Лабораторная работа № 8

Тема: Моделирование 3D анимации.

Цель: Научиться анимировать 3D объекты, работать с клавиатурой и мышью средствами OpenGL.

Краткая теория

Работа с клавиатурой

GLUT позволяет создавать приложения, которые работают с клавиатурой, обрабатывают нажатия клавиш. Требуется зарегистрировать в GLUT функции, которые будут выполнять необходимую обработку нажатий клавиш. GLUT предоставляет две функции для регистрации обратного вызова для событий клавиатуры. glutKeyboardFunc, используется, чтобы сообщить системе, что мы хотим обработать «нормальные» нажатия клавиш. Под «нормальными» нажатиями клавиш, имеются в виду буквы, цифры, все, что имеет ASCII код. Синтаксис этой функции заключается в следующем:

```
void glutKeyboardFunc(void (*func) (unsigned char key, int x, int y));
```

Параметры:

*func - имя функции, которая будет обрабатывать «нормальные» события клавиатуры. Передача NULL в качестве аргумента заставляет GLUT игнорировать «нормальные» события (или нажатия).

Функция, используемая в качестве аргумента glutKeyboardFunc должна иметь три аргумента. В первом содержится ASCII код нажатой клавиши, оставшиеся два аргумента обеспечивают положение курсора мыши при нажатии клавиши, относительно верхнего левого угла клиентской области окна. Возможная реализация этой функции - обеспечить выход из приложения, когда пользователь нажимает клавишу Esc. Обратите внимание, что функция glutMainLoop - бесконечный цикл, то есть он никогда не заканчивается. Единственный выход из этого цикла заключается в вызове функции выхода из системы. Так что это именно то, что наша функция будет делать, ползователь захочет выйти из исполняемого приложения в окне (используем stdlib.h в исходном коде). Код функции:

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

Обратите внимание, что используется точно такая же переменная, указанная в синтаксисе glutKeyboardFunc. Если не сделать это, получится ошибка при компиляции! GLUT предоставляет функцию glutSpecialFunc, так что можно зарегистрировать функцию для специальных нажатий клавиш. Синтаксис этой функции заключается в следующем:

```
void glutSpecialFunc(void (*func) (int key, int x, int y));
```

Параметры:

*func - имя функции, которая будет обрабатывать специальные нажатия клавиатуры. Передача NULL в качестве аргумента заставляет GLUT игнорировать специальные клавиши.

GLUT_KEY_* - это набор предопределенных констант в glut.h. Пример набора констант библиотеки:

GLUT_KEY_F1 - функциональная клавиша F1
GLUT_KEY_LEFT - функциональная клавиша стрелка влево
GLUT_KEY_RIGHT - функциональная клавиша стрелка вправо
GLUT_KEY_UP - функциональная клавиша стрелка вверх
GLUT_KEY_DOWN - функциональная клавиша стрелка вниз
GLUT_KEY_PAGE_UP - Page Up функциональная клавиша
GLUT_KEY_PAGE_DOWN - Page Down функциональная клавиша

Регистрирует функции обработки «нормальных» и «специальных» нажатий клавиатуры: glutKeyboardFunc GLUT и glutSpecialFunc. Вызов этих функции может быть выполнен в любом месте, что означает, что можно изменить функции обработки клавиатуры для обработки событий в любое время.

Иногда требуется знать, какая клавиша модификатор, т.е. CTRL, ALT или SHIFT нажата. GLUT предоставляет функцию, которая обнаруживает нажатие данных клавиш. Эта функция должна вызываться только внутри обработки процессов клавиатуры или мыши. Синтаксис для этой функции:

int glutGetModifiers(void);

Возвращаемое значение этой функции является либо одним из трех предопределенных констант (в glut.h) или любая их комбинация. Константы:

GLUT_ACTIVE_SHIFT - Нажаты клавиши Shift или Caps Lock. Если они зажаты вместе, то константа не устанавливается.

GLUT_ACTIVE_CTRL - Нажата клавиша CTRL.

GLUT_ACTIVE_ALT - Нажата клавиша Alt.

