

Работа с изображениями в языке C++

Бирюков В. А.

October 20, 2022

Система счисления по целочисленному основанию 16.

В качестве цифр этой системы обычно используются цифры от 0 до 9 и латинские буквы от A до F.

Примеры:

$$6 = 0x6$$

$$12 = 0xc$$

$$20 = 0x14$$

$$200 = 0xc8$$

$$255 = 0xff$$

$$256 = 0x100$$

$$1000 = 0x3E8$$

$$1024 = 0x400$$

Коды ASCII в шестнадцатеричной системе

Коды ASCII в шестнадцатеричной системе:

'\0' = 0x00	'0' = 0x30	'A' = 0x41	'a' = 0x61
'\n' = 0x0a	'1' = 0x31	'B' = 0x42	'b' = 0x62
'\r' = 0x0d	'2' = 0x32	'P' = 0x50	'p' = 0x70
' ' = 0x20	'3' = 0x33	'Z' = 0x5A	'z' = 0x7a

Текстовый режим открытия файла:

```
std::ofstream out{"my_file.txt"};  
out << "Cat\nDog";
```

То в файл запишется (Linux):

43 61 74 0a 44 6f 67

Или в файл запишется (Windows):

43 61 74 0d 0a 44 6f 67

Бинарный режим открытия файла:

```
std::ofstream out{"my_file.txt", std::ios::binary};  
out << "Cat\nDog";
```

То в файл запишется (Linux и Windows):

43 61 74 0a 44 6f 67

```
int a = 12345678;  
std::ofstream out{"my_file.txt"};  
out << a;
```

То в файл запишется строка, представляющая число 12345678:

31	32	33	34	35	36	37	38
----	----	----	----	----	----	----	----

```
int a = 12345678; // 12345678 == 0xBC614E
std::ofstream out{"my_file.txt", std::ios::binary};
out.write(reinterpret_cast<const char*>(&a), 4);
```

То в файл запишется байтовое представление числа в памяти:

4E 61 BC 00

Числа типа unsigned char и char воспринимаются оператором << как символы.

```
unsigned char a = 75; // 75 = 0x4b
std::ofstream out{"my_file.txt"};
out << a;
```

То в файл запишется символ К:

4B

Если вы откроете этот файл в текстовом редакторе, то увидите:

К

Чтобы число типа unsigned char воспринимается оператором << как число, нужно привести его к другому целочисленному типу.

```
unsigned char a = 75; // 75 = 0x4b
std::ofstream out{"my_file.txt"};
out << static_cast<int>(a);
```

То в файл запишется символ К:

37 35

Если вы откроете этот файл в текстовом редакторе, то увидите:

75

Числа типа unsigned char и char воспринимаются оператором << как символы.

```
unsigned char a = 75; // 75 = 0x4b
std::ofstream out{"my_file.txt", std::ios::binary};
out.write(reinterpret_cast<const char*>(&a), 1);
```

То в файл запишется байтовое представление числа:

4B

Если вы откроете этот файл в текстовом редакторе, то увидите:

K

Хранение информации о цвете в памяти

- RGB – цветовая модель, описывающая способ кодирования цвета с помощью трёх цветов: красного(R), зелёного(G) и синего(B).
- Чаще всего, в современных компьютерах, каждая компонента цвета кодируется одним байтом.
- Соответственно, значение каждой компоненты цвета кодируется числом из отрезка $[0, 255]$.

Хранение информации о цвете в памяти

(255, 0, 0)
(0, 255, 0)
(0, 0, 255)
(0, 255, 255)
(255, 0, 255)
(255, 255, 0)
(0, 0, 0)
(255, 255, 255)



Хранение информации о цвете в памяти

(13, 19, 33)

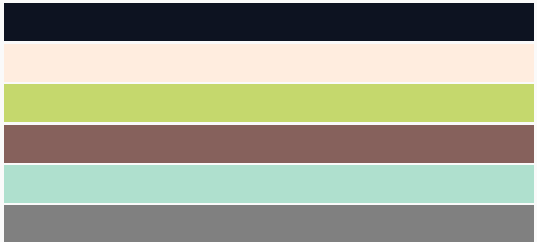
(255, 237, 223)

(197, 216, 109)

(134, 97, 92)

(175, 224, 206)

(128, 128, 128)



Хранение информации о цвете в памяти

0D1321

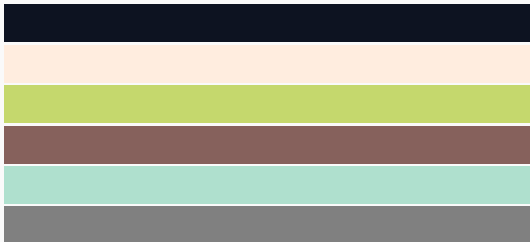
FFEDDF

C5D86D

86615C

AFE0CE

808080



Предположим, что мы хотим хранить в памяти следующий цвет:

