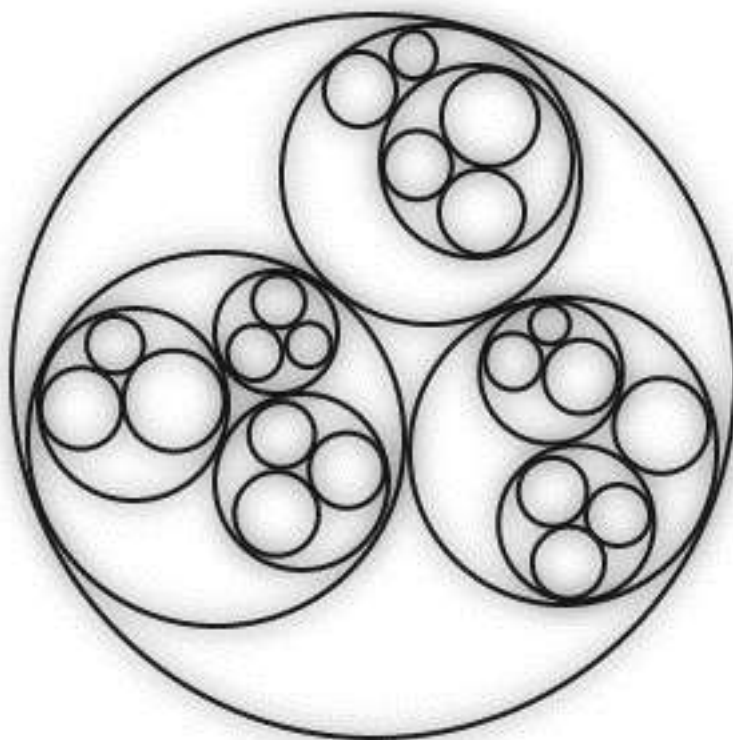


Визуализация программного обеспечения



ОБЩИЕ ПОЛОЖЕНИЯ

Материал, изложенный в методических указаниях, необходим при выполнении лабораторных работ по курсу "Визуализация программного обеспечения"

Выполнение лабораторных работ осуществляется с использованием языка программирования Visual C++ и библиотеки OpenGL.

Лабораторные работы предназначены:

- для закрепления теоретического материала по математическим и алгоритмическим основам компьютерной графики;
- получения практических знаний составления и реализации математических моделей средствами компьютерной графики;
- закрепления знаний основ объектно-ориентированного программирования (Visual C++);
- изучения библиотеки OpenGL.

Студентам предлагается выполнить ряд лабораторных работ (ЛР) по созданию двухмерной (2D) и трехмерной (3D) графики:

ЛР 1. Моделирование 2D объектов.

ЛР 2. Моделирование 3D объектов. Работа с цветом и светом.

ЛР 3. Текстурирование объектов.

ЛР 4. Моделирование 3D анимации.

ЛР 5. Выполнение индивидуального творческого задания.

1. ОСНОВЫ ПРОГРАММИРОВАНИЯ 3D ГРАФИКИ

1.1. Возможности OpenGL

OpenGL – Open Graphics Library, открытая графическая стандартная библиотека для программирования трехмерной графики для многих 32-разрядных операционных систем (Windows, Linux в том числе).

В отличие от Direct3D, которая характерна только для Windows, OpenGL содержит в себе более 250 процедур и функций для построения 3D графики и рендеринга. Они находятся в `opengl32.dll` (Windows\ system\) и в расширении `glu32.dll`.

К основным возможностям OpenGL можно отнести:

- *геометрические* (точки, линии, полигоны) и *растровые* (битовый массив (bitmap) и образ (image)) *примитивы*;
- *использование B-сплайнов* для рисования кривых по опорным точкам;
- *альфа-канал*. Позволяет делать предметы прозрачными, уровень прозрачности от 0 до 100 %;
- *антиалиасинг* (сглаживание) цветовых переходов для получения более ре-

листического изображения;

- *буфер аккумулятора.* Дополнительный буфер для спецэффектов и глобального сглаживания по всей сцене;
- *градиентная заливка* полигонов и отрезков;
- *двойная буферизация.* Для устранения мерцания при мультипликации. Изображение каждого кадра сначала рисуется во втором (невидимом) буфере, а потом, когда кадр полностью нарисован, весь буфер отображается на экране;
- *заливка и освещенность фактур.* К фактурам применяются эффекты освещенности и затенения в зависимости от характеристик «материала» ;
- *пространственные преобразования.* Масштабирование, вращение и перемещение объектов в пространстве;
- *текстуры (меппинг)* Наложение двухмерных изображений на объемные поверхности для придания сцене реализма;
- *атмосферные эффекты,* такие как туман, дым, дымка делают изображения, созданные компьютером, более реалистичными.

OpenGL позволяет:

- 1) создавать объекты из геометрических примитивов (точки, линии, грани и битовые изображения).
- 2) располагать объекты в трёхмерном пространстве и выбирать способ и параметры проецирования.
- 3) вычислять цвет всех объектов. Цвет может быть как явно задан, так и вычисляться с учётом источников света, параметров освещения, текстур.
- 4) переводить математическое описание объектов и связанной с ними информации о цвете в изображение на экране.

При этом OpenGL может осуществлять дополнительные операции, такие, как удаление невидимых фрагментов изображения.

1.2. Основные типы данных OpenGL

Все команды (процедуры и функции) OpenGL начинаются с префикса **gl**, а все константы – с префикса **GL_**. Кроме того, в имена функций и процедур OpenGL входят суффиксы, несущие информацию о числе передаваемых параметров и о их типе. В табл. 1 приводятся вводимые OpenGL типы данных, стандартные типы языка C, которым они соответствуют, и суффиксы, которым они соответствуют.

Некоторые команды OpenGL оканчиваются на букву *v*. Это говорит о том, что команда получает указатель на массив значений, а не сами эти значения в виде отдельных параметров. Многие команды имеют как векторные, так и не векторные версии. Например, конструкции:

```
glColor3f(1.0, 1.0, 1.0);
```

и

```
GLfloat color[] = {1.0, 1.0, 1.0};
```

```
glColor3fv(color);
```

ЭКВИВАЛЕНТНЫ.

Таблица 1. Типы данных OpenGL

Суффикс	Описание	Тип в C	Тип в OpenGL
b	8-битовое целое	char	GLbyte
s	16-битовое целое	short	GLshort
i	32-битовое целое	long	GLint GLsizei
f	32-битовое вещественное число	float	GLfloat, GLclampf
d	64-битовое вещественное число	double	GLdouble, GLclampd
ub	8-битовое беззнаковое целое	unsigned char	GLubyte, GLboolean
us	16-битовое беззнаковое целое	unsigned short	GLushort
ui	32-битовое беззнаковое целое	unsigned long	GLuint, GLenum, GLbitfield

OpenGL можно рассматривать как автомат, находящийся в одном из нескольких состояний. Внутри OpenGL содержится целый ряд переменных, например, текущий цвет или текущий режим закрашивания. Если установить текущий цвет, то все последующие объекты будут этого цвета до тех пор, пока текущий цвет не будет изменён.

По умолчанию каждая системная переменная имеет своё значение, и в любой момент значение каждой из этих переменных можно узнать. Обычно для этого используется одна из следующих функций: `glGetBooleanv()`, `glGetDoublev()`, `glGetFloatv()` и `glGetIntegerv()`. Для определения значений некоторых переменных служат специальные функции.

1.3. Рисование геометрических объектов

1.3.1. Работа с буферами и задание цвета объектов

OpenGL содержит внутри себя несколько различных буферов. Среди них фрейм буфер (где строится изображение), z-буфер, служащий для удаления невидимых поверхностей, буфер трафарета и аккумулирующий буфер.

Для очистки внутренних буферов служит процедура `glClear(GLbitfield mask)`, очищающая буферы, заданные переменной `mask`. Параметр `mask` является комбинацией следующих констант:

`GL_COLOR_BUFFER_BIT` – очистить буфер изображения (фреймбуфер);

`GL_DEPTH_BUFFER_BIT` – очистить z-буфер;

`GL_ACCUM_BUFFER_BIT` – очистить аккумулирующий буфер;

`GL_STENCIL_BUFFER_BIT` – очистить буфер трафарета.