Пример смены значения переменной red, если нажимаем "r", то = 0.0, и нажимаем Alt + r, то = 1.0. Следующий фрагмент кода будет обрабатывать данную комбинацию нажатий:

```
void processNormalKeys(unsigned char key, int x, int y) {
    if (key == 27)
        exit(0);
    else if (key=='r'){
        int mod = glutGetModifiers();
        if (mod == GLUT_ACTIVE_ALT)
            red = 0.0;
        else
            red = 1.0;
    }
}
```

Перемещение камеры

Рассмотрим перемещение камеры на примере снеговиков, а для перемещения будут клавиши со стрелками. Клавиши «влево» и «вправо» будут вращать камеру вокруг оси «y», т.е. в плоскости «xz», тогда как клавиши «вверх» и «вниз» будут двигать камеру вперед и назад в пределах текущего направления. Во-первых, нужны глобальные переменные для хранения параметров камеры. Переменные будут хранить как положение камеры, так и вектор, который дает нам цель направления. Также требуется хранить угол поворота. Там нет необходимости хранить y компоненту, поскольку он является постоянным. Следовательно, нам нужно:

– Угол поворота вокруг оси «y». Эта переменная позволит поворачивать камеру.

– float x, z: переменные, хранящие положение камеры в плоскости «xz».

– float lx, lz: координаты вектора, определяющие направление перемещения камеры.

Определим такие переменные:

```
// угол поворота камеры
float angle=0.0;
// координаты вектора направления движения камеры
float lx=0.0f,lz=-1.0f;
// XZ позиция камеры
float x=0.0f,z=5.0f;
```

Код, описывающий сцену со снеговиком.

```
void drawSnowMan() {

    glColor3f(1.0f, 1.0f, 1.0f);

// тело снеговика
    glTranslatef(0.0f ,0.75f, 0.0f);
    glutSolidSphere(0.75f,20,20);

// голова снеговика
    glTranslatef(0.0f, 1.0f, 0.0f);
    glutSolidSphere(0.25f,20,20);
// глаза снеговика
    glPushMatrix();
    glColor3f(0.0f,0.0f,0.0f);
    glTranslatef(0.05f, 0.10f, 0.18f);
    glutSolidSphere(0.05f,10,10);
    glTranslatef(-0.1f, 0.0f, 0.0f);
    glutSolidSphere(0.05f,10,10);
```

```
        glPopMatrix();
    // нос снеговика
        glColor3f(1.0f, 0.5f , 0.5f);
        glRotatef(0.0f,1.0f, 0.0f, 0.0f);
        glutSolidCone(0.08f,0.5f,10,2);
    }
```

Модификация функции визуализации

Первое изменение в функции gluLookAt. Параметры gluLookAt функции теперь стали переменными вместо фиксированных значений. Функция gluLookAt обеспечивает простой и интуитивно понятный способ определения положения и ориентации камеры. В основном это три группы параметров, каждый из которых состоит из 3 значений с плавающей точкой. Первые три значения указывают положение камеры. Второй набор значений определяет точку зрения камеры. На самом деле это может быть любая точка в линии обзора. Последняя группа параметров указывает на вектор, это, как правило, установленные значения (0.0, 1.0 , 0.0), это означает, что камера не наклонена. Если требуется наклонить камеру - просто изменяются параметры. Например, чтобы видеть все вверх тормашками, нужно ввести следующие значения (0.0 , -1.0, 0.0). Итак, первый набор параметров x, y, и z - начальное положение камеры. Второй набор параметров, это направление вектора обзора, рассчитывается сложением вектора, который определяет нашу линию обзора с положением камеры. Точка = прямая видимость + координаты положения камеры. Код функции рендеринга:

```
    void renderScene(void) {
        // Очистка буфера цвета и глубины.
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        // обнулить трансформацию
        glLoadIdentity();
        // установить камеру
        gluLookAt(   x, 1.0f,     z,
                  x+lx, 1.0f,  z+lz,
                  0.0f, 1.0f,  0.0f );
        // нарисуем «землю»
        glColor3f(0.9f, 0.9f, 0.9f);
        glBegin(GL_QUADS); // полигон с координатами
            glVertex3f(-100.0f, 0.0f, -100.0f);
            glVertex3f(-100.0f, 0.0f,  100.0f);
            glVertex3f( 100.0f, 0.0f,  100.0f);
            glVertex3f( 100.0f, 0.0f, -100.0f);
        glEnd();
        // Рисование 64 снеговиков
        for(int i = -4; i < 4; i++)
            for(int j=-4; j < 4; j++) {
                glPushMatrix();
                glTranslatef(i*5.0, 0, j * 5.0);
```

```
                drawSnowMan();
                glPopMatrix();
            }
        glutSwapBuffers();
    }
```

