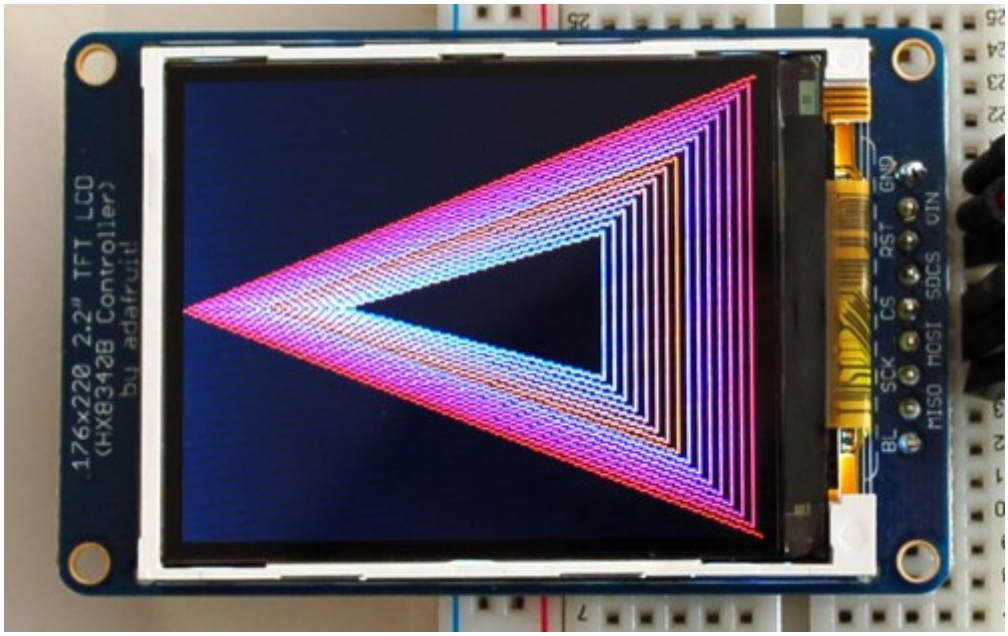
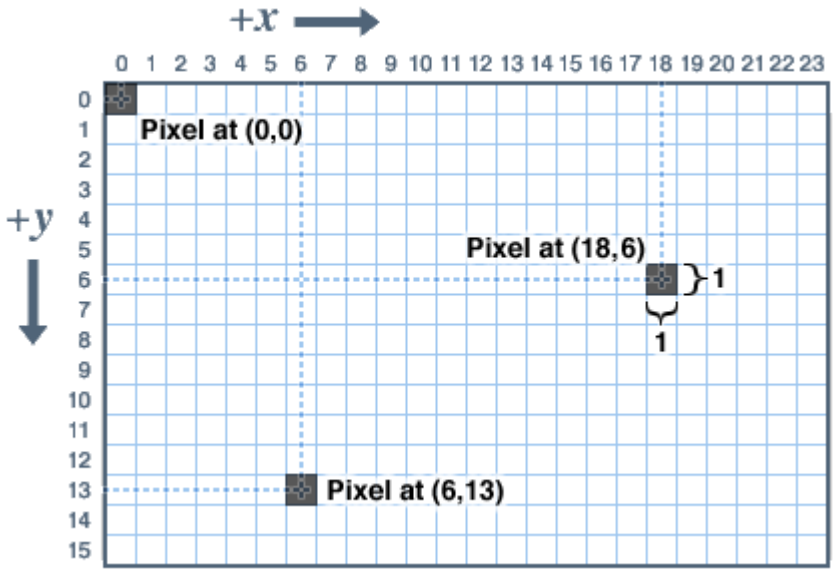
Описание библиотеки Adafruit_GFX для Arduino

Adafruit_GFX – это библиотека для Arduino, имеющая в своем наборе синтаксис и графические функции для LCD и OLED дисплеев. Она позволяет легко адаптировать скетч Ардуино для работы с дисплеем. Библиотека Adafruit_GFX всегда работает в паре со специализированной библиотекой для используемого дисплея. Например, если у вас монохромный дисплей OLED 128x64 с чипом SSD1306, вам нужно установить две библиотеки, Adafruit_GFX и Adafruit_SSD1306. Ссылки на библиотеки для различных дисплеев смотрите в конце статьи.



