Podstawy przetwarzania obrazu w środowisku Processing (012; 06.08.2009; *processing*)

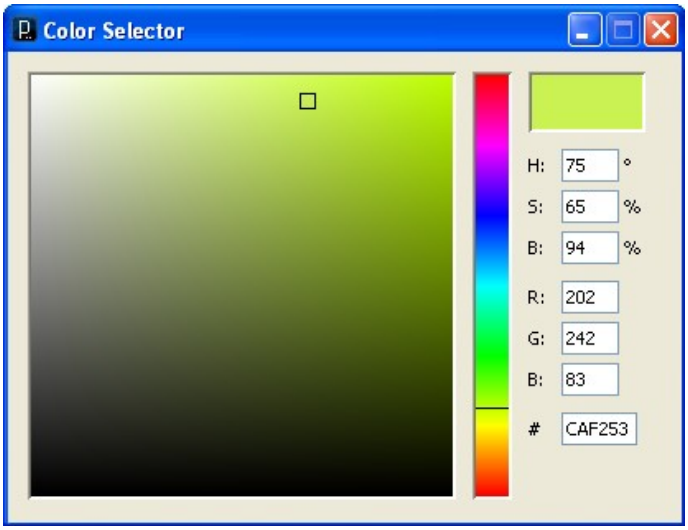
Ze względu na główne zastosowanie środowiska Processing, jakim jest tworzenie obrazu, niezbędna jest znajomość różnych sposobów zarządzania oknem graficznym programu. Przedstawię zatem postawy opisu koloru jako informacji, zarządzanie obrazkami i stosowanie filtrów.

O kolorze.

Podstawową metodą zapisu koloru jest format RGB (*red green blue* – czerwony zielony niebieski). Owe trzy podstawowe kolory mieszając się w różnych proporcjach i ilościach dają możliwość uzyskania wszelkich kolorów (przy czym mieszają się jak światło,zupełnie inaczej niż farby).

Praktycznie informacje o kolorze w formacie RGBA zapisujemy w postaci ciągu 32 bitów danych, przy czym pierwsze 8 bitów odpowiada za przezroczystość koloru (Alpha), kolejne 8 odpowiada za kolor czerwony, następne 8 bitów za zielony i ostatnie 8 bitów koduje informację o kolorze niebieskim. Zatem każdy z kanałów (A, R, G i B) opisany jest liczbą z przedziału zamkniętego 0-255. Mamy zatem do dyspozycji 16581375 kolorów (nie uwzględniając przezroczystości).

Aby praktycznie zobaczyć jaki związek mają wartości składowych R, G i B na kolor możemy uruchomić *Color Selector* z menu *Tools* w programie Processing. Oczywiście podobne edytory koloru możemy znaleźć w każdym programie edycji grafiki (np. w Gimpie).



Innym równie ważnym sposobem opisu koloru jest format HSB. Składowa koloru H (h*ue* – barwa) opisana kątem o wartości z przedziału 0-359 odpowiada za wybór barwy koloru (tak, jakbyśmy poruszali się w poprzek tęczy). Składowa S (s*aturation*) opisana wartością procentową z zakresu 0-99 odpowiada za nasycenie koloru daną barwą. Składowa B (*brightness*) również opisana procentowo wartością z przedziału 0-99 opisuje jasność danego koloru.

Do opisu koloru stosowany bywa również zapis szesnastkowy (heksadecymalny, w skrócie: hex. Używany np. do zapisu kolorów w języku html). Jest to w istocie skrócony zapis formatu RGB. Każdą ze składowych opisujemy liczbą z przedziału 0-255 zapisaną w systemie szesnastkowym (od 00 do FF). Opis koloru w języku java poprzedzany jest dwoma znakami '0x', w Processingu możemy użyć również prefiksu '#'. Warto poczytać więcej tutaj: http://pl.wikipedia.org/wiki/Heksadecymalno%C5%9B%C4%87

Operacje na kolorach.

