



Technologia i rozwiązania

HTML5 Canvas

Receptury

Ponad 80 receptur prezentujących użycie elementu canvas,
które zrewolucjonizują strony WWW!



Eric Rowell

[PACKT]
PUBLISHING

Tytuł oryginału: HTML5 Canvas Cookbook

Tłumaczenie: Piotr Rajca

ISBN: 978-83-246-5075-0

© Helion 2013.

All rights reserved.

Copyright © Packt Publishing 2011.

First published in the English language under the title 'HTML5 Canvas Cookbook'.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Wydawnictwo HELION dołożyło wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie bierze jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Wydawnictwo HELION nie ponosi również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<ftp://ftp.helion.pl/przyklady/ht5cre.zip>

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/ht5cre>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Zespół oryginalnego wydania	7
O autorze	8
O recenzentach	9
Wstęp	11
Zagadnienia opisywane w tej książce	11
Co jest potrzebne podczas lektury tej książki?	13
Dla kogo jest przeznaczona ta książka?	13
Czym są elementy canvas wprowadzone w HTML5?	13
Stosowane konwencje	14
Pobieranie przykładowych kodów towarzyszących książce	15
Rozdział 1. Wprowadzenie do operacji na ścieżkach i tekstach	17
Wprowadzenie	17
Rysowanie linii	18
Rysowanie łuku	21
Rysowanie krzywej kwadratowej	23
Rysowanie krzywej Béziera	25
Rysowanie zygzaków	26
Rysowanie spirali	28
Praca z tekstem	30
Rysowanie trójwymiarowego tekstu z cieniem	31
Wyzwalanie potęgi fraktali — rysowanie nawiedzonego drzewa	33
Rozdział 2. Rysowanie kształtów i elementów złożonych	37
Wprowadzenie	37
Rysowanie prostokąta	38
Rysowanie okręgu	40
Tworzenie własnych kształtów i stosowanie stylów wypełnienia	42
Zabawa z krzywymi Béziera — rysowanie chmurki	45

Rysowanie przezroczystych kształtów	47
Korzystanie ze stosu stanu kontekstu w celu zapisywania i odtwarzania stylów	48
Stosowanie operacji złożonych	51
Wykorzystanie pętli do tworzenia wzorców — rysowanie koła zębatego	56
Stosowanie wartości losowych we właściwościach kształtów — rysowanie tąki kwiatów	59
Tworzenie funkcji rysujących niestandardowe kształty — talie kart	62
Połączenie wszystkich wiadomości — rysowanie odrzutowca	67
Rozdział 3. Praca z obrazami i klipami wideo	75
Wprowadzenie	75
Wyświetlanie obrazu	76
Przycinanie obrazka	77
Kopiowanie i wklejanie fragmentów obszaru płótna	80
Korzystanie z klipów wideo	82
Pobieranie danych obrazu	84
Wprowadzenie do manipulowania danymi obrazu — odwracanie kolorów	87
Odwracanie kolorów w klipach wideo	89
Konwersja kolorów rysunku na skalę szarości	91
Przekształcanie rysunku na postać danych URL	93
Zapisywanie rysunku w formie obrazu	95
Wczytywanie zawartości rysunku przekazanej w formie danych URL	97
Wyostrażanie obrazka o powiększonych pikselach	99
Rozdział 4. Stosowanie przekształceń	103
Wprowadzenie	103
Przesuwanie kontekstu płótna	104
Obrót kontekstu płótna	105
Skalowanie kontekstu płótna	107
Tworzenie odbicia lustrzanego	109
Tworzenie przekształceń niestandardowych	110
Pochylanie kontekstu płótna	112
Obsługa wielu przekształceń z wykorzystaniem stosu stanu	113
Przekształcanie okręgu na owal	116
Obracanie obrazka	118
Rysowanie prostego logo i losowe określanie jego położenia, obrotu i skali	119
Rozdział 5. Ożywianie płócien poprzez zastosowanie animacji	123
Wprowadzenie	124
Tworzenie klasy Animation	124
Tworzenie ruchu liniowego	128
Tworzenie przyspieszenia	130
Tworzenie oscylacji	133
Oscylujący bąbelek	135
Ruchome wahadło	137
Animowane koła zębate	140
Animowany zegar	145

Symulacja fizyki cząstek	149
Tworzenie mikroskopijnych żyłatek	153
Działanie w warunkach zwiększonego obciążenia i prezentowanie liczby ramek na sekundę	157
Rozdział 6. Interakcja z elementami canvas	
— dołączanie procedur obsługi zdarzeń do kształtów i regionów	163
Wprowadzenie	164
Tworzenie klasy Events	164
Korzystanie ze współrzędnych myszy w obszarze elementu canvas	172
Dołączanie procedur obsługi zdarzeń myszy do regionów	174
Dołączanie procedur obsługi zdarzeń dotyku do regionów na urządzeniach przenośnych	178
Dołączanie procedur obsługi zdarzeń do obrazków	181
Przeciąganie i upuszczanie kształtów	185
Przeciąganie i upuszczanie obrazków	188
Tworzenie powiększania fragmentu obrazka	190
Tworzenie aplikacji graficznej	196
Rozdział 7. Tworzenie grafów i wykresów	203
Wprowadzenie	203
Tworzenie wykresu kołowego	204
Tworzenie wykresu słupkowego	209
Wizualizacja równań	216
Rysowanie danych punktowych przy użyciu wykresu liniowego	221
Rozdział 8. Ratujemy świat, pisząc nową grę	229
Wprowadzenie	229
Tworzenie arkuszy sprite'ów dla bohatera i jego przeciwników	232
Tworzenie obrazów poziomów oraz map obszarów	234
Tworzenie klasy Actor reprezentującej bohatera i jego przeciwników	238
Tworzenie klasy Level	243
Klasa HealthBar	245
Tworzenie klasy Controller	246
Tworzenie klasy Model	251
Tworzenie klasy View	260
Przygotowanie dokumentu HTML i uruchamianie gry	265
Rozdział 9. Wprowadzenie do WebGL	267
Wprowadzenie	267
Tworzenie klasy upraszczającej korzystanie z API WebGL	268
Rysowanie trójkąta	281
Obracanie płaskiego trójkąta w przestrzeni trójwymiarowej	283
Tworzenie obracającego się sześcianu	286
Dodawanie tekstur i oświetlenia	290
Tworzenie trójwymiarowego świata, który można eksplorować	296

Dodatek A. Wykrywanie obsługi elementów canvas	309
Treść zastępcza dla elementów canvas	309
Dodatek B. Bezpieczeństwo korzystania z elementów canvas	313
Dodatek C. Dodatkowe zagadnienia	315
Elementy canvas a efekty przejść i animacje CSS3	315
Wydajność elementów canvas na urządzeniach przenośnych	316
Skorowidz	317

Wprowadzenie do operacji na ścieżkach i tekstach

W tym rozdziale zostaną opisane następujące zagadnienia:

- rysowanie linii,
- rysowanie łuków,
- rysowanie krzywych kwadratowych,
- rysowanie krzywych Béziera,
- rysowanie zygzaków,
- rysowanie spiral,
- praca z tekstem,
- rysowanie trójwymiarowych tekstów z cieniem,
- wyzwolenie potęgi fraktali — rysowanie nawiedzonego drzewa.

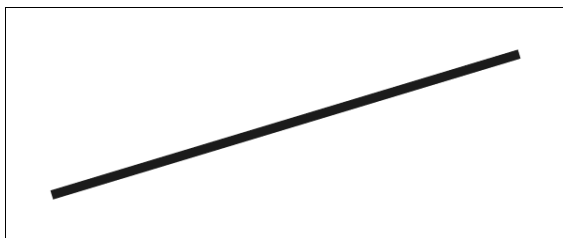
Wprowadzenie

Celem tego rozdziału jest przedstawienie najważniejszych możliwości elementów canvas wprowadzonych w języku HTML5 (nazywanych także „plótnem”). Możliwości te zostaną zademonstrowane w serii coraz bardziej złożonych przykładów. API — interfejs programowania aplikacji — elementów canvas dostarcza podstawowych narzędzi niezbędnych do rysowania i określania

wyglądu (stylu) różnych typów podścieżek, takich jak linie, łuki, krzywe kwadratowe, krzywe Béziera, oraz do łączenia ich w celu tworzenia ścieżek. Interfejs ten zapewnia także pełne wsparcie dla rysowania tekstów, udostępniając przy tym kilka właściwości pozwalających na określanie ich stylu. Zaczynamy!

Rysowanie linii

Podczas rozpoczynania nauki korzystania z płócien HTML5 większość osób jest zainteresowana rysowaniem podstawowych elementów. Ta receptura pokazuje, jak narysować linię.



Jak to zrobić

Aby narysować fragment pochylej linii prostej, należy wykonać następujące czynności:

1. Zdefiniować kontekst 2d (dwuwymiarowy) płótna i określić styl linii:

```
window.onload = function(){  
    // pobranie elementu canvas na podstawie jego identyfikatora  
    var canvas = document.getElementById("myCanvas");  
    // deklaracja kontekstu 2D przy użyciu metody getContext() obiektu canvas  
    var context = canvas.getContext("2d");  
    // ustawienie szerokości linii na 10 pikseli  
    context.lineWidth = 10;  
    // ustawienie koloru linii na niebieski  
    context.strokeStyle = "blue";
```

2. Umieścić kontekst płótna i narysować linię:

```
    // ustawienie położenia kursora  
    context.moveTo(50, canvas.height - 50);  
    // rysowanie linii  
    context.lineTo(canvas.width - 50, 50);  
    // wyświetlenie linii przy wykorzystaniu wybranego wcześniej koloru  
    context.stroke();  
};
```


3. Umieścić element canvas w treści dokumentu HTML:

```
<canvas id="myCanvas" width="600" height="250" style="border:1px
solid black;">
</canvas>
```

Pobieranie kodów przykładów

Zarówno kody przykładów, jak i inne zasoby można pobrać z serwera FTP wydawnictwa Helion — <ftp://ftp.helion.pl/przyklady/ht5cre.zip>.