Теперь нужно создать функцию перемещения камеры, обрабатывающую нажатия клавиш перемещения. Используем клавиши «влево» и «вправо», чтобы повернуть камеру, или изменить вектор, который определяет линию обзора. Клавиши «вверх» и «вниз» используются для перемещения вдоль нынешней линии обзора. Когда пользователь нажимает левую или правую клавиши, переменная угла наклона изменяется соответствующим образом. На основании значения угла она вычисляет соответствующие значения для новых компонентов lx и lz линии вектора обзора. Обратите внимание, что движения происходят в плоскости «xz», поэтому не нужно, чтобы менялась ly координата линии вектора обзора. Новые значения lx и lz отображаются на единичном круге на плоскости «xz». Поэтому, учитывая переменную угла ang, новые значения для lx и lz, расчитываются по формуле:

```
lx = sin(ang);
lz = -cos(ang);
```

Эти действия производятся, если требуется конвертировать полярные координаты в декартовы. lz является отрицательным, потому что начальное значение равно -1. Камера не двигается при обновлении lx и lz, положение камеры остается, только меняется точка обзора. Чтобы перемещать камеру вдоль линии обзора, необходимо добавить или вычесть часть линии вектора обзора текущей позиции, когда стрелки «вверх» и «вниз» будут нажаты соответственно. Например, для перемещения камеры вперед новые значения для переменных x и z будут:

```
x = x + lx * fraction;
z = z + lz * fraction;
```

fraction - возможная реализация скорости. (lx, lz) является единичным вектором (его точка в единичном круге), поэтому если fraction остается постоянным, то скорость останется постоянной. При увеличении значения fraction перемещение происходит быстрее.

Функция обработки нажатия клавиш перемещения

```
void processSpecialKeys(int key, int xx, int yy) {
    float fraction = 0.1f;
    switch (key) {
        case GLUT_KEY_LEFT :
            angle -= 0.01f;
            lx = sin(angle);
            lz = -cos(angle);
            break;
        case GLUT_KEY_RIGHT :
            angle += 0.01f;
```

```
                lx = sin(angle);
                lz = -cos(angle);
                break;
        case GLUT_KEY_UP :
                x += lx * fraction;
                z += lz * fraction;
                break;
        case GLUT_KEY_DOWN :
                x -= lx * fraction;
                z -= lz * fraction;
                break;
    }
}
```

Работа с мышью

Интерфейс мыши GLUT предоставляет собой множество вариантов использования, а именно обнаружение кликов и движения мыши.

Обнаружение щелчков мыши

Как и в случае с клавиатурой, GLUT предлагает способ, чтобы зарегистрировать функцию, которая будет отвечать за обработку событий, создаваемых щелчками клавиш мыши. Название этой функции glutMouseFunc. Синтаксис выглядит следующим образом:

void glutMouseFunc(void (*func)(int button, int state, int x, int y));

Параметры:

*func - имя функции, которая будет обрабатывать события мыши.

С момента подписания glutMouseFunc, функция, которая будет обрабатывать события мыши, должна иметь 4 параметра. Первый из них касается того, какая кнопка была нажата или отпущена. Этот аргумент может иметь одно из трех значений:

– GLUT_LEFT_BUTTON;

– GLUT_MIDDLE_BUTTON;

– GLUT_RIGHT_BUTTON.

Второй аргумент относится к состоянию кнопки, то есть идентификации момента нажатия и отпускания кнопки. Возможные значения:

– GLUT_DOWN;

– GLUT_UP.

Если ответный вызов генерируется со статусом GLUT_DOWN, приложение может предположить, что GLUT_UP будет после этого события, даже если мышь перемещается за пределы окна. Остальные два параметра обеспечивают (x, y) координаты мыши относительно левого верхнего угла рабочей области окна.

Определение перемещения мыши

В GLUT существует возможность обнаружения движения мыши для приложения. Есть два типа движения для GLUT: активные и пассивные

движения. Активное движение происходит при перемещении мыши и нажатия кнопки. Пассивные движения, когда мышь движется, но ни одна кнопка не нажата. Если приложение отслеживает движение манипулятора, будет сгенерировано событие в кадре во время периода, когда мышь движется.

GLUT позволяет определить две различные функции для обработки событий движения: одна для отслеживания пассивных движений, а другая, чтобы отслеживать активные движения.

Синтаксис GLUT функций слежения за перемещением:

void glutMotionFunc(void (*func) (int x,int y));

void glutPassiveMotionFunc(void (*func) (int x, int y));

Параметры:

*func - функция, которая будет отвечать за соответствующий тип движения.

Параметры функции обработки движения мыши являются (x, y) декартовы координаты курсора мыши относительно левого верхнего угла рабочей области окна.

Обнаружение попадания или выхода из рабочей области окна курсора мыши

GLUT должен определять, когда мышь покидает или входит в рабочую область окна. GLUT функцией регистрации является обратный вызов glutEntryFunc и синтаксис выглядит следующим образом:

void glutEntryFunc(void (*func)(int state));

Параметры:

*func - функция, которая будет обрабатывать эти события.

