Компьютерная графика

лекция

OpenGL : основы

«Язык» OpenGL
Введение

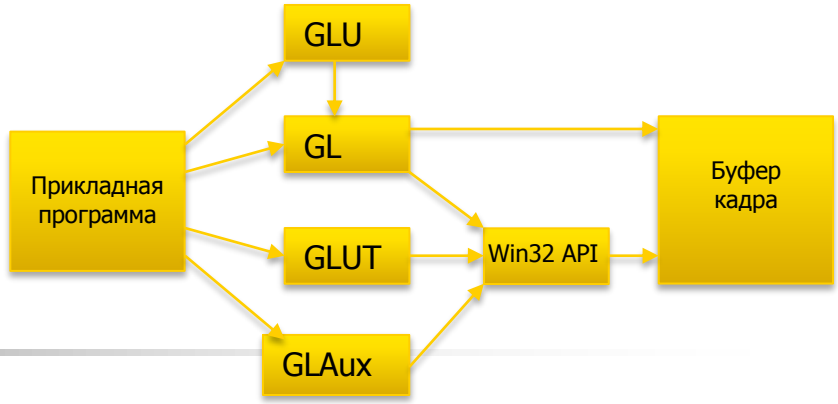
- OpenGL (Open Graphics Library) – универсальный программный интерфейс (API – Application Program Interface) для разработки приложений 2D и 3D графики под любые операционные системы и любое аппаратное обеспечение.

- Стандарт разработан в 1992 году и продвигается компанией Silicon Graphics (SGI). Состоит из ~250 базовых функций в нескольких библиотеках, обычно называемых OpenGL, GLU, GLAUX, GLUT.

- Наиболее востребованное применение: интерактивная реалистичная визуализация трёхмерных моделей

OpenGL

Основные качества



- Совместимость (новые возможности и расширения дополняют старые версии языка, не отменяя их)
- Переносимость базовых функций (одинаковая система команд на различных ОС с одинаковым графическим результатом)
  - Отсутствие привязки к подсистеме формирования графического окна и взаимодействия с пользователем (поддержка только функций рисования)
- Высокая скорость рисования (все современные видеокарты с поддержкой 3D графики аппаратно реализуют элементы интерфейса OpenGL)
- Широкая известность (OpenGL является стандартом отображения 3D графики)

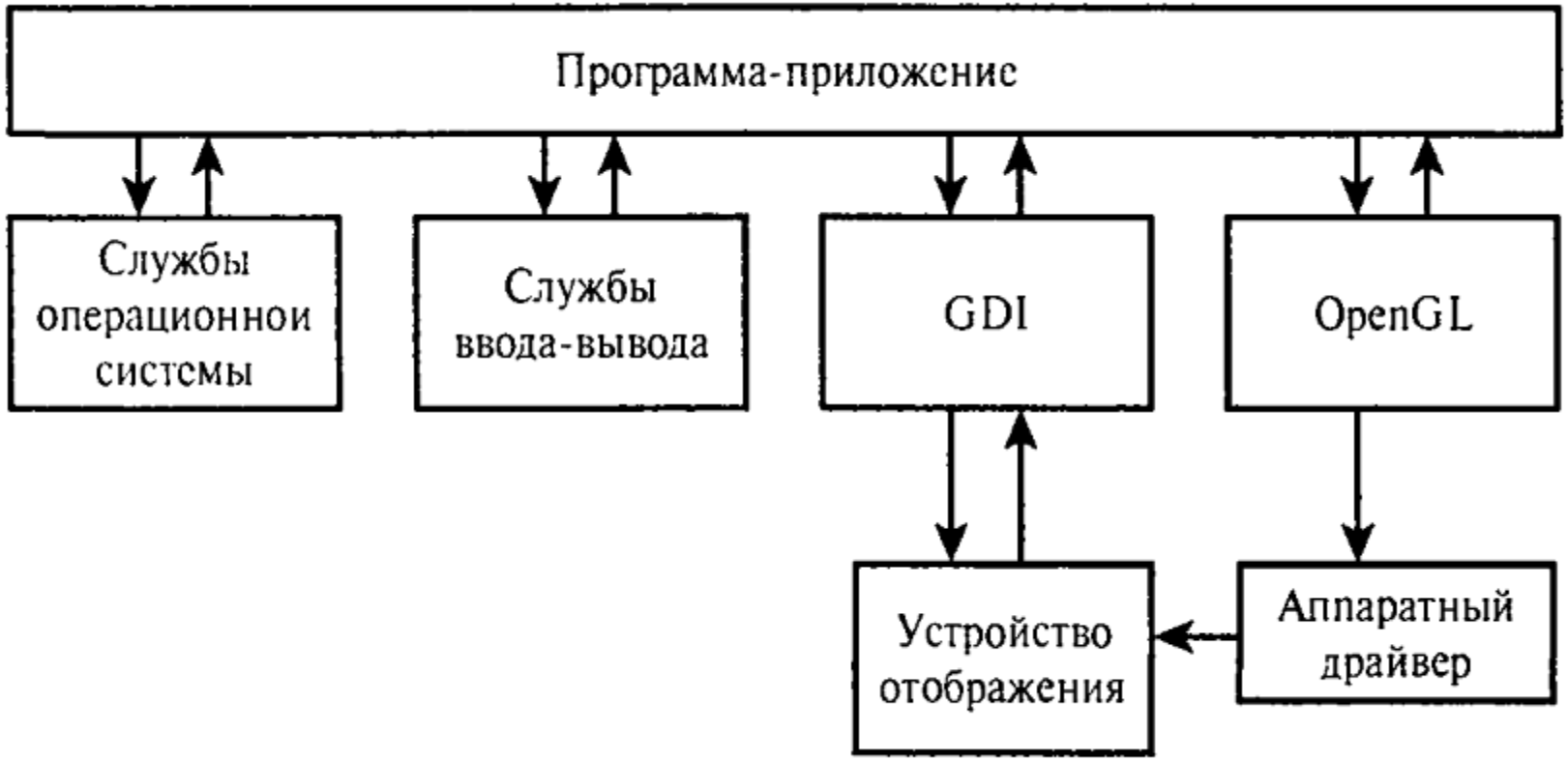
OpenGL

Основные возможности для достижения реалистичности

- Набор функции для
  - описания базовых примитивов: точек, линий, многоугольников, и т.п.
  - задания атрибутов: цвета, характеристик материала, текстур, параметров освещения
  - описания источников света
  - визуализации: для задания положения наблюдателя в виртуальном пространстве
- Видовые и координатные преобразования
- Удаление невидимых линий и поверхностей (z-буфер)
- Реалистичное закрашивание (метод Гуро и Фонга)
- Наложение текстур и добавление спецэффектов
- Альфа-буфер, сопряжение цветов (blending), устранение ступенчатости (antialiasing)

