КОМПЬЮТЕРНАЯ ГРАФИКА

1. Компьютерная графика / Блинова Т.А., Порев В.Н. – К.: Издательство Юниор, СПб.: КОРОНА принт, К.: Век+, 2006. – 520., ил.
2. Алгоритмические основы машинной графики./ Роджерс Д.: Пер. с англ. – М.: Мир, 1989. – 512 с., ил.
3. Математические основы машинной графики. / Роджерс Д., Адамс Дж.: Пер. с англ. – М.: Мир, 2001. —604с., ил.
4. Компьютерная графика и геометрическое моделирование. / Сидоренко Л. – Учебное пособие. Издательство Питер 2009. – 224с., ил.
5. Компьютерная геометрия и алгоритмы машинной графики. / Никулин Е.А. СПб.: БХВ-Петербург. – 2003. – 560с., ил.
6. Компьютерная графика. / Порев В.Н. – СПб.: БХВ-Петербург. – 2002. – 428с., ил.
7. OpenGL. Программирование компьютерной графики. / Хилл Ф. 2-е издание – СПб.: Питер, 2002. – 1088с.: ил.
8. OpenGL. Суперкнига / Ричард С. Райт-мл., Бенджамин Липчак. – 3-е издание.: Пер. с англ. – М.: Издательский дом «Вильямс», 2006. – 1040с.: ил.

КОМПЬЮТЕРНАЯ ГРАФИКА

Компьютерная графика – это область информатики, занимающаяся проблемами получения различных изображений (рисунков, чертежей, мультипликации) на компьютере.

Компьютерная графика – это отрасль знаний, которая, с одной стороны, представляет комплекс аппаратных и программных средств, используемых для формирования, преобразования и выдачи информации в визуальной форме на средства отображения ЭВМ. С другой стороны, под компьютерной графикой понимают совокупность методов и приемов для преобразования при помощи ЭВМ данных в графическое представление.

ОСНОВНЫЕ НАПРАВЛЕНИЯ

1. Визуализация научных (расчетных или экспериментальных) данных.
2. Геометрическое проектирование и моделирование.
3. Распознавание образов.
4. Изобразительное искусство.
5. Виртуальная реальность.
6. Цифровое видео.

КЛАССИФИКАЦИЯ

Компьютерная графика подразделяется на:

– *статичную* (неподвижная);

– *динамичную* (анимация, компьютерная мультипликация).

Каждая из которых в свою очередь делится на *2-х* и *3-х* мерную.

В зависимости от способа формирования изображений, компьютерную графику принято делить на:

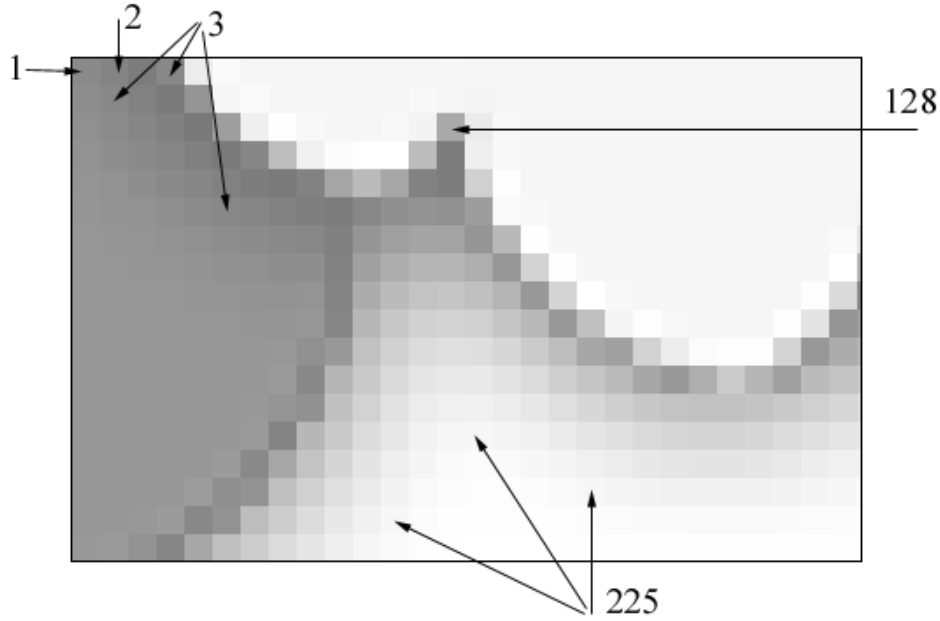
– *растровую*;

– *векторную*;

– *фрактальную*.

