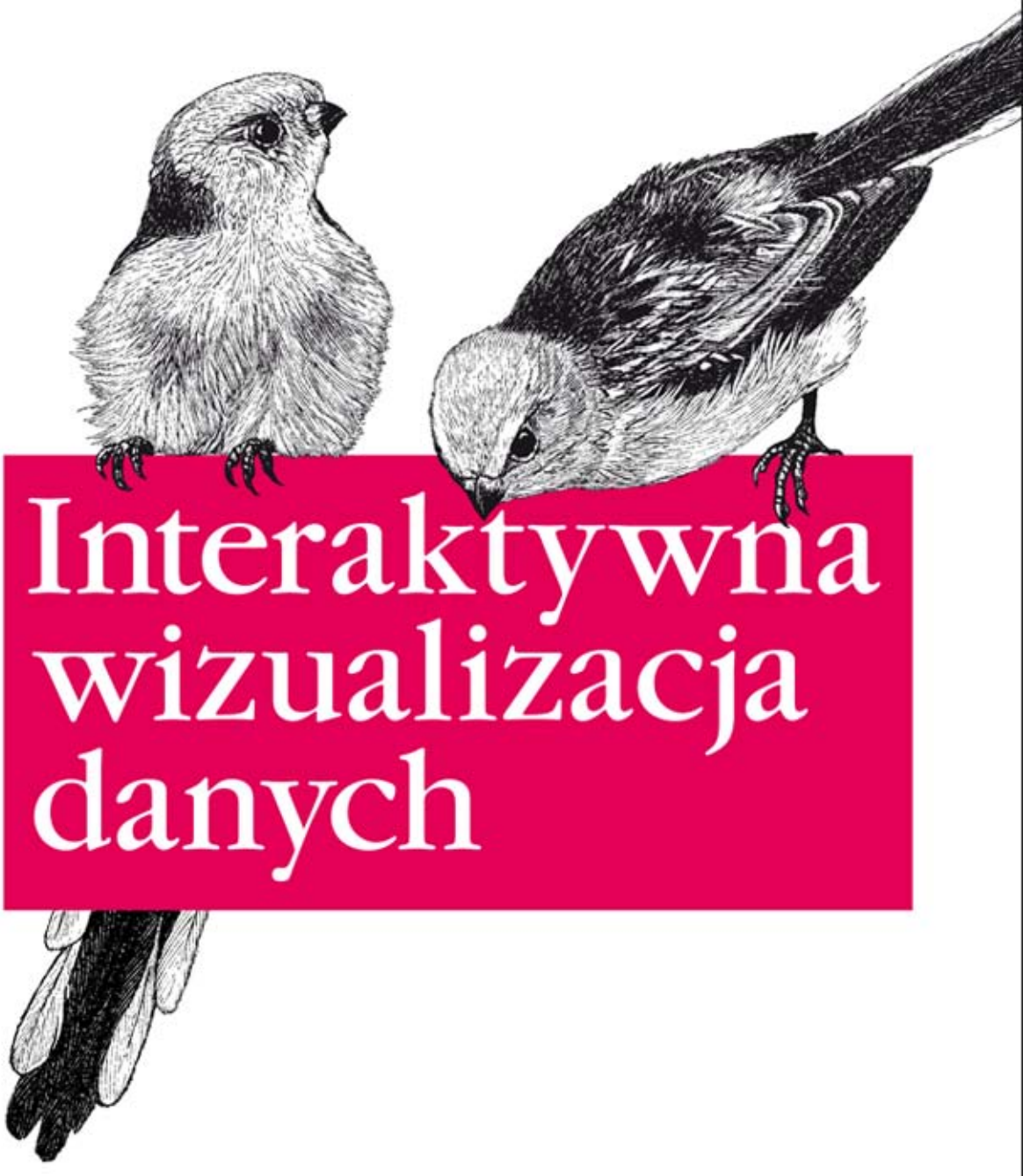
Uatrakcyjnij swoją stronę WWW!



Interaktywna wizualizacja danych

HELION

O'REILLY®

Scott Murray

Tytuł oryginału: Interactive Data Visualization for the Web

Tłumaczenie: Julia Szajkowska

ISBN: 978-83-246-8172-3

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)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/inwida
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/inwida.zip

Printed in Poland.

Spis treści

Przedmowa **9**

1. Wprowadzenie **13**
Dlaczego warto przedstawiać dane graficznie? 13
Dlaczego programować? 14
Po co nam interaktywność? 14
Dlaczego w internecie? 15
O czym jest ta książka? 16
Kim jesteś? 16
Czego nie znajdziesz w tej książce? 17
Kod w przykładach 17
Dziękuję! 18

2. Wprowadzenie do D3 ..... **19**
Funkcje biblioteki 19
Funkcje, których nie znajdziesz w bibliotece 20
Początki i tło historyczne 21
Inne możliwości 22
Wykresy od ręki 23
Grafy 24
Odwzorowania kartograficzne 24
Prawie od zera 25
W trzech wymiarach 25
Narzędzia przygotowane za pomocą biblioteki D3 26

3. Podstawy wykorzystywanych technologii **27**
Kod HTML 29
Treść i struktura 30
Budowanie struktury znacznikami 31
Najczęściej stosowane znaczniki 31

Atrybuty 33
Klasy i identyfikatory 34
Komentarze 34
DOM 35
Narzędzia programisty 35
Renderowanie i model pudełkowy 38
Kaskadowe arkusze stylów CSS 39
Selektory 40
Właściwości i wartości 41
Komentarze 41
Wprowadzanie stylów na stronę 41
Dziedziczenie, kaskadowość i szczegółowość 43
JavaScript 44
Witaj, konsolo! 45
Zmienne 45
Inne typy zmiennych 46
Tablice 46
Obiekty 47
Obiekty i tablice 48
Operatory matematyczne 50
Operatory porównania 50
Instrukcje sterujące 51
Funkcje 53
Komentarze 54
Wprowadzanie skryptów na stronę 54
Sposoby na JavaScript 54
SVG 58
Znacznik SVG 59
Proste kształty 59
Nadawanie stylów znacznika SVG 61
Warstwy i kolejność rysowania 63
Przezroczystość 63
O kompatybilności 65

4. Przygotowania do pracy 67
Pobieranie biblioteki D3 67
Odwoływanie się do biblioteki D3 68
Przygotowanie serwera stron WWW 69
Terminal z interpreterem języka Python 69
MAMP, WAMP i LAMP 70
W drogę 70

5. Dane 71
Tworzenie znaczników 71
Łączenie metod 73
Po jednym odwołaniu 74
Przekazywanie 75
Bez łączenia 75
Dowiązywanie danych 75
W więzi 76
Dane 76
Proszę dokonać wyboru 79
Powiązany i określony 80
Używanie danych 82
Funkcja — rzecz potrzebna 83
Dane chcą być gdzieś zapisane 84
Więcej niż tekst 85

6. Graficzne przedstawianie danych 87
Rysowanie za pomocą znaczników div 87
Określanie atrybutów 88
Kilka słów o klasach 89
Wróćmy do słupków 89
Nadawanie stylów 90
Potężna metoda data() 91
Dane losowe 92
Rysowanie za pomocą znaczników SVG 94
Tworzenie znaczników SVG 95
Kształty definiowane danymi 96
Jakie śliczne kolorki! 97
Przygotowywanie wykresu słupkowego 98
Stary wykres 98
Nowy wykres 99
Kolory 104
Etykiety 106
Przygotowywanie wykresu punktowego 108
Dane 108
Wykres punktowy 108
Rozmiar 110
Etykiety 111
Dalsze działania 112

7. Skalowanie 113
Jabłka i piksele 113
Dziedziny i zbiory wartości 114
Normalizacja 115

Przygotowywanie funkcji skalującej 115
Skalowanie wykresu punktowego 116
Funkcje d3.min() i d3.max() 116
Skalowanie dynamiczne 118
Stosowanie wartości przeskalowanych 118
Poprawianie wykresu 119
Inne rozwiązania 122
Inne funkcje skalujące 123

8. Osie ... 125
Poznaj osie 125
Metody przygotowywania osi 126
Porządki 127
Pod kreską 130
A dlaczego nie Y? 131
Ostateczny szlif 132
Formatowanie opisów osi 133

9. Aktualizacje, przejścia i ruch ... 135
Przeróbki w wykresie słupkowym 135
Jak działają skale porządkowe? 136
W zbiorze wartości tylko grupy 138
Referencja do skali porządkowej 139
Inne aktualizacje 139
Aktualizowanie danych 139
Interakcja za pomocą funkcji nasłuchujących zdarzeń 140
Zmienianie danych 141
Aktualizowanie elementów graficznych 141
Przejścia 144
Funkcja duration(), czyli ile to wszystko potrwa? 145
Ostrożnie, osiadamy! 146
Bez opóźnień, proszę! 147
Dane losowe 149
Aktualizowanie funkcji skalujących 150
Aktualizowanie osi 153
Każde przejście ma swój początek i koniec 155
Inne rodzaje aktualizacji 161
Dodawanie wartości (i znaczników) 162
Usuwanie wartości (i znaczników) 166
Łączenie danych za pomocą kluczy 169
Dodaj i usuń, czyli wszystko naraz 174
Powtórzenie 175

10. Interaktywność 177
Dowiązywanie funkcji nasłuchujących zdarzeń 177
Poznajmy zachowania 178
Najedź kursorem, żeby wyróżnić 179
Grupowanie znaczników SVG 182
Sortowanie kliknięciem 184
Podpowiedzi 187
Domyślne podpowiedzi przeglądarki 188
Podpowiedzi wyświetlane w znacznikach SVG 189
Podpowiedzi wyświetlane w znacznikach div 191
Urządzenia dotykowe 193
Pora ruszać dalej 193

11. Układy wykresów 195
Układ kołowy 196
Układ skumulowany 200
Układ siłowy 203

12. Mapy geograficzne 209
Poznaj GeoJSON 209
Ścieżki 211
Odwzorowania 212
Kartogram 214
Dodawanie punktów 217
Pobieranie danych geograficznych i ich parsowanie 221
Znajdź pliki kształtów 221
Wybierz rozdzielczość 221
Upraszczanie kształtów 223
Konwersja do GeoJSON 223

13. Eksportowanie 227
Mapy bitowe 227
Plik PDF 228
SVG 229

A Dalsza nauka 233

Skorowidz 237

ROZDZIAŁ 12.

Mapy geograficzne

Wykresy słupkowe, wykresy punktowe, wykresy kołowe, a nawet wykresy siłowe… „Wszystko to bardzo pięknie — myślisz sobie — ale przejdźmy już w końcu do map!”.

Poznaj GeoJSON

Standard JSON już znasz, pora zaznajomić się teraz z GeoJSON, czyli bazującym na formacie JSON standardzie zapisywania danych geograficznych na potrzeby aplikacji internetowych. GeoJSON nie jest niczym nowym, a jedynie wysoce wyspecjalizowaną metodą stosowania JSON.