OpenGL

Использование в приложении

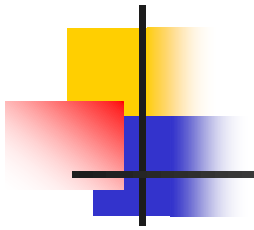


OpenGL
Архитектура

- Конвейер обработки данных. Команды OpenGL всегда обрабатываются в том порядке, в котором они поступают, при этом определение объекта сразу вызывает его визуализацию.
- Модель «клиент-сервер». Приложение выступает в роли клиента, сервер интерпретирует и выполняет их. Сервер может находиться либо на одном компьютере с клиентом, либо на удалённой графической станции.
- OpenGL как «конечный автомат» (state machine). Можно указать значения различным параметрам состояния (переменным), и они остаются в таком состоянии до тех пор, пока не будут принудительно изменены. Такими переменными являются значения текущего цвета, текущей нормали, координат текстуры, видовой трансформации и др.

OpenGL

Формат команд



```
GLfloat w[]={1.0,2.0,3.0};
glVertex3fv( w );
```

имя команды

Имя библиотеки
(gl, glu,glut,glaux)

Число аргументов
команды
(1,2,3,4)

Тип аргументов

b	– GLbyte
ub	– unsigned byte
s	– short
us	– unsigned short
i	– int
ui	– unsigned int
f	– float
d	– double

Аргументы

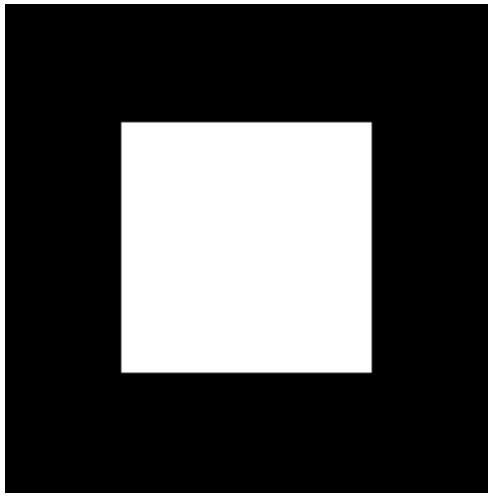
В качестве
параметров функции
используется
указатель на массив
значений

OpenGL

Пример рисования фигуры

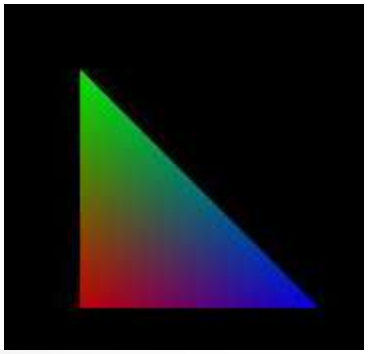
MS Windows:
```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
```
Linux X:
```
#include <X11/Xlib.h>
#include <GL/glx.h>
```

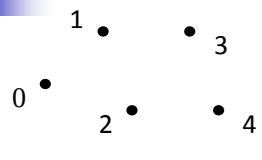
```
main() {
Откройте_графическое_окно_пожалуйста();
glClearColor (0.0, 0.0, 0.0, 0.0); //чёрный цвет очистки
glClear (GL_COLOR_BUFFER_BIT); // очистить окно
glColor3f (1.0, 1.0, 1.0);  // установить белый цвет
glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0); // сист. координат
glBegin(GL_POLYGON); // рисовать полигон
   glVertex3f (0.25, 0.25, 0.0); // по четырём вершинам
   glVertex3f (0.75, 0.25, 0.0);
   glVertex3f (0.75, 0.75, 0.0);
   glVertex3f (0.25, 0.75, 0.0);
glEnd(); // закончить ввод координат полигона
glFlush(); // немедленно выполнить все введенные команды
Измените_пожалуйста_содержимое_окна_и_отследите_события();
}
```



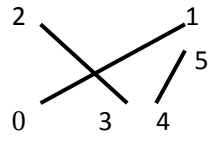
OpenGL

Геометрические примитивы

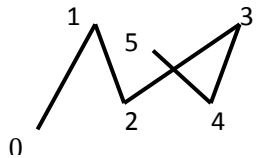




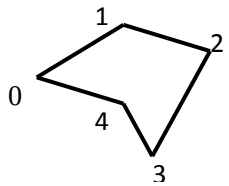
GL_POINTS



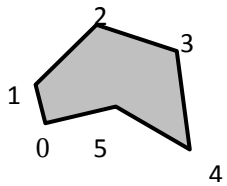
GL_LINES



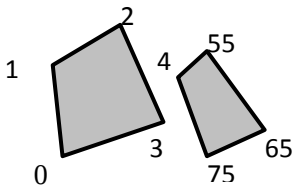
GL_LINE_STRIP



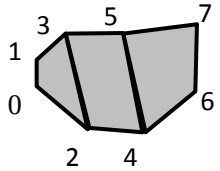
GL_LINE_LOOP



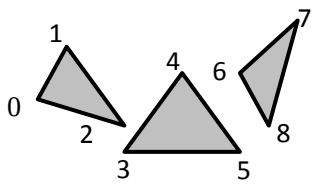
GL_POLYGON



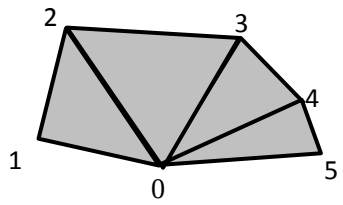
GL_QUADS



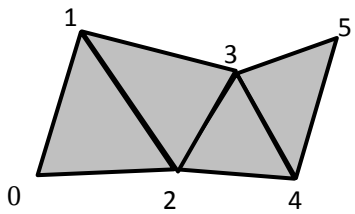
GL_QUAD_STRIP



GL_TRIANGLES



GL_TRIANGLE_FAN



GL_TRIANGLE_STRIP

```
glBegin(GL_TRIANGLES);
   glColor3d(1,0,0);
   glVertex2d(0,0);
   glColor3d(0,1,0);
   glVertex2d(0,3);
   glColor3d(0,0,1);
   glVertex2d(3,0);
glEnd();
```

