Wprowadzenie do p5.js

Krzysztof Gdawiec

UNIWERSYTET ŚLĄSKI
INSTYTUT INFORMATYKI

Czym jest p5.js?

p5.js jest biblioteką JavaScript nakierowaną na kreatywne programowanie. Autorzy biblioteki położyli duży nacisk, aby biblioteka była łatwa w użyciu dla artystów, projektantów, początkujących programistów itp.

Bibliotekę można ściągnąć ze strony domowej:

https://p5js.org/

API biblioteki jest silnie wzorowane na API Processingu. Zatem znając Processing bardzo łatwo jest się przesiąść na p5.js.

Do pracy nad projektami p5.js możemy wykorzystać edytor udostępniany przez twórców biblioteki pod adresem:

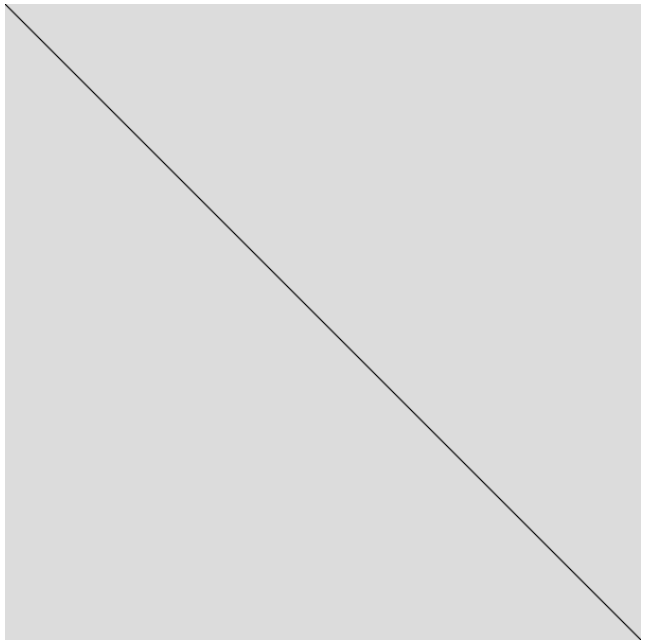
https://editor.p5js.org/

Jeśli chcemy mieć możliwość zapisywania projektów, to musimy założyć konto na stronie edytora.

W przypadku, gdy nie chcemy korzystać z tego edytora, to możemy wykorzystać dowolny edytor tekstowy. W przypadku Visual Studio Code istnieją rozszerzenia ułatwiające pracę z p5.js, np. p5.vscode.

Prosty skrypt p5.js

```
1  function setup()
2  {
3    createCanvas( 600, 600 );
4  }
5
6  function draw()
7  {
8    background( 220 );
9
10   line( 0, 0, width, height );
11 }
```



Podobnie jak w Processingu w p5.js wyróżniamy dwie podstawowe funkcje:

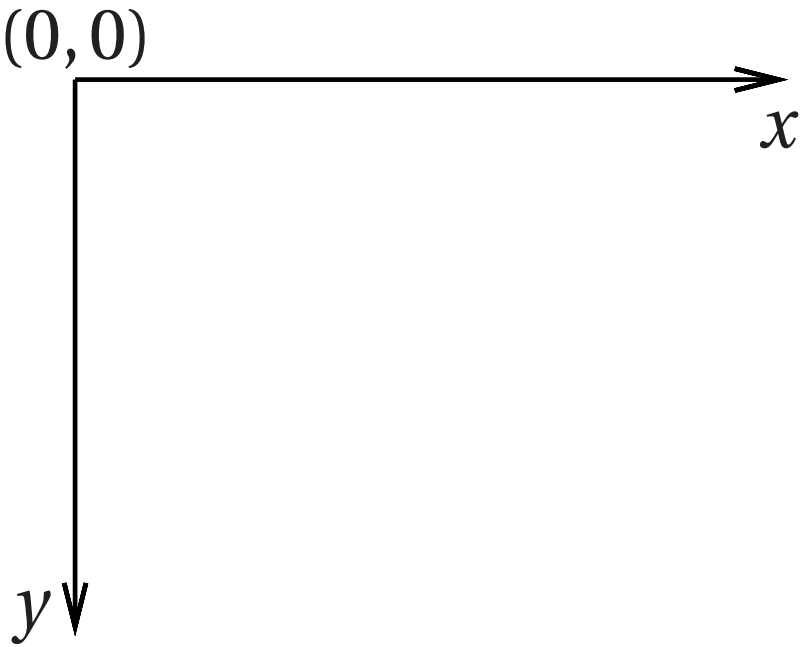
- `setup` – funkcja uruchamiana raz podczas startu programu. Używana jest do inicjowania początkowych ustawień takich jak wielkość elementu canvas;
- `draw` – funkcja uruchamiana zaraz po `setup` i wykonywana w pętli. Jest to główna funkcja programu, w której umieszczamy cały kod renderujący.

Oprócz funkcji `setup` i `draw` w p5.js mamy jeszcze funkcję `preload`, która uruchamiana jest przed `setup`. Jej głównym zadaniem jest ładowanie asynchroniczne plików zewnętrznych, np. obrazów. Jeśli zaimplementujemy funkcję `preload`, to funkcja `setup` będzie czekać na jej zakończenie.