Цвет, которым очищается буфер изображения, задаётся процедурой `glClearColor(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha)`. Значение, записываемое в z-буфер, при очистке задаётся процедурой `glClearDepth(GLfloat depth)`. Значение, записываемое в буфер трафарета, при очистке задаётся проце-

дурой `glClearStencil(GLint s)`. Цвет, записываемый в аккумулирующий буфер, при очистке задаётся процедурой `glClearAccum(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha)`.

Сама команда `glClear` очищает одновременно все заданные буферы, заполняя их соответствующими значениями.

Для задания цвета объекта служит процедура:

`glColor{3 4}{b s i f d u b u s u i}[v](TYPE red, ...)`.

Цифра 3 или 4 указывает на количество требуемых аргументов, а буква, следующая за цифрой, показывает тип аргументов. Например, в процедуру `glColor3i` будут переданы три параметра целого типа.

Если значение параметра не задано, то оно автоматически полагается равным единице. Версии процедуры `glColor`, где параметры являются переменными с плавающей точкой, автоматически обрезают переданные значения в отрезок $[0, 1]$.

Процедура `glFlush()` вызывает немедленное рисование ранее переданных команд. При этом ожидания завершения всех ранее переданных команд не происходит. С другой стороны, команда `glFinish()` ожидает, пока не будут завершены все ранее переданные команды.

Если нужно включить удаление невидимых поверхностей методом z-буфера, то z-буфер необходимо очистить и передать команду `glEnable(GL_DEPTH_TEST)`. Команду `glEnable()` можно выполнить только один раз при инициализации системных переменных OpenGL. Очистку z-буфера необходимо производить перед началом построения очередного кадра изображения.

1.3.2. Задание графических примитивов

Все геометрические примитивы в OpenGL задаются вершинами (набором чисел, определяющих их координаты в пространстве).

OpenGL работает с однородными координатами (x, y, z, w) . Если координата z не задана, то она считается равной нулю. Если координата w не задана, то она считается равной единице.

Под линией в OpenGL подразумевается отрезок, заданный своими начальной и конечной вершинами.

Под гранью (многоугольником) в OpenGL подразумевается замкнутый выпуклый многоугольник с несамопересекающейся границей.

Все геометрические объекты в OpenGL задаются посредством вершин, а сами вершины задаются процедурой:

`glVertex{2 3 4}{s i f d}[v](TYPE x, ...)`,

где реальное количество параметров определяется первым суффиксом (2, 3 или 4), а суффикс `v` означает, что в качестве единственного аргумента выступает массив, содержащий необходимое количество координат. Например:

`glVertex2s(1, 2);`

`glVertex3f(2.3, 1.5, 0.2);`

`GLdouble vect[] = {1.0, 2.0, 3.0, 4.0};`

```
glVertex4dv(vect);
```

Для задания геометрических примитивов необходимо как-то выделить набор вершин, определяющих этот объект. Для этого служат процедуры `glBegin()` и `glEnd()`. Процедура `glBegin(GLenum mode)` обозначает начало списка вершин, описывающих геометрический примитив. Тип примитива задаётся параметром `mode`, который принимает одно из следующих значений:

`GL_POINTS` – набор отдельных точек;

`GL_LINES` – пары вершин, задающих отдельные точки;

`GL_LINE_STRIP` – незамкнутая ломаная;

`GL_LINE_LOOP` – замкнутая ломаная;

`GL_POLYGON` – простой выпуклый многоугольник;

`GL_TRIANGLES` – тройки вершин, интерпретируемые как вершины отдельных треугольников;

`GL_TRIANGLE_STRIP` – связанная полоса треугольников;

`GL_TRIANGLE_FAN` – веер треугольников;

`GL_QUADS` – четвёрки вершин, задающие выпуклые четырёхугольники;

`GL_QUAD_STRIP` – полоса четырёхугольников.

Процедура `glEnd()` отмечает конец списка вершин.

Между командами `glBegin()` и `glEnd()` могут находиться команды задания различных атрибутов вершин `glVertex()`, `glColor()`, `glNormal()`, `glCallList()`, `glCallLists()`, `glTexCoord()`, `glEdgeFlag()`, `glMaterial()`. Между командами `glBegin()` и `glEnd()` все остальные команды OpenGL недопустимы и приводят к возникновению ошибок.

Рассмотрим в качестве примера задание окружности:

```
glBegin(GL_LINE_LOOP);
for (int i = 0; i < N; i++)
{
    float angle = 2 * M_PI * i / N;
    glVertex2f(cos(angle), sin(angle));
}
glEnd();
```

Хотя многие команды могут находиться между `glBegin()` и `glEnd()`, вершины генерируются при вызове `glVertex()`. В момент вызова `glVertex()` OpenGL присваивает создаваемой вершине текущий цвет, координаты текстуры, вектор нормали и т. д. Изначально вектор нормали полагается равным (0, 0, 1), цвет полагается равным (1, 1, 1, 1), координаты текстуры полагаются равными нулю.

1.3.3. Рисование точек, линий и многоугольников

Для задания размеров точки служит процедура `glPointSize(GLfloat size)`, которая устанавливает размер точки в пикселях, по умолчанию он равен единице.

Для задания ширины линии в пикселях служит процедура `glLineWidth(GLfloat width)`. Шаблон, которым будет рисоваться линия, можно задать при помощи процедуры `glLineStipple(GLint factor, GLushort pattern)`.

Шаблон задается переменной `pattern` и растягивается в `factor` раз. Использование шаблонов линии необходимо разрешить при помощи команды `glEnable(GL_UNE_STIPPLE)`. Запретить использование шаблонов линий можно командой `glDisable(GL_LINE_STIPPLE)`.

Многоугольники рисуются как заполненные области пикселей внутри границы, хотя их можно рисовать либо только как граничную линию, либо просто как набор граничных вершин.

Многоугольник имеет две стороны, лицевую и нелицевую, и может быть отрисован по-разному в зависимости от того, какая сторона обращена к наблюдателю. По умолчанию обе стороны рисуются одинаково. Для задания того, как именно следует рисовать переднюю и заднюю стороны многоугольника, служит процедура `glPolygonMode(GLenum face, GLenum mode)`. Параметр `face` может принимать значения `GL_FRONT_AND_BACK` (обе стороны), `GL_FRONT` (лицевая сторона) или `GL_BACK` (нелицевая сторона). Параметр `mode` может принимать значения `GL_POINT`, `GL_LINE` или `GL_FILL`, обозначая, что многоугольник должен рисоваться как набор граничных точек, граничная ломаная линия или заполненная область, например:

```
glPolygonMode(GL_FRONT, GL_FILL);
glPolygonMode(GL_BACK, GL_LINE).
```

По умолчанию вершины многоугольника, которые появляются на экране в направлении против часовой стрелки, называются лицевыми. Это можно изменить при помощи процедуры `glFrontFace(GLenum mode)`. По умолчанию параметр `mode` равняется `GL_CCW`, что соответствует направлению обхода против часовой стрелки. Если задать этот параметр равным `GL_CW`, то лицевыми будут считаться многоугольники с направлением обхода вершин по часовой стрелке.

При помощи процедуры `glCullFace(GLenum mode)` вывод лицевых или нелицевых граней многоугольников можно запретить. Параметр `mode` принимает одно из значений `GL_FRONT` (оставить только лицевые грани), `GL_BACK` (оставить нелицевые) или `GL_FRONT_AND_BACK` (оставить все грани). Для отсечения граней необходимо разрешить отсечение при помощи команды `glEnable(GL_CULL_FACE)`.

Шаблон для заполнения грани можно задать при помощи процедуры `glPolygonStipple(const GLubyte * mask)`, где `mask` задает массив битов размером 32 на 32.

Для разрешения использования шаблонов при выводе многоугольников служит команда `glEnable(GL_POLYGON_STIPPLE)`.

Вектор нормали для каждой вершины можно задать при помощи одной из следующих процедур:

```
glNormal3{b s i d f}(TYPE nx, TYPE ny, TYPE nz);
glNormal3{b s I d f}v(const TYPE * v).
```

В версиях с суффиксами `b`, `s` или `i` значения аргументов масштабируются в отрезок $[-1, 1]$.