РАСТРОВАЯ ГРАФИКА

Исходное изображение



Фрагмент оцифрованного изображения и номера цветов

РАСТРОВАЯ ГРАФИКА

Растровая графика – машинная графика, в которой изображение представляется двумерным массивом точек (элементов растра), цвет и яркость каждой из которых задается независимо.

Растр (растровый массив) – представление изображения в виде двумерного массива точек, упорядоченных в строки и столбцы. Для каждой точки растра указывается цвет и яркость.

Пиксель – элемент (точка) растра, минимальная единица изображения, цвет и яркость которой можно задать независимо от остального изображения.

РАСТРОВАЯ ГРАФИКА

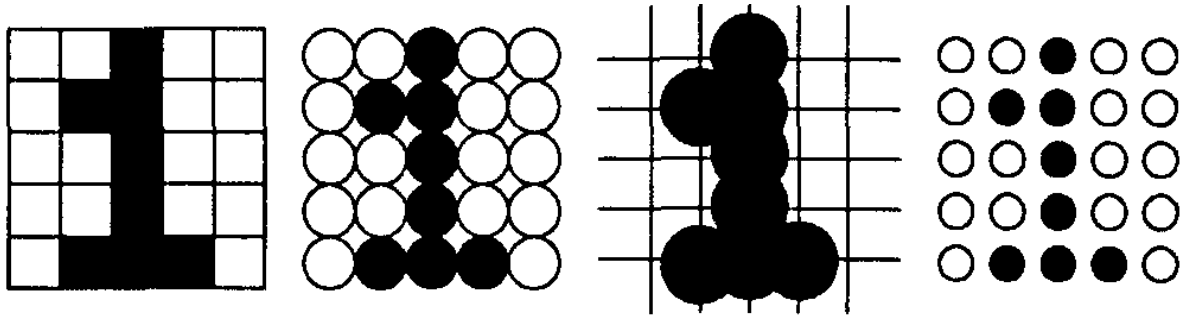
Размер растра обычно измеряется количеством пикселей по горизонтали и вертикали.

Разрешающая способность характеризует расстояние между соседними пикселями - шаг дискретной сетки растра. Разрешающую способность измеряют количеством пикселей на единицу длины. Наиболее популярная единица измерения - dpi (dots per inch) - количество пикселей в одном дюйме длины (2.54 см).

Удобен растр с одинаковым шагом для обеих осей, то есть dpi X= dpi Y. Это удобно для многих алгоритмов вывода графических объектов.

РАСТРОВАЯ ГРАФИКА

Форма пикселей растра определяется особенностями устройства графического вывода.



РАСТРОВАЯ ГРАФИКА

Применение: обработка фотоизображений, художественная графика, реставрационные работы, работа со сканером.

Достоинства растровой графики:

- растровая графика эффективно представляет реальные образы;
- растровое изображение наиболее адаптировано для распространенных растровых устройств вывода - лазерных принтеров и др.

РАСТРОВАЯ ГРАФИКА

Недостатки:

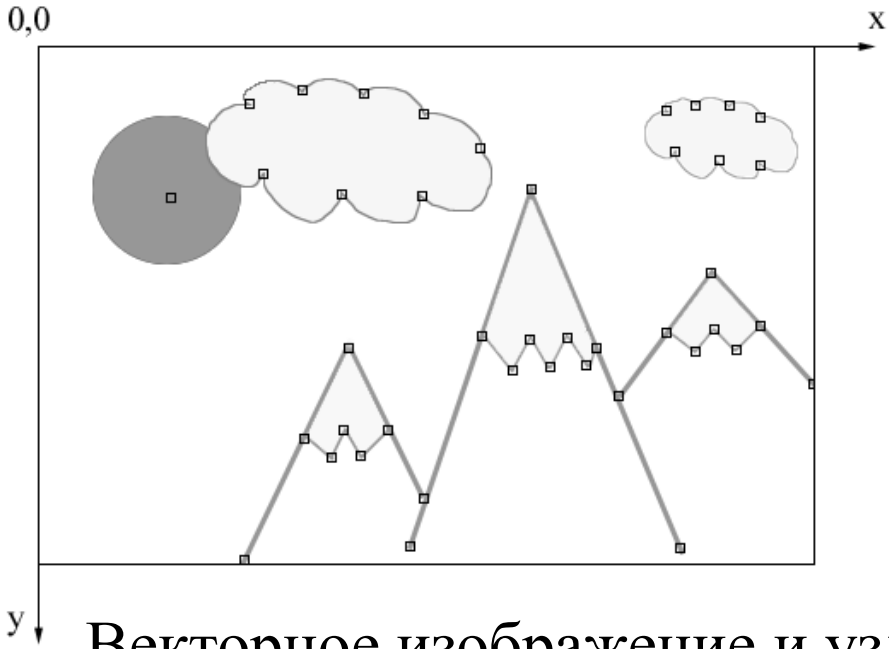
- занимают большой объем памяти;
- редактирование больших растровых изображений, занимающих большие массивы памяти, требуют большие ресурсы компьютера и, следовательно, требуют большего времени;
- трудоемкий процесс редактирования растровых изображений;
- при увеличении размеров изображения сильно ухудшается качество.

ВЕКТОРНАЯ ГРАФИКА

Векторная графика описывает изображение с помощью математических формул. Выполняется разбиение изображения на ряд графических примитивов – точки, прямые, ломаные, дуги, полигоны. Изображение хранит не все точки объекта, а координаты узлов примитивов и их свойства.

ВЕКТОРНАЯ ГРАФИКА

Исходное изображение



Векторное изображение и узлы его примитивов

отрезок (…);

окружность(…);

кривая_Безье (…).

ВЕКТОРНАЯ ГРАФИКА

Преимущества векторной графики:

- использует все преимущества разрешающей способности любого устройства вывода, что позволяет изменять размеры векторного рисунка без потери качества (т. е. при изменении масштаба изображения оно не теряет своего качества и не меняет размер файл);
- позволяет редактировать отдельные части рисунка, не оказывая влияния на остальные;
- векторные изображения, как правило, занимают относительно небольшое место в памяти компьютера.

ВЕКТОРНАЯ ГРАФИКА

Недостатки:

- векторные изображения выглядят искусственно;
- меньше оттенков и полутонов чем в растровой графике.

Применение: компьютерная полиграфия, системы компьютерного проектирования, компьютерный дизайн и реклама.

ФРАКТАЛЬНАЯ ГРАФИКА

Базовым элементом фрактальной графики является сама математическая формула, т.е. никаких объектов в памяти компьютера не хранится и изображение строится исключительно по уравнениям.

Изображение строится по уравнению (или по системе уравнений), поэтому ничего, кроме формулы, хранить не надо. Изменив коэффициенты в уравнении, можно получить совершенно другую картину.

Аппаратно-независимое программирование и OpenGL

OpenGL очень мобильная и очень эффективная библиотека трехмерной графики и моделирования.

OpenGL — это не язык программирования, как C или C++. Строго говоря, «программ OpenGL» не существует, правильно говорить о программах, которые пишутся с учетом того, что в качестве одного из их программных интерфейсов приложения (Application Programming Interfaces — API) будет использован OpenGL.

Программирование управляемости событиями

Система автоматически устанавливает очередь событий, которая получает сообщения о том, что произошло некоторое событие, и обслуживает их по принципу «первым пришел — первым обслужен».

Программист организует свою программу как набор функций обратного вызова (callback functions), которые выполняются, когда происходят события. Функция обратного вызова создается для каждого типа событий, которые могут произойти.