Jak to działa

Jak widać na powyższym przykładzie, zanim będzie można odwołać się do elementu canvas na podstawie jego identyfikatora, należy poczekać na wczytanie całej strony. Można to zrobić, używając inicjalizatora `window.onload`. Po zakończeniu wczytywania strony można już odwołać się do elementu canvas w drzewie DOM strony, używając w tym celu metody `document.getElementById()`, a następnie zdefiniować dwuwymiarowy (2d) kontekst płótna, przekazując parametr "2d" w wywołaniu metody `getContext()` obiektu canvas. W ostatnich dwóch rozdziałach książki pokazano, że można tworzyć także konteksty trójwymiarowe. W tym przypadku w wywołaniu metody `getContext()` należy przekazać parametr "webgl", "experimental-webgl" bądź jeszcze inny.

Podczas rysowania konkretnego elementu, takiego jak ścieżka, podścieżka lub kształt, konieczne należy pamiętać o tym, że jego styl można zmienić w każdej chwili — zarówno przed rozpoczęciem, jak i po zakończeniu rysowania — aby styl jednak został uwzględniony, trzeba go zastosować bezpośrednio po narysowaniu elementu. Grubość linii można określić przy użyciu właściwości `lineWidth`, a jej kolor przy zastosowaniu właściwości `strokeStyle`. Można uznać, że te czynności przypominają rysowanie na kartce papieru — zanim zaczniemy rysować, wybieramy kredkę (`strokeStyle`) o określonej grubości (`lineWidth`).

Kiedy już weźmiemy do ręki kredkę, możemy umieścić ją w dowolnym miejscu (punkcie rozpoczęcia rysowania). Do tego celu służy metoda `moveTo()`:

```
context.moveTo(x,y);
```

Kontekst płótna można sobie wyobrazić jako kursor służący do rysowania. Metoda `moveTo()` tworzy nową podścieżkę w określonym punkcie. Lewy górny wierzchołek płótna ma współrzędne (0,0), natomiast współrzędne prawego dolnego wierzchołka to szerokość i wysokość elementu canvas.

Po umieszczeniu kursora rysującego w określonym punkcie możemy narysować linię, używając metody `lineTo()`, w której wywołaniu przekazywane są współrzędne punktu końcowego:

```
context.lineTo(x,y);
```

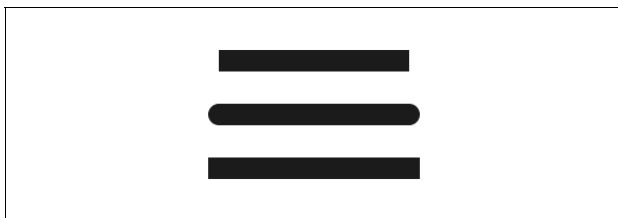
I w końcu, aby linia stała się widoczna, trzeba wywołać metodę `stroke()`. Jeśli przed narysowaniem linii nie wybierzemy innego koloru, to domyślnie będzie ona miała kolor czarny.

Poniżej przedstawiono podstawową procedurę, jaką należy wykonać w celu narysowania linii przy użyciu API elementów canvas:

1. Określenie stylu linii (odpowiadające wybraniu kredki o danej grubości).
2. Umieszczenie kontekstu płótna w wybranym miejscu przy użyciu metody `moveTo()` (odpowiadające umieszczeniu kredki w danym miejscu kartki papieru).
3. Narysowanie linii poprzez wywołanie metody `lineTo()`.
4. Wyświetlenie linii za pośrednictwem wywołania metody `stroke()`.

Dodatkowe informacje

Rysowane linie mogą mieć trzy różne rodzaje zakończeń: `butt` (prostokątne), `round` (okrągłe) oraz `square` (kwadratowe). Styl zakończeń można określać przy użyciu właściwości `lineCap` kontekstu płótna. Domyślnie stosowane jest zakończenie `butt`. Wszystkie trzy style zakończeń zostały przedstawione na poniższym rysunku. Na samej górze widoczny jest domyślny styl zakończeń — `butt`; środkową linię narysowano przy wykorzystaniu stylu zakończeń `round`, a najniższą — przy zastosowaniu stylu `square`.



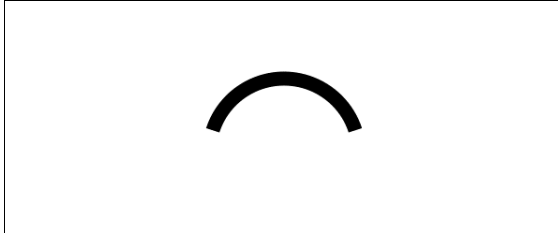
Należy zwrócić uwagę, że środkowa i dolna linia są nieco dłuższe od górnej, choć ich długości są takie same. Dzieje się tak dlatego, że w przypadku zastosowania stylu `round` oraz `square` zakończenia linii powiększają jej długość o wartość równą jej szerokości. Jeśli rysujemy na przykład linię o długości 200 pikseli i szerokości 10 pikseli i zastosujemy przy tym styl zakończeń `round`, to ostateczna długość linii wyniesie 210 pikseli, gdyż każde z jej zakończeń powiększy jej długość o 5 pikseli.

Patrz także

- „Rysowanie zygzaków”.
- „Połączenie wszystkich wiadomości — rysowanie odrzutowca” w rozdziale 2.

Rysowanie łuku

Czasami może się pojawić konieczność narysowania idealnego łuku. Ta receptura może się przydać, jeśli chcemy narysować radosną tęczę, uśmiechniętą buźkę lub jakieś diagramy.



Jak to zrobić

Aby narysować łuk, należy wykonać następujące czynności:

1. Zdefiniować kontekst 2d płótna i określić styl łuku:

```
window.onload = function(){  
    var canvas = document.getElementById("myCanvas");  
    var context = canvas.getContext("2d");  
    context.lineWidth = 15;  
    context.strokeStyle = "black"; // kolor linii
```

2. Narysować łuk:

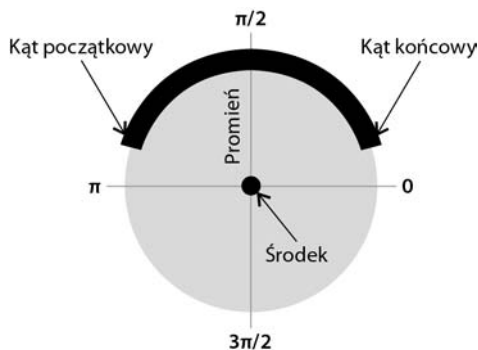
```
    context.arc(canvas.width / 2, canvas.height / 2 + 40, 80, 1.1 * Math.PI,  
    ↪ 1.9 * Math.PI, false);  
    context.stroke();  
};
```

3. Umieścić element canvas w treści dokumentu HTML:

```
<canvas id="myCanvas" width="600" height="250" style="border:1px solid black;">  
</canvas>
```

Jak to działa

Do narysowania łuku, zdefiniowanego jako fragment umownego okręgu, służy metoda `arc()`. Przyjrzyjmy się następującemu diagramowi:



Umowny okrąg jest definiowany przy użyciu punktu stanowiącego jego środek i promienia. Z kolei rysowany fragment okręgu definiujemy, podając parę kątów — początkowy i końcowy — oraz informację, czy łuk ma być rysowany zgodnie z kierunkiem ruchu wskazówek zegara, czy w kierunku przeciwnym.

```
context.arc( srodekX, srodekY, promien, katPoczątkowy,
             katKońcowy, przeciwnieDoRuchuWskazowekZegara );
```

Trzeba pamiętać, że kąty zaczynają się od wartości 0π z prawej strony okręgu i zmieniają się w kierunku zgodnym z kierunkiem ruchu wskazówek zegara, poprzez wartości $3\pi/2$, π , $\pi/2$, z powrotem do 0. W tym przykładzie zastosowaliśmy kąt początkowy o wartości $1,1\pi$ oraz kąt końcowy o wartości $1,9\pi$. Oznacza to, że kąt początkowy znajduje się nieco powyżej środka, z lewej strony umownego okręgu, a kąt końcowy — nieco powyżej środka, z prawej strony okręgu.

Dodatkowe informacje

Wartości kąta początkowego i końcowego nie muszą wcale zawierać się w zakresie od 0π do 2π — mogą to być dowolne liczby rzeczywiste, gdyż nic nie stoi na przeszkodzie, by kąty zachodziły na siebie.

Załóżmy, że kąt początkowy wynosi 3π . Odpowiada to jednemu pełnemu obrotowi wokół okręgu (2π) oraz połowie kolejnego obrotu (1π). Innymi słowy, wartość 3π jest odpowiednikiem wartości 1π . I jeszcze jeden przykład — wartość -3π także jest odpowiednikiem wartości 1π , gdyż w tym przypadku kąt zatoczy jeden pełny obrót i dodatkowe pół obrotu wokół okręgu, w kierunku przeciwnym do kierunku ruchu wskazówek zegara, i ostatecznie znajdzie się w położeniu odpowiadającym wartości kątowni 1π .

Innym sposobem rysowania łuków w elemencie canvas jest użycie metody `arcTo()`. W tym przypadku łuk definiowany jest na podstawie punktu kontekstu, punktu kontrolnego oraz promienia.

```
context.arcTo(punktKontrolnyX1, punktKontrolnyY1, punktKoncowyX, punktKoncowyY,
              ↪promien);
```

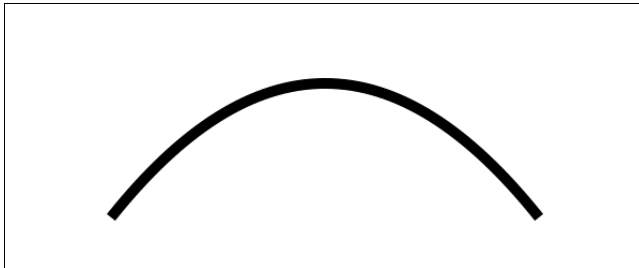
W odróżnieniu od metody `arc()`, która rysuje okrąg wokół określonego środka, metoda `arcTo()` zależy od punktu kontekstu, przez co jest nieco podobna do metody `lineTo()`. Metoda `arcTo()` jest najczęściej używana do tworzenia zaokrąglonych wierzchołków podczas rysowania ścieżek i kształtów.