Funkcja `createCanvas` jak sama nazwa wskazuje tworzy element canvas o zadanej szerokości i wysokości (w pikselach). Opcjonalnym, trzecim, argumentem tej funkcji jest sposób renderowania: `P2D`, `WEBGL`. Domyślnym sposobem renderowania jest `P2D`.

Grafika 2D w p5.js

Układ współrzędnych



Punkt

```
point( x, y )
point( vec )
```

`x`, `y` – współrzędne punktu, `vec` – współrzędne punktu dane za pomocą wektora, który jest instancją `p5.Vector`.

Odcinek

```
line( x1, y1, x2, y2 )
```

`x1`, `y1` – współrzędne początku, `x2`, `y2` – współrzędne końca.

Trójkąt

`triangle( x1, y1, x2, y2, x3, y3 )`

`x1, y1, x2, y2, x3, y3` – współrzędne wierzchołków trójkąta.

Czworokąt

`quad( x1, y1, x2, y2, x3, y3, x4, y4 )`

`x1, y1, x2, y2, x3, y3, x4, y4` – współrzędne wierzchołków czworokąta.

Prostokąt

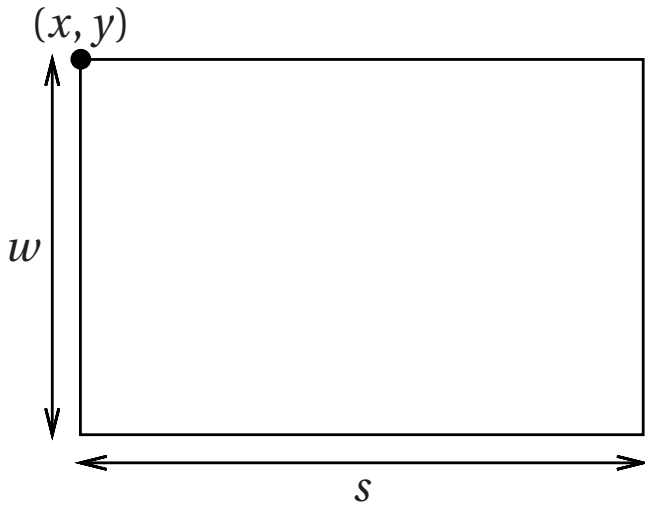
```
rect( x, y, s, w )
```

Interpretacja parametrów funkcji `rect` zależy od ustawionego trybu rysowania. Do zmiany trybu rysowania służy funkcja:

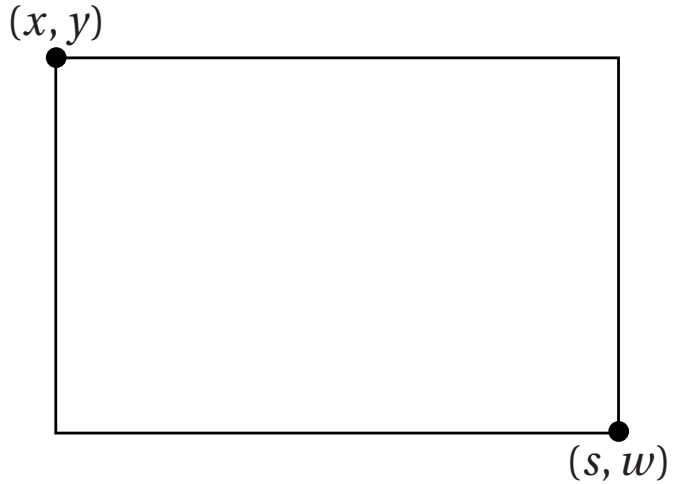
```
rectMode( mode )
```

`mode` przyjmuje jedną z wartości: `CORNER` (tryb domyślny), `CORNERS`, `CENTER`, `RADIUS`.

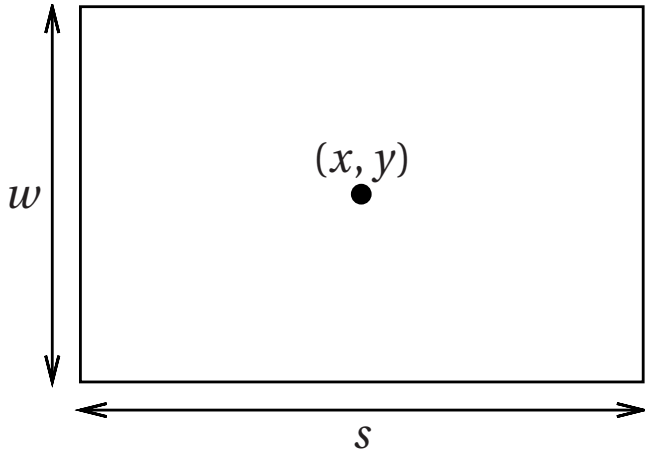
CORNER



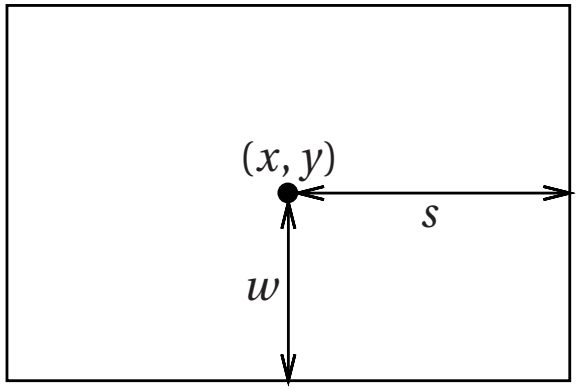
CORNERS



CENTER



RADIUS



Istnieje jeszcze funkcji `rect` rysująca prostokąt o zaokrąglonych rogach

```
rect( x, y, s, w, gl, gp, dp, dl )
```

`gl`, `gp`, `dp`, `dl` promienie zaokrąglenia odpowiednio górnego-lewego, górnego-prawego, dolnego-prawego, dolnego-lewego rogu.