Параметр функции, которая будет обрабатывать эти события сообщает нам, если мышь попадает в левую область окна. GLUT определяет две константы, которые можно использовать в приложении:

– GLUT_LEFT;

– GLUT_ENTERED.

Управление камерой с использованием мыши

Когда пользователь нажимает левую кнопку мыши, записывается X-координата положения мыши. При перемещении мыши будет проверяться новая позиция по X, и на основе разницы устанавливается переменная deltaAngle. Эта переменная будет добавлена в начальный угол для вычисления направления камеры. Переменная для хранения позиции, где происходит щелчок мыши по координате X также необходима.

float deltaAngle = 0.0f;

int xOrigin = -1;

xOrigin инициализируется в отрицательное значение, которое никогда не происходит, когда нажата кнопка мыши (она должна быть по меньшей мере равна нулю). Это позволит отличить, если пользователь нажимает на левую кнопку или любую другую.

Следующая функция отвечает за обработку изменения состояния кнопки:

```
void mouseButton(int button, int state, int x, int y) {

    // только при начале движения, если нажата левая кнопка
    if (button == GLUT_LEFT_BUTTON) {

        // когда кнопка отпущена
        if (state == GLUT_UP) {
            angle += deltaAngle;
            xOrigin = -1;
        }
        else  {// state = GLUT_DOWN
            xOrigin = x;
        }
    }
}
```

Обратите внимание, что переменная xOrigin имеет значение -1, если кнопка зажата.

Функция для обработки движения мыши:

```
void mouseMove(int x, int y) {

    // если левая кнопка опущена
    if (xOrigin >= 0) {

        // обновить deltaAngle
        deltaAngle = (x - xOrigin) * 0.001f;
        // Обновление направления камеры
        lx = sin(angle + deltaAngle);
        lz = -cos(angle + deltaAngle);
    }
}
```

В функции main() регистрируются две новые функции обратного вызова.

Наложение текстуры

Текстурирование позволяет наложить изображение на многоугольник и вывести этот многоугольник с наложенной на него текстурой, соответствующим образом преобразованной. OpenGL поддерживает одно- и двумерные текстуры, а также различные способы наложения текстуры.

Для использования текстуры надо сначала разрешить одно- или двумерное текстурирование при помощи команд glEnable(GL_TEXTURE1D) или glEnable(GL_TEXTURE_2D).

Для задания двумерной текстуры служит процедура: glTexlmage2D(GLenum target, GLint level, GLint component, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *pixels).

Параметр target зарезервирован для будущего использования и в текущей версии OpenGL должен быть равен GL_TEXTURE_2D. Параметр level используется в том случае, если задается несколько разрешений данной текстуры. При ровно одном разрешении он должен быть равным нулю.

Следующий параметр – component – целое число от 1 до 4, показывающее, какие из RGBA-компонентов выбраны для использования. Значение 1 выбирает компонент R, значение 2 выбирает R и A компоненты, 3 соответствует R, G и B, а 4 соответствует компонентам RGBA.

Параметры width и height задают размеры текстуры, border задает размер границы (бортика), обычно равный нулю. Как параметр width, так и параметр height, должны иметь вид $2^n + 2b$, где n – целое число, а b – значение параметра border. Максимальный размер текстуры зависит от реализации OpenGL, но он не менее 64 на 64.

При текстурировании OpenGL поддерживает использование пирамидального фильтрования (mip-mappping). Для этого необходимо иметь текстуры всех промежуточных размеров, являющихся степенями двух, вплоть до 1x1, и для каждого такого разрешения вызвать glTexImage2D с соответствующими параметрами level, width, height и image. Кроме того, необходимо задать способ фильтрования, который будет применяться при выводе текстуры.

Под фильтрованием здесь подразумевается способ, которым для каждого пикселя будет выбираться подходящий элемент текстуры (тексель). При текстурировании возможна ситуация, когда 1 пикселю соответствует небольшой фрагмент текселя (увеличение) или же, наоборот, когда 1 пикселю соответствует целая группа текселей (уменьшение).

Способ выбора соответствующего текселя как для увеличения, так и для уменьшения (сжатия) текстуры необходимо задать отдельно. Для этого используется процедура:

glTexParameteri(GL_TEXTURE_2D, GLenum p1, GLenum p2),

где параметр p1 показывает, задается ли фильтр для сжатия или для растяжения текстуры, принимая значение GL_TEXTURE_MIN_FLITER или

GL_TEXTURE_MAG_FILTER. Параметр p2 задает способ фильтрования.