Система координат:
Пиксели – это блоки из которых состоят цифровые картинки. Пиксель имеет фиксированный размер высоты и длинны (квадрат), размер пикселя зависит от разрешения дисплея. Каждый пиксель позиционируется на экране по горизонтальной оси X и вертикальной оси Y. Система координат начинается в левом верхнем углу экрана с точки X = 0, Y = 0 и продолжается с положительным увеличением оси X вправо, оси Y вниз. Это стандартная Декартовая (прямоугольная) система координат. В зависимости от ориентации дисплея «Портрет», «Пейзаж» или другого специфического расположения, можно указать какой из четырех углов будет обозначен как левый верхний (начальная точка осей).



Цвета:
Для цветных дисплеев, цвета представлены в беззнаковом 16-и битном формате. Некоторые из дисплеев могут передавать цвета с большим количеством битов, но библиотека оперирует только 16-и битными значениями, это упрощает работу Ардуино с различными дисплеями. Цвет складывается из трех основных цветов – красный, зеленый, синий, которые упакованы в одном 16-и битном значении. Первые 5 бит для красного, средние 6 бит для зеленого, последние 5 бит для синего. Человеческий глаз более чувствителен к зеленому цвету, поэтому для него выделено на 1-бит больше.

Для удобства использования большинства основных и вторичных цветов, вы можете добавить в свой код готовый набор. Вы можете использовать любой из 65,536 цветов.

```
// Color definitions
#define BLACK 0x0000
#define BLUE 0x001F
#define RED 0xF800
#define GREEN 0x07E0
#define CYAN 0x07FF
#define MAGENTA 0xF81F
#define YELLOW 0xFFE0
#define WHITE 0xFFFF
```

Для монохромных (одноцветных) дисплеев, цвет всегда определяется как 1 (светится) или 0 (не светится). Смысл значений «светится/не светится», меняется в зависимости от специфики отображения дисплеев. На дисплее с OLED подсветкой, значение “1” будет подсвечивать пиксель. На рефлективных дисплеях, значение 1 сделает цвет черным. Возможны исключения, но по умолчанию, цвет бекграунда при начальной инициализации дисплея имеет значение 0 (чистый, не светится).

Простейшая графика:

В библиотеке для каждого дисплея имеются свои конструкции и функции инициализации. Они описаны в отдельных инструкциях для каждого типа дисплея. Также, их можно найти в заголовочном файле библиотеки дисплея. Далее описана общие графические функции, которые одинаково работают вне зависимости от типа дисплея.

Описанные ниже функции всего лишь прототипы – дисплей может быть объявлен как объект в соответствии со структурой библиотеки используемого устройства. Смотрите пример работы библиотеки для конкретного дисплея. Например, чтобы вывести на экран символы используя команду print(3.14); Код для вашего дисплея будет содержать имя объявленного устройства непосредственно перед командой print. Если объект вашего дисплея объявлен как screen, то ваша строка будет выглядеть так: screen.print(3.14); На примере OLED 128x64 с чипом SSD1306, дисплей объявляется как объект с именем display. Тогда строка вывода на экран будет такая: display.print(3.14);.

Пиксель (точка):

Это самый простой способ нарисовать пиксель (точку) на экране. Вам нужно указать координаты расположения пикселя X, Y и его цвет.

void drawPixel(uint16_t x, uint16_t y, uint16_t color);

В вашем коде, эта строка будет выглядеть так (пример для монохромного OLED дисплея):

display.drawPixel(10, 15, 1);

10 – x координата расположения пикселя на экране,

15 – y координата расположения пикселя на экране,
1 – color цвет пикселя (светится).
Линия:

Для прорисовки линий используются начальная и конечная точка линии.