Zanim jednak przejdziemy do generowania map, musimy w jakiś sposób zdobyć dane znacznika `path` (czyli konturu) kształtu, który będziemy chcieli wyświetlić. Zaczniemy od najprostszego przykładu — odwzorowania granic Stanów Zjednoczonych. W katalogu z kodami przykładów znajdziesz plik *us-states.json*, który został pobrany bezpośrednio z jednego z licznych przykładów na stronie D3, w związku z czym jesteśmy winni Mike'owi Bostockowi gorące podziękowania za przygotowanie takich ładnych i starannych granic Stanów.

Po otwarciu pliku *us-states.json* zobaczysz coś, co przypomina nieco ten kod (choć tu prezentuję wersję sformatowaną i bardzo skróconą).

```
{
   "type":"FeatureCollection",
   "features":[
   {
      "type":"Feature",
      "id":"01",
      "properties":{"name":"Alabama"},
      "geometry":{
         "type":"Polygon",
         "coordinates":[[[-87.359296,35.00118],
            [-85.606675,34.984749],[-85.431413,34.124869],
            [-85.184951,32.859696],[-85.184951,32.859696],
            [-84.960397,32.421541],[-85.004212,32.322956],
            [-84.889196,32.262709],[-85.058981,32.13674]...
         ]]
      }
   },
{
   "type": "Feature",
   "id": "02",
   "properties": { "name": "Alaska" },
```

```
      "geometry": {
         "type": "MultiPolygon",
         "coordinates": [[[[-131.602021,55.117982],
            [-131.569159,55.28229],[-131.355558,55.183705],
            [-131.38842,55.01392],[-131.645836,55.035827],
            [-131.602021,55.117982]]],[[[-131.832052,55.42469],
            [-131.645836,55.304197],[-131.749898,55.128935],
            [-131.832052,55.189182],...
         ]]]
      }
   }
   ...
```

Dane, jak to w stylu GeoJSON, są zapisane w jednym wielkim obiekcie. (Nawiasy klamrowe, pamiętasz?). W obiekcie znajdziemy deklarację typu `type: FeatureCollecion`. Po typie pojawia się pole `features`, które przechowuje tablicę pojedynczych obiektów typu `Feature`. Każdy z nich odpowiada jednemu ze stanów. Nazwa stanu jest podana w polu `properties`.

Jednak sedno każdego pliku GeoJSON leży w polu `geometry`. Tam właśnie pojawiają się definicja typu (`type`) oraz współrzędne (`coordinates`) określające kontur danego elementu. Pole `coordinates` przechowuje szerokości i długości geograficzne zapisane parami w małych, dwuwartościowych tablicach. Przetwarzanie i udoskonalanie tych wartości stanowi sens życia każdego kartografa. Pokolenia badaczy i podróżników poświęcały swój czas, by zgromadzić ten zbiór, i za to powinniśmy być im wdzięczni. Liczby te, choć może niepozorne, kryją w sobie prawdziwą moc.

Warto też zaznaczyć, że w danych GeoJSON jako pierwszą podaje się zawsze *długość geograficzną*. Mimo że jesteśmy przyzwyczajeni do zapisu, w którym najpierw wymienia się szerokość, a potem długość, w świecie GeoJSON obowiązuje odwrotny porządek.

I jeszcze mała ściąga na wypadek, gdyby odróżnianie długości od szerokości geograficznej stanowiło dla Ciebie problem:

- Długość geograficzna, jak nazwa wskazuje, jest długa. Jej linie (południki) biegną pionowo (z północy na południe), jakby zwieszały się z góry na dół.
- Szerokość geograficzna jest szeroka. Jej linie (równoleżniki) rozciągają się poziomo, jakby starały się objąć Ziemię w pasie.

Równoleżniki i południki tworzą razem ogromną sieć obejmującą całą kulę ziemską. Na nasze szczęście współrzędne można łatwo przeliczyć na pary `x` i `y`, które z kolei pozwalają wyświetlać wszystko bez większych problemów na ekranie komputera. Gdy rysowaliśmy wykres słupkowy, wartości danych odwzorowywaliśmy na wartości na ekranie — przeliczaliśmy liczby na wysokość prostokątów. W przypadku tworzenia map geograficznych też generujemy odwzorowania — pary długość i szerokość przechodzą w pary `x` i `y`. Jednocześnie operowanie zmiennymi `x` i `y` pozwala w łatwy sposób ominąć niewygodny problem kolejności długości i szerokości podanych odwrotnie.

Get Lat+Lon (*http://teczno.com/squares/#12/37.8043/-122.2712*) to doskonałe narzędzie autorstwa Michała Migurskiego, dzięki któremu z łatwością sprawdzisz, czy podane współrzędne geograficzne są na pewno poprawne. Trzymaj tę stronę otwartą w oknie przeglądarki przez cały czas, gdy pracujesz nad mapami, zagląda się do niej bowiem bardzo często.

Ścieżki

Mamy już dane geograficzne. Pora coś z nimi zrobić.

Przede wszystkim należy zdefiniować *generator ścieżek*:

```
var path = d3.geo.path();
```

Mogę z czystym sumieniem powiedzieć, że `d3.geo.path()` to funkcja ratująca życie. To ona przekłada bałagan z pliku GeoJSON na jeszcze bardziej nieporządny zapis ze znaczników `path`. Oddajmy cześć `d3.geo.path()`!

Moglibyśmy oczywiście wkleić kod GeoJSON bezpośrednio do pliku HTML, ale fuj, tyle współrzędnych i nawiasów klamrowych? Co za bałagan! Znacznie porządniej będzie, gdy zastosujemy bardziej tradycyjne podejście, czyli pozostawimy dane geograficzne w osobnym pliku i wczytamy jego zawartość za pomocą funkcji `d3.json()`:

```
d3.json("us-states.json", function(json) {
   svg.selectAll("path")
      .data(json.features)
      .enter()
      .append("path")
      .attr("d", path);
});
```

Funkcja `d3.json()` przyjmuje dwa argumenty. Pierwszy to zmienna łańcuchowa określająca ścieżkę dostępu do pliku, którego zawartość należy wczytać, drugi zaś to funkcja zwrotna uruchamiana dopiero po wczytaniu i sparsowaniu zawartości pliku JSON. (Więcej na temat funkcji zwrotnych znajdziesz w ramce „Obsługa błędów wczytywania danych" z rozdziału 5.). Funkcja `d3.json()`, podobnie jak `d3.csv()`, jest *asynchroniczna*, a zatem nie wstrzymuje wykonywania pozostałej części kodu na czas wczytywania danych z pliku. Oznacza to, że kod umieszczony *po* wywołaniu funkcji zwrotnej będzie w rzeczywistości wykonany *przed* jej ciałem:

```
d3.json("someFile.json", function(json) {
   // Tu umieść kod zależny od danych z pliku JSON.
});

// Tu umieść tylko te instrukcje, które nie potrzebują do pracy zawartości pliku JSON.
console.log("Lubię koty.");
```

Dlatego zapamiętaj: podczas korzystania z zewnętrznych źródeł danych kod, którego wykonanie zależy od zawartości tych źródeł, musi się zawsze znaleźć wewnątrz funkcji zwrotnej. (Możesz też rozbić go na mniejsze porcje, które zapiszesz we własnych funkcjach i dopiero je wywołasz w ciele funkcji zwrotnej).

Wróćmy jednak do przykładu. Pora powiązać właściwości pobrane z pliku GeoJSON z nowymi znacznikami `path`. Podany kod tworzy osobną ścieżkę dla każdej z cech:

```
svg.selectAll("path")
   .data(json.features)
   .enter()
   .append("path")
   .attr("d", path);
```

Zauważ, że ostatni wiersz — ten, w którym zmienna `d` (atrybut przechowujący dane znacznika `path`) jest przekazywana do generatora ścieżek — po prostu pobiera wszystkie dane z pliku i przelicza je na kod SVG. Wynik jest widoczny na rysunku 12.1.

Rysunek 12.1. Pierwsze zetknięcie z danymi GeoJSON

Mapa! To zbyt proste, by było możliwe! Aby sprawdzić, czy uzyskasz ten sam efekt, otwórz plik *01_paths.html*. Dalsze działania będą miały na celu wyłącznie dostosowanie mapy do naszych potrzeb.

Jeśli interesują Cię zagadnienia związane z pracą ze ścieżkami i ich generatorami, polecam Ci zapoznanie się z dokumentacją (*https://github.com/mbostock/d3/wiki/Geo-Paths*).

Odwzorowania

Niewątpliwie rzuciło Ci się w oczy, że mapa nie przedstawia całych Stanów Zjednoczonych. Aby naprawić to niedociągnięcie, musimy zmienić używane *odwzorowanie*.

Czym jest *odwzorowanie*? Z pewnością nie uszło Twojej uwadze także to, że Ziemia jest kulista, a nie płaska. Wszystkie kuliste obiekty są trójwymiarowe, co znacznie utrudnia przedstawianie ich na powierzchniach dwuwymiarowych. Odwzorowanie jest algorytmem kompromisu; to metoda pozwalająca rzutować trójwymiarową przestrzeń na dwuwymiarową płaszczyznę.

Biblioteka D3 pozwala definiować mechanizmy rzutowania za pomocą wyglądającej znajomo konstrukcji:

```
var projection = d3.geo.albersUsa()
   .translate([w/2, h/2]);
```

Biblioteka pozwala korzystać z kilku dostępnych w niej algorytmów rzutowania. Odwzorowanie Albersa jest rzutem złożonym, przygotowanym tak, by na mapie zmieściły się także Alaska i Hawaje, które umieszcza się pod południowo-zachodnim wybrzeżem Stanów Zjednoczonych (zaraz zobaczysz, jak to wygląda). To właśnie `albersUsa` jest domyślnym odwzorowaniem używanym przez funkcję `d3.path.geo()`. Teraz, gdy określiliśmy je już jawnie, możemy dodatkowo podać kilka opcji, z jakimi miałoby być wykorzystywane, jak choćby wartość przesunięcia. Z przytoczonej instrukcji wynika, że przesuwamy odwzorowanie w kierunku środka obrazu (o połowę szerokości i o połowę wysokości).

Drugą zmianą, jaką zrobimy, będzie jawne zażądanie od generatora ścieżek, by wszystkie ścieżki wyznaczał w odniesieniu do wskazanego wcześniej własnego odwzorowania:

```
var path = d3.geo.path()
   .projection(projection);
```

W ten sposób otrzymaliśmy mapę przedstawioną na rysunku 12.2. Powoli zbliżamy się do celu. Kod znajdziesz w pliku *02_projection.html*.

Rysunek 12.2. Te same dane GeoJSON, ale przy wyśrodkowaniu rzutu

Możemy też dodać do odwzorowania metodę `scale()`, żeby zmniejszyć nieco mapę i osiągnąć tym samym wynik przedstawiony na rysunku 12.3.