- Атрибуты вершины:
  - Положение в пространстве
  - Материал
  - Цвет
  - Нормаль

OpenGL

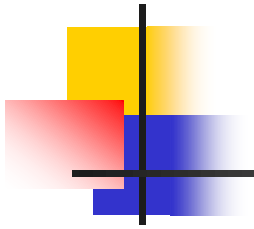
Геометрические примитивы

Команды, допустимые между вызовами glBegin() и glEnd()

Команда	Назначение
glVertex*()	установка координат вершины
glColor*()	установка текущего цвета
glIndex*()	установка текущего цветового индекса
glNormal*()	установка координат вектора нормали
glTexCoord*()	установка координат текстуры
glMultiTexCoord*ARB()	установка координат текстуры при мультитекстурировании
glEdgeFlag*()	контролирует рисование ребер
glMaterial*()	установка свойств материала
glArrayElement()	выделяет данные из массива вершин
glEvalCoords*(), glEvalPoint*()	генерирует координаты
glCallList(), glCallLists()	выполняет список отображения

OpenGL

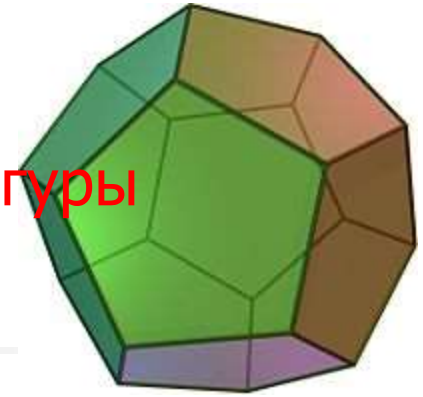
Библиотека GLUT: управление приложением



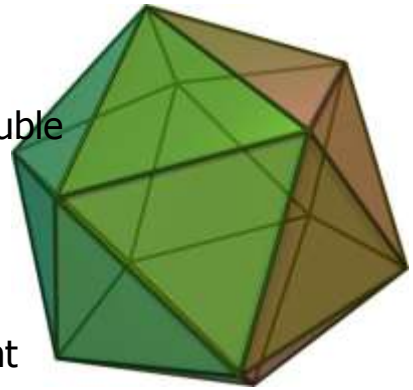
- *OpenGL Utility Toolkit* (GLUT) – графическая библиотека, реализующая функции инициализации графического окна в конкретной ОС, обработки событий в окне (например, чтение событий от клавиатуры и мыши) и рисования базовых высокоуровневых 3D объектов, созданных из примитивов (сфера, куб, тор, чайник и др.).
- В библиотеке GLUT определены функции с обратным вызовом (*callback function*). Если пользователь подвинул мышь, нажал на кнопку клавиатуры или изменил размеры окна, происходит событие и вызывается соответствующая функция – обработчик событий.
- Библиотека GLUT не входит в состав OpenGL, а является лишь переносимой прослойкой между OpenGL и оконной подсистемой, предоставляя минимальный интерфейс.

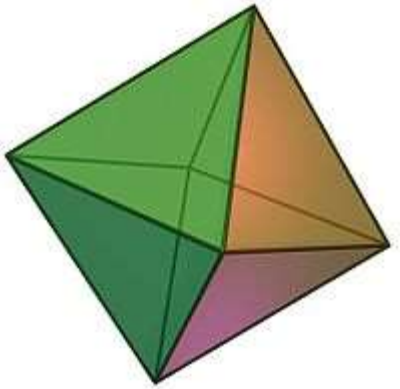
OpenGL

Библиотеки GLU и GLUT: сложные фигуры



- куб - void **glutSolidCube** (GLdouble *size*); void **glutWireCube** (GLdouble *size*)
- сфера - void **glutSolidSphere** (GLdouble *radius*, GLint *slices*, GLint *stacks*); void **gluSphere** (GLUquadricObj **qobj*, GLdouble *radius*, GLInt *slices*, GLint *stacks*);
- цилиндр – void **gluCylinder**(GLUquadricObj **qobj*, GLdouble *baseRadius*, GLdouble *topRadius*, GLdouble *height*, GLint *slices*, GLint *stacks*);
- диск – void **gluDisk** (GLUquadricObj **qobj*, GLdouble innerRadius, GLdouble outerRadius, GLint slices, GLint loops);
- конус - void **glutSolidCone** (GLdouble *radius*, GLdouble *height*, GLint *slices*, GLint *stacks*);
- тор - void **glutSolidTorus** (GLdouble *innerRadius*, GLdouble *outerRadius*, GLint *nsides*, GLint *rings*);
- тетраэдр - void **glutSolidTetrahedron** (void);
- додекаэдр - void **glutSolidDodecahedron** (GLdouble *radius*);
- икосаэдр - void **glutSolidIcosahedron** (void); - единичного радиуса
- октаэдр - void **glutSolidOctahedron** (void); - единичного радиуса
- чайник - void **glutSolidTeapot** (GLdouble *size*);





OpenGL

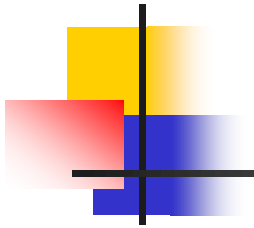
Программа рисования икосаэдра

```
#define X .525731112119133606
#define Z .850650808352039932

GLfloat vdata[12][3] = {
    {-X,0.0,Z},{X,0.0,Z},{-X,0.0,-Z},{X,0.0,-Z},
    {0.0,Z,X},{0.0,Z,-X},{0.0,-Z,X},{0.0,-Z,-X},
    {Z,X,0.0},{-Z,X,0.0},{Z,-X,0.0},{-Z,-X,0.0}
};

GLuint tindices[20][3] = {
    {1,4,0},{4,9,0},{4,5,9},{8,5,4},{1,8,4}, {1,10,8},{10,3,8},{8,3,5},{3,2,5},{3,7,2},
    {3,10,7},{10,6,7},{6,11,7},{6,0,11},{6,1,0}, {10,1,6},{11,0,9},{2,11,9},{5,2,9},{11,2,7}
};

int i;

glBegin(GL_TRIANGLES);
    for(i=0;i<20;i++)
{
    glVertex3fv(&vdata[tindices[i][0]][0]);
    glVertex3fv(&vdata[tindices[i][1]][0]);
    glVertex3fv(&vdata[tindices[i][2]][0]);
}
glEnd();
```

