

Лабораторная работа № 1

Тема: построение графических фигур на плоскости

Аппаратно-независимое программирование и OpenGL

Строго OpenGL определяется как программный интерфейс к графической аппаратуре. Этот интерфейс сделан в виде библиотеке функций (Open Graphics Library – OpenGL). Разработчик – фирма Silicon Gragphics. Таким образом, можно сказать что OpenGL очень мобильная и очень эффективная библиотека трехмерной графики и моделирования.

Сейчас OpenGL стал индустриальным стандартом, поддерживается многими операционными системами для разнообразных аппаратных платформ. Приятно, когда возможен одинаковый подход к написанию графических приложений, когда, например, одна и та же программа может быть скомпилирована и запущена в различных графических средах, когда есть гарантия, что на любом дисплее будет получено примерно одинаковое изображение. Такой подход и называется аппаратно-независимым графическим программированием. В OpenGL предлагается программное средство, при котором перенос графической программы требует только установки соответствующих библиотек OpenGL на новой машине, а само приложение не требует никаких изменений и вызывает из новой библиотеки те же самые функции с теми же параметрами. В результате получаются те же графические результаты. Метод создания графики в OpenGL был принят большим количеством компаний, и библиотеки OpenGL существуют для всех значимых графических сред.

Программирование управляемости событиями

Многие современные графические системы являются оконными и показывают на экране несколько перекрывающихся окон одновременно. Пользователь с помощью мыши может перемещать окна по экрану, а также изменять их размер. Общим свойством, которым обладает большая часть оконных

программ, является их управляемость событиями. Это означает, что программы реагируют на различные события, такие как щелчок мышью, нажатие на клавишу клавиатуры или изменение размеров окна на экране. 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. Когда система удаляет событие из очереди, оно просто выполняет функцию обратного вызова, связанную с данным типом событий. Тем программистам, которые привыкли делать программы по принципу «сделай то, затем сделай это», потребуется некоторое переосмысление. Новая структура программ может быть передана словами «не делай ничего, пока не произойдет какое-нибудь событие, а затем сделай определенную вещь».

Функции обратного вызова:

- `glutDisplayFunc(myDisplay)`. Когда система решает, что какое-нибудь окно на экране подлежит перерисовке, она создает событие «redraw» (обновить). Это имеет место при первом открытии окна, а также когда окно вновь становится видимым, когда другое окно перестало его заслонять. В данном случае функция `myDisplay()` зарегистрирована в качестве функции обратного вызова для события обновления.

- `glutReshapeFunc(myReshape)`. Форма экранного окна может быть изменена (reshape) пользователем, обычно путем захвата мышью угла окна и смещения его с помощью мыши (простое перемещение окна не приводит к событию «изменение формы».) В данном случае функция `myReshape()` зарегистрирована с событием изменения формы окна. Как мы увидим дальше, функции `myReshape()` автоматически передаются аргументы, информирующие о новой ширине и высоте изменившего свою форму окна.

– `glutMouseFunc(myMouse)`. Когда нажимают или отпускают одну из кнопок мыши, то возникает событие «мышь» (`mouse`). В данном случае функция `myMouse()` зарегистрирована в качестве функции, которая будет вызываться всякий раз, когда произойдет событие «мышь». Функции `myMouse()` автоматически передаются аргументы, описывающие местоположение мыши, а также вид действия, вызываемого нажатием кнопки мыши.

– `glutKeyboardFunc(myKeyboard)`. Эта команда регистрирует функцию `myKeyboard()` для события, заключающегося в нажатии или отпускании какой-либо клавиши на клавиатуре. Функции `myKeyboard()` автоматически передаются аргументы, сообщающие, какая клавиша была нажата. Для удобства этой функции также передается информация о положении мыши в момент нажатия клавиши.

Если в какой-либо программе мышь не используется, то соответствующую функцию обратного вызова не нужно писать и регистрировать. В этом случае щелчок мыши не окажет на программу никакого действия. Это же верно для программ, не использующих клавиатуру.