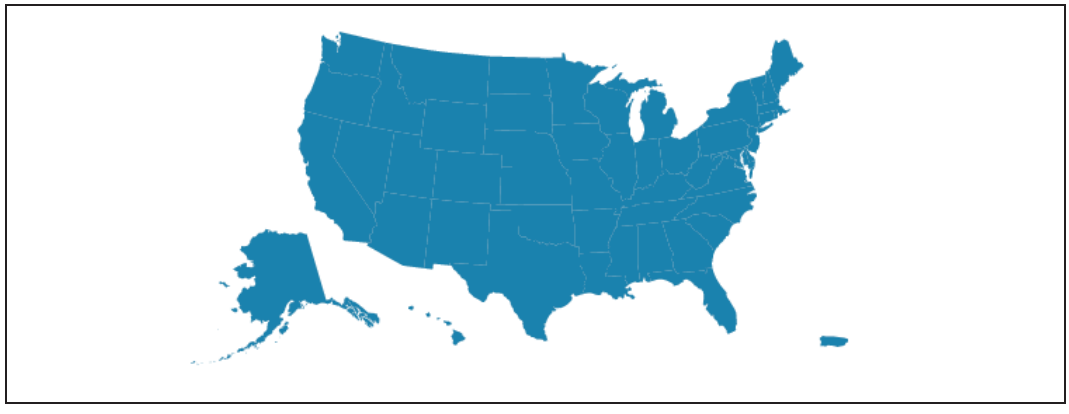
```
var projection = d3.geo.albersUsa()
   .translate([w/2, h/2])
   .scale([500]);
```

Rysunek 12.3. Mapa Stanów Zjednoczonych — przeskalowana i wyśrodkowana tak, by mieściła się w obszarze obrazu

Domyślną wartością skalowania jest 1000. Mniejsze liczby zmniejszają mapę, a większe powiększają.

Doskonale! Jeśli otworzysz w przeglądarce plik *03_scaled.html*, przekonasz się, że wszystko działa.

Wystarczy dodać jedną instrukcję `style()`, żeby zmienić kolor wypełnienia ścieżek na jakiś mniej przygnębiający. Sądzę, że niebieski widoczny na rysunku 12.4 nie jest brzydki.



Rysunek 12.4. Niebieski jest zdecydowanie lepszy

Cały kod znajdziesz w pliku *04_fill.html*. W ten sam sposób można zdefiniować kolor i szerokość pędzla obramowania.

Funkcje rzutujące, które pozwalają uzyskiwać różnego rodzaju odwzorowania kartograficzne, to niezwykle potężne algorytmy. Warto wiedzieć, że nie ma odwzorowań uniwersalnych — zmienia się je w zależności od tego, jakich informacji potrzebujemy i o którą część świata chodzi (inaczej odwzorowuje się bieguny, a inaczej okolice równika).

Zasługą głównie Jasona Daviesa (*http://www.jasondavies.com/*) jest to, że dodatki do biblioteki D3 odpowiedzialne za przygotowanie odwzorowań kartograficznych oferują dziś użytkownikom niemal wszystkie znane nam rodzaje rzutów. Szczegółowy opis pracy z rzutami znajdziesz w dokumentacji (*https://github.com/mbostock/d3/wiki/Geo-Projections*). Warto też zapoznać się z porównaniem przedstawiającym właściwości poszczególnych rzutów (*http://bl.ocks.org/mbostock/3711652*).

Kartogram

Karto *co?* To słowo, którego nie słyszy się na co dzień, określa mapę, na której całe obszary zostały wypełnione różnymi odcieniami (jasnymi lub ciemnymi) czy też różnymi kolorami tak, by oddać rozkład powiązanych z oznaczeniami wartości. W Stanach Zjednoczonych, szczególnie w okresie wyborów, często spotyka się kartogramy określane mianem „czerwonych i niebieskich stanów". Na mapach tych zaznacza się odpowiednim kolorem przewagę poparcia dla kandydatów Partii Demokratycznej bądź Partii Republikańskiej. Ale kartogramy wykonuje się z różnych przyczyn, niekoniecznie politycznych.

Takie odwzorowania cieszą się chyba największą popularnością wśród użytkowników D3. Pamiętaj jednak, że choć kartogramy potrafią być niezwykle pomocne, to mają jednak poważne ograniczenia. Ponieważ na tego rodzaju mapach wartości odwzorowuje się na pewnej *powierzchni*, łatwo może dojść do niezamierzonego przekłamania: duże powierzchnie o niskiej

gęstości danej wielkości (na przykład stan Nevada) mogą sprawiać wrażenie bardziej istotnych niż są w rzeczywistości. Typowy kartogram nie oddaje dobrze wartości liczonych „na głowę", bo na przykład Nevada jest zbyt duża, a Delaware zbyt mały. Jednocześnie kartogram zachowuje geografię miejsca oraz — jak na mapę — wygląda naprawdę imponująco. Zobaczmy zatem, co da się z tym zrobić. (Kod znajdziesz w pliku *05_choropleth.html*).

Po pierwsze zdefiniuję funkcję skalującą, która pobierze wartości liczbowe, a zwróci kolory. Na tym właśnie polega w dużej mierze praca nad kartogramem.

```
var color = d3.scale.quantize()
   .range(["rgb(237,248,233)", "rgb(186,228,179)","rgb(116,196,118)",
"rgb(49,163,84)","rgb(0,109,44)"]);
```

Funkcja `quantize` jest zaliczana do grupy skal liniowych, ale wynikiem jej działania jest zbiór wartości dyskretnych. Wartościami wyjściowymi mogą być liczby, kolory (jak w tym przypadku) czy cokolwiek innego, co uznasz za odpowiednie. Funkcja ta przydaje się bardzo, jeśli chcesz podzielić zbiór wartości na grupy. W omawianym przypadku zdecydowałem się utworzyć pięć grup docelowych, ale ich liczba nie jest w żaden sposób formalnie ograniczona.

Zauważ, że określiłem tu zbiór wartości wyjściowych, natomiast nie napisałem nic na temat dziedziny. (Czekam, aż wczytają się dane). Kolory, którymi się posłużyłem, pochodzą z pliku *colorbrewer.js* dostępnego pod adresem *https://github.com/mbostock/d3/tree/master/lib/colorbrewer*. Repozytorium to zawiera sporych rozmiarów zbiór barw dobranych przez Cynthię Brewer na podstawie prowadzonych przez nią badań.

Pora wczytać jakieś dane. Zawczasu przygotowałem plik *us-ag-productivity-2004.csv*, w którym znajdziesz dane ułożone w następujący sposób:

```
state,value
Alabama,1.1791
Arkansas,1.3705
Arizona,1.3847
California,1.7979
Colorado,1.0325
Connecticut,1.3209
Delaware,1.4345
...
```

Dane te, udostępnione przez Departament Rolnictwa Stanów Zjednoczonych, zawierają informacje na temat produkcji rolnej w poszczególnych stanach w 2004 roku. Dane liczbowe przedstawiają wyniki względne wyznaczane dla linii bazowej, jaką była produkcja rolna w Alabamie w 1996 roku (1,0). Większe od jedynki wartości należy zatem rozumieć jako wyższy poziom produkcji płodów rolnych, a mniejsze jako niższy. (Na stronie *http://data.gov* można znaleźć wiele oficjalnych zestawień statystycznych). Wydaje się, że te wartości powinny ładnie się przełożyć na kartogram produkcji rolnej.

Dane z pliku wczytamy za pomocą funkcji `d3.csv()`:

```
d3.csv("us-ag-productivity-2004.csv", function(data) { ...
```

Teraz (zanim zapomnę!), w funkcji zwrotnej, zdefiniuję parametr `color` dziedziny wartości wejściowych skali kwantującej:

```
color.domain([
   d3.min(data, function(d) { return d.value; }),
   d3.max(data, function(d) { return d.value; })
]);
```

W kodzie tym pojawiają się funkcje `d3.min()` i `d3.max()` pozwalające wyznaczyć i zwrócić najmniejszą i największą wartość, dzięki czemu dziedzina będzie wyznaczana dynamicznie.

Następnie wczytamy dane JSON — tę część operacji przeprowadzimy tak jak poprzednio. Główna różnica i nowość polega na tym, że tym razem chcę *włączyć* dane dotyczące rolnictwa *do* danych GeoJSON. Dlaczego? Ponieważ do znacznika można dołączać tylko jeden zestaw danych naraz. Zdecydowanie potrzebujemy danych GeoJSON, bo na ich podstawie tworzone są ścieżki, ale jednocześnie potrzebujemy też danych dotyczących upraw. Gdyby zatem udało się połączyć je w jedną monstrualną tablicę, wtedy moglibyśmy podpiąć całość pod znaczniki `path`. (Istnieje kilka sposobów rozwiązania tego problemu — jak zawsze prezentuję swoją ulubioną metodę).

```
d3.json("us-states.json", function(json) {
    // Łączy dane rolnicze z danymi GeoJSON.
    // Sprawdza w pętli raz cały zestaw danych rolniczych.
    for (var i = 0; i < data.length; i++) {
        // Pobiera nazwę stanu.
        var dataState = data[i].state;
        // Pobiera wartość danych i przekształca ją na liczbę zmiennoprzecinkową.
        var dataValue = parseFloat(data[i].value);
        // Odnajduje odpowiedni stan w danych GeoJSON.
        for (var j = 0; j < json.features.length; j++) {
            var jsonState = json.features[j].properties.name;
            if (dataState == jsonState) {
            // Kopiuje dane do zmiennej JSON.
                json.features[j].properties.value = dataValue;
              // Przestaje przeszukiwać JSON.
                break;
            }
        }
    }
}
```

Przyjrzyj się uważnie przytoczonemu fragmentowi kodu. Zasadniczo zaproponowane rozwiązanie polega na tym, by dla każdego stanu odnaleźć dane GeoJSON opatrzone tą samą nazwą (na przykład Colorado). Gdy to się uda, pobierzemy wartości danych zapisanych dla każdego stanu i dodamy je do `json.features[j].properties.value`. W ten sposób będą one połączone ze znacznikiem i dostępne później, gdy będziemy ich potrzebować.