В качестве примера приведем процедуру, строящую прямоугольный па-

параллелепипед с ребрами, параллельными координатным осям, по диапазонам изменения x , y и z .

```
#include <windows.h>
#include <gl\gl.h>
drawBox(GLfloat x1, GLfloat x2, GLfloat y1, GLfloat y2, GLfloat z1, GLfloat
z2)
{
    glBegin ( GL_POLYGON );
    glNormal3f ( 0.0, 0.0, 1.0 );
    glVertex3f ( x1, y1, z2 );
    glVertex3f ( x2, y1, z2 );
    glVertex3f ( x2, y2, z2 );
    glVertex3f ( x1, y2, z2 );
    glEnd ();
    glBegin ( GL_POLYGON );
    glNormal3f ( 0.0, 0.0, -1.0 );
    glVertex3f ( x2, y1, z1 );
    glVertex3f ( x1, y1, z1 );
    glVertex3f ( x1, y2, z1 );
    glVertex3f ( x2, y2, z1 );
    glEnd ();
    glBegin ( GL_POLYGON );
    glNormal3f ( -1.0, 0.0, 0.0 );
    glVertex3f ( x1, y1, z1 );
    glVertex3f ( x1, y1, z2 );
    glVertex3f ( x1, y2, z2 );
    glVertex3f ( x1, y2, z1 );
    glEnd ();
    glBegin ( GL_POLYGON );
    glNormal3f ( 1.0, 0.0, 0.0 );
    glVertex3f ( x2, y1, z2 );
    glVertex3f ( x2, y1, z1 );
    glVertex3f ( x2, y2, z1 );
    glVertex3f ( x2, y2, z2 );
    glEnd ();
    glBegin ( GL_POLYGON );
    glNormal3f ( 0.0, 1.0, 0.0 );
    glVertex3f ( x1, y2, z2 );
    glVertex3f ( x2, y2, z2 );
    glVertex3f ( x2, y2, z1 );
    glVertex3f ( x1, y2, z1 );
    glEnd ();
    glBegin ( GL_POLYGON );
    glNormal3f ( 0.0, -1.0, 0.0 );
```



```

glVertex3f ( x2, y1, z2 );
glVertex3f ( x1, y1, z2 );
glVertex3f ( x1, y1, z1 );
glVertex3f ( x2, y1, z1 );
glEnd ();
}

```

1.3.4. Трехмерные фигуры

Функции для построения сплошных 3D фигур:

- auxSolidSphere(R) // сфера
- auxSolidCube(width) // куб
- auxSolidBox(width, height, depth) // коробка
- auxSolidTorus(r,R) // тор
- auxSolidCylinder(r,height) // цилиндр
- auxSolidCone(r,height) // конус
- auxSolidIcosahedron(width) // многогранники
- auxSolidOctahedron(width)
- auxSolidTetrahedron(width)
- auxSolidDodecahedron(width)
- auxSolidTeapot(width) // рисует чайник

Для построения каркасных фигур вместо Solid необходимо использовать Wire. Пример:

```
auxWireCube(1) // рисует каркасную модель куба.
```

1.4. Преобразование объектов в пространстве

В процессе построения изображения координаты вершин подвергаются определенным преобразованиям. Подобным преобразованиям подвергаются заданные векторы нормали.

Изначально камера находится в начале координат и направлена вдоль отрицательного направления оси Oz.

В OpenGL существуют две матрицы, последовательно применяющиеся в преобразовании координат. Одна из них – матрица моделирования (modelview matrix), а другая – матрица проецирования (projection matrix). Первая служит для задания положения объекта и его ориентации, вторая отвечает за выбранный способ проецирования. OpenGL поддерживает два типа проецирования – параллельное и перспективное.

Существует набор различных процедур, умножающих текущую матрицу (моделирования или проецирования) на матрицу выбранного геометрического преобразования.

Текущая матрица задается при помощи процедуры glMatrixMode(GLenum mode). Параметр mode может принимать значения GL_MODELVIEW, GL_TEXTURE или GL_PROJECTION, позволяя выбирать в качестве текущей матрицы матрицу моделирования (видовую матрицу), матрицу проецирования

или матрицу преобразования текстуры.

Процедура `glLoadIdentity()` устанавливает единичную текущую матрицу.

Обычно задание соответствующей матрицы начинается с установки единичной матрицы и последовательного применения матриц геометрических преобразований.

Преобразование переноса задается процедурой `glTranslate{f d}(TYPE x, TYPE y, TYPE z)`, обеспечивающей перенос объекта на величину (x, y, z) .

Преобразование поворота задается процедурой `glRotate{f d}(TYPE angle, TYPE x, TYPE y, TYPE z)`, обеспечивающей поворот на угол `angle` в направлении против часовой стрелки вокруг прямой с направляющим вектором (x, y, z) .

Преобразование масштабирования задается процедурой `glScale{f d}(TYPE x, TYPE y, TYPE z)`.

Если указано несколько преобразований, то текущая матрица в результате будет последовательно умножена на соответствующие матрицы.

1.5. Получение проекций

Видимым объемом при перспективном преобразовании в OpenGL является усеченная пирамида.

Для задания перспективного преобразования в OpenGL служит процедура:

`glFrustum(GLdouble teft, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)`.

Параметры определяют плоскости, по которым проводится отсечение. Величины `near` и `far` должны быть неотрицательными.

Иногда для задания перспективного преобразования удобнее воспользоваться следующей процедурой из библиотеки утилит OpenGL:

`gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar)`.

Эта процедура создает матрицу для задания симметричного поля зрения и умножает текущую матрицу на неё. Здесь `fovy` – угол зрения камеры в плоскости Oxz , лежащий в диапазоне $[0, 180]$. Параметр `aspect` – отношение ширины области к её высоте, `zNear` и `zFar` – расстояния вдоль отрицательного направления оси Oz , определяющие ближнюю и дальнюю плоскости отсечения.

Существует ещё одна удобная функция для задания перспективного проецирования:

`gluLookAt (GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ,
GLdouble centerX, GLdouble centerY, GLdouble centerZ,
GLdouble upX, GLdouble upY, GLdouble upZ)`.

Вектор $(eyeX, eyeY, eyeZ)$ задаёт положение наблюдателя, вектор $(centerX, centerY, centerZ)$ – направление на центр сцены, а вектор (upX, upY, upZ) – направление вверх.

В случае параллельного проецирования видимым объемом является пря-

моугольный параллелепипед. Для задания параллельного проецирования служит процедура:

```
glOrtho(GLdouble left, GLdouble right, GLdouble bottom,
        GLdouble top, GLdouble near, GLdouble far).
```

Параметры `left` и `right` определяют координаты левой и правой вертикальных плоскостей отсечения, а `bottom` и `top` – нижней и верхней горизонтальных плоскостей.

Следующим шагом в задании проецирования (после выбора параллельного или перспективного преобразования) является задание области в окне, в которую будет помещено получаемое изображение. Для этого служит процедура:

```
glViewport(GLint x, GLint y, GLsizei width, GLsizei height).
```

Здесь `(x, y)` задаёт нижний левый угол прямоугольной области в окне, а `width` и `height` являются её шириной и высотой.

OpenGL содержит стек матриц для каждого из трёх типов преобразований. При этом текущую матрицу можно поместить в стек или извлечь матрицу из стека и сделать её текущей.

Для помещения текущей матрицы в стек служит процедура `glPushMatrix()`, для извлечения матрицы из стека – процедура `glPopMatrix()`.

1.6. Задание моделей закрашивания

Линия или заполненная грань могут быть нарисованы одним цветом (плоское закрашивание, `GL_FLAT`) или путём интерполяции цветов в вершинах (закрашивание Гуро, `GL_SMOOTH`).

Для задания режима закрашивания служит процедура `glShadeModel(GLenum mode)`, где параметр `mode` принимает значение `GL_SMOOTH` или `GL_FLAT`.

1.7. Освещение

OpenGL использует модель освещённости, в которой свет приходит из нескольких источников, каждый из которых может быть включён или выключен. Кроме того, существует еще общее фоновое (*ambient*) освещение.

Для правильного освещения объектов необходимо для каждой грани задать материал, обладающий определенными свойствами. Материал может испускать свой собственный свет, рассеивать падающий свет во всех направлениях (диффузное отражение) или подобно зеркалу отражать свет в определенных направлениях.

Пользователь может определить до восьми источников света и их свойства, такие, как цвет, положение и направление. Для задания этих свойств служит процедура:

```
glLight{i f}[v](GLenum light, GLenum pname, TYPE param),
```

которая задаёт параметры для источника света `light`, принимающего зна-

чения `GL_LIGHT0`, `GL_LIGHT1`, ..., `GL_LIGHT7`. Параметр `pname` определяет характеристику источника света, которая задается последним параметром.