`rgb(40, 80, 120) = #285078`



Для этого можно создать массив из трёх элементов:

```
unsigned char a[3] = {40, 80, 120};  
unsigned char b[3] = {0x28, 0x50, 0x78};
```


Предположим, что мы хотим хранить в памяти следующий цвет:

`rgb(40, 80, 120) = #285078`



В языке C++ лучше воспользоваться контейнером `std::array`:

```
std::array<unsigned char, 3> a = {40, 80, 120};  
std::array<unsigned char, 3> b = {0x28, 0x50, 0x78};
```

Предположим, что мы хотим хранить в памяти следующий цвет:

`rgb(40, 80, 120) = #285078`



Можно создать структуру, которая будет хранить компоненты цвета:

```
struct Color
{
    unsigned char r, g, b;
};
//...
Color a = {40, 80, 120};
```

```
#include <iostream>

struct Color
{
    unsigned char r, g, b;
};

int main()
{
    Color a = {40, 80, 120};
    std::cout << a.r << " " << a.g << " "
              << a.b << std::endl;
}
```

На экран напечатается: (P x

```
#include <iostream>

struct Color
{
    std::uint8_t r, g, b;
};

int main()
{
    Color a = {40, 80, 120};
    std::cout << a.r << " " << a.g << " "
               << a.b << std::endl;
}
```

Всё равно напечатается: (P x

```
#include <iostream>

struct Color
{
    unsigned char r, g, b;
};

int main()
{
    Color a = {40, 80, 120};
    std::cout << (int)a.r << " " <<
                (int)a.g << " " <<
                (int)a.b << std::endl;
}
```

На экран напечатается: 40 80 120

```
#include <iostream>

struct Color
{
    unsigned char r, g, b;
};

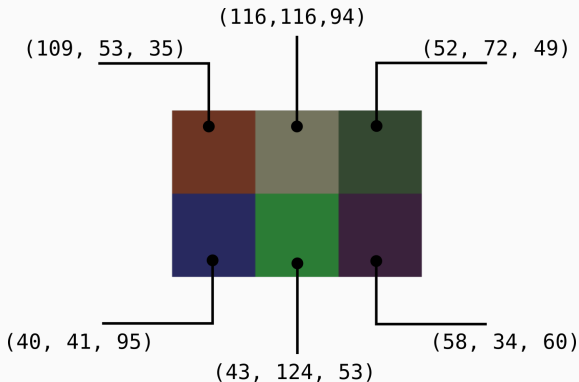
int main()
{
    Color a = {40, 80, 120};
    std::cout << static_cast<int>(a.r) << " " <<
                static_cast<int>(a.g) << " " <<
                static_cast<int>(a.b) << std::endl;
}
```

На экран напечатается: 40 80 120

Хранение изображения в памяти

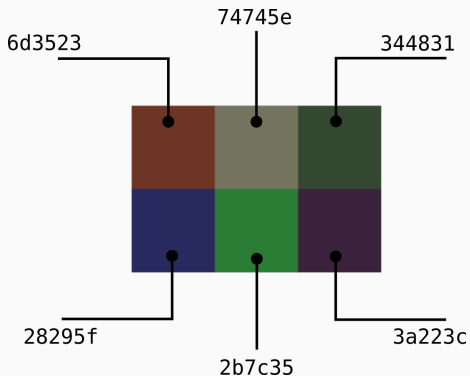
Ширина = 3 пикселя, высота = 2 пикселя

Цвет каждого пикселя кодируется 3-мя байтами

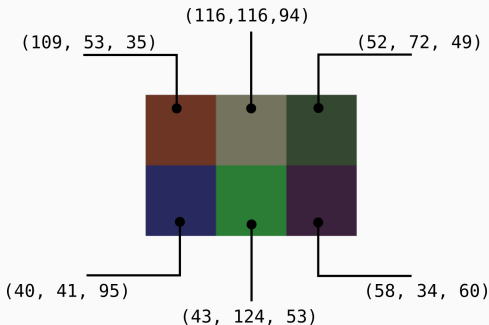


Ширина = 3 пикселя, высота = 2 пикселя

Цвет каждого пикселя кодируется 3-мя байтами

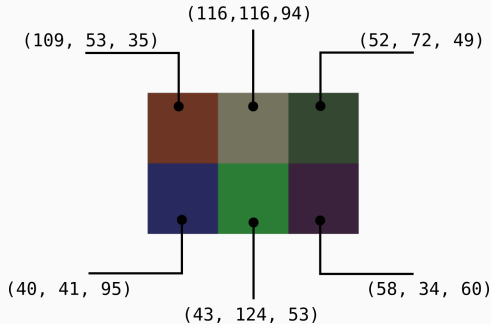


Хранение изображения в памяти(стиль C)