При использовании пирамидального фильтрования помимо выбора текселя на одном слое текстуры появляется возможность либо выбрать один соответствующий слой, либо проинтерполировать результаты выбора между двумя соседними слоями. Для правильного применения текстуры каждой вершине следует задать соответствующие ей координаты текстуры при помощи процедуры:

glTexCoord{1 2 3 4}{s i f d}[v](TYPE coord, ...).

Этот вызов задаёт значения индексов текстуры для последующей команды glVertex.

Если размер грани больше, чем размер текстуры, то для циклического повторения текстуры служат команды:

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_S_WRAP,

GL_REPEAT);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_T_WRAP, GL_REPEAT).

Координаты текстуры обычно подвергаются преобразованию при помощи матрицы текстурирования. По умолчанию она совпадает с единичной матрицей, но пользователь сам имеет возможность задать преобразования текстуры, например, следующим образом:

glMatrixMode(GL_TEXTURE);

glRotatef(...};

glMatrixMode(GL_MODELV1EW).

При выводе текстуры OpenGL может использовать линейную интерполяцию или точно учитывать перспективное искажение. Для задания точного текстурирования служит команда:

glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST).

Если качество не играет большой роли, а нужна высокая скорость рендеринга, то в качестве последнего аргумента следует использовать константу GL_FASTEST.

Описанный выше способ работ с текстурами используется в OpenGL версии 1.0. В более новых версиях OpenGL, начиная с версии 1.1, введены дополнительные функции, повышающие удобство работы с текстурами. В OpenGL 1.0 процедуру glTexImage2D необходимо вызывать всякий раз, когда нужно сменить текущую текстуру. Это достаточно медленный способ. В OpenGL 1.1 имеется возможность присваивать имена текстурам и затем сменять текущую текстуру только указанием имени новой текстуры, без повторной её загрузки в память процедурой glTexImage2D.

Имя текстуры представляет собой уникальное значение типа GLuint. Перед использованием текстуры необходимо присвоить ей имя. Имена текстур можно сгенерировать при помощи процедуры glGenTextures(GLsizei n, GLuint *textures).

Параметр n определяет количество текстур, для которых необходимо сгенерировать имена. Параметр textures является указателем на массив переменных типа GLuint, состоящим из n элементов. После вызова процедуры каждый элемент массива будет содержать уникальное имя текстуры, которое затем может быть использовано при работе.

Для выбора текущей (активной) текстуры используется функция: glBindTexture(GLenum target, GLuint texture).

Параметр target определяет тип текстуры (одномерная – GL_TEXTURE_1D или двумерная – GL_TEXTURE_2D). На практике более часто используются двумерные текстуры, которые представляют собой обычные двумерные изображения. Параметр texture определяет имя текстуры, которую необходимо сделать активной.

После того, как установлена активная текстура, можно вызвать процедуру glTexImage2D и задать параметры текстуры, а также сами её тексели. После вызова процедуры glTexImage2D текстура готова к применению.

Для того чтобы наложить текстуру на объект или многоугольник достаточно установить активную текстуру (процедура glBindTexture) и определить текстурные координаты при помощи процедуры glTexCoord.

Достоинство использования функций OpenGL 1.1 для работы с текстурами заключается не только в более высоком быстродействии по сравнению с использованием процедуры glTexImage2D, но и в повышенном удобстве работы с текстурами. Создав массив текстурных имён можно работать одновременно с несколькими текстурами, вызывая лишь функцию glBindTexture по мере необходимости.

При написании программы, которая покрывает куб текстурой, необходимо учитывать то, что саму текстуру требуется загружать из отдельного файла.

Для загрузки текстуры, а также задания ее параметров рекомендуется написать отдельную функцию. Например, если текстура размером 64x64, то считывание ее в массив выглядит так:

```
Bits: Array [0..63, 0..63, 0..2] of GLubyte;
For i := 0 to 63 do
    For j := 0 to 63 do begin
      bits [i, j, 0] := GetRValue(bitmap.Canvas.Pixels[i,j]);
      bits [i, j, 1] := GetGValue(bitmap.Canvas.Pixels[i,j]);
      bits [i, j, 2] := GetBValue(bitmap.Canvas.Pixels[i,j]);
end.
```

Последний компонент трехмерного массива – это цветовые составляющие пиксела RGB. Загрузить текстуру в память можно следующей командой:

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 64, 64,  0, GL_RGB, GL_UNSIGNED_BYTE, @Bits);
```

После этого необходимо разрешить работать с текстурой:

```
glEnable(GL_TEXTURE_2D);
```

Следующий этап – это привязка текстурных координат к координатам объектов. В обработчике создания окна пишем (для одной грани куба):