Patrz także

- „Rysowanie okręgu” w rozdziale 2.
- „Animowane koła zębate” w rozdziale 5.
- „Animowany zegar” w rozdziale 5.

Rysowanie krzywej kwadratowej

W tej recepturze wyjaśniono, jak rysować krzywe kwadratowe. Krzywe tego typu zapewniają znacznie większą elastyczność i pozwalają na rysowanie krzywizn bardziej naturalnych od swych kuzynów łuków i doskonale nadają się do tworzenia dowolnych kształtów.



Jak to zrobić

Aby narysować krzywą kwadratową, należy wykonać następujące czynności:

1. Zdefiniować kontekst 2d płótna:

```
window.onload = function(){  
    var canvas = document.getElementById("myCanvas");  
    var context = canvas.getContext("2d");  
  
    context.lineWidth = 10;  
    context.strokeStyle = "black"; // kolor linii
```

2. Odpowiednio umieścić punkt kontekstu i narysować krzywą kwadratową:

```
context.moveTo(100, canvas.height - 50);  
context.quadraticCurveTo(canvas.width / 2, -50, canvas.width
```

```

        - 100, canvas.height - 50);
    context.stroke();
};

```

3. Umieścić element canvas w treści dokumentu HTML:

```

<canvas id="myCanvas" width="600" height="250" style="border:1px solid black;">
</canvas>

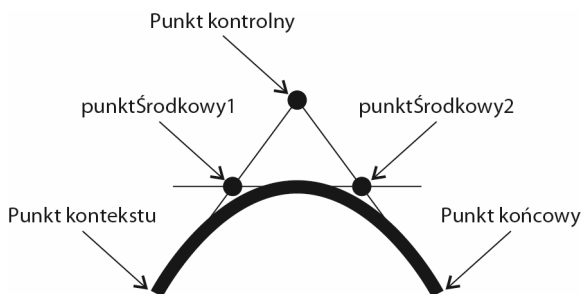
```

Jak to działa

Krzywe kwadratowe rysowane w elemencie canvas są definiowane przez punkt kontekstu, punkt kontrolny oraz punkt końcowy:

```
context.quadraticCurveTo(kontrolnyX,kontrolnyY,punktKocowyX,punktKoncowyY);
```

Przeanalizujmy następujący diagram:



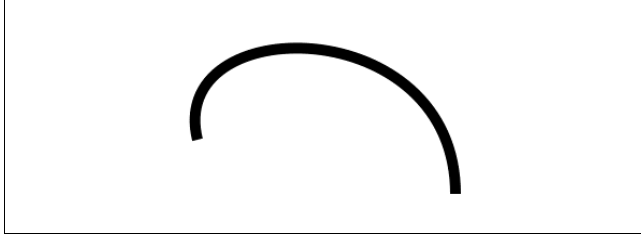
Krzywizna krzywej tego typu jest styczna do trzech stycznych charakterystycznych. W swojej pierwszej części krzywa kwadratowa jest styczna do umownej prostej przechodzącej przez punkt kontekstu i punkt kontrolny. W szczytowym miejscu wygięcia krzywa jest styczna do umownej prostej przechodzącej przez dwa punkty środkowe — punktŚrodkowy1 oraz punktŚrodkowy2. W swojej końcowej części krzywa jest styczna do umownej prostej przechodzącej przez punkty kontrolny i końcowy.

Patrz także

- „Połączenie wszystkich wiadomości — rysowanie odrzutowca” w rozdziale 2.
- „Wyzwolenie potęgi fraktali — rysowanie nawiedzzonego drzewa”.

Rysowanie krzywej Béziera

Jeśli krzywa kwadratowa nie zaspokaja naszych potrzeb, to być może zrobi to krzywa Béziera. Krzywe Béziera, nazywane także krzywymi sześciennymi, są najbardziej zaawansowanym rodzajem krzywych dostępnych w API elementach canvas.



Jak to zrobić

Aby narysować krzywą Béziera, należy wykonać następujące czynności:

1. Zdefiniować kontekst 2d płótna:

```
window.onload = function(){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");

    context.lineWidth = 10;
    context.strokeStyle = "black"; // kolor linii
    context.moveTo(180, 130);
```

2. Odpowiednio umieścić punkt kontekstu i narysować krzywą kwadratową:

```
    context.bezierCurveTo(150, 10, 420, 10, 420, 180);
    context.stroke();
};
```

3. Umieścić element canvas w treści dokumentu HTML:

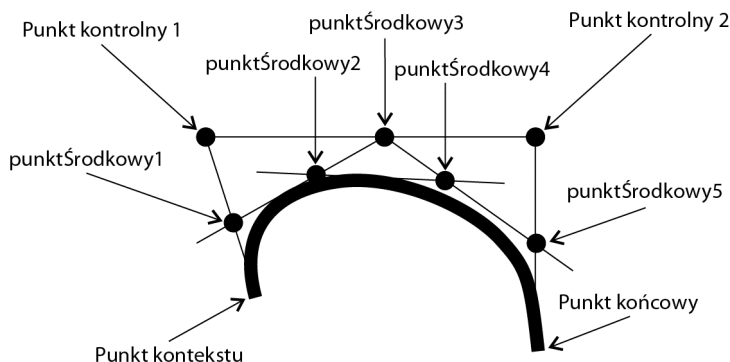
```
<canvas id="myCanvas" width="600" height="250" style="border:1px solid black;">
</canvas>
```

Jak to działa

Krzywe Béziera rysowane w elementach canvas są definiowane przez punkt kontekstu, dwa punkty kontrolne oraz punkt końcowy. Dodatkowy punkt kontrolny zapewnia znacznie większą kontrolę nad postacią krzywizny, niż było to możliwe w przypadku krzywych kwadratowych:

```
context.bezierCurveTo(punktKontrolnyX1, punktKontrolnyY1, punktKontrolnyX2,
    ↪ punktKontrolnyY2, punktKoncowyX, punktKoncowyY);
```

Przeanalizujmy następujący diagram:



W odróżnieniu od krzywych kwadratowych, definiowanych przez trzy styczne charakterystyczne, krzywe Béziera są definiowane przez pięć stycznych. Pierwsza, początkowa część krzywej jest styczna do umownej prostej przechodzącej przez punkt kontekstu i pierwszy punkt kontrolny. Kolejna część krzywej jest styczna do umownej prostej przechodzącej przez punktŚrodkowy1 oraz punktŚrodkowy3. Wierzchołek krzywizny jest styczny do umownej prostej przechodzącej przez punktŚrodkowy2 i punktŚrodkowy4. Czwartą część krzywej jest styczna do umownej prostej przechodzącej przez punktŚrodkowy3 oraz punktŚrodkowy5. Ostatnia część krzywej jest styczna do umownej prostej przechodzącej przez drugi punkt kontrolny i punkt końcowy.

Patrz także

- „Stosowanie wartości losowych we właściwościach kształtów — rysowanie pola kwiatów” w rozdziale 2.
- „Połączenie wszystkich wiadomości — rysowanie odrzutowca” w rozdziale 2.

Rysowanie zygzaków

Ta receptura przedstawia sposób rysowania ścieżki, która powstanie z połączenia podścieżek i utworzy łamaną — zygzak.



Jak to zrobić

Aby narysować łamaną, należy wykonać następujące operacje:

1. Zdefiniować kontekst 2d płótna:

```
window.onload = function(){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");

    var startX = 85;
    var startY = 70;
    var zigzagSpacing = 60;
```

2. Określić styl łamanej i rozpocząć rysowanie ścieżki:

```
context.lineWidth = 10;
context.strokeStyle = "#0096FF"; // kolor niebieskawy
context.beginPath();
context.moveTo(startX, startY);
```

3. Narysować siedem łączących się odcinków i wyświetlić ścieżkę, wywołując metodę `stroke()`:

```
// rysowanie siedmiu linii prostych
for (var n = 0; n < 7; n++) {
    var x = startX + ((n + 1) * zigzagSpacing);
    var y;
    if (n % 2 == 0) { // jeśli n jest parzyste
        y = startY + 100;
    }
    else { // jeśli n jest nieparzyste
        y = startY;
    }
    context.lineTo(x, y);
}

context.stroke();
```

4. Umieścić element canvas w treści dokumentu HTML:

```
<canvas id="myCanvas" width="600" height="250" style="border:1px solid black;">
</canvas>
```

Jak to działa

Aby narysować zygzak, należy połączyć ze sobą kilka ukośnych linii, tworząc w ten sposób ścieżkę. Można to zrobić przy użyciu pętli, w której w nieparzystych iteracjach będzie rysowana linia ukośna skierowana w dół i w prawo, a w iteracjach parzystych — linia ukośna skierowana w górę i w prawo.

Najważniejszym aspektem jest tu metoda `beginPath()`. Jej wywołanie deklaruje początek rysowania ścieżki. Dzięki temu koniec każdej linii — podścieżki — będzie definiował początek kolejnej podścieżki. W razie pominięcia wywołania tej metody trzeba by przy użyciu metody `moveTo()` mozolnie umieszczać punkt kontekstu płótna przed rozpoczęciem rysowania każdej linii, by zapewnić, że początek każdej kolejnej rysowanej linii będzie się pokrywał z końcem poprzedniej. Wywołanie metody `beginPath()` jest także konieczne w przypadku tworzenia kształtów, o czym mowa w następnym rozdziale.

Style połączeń linii

Warto zwrócić uwagę na to, że połączenia pomiędzy kolejnymi segmentami rysowanego zygzaka są spiczaste. Wynika to z tego, że domyślnym stylem połączeń linii w płótnach HTML5 jest `miter`. Korzystając z właściwości `lineJoin` kontekstu płótna, można także zmienić styl połączeń na `round` (połączenia zaokrąglone) lub `bevel` (połączenia ukośne).