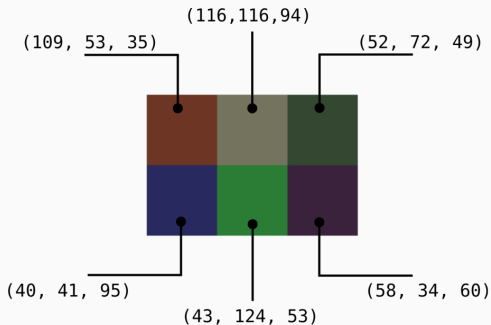
```
int width = 3, height = 2;  
unsigned char data[18] = {109, 53, 35, 116, 116, 94,  
                          52, 72, 49, 40, 41, 95,  
                          43, 124, 53, 58, 34, 60};
```

Хранение изображения в памяти(стиль C++)



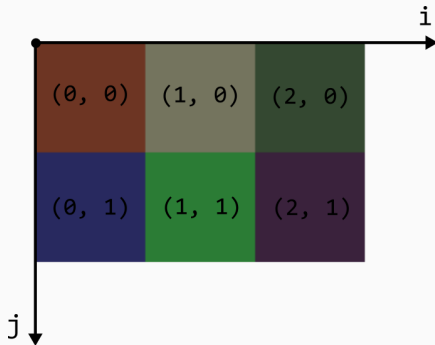
```
int width = 3, height = 2;  
std::vector<unsigned char> v {109, 53, 35, 116, 116, 94,  
                               52, 72, 49, 40, 41, 95,  
                               43, 124, 53, 58, 34, 60};
```

Хранение изображения в памяти



data

109	53	35	116	116	94	52	72	49	40	41	95	43	124	53	58	34	60
-----	----	----	-----	-----	----	----	----	----	----	----	----	----	-----	----	----	----	----



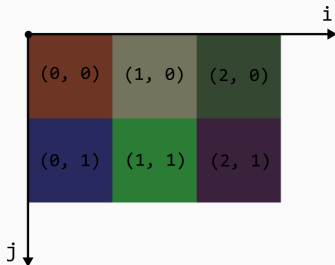
Индекс компоненты цвета пикселя в массиве

```
int width = getImageWidth();  
int height = getImageHeight();  
std::vector<unsigned char> data = getImageData();
```

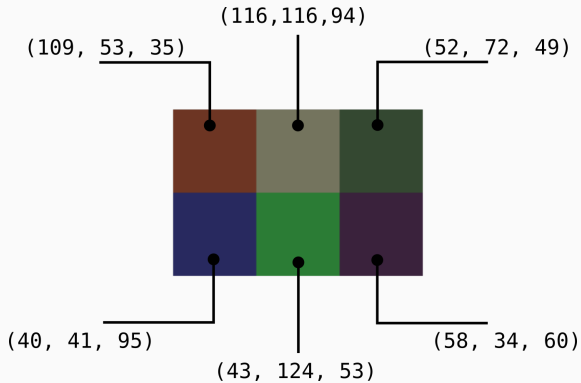
Изменим k -ю компоненту пикселя с координатами (i, j) :

```
data[3 * (i + j * width) + k] = 100;
```

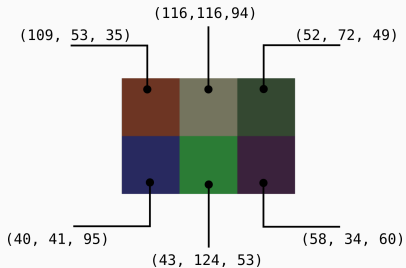
$i \in [0, \text{width} - 1], j \in [0, \text{height} - 1], k \in [0, 2],$



Формат .ppm



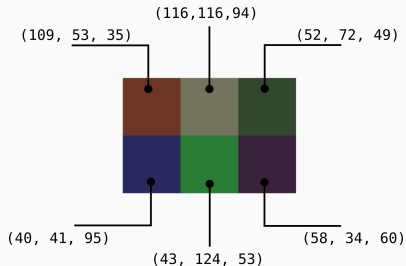
Текстовый формат .ppm изображения



Файл image.ppm

```
P3
3 2
255
109 53 35 116 116 94
52 72 49 40 41 95
43 124 53 58 34 60
```

Текстовый формат .ppm изображения. Побайтово.