Oprócz prostokąta w p5.js istnieje jeszcze funkcja rysująca kwadrat – `square` (patrz dokumentacja p5.js).

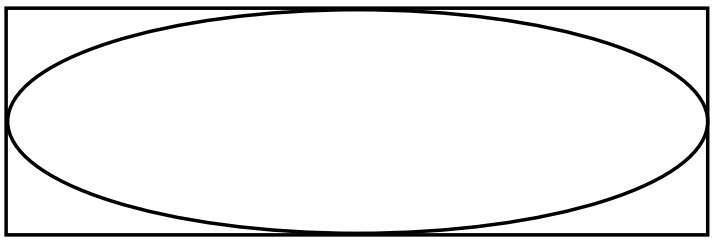
Elipsa

```
ellipse( x, y, s, w )
```

Podobnie jak w przypadku funkcji `rect` interpretacja parametrów zależy od trybu, który ustawiany jest za pomocą funkcji:

```
ellipseMode( mode )
```

`mode` przyjmuje jedną z wartości: `CORNER`, `CORNERS`, `CENTER` (tryb domyślny), `RADIUS`. Interpretacja trybów jest taka sama jak w przypadku funkcji `rectMode` tylko, że dotyczy prostokąta opisanego na elipsie.



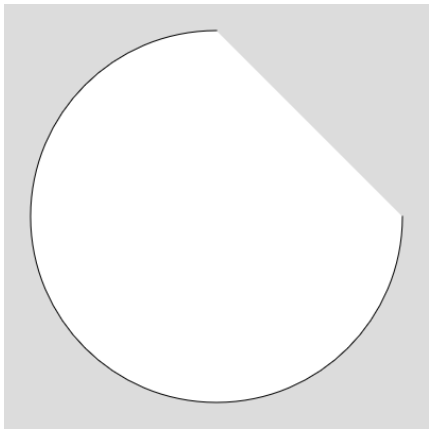
Dodatkowo w p5.js mamy jeszcze funkcję rysującą koło – `circle` (patrz dokumentacja p5.js).

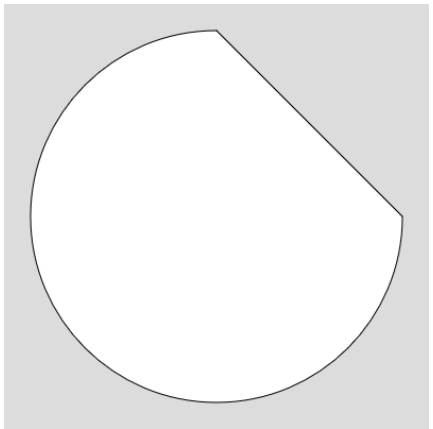
Łuk (wycinek elipsy)

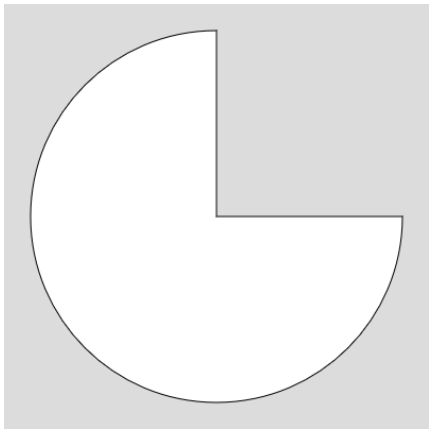
```
arc( x, y, s, w, start, stop )
```

Pierwsze cztery parametry mają te same znaczenie jak w przypadku funkcji `ellipse`, a ich interpretację można zmienić za pomocą funkcji `ellipseMode`. `start` – początkowy kąt łuku (w radianach), `stop` – końcowy kąt łuku (w radianach).

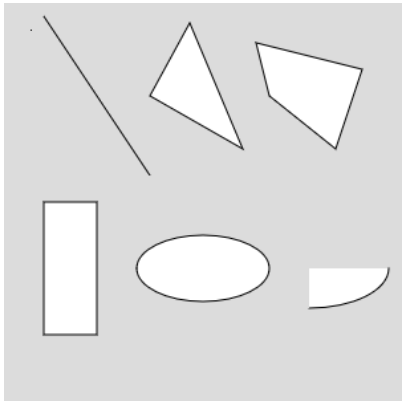
Istnieje jeszcze jeden opcjonalny argument, który określa sposób rysowania łuku, tzn. `OPEN` (domyślny sposób), `CHORD`, `PIE`.







```
1   function setup()
2   {
3     createCanvas( 300, 300 );
4   }
5
6   function draw()
7   {
8     background( 220 );
9
10    point( 20, 20 );
11    line( 30, 10, 110, 130 );
12    triangle( 140, 15, 110, 70, 180, 110 );
13    quad( 190, 30, 270, 50, 250, 110, 200,
          70 );
14
15    rectMode( CENTER );
16    rect( 50, 200, 40, 100 );
17
18    ellipse( 150, 200, 100, 50 );
19    arc( 230, 200, 120, 60, 0, HALF_PI );
20  }
```