Функция `glutMainLoop()`, которую необходимо использовать после регистрации функций обратного вызова, выполняется после того, как программа рисует начальную картину и входит в бесконечный цикл, находясь в котором она просто ждет наступления событий.

Первая задача при создании изображений заключается в открытии экранного окна для рисования. Эта работа является достаточно сложной и зависит от системы. Поскольку функции OpenGL являются аппаратно-независимыми, то в них не предусмотрено поддержки управления окнами определенных систем. Однако в OpenGL Utility Toolkit включены функции для открытия окна в любой используемой вами системе.

– `glutInit(&argc, argv)`. Эта функция инициализирует OpenGL Utility Toolkit. Ее аргументы для передачи информации о командной строке являются стандартными; не всегда есть необходимость их использовать.

– `glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB)`. Эта функция определяет, как должен быть инициализирован дисплей. Встроенные в нее константы `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. Естественно могут быть и другие аргументы. Например, для плавной анимации используют двойную буферизацию.

– `glutInitWindowSize(640,480)`. Эта функция устанавливает, что экранное окно при инициализации должно иметь 640 пикселей в ширину и 480 пикселей в высоту. Во время выполнения программы пользователь может изменять размер этого окна по своему желанию.

– `glutInitWindowPosition(100,150)`. Эта функция устанавливает, что верхний левый угол данного окна должен находиться в 100 пикселях от левого края и в 150 пикселях от верхнего края. Во время выполнения программы пользователь может перемещать это окно, куда пожелает.

– `glutCreateWindow("my first attempt")`. Эта функция фактически открывает и отображает экранное окно, помещая текст заголовка «my first attempt» (моя первая попытка) в строку заголовка (title bar) окна.

Рисование основных графических примитивов

Команды рисования помещаются в функцию обратного вызова, связанную с событием `redraw` (обновление), например, в функцию `myDisplay()`.

Для рисования объектов в OpenGL необходимо передать ему список вершин. Этот список возникает между двумя вызовами OpenGL-функций: `glBegin()` и `glEnd()`. Аргумент в `glBegin()` определяет, какой объект рисуется. Например, можно нарисовать три точки, нарисованные в окне шириной в 640 пикселей и высотой в 480 пикселей. Эти точки рисуются с помощью следующей последовательности команд:

```
glBegin(GL_POINTS);  
glVertex2i(100,50);
```

```
glVertex2i(100,130);
glVertex2i(150,130);
glEnd();
```

GL_POINTS является встроенной константой OpenGL, для рисования точек. Для рисования других примитивов следует заменить GL_POINTS на другие константы:

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

GL_LINE_STRIP используется для рисования ломаной линии (совокупности отрезков прямых, соединенных своими концами). В OpenGL ломаная рисуется посредством задания вершин в нужном порядке, между командами glBegin(GL_LINE_STRIP) и glEnd().

GL_LINE_LOOP используется если нужно соединить последнюю точку с первой, чтобы ломаная линия превратилась в полигон. Полигоны, нарисованные с помощью GL_LINE_LOOP, нельзя заполнять цветом или узором.

GL_POLYGON используется для рисования закрашенных полигонов.

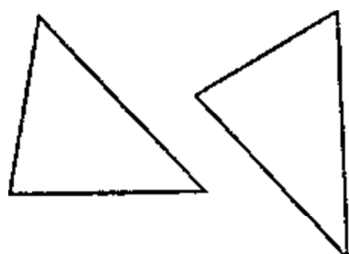
GL_TRIANGLES берет вершины из списка по три за один прием и рисует для каждой тройки отдельный треугольник.

GL_QUADS берет вершины по четыре за один прием и рисует для каждой четверки отдельный четырехугольник.

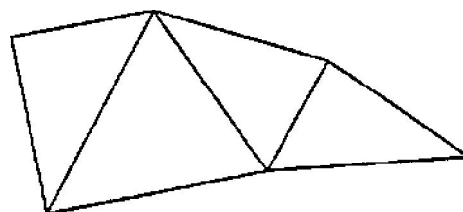
GL_TRIANGLE_STRIP (полоса из треугольников) рисует последовательность треугольников, опирающихся на тройки вершин: v_0, v_1, v_2 , затем v_2, v_1, v_3 , затем v_2, v_3, v_1 и т. д. (порядок следования такой, что все треугольники «кладутся поперек» в одном направлении, например, против часовой стрелки).

