II ГОРОДСКОЙ ФЕСТИВАЛЬ НАУКИ, ИСКУССТВА И СПОРТА
«ТАЛАНТЫ РОССИИ»

МАСТЕР-КЛАСС
«КОМПЬЮТЕРНАЯ ГРАФИКА ДЛЯ ОДАРЕННЫХ ШКОЛЬНИКОВ»

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБЩЕОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

САНКТ-ПЕТЕРБУРГСКИЙ ГУБЕРНАТОРСКИЙ ФИЗИКО-МАТЕМАТИЧЕСКИЙ ЛИЦЕЙ № 30

Группа компьютерной графики ФМЛ № 30

CGSG
ФМЛ № 30

14 марта 2019 года

Санкт-Петербург

Средний пр. ВО, д.31

История Санкт-Петербургского губернаторского физико-математического лицея №30 начинается с 1897 года, когда на Васильевском острове было открыто городское народное училище с двенадцатью классами им. Екатерины II. В 1961 году школа одной из первых в городе стала математической. В 2013 году ФМЛ № 30 получил статус Санкт-Петербургского губернаторского лицея.

В течение многих лет лицей успешно реализует общеобразовательные программы основного общего и среднего (полного) образования, обеспечивающие дополнительную (углубленную) подготовку обучающихся по математике, физике, информатике и программированию. Учителями лицея разработаны дифференцированные учебные программы, комплексы, учебники и методические пособия, используются современные информационно-коммуникационные технологии. В лицее есть уникальные лаборатории компьютерной физики и оптики, измерительной физики, шесть компьютерных кабинетов, мобильные компьютерные классы, лаборатория 3D моделирования, несколько кабинетов робототехники, конференц-зал с медиатекой. В лицее работает уникальный педагогический коллектив: почти треть педагогического состава является выпускниками лицея, заслуженные учителя РФ, Почетные работники общего образования РФ, кандидаты и доктора наук, лауреаты знака «За гуманизацию Санкт-Петербургской школы», победители и лауреаты профессиональных конкурсов разного уровня, победители олимпиад разного уровня, в том числе и международных.

Что привлекает учащихся в наш лицей?

Не только «сильные» математика, физика и программирование, но и традиционно отличное гуманитарное образование. Все наши выпускники поступают в лучшие ВУЗы города и России. Да, в «Тридцатке» учиться трудно, но интересно! Во внеурочное время учащиеся лицея посещают спецкурсы, факультативы, кружки, математические, физические и эрудитбои, школьные олимпиады, играют в “Дебаты” и занимаются спортом. Широкий выбор спецкурсов, факультативов и тематических лекций (организованных как лицеем, так и ведущими НИИ и университетами города) дает возможность нашим учащимся расширить и углубить свои знания в выбранной области, приобрести навыки исследовательской деятельности. Ежегодно лицеисты становятся дипломантами школьных, районных, городских, Всероссийских и международных научно-технических конференций по программированию, математике, физике, что создает условия для развития и становления одаренной личности. В лицее организовано отделение дополнительного образования детей, в рамках которого работают кружки по математике, физике, информатике и программированию, астрономии, баскетболу, волейболу, легкой атлетике, прикладной экономике и географии, туристско-краеведческий, театр на английском языке, литературная гостиная и другие.

Все это способствует тому, что наши учащиеся ежегодно становятся победителями олимпиад различного уровня по физике, математике, программированию, химии, экономике и предметам гуманитарного профиля. Нашим выпускникам значительно легче учиться в ВУЗах. Образование, полученное в лицее, позволяет им реализовать себя, найти свое место в жизни, быть востребованными и успешными.

Сильна «Тридцатка» и своими традициями. Ежегодно проводятся турниры математических боев, школьные командные и личные первенства по математике; летняя учебно-исследовательская практика проходит в рамках школы, а также в крупнейших университетах Санкт-Петербурга, в физических лабораториях ГОИ, СПбГУ и «Политеха»; несколько раз в год проводятся школьные командные олимпиады по программированию; организуются летние, осенние и весенние математические, физические и программистские лагеря; проводятся туристические слеты…

Группа компьютерной графики Санкт-Петербургского губернаторского физико-математического лицея № 30 была создана в 1992 году и за прошедшее время накопила огромный опыт в разработке больших проектов в области компьютерной графики и визуализации. За более чем четверть века, учащимися группы были созданы пакеты и библиотеки графических программ и функций, созданы объемные проекты обработки плоской и пространственной графики, цифровой обработке изображений, анимационные проекты, реализованы программы со сложной трехмерной визуализации и построением фотореалистичных изображений.