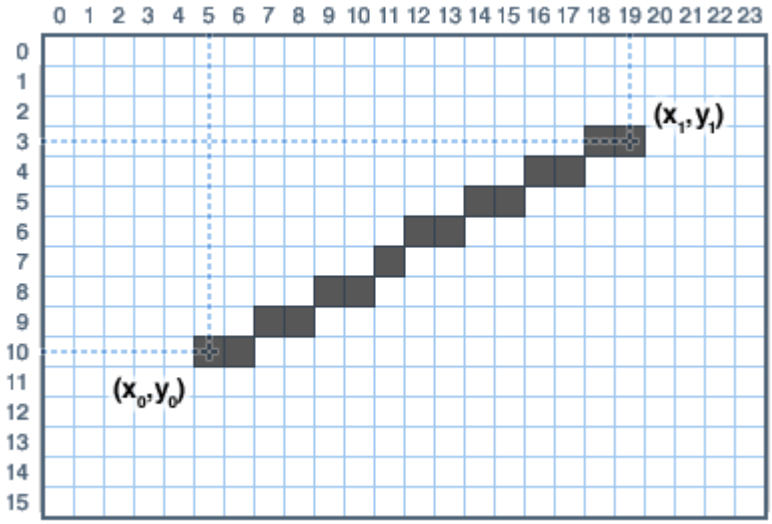
void drawLine(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1, uint16_t color);

В вашем коде, эта строка будет выглядеть так (пример для монохромного OLED дисплея):

display.drawLine (5, 10, 19, 3, 1);

5 – x0 начальная координата линии,

10 – y0 начальная координата линии,
19 – x1 конечная координата линии,
3 – y1 конечная координата линии,
1 – color цвет линии (светится).



Для горизонтальных и вертикальных линий существует более упрощенная функция. В ней указывается начальные координаты линии, её длинна в пикселях и цвет.
void drawFastVLine(uint16_t x0, uint16_t y0, uint16_t length, uint16_t color);

void drawFastHLine(uin86_t x0, uin86_t y0, uint8_t length, uint16_t color);
В вашем коде, строка для вертикальной линии будет выглядеть так (пример для монохромного OLED дисплея):

display.drawFastVLine (7, 12, 20, 1);

7 – x0 начальная координата вертикальной линии,

12 – y0 начальная координата вертикальной линии,
20 – length длинна вертикальной линии в пикселях,
1 – color цвет вертикальной линии (светится).
Прямоугольники:

Для прорисовки прямоугольников используются, координаты верхнего левого угла прямоугольника X, Y, ширина w, высота h, цвет. drawRect() – рисует линию по границам сторон прямоугольника. fillRect() – рисует прямоугольник площадь которого залита указанным цветом.

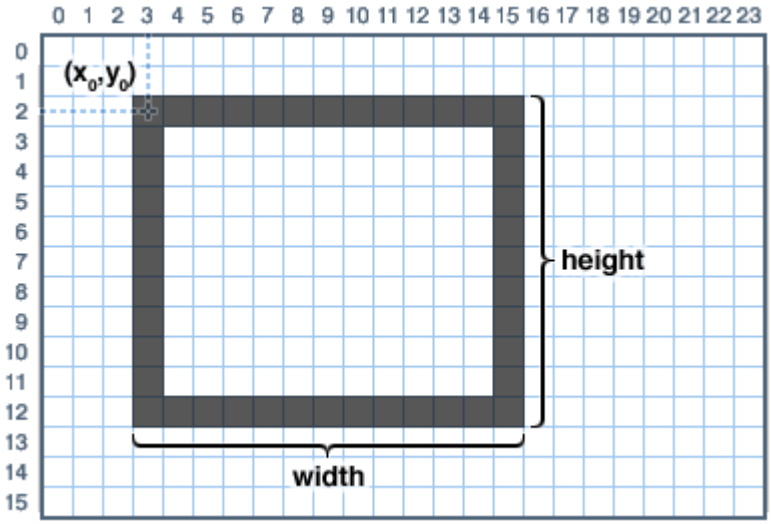
void drawRect(uint16_t x0, uint16_t y0, uint16_t w, uint16_t h, uint16_t color);

void fillRect(uint16_t x0, uint16_t y0, uint16_t w, uint16_t h, uint16_t color);
В вашем коде, строка для прорисовки прямоугольника будет выглядеть так (пример для монохромного OLED дисплея):

display.drawRect (3, 2, 12, 10, 1);