Функции управления событиями

1. `void glutDisplayFunc (void (*func)(void)).`

Пример вызова функции: glutDisplayFunc(myDisplay);

Это означает, что в данном случае функция myDisplay() зарегистрирована в качестве функции обратного вызова для события обновления.

2. `void glutReshapeFunc (void (*func)(int width, int height)).`

Пример вызова функции: glutReshapeFunc(myReshape);

Функция myReshape() будет зарегистрирована с событием изменения формы окна, в которую автоматически передаются аргументы – новая ширина и высота изменившего свою форму окна.

Функции управления событиями

3. `void glutMouseFunc (void (*func)(int button, int state, int width, int height)`.

Пример вызова функции: glutMouseFunc(myMouse);

В функци. myMouse() автоматически передаются аргументы, описывающие местоположение мыши, а также вид действия, вызываемого нажатием кнопки мыши.

4. `void glutMotionFunc (void (*func)(int x, int y))`.

Указывает функцию, которая будет вызываться при движении мыши внутри окна в то время, как на ней нажата одна или несколько клавиш.

Функции управления событиями

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

Пример вызова функции glutKeyboardFunc(myKeyboard).

В функцию myKeyboard() автоматически передаются аргументы, сообщающие, какая клавиша была нажата. Для удобства этой функции также передается информация о положении мыши в момент нажатия клавиши.

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

Функция аналогична но для опроса специальных клавиш.

Функции управления событиями

7. `void glutPostRedisplay (void).` Помечает, что текущее окно требует перерисовки. После этого при любой возможности будет вызвана функция перерисовки окна, зарегистрированная вызовом glutDisplayFunc().

8. `void glutIdleFunc (void (*func)(void)).` Задает функцию, выполняемую в случае, если отсутствуют сообщения. Выполнение этой функции обратного вызова можно отменить передачей glutIdleFunc() аргумента NULL.

9. Функцию `glutMainLoop()` необходимо использовать после регистрации функций обратного вызова, программа входит в бесконечный цикл, находясь в котором она просто ждет наступления событий.

Функции создания и открытия окна

– `glutInit(&argc,argv)-` функция инициализирует OpenGL Utility Toolkit.
– `glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB)-` функция определяет, как должен быть инициализирован дисплей.

Используются встроенные в нее константы, например: GLUT_SINGLE и GLUT_RGB с оператором OR (или) между ними показывают, что следует выделить один дисплейный буфер и что цвета задаются с помощью сочетаний красного, зеленого и синего цветов.

Функции создания и открытия окна

- `glutInitWindowSize(640,480)-` функция устанавливает размеры экранного окна при инициализации.
- `glutInitWindowPosition(100,150)-` функция устанавливает верхний левый угол экранного окна от верхнего края при инициализации.
- `glutCreateWindow("my first attempt") -` функция открывает и отображает экранное окно, помещая текст заголовка «my first attempt» в строку заголовка (title bar) окна.

Рисование графических примитивов

```
glBegin(GL_POINTS);
glVertex2i(l00,50);
glVertex2i(100,130);
glVertex2i(150,130);
glEnd();
```

Встроенные константы OpenGL для рисования примитивов:

GL_POINTS – для рисования точек.

GL_LINES – для рисования отрезков, при этом необходимо задавать четное количество вершин между командами glBegin(GL_LINES) и glEnd().

Рисование графических примитивов

GL_LINE_STRIP – для рисования ломаной линии. Ломаная рисуется посредством задания вершин в нужном порядке, между командами glBegin(GL_LINE_STRIP) и glEnd().

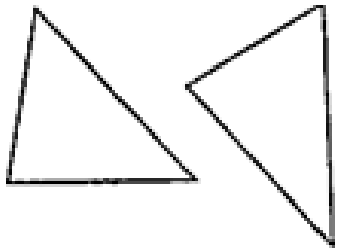
GL_LINE_LOOP – для рисования замкнутой ломаной линии, т.е. последняя точка соединяется с первой).

GL_POLIGON – для рисования полигона как объекта (используется для рисования закрашенных полигонов).