Jeśli łączone segmenty linii są stosunkowo cienkie i nie łączą się pod ostrymi kątami, to zauważenie jakichkolwiek różnic pomiędzy poszczególnymi stylami połączeń może być trudne. Zazwyczaj różnice pomiędzy nimi stają się wyraźne przy liniach, których grubość przekracza 5 pikseli, a kąty pomiędzy nimi są stosunkowo niewielkie.

Rysowanie spirali

Uwaga — ta receptura może działać hipnotycznie. W tym przykładzie narysujemy spiralę — ścieżkę składającą się z sekwencji krótkich linii.



Jak to zrobić

Aby narysować spiralę o określonym punkcie centralnym, należy wykonać następujące czynności:

1. Zdefiniować kontekst 2d płótna i zainicjować parametry spirali:

```
window.onload = function(){  
    var canvas = document.getElementById("myCanvas");
```

```
var context = canvas.getContext("2d");
```

```
var radius = 0;
```

```
var angle = 0;
```

2. Określić styl rysowanej spirali:

```
context.lineWidth = 10;
context.strokeStyle = "#0096FF"; // kolor niebieskawy
context.beginPath();
context.moveTo(canvas.width / 2, canvas.height / 2);
```

3. Zatoczyć trzy pełne obroty wokół punktu centralnego (przy czym na każdy obrót będzie przypadać po 50 segmentów linii). Każdy segment będzie rysowany przy użyciu metody `lineTo()`, od końca poprzedniego segmentu do aktualnie wyznaczonego punktu, przy czym za każdym razem promień będzie zwiększany o 0.75. Po zakończeniu trzech pełnych obrotów spirala zostanie wyświetlona poprzez wywołanie metody `stroke()`:

```
for (var n = 0; n < 150; n++) {
    radius += 0.75;
    // pełny obrót będzie się składał z 50 iteracji
    angle += (Math.PI * 2) / 50;
    var x = canvas.width / 2 + radius * Math.cos(angle);
    var y = canvas.height / 2 + radius * Math.sin(angle);
    context.lineTo(x, y);
}
context.stroke();
};
```

4. Umieścić element canvas w treści dokumentu HTML:

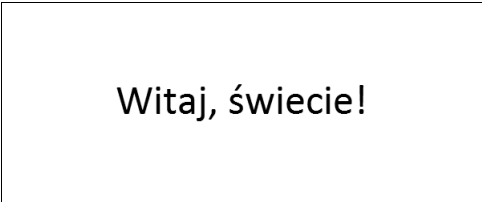
```
<canvas id="myCanvas" width="600" height="250" style="border:1px solid black;">
</canvas>
```

Jak to działa

Aby narysować w elemencie canvas spiralę, możemy umieścić kursor w jego środku, a następnie narysować sekwencję bardzo krótkich linii, zwiększając przy tym kąt i odległość od punktu centralnego, przy czym każda kolejna linia będzie się zaczynać w punkcie zakończenia poprzedniej. Analizując ten algorytm rysowania, można wyobrazić sobie, że jesteśmy dzieckiem, które stoi na chodniku z kawałkiem kredy w ręce. Pochylamy się, przykładamy kredę do chodnika i zaczynamy kręcić się w koło (nie za szybko, chyba że chcemy, by się nam zakręciło w głowie). Podczas kręcenia się odsuwamy kredę coraz dalej od siebie. Po kilku obrotach okaże się, że narysowaliśmy śliczną, małą spiralę.

Praca z tekstem

Niemal wszystkie aplikacje korzystają z wyświetlania tekstów, by efektywnie przekazywać informacje użytkownikom. Ta receptura pokazuje, jak wyświetlić w elemencie canvas optymistyczne powitanie.



Witaj, świecie!

Jak to zrobić

Aby wyświetlić na płótnie tekst, należy wykonać następujące czynności:

1. Zdefiniować kontekst 2d płótna i określić styl prezentowanego tekstu:

```
window.onload = function(){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");

    context.font = "40pt Calibri";
    context.fillStyle = "black";
```

2. Wyrównać tekst w poziomie i w pionie, a następnie go wyświetlić:

```
// wyrównanie tekstu w poziomie
context.textAlign = "center";
// wyrównanie tekstu w pionie
context.textBaseline = "middle";
context.fillText("Witaj, świecie!", canvas.width / 2, 120);
};
```

3. Umieścić element canvas w treści dokumentu HTML:

```
<canvas id="myCanvas" width="600" height="250" style="border:1px solid black;">
</canvas>
```

Jak to działa

Gdy wyświetlamy tekst w elemencie canvas, możemy zdefiniować styl oraz wielkość czcionki (właściwość `font`), kolor czcionki (właściwość `fillStyle`), wyrównanie tekstu w poziomie (właściwość `textAlign`) i w pionie (właściwość `textBaseline`). Właściwości `textAlign` można przypisać wartości `left`, `center` bądź `right`, a właściwości `textBaseline` wartości `top`, `hanging`, `middle`, `alphabetic`, `ideographic` lub `bottom`. Domyślną wartością właściwości `textAlign` jest `left`, natomiast właściwości `textBaseline` — `alphabetic`.

Dodatkowe informacje

Oprócz metody `fillText()` API elementów canvas udostępnia metodę `strokeText()`:

```
context.strokeText("Witaj, świecie!", x, y);
```

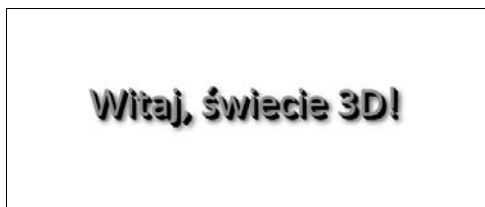
Metoda ta nie wyświetla samego tekstu, lecz wypełnia zajmowany przez niego obszar określonym kolorem. Aby wyświetlić tekst i jednocześnie wypełnić kolorem zajmowany przez niego obszar, należy wywołać zarówno metodę `fillText()`, jak i `strokeText()`. Aby tekst miał odpowiednią wielkość, w pierwszej kolejności należy wywołać metodę `fillText()`, a dopiero po niej metodę `strokeText()`.

Patrz także

- „Rysowanie trójwymiarowego tekstu z cieniem”.
- „Tworzenie odbicia lustrzanego” w rozdziale 4.
- „Rysowanie prostego logo i losowe określanie jego położenia, obrotu oraz skali” w rozdziale 4.

Rysowanie trójwymiarowego tekstu z cieniem

Osoby, które uważają, że zwyczajne, dwuwymiarowe teksty nie są szczególnie atrakcyjne, zainteresują się zapewne rysowaniem tekstów trójwymiarowych. Choć API elementów canvas nie zapewnia bezpośrednio możliwości rysowania takich tekstów, to jednak korzystając z tego API, stosunkowo łatwo można samodzielnie utworzyć funkcję `draw3dText()`.



Jak to zrobić

Aby narysować trójwymiarowy tekst, należy wykonać następujące czynności:

1. Zdefiniować kontekst 2d płótna i określić styl prezentowanego tekstu:

```
window.onload = function(){
    var canvas = document.getElementById("myCanvas");
```

```
var context = canvas.getContext("2d");
```

```
context.font = "40pt Calibri";
context.fillStyle = "black";
```

2. Określić wyrównanie i wyświetlić trójwymiarowy tekst:

```
// wyrównanie tekstu w poziomie
context.textAlign = "center";
// wyrównanie tekstu w pionie
context.textBaseline = "middle";
draw3dText(context, "Witaj, świecie 3D!", canvas.width / 2, 120, 5);
};
```

3. Zdefiniować funkcję draw3dText(), która będzie tworzyć kilka warstw tekstu i dodawać do niego cień:

```
function draw3dText(context, text, x, y, textDepth){
    var n;

    // rysowanie dolnych warstw
    for (n = 0; n < textDepth; n++) {
        context.fillText(text, x - n, y - n);
    }

    // rysowanie górnej warstwy z cieniem rzucanym na
    // warstwy niższe
    context.fillStyle = "#5E97FF";
    context.shadowColor = "black";
    context.shadowBlur = 10;
    context.shadowOffsetX = textDepth + 2;
    context.shadowOffsetY = textDepth + 2;
    context.fillText(text, x - n, y - n);
}
```

4. Umieścić element canvas w treści dokumentu HTML:

```
<canvas id="myCanvas" width="600" height="250" style="border:1px solid black;">
</canvas>
```

Jak to działa

Aby narysować trójwymiarowy tekst w elemencie canvas, można wyświetlić kilka nieznacznie przesuniętych względem siebie warstw zawierających ten sam tekst, tworząc w ten sposób imitację głębi. W tej recepturze głębokość tekstu wynosi 5, co oznacza, że nasza funkcja draw3dText() pięć razy wyświetli tekst Witaj, świecie 3D!, za każdym razem nieznacznie go przesuwając. Te dodatkowe warstwy tekstu zostaną wyświetlone na czarno, by zapewnić imitację głębi.

Następnie dodamy kolorową, wierzchnią warstwę. Ostatnią operacją jest dodanie rozmytego cienia poniżej tekstu, co można zrobić poprzez przypisanie odpowiednich wartości właściwościom `shadowColor`, `shadowBlur`, `shadowOffsetX` i `shadowOffsetY` kontekstu płótna. Właściwości te można stosować nie tylko przy wyświetlaniu tekstów, lecz także podścieżek, ścieżek i kształtów, o czym będzie jeszcze mowa w dalszej części książki.

Wyzwalanie potęgi fraktali — rysowanie nawiedzonego drzewa

Czym są fraktale? Można powiedzieć, że są one połączeniem matematyki ze sztuką. Znajdziemy je we wszelkiego rodzaju wzorcach występujących w naturze. Pod względem algorytmicznym fraktale są równaniami wykorzystującymi rekurencję. W tej recepturze narysujemy naturalnie wyglądające drzewo, zaczynające się od pnia rozdzielającego się na dwa konary, z których każdy następnie rozdziela się na dwie gałęzie. Po dwunastu takich powtórzeniach uzyskamy rozłożyste, pozornie chaotyczne drzewo z masą konarów i gałązek.