Dowolne wielokąty

Rysowanie dowolnych wielokątów i łamanych otrzymujemy za pomocą funkcji:

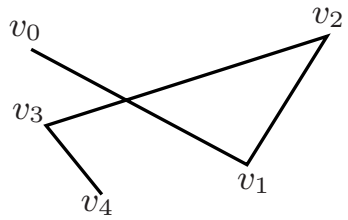
```
beginShape()

beginShape( mode )

vertex( x, y )

endShape()

endShape( mode )
```

Każde wywołanie funkcji `beginShape` musi posiadać odpowiadające wywołanie funkcji `endShape`. Pomiędzy tymi funkcjami umieszczamy wywołania funkcji `vertex`, które definiują kolejne wierzchołki kształtu.
W zależności od wartości parametru `mode` otrzymamy inny kształt.

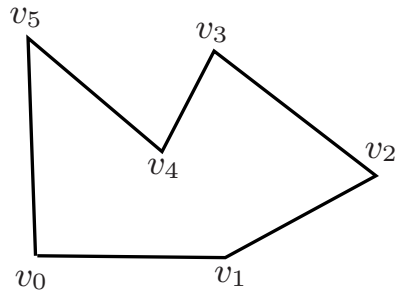
Parametr `mode` funkcji `beginShape` może przyjąć następujące wartości:

- `POINTS` – każde wywołanie `vertex` definiuje jeden punkt,
- `LINES` – każda para `vertex` definiuje jedną linię,
- `TRIANGLES` – każda trójka `vertex` definiuje jeden trójkąt,
- `TRIANGLE_STRIP` – pasek trójkątów,
- `TRIANGLE_FAN` – wachlarz trójkątów,
- `QUADS` – każda czwórka `vertex` definiuje jeden czworokąt,
- `QUAD_STRIP` – pasek czworokątów.

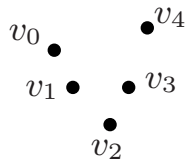
Parametr `mode` funkcji `endShape` może przyjąć tylko wartość `CLOSE` i oznacza domknięcie kształtu.