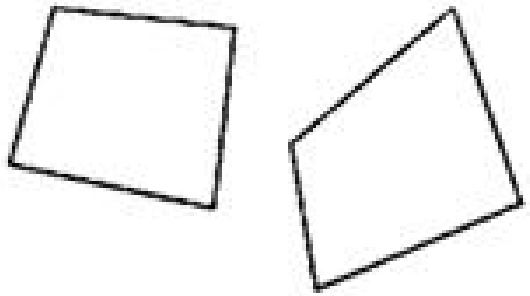
Рисование графических примитивов

GL_TRIANGLES – для рисования треугольников (из списка вершин берутся по три за один прием и рисуются для каждой тройки отдельные треугольники).

GL_QUADS – для рисования четырёхугольников(из списка вершин берутся по четыре за один прием и рисуются для каждой четверки отдельные четырехугольники).



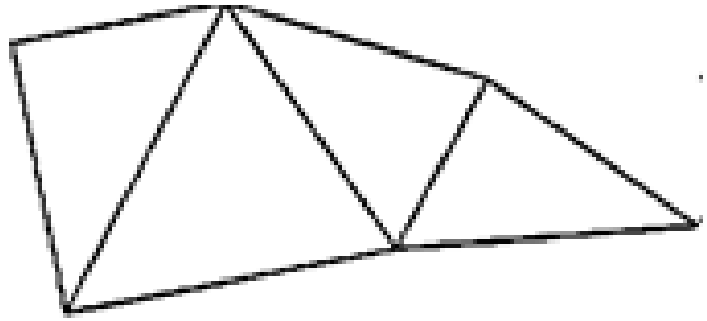
GL_TRIANGLES



GL_QUADS

Рисование графических примитивов

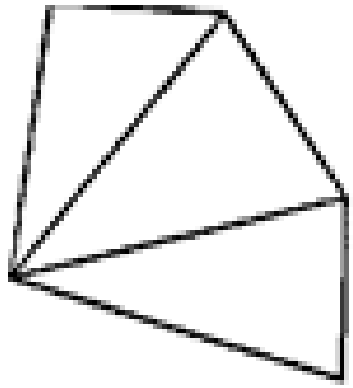
GL_TRIANGLE_STRIP – для рисования полос из треугольников (рисуется последовательность треугольников, опирающихся на тройки вершин: v_0, v_1, v_2, затем v_2, v_1,v_3, затем v_2, v_3, v_1 и т. д. (порядок следования такой, что все треугольники «кладутся поперек» в одном направлении, например, против часовой стрелки).



GL_TRIANGLE_STRIP

Рисование графических примитивов

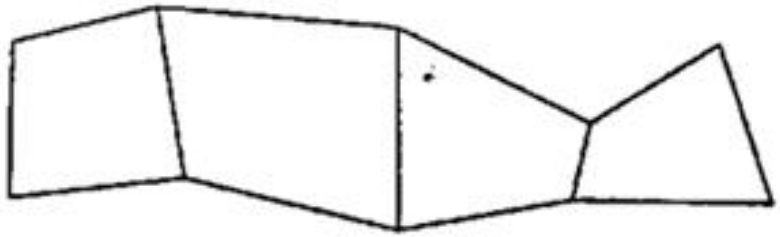
GL_TRIANGLE_FAN – для рисования веера из треугольников (рисуется последовательность соединенных между собой треугольников, опирающихся на тройки вершин: v_0, v_1, v_2 затем v_0, v_2, v_3, затем v_0, v_3, v_4 и т.д.)



GL_TRIANGLE_FAN

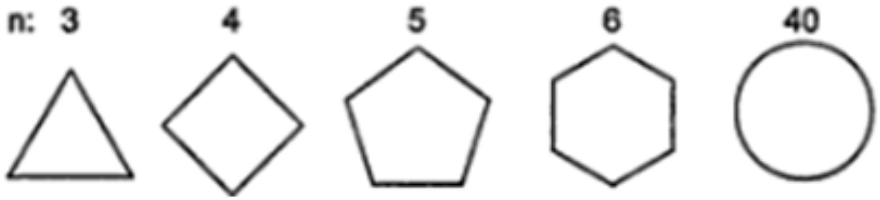
Рисование графических примитивов

GL_QUAD_STRIP – для рисования полос из четырехугольников (рисуется последовательность четырехугольников, опирающихся на четверки вершин: вначале v_0, v_1, v_3, v_2, затем v_2, v_3, v_5, v_4, затем v_4, v_5, v_7, v_6, и т. д.) Порядок следования такой, что все четырехугольники «кладутся поперек» в одном направлении, например против часовой стрелки.