12 вершин, 20 граней, дополнительно требуется установка освещения и атрибутов граней

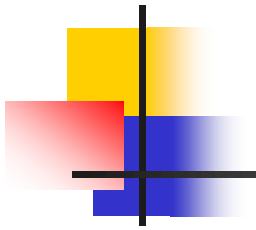
OpenGL

Структура приложения



```
main(int argc, char *argv[]) // главный цикл консольного приложения
{
    glutInit(&argc, argv); // начальная инициализация библиотеки GLUT
    glutInitDisplayMode(GLUT_RGB); // инициализация буфера кадра и
настройка режима экрана
    glutInitWindowSize(Width, Height); // задание начальных размеров
окна
    glutCreateWindow("My window") // визуализация окна на экране;
    glutDisplayFunc(Display); // регистрация функции рисования
    glutReshapeFunc(Reshape); // регистрация функции перерисовки;
    glutKeyboardFunc(Keyboard); // регистрация функции работы с клав.
    glutMainLoop(); // цикл контроля событий и вызова функций
}
```

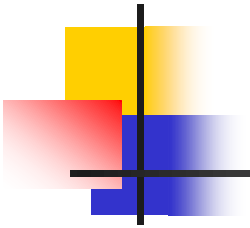
Управление приложением: функции GLUT



- Управление окном и реакция на события от устройств ввода
  - void glutDisplayFunc(void (*func)(void));
  - void glutReshapeFunc(void (*func)(int width, int height));
  - void glutKeyboardFunc(void (*func)(unsigned int key, int x, int y);
  - void glutMouseFunc(void (*func)(int button, int state, int x, int y));
  - void glutMotionFunc(void (*func)(int x, int y));
  - void glutPostRedisplay(void);
- Управление фоновым процессом
  - void glutIdleFunc(void (*func)(void));
- Запуск программы
  - void glutMainLoop(void);

OpenGL

Преобразования координат



- Модельное преобразование
  - изменение положения объекта в пространстве. Если не заданы модельные трансформации – модель не двигается и сохраняет свою позицию, ориентацию и размер.
- Видовые преобразования
  - ориентация камеры. Если видовое преобразование не задано, то «камера» по умолчанию находится в начале координат и направлена вдоль отрицательного направления оси *z*.
- Проекция на плоскость камеры
  - подбор «линз» центральной проекции
- Перевод в систему координат устройства
  - изменение размеров окна вывода

OpenGL

Преобразования координат

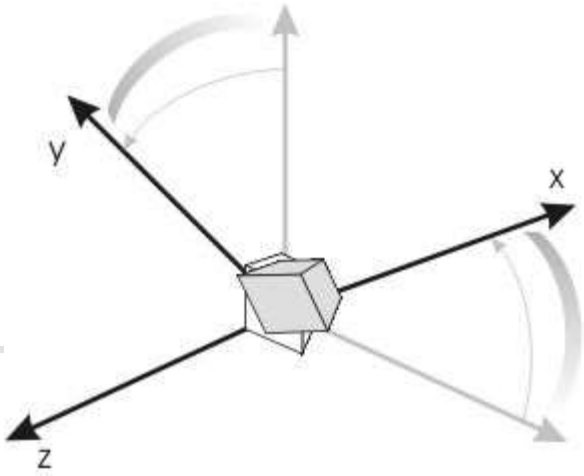
$$[M] = \begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix}$$

Для задания различных преобразований объектов сцены используются операции над матрицами размерности 4x4, при этом различают матрицы: видовую, проекций и текстуры.

GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE

- **glMatrixMode(¿)** – выбор матрицы для работы
- **glLoadMatrix(*M)** – определение элементов матрицы
- **glLoadIdentity()** - очистка текущей матрицы для будущих преобразований
- **glPushMatrix()** – сохранение матрицы в стек
- **glPopMatrix()** – восстановление матрицы из стека

OpenGL
Преобразования координат

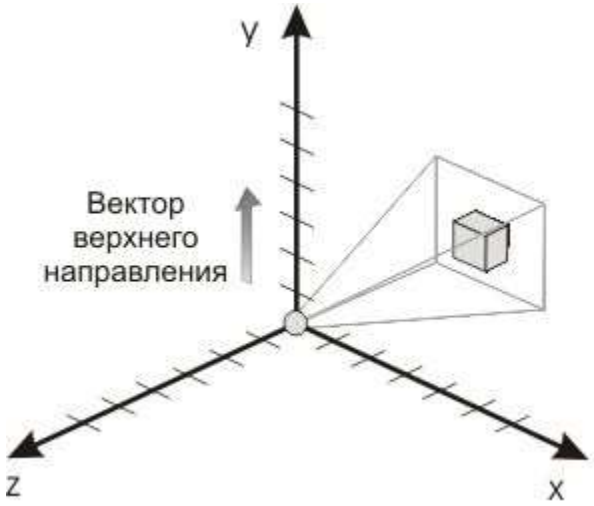


void **glTranslate**{fd} (TYPE *x*, TYPE *y*, TYPE *z*);

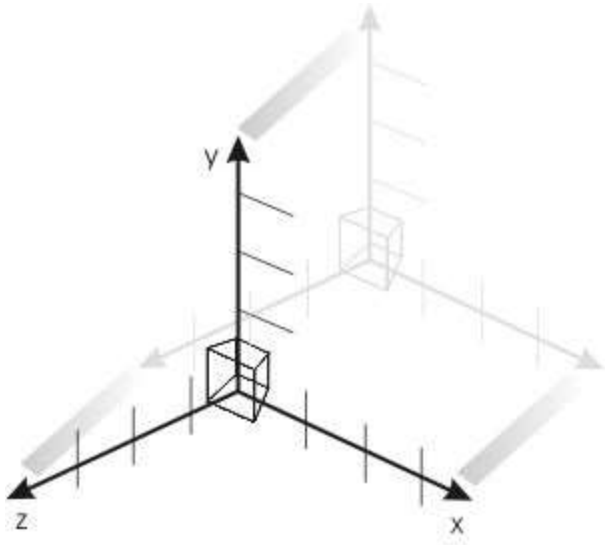
void **glRotate**{fd} (TYPE *angle*, TYPE *x*, TYPE *y*, TYPE *z*);

void **glScale**{fd} (TYPE *x*, TYPE *y*, TYPE *z*);

void **glu****LookAt** (GLdouble *eyex*, GLdouble *eyey*, GLdouble *eyez*,
GLdouble *centerx*, GLdouble *centery*, GLdouble *centerz*,
GLdouble *upx*, GLdouble *upy*, GLdouble *upz*);



Выбранная точка обзора задается аргументами *eyex*, *eyey* и *eyez*. Аргументы *centerx*, *centery* и *centerz* задают любую точку на линии обзора, обычно в середине обозреваемой сцены. Аргументы *upx*, *upy* и *upz* определяют, какое направление считается верхним (поворот камеры, по умолчанию 0,1,0).



