;

    if ( key == GLUT_KEY_LEFT )
        yRot += 2.0f * GL_PI / 180;

    if ( key == GLUT_KEY_RIGHT )
        yRot += -2.0f * GL_PI / 180;

    glutPostRedisplay(); // Обновляем окно

}

void Calculate_vertices () {
    /* Вычисляем координаты вершин основания пирамиды */

    float R = 50.0; // Задаем радиус основания

      float alpha = ( 360.0 / 16.0 ) * GL_PI / 180.0; //
Центральный угол основания пирамиды (в радианах)

    /* Зададим первую вершину произвольно */
    v[0].x = 0;
```

```
    v[0].y = -40;
    v[0].z = R;

    float angle = alpha;

    for (int i = 1; i < 16; angle += alpha, ++i ) {
        v[i].x = sin(angle) * R;
        v[i].y = -40;
        v[i].z = cos(angle) * R;
    }

    /* Задаем координаты вершины пирамиды */
    v[16].x = 0;
    v[16].y = 80;
    v[16].z = 0;

}

void Initialize () {
    /* Инициализация фона.
       Расчет координат вершин пирамиды. */

    /* Устанавливаем цвет фона и цвет граней пирамиды */
    glClearColor ( 0.0, 0.0, 0.0, 1 );
    glColor3f ( 0.98, 0.625, 0.12 );

     Calculate_vertices (); // Подсчитываем координаты точек в
заданной СК

    /* Задаем начальные значения точки наблюдения */
    obs_pnt.x = 0.0;
    obs_pnt.y = 0.0;
    obs_pnt.z = 200.0;

      /* Задаем координаты вспомогательной точки, находящейся
внутри фигуры */
    aux_pnt.x = 0.0;
    aux_pnt.y = 50.0;
    aux_pnt.z = 0.0;
}

void ChangeSize ( int w, int h ) {
```

```
    /* Меняет наблюдаемый объем и поле просмотра */

    GLfloat nRange = 100.0f;

    /* Предотвращает деление на нуль */
    if ( h == 0 )
        h = 1;

     glViewport ( 0, 0, w, h ); // Устанавливает поле просмотра
с размерами окна

    /* Обновляет стек матрицы проектирования */
    glMatrixMode ( GL_PROJECTION );
    glLoadIdentity ();

    /* Устанавливается объем отсечения с помощью отсекающих
       плоскостей (left, right, bottom, top, near, far) */
    if ( w <= h )
          glOrtho ( -nRange, nRange, -nRange*h / w, nRange*h /
w, -nRange, nRange );
    else
            glOrtho ( -nRange*w / h, nRange*w / h, -nRange,
nRange, -nRange, nRange );

    /* Обновляется стек матрицы проекции модели */
    glMatrixMode ( GL_MODELVIEW );
    glLoadIdentity ();
}

void RotateX () {
    /* Поворот осей координат вокруг оси X */

    float m[16];
       glGetFloatv ( GL_MODELVIEW_MATRIX, m ); // Считываем
матрицу проекции модели

      /* В цикле рассчитываем новые координаты матрицы после
поворота */
    for ( int i = 0; i < 16; i += 4 ) {
        float x, y, z;
        x = m[i];
        y = m[i + 1];
```

```
        z = m[i + 2];
        m[i] = x;
        m[i + 1] = y *cos ( xRot ) - z * sin ( xRot );
        m[i + 2] = y* sin ( xRot ) + z * cos ( xRot );
    }

    /* Обновляется стек матрицы проекции модели */
    glMatrixMode ( GL_MODELVIEW );
    glLoadMatrixf ( m );
}

void RotateY () {
    /* Поворот осей координат вокруг оси Y */

    float m[16];
  glGetFloatv ( GL_MODELVIEW_MATRIX, m ); // Считываем матрицу
проекции модели

     /* В цикле рассчитываем новые координаты матрицы после
поворота */
    for ( int i = 0; i < 16; i += 4 ) {
        float x, y, z;
        x = m[i];
        y = m[i + 1];
        z = m[i + 2];
        m[i] = x * cos ( yRot ) + z * sin ( yRot );
        m[i + 1] = y;
        m[i + 2] = (-1) * x * sin ( yRot ) + z * cos ( yRot );
    }

    /* Обновляется стек матрицы проекции модели */
    glMatrixMode ( GL_MODELVIEW );
    glLoadMatrixf ( m );
}

void Obs_pnt_reculc () {
    /* Перерассчитывает координаты точки наблюдения за моделью
*/

    float m [16];
    glGetFloatv ( GL_MODELVIEW_MATRIX, m );
    obs_pnt.x = 200 * m[2];
```

```
    obs_pnt.y = 200 * m[6];
    obs_pnt.z = 200 * m[10];
}

void Display () {
    /* Основная функция отрисовки */

    glClear ( GL_COLOR_BUFFER_BIT ); // Окно очищается текущим
цветом очистки

    /* Восстанавливаем изначальное состояние матрицы модели,
       после чего делаем соответствующий поворот.         */
    float m[] = {
        1, 0, 0, 0,
        0, 1, 0, 0,
        0, 0, 1, 0,
        0, 0, 0, 1
    };

    glMatrixMode ( GL_MODELVIEW );
    glLoadMatrixf ( m );