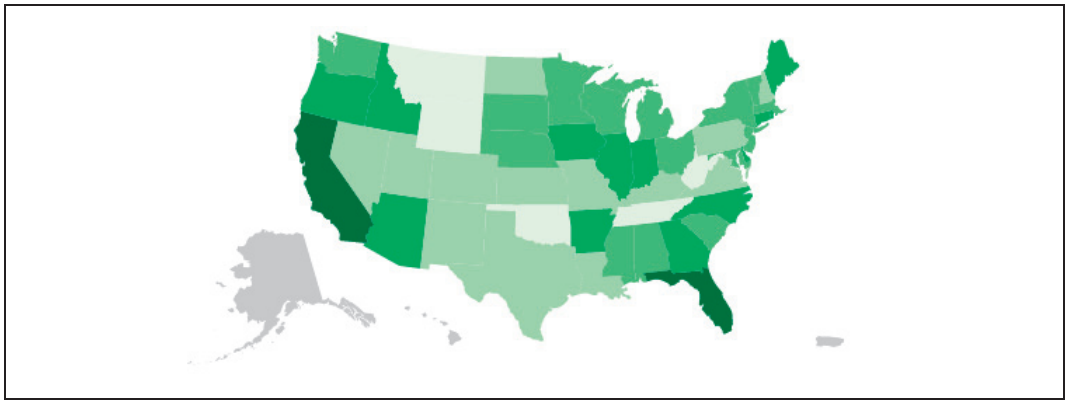
Wreszcie w ten sam sposób co ostatnio przygotujemy ścieżki, przy czym tym razem wartość funkcji `style()` będziemy przypisywać dynamicznie:

```
svg.selectAll("path")
    .data(json.features)
    .enter()
    .append("path")
    .attr("d", path)
    .style("fill", function(d) {
        // Pobiera wartości danych.
        var value = d.properties.value;
        if (value) {
            // Jeśli wartość istnieje...
            return color(value);
        } else {
            // Jeśli wartość jest niezdefiniowana...
            return "#ccc";
        }
});
```

Teraz każda ze ścieżek zostanie wypełniona innym kolorem, a nie jak miało to miejsce poprzednio, gdy wszystkie wypełniliśmy barwą `"steelblue"`. Problem polega na tym, że nie dysponujemy danymi ze *wszystkich* stanów. Zestawienie, którym się posłużyłem, nie obejmowało Alaski, Dystryktu Kolumbii, Hawajów i Portoryko (które wprawdzie nie jest stanem, ale jest uwzględnione w danych GeoJSON i pojawia się w odwzorowaniu).

Aby poradzić sobie jakoś z tymi wyjątkami, dodałem do kodu warunek logiczny — instrukcję `if()`, która sprawdza, czy dla danego stanu wartość danych została zdefiniowana. Jeśli dane istnieją, zwracana jest funkcja `color(value)`, co oznacza, że mogę przekazać dane do skali kwantującej i otrzymać w wyniku tego kod koloru. Dla wartości niezdefiniowanych ustaliłem domyślną wartość koloru `#ccc`, co odpowiada delikatnej szarości.

Pięknie! Spójrz tylko na wyniki widoczne na rysunku 12.5. Sprawdź, jak prezentuje się kod, i spróbuj z nim swoich sił. Znajdziesz go w pliku *05_choropleth.html*.



Rysunek 12.5. Kartogram przedstawiający poziom produkcji rolnej w poszczególnych stanach

Dodawanie punktów

Miło byłoby zaznaczyć na mapie choć kilka miast, by nadać grafice dokładniejszy kontekst, prawda? Warto byłoby też znaczyć, ile jest dużych obszarów gęsto zaludnionych w stanach o najwyższym (i najniższym) poziomie produktywności rolnej. Zaczniemy znów od znalezienia odpowiednich danych.

Na szczęście niedawno mieliśmy znów spis powszechny. (Na to idą nasze podatki!). Oto początek nieobrobionego pliku CSV zawierającego roczne szacunkowe zestawienie liczby ludności miast powyżej 50 000 mieszkańców (plik dostępny pod adresem *http://www.census.gov/popest/data/cities/totals/2011/tables/SUB-EST2011-01.csv*):

```
table with row headers in column A and column headers in rows 3 through 4,,,,,,,
,,,
"Table 1. Annual Estimates of the Resident Population for Incorporated Places
Over 50,000, Ranked by July 1, 2011 Population: April 1, 2010 to July 1, 2011"
,,,,,,,,,,
Rank,Geographic Area,,"April 1, 2010",,Population Estimate (as of July 1),,
,,,,Place,State,Census,Estimates Base,2010,2011,,,,
1,New York city,New York,"8,175,133","8,175,133","8,186,443","8,244,910",,,,
2,Los Angeles city,California,"3,792,621","3,792,625","3,795,761","3,819,702"
,,,,
3,Chicago city,Illinois,"2,695,598","2,695,598","2,698,283","2,707,120",,,,
```

```
4,Houston city,Texas,"2,099,451","2,099,430","2,108,278","2,145,146",,,,
5,Philadelphia city,Pennsylvania,"1,526,006","1,526,006","1,528,074","1,536,471"
,,,,
6,Phoenix city,Arizona,"1,445,632","1,445,656","1,448,531","1,469,471",,,,
7,San Antonio city,Texas,"1,327,407","1,327,606","1,334,431","1,359,758",,,,
8,San Diego city,California,"1,307,402","1,307,406","1,311,516","1,326,179",,,,
9,Dallas city,Texas,"1,197,816","1,197,816","1,201,715","1,223,229",,,,
10,San Jose city,California,"945,942","952,612","955,091","967,487",,,,
...
```

Niezły w tym bałagan i nie wszystkie dane są mi potrzebne. Otwieram więc plik w ukochanym arkuszu kalkulacyjnym, porządkuję nieco dane i usuwam zbędne kolumny. (Ty możesz zrobić to samo w programach LibreOffice Calc, Apple Numbers bądź Microsoft Excel). Jednocześnie stwierdzam, że interesuje mnie tylko pięćdziesiąt największych miast, więc spokojnie mogę pominąć niektóre dane. Po zapisaniu ostatecznej wersji w formacie CSV na dysku pozostało mi to:

```
rank,place,population
1,New York city,8175133
2,Los Angeles city,3792621
3,Chicago city,2695598
4,Houston city,2099451
5,Philadelphia city,1526006
6,Phoenix city,1445632
7,San Antonio city,1327407
8,San Diego city,1307402
9,Dallas city,1197816
10,San Jose city,945942
...
```

To przydatne informacje, ale żeby móc umieścić je na mapie, będę potrzebował ich współrzędnych geograficznych — szerokości i długości. Ręczne sprawdzenie tego zajęłoby *całe wieki*. Na szczęście możemy wykorzystać usługę *geokodowania*. Pozwala ona sprawdzić współrzędne geograficzne miejsc na podstawie ich nazw. Nazwy te są sprawdzane na mapie (a w zasadzie w bazie danych), a następnie usługa zwraca dokładne dane dotyczące szerokości i długości geograficznej. No, może przesadziłem nieco z tą dokładnością. Geokoder stara się jak może, ale czasami musi przyjmować pewne założenia, szczególnie jeśli dostanie nieprecyzyjne dane wejściowe. Gdy na przykład podasz mu nazwę „Paryż", koder założy, że chodzi o Paryż we Francji, a nie Paryż w Teksasie. Dlatego zawsze warto sprawdzić dane wyjściowe na mapie i ręcznie poprawić ewentualne pomyłki (mając na podorędziu stronę *http://www.teczno.com/squares/*).

Ruszam zatem do swojego ulubionego geokodera wsadowego (*http://www.gpsvisualizer.com/geocoder/*), wklejam nazwy miejscowości i klikam *Start*. Po kilku minutach otrzymuję kolejną listę — wartości oddzielonych przecinkami — na której wyraźnie rzucają się w oczy pary liczb oznaczających szerokość i długość geograficzną. Przenoszę wyniki do arkusza kalkulacyjnego i zapisuję nowy, ujednolicony plik CSV ze współrzędnymi interesujących mnie miejsc:

```
rank,place,population,lat,lon
1,New York city,8175133,40.71455,-74.007124
2,Los Angeles city,3792621,34.05349,-118.245323
3,Chicago city,2695598,45.37399,-92.888759
4,Houston city,2099451,41.337462,-75.733627
5,Philadelphia city,1526006,37.15477,-94.486114
6,Phoenix city,1445632,32.46764,-85.000823
7,San Antonio city,1327407,37.706576,-122.440612
8,San Diego city,1307402,37.707815,-122.466624
9,Dallas city,1197816,40.636,-91.168309
10,San Jose city,945942,41.209716,-112.003047
...
```

To było banalnie proste. Dziesięć lat temu przeprowadzenie takiej operacji wymagałoby godzin badań i żmudnego zbierania danych, a dziś w ciągu ułamka sekundy, kopiując i wklejając bezmyślnie dane, mamy całą pracę za sobą. Rozumiesz już chyba, dlaczego doświadczamy prawdziwego wysypu przeróżnych map internetowych.

Dane są gotowe, a my wiemy, w jaki sposób wprowadzić je do aplikacji:

```
d3.csv("us-cities.csv", function(data) {
   // Robi coś...
});
```

Możemy teraz dodać do funkcji zwrotnej kod odpowiedzialny za przygotowanie nowych znaczników `circle`, którymi oznaczymy każde miasto. Potem wystarczy *ustawić każde koło* na mapie zgodnie ze sprawdzonymi wcześniej współrzędnymi:

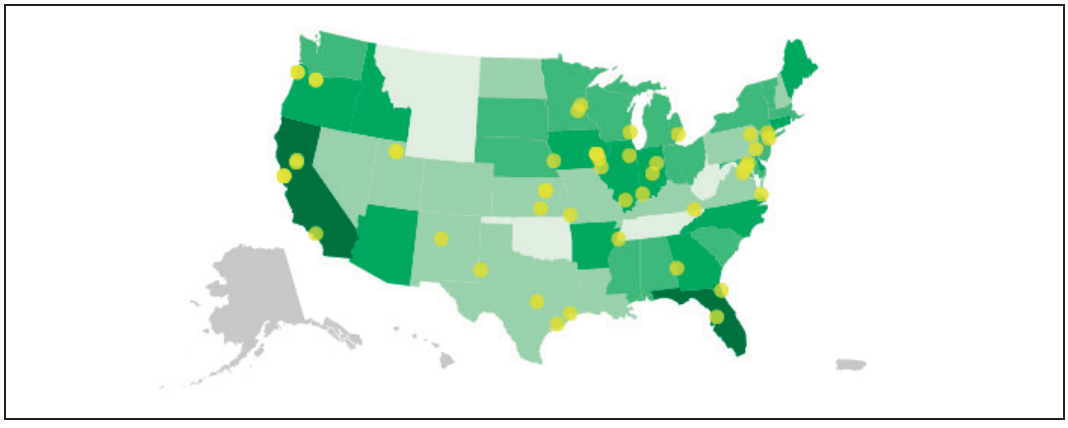
```
svg.selectAll("circle")
   .data(data)
   .enter()
   .append("circle")
   .attr("cx", function(d) {
      return projection([d.lon, d.lat])[0];
   })
   .attr("cy", function(d) {
      return projection([d.lon, d.lat])[1];
   })
   .attr("r", 5)
   .style("fill", "yellow")
   .style("opacity", 0.75);
```

W podanym kodzie najistotniejsze są wyrażenia `attr()`, w których określamy wartości zmiennych `cx` i `cy`. Jak widzisz, dostanie się do wartości szerokości i długości geograficznych, które uzyskałem przed chwilą, nie sprawia żadnych problemów — wystarczy odwołać się do odpowiedniego pola obiektu `d.lat` lub `d.lon`. Prawdziwa trudność polega na tym, że potrzebujemy nie tyle współrzędnych geograficznych, co *współrzędnych na ekranie*, czyli wartości `x` i `y`.