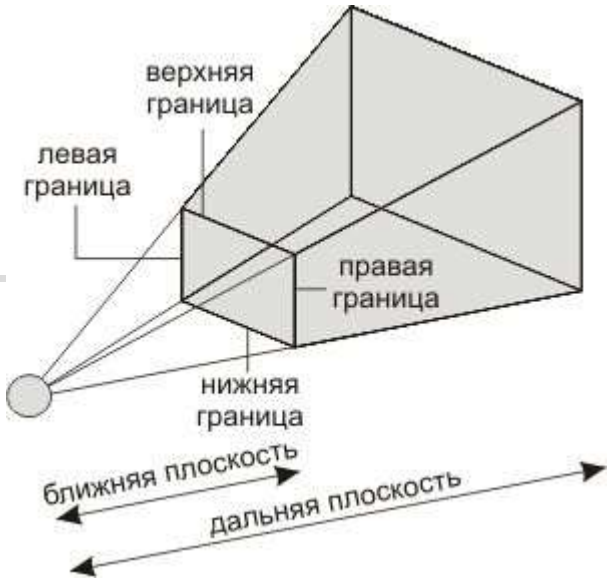
OpenGL

Программа рисования куба: применение трансформаций

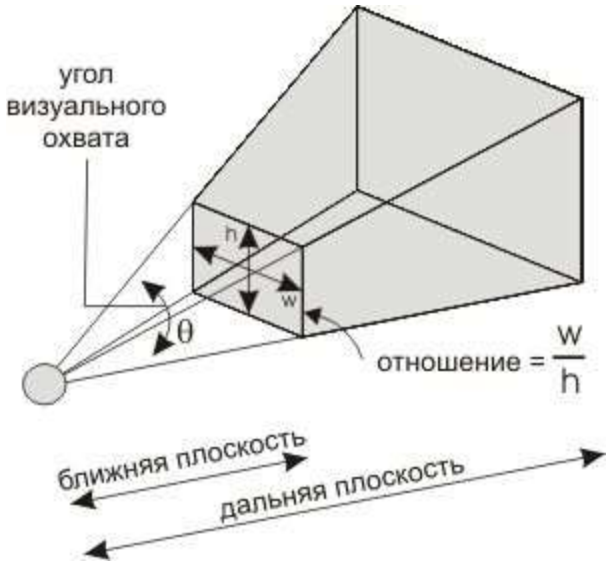
```
#include <glut.h>

void init(void) // Инициализация
{
   glClearColor(0.0,0.0,0.0,0.0);
   glShadeModel(GL_FLAT);
}

void display(void) //Отображение
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0,1.0,1.0);

   //Очистить матрицу
   glLoadIdentity();

   //Видовая трансформация(камера)
   gluLookAt(0.0,0.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);

   //Модельная трансформация
   glScalef(1.0,2.0,1.0);
   glutWireCube(1.0);
   glFlush();
}
```

```
//Изменение размеров окна
void reshape(int w, int h)
{
   glViewport(0,0,(GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glFrustum(-1.0,1.0,-1.0,1.0,1.5,20.0);
   glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char** argv)
{
   glutInit(&argc,argv);
   glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
   glutInitWindowSize(500,500);
   glutInitWindowPosition(100,100);
   glutCreateWindow("Transformed Cube");
   init();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutMainLoop();
   return 0;
}
```

OpenGL
Проецирование



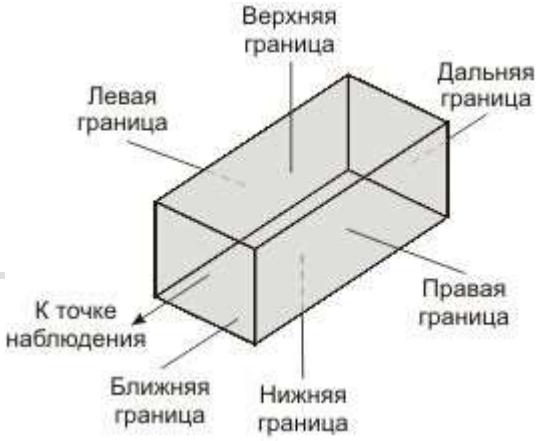
void **glFrustum** (GLdouble *left*, GLdouble *right*, GLdouble *bottom*, GLdouble *top*, GLdouble *near*, GLdouble *far*); - Создает матрицу перспективного проецирования и умножает на нее текущую матрицу. Объем видимости задается параметрами (*left*, *bottom*,*-near*) и (*right*, *top*,*-near*) определяющими координаты (*x*, *y*, *z*) левого нижнего и правого верхнего углов ближней отсекающей плоскости; *near* и *far* задают дистанцию от точки наблюдения до ближней и дальней отсекающих плоскостей.



void **glu****Perspective** (GLdouble *fovy*, GLdouble *aspect*, GLdouble *near*, GLdouble *far*); - Создает матрицу для пирамиды симметричного перспективного вида и умножает на нее текущую матрицу. Параметр *fovy* задает угол визуального охвата в плоскости *yz*. Параметр *aspect* – отношение ширины пирамиды к ее высоте. Параметры *near* и *far* представляют дистанции от точки наблюдения до ближней и дальней плоскостей отсечения вдоль отрицательного направления оси *z*.