GL_TRIANGLE_FAN (веер из треугольников) рисует 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.

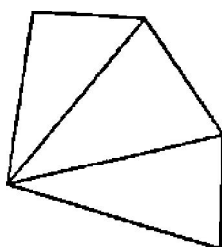
GL_QUAD_STRIP (полоса из четырехугольников) рисует последовательность четырехугольников, опирающихся на четверки вершин: вначале v_0, v_1, v_3, v_2 , затем v_2, v_3, v_5, v_4 , затем v_4, v_5, v_7, v_6 , и т. д. (порядок следования такой, что все четырехугольники «кладутся поперек» в одном направлении,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



GL_TRIANGLES



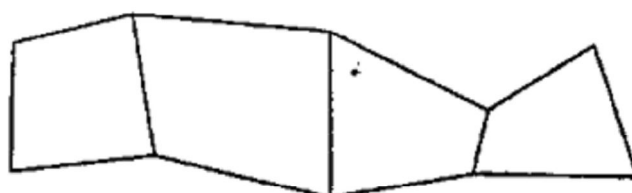
GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN



GL_QUADS



GL_QUAD_STRIP

Рисунок 1.1 - Рисование примитивов

Эти команды посылают информацию о каждой вершине в «графический конвейер» («graphics pipeline»), в котором она проходит ряд преобразований. Во внутреннем представлении OpenGL использует конвейер, последовательно обрабатывающий команды. Команды OpenGL часто выстраиваются в очередь, чтобы потом драйвер OpenGL обработал несколько команд одновременно. Такая схема увеличивает производительность, поскольку связь с аппаратным обеспечением происходит медленно. Функция *glFlush()* сообще-

ет OpenGL что нужно обрабатывать команды рисования, переданных на этот момент и ожидать следующих, поэтому ее следует вызывать после рисования объекта.

Пример законченной программы на OpenGL

```
// Пример рисования 3-х точек
#include <GL/glut.h> // подключение библиотек
#include <GL/gl.h> //в папке include (вашей про-
граммной 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

void display() // функция обратного вызова для со-
бытия обновления окна
{
    glClear(GL_COLOR_BUFFER_BIT); // очистка окна цве-
том 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
}
int main(int argc, char** argv)
{
```

```

    glutInit(&argc, argv);    // инициализирует OpenGL
Utility Toolkit

    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB); // иници-
ализирует дисплей (GLUT_SINGLE - один дисплейный бу-
фер, GLUT_RGB - задание цвета как сочетание красного,
зеленого и синего

    glutInitWindowSize(640,480); // задаем ширину и вы-
соту окна при его инициализации

    glutInitWindowPosition(100,150); // задаем коорди-
наты верхнего левого угла окна относительно верхнего
левого угла экрана

    glutCreateWindow("Primer"); // открываем и отобра-
жаем окно с указанным названием

    glClearColor(1.0, 1.0, 1.0, 0.0); // цвет фона
(красный, зеленый, синий, прозрачность)

    glColor3f(0.0,0.0,0.0);    // цвет рисования (крас-
ный, зеленый, синий)

    glPointSize(4.0);    // устанавливаем размер точки
(одна точка квадратик в 4 пиксела)

    glMatrixMode(GL_PROJECTION);    // установка
glLoadIdentity(); // простой системы координат
gluOrtho2D(0, 640.0, 0, 480.0);

    glutDisplayFunc(display); // регистрируем функцию
обновления окна

    glutMainLoop(); // входим в бесконечный главный
цикл

    return 0;

}

```


Правильные многоугольники

Многоугольник называется правильным, если он является простым, если все его стороны имеют равную длину и, если его смежные стороны образуют равные внутренние углы.

Простым называется многоугольник, если никакие две его стороны не пересекаются (только смежные стороны соприкасаются и только в их общей концевой точке).