```
glBegin (GL_QUADS);
      glTexCoord2d (1.0, 0.0);
      glVertex3f (-1.0, -1.0, 1.0);
      glTexCoord2d (1.0, 1.0);
      glVertex3f (1.0, -1.0, 1.0);
      glTexCoord2d (0.0, 1.0);
      glVertex3f (1.0, -1.0, -1.0);
      glTexCoord2d (0.0, 0.0);
      glVertex3f (-1.0, -1.0, -1.0);
glEnd.
```

Пример построения и анимации снеговика.

Результат выполнения программы

Текст программы

```
#include "stdafx.h"
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>
AUX_RGBImageRec* photo_image;
 unsigned int photo_tex;
void CALLBACK resize(int width,int height)
{glViewport(0,0,width,height);
   glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glOrtho(-5,5, -5,5, 2,12);
   gluLookAt( 0,0,5, 0,0,0, 0,1,0 );
   glMatrixMode( GL_MODELVIEW );
}
void CALLBACK display(void)
{
GLUquadricObj *quadObj;
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
        quadObj = gluNewQuadric();
        gluQuadricTexture(quadObj, GL_TRUE);
 gluQuadricDrawStyle(quadObj, GLU_FILL);
glColor3d(1,1,1);
 glRotated(0.5, 0,1,0);
 glPushMatrix();
 glTranslated(-0.8,-3,0);
  glRotated(-90, 1,0,0);
  gluSphere(quadObj,1.3, 16, 16);
 glPopMatrix();
gluDeleteQuadric(quadObj);
```

```
        glPushMatrix ();
        glTranslated(-0.8,-3,0);
      glColor3d(1,1,1);
     //auxSolidSphere(1.3);
        glTranslated(0,2.1,0);
     glColor3d(1,1,1);
      auxSolidSphere(1);
        glTranslated(0,1.5,0); //голова
      glColor3d(1,1,1);
      auxSolidSphere(0.7);
        glTranslated(-0.25,0.35,0.6); //первый глаз
      glColor3d(0,0,1);
      auxSolidSphere(0.1);
        glTranslated(0.5,0,0); //второй глаз
      glColor3d(0,0,1);
      auxSolidSphere(0.1);
        glTranslated(0.9,-1.4,-0.6); //первая рука
     glColor3d(1,1,1);
     auxSolidSphere(0.4);
        glTranslated(-2.2,0,0); //вторая рука
      glColor3d(1,1,1);
      auxSolidSphere(0.4);
        glTranslated(1.05,1.2,0); //нос
      glColor3d(1,0,0);
      auxSolidCone(0.2,2);
     glTranslated(0,0.4,0);
        glRotated(-90,1,0,0);
     glColor3d(0,0,1);
      auxSolidCone(0.5,2);
        glPopMatrix();
        auxSwapBuffers();
}
void main() {
float pos[4] = {3,3,3,1};
float color[4] = {1,1,1,1};
float sp[4] = {1,1,1,1};
 float mat_specular[] = {1,1,1,1};
    auxInitPosition( 50, 10, 400, 400);
    auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
     auxInitWindow( "Lab Work OpenGL" ); //заголовок окна
    auxIdleFunc(display);
     auxReshapeFunc(resize);
     glEnable(GL_DEPTH_TEST);
     glEnable(GL_COLOR_MATERIAL);
     glEnable(GL_LIGHTING);
```