Для использования источников света расчёт освещенности следует разрешить командой `glEnable(GL_LIGHTING)`, а применение соответствующего источника света разрешить (включить) командой `glEnable`, например: `glEnable(GL_LIGHT0)`.

Источник света можно рассматривать как имеющий вполне определенные координаты и светящий во всех направлениях или как направленный источник, находящийся в бесконечно удаленной точке и светящий в заданном направлении (x, y, z).

Если параметр `w` в команде `GL_POSITION` равен нулю, то соответствующий источник света – направленный и светит в направлении (x, y, z). Если же `w` отлично от нуля, то это позиционный источник света, находящийся в точке с координатами ($x/w, y/w, z/w$).

Заданием параметров `GL_SPOT_CUTOFF` и `GL_SPOT_DIRECTION` можно создавать источники света, которые будут иметь коническую направленность. По умолчанию значение параметра `GL_SPOT_CUTOFF` равно 180° , т. е. источник светит во всех направлениях с равной интенсивностью. Параметр `GL_SPOT_CUTOFF` определяет максимальный угол от направления источника, в котором распространяется свет от него. Он может принимать значение 180° (не конический источник) или от 0 до 90° .

Интенсивность источника с расстоянием – убывает (параметры этого убывания задаются при помощи параметров `GL_CONSTANT_ATTENUATION`, `GL_LINEAR_ATTENUATION` и `GL_QUADRATIC_ATTENUATION`). Только собственное свечение материала и глобальная фоновая освещенность с расстоянием не ослабевают.

Глобальное фоновое освещение можно задать при помощи команды `glLightModel{i f} [v] (GL_LIGHT_MODEL_AMBIENT, ambientColor)`.

Местонахождение наблюдателя оказывает влияние на блики на объектах. По умолчанию при расчётах освещённости считается, что наблюдатель находится в бесконечно удалённой точке, т. е. направление источника света на наблюдателя постоянно для любой вершины. Можно включить более реалистичное освещение, когда направление источника света на наблюдателя будет вычисляться для каждой вершины отдельно. Для этого служит команда:

`glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE)`.

Для задания освещения как лицевых, так и нелицевых граней (для нелицевых граней вектор нормали переворачивается) служит следующая команда:

`glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE)`.

Причём существует возможность отдельного задания свойств материала для каждой из сторон.

Свойства материала, из которого сделан объект, задаются при помощи процедуры:

`glMaterial{i f}[v](GLenum face, GLenum pname, TYPE param)`.

Параметр `face` указывает, для какой из сторон грани задается свойство, и

принимает одно из следующих значений: GL_BACK, GL_FRONT_AND_BACK, GL_FRONT.

Параметр pname указывает, какое именно свойство материала задается.

Расчёт освещённости в OpenGL не учитывает затенения одних объектов другими.

1.8. Полупрозрачность

Полупрозрачность объектов устанавливается с использованием α -канала (в RGBA-представлении цвета). Если его значение равно единице – объект непрозрачен. Задавая значения, отличные от единицы, можно смешивать цвет выводимого пикселя с цветом пикселя, уже находящегося в соответствующем месте на экране, создавая тем самым эффект прозрачности.

α -значение характеризует степень поглощения фрагментом проходящего через него света. Так, если у стекла установить значение, равное 0,2, то в результате вывода цвет получившегося фрагмента будет на 20 % состоять из собственного цвета стекла и на 80 % – из цвета фрагмента под ним.

Для использования α -канала необходимо сначала разрешить режим прозрачности и смешения цветов командой glEnable(GL_BLEND).

В процессе смешения цветов цветовые компоненты выводимого фрагмента $R_s G_s B_s A_s$ смешиваются с цветовыми компонентами уже выведенного фрагмента $R_d G_d B_d A_d$ по формуле

$$(R_s S_r + R_d D_r, G_s S_g + G_d D_g, B_s S_b + B_d D_b, A_s S_a + A_d D_a),$$

где (S_r, S_g, S_b, S_a) и (D_r, D_g, D_b, D_a) – коэффициенты смешения.

Для задания связи этих коэффициентов с α -значениями используется следующая функция:

glBlendFunc(GLenum sfactor, GLenum dfactor).

Здесь параметр sfactor задаёт то, как нужно вычислять коэффициенты (S_r, S_g, S_b, S_a) , а параметр dfactor – коэффициенты (D_r, D_g, D_b, D_a) .

1.9. 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:

glTexImage2D(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 (`mir-mapping`). Для этого необходимо иметь текстуры всех промежуточных размеров, являющихся степенями двух, вплоть до 1×1 , 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_FILTER` 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_MODELVIEW).
```

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 доста-

точно 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.
```

2. ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ ЗАДАНИЙ

Лабораторные работы необходимо выполнять с использованием языка Visual C++. Или по согласованию с преподавателем на Builder C++, Delphi.

На первые работы приведены листинги программ. Последующие работы планируется выполнять самостоятельно.

При выполнении работ необходимо оформлять отчет, который должен быть представлен в печатном и электронном виде. Далее студент проходит собеседование.

Печатный отчет студента по заданию должен содержать:

1. Титульный лист.
2. Лист с формулировкой задания.
3. Листы с текстом программы (листингом). В котором сложные или ключевые моменты должны быть прокомментированы.
4. В отчет необходимо вставлять рисунки, содержащие результаты выполнения программ.
5. Заключительный лист может содержать рекомендации по улучшению программы.

Электронный отчет может находиться на любом носителе информации (дискета, компакт-диск и др.) и должен включать отчет, исполняемый файл программы и ее исходный текст.

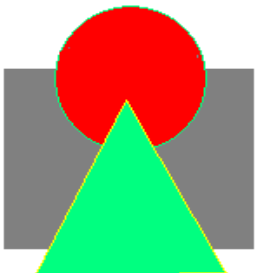
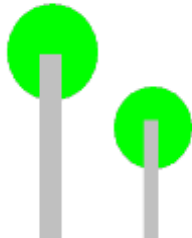
3. УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЗАДАНИЙ

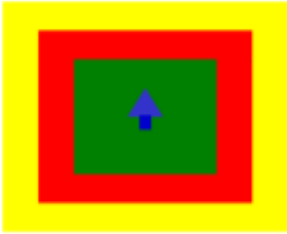

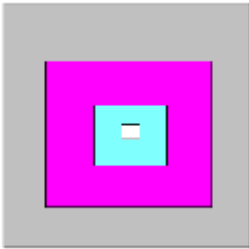
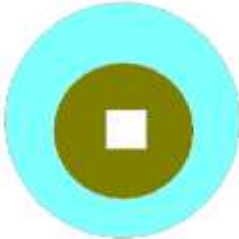

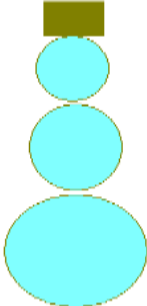


3.1. Лабораторная работа № 1. «Моделирование 2D объектов с использованием Visual C++ и OpenGL»

Задание

1. Изучить теоретические основы OpenGL (пп. 1.1–1.3), прил. 1 и 2.
2. Построить прямоугольник. Листинг программы приведен в прил. 1. Изменить параметры освещенности, цвета объекта и фона, размера.
3. В соответствии со своим вариантом (табл. 2) построить 2D объекты.
4. Составить и защитить отчет по работе.

Таблица 2. Исходные задания к лабораторной работе № 1

Номер варианта	Задание	Номер варианта	Задание
1		6	

Номер варианта	Задание	Номер варианта	Задание
2		7	
3		8	
4		9	
5		10	

**3.2. Лабораторная работа № 2. «Моделирование
3D объектов с использованием Visual C++ и OpenGL.
Работа с цветом и светом»**

Задание

1. Ознакомиться с теоретическими основами (пп. 1.3.4, 1.4–1.7) и прил. 3–8.
2. Построить каркасные и сплошные модели фигур и поверхностей, (прил. 3). Изменить цвет объектов.
3. Выполнить задание в соответствии со своим вариантом (табл. 3) (по согласованию с преподавателем задание может быть изменено). Ввести различное освещение объектов.

Таблица 3. Исходные задания к лабораторной работе № 2

Номер варианта	Задание	Номер варианта	Задание
1	Собака	6	Ель
2	Олимпийские кольца	7	Марсианин
3	Цветы	8	Медведь
4	Ваза	9	Снеговик
5	Стол с чайником	10	Пингвин

4. Составить и защитить отчет по работе.