Jak to zrobić

Oto czynności, jakie należy wykonać, aby narysować drzewo, korzystając z fraktali:

1. Utworzyć rekurencyjną funkcję, która będzie rysować jedną gałąź, rozdzielającą się na dwie mniejsze gałęzie; następnie funkcja będzie wywoływać rekurencyjnie samą siebie, by narysować te dwie gałęzie, zaczynając od ich punktów końcowych:

```

function drawBranches(context, startX, startY, trunkWidth, level){
  if (level < 12) {
    var changeX = 100 / (level + 1);
    var changeY = 200 / (level + 1);

    var topRightX = startX + Math.random() * changeX;
    var topRightY = startY - Math.random() * changeY;

    var topLeftX = startX - Math.random() * changeX;
    var topLeftY = startY - Math.random() * changeY;

    // rysowanie prawej gałęzi
    context.beginPath();
    context.moveTo(startX + trunkWidth / 4, startY);
    context.quadraticCurveTo(startX + trunkWidth /
    ↪4, startY - trunkWidth, topRightX, topRightY);
    context.lineWidth = trunkWidth;
    context.lineCap = "round";
    context.stroke();

    // rysowanie lewej gałęzi
    context.beginPath();
    context.moveTo(startX - trunkWidth / 4, startY);
    context.quadraticCurveTo(startX - trunkWidth / 4, startY -
    trunkWidth, topLeftX, topLeftY);
    context.lineWidth = trunkWidth;
    context.lineCap = "round";
    context.stroke();

    drawBranches(context, topRightX, topRightY, trunkWidth * 0.7, level + 1);
    drawBranches(context, topLeftX, topLeftY, trunkWidth * 0.7, level + 1);
  }
}

```

2. Zainicjować kontekst płótna i rozpocząć rysowanie fraktala drzewa poprzez wywołanie funkcji `drawBranches()`:

```

window.onload = function(){
  canvas = document.getElementById("myCanvas");
  context = canvas.getContext("2d");

  drawBranches(context, canvas.width / 2, canvas.height, 50, 0);
};

```

3. Umieścić element `canvas` w treści dokumentu HTML:

```

<canvas id="myCanvas" width="600" height="250" style="border:1px solid black;">
</canvas>

```


Jak to działa

Aby narysować drzewo przy użyciu fraktala, należy przygotować rekurencyjną funkcję, która zdefiniuje matematyczną naturę drzewa. Jeśli poświęcimy chwilę na przestudiowanie wyglądu drzew (gdy to zrobimy, okaże się, że są naprawdę piękne), zauważymy, że wszystkie ich gałęzie rozdzielają się na mniejsze gałązki. Oznacza to, że nasza rekurencyjna funkcja powinna rysować jedną gałąź, która rozdziela się, dając początek dwóm kolejnym gałęziom, a następnie rekurencyjnie wywołać samą siebie, by narysować te dwie gałęzie wraz z kolejnymi, jeszcze mniejszymi gałęziami.

Teraz, kiedy już wiemy, jak ma działać nasz fraktal, możemy go zaimplementować, korzystając z API elementów canvas. Najprostszym sposobem utworzenia gałęzi rozdzielającej się na dwie kolejne jest narysowanie dwóch krzywych kwadratowych, wygiętych w przeciwnych kierunkach.

Gdybyśmy zastosowali dla każdej iteracji dokładnie tę samą procedurę, to narysowane drzewo byłoby idealnie symetryczne i mało interesujące. Aby wyglądało ono bardziej naturalnie, położenie punktów końcowych poszczególnych gałęzi będzie modyfikowane o wartości losowe.

Dodatkowe informacje

Najciekawszym aspektem tej receptury jest to, że każde narysowane drzewo będzie inne. Jeśli Czytelnik zaimplementuje powyższy przykład i będzie go wielokrotnie wyświetlał w przeglądarce, to każde z wygenerowanych drzew będzie unikalne. Można także spróbować zmodyfikować algorytm rysujący gałęzie, by tworzyć różne rodzaje drzew albo nawet dorysowywać liście na końcach najmniejszych gałązek.

Do doskonałymi przykładami fraktali w naturze są muszle, płatki śniegu, pióra, rośliny, kryształy, góry, rzeki i błyskawice.

Skorowidz

A

Actor, 238
 attack(), 239
 damage(), 242
 draw(), 240, 242
 fade(), 240
 getCenter(), 242
 isFacingRight(), 239
 jump(), 240
 moveLeft(), 240
 moveRight(), 239
 sterowanie postaciami, 242
 stop(), 239
 tworzenie, 238
 updateSpriteMotion(), 240, 242
 updateSpriteSeqNum(), 241, 242
 zarządzanie sprite'ami, 242
animacja, 124
 Animation, 124, 127
 animationLoop(), 127
 clear(), 126, 128
 getCanvas(), 125
 getContext(), 125
 getFps(), 127
 getTime(), 128
 getTimeInterval(), 126, 128
 requestAnimationFrame(), 127
 requestAnimFrame(), 127
 setDrawStage(), 126
 start(), 126, 128, 130
 stop(), 126, 128, 130
 tworzenie, 125

FPS, 124, 161
 wyświetlanie, 157
Gear, 141
 draw(), 141, 145
 właściwości, 144
koła zębate, 140
 draw(), 141, 145
 start(), 145
oscylacje, 133
 arc(), 137
 rect(), 134
 ruchome wahadło, 137
 ruchomy bąbelek, 135
 stage(), 134
 start(), 134, 137, 140
przyspieszenie, 130, 132
 stop(), 132
requestAnimationFrame(), 124
ruch cząsteczki, 149
 start(), 152
ruch liniowy, 128
ruchome mikroby, 153
 zwiększanie obciążenia, 162
zegar, 145
 arc(), 148
 fillText(), 148
 shadowOffsetX, 148
 shadowOffsetY, 148
 start(), 148
 stroke(), 148
 translate(), 148
Animation, 124, 127
animationLoop(), 127, 271
API, 11, 17
 copy, 52
 destination-atop, 52

 destination-in, 52
 destination-out, 52
 destination-over, 52
 lighter, 52
 source-atop, 52
 source-in, 52
 source-out, 52
 source-over, 52
 xor, 52
arkusze sprite'ów, 232, 233

B

BarChart, 210
 drawBars(), 212, 215
 drawGridlines(), 213
 drawXAxis(), 214
 drawXLabels(), 212
 drawYAxis(), 214
 drawYValues(), 212
 getLabelAreaHeight(), 211
 getLongestValueWidth(), 211
 tworzenie, 210

C

canvas, 11, 13
 API, 11, 17
 addColorStop(), 44
 arc(), 21, 41
 arcTo(), 22
 beginPath(), 28, 44
 bezierCurveTo(), 25, 122
 closePath(), 44
 createLinearGradient(), 44, 59

canvas

API

createPattern(), 45
 createRadialGradient(), 44
 document.getElementById
 ↳ ById(), 19
 draw(), 62
 draw3dText(), 31
 drawBranches(), 34
 drawImage(), 77, 79, 82, 119
 drawTriangle(), 44
 Events, 165
 fill(), 39, 41
 fillRect(), 39
 fillText(), 31
 getContext(), 19
 getImageData(), 85, 87
 globalAlpha, 50
 globalCompositeOperation, 56
 isPointInPath(), 171
 krzywe Béziera, 25
 lineTo(), 19
 moveTo(), 19, 44
 obraz, 75
 operacje złożone, 52
 procedury obsługi
 zdarzeń, 165
 putImageData(), 89
 quadraticCurveTo(), 24
 rect(), 39
 request.responseText, 98
 requestAnimationFrame(), 84, 91
 restore(), 50, 115
 rotate(), 105, 119
 save(), 50, 115
 scale(), 107, 110, 117
 setInterval(), 102
 setTransform(), 111
 stos stanu kontekstu, 48
 stroke(), 20
 strokeRect(), 39
 strokeText(), 31
 style wypełnień, 42
 toDataURL(), 94
 transform(), 113
 translate(), 119

WebGL, 268
 atrybuty, 14
 height, 14
 id, 14
 width, 14
 bezpieczeństwo danych, 313
 drawImage(), 313
 fillStyle, 314
 fillText(), 314
 flaga prawidłowego źródła, 313
 getImageData(), 314
 measureText(), 314
 SECURITY_ERR, 314
 strokeStyle, 314
 strokeText(), 314
 toDataURL(), 314
 kontekst 2d, 14
 kontekst 3d, 267
 podstawowy szablon, 14
 rodzaje kontekstów, 310
 udostępnianie treści
 zastępczej, 309
 getContext(), 309
 isCanvasSupported(), 310
 WebGL, 267
 własna gra, 229
 Actor, 238
 aktualizacja danych, 251
 aktualizacja poziomu życia bohatera, 245
 arkusz sprite'ów bohatera, 233
 arkusz sprite'ów przeciwników, 233
 Controller, 246
 HealthBar, 245
 implementowanie silnika gry, 246
 Level, 243
 mapa obszarów, 236
 Model, 251
 najważniejsze cechy, 230
 obrazy poziomów, 234
 obsługa bohatera, 238
 stany gry, 251, 260, 262
 sterowanie postaciami, 242
 tworzenie dokumentu HTML, 265

uruchamianie, 265
 View, 260
 wyświetlanie poziomu, 243
 zarządzanie przebiegiem, 251

Controller, 246, 307
 addKeyboardListeners(), 248, 251
 handleKeyDown(), 249
 handleKeyUp(), 248
 initGame(), 248, 250
 loadImages(), 248
 resetGame(), 250
 copy, 52

D

destination-atop, 52
 destination-in, 52
 destination-out, 52
 destination-over, 52