Группа ведет исследования в областях, связанных с динамической визуализацией, построению систем виртуальной реальности, использованием вычислительных средств современных видеокарт для разного рода вычислений и т.п. Программа группы строится из разных блоков. Это и спецкурс по компьютерной графике в течение года, летняя учебно-исследовательская практика, программистские лагеря на всех каникулах. На занятиях охватываются темы аппаратных аспектов (устройство компьютера, регистровое и прямое программирования видео-адаптеров), использования различных API и SDK для программирования графических задач, вопросы программирования под ОС MS-Windows, использования графических библиотек OpenGL, DirectX и Vulkan, попутные библиотеки. На практических занятиях создаются графические и игровые программы, большие системы 2D и 3D анимации. Изучаются методы работы с растровыми и векторными изображениями и шрифтами, алгоритмы обработки и преобразования изображений, сложные механизмы построения 3D моделей и сцен.

Опыт, накопленный группой компьютерной графики ФМЛ № 30, позволяет изучать сложные темы из областей математики и программирования, создавать большие проекты, позволяет учащимся освоить современные методы программирования задач компьютерной графики, освоить принципы командной работы.

С 2018 года в лицее реализуется инновационная образовательная программа по подготовке инженеров будущего со школьной скамьи. Лицей является опытно-экспериментальной площадкой по теме «Сочетание основного и дополнительного образования в учебном процессе».

Предлагаемый мастер-класс является кратким введением в базовые методы визуализации и построение графических программ. Занятие проводит руководитель группы компьютерной графики ФМЛ № 30 Галинский Виталий Александрович.

Дни открытых дверей ФМЛ № 30 проходят ежегодно, в субботу, на 2-й или 3-й неделе апреля.
Адрес лицея: *199004, Санкт-Петербург, Васильевский Остров, Средний пр. ВО, д.31.* Тел.: *323-4778 - директор*, *323-4253, 355-8857 - зам. директора*
E-mail: info@school30.spb.ru http://www.school30.spb.ru

Copyright © 2019 Computer Graphics Support Group. Санкт-Петербургский губернаторский физико-математический лицей № 30.

Компьютерная графика для одаренных школьников.

Введение.

Задача предлагаемого мастер-класса — познакомить слушателей с базовыми принципами построения графических приложений. В качестве базового интерфейса с графической подсистемой компьютера используется библиотека OpenGL (Open Graphics Library) — одной и популярнейших, кроссплатформенных библиотек построения динамических систем визуализаций.

Для взаимодействия с OpenGL программе необходимы:

- заголовочные файлы — **<gl/gl.h>** (основные функции) и **<gl/glu.h>** (вспомогательные функции)
- библиотеки **opengl32.lib** и **glu32.lib**

Современные компиляторы популярных языков программирования поддерживают библиотеку OpenGL в изначальной поставке. Однако, остается одна из проблем, с которой сталкивается начинающий программист компьютерной графики — инициализация графической подсистемы в современных операционных системах. Существует множество "фреймворков", помогающих решить данный вопрос. В данном мастер-классе используется популярная и простая библиотека GLUT (OpenGL Utility Toolkit).

Необходимые сведения и ресурсы:

- информация по OpenGL — **`http://www.opengl.org`**
- скачать библиотеку GLUT — **`http://www.opengl.org/resources/libraries/glut/glut_downloads.php`**
  или современный ее вариант (FreeGLUT) — **`http://sourceforge.net/projects/freeglut/`**

Для работы с OpenGL с использованием GLUT нам потребуется:

- заголовочный файл **"glut.h"**, находящийся в "видимых" путях включения заголовочных файлов компилятора Си/Си++ или в директории с проектом; данный файл включает в себя заголовочные файлы OpenGL, а для компилятора MS Visual Studio также автоматически подключает необходимые библиотеки;
- библиотека **glut32.lib**, находящаяся в "видимых" путях подключения библиотек или в директории с проектом;
- библиотека динамической компоновки (для программирования под ОС MS-Windows) **glut32.dll**, расположенная в системном директории ОС (.../WINDOWS/System32) или в директории с проектом.