50	33	0a	33	20	32	0a	32
35	35	0a	31	30	39	20	35
33	20	33	35	20	0a	31	31
36	20	31	31	36	20	39	34
20	0a	35	32	20	37	32	20
34	39	0a	34	30	20	34	31
20	39	35	20	0a	34	33	20
31	32	34	20	35	33	20	0a
35	38	20	33	34	20	36	30

Запись изображения формата ppm P3

```
const int width = 200;
const int height = 100;

std::ofstream out {"my_image.ppm"};
out << "P3\n" << width << " " << height << "\n255\n";

for (int j = 0; j < height; ++j)
{
    for (int i = 0; i < width; ++i)
        out << 40 << " " << 80 << " " << 120 << "\n";
}
```

```
std::ifstream in {"my_file.ppm"};

std::string type;
in >> type;
if (type != "P3")
{
    std::cout << "Error. Format should be P3\n";
    std::exit(1);
}

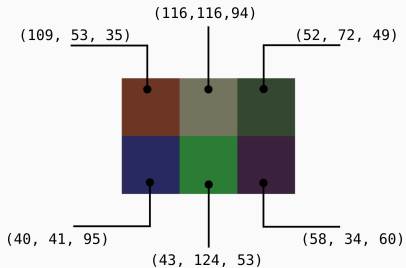
int width, height, maxValue;
in >> width >> height >> maxValue;
...
```

```
...
std::vector<unsigned char> data(width * height);

for (int j = 0; j < height; ++j)
{
    for (int i = 0; i < width; ++i)
    {
        int ri, gi, bi;
        in >> r >> g >> b;
        unsigned char r = ri, g = gi, b = bi;
        data[3 * (j * width + i) + 0] = r;
        data[3 * (j * width + i) + 1] = g;
        data[3 * (j * width + i) + 2] = b;
    }
}
```

- P1: На каждый пиксель приходится 1 бит информации
- P2: На каждый пиксель приходится 1 байт информации
- P3: На каждый пиксель приходится 3 байта информации
- P4: То же самое, что и P1, но в бинарном формате
- P5: То же самое, что и P2, но в бинарном формате
- P6: То же самое, что и P3, но в бинарном формате

Бинарный формат .ppm изображения



Если изображение открыть в текстовом редакторе:

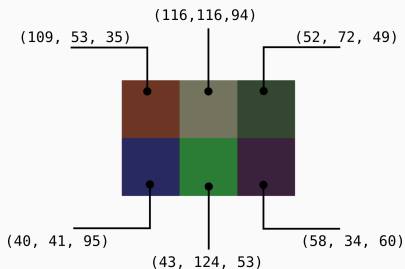
P6

3 2

255

m5#tt^4H1()_|5:"<

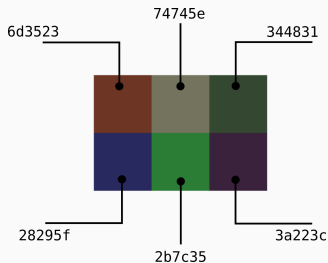
Бинарный формат .ppm изображения. Побайтово.



Байты файла изображения:

50	36	0a	33	20	32	0a	32
35	35	0a	6d	35	23	74	74
5e	34	48	31	28	29	5f	2b
7c	35	3a	22	3c			

Бинарный формат .ppm изображения. Побайтово.



Байты файла изображения:

50	36	0a	33	20	32	0a	32
35	35	0a	6d	35	23	74	74
5e	34	48	31	28	29	5f	2b
7c	35	3a	22	3c			

```
const int width = getWidth();
const int height = getHeight();
std::vector<unsigned char> data = getData();

std::ofstream out {"result.ppm", std::ios::binary};

out << "P6\n" << width << " " << height << "\n255\n";

out.write(reinterpret_cast<const char*>(&data[0]),
    data.size());
```

```
std::ifstream in {"image.ppm", std::ios::binary};
if (in.fail())
{
    std::cout << "Error. Can't open file!\n";
    std::exit(1);
}

std::string type;
in >> type;
if (type != "P6")
{
    std::cout << "Error. File should be type P6\n";
    std::exit(1);
}
...
```

```
...  
int width, height, maxValue;  
in >> width >> height >> in >> maxValue;  
  
char temp;  
in >> std::noskipws >> temp;  
  
std::vector<unsigned char> data(3 * width * height);  
in.read(reinterpret_cast<char*>(&data[0]),  
        data.size());
```

Формат jpeg