GL_QUAD_STRIP

Правильные многоугольники и рисование окружности



$$x_i = R \cdot \cos(2\pi \cdot i / n),$$

$$y_i = R \cdot \sin(2\pi \cdot i / n),\ i = 0,\ldots,n-1.$$

```
void n_ygol(int n, float r)
{
if (n<3) {exit(0);}
GLfloat x,y;
     glBegin(GL_LINE_LOOP);
          for (int i=0; i<n; i++)
          {
          x=r*cos(2*3.14*i/n);
          y=r*sin(2*3.14*i/n);
          glVertex2f(x,y);
          }
     glEnd();
}
```

Правила именования

Типы данных OpenGL

Суффикс	Тип данных	Обычный тип C или C++	Название типа в OpenGL
b	8-битное целое (символ со знаком)	signed char	GLbyte
s	16-битное целое (короткое целое)	short	GLshort
i	32-битное целое (длинное целое)	int или long	GLint, GLsizei (обозначает параметр размера, предполагает целые числа)
f	32-битное вещественное (с плавающей точкой)	float	GLfloat, GLclampf (доп. диапазон 0?1)

Типы данных OpenGL

d	64-битное вещественное (двойной точности)	double	GLdouble, GLclampd (доп. диапазон 0?1)
ub (True or False)	8-битное число без знака (символ без знака)	unsigned char	GLubyte, GLboolean
us	16-битное число без знака (короткое целое без знака)	unsigned short	GLushort
ui	32-битное число без знака (целое без знака или длинное целое без знака)	unsigned int или unsigned long	GLuint, GLenum, GLbitfield (для переменных использующие двоичные разряды)

«Состояние» OpenGL

`void glPointSize (GLfloat size)` – задает размер точки. Если этот аргумент равен 3.0, то точка обычно рисуется как квадратик со стороной в три пиксела.

Антиалиасинг – это техника сглаживания рисуемых точек и линий (выключен по умолчанию). Если антиалиасинг *выключен*, дробные размеры точек округляются до ближайшего целого. Когда антиалиасинг *включен*, рисуется группа пикселей, и пиксели по границе обычно имеют цвет с пониженной интенсивностью для придания границе гладкого вида. В этом режиме дробные размеры точек до целого не округляются.

«Состояние» OpenGL

`glColor3f(red, green, blue)` – задает цвет для рисования может быть задан с помощью функции где значения переменных red, green и blue изменяются от 0,0 до 1,0.

`glClearColor(red, green, blue, alpha)` – задает цвет фона (переменная alpha определяет степень прозрачности, которая используется при смещении и создании таких специальных эффектов, как просвечивание).

`glClear(GL_COLOR_BUFFER_BIT)` – используется для очистки всего окна до цвета фона. Аргумент GL_COLOR_BUFFER_BIT представляет собой также встроенную в OpenGL константу.

«Состояние» OpenGL

Составной цвет	Красный компонент	Зеленый компонент	Синий компонент
Черный	0.0	0.0	0.0
Красный	1.0	0.0	0.0
Зеленый	0.0	1.0	0.0
Желтый	1.0	1.0	0.0
Синий	0.0	0.0	1.0
Пурпурный	1.0	0.0	1.0
Голубой	0.0	1.0	1.0
Темно-синий	0.25	0.25	0.25
Светло-серый	0.75	0.75	0.75
Белый	1.0	1.0	1.0

«Состояние» OpenGL

`glEnable(GL_LINE_STIPPLE) / glDisable(GL_LINE_STIPPLE)`

Если штриховой шаблон задан, то он будет / не будет применяться для вычерчивания линии после активизации с помощью команды и до тех пор, пока его активизация не будет отменена командой.