Библиотека GLUT включает в свой функционал все, что связанно с интерфейсом ОС — создание рабочего окна, "цикла" сообщений и т.п., организация ввода (мышь, клавиатура, джойстик и т.д.).

Соглашения об именах, принятые в OpenGL и GLUT (каждая библиотека имеет свой префикс, например, **gl**, **glu**, **glut**, который добавляется в начало всех функций, типов и констант):

- `GLname` – типы – **`GLint GLfloat GLdouble GLuint GLbyte`**
- `glNameName` – функции – **`glClearColor glShadeModel`**
- `GL_NAME_NAME` – константы-параметры – **`GL_COLOR_MATERIAL GL_COLOR_BUFFER_BIT`**

для **`gl.h`** -> `GLname glNameName GL_NAME_NAME`
для **`glu.h`** -> `GLUname gluNameName GLU_NAME_NAME`
для **`glut.h`** -> `GLUTname glutNameName GLUT_NAME_NAME`

+ в концовках некоторых функций добавляется суффикс о количестве параметров и их типе. Например:

```
glColor:
  glColor3d( GLdouble R, GLdouble G, GLdouble B );  -- 3 числа double
      glColor3d(1.0, 1.0, 1.0);
  glColor4f( GLfloat R, GLfloat G, GLfloat B, GLfloat A );  -- 4 числа float
      glColor4f(1.0F, 1.0F, 1.0F, 1.0F);
  glColor3ub( GLubyte R, GLubyte G, GLubyte B ); -- 3 числа unsigned char
      glColor3ub(255, 255, 255);
  glColor3fv( GLfloat color[3] ); -- указатель на массив из 3 чисел float
    GLfloat c[3] = {1.0f, 1.0f, 1.0f};
    glColor3fv(c);
```

Общая схема работы (организация программы).

1.Полная "шапка" функции main:

```
int main( int argc, char *argv[] )
{
  . . .
  return 0;
}
```

2. Инициализация библиотек:
непосредственно инициализация:

```
glutInit(&argc, argv);                          -- инициализация всего
glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);    -- инициализация отображения
```

3. Создание рабочего окна

```
glutInitWindowPosition(0, 0);
glutInitWindowSize(800, 800);
glutCreateWindow("CG Open Lesson");
```

4. Регистрация "функций обратного вызова" (callbacks)
Это функции, которые вызываются асинхронно (т.е. не по действию пользователя) операционной системой в определенных ситуациях, связанными с опр. событиями (изменение состояния мыши, клавиатуры; перерисовать содержимое окна, изменение размера окна...)

Минимальный набор — сигнал перерисовки и клавиатура:

- регистрация в функции **main**:

```
glutDisplayFunc(Display);
glutKeyboardFunc(Keyboard);
```

- пример реализации:

```
void Display( void )
{
  /* типичное содержимое */
  glClearColor(0.3, 0.5, 0.7, 1); /* цвет очистки фона - r, g, b, a - 0..1 */
  glClear(GL_COLOR_BUFFER_BIT);   /* очистка фона */

  /* отрисовка */
  glColor3d(1, 0, 0);      /* устанавливаем цвет рисования - красный */
  glRectd(0, 0, 0.5, 0.5); /* рисуем прямоугольник - (x1,y1,x2,y2) - диагональ */

  /* завершение */
  glFinish();

  /* отсылка готового кадра при двойной буферизации */
  glutSwapBuffers();
}
```

```
void Keyboard( unsigned char Key, int X, int Y )
{
  if (Key == 27) /* выход по Escape */
    exit(0);
}
```