3.3. Лабораторная работа № 3. «Текстурирование»

Задание

1. Ознакомиться с теоретическими основами (п.п. 1.8, 1.9) и прил. 5, 9.
2. На объекты, реализованные в лабораторной работе № 2 нанести текстуры (по согласованию с преподавателем задание может быть изменено).
3. Составить и защитить отчет по работе.

3.4. Лабораторная работа № 4. «Моделирование 3D анимации»

Задание

1. Ознакомиться с прил. 9–12.
2. Анимировать результаты выполненной лабораторной работы № 3 (по согласованию с преподавателем задание может быть изменено).
3. Составить и защитить отчет по работе.

3.5 Лабораторная работа № 5. «Индивидуальное творческое задание»

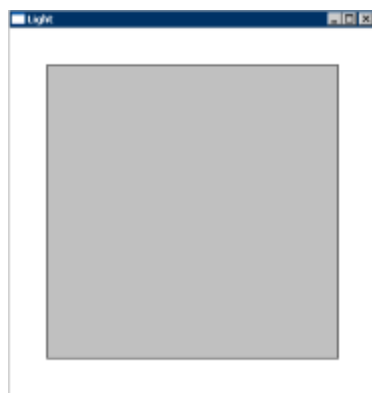
Задание

1. Ознакомиться с прил. 1–12.
2. По согласованию с преподавателем определиться с заданием, которое должно отвечать следующим требованиям:
 - содержать трехмерные объекты, связанные по смыслу;
 - включать элементы анимации, текстуры, освещения.
3. Составить и защитить отчет по работе.

ПРИЛОЖЕНИЕ 1

Построение квадрата

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, -5,5, 2,12);
    gluLookAt( 0,0,5, 0,0,0, 0,1,0 );
    glMatrixMode( GL_MODELVIEW );
}
void CALLBACK display(void)
{
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glColor3d(1,1,1);
    glBegin(GL_QUADS);
        glVertex3d(-4,-4,0);
        glVertex3d(-4, 4,0);
        glVertex3d( 4, 4,0);
        glVertex3d( 4,-4,0);
    glEnd();
    /*
    glColor3d(1,0,0);
    auxSolidSphere(0.1);

    */
    auxSwapBuffers();
}
```

```

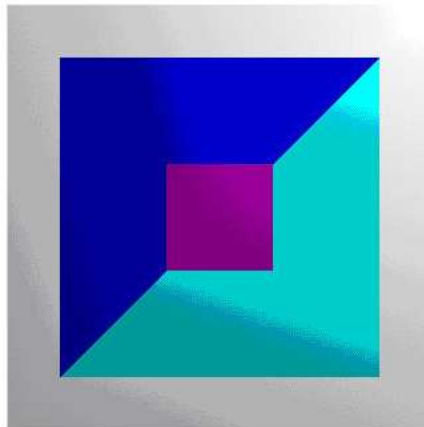
void main()
{
float pos[4] = {3,3,3,0.5};
float dir[3] = {-1,-1,-1};
    GLfloat mat_specular[] = {1,1,1,1};
    auxInitPosition( 50, 10, 400, 400);
    auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
    auxInitWindow( "Light" );
    auxIdleFunc(display);
    auxReshapeFunc(resize);
    glEnable(GL_DEPTH_TEST);
    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_SPECULAR, mat_specular);
    glMaterialf(GL_FRONT, GL_SHININESS, 128.0);
    glLighti(GL_LIGHT0, GL_SPOT_EXPONENT, 0);
    glLighti(GL_LIGHT0, GL_SPOT_CUTOFF, 90);
auxMainLoop(display);
}

```

ПРИЛОЖЕНИЕ 2

Построение 2D объектов

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, -5,5, 2,12);
    gluLookAt( 0,0,5, 0,0,0, 0,1,0 );
    glMatrixMode( GL_MODELVIEW );
}
void CALLBACK display(void)
{
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3d(1,1,1);
    glBegin(GL_QUADS);
        glVertex3d(-4,-4,0);
        glVertex3d(-4, 4,0);
        glVertex3d( 4, 4,0);
        glVertex3d( 4,-4,0);
    glEnd();
    glColor3d(0,1,1);
    glBegin(GL_QUADS);
        glVertex3d(-3,-3,1);
        glVertex3d(-3, 3,1);
        glVertex3d( 3, 3,1);
        glVertex3d( 3,-3,1);
    glEnd();
    glColor3d(0,0,1);
    glBegin(GL_TRIANGLES);
        glVertex3d(-3,-3,2);
        glVertex3d(-3, 3,2);
        glVertex3d( 3, 3,2);
        //glVertex3d( 3,-3,1);
    glEnd();
    glColor3d(1,0,1);
    glBegin(GL_QUADS);
        glVertex3d(-1,-1,3);
        glVertex3d(-1, 1,3);
        glVertex3d( 1, 1,3);
        glVertex3d( 1,-1,3);
    glEnd();
    /*
    glColor3d(1,0,0);
    auxSolidSphere(0.1);
    */
    auxSwapBuffers();
}
void main()
{
    float pos[4] = {3,3,3,0.5};
    float dir[3] = {-1,-1,-1};
    GLfloat mat_specular[] = {1,1,1,1};
    auxInitPosition( 50, 10, 400, 400);
    auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
}

```

```

auxInitWindow( "Light" );
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_SPECULAR, mat_specular);
glMaterialf(GL_FRONT, GL_SHININESS, 128.0);
glLighti(GL_LIGHT0, GL_SPOT_EXPONENT, 0);
glLighti(GL_LIGHT0, GL_SPOT_CUTOFF, 90);
auxMainLoop(display);
}

```

ПРИЛОЖЕНИЕ 3

Построение 3D объектов

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, -5,5, 2,12);
    gluLookAt( 0,0,5, 0,0,0, 0,1,0 );
    glMatrixMode( GL_MODELVIEW );
}

```

```

void CALLBACK display(void)
{
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    glPushMatrix();
    glTranslated(0.5,4,0);
    glColor3d(0,0,1);
    auxSolidCube(1);      // куб
    glTranslated(0,-2,0);
    glColor3d(0,1,0);
    auxSolidBox(1,0.75,0.5); // коробка
    glTranslated(0,-2,0);
    glColor3d(0,1,1);
    auxSolidTorus(0.2,0.5); // тор
    glTranslated(0,-2,0);
    glColor3d(1,0,0);
    auxSolidCylinder(0.5,1); // цилиндр
    glTranslated(0,-2,0);
    glColor3d(0,1,0);
    auxSolidCone(1,1);    // конус
    glTranslated(2,8,0);
    glColor3d(1,0,1);
    auxSolidIcosahedron(1); // многогранники
    glTranslated(0,-2,0);
    glColor3d(1,1,1);
    auxSolidOctahedron(1);
    glTranslated(0,-2,0);
    glColor3d(0,1,1);
    auxSolidTeapot(0.7);  // чайник
    glTranslated(0,-2,0);
    glColor3d(0,1,0);
    auxSolidTetrahedron(1);
    glTranslated(0,-2,0);
    glColor3d(1,1,0);
    auxSolidDodecahedron(1);
    glTranslated(-6,8,0);
    glColor3d(0,0,1);
    auxWireCube(1);      // каркасная модель куба
    glTranslated(0,-2,0);
    glColor3d(0,1,0);
    auxWireBox(1,0.75,0.5); // каркасная модель параллелограмма
    glTranslated(0,-2,0);
    glColor3d(0,1,1);
    auxWireTorus(0.2,0.5); // каркасная модель тора
    glTranslated(0,-2,0);
    glColor3d(1,0,0);
    auxWireCylinder(0.5,1); // каркасная модель цилиндра
    glTranslated(0,-2,0);
    glColor3d(0,1,0);
    auxWireCone(1,1);    // каркасная модель конуса
    glTranslated(2,8,0);
    glColor3d(1,0,1);
}

```