Dlatego też znów odwołamy się do pomocy przewspaniałej metody `projection()`, która w zasadzie jest odmianą dwuwymiarowej funkcji skalującej. Funkcje skalujące przyjmowały jako argument jedną liczbę i zwracały inną w charakterze wyniku. Funkcje rzutujące pobierają dwie liczby i zwracają dwie. (Oczywiście między tymi funkcjami istnieje zasadnicza różnica, bo obliczenia prowadzone wewnątrz funkcji rzutowania są znacznie bardziej złożone niż te, których dokonuje funkcja skalujące).

Argumentem funkcji rzutowania jest dwuwartościowa tablica, w której jako pierwsza pojawia się *długość geograficzna* (pamiętaj: długość i szerokość, a nie szerokość i długość, jesteśmy bowiem w GeoJSON-landii), a wynikiem jej działania — dwuwartościowa tablica ze współrzędnymi ekranu `x` i `y`. Dlatego atrybutowi `cx` przypiszemy `[0]`, bo chcemy, żeby przyjął wartość *pierwszej* z tych liczb, czyli `x`. Z kolei dla atrybutu `cy` przypiszemy `[1]`, gdyż chcemy przekazać do niego *drugą* wartość, czyli `y`. Czy to ma sens?

Mapa, którą otrzymaliśmy w efekcie tych działań (rysunek 12.6), jest prześliczna! Sprawdź też kod w pliku *06_points.html*.



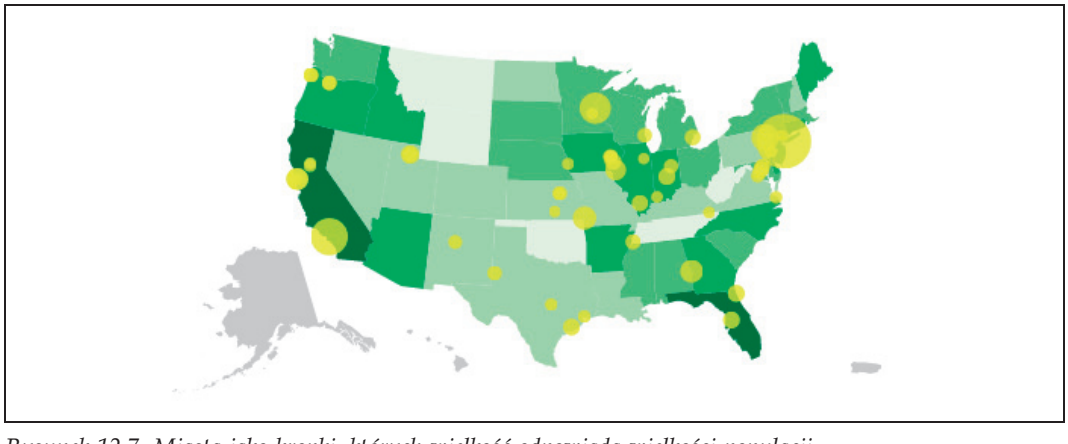
Rysunek 12.6. Pięćdziesiąt największych miast USA przedstawionych w postaci ślicznych żółtych kropeczek

No dobrze, ale wszystkie kropki mają taki sam rozmiar. Spróbujmy połączyć teraz liczebność mieszkańców z rozmiarem reprezentującej miasto kropki. Zatem zamiast statycznego pola powierzchni przekażemy do funkcji wartość `population`:

```
.attr("r", function(d) {
   return Math.sqrt(parseInt(d.population) * 0.00004);
})
```

W tym miejscu przechwytujemy wartość parametru `d.population`, umieszczamy ją w funkcji `parseInt()`, by przekształcić zmienną łańcuchową w liczbę całkowitą, skalujemy w dół, mnożąc przez ustaloną z góry liczbę, i wreszcie wyciągamy z wyniku pierwiastek (żeby z jednostek pola powierzchni przejść do jednostek promienia). Kod aplikacji znajdziesz w pliku *07_points_sized.html*.

Jak widać na rysunku 12.7, teraz największe miasta naprawdę rzucają się w oczy. Nie sposób pominąć różnic w rozmiarze. Tego rodzaju zależności byłyby zapewne lepiej widoczne na skali logarytmicznej, szczególnie jeśli uwzględnialibyśmy także małe miasta. Poza tym zamiast mnożyć wynik przez 0,00004, można by użyć funkcji skalującej. (Ten problem pozostawię Tobie).



Rysunek 12.7. Miasta jako kropki, których wielkość odpowiada wielkości populacji

Chciałbym w tym momencie podkreślić, że udało się nam wczytać i przedstawić graficznie na mapie dwa rodzaje danych. (A w zasadzie trzy, jeśli uwzględnić współrzędne miast z geokodera, które włączyliśmy do zestawu!).

Pobieranie danych geograficznych i ich parsowanie

Gdyby zależało nam wyłącznie na przygotowywaniu map Stanów Zjednoczonych, mielibyśmy już gotowe wszystkie potrzebne dane GeoJSON. Ale przecież świat jest znacznie większy i pełen miejsc, które warto nanieść na mapy. Pozwól mi zatem na niewielką dygresję, chciałbym bowiem odnieść się do tematu pozyskiwania danych geograficznych dowolnie wybranej okolicy i parsowania ich na potrzeby dalszego użycia. Naszym celem będzie uzyskanie pliku GeoJSON, takiego jak *us-states.json*, który byłby zrozumiały dla biblioteki D3.

Znajdź pliki kształtów

Tak zwane *pliki kształtów* (ang. *shapefiles*) pochodzą z czasów znacznie poprzedzających boom na tworzenie map internetowych i interaktywnych wizualizacji. Zawierały zasadniczo te same informacje, które dziś przechowuje się w plikach GeoJSON (granice krain geograficznych oraz leżące w ich obrębie punkty), ale zawarty w nich był nie tylko tekst, więc odczytywanie ich było mocno utrudnione. Format plików kształtów wyrósł w kręgu geografów, kartografów i naukowców korzystających z oprogramowania GIS (*Geographic Information Systems*). Jeśli masz dostęp do drogiego oprogramowania GIS, to pliki kształtów zostaną pewnie Twoimi najlepszymi przyjaciółmi. Wątpię jednak, żeby wielu moich Czytelników zaliczało się do tego wąskiego grona, z kolei przeglądarki internetowe i tak nie mogą przetworzyć zawartości plików kształtów.

Jeśli nie uda Ci się znaleźć pliku GeoJSON, który opisywałby interesujący Cię teren, wtedy faktycznie nie pozostanie Ci nic innego jak znaleźć odpowiedni plik kształtów. Dobrym punktem wyjścia dla dalszych poszukiwań są strony rządowe, szczególnie jeśli interesują Cię konkretne rejony. Ja lubię najbardziej dwie strony:

Natural Earth (*http://www.naturalearthdata.com/*)
: Witryna ta to dostępna w domenie publicznej szeroka gama danych geograficznych przygotowanych z myślą o przedstawianiu kulturowego i politycznego dorobku ludzkości. Odwzorowywanie granic państwowych ma charakter wybitnie polityczny i Natural Earth często publikuje wyjaśnienia swoich decyzji.

Strona spisu powszechnego w Stanach Zjednoczonych (*http://www.census.gov/#*)
: Znajdziesz tu opracowania graficzne wszystkich stanów i hrabstw, sieci dróg, szlaków wodnych — co tylko chcesz.

Wybierz rozdzielczość

Zanim zaczniesz cokolwiek pobierać, sprawdź *rozdzielczość* danych. Wszystkie pliki kształtów są zapisywane w postaci wektorowej (a nie bitmapy), więc pisząc o rozdzielczości, nie mam na myśli pikseli, lecz poziom szczegółowości kartograficznej, czyli tak zwaną ziarnistość.

Zestawy danych przygotowywane przez Natural Earth występują w jednej z trzech skal:

- 1:10 000 000,
- 1:50 000 000,
- 1:110 000 000.

Oznacza to, że w przypadku danych o największej rozdzielczości jednej jednostce wyznaczonej na mapie odpowiada dziesięć milionów takich jednostek w prawdziwym świecie. Można też podejść do tego zagadnienia odwrotnie i powiedzieć, że dziesięć milionów jednostek uproszczono do jednej. Znaczyłoby to, że 10 000 000 centymetrów po przełożeniu do skali mapy odpowiada 1 centymetrowi.

Współczynniki skali można zapisać w prostszej postaci:

- 1:10m,
- 1:50m,
- 1:110m.

W przypadku mniej szczegółowych (oddalonych) map świata w zupełności wystarczy rozdzielczość rzędu 1:110m, ale już żeby pokazać wyraźnie kształt granicy konkretnego stanu, potrzeba mapy o skali co najmniej 1:10m. Jeśli zaś przygotowujesz mapę naprawdę niewielkiego obszaru (mocno przybliżonego) — miasta czy dzielnicy — nawet taka skala byłaby niewystarczająca. (W takim przypadku sugeruję poszukać informacji na stronach WWW stanów, województw czy miast).

Różne źródła oferują mapy o różnych skalach. Wiele z oferowanych na stronie amerykańskiego spisu powszechnego plików kształtu jest wykonanych w jednej z następujących skal:

- 1:500 000 (1:500k),
- 1:5 000 000 (1:5m),
- 1:20 000 000 (1:20m).

Wybierz skalę i pobierz plik. Najczęściej będzie miał on postać archiwum ZIP, w którym zapisano kilka innych plików. Dam Ci przykład. Pobiorę teraz plik przedstawiający ocean odwzorowany w skali 1:110m (mało szczegółowej). Jest on dostępny na stronie Natural Earth pod adresem *http://www.naturalearthdata.com/downloads/110m-physical-vectors/110m-ocean/*.

W archiwum znajdują się następujące pliki:

ne_110m_ocean.dbf
ne_110m_ocean.prj
ne_110m_ocean.README.html
ne_110m_ocean.shp
ne_110m_ocean.shx
ne_110m_ocean.VERSION.txt

Cóż, to się nazywa kolekcja dziwacznych rozszerzeń. Nas interesują pliki o rozszerzeniu *.shp*, ale nie pozbywajmy się jeszcze pozostałych.

Upraszczanie kształtów

W idealnym świecie bez trudu znaleźlibyśmy pliki kształtów o takiej rozdzielczości, jaka byłaby nam potrzebna. Ale co zrobić, jeśli jedynym dostępnym będzie plik o superwysokiej rozdzielczości, na przykład, 1:100k? Jego rozmiary byłyby olbrzymie. A skoro zajmujesz się już programowaniem w JavaScripcie, wydajność kodu powinna mieć dla Ciebie pierwszorzędne znaczenie. Dlatego wysyłanie miliarda bajtów danych geograficznych do przeglądarki zdecydowanie odpada.