5. Запуск "бесконечного" цикла обслуживания сообщений.

```
glutMainLoop();
```

Простой макет программы с созданием окна вывода.

```
/* T01START.cpp
 * Общий макет.
 */

/* Включение файла-заголовка стандартных функций (для 'exit') */
#include <cstdlib>

/* Включение файла-заголовка библиотеки GLUT */
#include "glut.h"

/* Функция обратного вызова, вызываемая при перерисовке окна */
void Display( void )
{
} /* End of 'Display' function */

/* Функция обратного вызова, вызываемая при нажатии клавиш на клавиатуре */
void Keyboard( unsigned char Key, int MouseX, int MouseY )
{
  /* 'Key' принимает значение нажатой клавиши ('A', 'e' и т.п.),
   *       код клавиши 'ESC' (escape) - 27
   * 'MouseX' и 'MouseY' - значение координат мышки во время нажатия
   */
  if (Key == 27)
    exit(0); /* Выход по клавише 'Escape' - стандартной функцией 'exit'*/
} /* End of 'Keyboard' function */

/* Главная функция программы */
void main( int argc, char *argv[] )
{
  /* Инициализация библиотеки */
  glutInit(&argc, argv);
  /* Инициализация отображения:
   * параметр - комбинация значений (с помощью побитовой операции или '|'):
   *   GLUT_DOUBLE - двойная буферизация (альтернатива - GLUT_SINGLE - рисуем напрямую)
   *   GLUT_RGB - режим задания цвета по значениям красно, зеленой и синей компонент
   *   GLUT_ALPHA - включение возможности использования буфера прозрачности
   *   GLUT_DEPTH - включение буфера глубины - для удаления невидимых поверхностей
   */
  glutInitDisplayMode(GLUT_RGB);
  /* Задаем размеры и позицию окна */
  glutInitWindowSize(800, 800);
  glutInitWindowPosition(0, 0);
  /* Создаем окно */
  glutCreateWindow("Open CG Lesson. T01");

  /* Установка функций 'обратного вызова' */
  glutDisplayFunc(Display);
  glutKeyboardFunc(Keyboard);

  /* Запускаем основной цикл построения */
  glutMainLoop();
} /* End of 'main' function */

/* END OF 'T01START.cpp' FILE */
```

Вывод растровых изображений из массива.

1. Макет.

OpenGL содержит в себе удобный функционал для вывода растровых изображений из двумерных массивов.

Пример построения функции вывода с отображением буфера кадра из массива Frame:

```
/* Буфер кадра и его размеры */
#define FRAME_W 400
#define FRAME_H 300
unsigned char Frame[FRAME_H][FRAME_W][3];

/* Функция рисования точки */
void PutPixel( int X, int Y, int R, int G, int B )
{
  /* проверка попадания в кадр */
  if (X < 0 || Y < 0 || X >= FRAME_W || Y >= FRAME_H)
    return;
  /* занесение цвета */
  Frame[Y][X][0] = B;
  Frame[Y][X][1] = G;
  Frame[Y][X][2] = R;
} /* End of 'PutPixel' function */

/* Функция обратного вызова, вызываемая при перерисовке окна */
void Display( void )
{
  glClearColor(0.3, 0.5, 0.7, 1);
  glClear(GL_COLOR_BUFFER_BIT);

  /* рисуем точку */
  PutPixel(47, 30, 0, 255, 0);

  /* позиционирование в левый-верхний угол */
  glRasterPos2d(-1, 1);

  /* разворот "вниз" с масштабированием */
  glPixelZoom(3, -3);

  /* вывод буфера кадра */
  glDrawPixels(FRAME_W, FRAME_H, GL_BGR_EXT, GL_UNSIGNED_BYTE, Frame);

  glFinish();
  /* посылка сигнала на перерисовку */
  glutPostRedisplay();
} /* End of 'Display' function */
```

Следует заметить, что:

- изначально система координат выглядит так: от -1 до 1 по всем осям (X вправо, Y вверх, Z на нас)
- можно насильно послать сигнал на перерисовку окна (например для динамического изменения содержимого кадра):

  ```
  glutPostRedisplay();
  ```

- существует функция перехода в полноэкранный режим:

  ```
  glutFullScreen();
  ```

- функция вывода массива на экран (начало массива – левый-нижний угол):

  ```
  glDrawPixels(ширина в точках, выстота в точках,
    смысл каждого значения в массиве, тип каждого значения, массив);
  ```

- примеры для вывода массивов разных типов:

```
TYPE Frame[H][W];
```

TYPE	*смысл*	*тип*	*Пример*
unsigned char	GL_LUMINANCE *(яркость ч/б)* GL_RED GL_GREEN GL_BLUE	GL_UNSIGNED_BYTE	unsigned char Frame[H][W];
float [3]	GL_BGR_EXT	GL_FLOAT	float Frame[H][W][3];
double [3]	GL_RGB	GL_DOUBLE	double Frame[H][W][3];
unsigned char [4]	GL_BGRA_EXT	GL_UNSIGNED_BYTE	unsigned char Frame[H][W][4];
unsigned char [3]	GL_BGR_EXT	GL_UNSIGNED_BYTE	unsigned char Frame[H][W][3];

- функция зуммирования:

```
glPixelZoom(sx, sy);
```

- Позиционирование в левый верхний угол (система координат как на экране, X вправо, Y вниз):

```
glRasterPos2d(-1, 1);
glPixelZoom(s, -s);
```

2. Обработка сигналов устройства "мышь".