Рисунок 1.2 - Примеры правильных n-угольников

Если число сторон n-угольника велико, то этот многоугольник по внешнему виду стремится к окружности. Вершины n-угольника лежат на так называемой порождающей окружности данного n-угольника и их расположение легко вычисляется:

$$x_i = R \cdot \cos(2\pi \cdot i / n), \text{ и } y_i = R \cdot \sin(2\pi \cdot i / n), \quad i = 0, \dots, n-1. \quad (1.1)$$

$$y_i = R \cdot \sin(2\pi \cdot i / n), \quad i = 0, \dots, n-1. \quad (1.2)$$

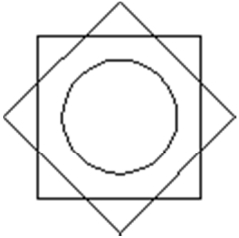

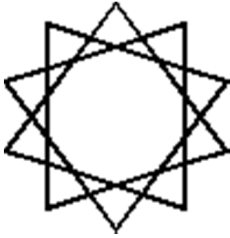

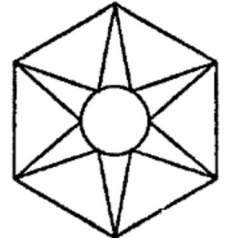
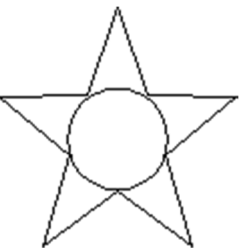


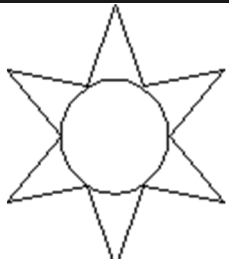
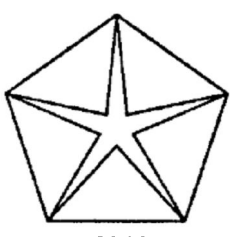
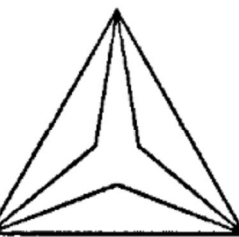

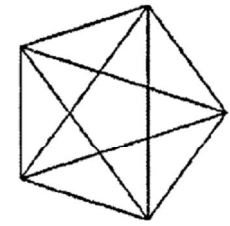
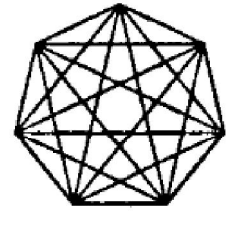
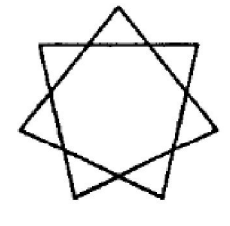
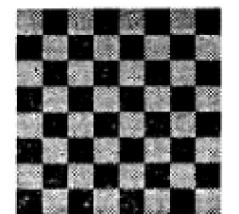



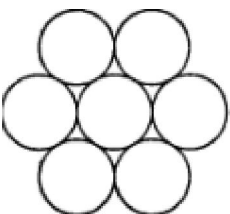
Задание

Написать и отладить программу на языке высокого уровня с использованием функциональных возможностей OpenGL для построения графической фигуры. Для этого:

1. Написать функцию для вывода на экран заданной про варианту графической фигуры (табл. 1.1). В функцию передавать минимально необходимый набор параметров (Н-р: координаты центра или другой точки фигуры и один или два радиуса, в зависимости от вида фигуры).

2. Нарисовать заданную по варианту фигуру (табл. 1.1) в центре экрана.

Таблица 1.1 – Варианты графических фигур

 №1	 №2	 №3	 №4	 №5
 №6	 №7	 №8	 №9	 №10
 №11	 №12	 №13	 №14	 №15
 №16	 №17	 №18	 №19	 №20

Содержание отчета

1. Титульный лист.
2. Задание.
3. Краткие теоретические сведения.
4. Текст программы.

При защите лабораторной работы тестирование программы **обязательно!**