3 – x0 начальная координата прямоугольника,

2 – y0 начальная координата прямоугольника,
12 – w ширина прямоугольника в пикселях,
10 – h высота прямоугольника в пикселях,
1 – color цвет прямоугольника (светится), для функции fillRect() – это цвет заливки площади прямоугольника.



Круг:
Функция прорисовки круга использует следующие значения: центральные координаты круга X, Y, радиус, цвет. drawCircle() – рисует круг. fillCircle – рисует круг, площадь которого залита выбранным цветом.

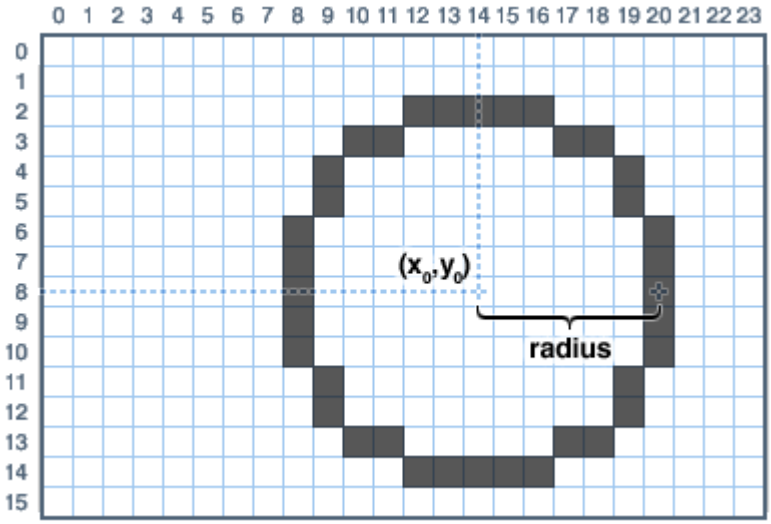
void drawCircle(uint16_t x0, uint16_t y0, uint16_t r, uint16_t color);

void fillCircle(uint16_t x0, uint16_t y0, uint16_t r, uint16_t color);
В вашем коде, строка для прорисовки круга будет выглядеть так (пример для монохромного OLED дисплея):

display.drawCircle (14, 8, 7, 1);

14 – x0 центральная координата круга по оси X,

8 – y0 центральная координата круга по оси Y,
7 – r радиус круга в пикселях,
1 – color цвет круга (светится), для функции fillCircle() – это цвет заливки площади круга.



Прямоугольник со скругленными углами:
Функция прорисовки прямоугольника со скругленными углами использует: начальные координаты левого верхнего угла прямоугольника X, Y, ширину, высоту, радиус скругления угла в пикселях и цвет. drawRoundRect() – прорисовывает прямоугольник со скругленными углами. fillRoundRect() – прорисовывает прямоугольник со скругленными углами, площадь которого залита выбранным цветом.

void drawRoundRect(uint16_t x0, uint16_t y0, uint16_t w, uint16_t h, uint16_t radius, uint16_t color);