E

Events, 165
 addRegionEventListener(), 169
 beginRegion(), 171
 clear(), 166
 closeRegion(), 172
 getCanvas(), 166
 getCanvasPos(), 166
 getContext(), 166
 getMousePos(), 168
 getTouchPos(), 168
 listen(), 167, 172
 procedury obsługi zdarzeń, 165
 reset(), 166
 setDrawStage(), 166, 171
 setMousePos(), 168
 setTouchPos(), 169
 tworzenie, 165
 współrzędne wskaźnika myszy, 172
 getMousePos(), 174

F

font, 30
 FPS, 157, 161
 wyświetlanie, 260
 funkcje
 addPoint(), 197
 applyPhysics(), 149, 152
 draw3dText(), 31, 32
 drawBranches(), 34
 drawClub(), 65
 drawDiamond(), 66
 drawFps(), 158, 161
 drawFrame(), 83, 90
 drawHeart(), 64
 drawImage(), 189
 drawImages(), 183, 184, 193
 drawLogo(), 120, 122
 drawMagnifier(), 195
 drawMicrobes(), 155
 drawMicrobes(), 157, 160
 drawPath(), 197
 drawSpade(), 63, 81
 drawStage(), 282, 285, 288, 293
 drawTriangle(), 44
 focusImage(), 99
 getCanvasImg(), 198
 getContextSupport(), 310
 getFrame(), 126
 getRandomAngle(), 121
 getRandomColor(), 153, 158
 getRandomSize(), 121
 getRandomTheta(), 153, 158
 getRandomX(), 120
 getRandomY(), 121
 imageMagnifier(), 191
 initBuffers(), 282, 284, 286, 290
 isAnimating(), 126
 loadCanvas(), 97
 loadImages(), 184
 loadTexture(), 294
 requestAnimationFrame(), 125
 stage(), 126, 130, 135, 138, 140, 146, 148, 152, 156, 157, 161, 166, 171, 175, 179, 183, 188, 194, 200, 283
 updateColorSequence(), 198
 updateMicrobes(), 153, 157, 158

writeMessage(), 173, 179, 182, 185, 188

G

Gear, 141, 144
 globalAlpha, 48, 50
 globalCompositeOperation, 56
 Graph, 216
 drawEquation(), 219, 220
 drawXAxis(), 217
 drawYAxis(), 218
 parametry, 220
 transformContext(), 219
 tworzenie, 216

H

HealthBar, 245
 draw(), 246
 setHealth(), 245
 HTML5, 11
 animacje, 315
 canvas, 11, 13
 API, 17
 efekty przejść, 315
 formaty wideo, 82
 fraktale, 33
 drawBranches(), 34
 konwersja obrazu na skalę szarości, 92
 kopiowanie fragmentów obrazu, 80
 drawImage(), 82
 krzywa Béziera, 25
 bezierCurveTo(), 25
 łączenie, 45
 punkt kontekstu, 25
 punkt końcowy, 25
 punkty kontrolne, 25
 niestandardowe
 przesunięcie, 110
 transform(), 111
 obracanie obrazu, 118
 drawImage(), 119
 rotate(), 119
 translate(), 119
 odwracanie kolorów obrazu, 88
 putImageData(), 89

odwracanie kolorów
 w klipach wideo, 90
 getImageData(), 91
 requestAnimationFrame(), 91
 operacje złożone, 51
 płótno, 18
 closePath(), 44
 createLinearGradient(), 44
 createPattern(), 45
 createRadialGradient(), 44
 fillStyle, 39
 fps, 100
 globalCompositeOperation, 56
 kontekst 2d, 18
 lineCap, 20
 lineJoin, 28
 lineWidth, 19
 lustrzane odbicie kontekstu, 109
 obrót kontekstu, 105, 118
 pochylenie kontekstu, 112
 przesuwanie kontekstu, 104
 rotate(), 105
 scale(), 107, 110
 shadowBlur, 33
 shadowColor, 33
 shadowOffsetX, 33
 shadowOffsetY, 33
 skalowanie kontekstu, 107
 stos stanu, 49
 strokeStyle, 19
 pobieranie danych obrazu, 86
 getImageData(), 87
 przekształcanie okręgu na owal, 116
 scale(), 117
 translate(), 117
 przekształcenia
 z wykorzystaniem stosu stanu, 113
 restore(), 115
 save(), 115
 rysowanie krzywej kwadratowej, 23
 punkt kontekstu, 24
 punkt kontrolny, 24
 punkt końcowy, 24
 quadraticCurveTo(), 24

HTML5

- rysowanie linii, 18
 - lineCap, 20
 - lineTo(), 19
 - lineWidth, 19
 - moveTo(), 19
 - stroke(), 20
 - strokeStyle, 19
- rysowanie logo, 120
 - bezierCurveTo(), 122
 - drawLogo(), 122
- rysowanie łuku, 21
 - arc(), 21
 - arcTo(), 22
- rysowanie okręgu, 40
 - arc(), 41
 - fill(), 41
- rysowanie prostokąta, 38
 - fill(), 39
 - fillRect(), 39
 - rect(), 39
 - strokeRect(), 39
- rysowanie przezroczystego koła, 47
 - globalAlpha, 48
- rysowanie spirali, 28
 - lineTo(), 29
 - stroke(), 29
- rysowanie symboli kolorów w talii kart, 63
 - drawClub(), 65
 - drawDiamond(), 66
 - drawHeart(), 64
 - drawSpade(), 63
- rysowanie trójkąta, 42
 - beginPath(), 44
 - closePath(), 44
 - drawTriangle(), 44
 - lineTo(), 44
 - moveTo(), 44
- rysowanie trójwymiarowego tekstu, 31
 - draw3dText(), 32
- rysowanie zygzaków, 26
 - beginPath(), 28
 - lineJoin, 28
 - moveTo(), 28
- stosowanie wartości losowych, 60
- tworzenie dokumentu, 265

URL, 93

- getDataURL(), 97
- konwersja obrazu, 93
- loadCanvas(), 98
- request.responseText, 98
- toDataURL(), 94
- wyświetlanie, 97
- zapisywanie obrazu, 96
- wklejanie fragmentów obrazu, 80
 - drawImage(), 82
- wycinanie fragmentu obrazu, 78
 - drawImage(), 79
- wykorzystanie pętli, 56
- wyostrzanie obrazu, 99
 - setInterval(), 102
- wyświetlanie klipów wideo, 83
 - drawFrame(), 83
 - drawImage(), 83
 - requestAnimFrame(), 84
- wyświetlanie obrazu, 76
 - drawImage(), 77
 - newImage(), 77
 - onload, 77
- wyświetlanie tekstu, 30
 - fillStyle, 30
 - fillText(), 31
 - font, 30
 - strokeText(), 31
 - textAlign, 30
 - textBaseline, 30

K

klasy

- Actor, 238
- Animation, 125, 127
- BarChart, 210
- Controller, 246, 307
- Events, 165
- Flower, 62
- Gear, 141, 144
- Graph, 216
- HealthBar, 245
- Level, 243
- LineChart, 222
- Model, 251, 307
- PieChart, 204

View, 260, 307

- WebGL, 268
- kontekst 2d, 14

L

- Level, 243
 - draw(), 243
 - getZoneInfo(), 244
 - setBoundsData(), 243, 244
- lighter, 52
- lineCap, 20
- LineChart, 222
 - drawLine(), 224, 227
 - drawXAxis(), 223
 - drawYAxis(), 224
 - getLongestValueWidth(), 223
 - transformContext(), 225
 - tworzenie, 222
 - właściwości, 227
- lineJoin, 28
- lineWidth, 19, 39

M

- mapy obszarów, 234
 - alternatywy, 237
 - inBounds, 245
 - levitating, 245
 - tworzenie, 236
- metody
 - addColorStop(), 44, 47
 - addKeyboardListeners(), 248, 251
 - addRegionEventListener(), 169, 184, 187
 - animationLoop(), 127, 271
 - arc(), 21, 41, 137, 148, 208
 - arcTo(), 22
 - attacheListeners(), 299
 - attack(), 239
 - beginPath(), 28, 44
 - beginRegion(), 169, 171, 175, 179, 184, 187
 - bezierCurveTo(), 122
 - clear(), 126, 128, 166, 270
 - closePath(), 44
 - closeRegion(), 169, 172, 176, 179
 - createArrayBuffer(), 277