Na szczęście plik kształtu można *uprościć*, uzyskując jego mniej szczegółową wersję. Proces ten pięknie ilustruje interaktywna aplikacja Mike'a Bostocka dostępna pod adresem *http://bost.ocks.org/mike/simplify/*.

MapShaper (*http://mapshaper.org/*) autorstwa Matta Blocha jest doskonałym, łatwym w obsłudze interaktywnym narzędziem służącym właśnie do upraszczania map. Aplikacja pozwala wczytać własne pliki kształtów i wyregulować zwykłym suwakiem poziom ich skomplikowania. Jak zawsze w takich przypadkach chodzi o osiągnięcie dobrego kompromisu między odpowiednią szczegółowością a rozmiarem pliku.

Jeśli zdecydujesz się używać MapShapera, przed wyeksportowaniem mapy wybierz opcję *Shapefile-polygons* (plik kształtu — wielokąty). W ten sposób otrzymasz od razu pliki *.shp* i *.shx*. Pobierz obydwa na dysk, zmień ich nazwy na identyczne z oryginalnymi plikami o tych rozszerzeniach. Potem, przed przeprowadzeniem konwersji do formatu GeoJSON, wykonaj kopię oryginalnego pliku *.dbf* i umieść ją w tym samym katalogu co uproszczone pliki *.shp* i *.shx*. To ważny krok, dzięki któremu na pewno nie stracisz istotnych metadanych zapisanych w pliku *.dbf*, na przykład identyfikatora kraju czy identyfikatorów ścieżek.

Możesz też skorzystać z napisanego przez Michała Migurskiego w języku Python skryptu Bloch (*https://github.com/migurski/Bloch*), wykorzystującego algorytmy upraszczające Matta Blocha i będącego dodatkiem do biblioteki `d3.simplify` (użytego zresztą do przygotowania wspomnianego wcześniej demo Mike'a Bostocka). Programistom przyświeca jeden cel: by pewnego dnia móc przeprowadzić uproszczenie liniowe bezpośrednio z poziomu samego JavaScriptu i wyeksportować wyniki do uproszczonej wersji formatu JSON, żeby móc potem korzystać z nich w innych projektach. Wachlarz dostępnych narzędzi rozszerza się z każdą chwilą, wiec miej oczy otwarte! (Dokładnie wtedy, gdy opracowywałem ten akapit, Mike Bostock udostępnił społeczności wersję próbną (*http://bl.ocks.org/mbostock/4090870*) nowej aplikacji służącej do przeprowadzania geometrycznych uproszczeń — TopoJSON (*https://github.com/mbostock/topojson*). Stan rzeczy zmienia się dosłownie z dnia na dzień! Z kolei teraz, gdy czytasz te słowa, narzędzie wiersza poleceń TopoJSON jest prawdopodobnie naszą największą nadzieją. Obecnie potrafi ono wczytywać pliki kształtu, upraszczać je *i* przekształcać dane do formatu JSON. Oczywiście TopoJSON jest zaprojektowany tak, by współpracować z biblioteką D3, mimo że zapisuje wyniki obliczeń w nowym formacie — TopoJSON — podobnym do GeoJSON, lecz bardziej wydajnym).

Konwersja do GeoJSON

Jeśli nie masz jeszcze zainstalowanego odpowiedniego oprogramowania, prawdopodobnie zbliżasz się właśnie do najbardziej ryzykownego etapu całego procesu. Ostatecznie chcielibyśmy uzyskać dostęp do narzędzia wiersza poleceń `ogr2ogr` niezależnie od tego, czy używasz systemu Mac, Unix czy Windows. Problem polega na tym, że `ogr2ogr` nie zadziała bez asysty kilku innych szkieletów aplikacji, bibliotek i innych im podobnych.

Nie będę tu wnikał w niuanse instalacji, ale spróbuję ułatwić Ci nieco zadanie.

Przede wszystkim będziesz potrzebować biblioteki *Geospatial Data Abstraction Library*, czyli tak zwanej GDAL (*http://www.gdal.org/*). Narzędzie `ogr2ogr` jest jej częścią.

Poza tym musisz pobrać także GEOS, czyli *Geometry Engine, Open Source* (*trac.osgeo.org/geos/*).

Jeśli na Twoim komputerze działa system Windows lub Unix/Linux, możesz teraz przystąpić do najlepszej części, a mianowicie rozpocząć pobieranie plików źródłowych i instalowanie ich za pomocą zabawnie wyglądających poleceń typu **`build`**, **`make`** czy **`dlaczego to nie idzie omg prosze prosze zainstaluj sie teraz bo mam juz dosc`**.

Nie pamiętam dokładnie nazw poleceń, ale tak mniej więcej brzmiały. (A mówiąc poważnie: musisz mieć świadomość, że problemy na tym etapie nie powinny Cię dziwić. Kwestii pobierania i instalowania pakietów oprogramowania poświęcono w całości niejedną książkę i jeśli przejrzysz ofertę wydawnictwa Helion, z pewnością trafisz na kilka z nich).

Jeśli pracujesz na macu, na którym jakimś przypadkiem są zainstalowane Xcode i Homebrew *naraz*, wpisz po prostu w wierszu poleceń **`brew install gdal`**. I to już wszystko! (Jeśli nie masz żadnego z tych wspaniałych narzędzi, poszukaj ich. Obydwa są dostępne za darmo, ale zainstalowanie ich będzie wymagało od Ciebie nakładu czasu i energii. Xcode można pobrać ze strony AppStore (*https://itunes.apple.com/us/app/xcode/id497799835?mt=12*). Gdy Xcode znajdzie się już na dysku Twojego komputera, instalacja Homebrew powinna teoretycznie sprowadzić się do wpisania w oknie terminalu prostego polecenia *http://brew.sh/*. Z doświadczenia wiem jednak, że należy spodziewać się błędów podczas tego etapu pracy).

Do wszystkich użytkowników komputerów marki Mac, którzy nie korzystają z Xcode ani Homebrew: macie szczęście, że jakiś dobry człowiek przygotował przyjazny instalator GUI umieszczający w komputerze GDAL, GEOS oraz kilka innych narzędzi o nazwach, których naprawdę nie musisz znać. Poszukaj najnowszej wersji pakietu GDAL Complete (*http://www.kyngchaos.com/software/frameworks*), ale zanim cokolwiek zrobisz, zapoznaj się z zawartością pliku *GDAL Read Me*. Ukończenie instalowania nie oznacza, że możesz od razu wpisać w konsoli **`ogr2ogr`**. Najpierw należy dodać programy z rodziny GDAL do ścieżki prowadzącej do powłoki. Najprostszym na to sposobem jest otworzyć nowe okno terminala, wpisać w nim **`nano. bash_profile`**, wkleić do okna poleceń export `PATH=/Library/Frame,works/GDAL.framework/Programs:$PATH`, a następnie nacisnąć *Control+X* i *Control+y*, by zachować zmiany. Zakończ potem sesję, wpisując w oknie **`exit`**, i otwórz nowe okno konsoli. W nim możesz wpisać **`ogr2ogr`** i mieć nadzieję, że zadziała.

Niezależnie jednak od rodzaju systemu operacyjnego, z jakiego korzystasz, gdy skończysz już instalować niezbędne oprogramowanie, otwórz okno konsoli i udaj się do katalogu, w którym trzymasz pliki kształtów (na przykład **`cd ~/ocean_shapes/`**). Następnie przepisz polecenie:

```
ogr2ogr -f "GeoJSON" output.json filename.shp
```

Nakazuje ono skryptowi `ogr2ogr` pobrać plik *filename*, który powinien mieć rozszerzenie typu *.shp*, przekonwertować jego zawartość do formatu GeoJSON i zapisać wynik w pliku *output.json*.

W przypadku przykładowego pliku z mapą oceanów uruchomienie narzędzia `ogr2ogr` powinno wyglądać następująco:

```
ogr2ogr -f "GeoJSON" output.json ne_110m_ocean.shp
```

Wpisz to w konsoli i trzymaj kciuki, żeby nic się nigdzie nie wyświetliło.

Jakież to pozbawione klimatu! Wiem, wiem… Po godzinach spędzonych nad łamaniem kodu niezbędnego do zainstalowania starego, dobrego `ogr` człowiek chciałby jakiegoś spektakularnego finału, uczucia, które Ci towarzyszyło, gdy po przejściu całego *Super Mario 3* znów ratowało się księżniczkę. (Prawdę mówiąc, nigdy mi się to nie udało, ale podejrzewam, że wrażenia muszą być niezapomniane).

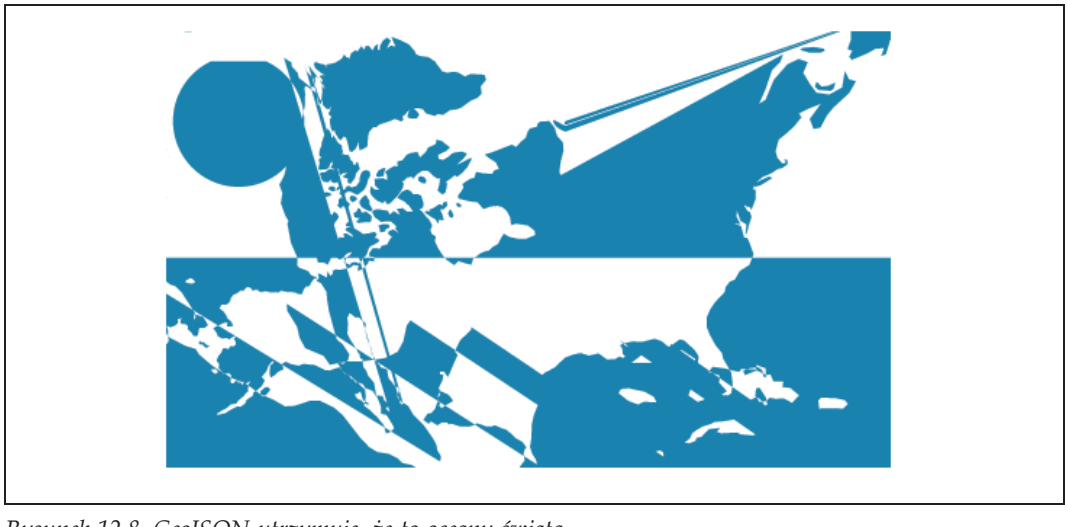
Nic z tego. Tym razem trzeba mieć nadzieję, że nie stało się nic. Za cały spektakularny efekt musi wystarczyć pojawienie się w tym samym katalogu nowego pliku o nazwie *output.json*.