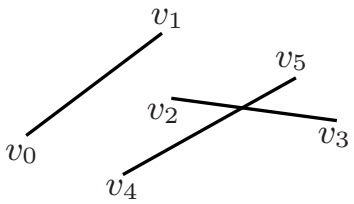
bez parametrów



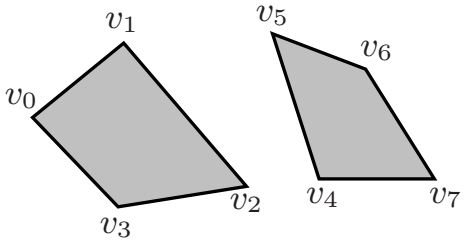
CLOSE



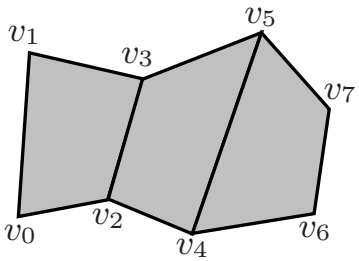
POINTS



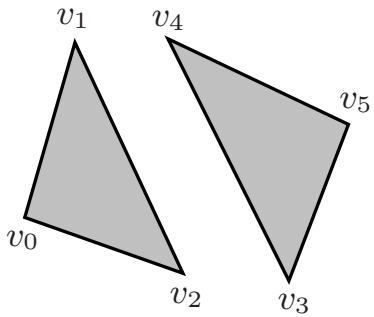
LINES



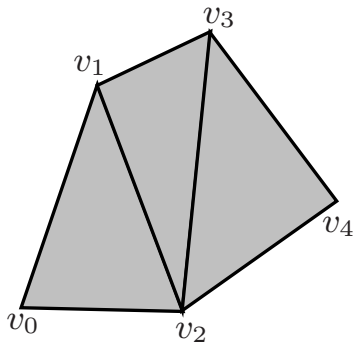
QUADS



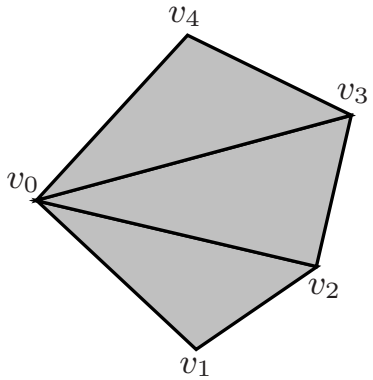
QUAD_STRIP



TRIANGLES

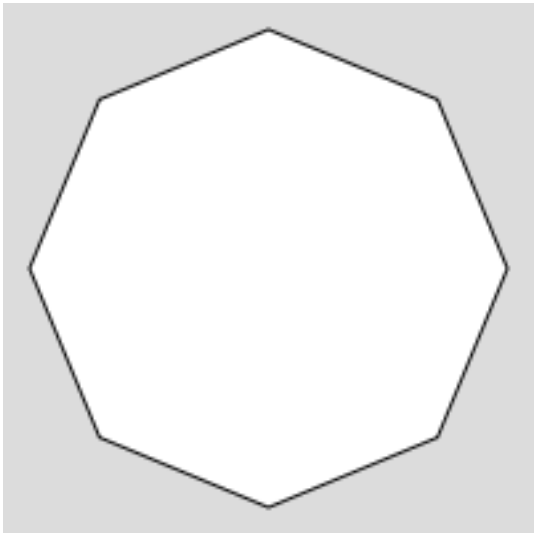


TRAINGLE_STRIP



TRIANGLE_FAN

```
1   function setup()
2   {
3     createCanvas( 200, 200 );
4   }
5
6   function draw()
7   {
8     background( 220 );
9
10    let n = 8;
11    let r = 90;
12
13    beginShape();
14    for(let i = 0; i < n; ++i)
15    {
16      let a = i * TWO_PI / n;
17
18      vertex( width/2 + r * cos(a),
              height/2 + r * sin(a) );
19    }
20    endShape( CLOSE );
21  }
```

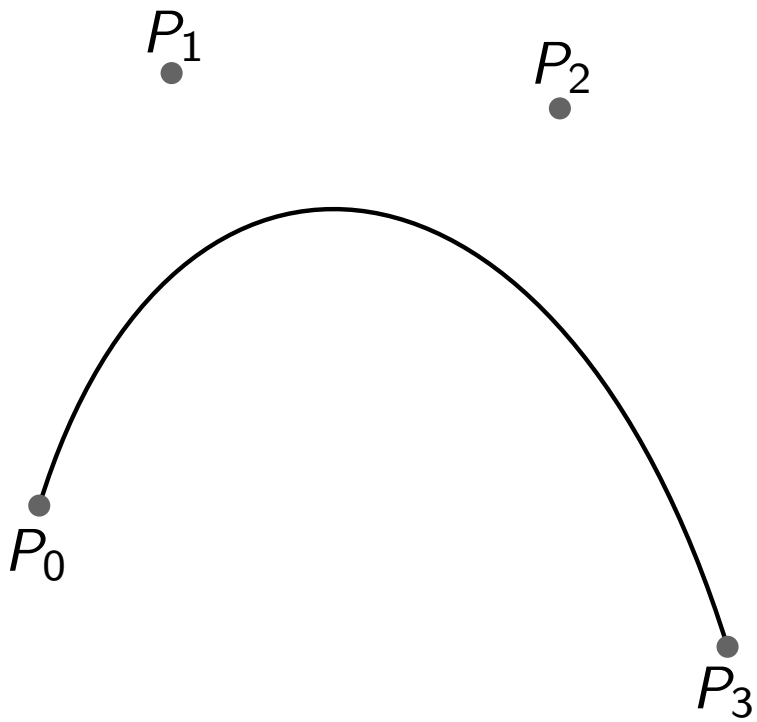


Krzywe

W p5.js mamy do dyspozycji kilka rodzajów krzywych:

- kwadratowe krzywe Béziera,
- kubiczne krzywe Béziera,
- kubiczne krzywe Catmulla-Roma.

Kubiczna krzywa Béziera



p5.js umożliwia rysowanie kubicznych krzywych Béziera na dwa sposoby. Pierwszym jest użycie funkcji:

```
bezier( x1, y1, cx1, cy1, cx2, cy2, x2, y2 )
```

gdzie poszczególne parametry to współrzędne punktów kontrolnych krzywej.

Drugi sposób rysowania kubicznych krzywych Béziera oparty jest na funkcjach `beginShape`, `endShape`. Obie te funkcje wywołujemy bez parametrów, a pomiędzy nimi umieszczamy wywołania funkcji:

```
bezierVertex( cx1, cy1, cx2, cy2, x2, y2 )
```

gdzie poszczególne parametry to współrzędne drugiego, trzeciego i czwartego punktu kontrolnego.

Widzimy, że brakuje pierwszego punktu kontrolnego. W przypadku pierwszej krzywej punkt ten musimy postawić za pomocą funkcji `vertex`, zaś przy kolejnych wywołaniach `bezierVertex` jako pierwszy punkt kontrolny przyjmowany jest ostatni punkt kontrolny poprzedniej krzywej.

W p5.js mamy też możliwość wyznaczenia współrzędnych punktu krzywej oraz stycznej do krzywej dla danej wartości parametru t. Służą do tego funkcje:

- wyznaczanie punktu krzywej

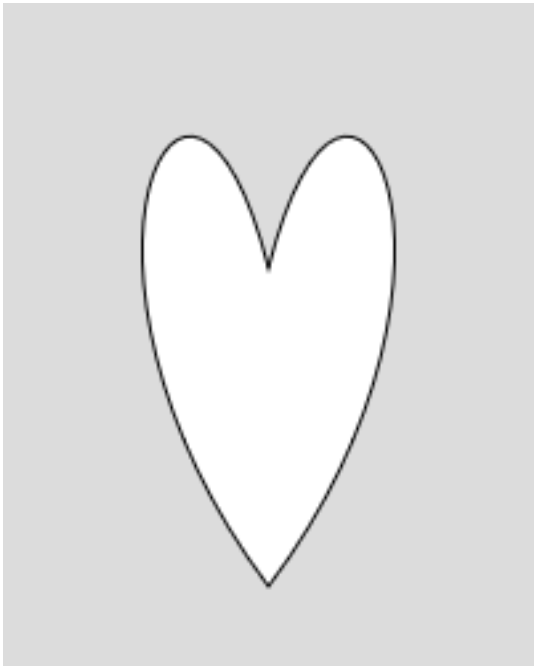
  `bezierPoint( a, b, c, d, t )`

- wyznaczanie stycznej do krzywej

  `bezierTangent( a, b, c, d, t )`

W obu przypadkach `a, b, c, d` to współrzędne x lub y (w zależności, którą współrzędną chcemy wyznaczyć) punktów kontrolnych, a `t` to wartość parametru czyli wartość z przedziału $[0, 1]$.

```
1   function setup()
2   {
3     createCanvas(200, 250);
4   }
5
6   function draw()
7   {
8     background(220);
9
10    beginShape();
11    vertex(100, 220);
12    bezierVertex(10, 100, 70, -20,
          100, 100);
13    bezierVertex(130, -20, 190,
          100, 100, 220);
14    endShape();
15  }
```



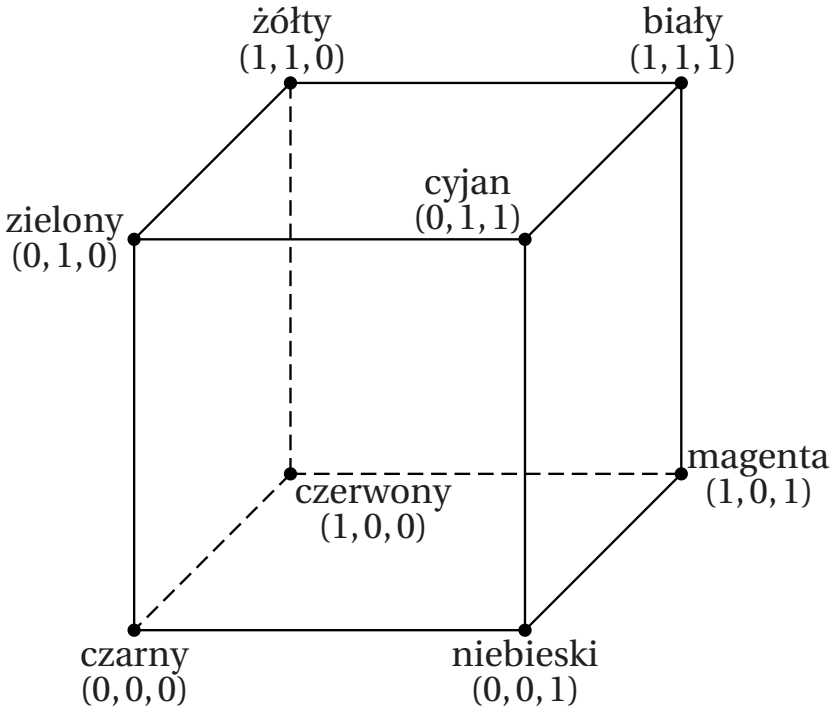
Kolory, wypełnienie i właściwości linii

W p5.js mamy do dyspozycji dwie przestrzenie barw: RGB (domyślna przestrzeń), HSB.

p5.js domyślnie pracuje w reprezentacji barw za pomocą liczb ze zbioru $\{0, 1, \ldots, 255\}$.

W przestrzeni RGB każdy kolor reprezentowany jest przez odcienie trzech barw: czerwonej (Red), zielonej (Green) i niebieskiej (Blue). Te trzy barwy dodawane są do siebie tworząc nowy kolor.

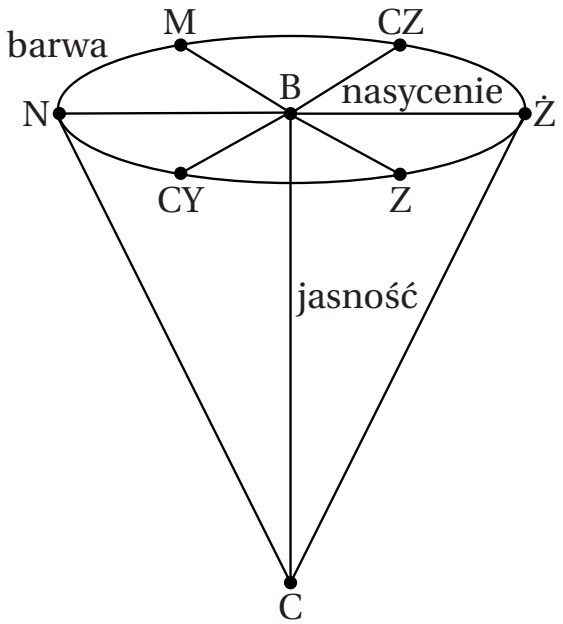
Przestrzeń RGB możemy reprezentować jako sześcian, w którym każdy punkt odpowiada jednemu kolorowi.



Na rysunku podano kolory korzystając z reprezentacji za pomocą liczb z przedziału $[0, 1]$, a nie ze zbioru $\{0, 1, \ldots, 255\}$. Możemy przechodzić pomiędzy reprezentacjami za pomocą mnożenia i dzielenia przez 255.

Przestrzeń HSB powstała opierając się na sposobie postrzegania barw przez człowieka. Opisana jest ona przez główne cechy koloru: barwa (Hue), nasycenie (Saturation) i jasność (Brightness).