    RotateX ();
    RotateY ();

    Obs_pnt_reculc ();

    /* Рисуем боковые грани */
    glBegin ( GL_LINES );

    for ( int i = 0; i < 16; ++i ) {
        /* Составляем два вектора на плоскости */
        vector v1, v2;
        v1.x = v[16].x - v[i].x;
        v1.y = v[16].y - v[i].y;
        v1.z = v[16].z - v[i].z;
        v2.x = v[( i + 1 ) % 16].x - v[i].x;
        v2.y = v[( i + 1 ) % 16].y - v[i].y;
        v2.z = v[( i + 1 ) % 16].z - v[i].z;

        /* Вычисляем вектор нормали к плоскости */
        vector n;
```

```
        n.x = v1.y*v2.z - v1.z*v2.y;
        n.y = v1.z*v2.x - v1.x*v2.z;
        n.z = v1.x*v2.y - v1.y*v2.x;

        /* Нормируем вектор нормали */
        float normal = sqrt ( n.x*n.x + n.y*n.y + n.z*n.z );
        n.x = n.x / normal;
        n.y = n.y / normal;
        n.z = n.z / normal;

        /* Проверка направления нормали */
        float d = -n.x*v[i].x - n.y*v[i].y - n.z*v[i].z;

         bool wrong_direction = ( aux_pnt.x*n.x + aux_pnt.y*n.y
+ aux_pnt.z*n.z + d ) > 0;
        if ( wrong_direction ) {
            n.x *= -1;
            n.y *= -1;
            n.z *= -1;
        }

         /* Проверяем видима ли данная грань и если так, рисуем
её */
            bool visible = ( obs_pnt.x*n.x + obs_pnt.y*n.y +
obs_pnt.z*n.z ) > 0;
        if ( visible ) {
                glVertex3f ( v[16].x * scale, v[16].y * scale,
v[16].z * scale );
                 glVertex3f ( v[i].x * scale, v[i].y * scale,
v[i].z * scale );
                glVertex3f ( v[16].x * scale, v[16].y * scale,
v[16].z * scale );
              glVertex3f ( v[( i + 1 ) % 16].x * scale, v[( i +
1 ) % 16].y * scale, v[( i + 1 ) % 16].z * scale );
                 glVertex3f ( v[i].x * scale, v[i].y * scale,
v[i].z * scale );
              glVertex3f ( v[( i + 1 ) % 16].x * scale, v[( i +
1 ) % 16].y * scale, v[( i + 1 ) % 16].z * scale );
        }
    }

    glEnd ();
```

```
    /* Проверяем видимо ли основание пирамиды */
    glBegin ( GL_LINES );

    vector v1, v2;
    v1.x = v[2].x - v[1].x;
    v1.y = v[2].y - v[1].y;
    v1.z = v[2].z - v[1].z;
    v2.x = v[0].x - v[1].x;
    v2.y = v[0].y - v[1].y;
    v2.z = v[0].z - v[1].z;

    /* Составляем вектор нормали */
    vector n;
    n.x = v1.y*v2.z - v2.y*v1.z;
    n.y = v1.z*v2.x - v1.x*v2.z;
    n.z = v1.x*v2.y - v1.y*v2.x;

    /* Нормируем вектор нормали */
    float normal = sqrt ( n.x*n.x + n.y*n.y + n.z*n.z );
    n.x = n.x / normal;
    n.y = n.y / normal;
    n.z = n.z / normal;

    /* Проверка направления нормали */
     bool wrong_direction = ( aux_pnt.x*n.x + aux_pnt.y*n.y +
aux_pnt.z*n.z ) > 0;
    if ( wrong_direction ) {
        n.x *= -1;
        n.y *= -1;
        n.z *= -1;
    }

     /* Проверяем видима ли данная грань и если так, рисуем её
*/
        bool  visible  =  ( obs_pnt.x*n.x  +  obs_pnt.y*n.y  +
obs_pnt.z*n.z ) > 0;
    if ( visible ) {
        for ( int i = 0; i < 16; ++i ) {
                glVertex3f ( v[i].x * scale, v[i].y * scale,
v[i].z * scale );
             glVertex3f ( v[( i + 1 ) % 16].x * scale, v[( i +
```

```
1 ) % 16].y * scale, v[( i + 1 ) % 16].z * scale );
        }
    }

    glEnd ();

    glFlush (); // Очищаем стек команд рисования
}

int main ( int argc, char** argv ) {
    /* Инициалищация */

    glutInit ( &argc, argv );
      glutInitDisplayMode ( GLUT_RGB | GLUT_SINGLE ); // Мод
рисования
     glutInitWindowSize ( 500, 500 ); // Инициализация размеров
окна
    glutInitWindowPosition ( 500, 200 ); // Позиция окна
    glutCreateWindow ( "lab_2" ); // Создаем окно

    glutKeyboardFunc ( KeyBoard );
    glutSpecialFunc ( SKeyBoard );
    Initialize ();
    glutDisplayFunc ( Display );
     glutReshapeFunc ( ChangeSize ); // Настройка при изменении
размеров окна
    glutMainLoop (); // Основной цикл обработки GLUT
    return 0;
}
```