OpenGL

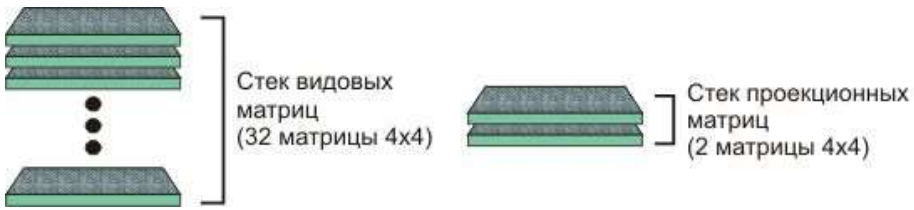
Проецирование



void **glOrtho** (GLdouble *left*, GLdouble *right*, GLdouble *bottom*, GLdouble *top*, GLdouble *near*, GLdouble *far*);

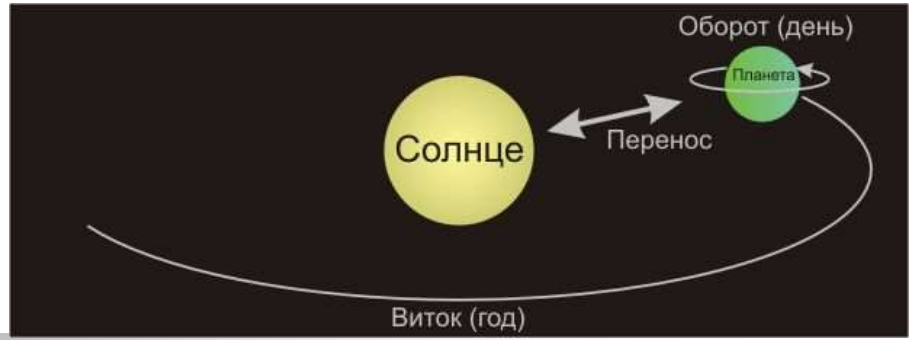
void **gluOrtho2D** (GLdouble *left*, GLdouble *right*, GLdouble *bottom*, GLdouble *top*);

void **glViewport** (GLint *x*, GLint *y*, GLint *width*, GLint *height*); - Задает прямоугольник пикселей в окне, в который будет перенесено финальное изображение. Параметры (*x*,*y*) задают нижний левый угол порта просмотра, параметры *width* и *height* – размер прямоугольника порта просмотра. По умолчанию левый нижний угол порта просмотра находится в левом нижнем углу окна, а его размер совпадает с размерами окна.



OpenGL

Пример программы




```
#include <glut.h>
int year=0, day=0;

void init(void) //Инициализация
{
  glClearColor(0.0,0.0,0.0,0.0);   glShadeModel(GL_FLAT);
}

void reshape(int w, int h) //Изменение размеров окна
{
  glViewport(0,0,(GLsizei) w, (GLsizei) h);
  glMatrixMode(GL_PROJECTION); glLoadIdentity();
  gluPerspective(60.0,(GLfloat) w/ (GLfloat) h,1.0,20.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity(); gluLookAt(0.0,0.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
}

void display(void) //Отображение
{
  glClear(GL_COLOR_BUFFER_BIT); glColor3f(1.0,1.0,1.0); glPushMatrix();

  //Рисуем солнце
  glutWireSphere(1.0,20,16); glRotatef((Glfloat)year,0.0,1.0,0.0);
  glTranslatef(2.0,0.0,0.0); glRotatef((GLfloat)day,0.0,1.0,0.0);

  //Рисуем планету
  glutWireSphere(0.2,10,8); glPopMatrix(); glutSwapBuffers();
}
```

```
//Реакция на клавиатуру
void keyboard(unsigned char key,int x, int y)
{
  switch(key)
  {
    case 'd':
      day=(day+10)%360;
      glutPostRedisplay();
      break;
    case 'D':
      day=(day-10)%360;
      glutPostRedisplay();
      break;
    case 'y':
      year=(year+5)%360;
      glutPostRedisplay();
      break;
    case 'Y':
      year=(year-5)%360;
      glutPostRedisplay();
      break;
    default:
      break;
  }
}
```

Язык OpenGL

Модель освещения

Если механизм расчёта освещённости не включен, текущий цвет ассоциируется с цветом текущей вершины, и не производится никаких вычислений, касающихся нормалей, источников света, модели освещения и свойств материала. Расчет освещенности следует включить командой:
glEnable(GL_LIGHTING);

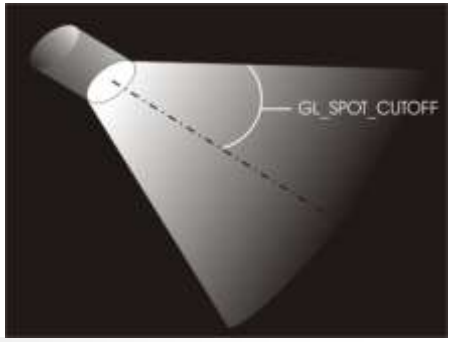
void **glLightModel**{if} (GLenum *pname*, TYPE *param*);

Имена параметров	**Значения по умолчанию**	**Смысл**
GL_LIGHT_MODEL_AMBIENT	(0.2,0.2,0.2,1.0)	RGBA интенсивность всей сцены
GL_LIGHT_MODEL_LOCAL_VIEWER	0.0 или GL_FALSE	способ вычисления углов зеркального отражения
GL_LIGHT_MODEL_TWO_SIDE	0.0 или GL_FALSE	выбор между односторонним и двухсторонним освещением
GL_LIGHT_MODEL_COLOR_CONTROL	GL_SINGLE_COLOR	вычисляется ли зеркальный цвет отдельно от фонового и диффузного

Локальная точка наблюдения позволяет получать более реалистичный результат по расчёту освещения, но, поскольку должны быть рассчитаны направления между ней и каждой вершиной, среднее быстродействие приложения может значительно снизиться.