Przestrzeń HSB możemy reprezentować jako stożek. C – czarny, B – biały, N – niebieski, M – magenta, CZ – czerwony, Ż – żółty, Z – zielony, CY – cyjan.



Do zmiany przestrzeni barw w p5.js służy funkcja:

```
colorMode( mode )
```

gdzie parametr `mode` przyjmuje jedną z wartości: `RGB`, `HSB`.

Kolor tła:

```
background( gray )
```

```
background( v1, v2, v3 )
```

Pierwsza wersja funkcji korzysta z odcieni szarości, zaś druga w zależności od ustawionej przestrzeni barw ze składowych RGB lub HSB. Funkcja `background` podobnie jak inne funkcje modyfikujące kolor posiada wiele więcej wariantów (patrz dokumentacja).

Do zmiany koloru wypełnienia kształtów możemy użyć funkcji:

```
fill( gray )
```

```
fill( v1, v2, v3 )
```

Po ustawieniu koloru wypełnienia kolor ten będzie obowiązywać aż do następnego wywołania funkcji mającej wpływ na wypełnienie.

Jeśli chcielibyśmy wyłączyć wypełnienie rysowanego kształtu, to możemy skorzystać z funkcji:

```
noFill()
```

Do zmiany koloru brzegu kształtów możemy użyć funkcji:

```
stroke( gray )
```

```
stroke( v1, v2, v3 )
```

Po ustawieniu koloru brzegu kolor ten będzie obowiązywać aż do następnego wywołania funkcji mającej wpływ na brzeg.

Jeśli chcielibyśmy wyłączyć brzeg rysowanego kształtu, to możemy skorzystać z funkcji:

```
noStroke()
```

Do zmiany grubości linii używamy funkcji:

```
strokeWeight( g )
```

gdzie `g` jest to grubość linii w pikselach.

W p5.js linie domyślnie są wygładzane. W celu wyłączenia wygładzania linii używamy funkcji:

```
noSmooth()
```

Transformacje afiniczne w 2D

Postać transformacji afinicznej w 2D jest następująca:

$$F\left(\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}\right) = \begin{pmatrix} f_0 & f_2 & f_4 \\ f_1 & f_3 & f_5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix},$$

gdzie $f_0, f_1, \ldots, f_5 \in \mathbb{R}$, a $x, y \in \mathbb{R}$ to współrzędne przekształcanego punktu.

To jaka transformacja zostanie wykonana zależy od wartości współczynników $f_0, f_1, \ldots, f_5$.

Tego typu reprezentacja jest stosowana ze względu na łatwość składania takich transformacji. Jeśli chcemy złożyć dwie transformacje, to wystarczy wymnożyć ich macierze (w odpowiedniej kolejności).

Zobaczmy jakie transformacje mamy do dyspozycji w p5.js.

Pierwszą transformacją jest translacja:

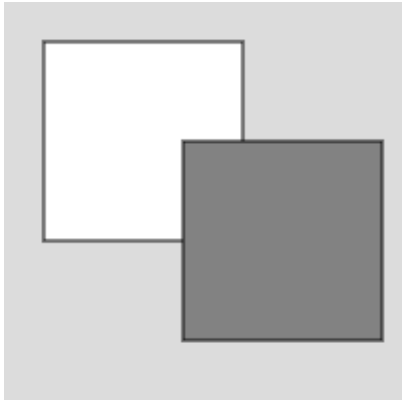
```
translate( tx, ty )
```

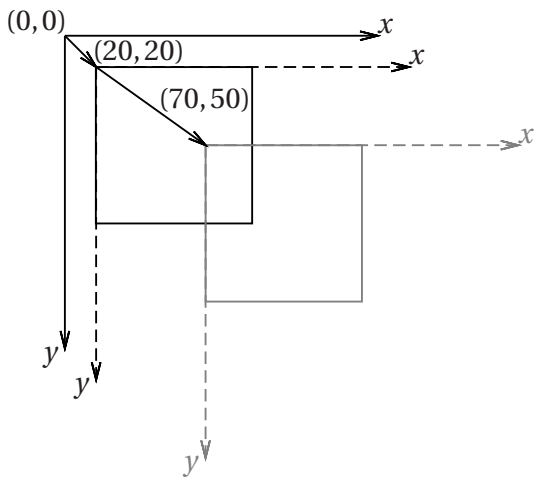
gdzie `tx`, `ty` są to współrzędne wektora translacji. Macierz tej transformacji wygląda następująco:

$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

Przykład:

```
1   function setup()
2   {
3     createCanvas(200, 200);
4   }
5
6   function draw()
7   {
8     background(220);
9
10    fill(255);
11    translate(20, 20);
12    rect(0, 0, 100, 100);
13
14    fill(130);
15    translate(70, 50);
16    rect(0, 0, 100, 100);
17  }
```





Kolejną transformacją jest rotacja wokół początku układu współrzędnych:

```
rotate( angle )
```

gdzie `angle` to kąt obrotu (w radianach). Macierz tej transformacji jest następująca:

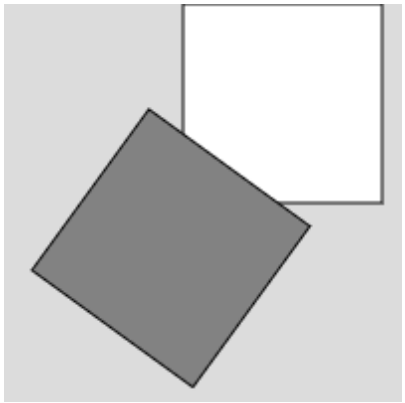
$$\begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