Zmienną 32 bitową, przechowującą kolor w środowisku Processing deklarujemy poleceniem 'color'. Domyślnie stosowany jest tryb RGBA opisany liczbami z przedziału 0-255. Równoważne sobie, przykładowe sposoby utworzenia koloru czerwonego:

```
color c1                        //deklaracja zmiennej typu kolor
c1 = color(255, 0, 0);          //RGB
c1 = color(255, 0, 0, 0);       //RGBA
c1 = color(#FF0000);            //Hex
c1 = color(#FF0000, 255);       //HexA
c1 = #FF0000;
```

Jeśli chcemy używać kolorów w skali szarości (ale zapisanych jako RGB) możemy użyć poleceń color(gray) lub color(gray, alpha), gdzie gray jest wartością z przedziału 0-255.

Jeśli interesuje nas wyłącznie wartość składowej koloru, możemy użyć funkcji: red(c2), green(c2), blue(c2), alfa(c2), hue(c2), saturation(c2) lub brightness(c2), gdzie c2 jest zmienną typu kolor.

Tryby pracy z kolorami możemy zmienić stosując polecenia: colorMode(RGB) lub colorMode(HSB). Jeśli domyślne zakresy zmian wartości nam nie odpowiadają (zawsze 0-255), możemy podać własne górne granice zakresów (dolne zawsze równe zero) następującym poleceniem: colorMode(mode, range1, range2, range3, range4). Pamiętać należy jednak, że może spowodować to utratę danych na skutek przybliżeń w dopasowywaniu zakresów.

Aby zwrócić uwagę na formę zapisu koloru, zastosujmy następujący kod wysyłający kolor na konsolę w różnej postaci:

```
color c3 = color(250, 102, 27);
println(c3);
println(binary(c3));
println(hex(c3));
println(red(c));
println(green(c));
println(blue(c));
println(hue(c));
println(saturation(c));
println(brightness(c));
```

W konsoli wyświetliły nam się następujące wartości:

```
-367077
11111111111101001100011000011011
FFFA661B
250.0
102.0
27.0
14.29372
227.46
250.0
```

Zwróćmy uwagę, że ostatnie 6 poleceń zwraca wartości zmiennoprzecinkowe (float), a pierwsze polecenie zwraca liczbę ujemną.

Gdy zależy nam, aby polecenia wykonywały się szybciej, do zarzadzania kolorami możemy użyć operacji bezpośrednio na bitach danych. Przykładowy kod służący do stworzenia koloru tą metodą:

```
int a = 255;   // binarnie: 00000000000000000000000011111111
int r = 148;   // binarnie: 00000000000000000000000010010100
int g = 204;   // binarnie: 00000000000000000000000011001100
int b = 75;    // binarnie: 00000000000000000000000001001011
a = a << 24;   // binarnie: 11111111000000000000000000000000
r = r << 16;   // binarnie: 00000000100101000000000000000000
g = g << 8;    // binarnie: 00000000000000001100110000000000
color c4 = a|r|g|b;  //bin: 11111111100101001100110001001011
```

Jeśli chcemy z koloru wydobyć składowe R G i B możemy użyć następujących poleceń:

```
color c5 = color(148, 204, 75, 255);
                    // bin: 11111111100101001100110001001011
int a = (c5 >> 24) & 0xFF;
                    // bin: 00000000000000000000000011111111
int r = (c5 >> 16) & 0xFF;
                    // bin: 00000000000000000000000010010100
int g = (c5 >> 8) & 0xFF;
                    // bin: 00000000000000000000000011001100
int b = c5 & 0xFF;
                    // bin: 00000000000000000000000001001011
```

Przy czym 0xFF jest liczbą o zapisie binarnym: 00000000000000000000000011111111.

Zastosowane operacje na bitach:

<< (left shift) – przesunięcie bitów danych w zmiennej integer o zadaną wartość w lewo z uzupełnieniem wolnych pól po prawej zerami. Przesunięcie o jeden bit powoduje wymnożenie liczby przez 2.

>> (right shift) – przesunięcie bitów danych w zmiennej integer o zadaną wartość w prawo z uzupełnieniem powstałych wolnych pól z lewej zerami. Przesunięcie o jeden bit powoduje podzielenie liczby przez 2.

& (bitwise AND) – operacja logiczna 'i' z zastosowaniem do zapisu binarnego.

| (bitwise OR) – operacja logiczna 'albo' z zastosowaniem do zapisu binarnego.

Nadawanie koloru punktom obszaru roboczego.

Height – funkcja zwracająca wartość wysokości obszaru roboczego okna programu.
Width – funkcja zwracająca wartość szerokości obszaru roboczego okna programu.