Язык OpenGL

Источники света



GL_LIGHT0, GL_LIGHT1, ..., GL_LIGHT7

void **glLight**{if}**v** (GLenum *light*, GLenum *pname*, TYPE **param*);

Имена параметров	**Значения по умолчанию**	**Смысл**
GL_AMBIENT	(0.0,0.0,0.0,1.0)	Интенсивность фонового света
GL_DIFFUSE	(1.0,1.0,1.0,1.0) или (0.0,0.0,0.0,1.0)	Интенсивность диффузного света (значение по умолчанию для 0-го источника - белый свет, для остальных - черный)
GL_SPECULAR	(1.0,1.0,1.0,1.0) или (0.0,0.0,0.0,1.0)	Интенсивность зеркального света (значение по умолчанию для 0-го источника - белый свет, для остальных - черный)
GL_POSITION	(0.0,0.0,1.0,0.0)	Положение источника света (x,y,z,w)
GL_SPOT_DIRECTION	(0.0,0.0,-1.0)	Направление света прожектора (x,y,z)
GL_SPOT_EXPONENT	0.0	Концентрация светового луча
GL_SPOT_CUTOFF	180.0	Угловая ширина светового луча
GL_CONSTANT_ATTENUATION	1.0	Постоянный фактор ослабления
GL_LINEAR_ATTENUATION	0.0	Линейный фактор ослабления
GL_QUADRATIC_ATTENUATION	0.0	Квадратичный фактор ослабления

Каждый источник света нужно включить командой **glEnable(¿)**

Язык OpenGL

Пример создания источника света

```
void init (void)
{
// одномерные массивы аргументов для функций glLightModel и glLight
GLfloat light_ambient[] = { 0.3, 0.3, 0.3, 1.0 }; // цвет и интенсивность фоновой подсветки
GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 }; // цвет и интенсивность диффузного освещения
GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 }; // цвет при зеркальном отражении от объекта
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 }; // положение источника

glLightModelfv(GL_LIGHT_MODEL_AMBIENT, light_ambient); // включение фонового освещения
// устанавливаем параметры для одного источника света
glLightfv (GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv (GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv (GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv (GL_LIGHT0, GL_POSITION, light_position);

glEnable (GL_LIGHTING); //включение расчётов освещённости
glEnable (GL_LIGHT0); //включение источника света
…
}
```

Язык OpenGL

Второй источник света

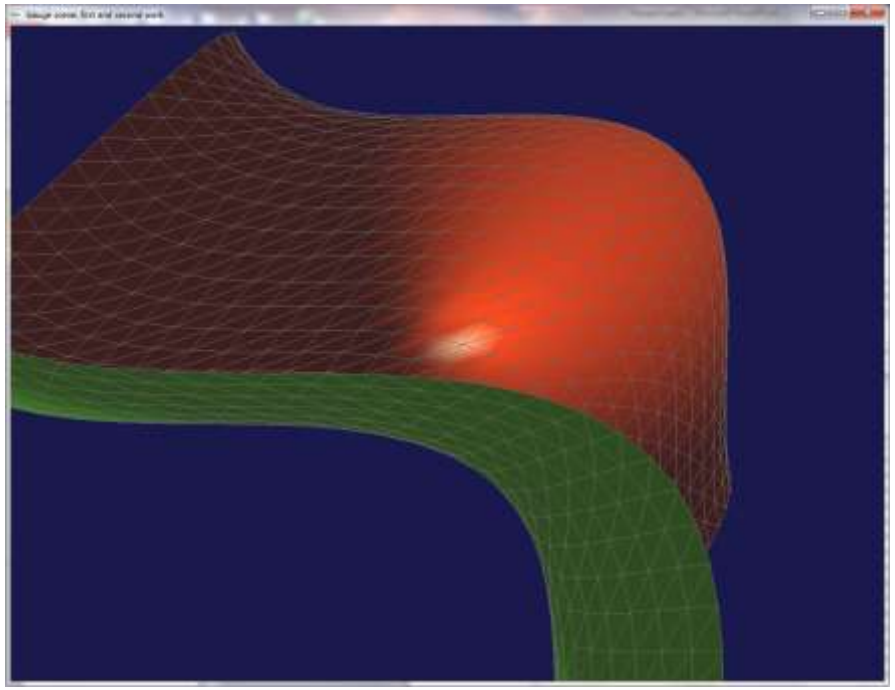
```
Glfloat light_ambient[]={0.2,0.2,0.2,1.0};
GLfloat light_diffuse[]={1.0,1.0,1.0,1.0};
GLfloat light_specular[]={1.0,1.0,1.0,1.0};
GLfloat light_position[]={-2.0,2.0,1.0,1.0};
GLfloat spot_direction[]={-1.0,-1.0,0.0};

glLightfv(GL_LIGHT1,GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT1,GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT1,GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT1,GL_POSITION, light_position);
glLightf(GL_LIGHT1,GL_CONSTANT_ATTENUATION, 1.5);
glLightf(GL_LIGHT1,GL_LINEAR_ATTENUATION, 0.5);
glLightf(GL_LIGHT1,GL_QUADRATIC_ATTENUATION, 0.2);
glLightf(GL_LIGHT1,GL_SPOT_CUTOFF, 45.0);
glLightfv(GL_LIGHT1,GL_SPOT_DIRECTION, spot_direction);
glLightf(GL_LIGHT1,GL_SPOT_EXPONENT, 2.0);

glEnable(GL_LIGHT1);
```

Язык OpenGL

Переключение режима расчёта цвета граней

1.

```
glClearColor(0.0,0.0,0.0,0.0);
glShadeModel(GL_SMOOTH);
GLfloat lmodel_ambient[]={1.0,1.0,1.0,1.0};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_NORMALIZE);
glEnable(GL_DEPTH_TEST);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
…
glDisable(GL_LIGHTING);
```

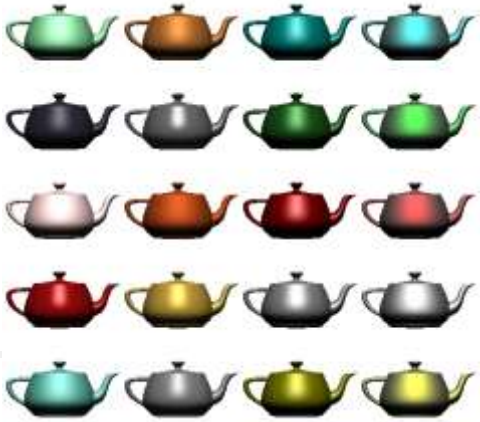


2.

```
glEnable(GL_COLOR_MATERIAL);
glColorMaterial(GL_FRONT, GL_DIFFUSE);
glColor3f(0.9,0.0,0.2);
…
glDisable(GL_COLOR_MATERIAL);
```

Язык OpenGL

Свойства материалов



Для корректного расчёта освещённости необходимо использовать единичные нормали к поверхностям (вершинам), однако преобразования координат изменяют модули длины нормалей. Для автоматической коррекции длин нормалей следует включить механизм нормализации командой **glEnable(GL_NORMALIZE)**.

void **glMaterial**{if}**v**(GLenum *face*, GLenum *pname*, TYPE **param*);