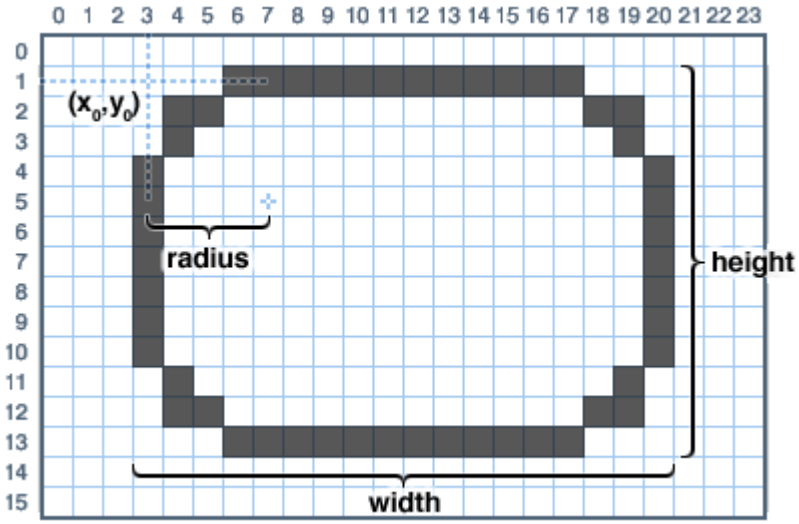
void fillRoundRect(uint16_t x0, uint16_t y0, uint16_t w, uint16_t h, uint16_t radius, uint16_t color);

В вашем коде, строка для прорисовки прямоугольника со скругленными углами будет выглядеть так (пример для монохромного OLED дисплея):

display.drawRoundRect (3, 1, 18, 13, 5, 1);

3 – x0 начальная координата прямоугольника,

1 – y0 начальная координата прямоугольника,
18 – w ширина прямоугольника в пикселях,
13 – h высота прямоугольника в пикселях,
5 – radius радиус скругления углов прямоугольника,
1 – color цвет прямоугольника (светится), для функции fillRoundRect() – это цвет заливки площади прямоугольника.



Использование функции прямоугольника со скругленными углами имеет дополнительную возможность. Если радиус скругления углов сделать равным половине ширины или высоты прямоугольника, то вы сможете получить овал. Радиус скругления углов применяется ко всем четырем углам прямоугольника.
Треугольник:

Функция прорисовки треугольника использует следующие параметры: начальные координаты первого угла, координаты второго угла, координаты третьего угла, цвет. drawTriangle() – прорисовывает треугольник. fillTriangle() – прорисовывает треугольник площадь которого залита выбранным цветом.

void drawTriangle(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint16_t color);

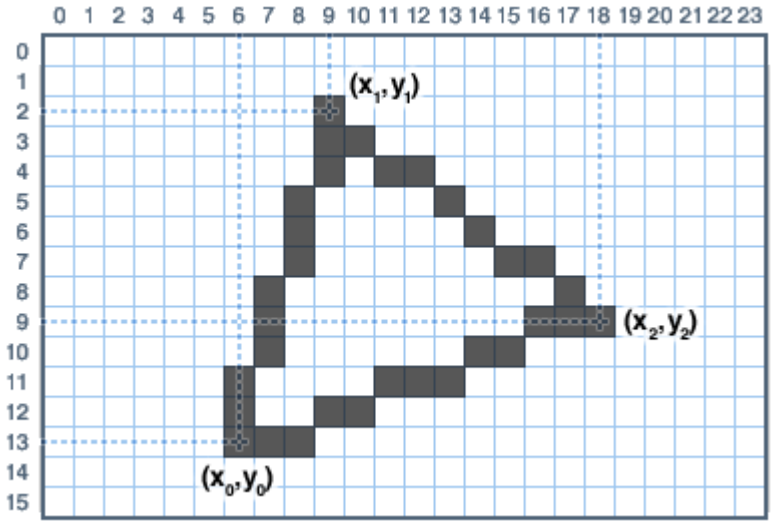
void fillTriangle(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint16_t color);

В вашем коде, строка для прорисовки прямоугольника будет выглядеть так (пример для монохромного OLED дисплея):

display.drawTriangle (6, 13, 9, 2, 18, 9, 1);

7 – x0 координата первого угла,

12 – y0 координата первого угла,
7 – x1 координата второго угла,
12 – y1 координата второго угла,
7 – x2 координата третьего угла,
12 – y2 координата третьего угла,
1 – color цвет треугольника (светится), для функции fillTriangle () – это цвет заливки площади треугольника.