Oto, jak wygląda początek mojego pliku:

```
{
"type": "FeatureCollection",
"features": [ { "type": "Feature", "properties":
{ "scalerank": 0, "featurecla": "Ocean" },
"geometry": { "type": "Polygon", "coordinates":
   [ [ [ 49.110290527343778, 41.28228759765625 ],
         [ 48.584472656250085, 41.80889892578125 ],
   [ 47.492492675781335, 42.9866943359375 ],
         [ 47.590881347656278, 43.660278320312528 ],
   [ 46.682128906250028, 44.609313964843807 ],
         [ 47.675903320312585, 45.641479492187557 ],
   [ 48.645507812500085, 45.806274414062557 ]
   ...
```

Zaczyna wyglądać podobnie!

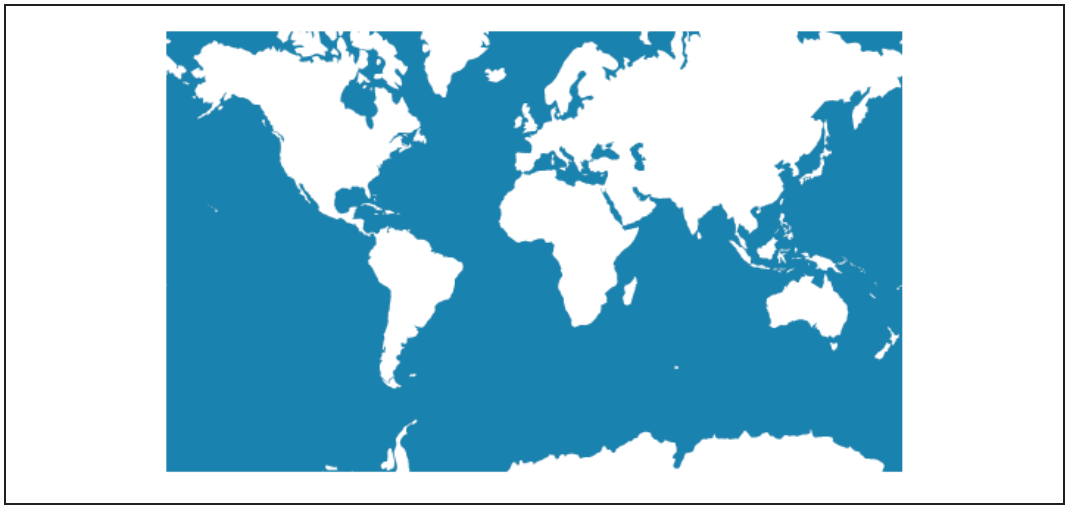
Pełni nadziei i podekscytowani wizją zbliżającego się sukcesu kopiujemy nowy plik GeoJSON do katalogu biblioteki D3. Swojemu zmieniłem nazwę na *oceans.json*. Skopiowałem też omawiany wcześniej plik HTML i w kodzie D3 zmieniłem tylko nazwę z `us-states.json` na `oceans.json`. Po otwarciu pliku w oknie przeglądarki otrzymałem wynik jak na rysunku 12.8.



Rysunek 12.8. GeoJSON utrzymuje, że to oceany świata…

Fuj! Co to ma być?! Nie wiem, ale jeśli chcesz zobaczyć to na własnym ekranie, zajrzyj do pliku *08_oceans.html*.

Tak się śpieszyłem, że zapomniałem zaktualizować funkcję rzutującą! Zmiana jest niewielka: wystarczy zastąpić `albersUsa` wartością `mercator` (rysunek 12.9).



Rysunek 12.9. GeoJSON prezentuje oceany świata — tym razem we właściwym rzucie

Poprawny kod znajdziesz w pliku *09_mercator.html*, gdzie możesz zobaczyć ścieżki GeoJSON przedstawiające kontury linii brzegowych — pobrane, sparsowane i wyświetlone.

Skorowidz

A

adres
  URI, 28
  URL, 28
akapity
  dynamicznie przypisywane style, 86
aktualizacja
  rodzaje, 161
aktualizowanie
  danych, 139
  elementów graficznych, 141
  funkcji skalujących, 150
  osi, 153
  referencji, 170
animacje
  efekty, 147
Apache, 28, 70
Arbor.js, 24
argument
  bounce, 147
  circle, 147
  elastic, 147
argumenty funkcji
  przekazywanie, 53
atrybut
  alt, 88
  class, 88
  fill, 62
  id, 88
  metody attr(), 88
  określanie, 88
  opacity, 62, 64
  rgba(), 64
  shape-rendering, 128
  src, 88
  stroke, 62
  text-anchor, 107
  transform, 129
  width, 88

B

biblioteka D3, 16, 19, 22
  dane, 76
    dowiązywanie, 75
  dodawanie etykiet, 106
  dokumentacja, 234
  funkcje, 19
    brakujące, 20
    skalujące, 115, 123
  generowanie kolorów kategorii, 198
  kompatybilność, 66
  mechanizmy rzutowania, 212
  nakładki, 26
  odwołania, 68
  odwzorowania wielowartościowe, 105
  osie wykresów, 125
  pętle, 52
  pobieranie, 67
  układy wykresów, 195
  zamienniki, 23
  zaznaczenia, 162
  znaczniki
    wybieranie, 79
  zwracanie metod, 75
blokowy zakres, 56
błędy wczytywania danych
  obsługa, 78

C

centroid, 199
comma-separated values, 76
Crossfilter, 26
CSS
  komentarze, 41
  pseudoklasa hover, 179
  wprowadzanie stylów SVG
    korzyści, 63
CSV, 76
  przechowywanie wartości, 77
Cubism, 26

D

D3, 19
dane, 71
  a grafika, 75
  aktualizowanie, 139
  definiowanie kształtów, 96
  dowiązywanie, 75, 80
  geograficzne, 209
    parsowanie, 221
    pobieranie, 221
    rozdzielczość, 221
    szerokość i długość, 218
    upraszczanie kształtów, 223
  geolokalizacyjne w JavaScript, 49
  jednoczesne dodawanie i usuwanie, 174
  kodowanie wartości w kolorach, 104
  losowe, 92, 149
  łączenie, 169
  miejsce zapisywania, 84
  o złożonej strukturze, 109
  odwzorowanie graficzne, 14
  powiązanie, 71
  przedstawianie w formie graficznej, 87
  sieciowe, 203
  skumulowane, 201
  sortowanie, 184
  tekstowe, 71
  używanie, 82
  wczytywanie
    format JSON, 79
    obsługa błędów, 78
    z pliku CSV, 76
  wykres punktowy, 108
  zmienianie, 141
  zmienne, 135
Dashku, 26
Data-Driven Documents, *Patrz* biblioteka D3
DataWrapper, 23
dc.js, 26
dimensional charting, 26
długość geograficzna, 210
Document Object Model, DOM 35, 41
  zmiana wyglądu elementów, 39
dodawanie etykiet, 106
dowiązywanie danych, 75
dziedzina wartości wejściowych, 114

E

edytory dla programistów, 68
eksportowanie grafiki, 227
elementy graficzne
  aktualizowanie, 141
  przechowywanie w ścieżkach
    przycinających, 158
  wyróżnianie kursorem myszy, 179
event listener, 141

F

Flare, 22
Flot, 23
for, 51
funkcja, 53, 73
  aktualizująca, 135
  anonimowa, 83
    jako wartość własności, 105
  asynchroniczna, 211
  call(), 127
  d3.max(), 116, 216
  d3.min(), 116, 216
  d3.scale.category(), 123
  d3.scale.ordinal(), 138
  d3.time.scale(), 123
  delay(), 147
  dostępowa, 117, 171
  duration(), 145, 148
  ease(), 146
  formatująca
    sprawdzanie poprawności, 134
  klucza, 169, 171
  kumulująca, 201
  log, 123
  Modernizr.load(), 66

nasłuchująca zdarzeń, 140
dowiązywanie, 177
powiązanie ze znacznikiem, 182
nazwana, 83
ordinal, 123
porównująca, 185
pow, 123
push, 132
quantile, 123
quantize, 123, 215
rangeBands(), 138
rangeRoundBands(), 138
rzutująca, 214
argumenty, 219
scale, 115
skalująca, 113
aktualizowanie, 150
dziedzina wartości wejściowych, 114
porządkowo, 136
przygotowywanie, 115
zbiór wartości wyjściowych, 114
sortująca, 184
sqrt, 123
struktura definicji, 83
tickFormat(), 133
ticks(), 130
tworzenie własnych, 83
xAxis, 126
xScale, 118
zwrotna, 77, 78

G

GDAL, 224
generator ścieżek, 211
Geographic Information Systems, 221
GeoJSON, 49, 209
geokoder, 218
geokodowanie, 218
Geometry Engine, Open Source, 224
GEOS, 224
Geospatial Data Abstraction Library, 224
Get Lat+Lon, 210
ggplot2, 20
GIS, 221
globalna przestrzeń nazw, 56
Google Chart Tools, 23
graf, 24, 203
grafika trójwymiarowa
narzędzia, 25
grafiki wektorowe, 20
gRaphaël, 23

H

HighchartsJS, 23
Homebrew, 224
HTML, 29
atrybuty znaczników, 33
dokument
dodawanie reguł stylów, 41
element, 35
blokowy, 39
hierarchiczna struktura dokumentu, 35
inline, 42
inspektor kodu, 37
wyświetlanie znaczników, 39
klasy i identyfikatory, 34
kod, 29
komentarze, 34
narzędzia programisty w przeglądarkach, 35
odwołanie do zewnętrznego arkusza stylów, 42
osadzenie kodu CSS, 41
parsowanie, 35
podgląd kodu źródłowego, 36
szablon strony, 68
treść i struktura, 30
znaczniki, 31
budowanie struktury, 31
podpowiedzi, 191
reguły stylów, 42
zagnieżdżanie, 31
HTTP, 28
HTTPS, 28
Hypertext Transfer Protocol, *Patrz* HTTP

I

identyfikator
tooltip, 190
instrukcja
for, 51
if, 51
interaktywność wizualizacji, 177
internet
zasada działania, 27
interpreter języka Python, 69

J

JavaScript, 44
  funkcje, 53
    anonimowe, 54
  GeoJSON, 49
  globalna przestrzeń nazw, 56
  instrukcje, 46
    sterujące, 51
  JSON, 49
  konsola, 45
  komentarze, 54
  łączenie struktur danych, 48
  model zdarzeń, 177
  notacja obiektowa, 49
  obiekty, 47
  operatory
    matematyczne, 50
    porównania, 50
  tablice, 46
    wykorzystanie, 52
  typowanie dynamiczne, 55
  wartości, 47
  właściwości, 47
  wprowadzanie skryptów na stronę, 54
  zasięg funkcyjny, 56
  zmienne, 45
    i, 52
    łańcuchowe, 46
    typy, 55
    unoszenie, 55
  znaki końca linii, 74
JavaScript InfoVis Toolkit, 23
JavaScript Object Notation, 49
język luźno typowany, 55
JIT, *Patrz* JavaScript InfoVis Toolkit
jqPlot, 23
jQuery
  przejścia, 157
jQuery Sparklines, 24
JSON, 49