createElementArrayBuffer(), 277
 createLinearGradient(), 44, 59
 createPattern(), 45
 createRadialGradient, 46
 createRadialGradient(), 44
 damage(), 242
 document.getElementById(), 14, 19
 draw(), 62, 141, 145, 240, 242, 243, 246
 drawArrays(), 279, 282, 283, 284
 drawBadGuys(), 261
 drawBars(), 212, 215
 drawCeiling(), 304
 drawCrates(), 305
 drawElements(), 279, 288, 289, 293
 drawEquation(), 220
 drawFloor(), 304
 drawFps(), 261
 drawGridlines(), 213
 drawImage(), 77, 79, 82, 119, 313
 drawLegend(), 208
 drawLine(), 224, 227
 drawPieBorder(), 205, 208
 drawScreen(), 262
 drawSlices(), 208
 drawWalls(), 305
 drawXAxis(), 214, 217, 223
 drawXLabels(), 212
 drawYAxis(), 214, 218, 224
 drawYValues(), 212
 enableLighting(), 279, 295
 fade(), 240
 fill(), 39, 41, 47
 fillRect(), 39
 fillText(), 31, 148, 314
 getCanvas(), 166
 getCenter(), 242
 getCanvasPos(), 166
 getContext(), 19, 125, 166, 309
 getDataURL(), 97
 getFps(), 127, 271
 getFragmentShaderGLSL(), 272
 getFrame(), 270
 getImageData(), 85, 87, 91, 314
 getLabelAreaHeight(), 211
 getLegendWidth(), 205
 getLongestValueWidth(), 211, 223
 getMousePos(), 168, 174, 298
 getTime(), 128, 271
 getTimeInterval(), 126, 128, 271
 getTotalValue(), 206
 getTouchPos(), 168
 getVertexShaderGLSL(), 273
 getZoneInfo(), 244
 handleKeyDown(), 249, 298
 handleKeyUp(), 248, 299
 handleMouseDown(), 298
 handleMouseMove(), 298
 identity(), 275, 283
 init(), 294
 initBadGuys(), 255
 initBuffers(), 303
 initColorShader(), 276
 initCratePosition(), 301
 initCubeBuffers(), 301
 initFloorBuffers(), 302
 initGame(), 248, 250
 initHealthBar(), 254
 initHero(), 254
 initLevel(), 254
 initLightingShader(), 277
 initNormalShader(), 276
 initPositionShader(), 276
 initShaders(), 274
 initTexture(), 277
 initTextureShader(), 276
 initWallBuffers(), 303
 isAnimating(), 270
 isCanvasSupported(), 310
 isFacingRight(), 239
 isPointInPath(), 171
 jump(), 240
 lineTo(), 19, 29
 listen(), 167, 172
 loadImages(), 248
 loadTextures(), 297
 measureText(), 314
 moveBadGuys(), 256, 260
 moveLeft(), 240
 moveRight(), 239
 moveTo(), 19, 28, 44
 nearby(), 259
 perspective(), 275, 283
 pushColorBuffer(), 278
 pushIndexBuffer(), 278
 pushNormalBuffer(), 278
 pushPositionBuffer(), 278, 283
 pushTextureBuffer(), 278
 putImageData(), 89
 rect(), 39, 134
 removeDefeatedBadGuys(), 252
 requestAnimationFrame(), 124, 127, 269
 requestAnimFrame(), 84, 91, 127, 269
 reset(), 166
 resetGame(), 250
 restore(), 50, 115, 272
 rotate(), 105, 119, 276, 285
 save(), 50, 115, 272
 scale(), 107, 110, 117
 setAmbientLighting(), 279, 295
 setBoundsData(), 243, 244
 setDirectionalLighting(), 280, 295
 setDrawStage(), 126, 166, 171, 270
 setHealth(), 245
 setInterval(), 102
 setMatrixUniforms(), 279
 setMousePos(), 168
 setShaderProgram(), 275, 295
 setStage(), 294
 setTouchPos(), 169
 setTransform(), 111
 stage(), 134, 262, 306
 start(), 126, 128, 130, 134, 137, 140, 145, 148, 152, 285
 startAnimation(), 270
 stop(), 126, 128, 130, 132, 239
 stopAnimation(), 271
 stroke(), 20, 27, 29, 148
 strokeRect(), 39
 strokeText(), 31, 314
 toDataURL(), 94, 201, 314

metody

- transform(), 111, 113
- transformContext(), 219, 225
- translate(), 117, 119, 148, 276
- updateActor(), 258
- updateActorVY(), 258
- updateActorX(), 259
- updateActorY(), 258
- updateBadGuys(), 252
- updateCameraPos(), 303
- updateHeroCanvasPos(), 257
- updateLevel(), 257
- updateSpriteMotion(), 240, 242
- updateSpriteSeqNum(), 241, 242
- updateStage(), 253
- writeMessage(), 175

Model, 251, 307

- initBadGuys(), 255
- initHealthBar(), 254
- initHero(), 254
- initLevel(), 254
- moveBadGuys(), 256, 260
- nearby(), 259
- removeDefeatedBadGuys(), 252
- updateActor(), 258
- updateActorVY(), 258
- updateActorX(), 259
- updateActorY(), 258
- updateBadGuys(), 252
- updateHeroCanvasPos(), 257
- updateLevel(), 257
- updateStage(), 253
- zadania, 260

mousedown, 176, 187, 190, 199, 201

mousemove, 176, 187

mouseout, 176, 187, 190

mouseover, 176, 187, 190

mouseup, 176, 187, 190, 200, 201

O

obraz, 75

- animacja, 124
- konwersja na skalę szarości, 92
- kopiowanie fragmentu, 80
 - drawImage(), 82

lustrzane odbicie kontekstu

- plótina, 109
- scale(), 110

niestandardowe przesunięcie, 110

- transform(), 111

obraccanie, 118

- drawImage(), 119
- rotate(), 119
- translate(), 119

obrót kontekstu plótina, 105

- rotate(), 105

odwracanie kolorów, 88

- putImageData(), 89

pobieranie danych, 86

- getImageData(), 87

pochylanie kontekstu plótina, 112

- transform(), 113

powiększanie fragmentu, 190, 195

procedury obsługi zdarzeń, 181

- beginRegion(), 184
- mousedown, 199, 201
- mouseup, 200, 201

prosta aplikacja graficzna, 196

- isMouseDown, 198
- mousedown, 199, 201
- mouseup, 200, 201
- podstawowe cechy, 201

przekształcanie okręgu na owal, 116

- scale(), 117
- translate(), 117

przekształcenia z wykorzystaniem stosu stanu, 113

- restore(), 115
- save(), 115

przesuwanie kontekstu plótina, 104

przycinanie, 77

- drawImage(), 79

RGB, 85

- konwersja na skalę szarości, 93

rysowanie logo, 120

- bezierCurveTo(), 122

- drawLogo(), 122

skalowanie kontekstu plótina, 107

przekształcanie okręgu na owal, 116

- scale(), 107

technika przeciągnij i upuść, 188

- mousedown, 190
- mouseout, 190
- mouseover, 190
- mouseup, 190

URL, 93

- getDataURL(), 97
- konwersja, 93
- loadCanvas(), 98
- request.responseText, 98
- toDataURL(), 94
- wyświetlanie, 97
- zapisywanie, 96

wklejanie fragmentu, 80

- drawImage(), 82

wyostrzanie, 99

- setInterval(), 102

wyświetlanie, 76

- drawImage(), 77
- new Image(), 77
- onload, 77

obrazy poziomów, 234

- alternatywy, 237
- układ mozaiki, 238
- wczytywanie z opóźnieniem, 237
- tworzenie, 234
- wyświetlanie, 243

onload, 77

P

PieChart, 204

- drawLegend(), 207, 208
- drawPieBorder(), 205, 208
- drawSlices(), 206, 208
- getLegendWidth(), 205
- getTotalValue(), 206
- tworzenie, 204

procedury obsługi zdarzeń, 165

- dotyk, 178
- beginRegion(), 179

closeRegion(), 179
 touchend, 180, 181
 touchmove, 179
 touchstart, 180
 isPointInPath(), 171
 mousedown, 176, 187, 190,
 199, 201
 mousemove, 174, 176, 187
 mouseout, 174, 176, 187, 190
 mouseover, 176, 187, 190
 mouseup, 176, 187, 190,
 200, 201
 mysz, 174
 beginRegion(), 175
 closeRegion(), 176
 mousedown, 176
 mousemove, 176
 mouseout, 176
 mouseover, 176
 mouseup, 176
 writeMessage(), 175
 obraz, 181
 addRegionEventListener(),
 184
 beginRegion(), 184
 powiększanie fragmentu
 grafiki, 190, 195
 technika przeciągnij i upuść,
 185, 188
 addRegionEventListener(),
 187
 beginRegion(), 187
 fazy, 187
 mousedown, 187, 190
 mousemove, 187
 mouseout, 187, 190
 mouseover, 187, 190
 mouseup, 187, 190
 touchend, 180, 181
 touchmove, 179
 touchstart, 180, 181

R

receptury
 animacja
 koła zębate, 140
 zegar, 145

klipy wideo
 odwracanie kolorów, 89
 wyświetlanie, 82
 kontekst płót
 obrót, 105
 przesuwanie, 104
 pochylanie, 112
 skalowanie, 107
 niestandardowe
 przesunięcie, 110
 obraz
 konwersja kolorów
 rysunku na skalę
 szarości, 91
 kopiowanie fragmentów, 80
 obracanie, 118
 odwracanie kolorów, 87
 pobieranie danych, 84
 powiększanie fragmentu,
 190
 przycinanie, 77
 wyostrzanie obrazka
 o powiększonych
 pikselach, 99
 wklejanie fragmentów, 80
 operacje złożone, 51
 procedury obsługi zdarzeń
 dołączanie do obrazków, 181
 dotyku, 178
 myszy, 174
 przeciągnij i upuść
 kształt, 185
 obrazek, 188
 przekształcenia
 okrąg na owal, 116
 rysunek na postać danych
 URL, 93
 wykorzystanie stosu stanu,
 113
 ruchome wahadło, 137
 rysowanie
 chmurka, 45
 koło zębate, 56
 krzywa Béziera, 25
 krzywa kwadratowa, 23
 linia, 18
 łąka kwiatów, 59
 łuk, 21
 nawiedzone drzewo, 33

odrzutowiec, 67
 okrąg, 40
 proste logo, 119
 prostokąt, 38
 przezroczyste kształty, 47
 spirala, 28
 trójkąt, 281
 trójwymiarowy tekst
 z cieniem, 31
 zygzak, 26
 style wypełnienia, 42
 symulacja fizyki cząstek, 149
 tekstury i oświetlenie, 290
 tworzenie
 aplikacja graficzna, 196
 funkcja rysująca
 niestandardowe
 kształty, 62
 mikroskopijne żyłki, 153
 obracający się sześcián,
 286
 odbicie lustrzane, 109
 oscylacje, 133
 przyspieszenie, 130
 ruch liniowy, 128
 trójwymiarowy świat, 296
 własne kształty, 42
 tworzenie klasy
 Actor, 238
 Animation, 124
 Controller, 246
 Events, 164
 HealthBar, 245
 Level, 243
 Model, 251
 upraszczająca korzystanie
 z API WebGL, 268
 View, 260
 tworzenie wykresu
 kołowy, 204
 liniowy, 221
 słupkowy, 209
 wizualizacja równań, 216
 wyświetlanie
 klipów wideo, 82
 obrazu, 76
 tekstu, 30
 współrzędne myszy, 172
 rect(), 39, 134