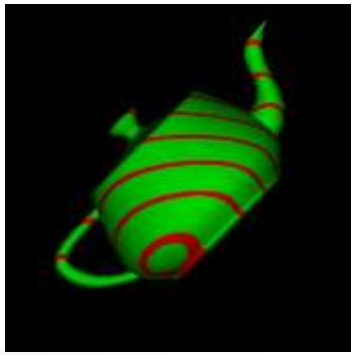
Имена параметров	**Значения по умолчанию**	**Смысл**
GL_AMBIENT	(0.2,0.2,0.2,1.0)	фоновый цвет материала
GL_DIFFUSE	(0.8,0.8,0.8,1.0)	диффузный цвет материала
GL_AMBIENT_AND_DIFFUSE		фоновый и диффузный цвет материала
GL_SPECULAR	(0.0,0.0,0.0,1.0)	цвет материала на отражение
GL_SHININESS	0.0	показатель зеркального отражения
GL_EMISSION	(0.0,0.0,0.0,1.0)	самосвечение материала
GL_COLOR_INDEXES	(0,1,1)	индексы фонового, диффузного и зеркального цветов

Язык OpenGL

Пример создания материала для объекта сцены

```
// одномерные массивы аргументов для определения свойств материала
float mat1_dif[]={0.2f,0.9f,0.0f}; // цвет диффузного отражения при освещении
float mat1_amb[]= {0.2f,0.2f,0.5f}; // рассеянный свет материала (цвет в тени)
float mat1_spec[]={0.6f,0.1f,0.6f}; // цвет при отражении (пятно от источника света)
float mat1_shininess=0.9f*128; // степень зеркальности материала (от 0 до 128)
…
void display (void)
{
…
 glPushMatrix (); //сохраняем матрицу преобразований
 // задание параметров текущего материала
 glMaterialfv (GL_FRONT,GL_AMBIENT,mat1_amb);
 glMaterialfv (GL_FRONT,GL_DIFFUSE,mat1_dif);
 glMaterialfv (GL_FRONT,GL_SPECULAR,mat1_spec);
 glMaterialf  (GL_FRONT,GL_SHININESS,mat1_shininess);
 glutSolidIcosahedron(); //объект, которому автоматически назначается текущий материал
 glPopMatrix (); //восстанавливаем матрицу преобразований
…
}
```

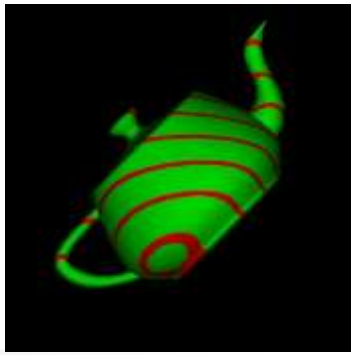
Язык OpenGL
Текстурирование



- Считывание графических данных из файла (формата BMP или DIB) удобнее всего проводить с помощью функции **aux**DIBImageLoad (filename). В качестве результата функция возвращает указатель на область памяти, где хранятся преобразованные данные.
- int **gluScaleImage** (GLenum *format*, GLint *widthin*, GLint *heightin*, GLenum *typein*, const **datain*, GLint *widthout*, GLint *heightout*, GLenum *typeout*, void **dataout*) – масштабирует изображения, используя установленный режим хранения пикселей для распаковки данных из *datain*. В качестве значения параметра *format* обычно используется константа GL_RGBA, определяющая формат хранения информации.
- void **glTexCoord**[1 2 3 4][s i f d] (type *coord*)
- void **glTexGen**{ifd} (GLenum *coord*, GLenum *pname*, TYPE *param*);
- void **gluQuadricTexture** (GLUquadricObj* *quadObject*, GLboolean *textureCoords*)

Язык OpenGL

Текстурирование



```
#include <GL\glut.h>
#include <GL\glaux.h>
#include <math.h>
#define TETR_LIST 1
...
GLUquadricObj* QuadrObj;
GLuint TexId;
```

```
void TextureInit()
{
 char strFile[]="texture.bmp";
 //--Выравнивание в *.bmp по байту
 glPixelStorei(GL_UNPACK_ALIGNMENT,1);
 //--Создание идентификатора для текстуры
 glGenTextures(1,&TexId);
 //--Загрузка изображения в память
 AUX_RGBImageRec *pImage = auxDIBImageLoad(strFile);
 int BmpWidth  = pImage->sizeX;
 int BmpHeight = pImage->sizeY;
 void* BmpBits = pImage->data;
 //--Начало описания свойств текстуры
 glBindTexture (GL_TEXTURE_2D,TexId);
 //--Создание уровней детализации и инициализация текстуры
 gluBuild2DMipmaps(GL_TEXTURE_2D,3,BmpWidth, BmpHeight,
                   GL_RGB,GL_UNSIGNED_BYTE,BmpBits);
 //--Разрешение наложения этой текстуры на quadric-объекты
 gluQuadricTexture(QuadrObj, GL_TRUE);
//--Повтор изображения по параметрическим осям s и t
 glTexParameteri (GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_REPEAT);
 glTexParameteri (GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT);
 //--Не использовать интерполяцию при выборе точки на текстуре
 glTexParameteri (GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
 glTexParameteri (GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
 //--Совмещать текстуру и материал объекта
 glTexEnvi (GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,GL_MODULATE);
}
```

Язык OpenGL

Пример создания материала с текстурой

```
// одномерные массивы аргументов для определения свойств материала
float mat1_dif[]={0.2f,0.9f,0.0f}; // цвет диффузного отражения при освещении
float mat1_amb[]= {0.2f,0.2f,0.5f}; // рассеянный свет материала (цвет в тени)
float mat1_spec[]={0.6f,0.1f,0.6f}; // цвет при отражении (пятно от источника света)
float mat1_shininess=0.9f*128; // степень зеркальности материала (от 0 до 128)
…
void display (void)
{
…
   glPushMatrix ();
   glMaterialfv (GL_FRONT,GL_AMBIENT,mat1_amb);
   glMaterialfv (GL_FRONT,GL_DIFFUSE,mat1_dif);
   glMaterialfv (GL_FRONT,GL_SPECULAR,mat1_spec);
   glMaterialf  (GL_FRONT,GL_SHININESS,mat1_shininess);
   glEnable(GL_TEXTURE_2D);
   gluSphere (QuadrObj, 0.5, 25, 25);
   glDisable(GL_TEXTURE_2D);
   glPopMatrix ();
…
}
```