```

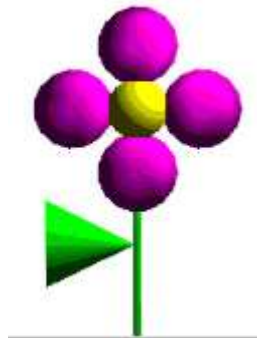
auxWireIcosahedron(1); // каркасные модели многогранников
glTranslated(0,-2,0);
    glColor3d(1,1,1);
auxWireOctahedron(1);
glTranslated(0,-2,0);
    glColor3d(0,1,1);
auxWireTeapot(0.7); // каркасная модель чайника
glTranslated(0,-2,0);
    glColor3d(0,1,0);
auxWireTetrahedron(1);
glTranslated(0,-2,0);
    glColor3d(1,1,0);
auxWireDodecahedron(1);
glPopMatrix();
auxSwapBuffers();
}
void main()
{
    float pos[4] = {3,3,3,1};
    float dir[3] = {-1,-1,-1};
    auxInitPosition( 50, 10, 400, 400);
    auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
    auxInitWindow( "Shapes" );
    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);
    auxMainLoop(display);
}

```

ПРИЛОЖЕНИЕ 4

Цветок

Результат выполнения программы



Текст программы

```

#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/aux.h>
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)
{
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    glPushMatrix();
    glColor3d(0,1,0);
    auxSolidCylinder(0.1,10);
    glColor3d(1,1,0);
    auxSolidSphere(0.7);
    glColor3d(1,0,1);
    glTranslated(0,-1.4,0);
    auxSolidTorus(0.8,0.1);
    glTranslated(0,2.8,0);
    auxSolidTorus(0.8,0.1);
    glTranslated(1.4,-1.4,0);
    auxSolidTorus(0.8,0.1);
    glTranslated(-2.8,0,0);
    auxSolidTorus(0.8,0.1);
    glTranslated(-0.6,-3,0);
    glColor3d(0,1,0);
    glRotated(90,0,1,0);
    auxSolidCone(1,2);
    glPopMatrix();
    auxSwapBuffers();
}
void main()
{
    float pos[4] = {3,3,3,1};
    float dir[3] = {-1,-1,-1};
    auxInitPosition( 50, 10, 400, 400);
    auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
    auxInitWindow( "Shapes" );
    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);
auxMainLoop(display);
}

```

ПРИЛОЖЕНИЕ 5

Снеговик с текстурой

Результат выполнения программы



Текст программы

```

#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>
GLuint texture[2];
GLvoid LoadGLTextures()
{
    AUX_RGBImageRec *texture1;
    texture1 = auxDIBImageLoad("c:/Data/Tela1.bmp");
// Создание текстуры
    glGenTextures(1, &texture[0]);
    glBindTexture(GL_TEXTURE_2D, texture[0]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, texture1->sizeX, texture1->sizeY, 0,
        GL_RGB, GL_UNSIGNED_BYTE, texture1->data);
}
void CALLBACK resize(int width, int height)
{
    LoadGLTextures();           // Загрузка текстур
}

```

```

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 *q;    // Квадратичный объект для рисования сферы
    q = gluNewQuadric(); // Создать квадратичный объект
    // тип генерируемых нормалей для него – «сглаженные»
    gluQuadricNormals(q, GL_SMOOTH);
    // Включить текстурные координаты для объекта
    gluQuadricTexture(q, GL_TRUE);
    // Настройка сферического наложения
    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
    // Настройка отображения сферы
    glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    glBindTexture(GL_TEXTURE_2D, texture[0]);
    glPushMatrix();
    glEnable(GL_TEXTURE_2D);           // Разрешение наложение текстуры
    gluSphere(q,1.5,32,16); // Рисование первой сферы
    glDisable(GL_TEXTURE_2D);         // Разрешение наложение текстуры
    glTranslated(0,-3,0);
    auxSolidSphere(2.0);
    glTranslated(0,3,0);
    auxSolidSphere(1.5);
    glTranslated(0,2,0);
    auxSolidSphere(1);
    glColor3d(0,0,1);    //Левый глаз
    glTranslated(-0.3,0.3,1);
    auxSolidSphere(0.1);
    glColor3d(0,0,0);    //Правый глаз
    glTranslated(0.6,0,0);
    auxSolidSphere(0.2);
    glTranslated(-0.3,-0.3,0.1); //Бандана
    glColor3d(1,0,0);
    auxSolidSphere(0.3);
    glTranslated(0,0.50,-1);    //Нос
    glColor3d(0,1,0);
    auxSolidSphere(0.75);
    //glTranslated(0,0.75,-1);
    //glColor3d(0.75,0.75,0.75);
    //auxSolidSphere(0.75);
    glBegin(GL_LINES);        //Повязка
    glColor3d(0,0,0);
    glLineWidth(20);
    glVertex3d(-0.15,0.25,0.69);
    glVertex3d(0.94,-0.8,1);

```

```

glEnd();
glBegin(GL_TRIANGLES);      //Завязка1
    glColor3d(0,1,0);
    glVertex3d(0.75,0.2,1);
    glVertex3d(1,0.8,1);
    glVertex3d(1.5,0.6,1);
glEnd();
glBegin(GL_TRIANGLES);      //Завязка2
    glColor3d(0,1,0);
    glVertex3d(0.75,0.2,1);
    glVertex3d(1.2,-0.2,1);
    glVertex3d(1.5,0.2,1);
glEnd();
glColor3d(0,1,0);           //Узелок
glTranslated(0.78,0.21,1);
auxSolidSphere(0.1);
glColor3d(0.75,0.75,0.75); //Рука правая
glTranslated(1,-2,0);
auxSolidSphere(0.55);
glColor3d(0.75,0.75,0.75); //Рука левая
glTranslated(-3.5,-0.12,0);
auxSolidSphere(0.55);
glPopMatrix();
auxSwapBuffers();
}
void main()
{
    float pos[4] = {3,3,3,1};
    float dir[3] = {-1,-1,-1};
    auxInitPosition( 50, 10, 400, 400);
    auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
    auxInitWindow( "Snowman" );
    auxIdleFunc(display);
    auxReshapeFunc(resize);
    glEnable(GL_ALPHA_TEST);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_COLOR_MATERIAL);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glLightfv(GL_LIGHT0, GL_POSITION, pos);
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir);
    auxMainLoop(display);
}

```

ПРИЛОЖЕНИЕ 6

Новогодняя ель

Результат выполнения программы



Текст программы

```
#include "stdafx.h"
#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, -5,5, 2,12);
    gluLookAt( 0,0,5, 0,0,0, 0,1,0 );
    glMatrixMode( GL_MODELVIEW );
}
void CALLBACK display(void)
{
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    glPushMatrix();

    glColor3d(0,1,0);
    glTranslated(0,-4,0);
    glRotated(90,0,1,0);
    glRotated(270,1,0,0);
    auxSolidCone(4,4);
    glTranslated(0.5,0,-0.5);
    glColor3d(1,0.4,0.2);
    auxSolidCylinder(0.5,2);
    glTranslated(0,0,0.5);
    glColor3d(0,1,0);
    glTranslated(0,0,2);
```

```

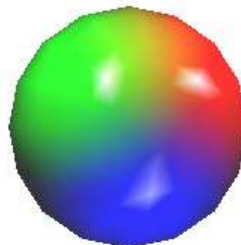
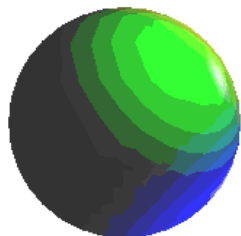
    auxSolidCone(3.5,3.5);
    glTranslated(0,0,2);
    auxSolidCone(3,3);
    glTranslated(0,0,2);
    auxSolidCone(2,2);
    glTranslated(-3.4,-1,-2);
    glColor3d(1,0,0);
    auxSolidSphere(0.5);
    glTranslated(-0.12,-1,-2);
    glColor3d(0,1,1);
    auxSolidSphere(0.7);
    glTranslated(-0.23,3,3);
    glColor3d(1,1,0);
    auxSolidSphere(0.5);
    glTranslated(0,0,-3);
    glColor3d(1,0,1);
    auxSolidSphere(0.5);
    glPopMatrix();
    auxSwapBuffers();
}
int main(int argc, char* argv[])
{
    float pos[4] = {3,3,0.24,1};
    float dir[3] = {1,1,1};
    auxInitPosition( 50, 10, 400, 400);
    auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
    auxInitWindow( "С Новым Годом!!!" );
    auxIdleFunc(display);
    auxReshapeFunc(resize);
    glEnable(GL_ALPHA_TEST);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_COLOR_MATERIAL);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glLightfv(GL_LIGHT0, GL_POSITION, pos);
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir);
    auxMainLoop(display);
    return 0;
}

```

ПРИЛОЖЕНИЕ 7

Работа с источниками освещения

Результат выполнения программы



Текст программы