`void glLineStipple(GLint factor, GLushort pattern)`- задает шаблон штриховки. Величина `pattern` (которая имеет тип GLushort, 16-битное число без знака) представляет собой последовательность нулей и единиц, которая определяет, какие точки будут рисоваться вдоль линии, 1 означает, что точка рисуется, 0 — что нет.

«Состояние» OpenGL

Шаблон	Множитель	Результирующая штриховка
0xFF00	1	....
0xFF00	2	....
0x5555	1	. . . . .
0x3333	2	.. ..
0x7733	1	... ...

«Состояние» OpenGL

```
glEnable(GL_POLYGON_STIPPLE)
glDisable(GL_POLYGON_STIPPLE)
```

Если шаблон задан, он применяется / не применяется к заполнению полигонов.

*void glPolygonStipple (const GLubyte *mask)* – задает шаблон функцией предварительно подготовив битовую карту размером 32? 32 бит.

«Состояние» OpenGL

Значения переменных состояния OpenGL, можно получить функциями:

- `GLboolean glIsEnabled(GLenum cap),`
- `void glGetIntegerv(GLenump name, GLint *params),`
- `void glGetFloatv(GLenump name, GLfloat *params),`
- `void glGetDoublev(GLenump name, GLdouble *params ),`
- `void glGetBooleanv(GLenump name, GLboolean * params).`

Установка системы координат

Для установка простой системы координат нужно использовать следующие функции:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0, 640.0, 0, 480.0);
```

Функция `glFlush()` должна быть вызвана после того, как точки нарисованы, чтобы убедиться, что все данные должным образом обработаны и отправлены на дисплей.

Законченная программа на OpenGL

```
// Пример рисования 3-х точек
#include <GL/glut.h> // подключение библиотек
#include <GL/gl.h> //в папке include (вашей
программной среды) создать папку GL и поместить файлы
gl.h, glu.h, glut.h
// в папку lib (вашей программной среды) поместить
файлы opengl32.lib, glu32.lib, glut32.lib
// в папку system (вашей ОС) поместить файлы
opengl32.dll, glu32.dll, glut32.dll,glut.dll
```

Законченная программа на OpenGL

```
void display()  // функция обратного вызова для события обновления окна
{
glClear(GL_COLOR_BUFFER_BIT);  // очистка окна цветом фона
 glBegin(GL_POINTS); // рисуем точки
 glVertex2i(100,50);
 glVertex2i(100,100); // координаты вершин
 glVertex2i(50,50);
 glEnd();                // прекращаем рисовать точки
 glFlush(); // отправляем весь вывод на дисплей
}
```

Законченная программа на OpenGL

```
int main(int argc, char** argv)
{
glutInit(&argc, argv);   // инициализирует OpenGL Utility Toolkit
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);      // инициализирует дисплей (GLUT_SINGLE - один дисплейный буфер, GLUT_RGB - задание цвета как сочетание красного, зеленого и синего
```

Законченная программа на OpenGL

```
  glutInitWindowSize(640,480); // задаем ширину и высоту окна при его инициализации
  glutInitWindowPosition(100,150); // задаем координаты верхнего левого угла окна относительно верхнего левого угла экрана
  glutCreateWindow("Primer"); // открываем и отображаем окно с указанным названием
```

Законченная программа на OpenGL

```
  glClearColor(1.0, 1.0, 1.0, 0.0); // цвет фона
(красный, зеленый, синий, прозрачность)
  glColor3f(0.0,0.0,0.0);       // цвет рисования
(красный, зеленый, синий)
  glPointSize(4.0);   // устанавливаем размер точки
(одна точка квадратик в 4 пиксела)
```

Законченная программа на OpenGL

```
  glMatrixMode(GL_PROJECTION);    // установка
  glLoadIdentity(); //   простой системы координат
  gluOrtho2D(0, 640.0, 0, 480.0);
  glutDisplayFunc(display);  //  регистируем  функцию
обновления окна
  glutMainLoop();  //  входим  в  бесконечный  главный
цикл
  return 0;
  }
```