Символы и текст:
В библиотеке имеются две простые функции вывода текста на экран. Первая, для вывода одного единственного символа. Вы можете поместить один символ в любое место экрана и задать этому символу нужный цвет. Основной размер символа 5x8 пикселей. Но размеры можно пропорционально изменять. Например, задав символу размер size=2, размер символа станет 10x16 пикселей. Эта скомпонованная функция сделана для уменьшения программы.

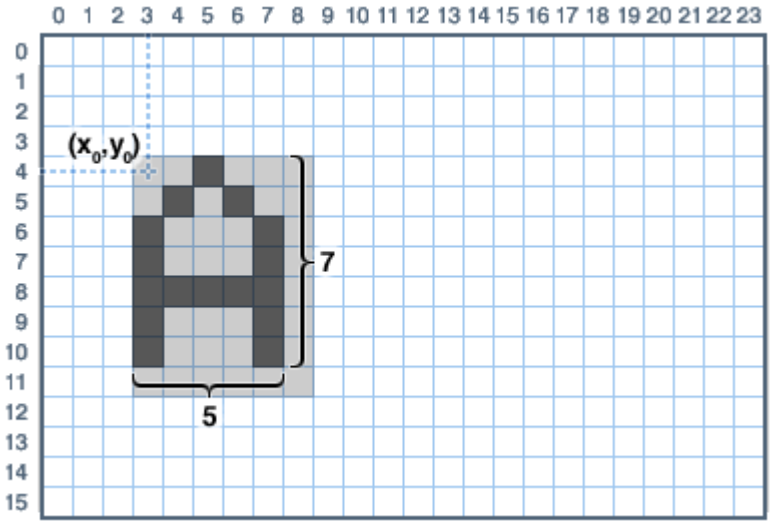
void drawChar(uint16_t x, uint16_t y, char c, uint16_t color, uint16_t bg, uint8_t size);

В вашем коде, строка для одного символа будет выглядеть так (пример для монохромного OLED дисплея):

display.drawChar (3, 4, ‘A’, 1, 0, 1);

3 – x координата символа,

4 – y координата символа,
‘A’ – выводимый символ, должен быть заключен в одинарные ковычки,
1 – color цвет символа,
0 – bg фон (бекграунд) символа,
1 – size размер символа,



Вывод строки символов (текст), отличается от вывода одного символа. Для того, чтобы задать параметры для строки, нужно использовать несколько функций. Задав координаты, цвет и размеры в разных функциях, строка выводится функцией print() которая использует заданные раннее параметры. Такой способ вывода похож на известную вам функции Serial.print(). Ниже приведен набор функций используемых для вывода строк.

```
void setCursor(uint16_t x0, uint16_t y0);
void setTextColor(uint16_t color);
void setTextColor(uint16_t color, uint16_t backgroundcolor);
void setTextSize(uint8_t size);
void setTextWrap(boolean w);
```

В первую очередь, вам нужно задать координаты начала строки, функция setCursor(x, y), задает координаты верхнего левого угла строки. По умолчанию, её значения (0, 0) (верхний левый угол экрана). Затем задаем цвет строки, функция setTextColor(color). По умолчанию, её значение 0 (текст не видим / не светится). Каждый символ в строке имеет свой фон. Фону можно задать цвет дополнительным (не обязательным) вторым параметром функции setTextColor() (цвет фона задается для всей строки символов) setTextColor(color, backgroundcolor); Далее задаем размер символов в строке setTextSize(size). По умолчанию значение size=1, допустимые значения 2 и 3.