removeDefeatedBadGuys(), 252
 request.responseText, 98
 requestAnimationFrame(), 125, 127, 269
 requestAnimFrame(), 84, 91, 127, 269
 reset(), 166
 resetGame(), 250
 restore(), 50, 115, 272
 RGB, 85
 konwersja na skalę szarości, 93
 rotate(), 105, 119, 276, 285
 rysowanie, 17
 chmurka, 45
 addColorStop(), 47
 createRadialGradient, 46
 fill(), 47
 drzewo, 33
 drawBranches(), 34
 koło zębate, 56
 createLinearGradient(), 59
 krzywa Béziera, 25
 bezierCurveTo(), 25
 łączenie, 45
 punkt kontekstu, 25
 punkt końcowy, 25
 punkty kontrolne, 25
 krzywa kwadratowa, 23
 punkt kontekstu, 24
 punkt kontrolny, 24
 punkt końcowy, 24
 quadraticCurveTo(), 24
 kwiaty, 60
 draw(), 62
 linia, 18
 lineCap, 20
 lineTo(), 19
 lineWidth, 19
 moveTo(), 19
 stroke(), 20
 strokeStyle, 19
 logo, 120
 bezierCurveTo(), 122
 drawLogo(), 122
 luk, 21
 arc(), 21
 arcTo(), 22
 odrzutowiec, 68

okrag, 40
 arc(), 41
 fill(), 41
 prostokąt, 38
 fill(), 39
 fillRect(), 39
 rect(), 39
 strokeRect(), 39
 przezroczyste koło, 47
 globalAlpha, 48
 spirala, 28
 lineTo(), 29
 stroke(), 29
 style wypełnień, 42
 gradienty kołowe, 44
 gradienty liniowe, 44
 kolory gradientu, 44
 wzorce, 45
 symbole kolorów w talii kart, 63
 drawClub(), 65
 drawDiamond(), 66
 drawHeart(), 64
 drawSpade(), 63
 trójkąt, 42
 beginPath(), 44
 closePath(), 44
 drawTriangle(), 44
 lineTo(), 44
 moveTo(), 44
 trójwymiarowy tekst, 31
 draw3dText(), 32
 zygzak, 26
 beginPath(), 28
 lineJoin, 28
 moveTo(), 28

S

shadowBlur, 33
 shadowColor, 33
 shadowOffsetX, 33, 148
 shadowOffsetY, 33, 148
 source-atop, 52
 source-in, 52
 source-out, 52
 source-over, 52
 stos stanu kontekstu, 113
 odtworzenie przekształceń, 114
 restore(), 115

 zapisywanie przekształceń, 114
 save(), 115
 stos stanu płótna, 49
 dodanie na stos, 49
 odczyt wierzchołka, 49
 pobranie ze stosu, 49
 przywracanie stanu kontekstu, 50
 restore(), 50
 struktura danych, 49
 zapisanie stanu kontekstu, 50
 save(), 50

T

tekst, 30
 rysowanie trójwymiarowego tekstu, 31
 draw3dText(), 32
 wyświetlanie w elemencie canvas, 30
 fillStyle, 30
 fillText(), 31
 font, 30
 strokeText(), 31
 textAlign, 30
 textBaseline, 30
 textAlign, 30
 textBaseline, 30
 touchend, 180, 181
 touchmove, 179, 181
 touchstart, 180, 181

V

View, 260, 307
 drawBadGuys(), 261
 drawFps(), 261
 drawScreen(), 261, 262
 stage(), 262

W

WebGL, 267, 268
 animationLoop(), 271
 clear(), 270
 createArrayBuffer(), 277
 createElementArrayBuffer(), 277

- dodawanie tekstur
 - i oświetlenia, 290
 - bufor normalnych, 295
 - bufor tekstur, 295
 - drawElements(), 293
 - enableLighting(), 295
 - init(), 294
 - oświetlenie kierunkowe, 295
 - oświetlenie otoczenia, 295
 - setAmbientLighting(), 295
 - setDirectionalLighting(), 295
 - setShaderProgram(), 295
 - setStage(), 294
- drawArrays(), 279
- drawElements(), 279
- enableLighting(), 279
- experimental-webgl, 283
- getFps(), 271
- getFragmentShaderGLSL(), 272
- getFrame(), 270
- getTime(), 271
- getTimeInterval(), 271
- getVertexShaderGLSL(), 273
- identity(), 275
- initColorShader(), 276
- initLightingShader(), 277
- initNormalShader(), 276
- initPositionShader(), 276
- initShaders(), 274
- initTexture(), 277
- initTextureShader(), 276
- isAnimating(), 270
- obracanie trójkąta, 284
 - drawArrays(), 284
 - rotate(), 285
 - start(), 285
- określanie dostępnych kontekstów, 310
- operacje na macierzach, 280
- perspective(), 275
- prosty model trójwymiarowy, 286
 - bufor indeksów, 289
 - bufor kolorów, 289
 - bufor położenia, 289
 - drawElements(), 288, 289
 - pushColorBuffer(), 278
 - pushIndexBuffer(), 278
 - pushNormalBuffer(), 278
 - pushPositionBuffer(), 278
 - pushTextureBuffer(), 278
 - requestAnimationFrame(), 269
 - requestAnimFrame(), 269
 - restore(), 272
 - rotate(), 276
 - rysowanie trójkąta, 281
 - drawArrays(), 282, 283
 - identity(), 283
 - perspective(), 283
 - pushPositionBuffer(), 283
 - save(), 272
 - setAmbientLighting(), 279
 - setDirectionalLighting(), 280
 - setDrawStage(), 270
 - setMatrixUniforms(), 279
 - setShaderProgram(), 275
 - shadery, 280
 - BLUE_COLOR, 283
 - TEXTURE_DIRECTIONAL_LIGHTING, 295
 - VARYING_COLOR, 288
 - startAnimation(), 270
 - stopAnimation(), 271
 - translate(), 276
 - trójwymiarowy świat, 296
 - attacheListeners(), 299
 - bufory podłogi, 296
 - bufory sześcianów, 296
 - bufory ścian, 296
 - Controller, 307
 - drawCeiling(), 304
 - drawCrates(), 305
 - drawFloor(), 304
 - drawWalls(), 305
 - getMousePos(), 298
 - handleKeyDown(), 298
 - handleKeyUp(), 299
 - handleMouseDown(), 298
 - handleMouseMove(), 298
 - initBuffers(), 303
 - initCratePosition(), 301
 - initCubeBuffers(), 301
 - initFloorBuffers(), 302
 - initWallBuffers(), 303
 - loadTextures(), 297
 - Model, 307
 - stage(), 306
 - tworzenie, 296
 - updateCameraPos(), 303
 - View, 307
- wideo, 82
 - formaty, 82
 - H.264, 82
 - odwracanie kolorów w klipach, 90
 - getImageData(), 91
 - requestAnimFrame(), 91
 - Ogg Theora, 82
 - WebM, 82
 - wyświetlanie klipów, 83
 - drawFrame(), 83
 - drawImage(), 83
 - requestAnimFrame(), 84
- window.onload, 19
- wykresy, 203
 - kołowy, 204
 - arc(), 208
 - drawLegend(), 207, 208
 - drawPieBorder(), 205, 208
 - drawSlices(), 206, 208
 - getLegendWidth(), 205
 - getTotalValue(), 206
 - PieChart, 204
- liniowy, 221
 - drawLine(), 224, 227
 - drawXAxis(), 223
 - drawYAxis(), 224
 - getLongestValueWidth(), 223
 - LineChart, 222
 - transformContext(), 225
- slupkowy, 209
 - BarChart, 210
 - drawBars(), 212, 215
 - drawGridlines(), 213
 - drawXAxis(), 214
 - drawXLabels(), 212
 - drawYAxis(), 214
 - drawYValues(), 212
 - getLabelAreaHeight(), 211

getLongestValueWidth(), 211	drawXAxis(), 217	X
wizualizacja równań, 216	drawYAxis(), 218	
drawEquation(), 219, 220	Graph, 216	
	transformContext(), 219	
		xor, 52

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA



Helion SA

HTML5 Canvas

Receptury

HTML5 szturmem zdobywa rynek stron (a w zasadzie już aplikacji) WWW. Co sprawia, że tak się dzieje? Ogrom nowych funkcji, które otwierają przed projektantami nowe możliwości. Usługi geolokalizacyjne, lokalne przechowywanie plików lub obsługa plików multimedialnych to tylko niektóre z nich. Wśród nowości jest również ta jedna wzbudzająca najwięcej emocji — *canvas* (płótno). Jest to element, który rewolucjonizuje grafikę i wizualizację na stronach WWW.

Dzięki tej książce błyskawicznie zaczniesz korzystać z potencjału elementu *canvas*. W trakcie lektury poznasz podstawowe możliwości API tego elementu, a wraz z kolejnymi stronami zaczniesz wykorzystywać coraz bardziej zaawansowane techniki tworzenia animacji, wykonywania operacji na obrazach i pisania gier. Rysowanie łuków, przekształcenia i obroty nie będą stanowiły dla Ciebie żadnego problemu. Ponadto w mgnieniu oka opanujesz zasady tworzenia wykresów oraz wizualizacji przestrzennych. Ta książka zasługuje na Twoją szczególną uwagę. Sięgnij po nią i twórz oszałamiające strony w sieci!

Element *canvas* to:

- nowość w języku HTML5
- nowe możliwości graficzne na Twojej stronie
- efektowna wizualizacja danych
- baza do tworzenia gier dla przeglądarek

Gotowe przepisy do wykorzystania na Twojej stronie!

helion.pl
księgarnia
internetowa

(Nr katalogowy: 11931)

Księgarnia internetowa:
<http://helion.pl>

Zamówienia telefoniczne:
0 801 339900
0 601 339900



Helion

Sprawdź najnowsze promocje:
• <http://helion.pl/promocje>
Książki najchętniej czytane:
• <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
• <http://helion.pl/news>

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 210 98 63
e-mail: helion@helion.pl
<http://helion.pl>

sięgnij po **WIĘCEJ**



KOD KORZYSCI

ISBN 978-83-246-5075-0



Cena: 59,00 zł

Informatyka w najlepszym wydaniu