```
            glEnable(GL_LIGHT0);
            glEnable(GL_LIGHT1);
            glEnable(GL_LIGHT2);
           glLightfv(GL_LIGHT0, GL_SPECULAR, sp);
            glLightfv(GL_LIGHT1, GL_SPECULAR, sp);
           glLightfv(GL_LIGHT2, GL_SPECULAR, sp);     color[1]=color[2]=0;
           glLightfv(GL_LIGHT0, GL_DIFFUSE, color);
            color[0]=0;color[1]=1;
           glLightfv(GL_LIGHT1, GL_DIFFUSE, color);
            color[1]=0;color[2]=1;
           glLightfv(GL_LIGHT2, GL_DIFFUSE, color);
           glLightfv(GL_LIGHT0, GL_POSITION, pos);
              pos[0] = -3;
           glLightfv(GL_LIGHT1, GL_POSITION, pos);
               pos[0]=0;pos[1]=-3;
           glLightfv(GL_LIGHT2, GL_POSITION, pos);
glEnable(GL_TEXTURE_2D);
           photo_image = auxDIBImageLoad("texture.bmp");
              glGenTextures(1, &photo_tex);
           glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
              glBindTexture(GL_TEXTURE_2D,photo_tex);
           gluBuild2DMipmaps(GL_TEXTURE_2D, 3,
                     photo_image->sizeX,
                     photo_image->sizeY,
                     GL_RGB, GL_UNSIGNED_BYTE,
                     photo_image->data);
         auxMainLoop(display);
        }
```

Пример построения вентилятора с вращающимися лопастями.

Результат выполнения программы

Текст программы

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
 #include <GL/glaux.h>
void CALLBACK resize(int width,int height)
{
   glViewport(0,0,width,height);
   glMatrixMode( GL_PROJECTION );
   glLoadIdentity();
    glOrtho(-5,5, -1,9, -10,10);
   gluLookAt( 2,0,5, 0,0,0, 0,1,0 );
   glMatrixMode( GL_MODELVIEW );
}
void CALLBACK display(void)
{
        static int angle=0;
        const speed=5;
GLdouble equation[4] = {0,-1,0,0.5};
GLdouble equation1[4] = {0,1,0,0.5};
GLdouble equation2[4] = {-1,0,0,0.5};
 GLdouble equation3[4] = {1,0,0,0.5};
 glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
 // Podstavka
 glEnable(GL_CLIP_PLANE0);
  glClipPlane(GL_CLIP_PLANE0, equation);
   glEnable(GL_CLIP_PLANE1);
  glClipPlane(GL_CLIP_PLANE1, equation1);
    glColor3d(1,0.75,0.75);
     auxSolidSphere( 3 );
 glDisable(GL_CLIP_PLANE0);
 glDisable(GL_CLIP_PLANE1);
 //nozka
 glPushMatrix();
        glTranslated(0,2.5,0);
        auxSolidCylinder(0.5,4);
  glPopMatrix();
  //korpus;
glPushMatrix();
        glTranslated(0,4,0);
        glRotated(90,0,0,1);
        glTranslated(0,1,0);
```

```
        auxSolidCylinder(1,5);
        glTranslated(0,-4.1,0);
        equation[3]=0;
        glEnable(GL_CLIP_PLANE0);
glClipPlane(GL_CLIP_PLANE0, equation);
        auxSolidSphere(0.8);
        glDisable(GL_CLIP_PLANE0);
        auxSolidCylinder(0.5,1);
        glTranslated(0,-0.3,0);
        glPushMatrix();
        glRotated(-angle*speed,0,1,0);
        glTranslated(0,0,1.5);
        glRotated(30,0,0,1);
        auxSolidBox(0.1,0.5,2);
        glPopMatrix();
        glPushMatrix();
        glRotated(-angle*speed+120,0,1,0);
        glTranslated(0,0,1.5);
        glRotated(30,0,0,1);
        auxSolidBox(0.1,0.5,2);
        glPopMatrix();
        glPushMatrix();
        glRotated(-angle*speed-120,0,1,0);
        glTranslated(0,0,1.5);
        glRotated(30,0,0,1);
        auxSolidBox(0.1,0.5,2);
        glPopMatrix();
  glPopMatrix();
   angle++;
   auxSwapBuffers();
 }
  void main()
  {
 float pos[4] = {3,3,3,1};
  float dir[3] = {-1,-1,-1};
     GLfloat mat_ambient[] = { 1.0f, 1.0f, 1.0f, 1.0f };
     GLfloat mat_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
      auxInitPosition( 50, 10, 400, 400);
     auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
     auxInitWindow( "ClipPlane" );
      auxIdleFunc(display);
      auxReshapeFunc(resize);
      glEnable(GL_DEPTH_TEST);
      glEnable(GL_COLOR_MATERIAL);
     glEnable(GL_LIGHTING);
```

```
    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_POSITION, pos);
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir);
   glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
   glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialf(GL_FRONT, GL_SHININESS, 50.0);
   auxMainLoop(display);
}
```

Пример построения анимированного чайника.

Результат выполнения программы

Трехмерный вращающийся чайник, который:

– по событию левой кнопки мыши меняет цвет;

– по событию правой кнопки мыши включает или выключает режим вращения;

– по событию клавиш ВВЕРХ/ВНИЗ увеличивает/замедляет вращение;

– по событию клавиши ПРОБЕЛ меняет оси вращения.