- регистрация в функции **main** (события от нажатия/отпускания кнопок мыши и перемещения с нажатой кнопкой):

```
glutMouseFunc(Mouse);
glutMotionFunc(Motion);
```

- пример реализации:

```
/* Функция обратного вызова, вызываемая при нажатии/отпускании кнопки мыши */
void Mouse( int Button, int State, int MouseX, int MouseY )
{
  /* 'Button' принимает значения:
   *   GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON
   * 'State' принимает значения:
   *   GLUT_DOWN, GLUT_UP
   * 'MouseX' и 'MouseY' - координаты курсора мышки относительно
   *   левого-верхнего угла окна
   */
  if (Button == GLUT_LEFT_BUTTON)
    if (State == GLUT_DOWN)
      /* ставим точку с учетом масштабирования */
      PutPixel(MouseX / 3, MouseY / 3, 255, 255, 255);
} /* End of 'Mouse' function */
```

```
/* Функция обратного вызова, вызываемая при движении мыши с нажатой кнопкой */
void Motion( int MouseX, int MouseY )
{
  /* ставим точку с учетом масштабирования */
  PutPixel(MouseX / 3, MouseY / 3, 255, 255, 255);
} /* End of 'Motion' function */
```

Макет 3D вывода.

1. Преобразования системы координат (СК).

Преобразования осуществляются в текущей СК. Все преобразования влияют на дальнейшее отображение, при этом все они (кроме ***glLoadIdentity()***) добавляют себя к уже сделанным преобразованиям.

Преобразования:

- Сброс всех преобразований (СК становиться такой, что ось *X* смотрит вправо [-1..1], ось *Y* смотрит вверх [-1..1], ось *Z* — на нас)
  ```
  glLoadIdentity();
  ```
- Параллельный перенос на вектор *(dx, dy, dz)* (суффикс 'd' в названии функции от *double* — тип всех параметров)
  ```
  glTranslated(dx, dy, dz);
  ```
- Поворот на угол α (в градусах) относительно радиус вектора *(x, y, z)* против часовой стрелки.
  ```
  glRotated(α, x, y, z);
  ```
- Масштабирование относительно начала координат вдоль оси *X* в *sx* раз, *Y* — в *sy* раз, *Z* — в *sz* раз
  ```
  glScaled(sx, sy, sz);
  ```
- Функция позиционирования СК так, что "находимся" в точке *(ex, ey, ez)*, "смотрим" в точку *(ax, ay, az)*, направление "вверх" приблизительно *(ux, uy, uz)*
  ```
  gluLookAt(ex, ey, ez,   ax, ay, az,   ux, uy, uz);
  ```

Сохранение контекста изменения СК:

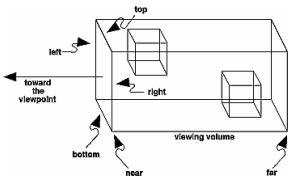
- Запомнить состояние изменения текущей СК (каждая СК запоминается персонально)
  ```
  glPushMatrix();
  ```
- Восстановить последнее запомненное с помощью ***glPushMatrix()*** состояние СК
  ```
  glPopMatrix();
  ```

2. Проецирование и изменение размеров окна.

Проецирование

Переход от 3-хмерного пространства к 2-умерному. По типу прохождения проекционных лучей: параллельная проекция (лучи параллельны друг другу) и центральная (лучи исходят из одной точки, получается эффект перспективы).