```
#include <windows.h>
#include <math.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/GLaux.h>
float color1[4] = { 1,0,0,1 };
float color2[4] = { 0,1,0,1 };
float color3[4] = { 0,0,1,1 };
int time=0;
void CALLBACK resize(int width,int height)
{
    glViewport(0,0,width,height);
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glOrtho(-5,5, -5,5, -5,5);
    gluLookAt( 4*cos(time),0,4*sin(time), 0,0,0, 0,1,0 );
    glMatrixMode( GL_MODELVIEW );
    time++;
}
void CALLBACK display(void)
{
    float l1,l2,l3;
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    glColor3d(1,1,1);
    auxSolidSphere(2);
    //glRotated(2,1,0,1);
    auxSwapBuffers();
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glOrtho(-5,5, -5,5, -5,5);
    gluLookAt( 4*cos(time/10),0,4*sin(time/10), 0,0,0, 0,1,0 );
    glMatrixMode( GL_MODELVIEW );
    time++;
}
```



```

void main()
{
float pos[4] = { 3,3,3,1 };
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( "Glaux Template" );
    auxIdleFunc(display);
    auxReshapeFunc(resize);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_COLOR_MATERIAL);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT3);
    glEnable(GL_LIGHT5);
    glEnable(GL_LIGHT6);
    glLightfv(GL_LIGHT3, GL_SPECULAR, sp);
    glLightfv(GL_LIGHT5, GL_SPECULAR, sp);
    glLightfv(GL_LIGHT6, GL_SPECULAR, sp);
    //color[1]=color[2]=0;
    glLightfv(GL_LIGHT3, GL_DIFFUSE, color1);
    //color[0]=0;   color[1]=1;
    glLightfv(GL_LIGHT5, GL_DIFFUSE, color2);
    //color[1]=0;   color[2]=1;
    glLightfv(GL_LIGHT6, GL_DIFFUSE, color3);
    glLightfv(GL_LIGHT3, GL_POSITION, pos);
    pos[0] = -3;
    glLightfv(GL_LIGHT5, GL_POSITION, pos);
    pos[0]=0;pos[1]=-3;
    glLightfv(GL_LIGHT6, GL_POSITION, pos);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialf(GL_FRONT, GL_SHININESS, 128.0);
    auxMainLoop(display);
}

```

ПРИЛОЖЕНИЕ 8

Текстурирование и анимация 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);
}

```

ПРИЛОЖЕНИЕ 9

Вентилятор 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);
}

```

ПРИЛОЖЕНИЕ 10

Анимация – чайник

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 мы-
ши
void CALLBACK mouse_rightbtn(AUX_EVENTREC* event);// обработчик правой клавиши
мышы
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();
}

```

```

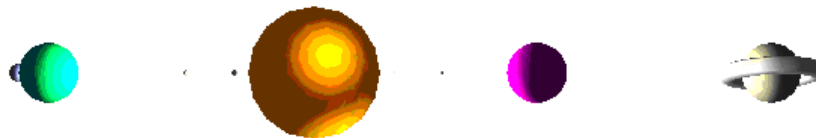
switch (ax)
{
case 0: glRotatef(angle,1,0,0); break;
case 1: glRotatef(angle,0,1,0); break;
case 2: glRotatef(angle,0,0,1); break;
default : glRotatef(angle,0.5,0.5,0);
}
auxWireTeapot(2);
glPopMatrix();
if (flag)
{
angle+=d;
if (angle>360) angle-=360;
}
//копирование содержимого буфера кадра на экран
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;
}

```


ПРИЛОЖЕНИЕ 11

Модель солнечной системы

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);
void CALLBACK display(void);
void main()
{
    auxInitPosition( 50, 10, 800, 400);
    auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
    auxInitWindow( "Модель солнечной системы" );
    auxReshapeFunc(resize);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_ALPHA_TEST);
    glEnable(GL_COLOR_MATERIAL);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    float pos[4] = {0,0,0,5};
    float dir[3] = {0,-1,-1};
    float amb0[4]={1,1,0,5};
    glLightfv(GL_LIGHT0, GL_POSITION, pos);
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir);
    glLightfv(GL_LIGHT0, GL_DIFFUSE,amb0);
    float amb[4]={0,0,0,1};
    float def[4]={1,1,1,1};
    float spe[4]={1,1,1,1};
    float pos1[4] = {10,10,10,5};
    float dir1[3] = {0,1,0};
    glLightfv(GL_LIGHT1, GL_AMBIENT,amb);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, def);
    glLightfv(GL_LIGHT1, GL_SPECULAR, spe);
    glLightfv(GL_LIGHT1, GL_POSITION, pos1);
    glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, dir1);
    glEnable(GL_LIGHT1);
    float pos2[4] = {10,-10,0,7};
```

```

        float dir2[3] = {0,1,0};
        glLightfv(GL_LIGHT2, GL_AMBIENT, amb);
        glLightfv(GL_LIGHT2, GL_DIFFUSE, def);
        glLightfv(GL_LIGHT2, GL_SPECULAR, spe);
        glLightfv(GL_LIGHT2, GL_POSITION, pos2);
        glLightfv(GL_LIGHT2, GL_SPOT_DIRECTION, dir2);
        glEnable(GL_LIGHT2);
    auxIdleFunc(display);
    auxMainLoop(display);
}
void CALLBACK resize(int width,int height)
{
    glViewport(0,0,width,height);
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glOrtho(-20,20, -10,10,-50,50);
    gluLookAt( 3,0,5, 0,0,0, 0,1,0 );
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
}
void CALLBACK display(void)
{
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    static float angle=-360;
    static float ang=-1;
    if (angle<0) glColor3f(1,0,0);else    if (angle>0) glColor3f(0.5,0.5,1);
    glRotated(1,0,1,0);
    glPushMatrix();
    glColor3d(2,1,0);
    glTranslated(0,1.2,0);
    auxSolidSphere(2.2);
    glRotated(2,0,1,0);
    glPushMatrix();
    glColor3d(0.5,0.5,0.5);
    glTranslated(16,0,0);
    auxSolidSphere(1);
    glRotated(80,1,0,0);
    glTranslated(0,0,0);
    glColor3d(1,1,1);
    auxSolidTorus(0.3,1.5);
    glPopMatrix();
    glPushMatrix();
    glRotated(ang+angle,0,1,0);
    glColor3d(1.5,0,0);
    glTranslated(-3,0,0);
    auxSolidSphere(0.5);
    glPopMatrix();
    glPushMatrix();
    glRotated(angle,0,1,0);
    glColor3d(1,0,1);
    glTranslated(0,0,-8);

```

```

auxSolidSphere(1);
glPopMatrix();
glPushMatrix();
glRotated(ang,0,1,0);
glColor3d(0,1,0.5);
glTranslated(0,0,-10);
auxSolidSphere(1);
glRotated(angle*5,0,1,0);
glColor3d(0.5,0.5,1);
glTranslated(1.5,0,0);
auxSolidSphere(0.3);
glPopMatrix();
glPushMatrix();
glColor3d(1,1,1);
glRotated((ang+angle)*10,0,1,0);
glPushMatrix();
glTranslated(5,0,0);
auxSolidSphere(0.1);
glTranslated(-10,0,0);
auxSolidSphere(0.05);
glPopMatrix();
glPushMatrix();
glTranslated(0,0,5);
auxSolidSphere(0.1);
glTranslated(0,0,-10);
auxSolidSphere(0.05);
glPopMatrix();
glPopMatrix();
glPopMatrix();
angle+=5;
if (angle>3600)angle=0;
ang+=1;
if (ang>720)ang=0;
glFlush();
auxSwapBuffers();
}

```

ПРИЛОЖЕНИЕ 12

Модель самолета

Результат выполнения программы

Вращающийся самолет с двигающимися лопастями пропеллера.

Текст программы

1. Листинг выполненной программы:

```

#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>

```

```

#include <GL/glaux.h>
#include <math.h>
int i;
float color1[4] = {1,0,0,1};
float color2[4] = {0,1,0,1};
float color3[4] = {0,0,1,1};
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(-13,13, -13,12, -5,13);
    gluLookAt( 0,0,5, 0,0,0, 0,1,0 );
    glMatrixMode( GL_MODELVIEW );
}
void CALLBACK display(void)
{
    static int angle=0;
    const speed=5000;
    glClearColor(0.5,0.5,0.5,1);
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    photo_image = auxDIBImageLoad("4.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);
    glPushMatrix();
    // glEnable(GL_TEXTURE_2D);
    // glEnable(GL_TEXTURE_GEN_S);
    // glEnable(GL_TEXTURE_GEN_T);
    //glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP );
    //glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
    glPushMatrix();
    glRotated(90,0,0,1);
    glTranslated(0,5,0);
    glColor3d(1,1,1);
    auxSolidCylinder(1,10);
    glPopMatrix();          //корпус
    glPushMatrix();
    glTranslated(4.2,0,0);
    glColor3d(1,1,1);
    auxSolidSphere(1);
    glColor3d(1,1,1);
    glRotated(90,0,0,1);
    glTranslated(-0.5,-0.2,0);
    auxSolidCylinder(0.5,2);
    glColor3d(1,1,1);
    glRotated(90,1,0,0);
    glTranslated(0,0,1);
}

```

```

auxSolidCone(0.5,0.5);      //нос
glColor3d(1,1,1);
glTranslated(0.5,0,-11.5);
auxSolidSphere(1);          //задняя часть корпуса
glColor3d(1,1,1);
glRotated(90,0,0,1);
glTranslated(0,-1,0);
auxSolidCone(0.5,5);
glColor3d(1,1,1);
glRotated(90,1,0,0);
glTranslated(0,-0.08,-1);
auxSolidCone(1,3);
glColor3d(1,1,1);
glRotated(90,0,1,0);
glTranslated(-1,0,0);
auxSolidBox(0.3,0.5,4);
glPopMatrix();              //хвост
glPushMatrix();
glRotated(90,0,0,1);
auxSolidBox(0.4,2,12);      //крылья
glTranslated(-0.5,0,3.5);
auxSolidCylinder(0.5,2.2);
glTranslated(0,0,-7);
auxSolidCylinder(0.5,2.2);
glTranslated(0,-1.1,0);
auxSolidSphere(0.5);
glTranslated(0,2.2,0);
auxSolidSphere(0.5);
glTranslated(0,-2.2,7);
auxSolidSphere(0.5);
glTranslated(0,2.2,0);
auxSolidSphere(0.5);
glPopMatrix();
glPushMatrix();
glRotated(90,1,0,0);
glTranslated(0,0.8,1);
auxSolidBox(0.2,0.2,1);
glTranslated(0,-1.6,0);
auxSolidBox(0.2,0.2,1);
glRotated(90,1,0,0);
glColor3d(0,0,0);
glTranslated(0,0.55,0);
auxSolidTorus(0.15,0.25);
glTranslated(0,0,-1.6);
auxSolidTorus(0.15,0.25); //шасси
glRotated(90,0,0,1);
glTranslated(-1,-1.2,-2.8);
    glPushMatrix();
        glRotated(-angle*speed,0,1,0);
        glTranslated(0,0,0.8);
        glRotated(30,0,0,1);

```

```

        glColor3d(0,0,0);
        auxSolidBox(0.1,0.5,1);
glPopMatrix();
    glPushMatrix();
        glRotated(-angle*speed+120,0,1,0);
        glTranslated(0,0,0.8);
        glRotated(30,0,0,1);
        glColor3d(0,0,0);
        auxSolidBox(0.1,0.5,1);
    glPopMatrix();
    glPushMatrix();
        glRotated(-angle*speed-120,0,1,0);
        glTranslated(0,0,0.8);
        glRotated(30,0,0,1);
        glColor3d(0,0,0);
        auxSolidBox(0.1,0.5,1);
    glPopMatrix();
glTranslated(0,0,7);
    glPushMatrix();
        glRotated(-angle*speed,0,1,0);
        glTranslated(0,0,0.8);
        glColor3d(0,0,0);
        glRotated(30,0,0,1);
        auxSolidBox(0.1,0.5,1);
    glPopMatrix();
    glPushMatrix();
        glRotated(-angle*speed+120,0,1,0);
        glTranslated(0,0,0.8);
        glRotated(30,0,0,1);
        glColor3d(0,0,0);
        auxSolidBox(0.1,0.5,1);
    glPopMatrix();
    glPushMatrix();
        glRotated(-angle*speed-120,0,1,0);
        glTranslated(0,0,0.8);
        glRotated(30,0,0,1);
        glColor3d(0,0,0);
        auxSolidBox(0.1,0.5,1);
    glPopMatrix(); //propelleri
// glDisable(GL_TEXTURE_GEN_T);
// glDisable(GL_TEXTURE_2D);
// glDisable(GL_TEXTURE_GEN_S);
glPopMatrix();
glRotated(2,0,90,0); //vraschenie
    angle++;
    auxSwapBuffers();
}
void main()
{
float pos[4] = {5,5,5,1};
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( "Samolet" );
auxIdleFunc(display);
auxReshapeFunc(resize);
glEnable(GL_DEPTH_TEST);
glEnable(GL_COLOR_MATERIAL);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT3);
    glEnable(GL_LIGHT5);
    glEnable(GL_LIGHT0);
glLightfv(GL_LIGHT3, GL_SPECULAR, sp);
glLightfv(GL_LIGHT5, GL_SPECULAR, sp);
glLightfv(GL_LIGHT0, GL_SPECULAR, sp);
glLightfv(GL_LIGHT3, GL_DIFFUSE, color1);
glLightfv(GL_LIGHT5, GL_DIFFUSE, color2);
glLightfv(GL_LIGHT0, GL_DIFFUSE, color3);
    glLightfv(GL_LIGHT3, GL_POSITION, pos);
    pos[0] = -5; pos[1]=5;
glLightfv(GL_LIGHT5, GL_POSITION, pos);
    pos[0]=5;pos[1]=-5;
glLightfv(GL_LIGHT0, GL_POSITION, pos);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialf(GL_FRONT, GL_SHININESS, 128.0);
auxMainLoop(display);
}

```

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ву М. OpenGL. Руководство по программированию / М. Ву [и др.]. 4-е изд. – СПб. : Питер, 2006 – 624 с.
2. Дэвис Стефан. С++ «для чайников» / Стефан Дэвис. – М. : Вильямс, 2005. – 384 с.
3. Черносвитов А. Visual C++7: Учебный курс / А. Черносвитов. – СПб. : Питер 2002. – 528 с.

ОГЛАВЛЕНИЕ

Общие положения.....	3
1. Основы программирования 3D графики.....	3
1.1. Возможности OpenGL.....	3
1.2. Основные типы данных OpenGL.....	4
1.3. Рисование геометрических объектов.....	5
1.3.1. Работа с буферами и задание цвета объектов.....	5
1.3.2. Задание геометрических примитивов.....	6
1.3.3. Рисование точек, линий и многоугольников.....	7
1.3.4. Трехмерные фигуры.....	10
1.4. Преобразование объектов в пространстве.....	10
1.5. Получение проекций.....	11
1.6. Задание моделей закрашивания.....	12
1.7. Освещение.....	12
1.8. Полупрозрачность.....	14
1.9. Наложение текстуры.....	14
2. Требования к выполнению заданий.....	17
3. Указания к выполнению заданий.....	18
3.1. Лабораторная работа №1. «Моделирование 2D объектов с использованием Visual C++ и OpenGL».	18
3.2. Лабораторная работа №2. «Моделирование 3D объектов с использованием Visual C++ и OpenGL. Работа с цветом и светом».....	19
3.3. Лабораторная работа №3. «Текстурирование».....	20
3.4. Лабораторная работа №4. «Моделирование 3D анимации»....	20
3.5. Лабораторная работа №5. «Индивидуальное творческое задание».....	20
Приложение 1. Построение квадрата.....	21
Приложение 2. Построение 2D объектов.....	22
Приложение 3. Построение 3D объектов.....	24
Приложение 4. Цветок.....	26
Приложение 5. Снеговик с текстурой.....	28
Приложение 6. Новогодняя ель.....	31
Приложение 7. Работа с источниками освещения.....	33
Приложение 8. Текстурирование и анимация снеговика.....	34
Приложение 9. Вентилятор с вращающимися лопастями.....	37
Приложение 10. Анимация чайник.....	39
Приложение 11. Модель солнечной системы.....	42
Приложение 12. Модель самолета.....	44
Библиографический список.....	48