Liczbę punktów obszaru roboczego liczymy oczywiście mnożąc wartości height i width.

Polecenie set(x, y, c6) powoduje nadanie punktowi obszaru roboczego o współrzędnych (x,y) koloru przechowywanego w zmiennej c6 typu color (przypominam, że współrzędne numerowane są od warości zero). Funkcją do pobierania koloru punktu obszaru roboczego o współrzędnych (x,y) jest get(x, y). Możemy również pobrać prostokątny wycinek punktów obszaru poleceniem get(x, y, width, height).

Pixels[] – wektor danych przechowujący zmienne kolorów wszystkich pikseli obszaru roboczego. Pisele numerowane są kolejno od lewej do prawej w wierszach jednym ciągiem dla kolejnych wierszy, przy czym numeracja rozpoczyna się od zera i kończy na wartości liczby punktów obszaru roboczego pomniejszonej o jeden. Zanim jednak zaczniemy korzystać z polecenia pixels[] musimy wczytać do tego wektora aktualny stan obszaru roboczego, do czego służy funkcja: loadPixels().

Odczytanie i przypisanie do zmiennej c7 koloru z 230 piksela obszaru roboczego realizujemy poleceniem:
color c7 = pixels[230].

Nadanie pikselowi o numerze 125 koloru c8 realizujemy za pomocą polecenia:
pixels[125] = c8.

Gdy dokonamy zmiany wartości kolorów pikseli w wektorze pixels[] musimy przenieść te dane na obszar roboczy – wywołujemy funkcję updatePixels().

Polecenie get(x, y) warto zastępować szybszym pixels[y*width+x].

Wszystkie opisane wyżej polecenia bazują na zapisie koloru w formacie RGBA.

Wstawienie obrazka.

Następujący kod deklaruje zmienną typu PImage o nazwie 'obrazek' i zawierającą podany plik: PImage obrazek = loadImage("nazwa.jpg"). Polecenie loadImage(url, "png") powoduje wczytanie obrazka typu png z adresu url.

Wstawienie obrazka za lewy górny róg w punkcie obszaru roboczego o współrzędnej (x,y) realizuje polecenie: image(obrazek, x, y). Jeśli nie planujemy dalszej pracy na obrazku jako zmiennej możemy użyć szybszego polecenia przenoszącego obrazek na obszar roboczy: set(x, y, obrazek).

Tworzenie nowego pustego obrazka zrealizuje poleceniem:
PImage nowyObrazek = createImage(100, 100, RGB).

Edycja i zapis obrazu.

Processing udostępnia również inne ważne polecenia do pracy na obrazkach, a mianowicie możliwość kopiowania – copy(), tworzenie maski – mask(), mieszanie – Blend(), zmianę rozmiaru obrazka – obrazek.resize(wide, high) i inne.

Zapis obrazka ze zmiennej do pliku: obrazek.save("obrazek.jpg").

Zapis aktualnego stanu okna do pliku: saveFrame("screeen####.png"), przy czym dla każdego kolejnego pliku tworzona będzie nowa nazwa z kolejnym numerem czterocyfrowym.

Filtry.

Aby dla danego obrazka zastosować filtr używamy polecenia: img.filter(MODE) lub img.filter(MODE, par), gdzie MODE jest jednym z typów filtrów opisanych poniżej, par jest parametrem danego filtru, a img jest nazwą zmiennej obrazka.

THRESHOLD – przetwarza obraz z użyciem koloru białego i czarnego, przy czym czarne stają się piksele o jasności mniejszej niż ustalona wartość parametru (z przedziału 0.0 – 1.0), a pozostałe będą białe.

GRAY – przetwarza kolory obrazu na skalę szarości.

INVERT – tworzy negatyw obrazu.

POSTERIZE – ograniczenie liczby kolorów każdego kanału, prametr ustalamy na wartość z przedziału 2-255.

BLUR – rozmycie obrazu tym większe, im większa wartość parametru (jednocześnie rośnie czas wykonywania operacji nałożenia filtru)

OPAQUE – powoduje, że kanał alfa obrazu staje się nieprzezroczysty.

Oczywiście prosto możemy tworzyć własne filtry, bazując na pętli przetwarzającej kolor każdego piksela obszaru roboczego.