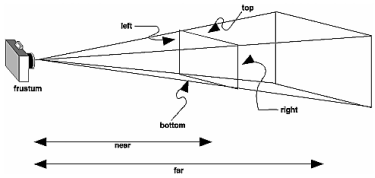
Параллельная проекция (ортогональная)



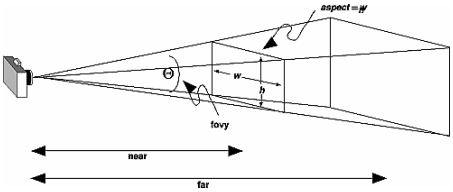

```
void glOrtho(left, right, bottom, top, near, far);
void gluOrtho2D(left, right, bottom, top); (near ← -1, far ← 1)
```

На экран попадают фигуры, отсеченные параллелепипедом видимости, все растягивается (сжимается) так, что "передняя" плоскость занимает всю видимую область вывода (на экране или в окне), задаваемую функцией `glViewport(x, y, w, h)`.

Центральная проекция




```
void glFrustum(left, right, bottom, top, near, far);
```

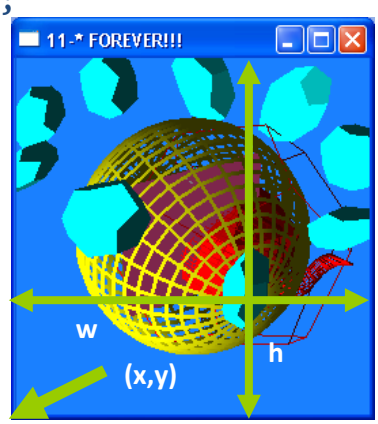



```
void gluPerspective(fovy, aspect, near, far);
```

Аналогично в область вывода идет все с “передней” плоскости отсечения.

- Зона вывода в окне задается функцией

```
glViewport(x, y, w, h);
```



Обработка изменения размеров окна

- регистрация в функции **main** (события — изменени размера окна):

```
glutReshapeFunc(Reshape);
```

- пример реализации:

```
/* Функция обратного вызова, вызываемая при изменении размеров окна */
void Reshape( int W, int H )
{
  /* устанавливаем область просмотра - все окно */
  glViewport(0, 0, W, H);

  /* устанавливаем систему координат "камеры" (проекции) */
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(60, (double)W / H, 0.1, 300);

  /* устанавливаем "мировую" систему координат */
  glMatrixMode(GL_MODELVIEW);
} /* End of 'Reshape' function */
```

3. Изменение в инициализации и удаление невидимых поверхностей.

Удаление невидимых поверхностей

Для удаления невидимых поверхностей в OpenGL используется алгоритм z-буфера (или буфера глубины — *depth buffer*). Идея: для каждой рисуемой точки запоминается ее z координата (глубина, расстояние до нее) и при выводе проверяется z координата очередной точки, отрисовывается та, что ближе.

Общий алгоритм:

- очищаем буфер глубины:

```
для всех x и y
   zbuf[y][x] = бесконечность
```

- отрисовываем сцену:

```
по всем фигурам и их полигонам (треугольникам):
      для всех x, y и z текущего многоугольника (z – добавляется):
            if (z < zbuf[y][x])
            {
                  zbuf[y][x] = z;
                  color[y][x] = цвет объекта
            }
```

- готовый кадр помещаем на экран

Изменения в программе:

В OpenGL проверка на глубину, сам буфер глубины и его очистка включается отдельно (далее приводится фрагменты исправлений в программе):

При инициализации в **main** добавляем: `glutInitDisplayMode( ... | GLUT_DEPTH);`

В очистке кадра (**Display**):

`glClear(GL_DEPTH_BUFFER_BIT);`

или `glClear(... | GL_DEPTH_BUFFER_BIT);`

И дополнительно: `glEnable(GL_DEPTH_TEST);` — в **main** (включение теста глубины для удаления невидимых поверхностей).

OpenGL моделирует простейшие источники света и освещение. Инициализация урощенног освещения в функции **main**:

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_COLOR_MATERIAL);
```

после этого `glColor3d(r, g, b);` задает цвет поверхности.

4. Синхронизация по времени.

Для обеспечения анимации от времени средствами стандартной библиотеки можно реализовать модель таймера, измеряющего время, прошедшее с момента старта программы.

- Пример реализации:

```
/* Включение файла-заголовка функций работы со временем (для 'clock') */
#include <ctime>

double SyncTime;  /* время в секундах с момента старта программы */

/* Функция - "таймер" */
void Timer( void )
{
  long t = clock();
  static long StartTime = -1;

  if (StartTime == -1)
    StartTime = t;
  SyncTime = (t - StartTime) / (double)CLOCKS_PER_SEC;
} /* End of 'Timer' function */
```

- Пример использования при построении кадра:

```
/* Функция обратного вызова, вызываемая при перерисовке окна */
void Display( void )
{
  glClearColor(0.3, 0.5, 0.7, 1);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  /* опрос таймера */
  Timer();

  /* сброс системы координат в начале кадра */
  glLoadIdentity();
  /* позиционирование "камеры" */
  gluLookAt(3, 3, 3, 0, 0, 0, 0, 1, 0);

  /* поворот фигуры на 30 градусов в секунду вокруг вертикальной оси */
  glRotated(SyncTime * 30, 0, 1, 0);
  glutSolidTeapot(0.5);

  glFinish();
  /* Копирование фонового кадра на экран  */
  glutSwapBuffers();
  /* посылка сигнала на перерисовку */
  glutPostRedisplay();
} /* End of 'Display' function */
```

- Сложная модель таймера с режимом паузы:

```
double SyncTime;  /* время в секундах с момента старта программы */
double DeltaTime; /* время в секундах межкадровой разницы */
int IsPause;      /* флаг паузы */

void Timer( void )
{
  long t = clock();
  static long StartTime = -1, PauseTime, OldTime;

  if (StartTime == -1)
    StartTime = OldTime = t;
  if (IsPause)
    PauseTime += t - OldTime, DeltaTime = 0;
  else
  {
    SyncTime = (t - PauseTime - StartTime) / (double)CLOCKS_PER_SEC;
    DeltaTime = (t - OldTime) / (double)CLOCKS_PER_SEC;
  }
  OldTime = t;
}
void Keyboard( unsigned char Key, int X, int Y )
{
  if (Key == 27) /* выход по Escape */
    exit(0);
  if (Key == 'p') /* включение/выключение паузы */
    IsPause = !IsPause;
}
```

4. Трехмерные фигуры GLUT.

```
/* GLUT pre-built models sub-API (glut.h) */
void glutWireSphere( double radius, int slices, int stacks );
void glutSolidSphere( double radius, int slices, int stacks );
void glutWireCone( double base, double height, int slices, int stacks );
void glutSolidCone( double base, double height, int slices, int stacks );
void glutWireCube( double size );
void glutSolidCube( double size );
void glutWireTorus( double innerRadius, double outerRadius, int sides, int rings );
void glutSolidTorus( double innerRadius, double outerRadius, int sides, int rings );
void glutWireDodecahedron( void );
void glutSolidDodecahedron( void );
void glutWireTeapot( double size );
void glutSolidTeapot( double size );
void glutWireOctahedron( void );
void glutSolidOctahedron( void );
void glutWireTetrahedron( void );
void glutSolidTetrahedron( void );
void glutWireIcosahedron( void );
void glutSolidIcosahedron( void );

/* FREEGLUT pre-built models sub-API (freeglut.h) */
void glutWireRhombicDodecahedron( void );
void glutSolidRhombicDodecahedron( void );
void glutWireSierpinskiSponge( int num_levels, double offset[3], double scale );
void glutSolidSierpinskiSponge( int num_levels, double offset[3], double scale );
void glutWireCylinder( double radius, double height, int slices, int stacks );
void glutSolidCylinder( double radius, double height, int slices, int stacks );
```

Онлайн курсы Группы компьютерной графики Санкт-Петербургского губернаторского физико-математического лицея № 30 (методические и рабочие материалы, слайды и видеолекции):

- **http://www.school30.spb.ru/cgsg/cgc/**
- **http://www.school30.spb.ru/cgsg/cgc2018/**

ГБОУ "Санкт-Петербургский губернаторский физико-математический лицей № 30"
7-я линия ВО, д.52 (Средний пр. ВО, д.31)
Санкт-Петербург, Россия
199004
URL: http://www.school30.spb.ru
E-mail: info@school30.spb.ru
Тел. ☎ +7 (812) 323-4253, 353-4778, 323 3555

Computer Science Department of Phys-Math Lyceum # 30:
http://www.school30.spb.ru/csd/
Computer Graphics Support Group of 30 Phys-Math Lyceum:
http://www.school30.spb.ru/cgsg/

Computer Science Department
Physics and Mathematics Lyceum # 30

Computer Graphics Support Group of 30 Phys-Math Lyceum