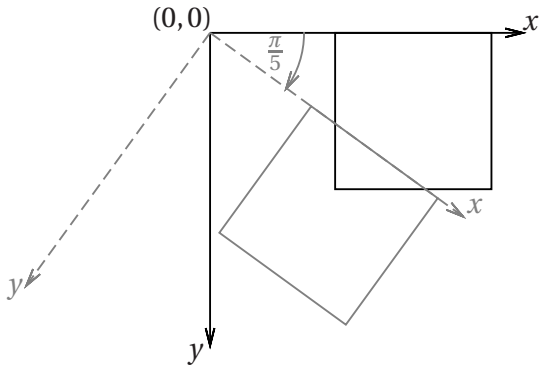
gdzie φ jest to kąt obrotu.

Obrót dokonywany jest zgodnie z ruchem wskazówek zegara.

Przykład:

```
1   function setup()
2   {
3     createCanvas(200, 200);
4   }
5
6   function draw()
7   {
8     background(220);
9
10    fill(255);
11    rect(90, 0, 100, 100);
12
13    fill(130);
14    rotate(PI / 5.0);
15    rect(90, 0, 100, 100);
16  }
```





Do podstawowych transformacji dostępnych w p5.js zaliczamy również skalowanie:

```
scale( s )
```

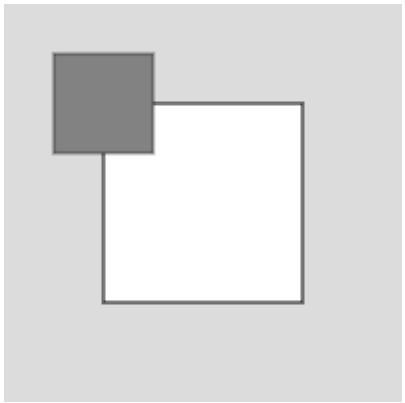
```
scale( sx, sy )
```

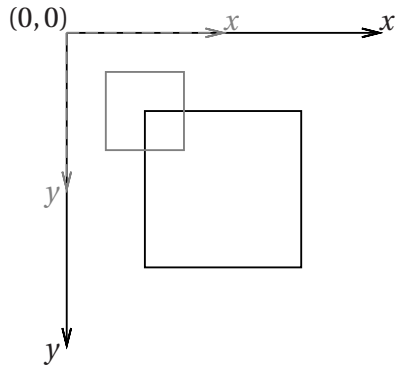
gdzie `s` to współczynnik skali w obu kierunkach, `sx` to współczynnik skali w kierunku X, `sy` to współczynnik skali w kierunki Y. Macierz tej transformacji ma postać:

$$\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Przykład:

```
1   function setup()
2   {
3     createCanvas(200, 200);
4   }
5
6   function draw()
7   {
8     background(220);
9
10    fill(255);
11    rect(50, 50, 100, 100);
12
13    fill(130);
14    scale(0.5);
15    rect(50, 50, 100, 100);
16  }
```





W przykładach widzieliśmy użycie pojedynczej transformacji, a co jeśli chcielibyśmy użyć kilku transformacji po sobie?

Coś takiego jest możliwe ponieważ p5.js przechowuje macierz aktualnej transformacji, która na samym początku jest macierzą jednostkową (transformacja identycznościowa) i w momencie wywołania funkcji transformacji macierz tej transformacji jest domnażana do macierzy aktualnej transformacji. W ten sposób p5.js składa transformacje w jedną.

Ustawienie aktualnej macierzy transformacji na macierz jednostkową:

```
resetMatrix()
```

W p5.js nie jesteśmy skazani tylko na transformacje jakie udostępnia p5.js. Mamy możliwość domnożenia własnej macierzy transformacji. Do tego celu służy funkcja:

```
applyMatrix( f0, f1, f2, f3, f4, f5 )
```

gdzie `f0`, `...`, `f5` są to współczynniki macierzy.

W p5.js mamy dostępny stos macierzy przekształceń oraz stylu rysowania.

Jest to standardowy stos, który przechowuje macierze przekształceń oraz styl jakim rysowany jest model (kolory wypełnienia, linii, tryb rysowania prostokąt itp.). Na stosie tym możemy wykonywać dwie operacje:

- odłożyć aktualną macierz transformacji oraz styl na stos

  `push()`

- zdjąć macierz oraz styl ze szczytu stosu i zastąpić nią aktualną macierz transformacji oraz styl

  `pop()`

```
1   function setup()
2   {
3     createCanvas(200, 200);
4   }
5
6   function draw()
7   {
8     background(220);
9
10    fill(255, 0, 0); // czerwony
11    rect(0, 0, 100, 100);
12
13    push();
14    translate(80, 80);
15    fill(0, 255, 0); // zielony
16    rect(0, 0, 100, 100);
17
18    translate(20, 0);
19    fill(255, 255, 0); // zolty
20    rect(0, 0, 40, 40);
21    pop();
22
23    fill(0, 0, 255); // niebieski
24    rect(0, 0, 40, 40);
25
26    translate(80, 0);
27    fill(255, 0, 255); // rozowy
28    rect(0, 0, 40, 40);
29  }
```