Текст программы

```
#include "stdafx.h"
#include <windows.h>
 #include <GL\gl.h>
#include <GL\GLU.h>
 #include <GL\GLAUX.h>
int clr_number=2;//порядковой номер текущего цвета bool
flag=true;//включение/выключение вращения int d=5;//приращение угла поворота
 int ax=1;//положение оси вращения
  void CALLBACK resize(int width, int Height);
void CALLBACK display(void);
void CALLBACK mouse_leftbtn( AUX_EVENTREC* event); //обработчик левой клавиши мыши
void CALLBACK mouse_rightbtn(AUX_EVENTREC* event); // обработчик правой клавиши мыши
void CALLBACK keydown(void); //обработчик нажатия клавиши DOWN void CALLBACK keyup(void); //обработчик нажатия клавиши UP
void CALLBACK keyspace(void); // обработчик нажатия клавиши SPACE int main(int argc, char* argv[])
{
    //цветовой режим, удаление невидимых линий и двойная буферизация
```

```
            auxInitDisplayMode( AUX_RGBA | AUX_DEPTH | AUX_DOUBLE);
            auxInitPosition(50,10,400,400); //позиция и размеры окна
            auxInitWindow("Вращающийся чайник");//заголовок окна
            auxReshapeFunc(resize);//реакция на изменение размеров окна
            glEnable(GL_ALPHA_TEST); //учет прозрачности
          glEnable(GL_DEPTH_TEST);//удаление невидимых линий
            glEnable(GL_COLOR_MATERIAL);//синхронное задание цвета рисования и материала
            glEnable(GL_BLEND);//разрешение смешивания цветов
            glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
            glEnable(GL_LIGHTING);//учет освещения
            glEnable(GL_LIGHT0);//включение нулевого источника света

        //задание положения и направления нулевого источника света
            float pos[4]={0,5,5,1};
            float dir[3]={0,-1,-1};
            glLightfv(GL_LIGHT0, GL_POSITION, pos);
            glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir);
            auxIdleFunc(display);//задание анимации
            auxMouseFunc( AUX_LEFTBUTTON,AUX_MOUSEDOWN, mouse_leftbtn);
            auxMouseFunc(AUX_RIGHTBUTTON, AUX_MOUSEDOWN, mouse_rightbtn);
            auxKeyFunc(AUX_DOWN, keydown);
            auxKeyFunc(AUX_UP, keyup);
            auxKeyFunc(AUX_SPACE, keyspace);
            auxMainLoop(display);//перерисовка окна
            return 0;
        }
        void CALLBACK resize (int width, int height)
        {
            glViewport(0,0,width,height);
            glMatrixMode(GL_PROJECTION);
            glLoadIdentity();
            glOrtho(-5,5,-5,5,2,12);//задание типа проекции
            gluLookAt( 3,0,5, 0,0,0, 0,1,0);//задание позиции наблюдателя
            glMatrixMode(GL_MODELVIEW);
            glLoadIdentity();
        }
        void CALLBACK display (void)
        {
            //очистка буфера кадра
            glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
            static float angle=0;//угол поворота чайника
```

```
        switch (clr_number)
        {
        case 0: glColor3f(0.5,0.5,0);
 break;
        case 1: glColor3f(1,0,0);
break;
        case 2: glColor3f(0,1,0);
 break;
        case 3: glColor3f(0,0,1);
 break;
        default : glColor3f(1,1,1);
        }
        glPushMatrix();
        switch (ax)
        {
        case 0: glRotatef(angle,1,0,0); break;
        case 1: glRotatef(angle,0,1,0); break;
        case 2: glRotatef(angle,0,0,1); break;
        default : glRotatef(angle,0.5,0.5,0);
        }
        auxWireTeapot(2);
        glPopMatrix();
        if (flag)
        {
          angle+=d;
          if (angle>360) angle-=360;
        }
        //копирование содержимого буфера кадра на экран
        glFlush();
        auxSwapBuffers();
}
void CALLBACK mouse_leftbtn( AUX_EVENTREC* event)
{
        if ( ++clr_number==4)
                clr_number=0;
}
void CALLBACK mouse_rightbtn(AUX_EVENTREC* event)
{
        if (flag) flag=false;
                else flag=true;
}
void CALLBACK keydown(void)
{
        if (d>1) d--;
}
```

```
void CALLBACK keyup(void)
{
	if (d<5) d++;

}
void CALLBACK keyspace(void)
{
	if ( ++ax==3) ax=0;
}
```

Ход работы

В рамках данной лабораторной работы необходимо, анимировать объекты, реализованные в лабораторной работе №7. Нанести текстуры на объект и реализовать возможность вращения камеры вокруг него при помощи мыши.

Контрольные вопросы

1. Опишите функции для работы с клавиатурой в OpenGL?
2. Опишите функции для работы с мышью в OpenGL.
3. Что такое «нормальные» нажатия клавиш?
4. Опишите работу с камерой в OpenGL.
5. Назовите функцию для обнаружения щелчков мыши в OpenGL?