K

kartogram, 214
Kartograph, 24
kaskadowe arkusze stylów, 39
  dziedziczenie, 43
  kaskadowość, 44
  szczegółowość, 44
klasa, 89
  axis, 127
  hidden, 192
klienty sieciowe, 28
kod w przykładach, 17
kolejność indeksowania, 169, 175
kolory
  formaty, 41
konwencja typograficzna, 10
kreślenie wymiarowe, 26

L

LAMP, 70
Leaflet, 25
licencja BSD, 19
linia bazowa, 202

Ł

łączenie metod, 73

M

MAMP, 70
MapShaper, 223
mapy
  bitowe, 20, 227
  geograficzne
    dodawanie punktów, 217
    skala, 222
marginesy, 121
metoda, 73
  append(), 74, 96, 126
  arc.centroid(), 199
  Array.shift(), 169
  asynchroniczna, 78
  attr(), 85, 88
  axis(), 126
  bars.enter(), 163
  clamp(), 122
  classed(), 89
  d3.csv(), 76
  d3.geo.path(), 211
  d3.json(), 79, 211
  d3.layout.pie(), 196
  d3.layout.stack(), 200
  d3.max(), 116, 216
  d3.min(), 116, 216
  d3.ordinal.categoty10(), 198

d3.range(), 137
d3.scale.category(), 123
d3.scale.linear, 122
d3.scale.ordinal(), 138
d3.svg.arc(), 197
d3.svg.axis(), 126
d3.time.scale(), 123
d3.tsv(), 79
data(), 83, 91, 96, 175
domain(), 115
duration(), 145, 148
each(), 155
bez przejścia, 158
ease(), 146
enter(), 79, 96
łączenie, 73
Math.floor(), 94
Math.random(), 93
Math.round(), 94
Math.sqrt(), 110
nice(), 122
on(), 178
parseFloat(), 189
pie(), 197
projection(), 219
przekazywanie, 75
range(), 116
rangeRound(), 122
remove(), 167
select(), 74
selectAll(), 74, 96, 174, 178
selection.data(), 76
selection.on(), 141
sort(), 185
sortBars(), 184
style(), 85, 88, 90
text(), 74, 85
transition(), 145
model pudełkowy, 38
Modernizr, 66
Modest Maps, 25
MySQL, 70

N

nakładanie maski, 159
normalizacja, 115
NVD3, 26

O

obiekt
window, 56
dodawanie wartości, 58
obiektowy model dokumentu, 35
obrazy wektorowe, 58
odwoływanie
do znaczników, 96
odwzorowanie, 212
Albersa, 212
kartograficzne, 24
promienia, 110
wielowartościowe, 105
odwzorowywanie, 24
danych
graficzne, 14
reguły, 14
operator
łączenia
znak +, 111
przypisania, 45
osiadanie, 146
osie, 113
aktualizowanie, 153
wykresu, 125
oznaczanie, 31

P

Paper.js, 25
Peity, 24
PhiloGL, 26
PHP, 70
piksel, 59
pliki
kształtów, 221
poziom szczegółowości, 221
upraszczanie, 223
źródła, 221
PDF, 228
SVG, 229
podpowiedzi, 187
przeglądarki, 188
wyświetlane w znacznikach
div, 191
SVG, 189
podwójne kodowanie, 104
pole
coordinates, 210
geometry, 210

Polychart.js, 26
Polymaps, 25
Portable Document Format, 228
prefuse, 21
Processing, 25
Processing.js, 25
Protovis, 22
przeglądarki
  kod HTML, 35
  rozwój, 21
przejścia, 135, 144
  charakter ruchu, 146
  etapowe, 148
  początek i koniec, 155
  skalowanie opóźnień, 148
  usuwanie elementów, 167
  wyjściowe, 172
przezroczystość, 63
przycinanie ścieżek, 159
pseudoklasa
  hover, 179

R

radiany, 196
Raphaël, 25
referencje
  aktualizowanie, 170
reguła
  arkusza CSS, 40
  pointer-events: none, 192
  rect: hover, 179
renderowanie, 38
Rickshaw, 26

S

Scalable Vector Graphics, 58
selektor, 39
  identyfikatorów, 40
  klasowy, 40
  potomny, 40
  rodzaje, 40
  szczegółowość, 44
serwer
  stron WWW
    języka Python, 69
    konfiguracja, 69
    przygotowanie, 69
    internetowy, 28
    lokalny, 28
    zdalny, 28
shapefiles, 221
Shneiderman Ben, 15
Sigma.js, 24
skala, 113
  ilościowa
    zbiór wartości wyjściowych, 138
  liniowa
    normalizacja, 115
  porządkowa, 136
    dziedzina danych wejściowych, 137
    referencja, 139
    zbiór danych wyjściowych, 138
skalowanie, 113
  dynamiczne, 118
  osi, 118
  potęgowe, 123
  promieni punktów, 121
składnia łączenia, 73
słowo kluczowe
  this, 180
  var, 45
sortowanie
  kliknięciem, 184
spójność obiektowa, 169
struktura
  dokumentu HTML, 31
  semantyczna, 30
  wizualna, 30
SVG, 58
  definiowanie stylów znaczników, 62
  kod
    dodawanie do dokumentu HTML, 59
  kolejność rysowania, 63
  kompatybilność, 65
  nazwy właściwości, 128
  pliki, 59
  praca z tekstem, 61
  proste kształty, 59
  przezroczystość, 63
  przycinanie ścieżek, 159
  przygotowanie plików, 59
  transformacje, 129
  tworzenie znaczników, 95
  warstwy, 63
  właściwości, 94
  zapisywanie obrazów, 229

znacznik, 59
atrybuty, 128
grupowanie, 182
podpowiedzi, 189
reguły stylów, 128
systemy projektowania, 14
szerokość geograficzna, 210

Ś

ścieżki, 211

T

Tableau, 20
tablica
danych, 77
obietów, 170
Three.js, 26
Timeline.js, 24
tooltips, 187
TopoJSON, 223
transformacja translacyjna, 129
transition, 135
Tributary, 26
TSV, 79
Tweeter, 235
tworzenie
akapitów, 80
form wizualnych
aplikacje, 25
znaczników
DOM, 71
SVG, 95
typowanie dynamiczne, 55

U

układy wykresów
kołowy, 196
siłowy, 203
skumulowany, 200
unoszenie zmiennych, 55
update, 135
urządzenia dotykowe, 193
User Interface Library, 24

V

variable hoisting, 56

W

W3, 19
WAMP, 70
wartości przeskalowane, 118
WebGL, 25
wizualizacja
dynamiczna, 15
interaktywna, 15
objaśniająca, 20
poznawcza, 20
statyczna, 14
w internecie, 15
wizualizowanie, 13
właściwość, 39
i wartość, 41
World Wide Web, 19
wprowadzanie
skryptów na stronę, 54
stylów, 41
wykres
kolumnowy, 87
kołowy, 196
nadawanie stylów, 90
osie, 125
formatowanie opisów, 133
pionowe, 131
pierścieniowy, 199
punktowy, 108, 125
etykiety, 111
etykiety osi, 126
poprawianie, 119
przygotowanie, 108
rozmiar punktów, 110
skalowanie, 116
z danymi generowanymi losowo, 133
siłowy, 203
słupkowy, 87
aktualizowanie danych, 142
dodawanie wartości, 162
etykiety, 106
kolorowanie, 104
położenie słupków, 101
przygotowywanie, 98
skumulowany, 200
szerokość słupków, 102
usuwanie wartości, 166
sparkline, 24
układy, 195
wywołanie funkcji, 53

X

XAMPP, 70
Xcode, 224

Y

YUI Charts, 24

Z

zasięg funkcyjny zmiennej, 56
zaznaczenie
  aktualizujące, 162, 164, 175
  wejściowe, 163, 175
  wyjściowe, 166
zbiór wartości wyjściowych, 114
zdarzenie
  click, 141
  mouseout, 181
  mouseover, 141
  w JavaScript, 177
  wskaźnikowe, 183
ziarnistość, 221
zmienna
  bars, 163
  charge, 204
  cy, 97
  error, 78
  i, 97
  padding, 120
znacznik
  <!DOCTYPE html>, 32
  a, 32
  body, 32
  circle, 60, 96
  clipPath, 159
  div, 32, 87
  DOM
    tworzenie, 71
  ellipse, 60
  em, 32
  g, 126
  grupowania, 182
  h1, 32
  head, 32
  html, 32
  li, 32
  line, 60
  meta, 68
  nadawanie stylów, 61
  nakładanie a zdarzenia, 183
  odwoływanie, 96
  ol, 32
  p, 32
  path, 61, 197
  rect, 60
    punkty odniesienia, 103
  script, 68
  span, 32
  strong, 32
  svg, 59
  text, 61, 106
  title, 32
  ul, 32
  usuwanie, 166
  wchodzący, 163
  wyjściowy, 166

Ź

źródła
  książki, 234
  strony WWW, 234

Programowanie

O'REILLY®

Interaktywna wizualizacja danych

Skoro mawia się, że obraz jest wart więcej niż tysiąc słów, można też powiedzieć, że jeden wykres jest wart więcej niż tysiąc tabel z danymi. Dlatego właśnie użytkownicy uwielbiają wykresy! Jeden rzut oka pozwala wyrobić sobie zdanie na temat każdego problemu. Jeżeli do tego dodać interaktywne rozwiązania, efekt może być tylko jeden — zachwyt Twoich klientów! Biblioteka D3 pozwala na budowanie interaktywnych wykresów, które pozwolą Ci na efektowną prezentację posiadanych informacji.

W trakcie lektury tej książki przekonasz się, jak szybko można przygotować atrakcyjny wykres, prezentujący nawet najbardziej skomplikowane dane. Najpierw gruntownie poznasz podstawy HTML-a, JavaScriptu oraz formatu SVG. Po tym wstępie będziesz gotowy, by rozpocząć przygodę z biblioteką D3! Określanie osi, skalowanie, efekty specjalne to tylko niektóre z poruszanych tematów. Gdy już opanujesz tworzenie wykresów, pora przejść do kolejnego rozdziału. Dowiesz się z niego, jak na posiadany wykres nanieść interaktywne dodatki. Na koniec zobaczysz, jak nakładać dane na mapy geograficzne oraz eksportować stworzone wykresy. Tę książkę doceni każdy programista, który kiedykolwiek stanął przed problemem wizualizacji danych.

Przekonaj się, jak łatwo można:

- tworzyć wykresy
- korzystać z danych w formacie CSV
- skalować wykresy
- dodawać do wykresów interaktywne informacje

Zachwyć użytkowników przydatnymi wykresami!

ISBN 978-83-246-8172-3
9 788324 681723
Cena 59,00 zł