После того как вы установили все нужные настройки, вы можете выводить строку символов на экран, используя функцию print() или println(). Точно также, как вы выводите текст при помощи функции Serial.print(). Например, для того, чтобы вывести строку (текст) на экран, используйте print(“Hello world”) – это первая строка на фотографии экрана. Вы также можете выводить цифры и переменные, например, print(1234.56) – это вторая строка на фотографии с экраном. И третья строка, это print(0xDEADBEEF, HEX).
По умолчанию, длинные строки выводятся слева направо, если строка длиннее размеров экрана, то происходит перенос текста на новую строку. За это отвечает функцию setTextWrap(), по умолчанию setTextWrap(1). Если применить setTextWrap(0), то длинная строка не будет переноситься и уйдет за пределы экрана, что подходит для скрола (можно сделать бегущую строку).

Ваш код для вывода строки символа будет выглядеть так (пример для монохромного OLED дисплея):

```
// Устанавливаем курсор (X = 5, Y = 3)
display.setCursor(5, 3);
// Задаем цвет текста (1) (видимый / светится)
display.setTextColor(1);
// Если дополнительно нужно задать цвет фона строки,
// используем ниже идущую функцию
// в этом случае использовать функцию display.setTextColor(1); не нужно.
// Указываем цвет текста и фона строки, в данном примере будет темный текст на белом фоне.
display.setTextColor(0, 1);
// Указываем размер символов в строке (1), каждый символ размером 5x7 пикселей.
display.setTextSize(1);
// Выводим текст на экран, текст будет выведен с указанными выше настройками
display.print(“Hello world”);
```

Bitmaps (растровые изображения):

Вы можете выводить на экран небольшие монохромные (одноцветные) растровые изображения (bitmaps). Функция drawBitmap() хорошо подходит для спрайтов, мини анимации и иконок.

void drawBitmap(int16_t x, int16_t y, uint8_t *bitmap, int16_t w, int16_t h, uint16_t color);

Эта функция выводит блок битов на дисплей, где ‘1’ бит выводит (окрашивает) один пиксель на экране. В то время как ‘0’ бит пропускает (не выводит на экран) пиксель. Координаты X, Y, указывают верхний левый угол с которого начинается прорисовка картинки на экране. w и h, указывают ширину и высоту картинки в пикселях. Bitmap должен быть записан в память устройства, для этого используется PROGMEM директивы. Для ознакомления предлагается прочитать инструкцию по применению PROGMEM (ссылка).

Очистка и заливка экрана:

Функция fillScreen() заливает дисплей выбранным цветом стирая всё что есть на экране. В библиотеках для разных дисплеев могут быть свои функции очистки дисплея.

void fillScreen(uint16_t color);

Поворот экрана:

Вы можете изменять ориентацию выводимой графики и текста на экране. Эта функция не изменяет положение уже выведенных элементов, но изменит расположение системы координат для последующего выведения. Функция setRotation() может применить всего один раз, в разделе setup(). Можно повернуть экран только на 0, 90, 180 или 270 градусов. Остальные углы требуют более мощного оборудования, это делает невозможным их вычисление на Arduino.

void setRotation(uint8_t rotation);

Параметры поворота экрана могут быть 0, 1, 2, 3. Для разных типов экранов, начальное положение (0 градусов) может быть в разных положениях. Меняйте значение в функции setRotation(), чтобы повернуть экран в нужное положение.

В вашем коде, строка для поворота экрана будет выглядеть так (пример для монохромного OLED дисплея):
display.setRotation(1);

1 – rotation угол поворота экрана в градусах: 0 = 0, 1 = 90, 2 = 180, 3 = 270.

Ссылки и информация:

Подробное описание всех функций библиотеки Adafruit_SSD1306 и примеры их применения читайте тут (ссылка скоро появиться).

Описание и технические возможности дисплеев Adafruit ищите на официальном сайте производителя https://www.adafruit.com/

Библиотеки для дисплеев ищите на GitHub Adafruit https://github.com/adafruit (воспользуйтесь поиском).

В статье использовались материалы с сайта https://learn.adafruit.com.

Оригинал статьи https://learn.adafruit.com/adafruit-gfx-graphics-library?view=all.
Возможен не точный перевод! Автор перевода Зайцев Сергей (ZSeregaA).

Это может быть вам интересно: