

Odnieś sukces na rynku gier!



# Profesjonalne tworzenie gier internetowych dla systemu Android w językach HTML5, CSS3 i JavaScript

Juriy Bura • Paul Coates



Apress®



Tytuł oryginału: Pro Android Web Game Apps: Using HTML5, CSS3 and JavaScript

Tłumaczenie: Krzysztof Rychlicki-Kicior

ISBN: 978-83-246-7401-5

Original edition copyright © 2012 by Juriy Bura and Paul Coates.

All rights reserved.

Polish edition copyright © 2013 by HELION SA.

All rights reserved.

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.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą 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](mailto: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/prtwgi>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

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

Printed in Poland.

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

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

# Spis treści

	<b>O autorach .....</b>	<b>13</b>
	<b>Podziękowania .....</b>	<b>15</b>
	<b>Wprowadzenie .....</b>	<b>17</b>
<b>Część I</b>	<b>Światy dwuwymiarowe</b>	
<b>Rozdział 1.</b>	<b>Zaczynamy .....</b>	<b>23</b>
	Narzędzia .....	24
	Co jest nam potrzebne? .....	24
	Java Development Kit .....	27
	Zintegrowane środowisko programistyczne .....	27
	Serwer stron internetowych (WWW) .....	33
	SDK systemu Android i emulator .....	34
	Techniki .....	38
	Kod .....	38
	Programowanie zorientowane obiektowo .....	43
	Słowo o przeglądarkach mobilnych .....	51
	Podsumowanie .....	51
<b>Rozdział 2.</b>	<b>Grafika w przeglądarce — płótno .....</b>	<b>53</b>
	Gry od podszewki .....	53
	Rysowanie wewnątrz przeglądarki .....	55
	Podstawowe ustawienia strony HTML .....	55
	Czym jest płótno? .....	56
	Kontekst .....	57
	Układ współrzędnych .....	58

Rysowanie kształtów .....	61
Prostokąty .....	62
Ścieżki .....	63
Podścieżki .....	71
Kontury i wypełnienia .....	72
Kolory jednolite .....	73
Gradienty .....	73
Wzorce .....	77
Stan kontekstu a przekształcenia .....	79
Przesunięcie .....	80
Skalowanie .....	81
Obrót .....	82
Grupowanie transformacji .....	82
Stan kontekstu .....	84
Przekształcenia kontekstu w praktyce .....	85
Widok przykładowego projektu — efekt końcowy .....	86
Podsumowanie .....	89
<b>Rozdział 3. Tworzymy pierwszą grę! .....</b>	<b>91</b>
Szkielet gry HTML5 .....	92
Podstawowy szkielet .....	92
Wymuszanie orientacji .....	96
Architektura gry .....	98
Tworzymy grę .....	99
Renderowanie planszy .....	100
Stan i logika gry .....	105
Scalanie komponentów — klasa Game .....	110
Dodanie gry do szkieletu HTML .....	113
Podsumowanie .....	115
<b>Rozdział 4. Animacje i sprite'y .....</b>	<b>117</b>
Sprite'y .....	118
Wczytywanie obrazków .....	119
Rysowanie obrazka .....	129
Arkusze sprite'ów .....	133
Podstawy animacji .....	136
Najprostsza animacja .....	136
Wątki a JavaScript .....	137
Timery .....	138
Poprawianie jakości animacji .....	141
Podsumowanie .....	153
<b>Rozdział 5. Obsługa zdarzeń i zachowań użytkownika .....</b>	<b>155</b>
Zdarzenia w przeglądarce .....	156
Przeglądarka stacjonarna a wejście przeglądarki systemu Android .....	156
Stosowanie zdarzeń do przechwytywania wejścia użytkownika .....	157

Co jeszcze kryje się za zdarzeniami? .....	160
Sposób obsługi różnic pomiędzy interfejsami dotykowymi a myszą .....	162
Własne zdarzenia .....	165
Generatory i funkcje nasłuchujące własnych zdarzeń .....	166
EventEmitter — klasa bazowa .....	166
Zdarzenia a wywołania zwrotne .....	169
Własne zdarzenia .....	170
Implementacja klasy InputHandlerBase .....	173
Tworzymy klasę MouseInputHandler .....	176
Tworzymy klasę TouchInputHandler .....	178
Zaawansowane operacje wejściowe .....	179
Przeciągnij i upuść .....	179
Doskonałe wybieranie a maski obrazków .....	181
Złożone operacje .....	182
Symulowanie dżojstika .....	185
Podsumowanie .....	188
<b>Rozdział 6. Renderowanie wirtualnych światów .....</b>	<b>189</b>
Mapy z kafelków .....	189
Zasada działania map kafelkowych .....	190
Implementacja mapy kafelków .....	191
Mierzymy FPS .....	196
Optymalizacja wydajności renderowania .....	198
Rysuj tylko to, co potrzebne .....	198
Bufor pozaekranowy .....	200
Przechowywanie okolic obszaru widoku w pamięci .....	203
Obiekty świata gry .....	206
Układy współrzędnych .....	206
Implementacja klasy WorldObjectRenderer .....	208
Porządek sortowania .....	211
Optymalizacje .....	213
Widok izometryczny .....	214
Podsumowanie .....	217
<b>Rozdział 7. Tworzymy silnik izometryczny .....</b>	<b>219</b>
Konfiguracja .....	220
Plan .....	221
Przygotowanie przestrzeni roboczej .....	222
Podstawowy kod .....	222
Przydatne mechanizmy .....	225
Teren izometryczny .....	231
Układy współrzędnych .....	231
Renderowanie kafelków .....	232
Implementowanie klasy IsometricTileLayer .....	237

Renderowanie obiektów .....	244
Implementacja klastrów obiektów .....	248
Pamięć podręczna obiektów .....	250
Obsługa ruchu .....	253
Obiekty złożone .....	255
Warstwa obiektów — kolejne czynności .....	257
Brudne prostokąty .....	257
Jak to działa? .....	258
Implementacja .....	259
Integracja z warstwami .....	263
Oznaczanie brudnych prostokątów .....	266
Interfejs użytkownika a menedżer warstw .....	268
Menedżer warstw .....	268
Interfejs użytkownika .....	271
Interakcja .....	274
Propagowanie i obsługa zdarzeń .....	275
Zatrzymywanie propagowania zdarzeń .....	278
Podsumowanie .....	279

## Część II Światy trójwymiarowe

<b>Rozdział 8. 3D w przeglądarce .....</b>	<b>281</b>
Renderowanie 3D — wprowadzenie .....	282
Jak działa renderowanie 3D? .....	283
Przekształcenia matematyczne .....	283
Przykład w trójwymiarze .....	283
Uproszczony silnik 3D .....	285
Model i scena .....	286
Renderowanie .....	287
Podsumowanie .....	299
<b>Rozdział 9. WebGL .....</b>	<b>301</b>
Podstawy WebGL .....	302
Inicjalizacja WebGL .....	302
Geometrie .....	304
Potok renderowania OpenGL ES 2.0 .....	305
Stosowanie buforów .....	307
Shadery i GLSL .....	308
Prosty przykład — renderujemy sześcian .....	313
Zastosowanie shaderów na stronie internetowej .....	313
Renderowanie aplikacji typu Witaj, świecie! .....	315
Eksplorowanie świata WebGL .....	319
Kolor .....	319
Tekstury .....	324
Ładowanie modeli .....	327
Podsumowanie .....	330

## Część III Łączenie światów

<b>Rozdział 10. Pora na serwer .....</b>	<b>331</b>
Podstawy Node.js .....	332
Wprowadzamy bibliotekę Node.js .....	332
Jak programować w Node.js? .....	333
Instalacja Node.js .....	335
Debugowanie skryptów Node.js .....	336
Pisanie skryptów w Node.js .....	337
Wyjątki a stos wywołań .....	338
Globalna przestrzeń nazw a moduły Node .....	338
Tworzymy pierwszy moduł .....	341
Wykrywanie modułów .....	344
Stosowanie NPM .....	345
Node.js w praktyce — tworzymy serwer dla gry .....	347
Frameworki do tworzenia aplikacji webowych w Node .....	347
Najpierw podstawy .....	348
Renderowanie stron internetowych .....	351
Parsowanie wejścia użytkownika .....	355
Stosowanie sesji .....	357
Warstwa pośrednia .....	358
Zachowanie porządku .....	360
Zgłaszanie błędów .....	360
Obsługa dziennika zdarzeń .....	364
Konfiguracje serwera .....	366
Podsumowanie .....	368
<b>Rozdział 11. Rozmawiamy z serwerem .....</b>	<b>369</b>
Ewolucja komunikacji sieciowej w przeglądarkach .....	369
Konfiguracja serwera .....	371
XMLHttpRequest API do obsługi żądań HTTP .....	372
Obsługa żądania HTTP przy użyciu XHR .....	373
Obsługa błędów w XHR .....	374
XMLHttpRequest Level 2 .....	375
Obsługa danych binarnych .....	376
Odwrotny Ajax .....	378
Problem .....	378
Rozwiązania .....	378
Najlepsze rozwiązania .....	379
Dopuszczalne rozwiązania .....	380
Przestarzałe rozwiązania .....	385
Testujemy transporty .....	386
DDMS .....	386
Symulowanie złych warunków sieci przy użyciu zewnętrznego oprogramowania .....	387
Podsumowanie .....	388

<b>Rozdział 12. Tworzymy grę sieciową .....</b>	<b>389</b>
Architektura gry sieciowej .....	389
Architektura gry — od jednego do wielu graczy .....	391
Struktura projektu .....	393
Lobby w grze przy użyciu Socket.IO .....	394
Komunikacja klient-serwer .....	395
Dodawanie ekranu lobby do gry .....	397
Dodawanie obsługi rozgrywki .....	402
Współdzielenie logiki pomiędzy klientem a serwerem .....	403
Warstwa serwerowa .....	404
Warstwa kliencka .....	408
Podsumowanie .....	414
<b>Część IV Usprawnianie światów</b>	
<b>Rozdział 13. Sztuczna inteligencja w grach .....</b>	<b>415</b>
Czy potrzebuję AI w mojej grze? .....	416
Wyszukiwanie ścieżek .....	416
Grafy .....	418
Czym jest graf? .....	418
Implementacja grafów w języku JavaScript .....	420
Algorytmy wyszukiwania ścieżek .....	424
Algorytm A* .....	424
Metody tworzenia grafu wyszukiwania ścieżek .....	430
Podjmowanie decyzji .....	432
Podsumowanie .....	435
<b>Rozdział 14. Silniki gier w języku JavaScript .....</b>	<b>437</b>
API graficzne, biblioteki i silniki gier .....	438
API graficzne .....	438
Biblioteki graficzne .....	439
Silniki gier .....	440
Crafty .....	441
System komponentów encji .....	441
Witaj, świecie! w Crafty .....	444
Tworzymy grę w Crafty .....	447
Ostateczna wersja .....	453
Podsumowanie .....	455
<b>Rozdział 15. Tworzenie natywnych aplikacji .....</b>	<b>457</b>
Aplikacje natywne .....	458
Konfiguracja Apache Cordova (PhoneGap) .....	459
Konfiguracja narzędzia Cordova .....	460
Konfiguracja Apache Ant .....	460



Budowanie aplikacji natywnej .....	461
Tworzenie pustego projektu dla systemu Android .....	461
Testowanie szkieletu aplikacji Android .....	462
Podstawowy projekt Cordova .....	463
Obsługa sieci .....	467
Ostatnie modyfikacje: nazwa, ikona i tryb pełnoekranowy .....	468
Stosowanie natywnych API .....	471
Przygotowanie do publikacji .....	473
Podpisywanie aplikacji .....	474
Publikowanie w serwisie Google Play .....	476
Aktualizacja aplikacji .....	479
Podsumowanie .....	481
<b>Rozdział 16. Obsługa dźwięków .....</b>	<b>483</b>
Dźwięki na stronach internetowych .....	484
Znacznik audio .....	484
Web Audio API .....	485
Dźwięk w domyślnej przeglądarce systemu Android .....	486
Stosowanie biblioteki SoundManager2 .....	487
Wstępna konfiguracja .....	487
Odtwarzanie w pętli .....	489
Obsługa dźwięku w grze .....	490
Odtwarzanie dźwięków w aplikacjach Cordova .....	493
Doświadczenie użytkownika .....	493
Podsumowanie .....	494
Co dalej? .....	494
<b>Dodatek A. Debugowanie aplikacji klienckich w JavaScriptcie .....</b>	<b>495</b>
Przykład do zdebugowania .....	495
Debugowanie w przeglądarce stacjonarnej .....	496
Debugowanie urządzeń mobilnych .....	499
Rejestrowanie informacji (niemal) bez zmian w kodzie .....	500
weinre .....	502
Podsumowanie .....	503
<b>Skorowidz .....</b>	<b>505</b>



## ROZDZIAŁ 8



# 3D w przeglądarce

W pierwszej części książki nauczyliśmy się tworzyć gry dwuwymiarowe. Przez wiele lat tego rodzaju gry definiowały maksimum możliwości, jakie można było osiągnąć. Dopiero później pojawienie się coraz bardziej realistycznych gier trójwymiarowych zaczęło zmieniać ten stan rzeczy. Jedynym powodem, dla którego nie możesz uruchomić *Skyrima* na swoim telefonie, są możliwości Twojego urządzenia.

Silniki trójwymiarowe wymagają znacznie więcej mocy obliczeniowej w porównaniu z dwuwymiarowymi. W przypadku silników dwuwymiarowych renderowanie jest całkiem proste — wystarczy wyświetlić na płótnie piksele pobrane z obrazka i ewentualnie uwzględnić jego przezroczystość. Silnik trójwymiarowy wymaga znacznie więcej mocy — związane z nim przekształcenia matematyczne są znacznie bardziej skomplikowane. Jest to też jeden z istotnych powodów, dla których przeglądarki (zarówno stacjonarne, jak i mobilne) do niedawna nie były w stanie natywnie obsługiwać możliwości 3D.

W tym rozdziale skupimy się wyłącznie na tworzeniu trójwymiarowych aplikacji przy użyciu przeglądarki. Obsługa trzech wymiarów w przeglądarce jest możliwa dzięki „standardowej” implementacji, nazywanej WebGL. W świecie mobilnym WebGL stawia dopiero pierwsze kroki. W momencie pisania tej książki jedyną przeglądarką (dostępną dla urządzeń z systemem Android), która obsługuje ten standard, jest Firefox Mobile<sup>1</sup>. Sony Xperia PLAY obsługuje WebGL w domyślnej przeglądarce. Ze względu na dynamiczny rozwój tej technologii w przeglądarkach stacjonarnych możemy się spodziewać, że niebawem będzie ona powszechnie dostępna również na urządzeniach mobilnych.

Im więcej urządzeń będzie obsługiwało grafikę trójwymiarową, tym więcej gier w tej technologii będzie dostępnych w naszych przeglądarkach. Z tego względu programista powinien wiedzieć, jak działają mechanizmy renderowania 3D i co trzeba samodzielnie zaimplementować, aby wszystko działało jak należy. Ten rozdział jest w całości poświęcony podstawom grafiki trójwymiarowej. Jednocześnie stanowi on niejako wprowadzenie do rozdziału 9., w którym poznamy podstawy WebGL.

W tym rozdziale poznasz następujące zagadnienia:

- podstawy renderowania 3D: jak przedstawiać bryły na płaskim ekranie,
- podstawy algebry macierzy: podstawowe przekształcenia matematyczne kryjące się za grafiką trójwymiarową,
- perspektywa i rzuty: jak sprawić, by scena była bardziej naturalna.

Celem tego rozdziału jest utworzenie prostego dema, w którym wyrenderujemy podstawowy model 3D. Rozpocznemy od najprostszego kształtu — sześcianu.

---

<sup>1</sup> W momencie tłumaczenia częściowe wsparcie oferuje też Opera Mobile — *przyj. tłum.*

## Renderowanie 3D — wprowadzenie

Czym właściwie jest grafika 3D? Pod jakim względem różni się ona od grafiki dwuwymiarowej, którą stosowaliśmy w poprzednich rozdziałach? Grafika 3D jest rzecz jasna iluzją — podobnie jak animacje. Renderując animację, tak naprawdę zmieniamy ramki na tyle szybko, aby użytkownik odniósł wrażenie, że obiekt się porusza. W rzeczywistości nie ma mowy o jakimkolwiek prawdziwym ruchu. Piksele pozostają na swoim miejscu — zmienia się jedynie ich kolor, co nie zmienia faktu, że na ekranie osiągamy efekt ruchu.

Renderowanie sceny trójwymiarowej przebiega podobnie. Wyświetlacz jest płaski, ale scena jest renderowana w taki sposób, aby użytkownik miał wrażenie, że patrzy na trójwymiarowy świat przez okno. Teraz musimy zadać sobie pytanie: czy skoro grafika 3D stanowi tak naprawdę specjalny przypadek grafiki 2D (którą dobrze znamy), to możemy zastosować kontekst płótna 2D do rysowania sceny trójwymiarowej? Jest to w istocie bardzo dobre pytanie. W praktyce utworzenie bardzo prostego silnika 3D jedynie przy użyciu płótna jest możliwe, jednak będziemy musieli sobie poradzić z dwoma podstawowymi problemami. Przede wszystkim będziemy musieli zmagać się ze znacznie większą liczbą obliczeń i przekształceń matematycznych. Ilość pracy, jaką trzeba włożyć od strony matematycznej (także biorąc pod uwagę wszelkiego rodzaju optymalizacje), w przypadku grafiki 3D i 2D różni rząd wielkości. Nawet jeśli poświęciłbyś kilka lat na nauczanie się wszystkiego, co konieczne do stworzenia własnego silnika 3D, szybko napotkałbyś inny problem: brak mocy urządzenia potrzebnej do obsługi zaimplementowanego własnoręcznie renderera 3D.

Gdy gracze, programiści tworzący gry i producenci sprzętu zrozumieli, że grafika trójwymiarowa stanowi niezbędny element gier komputerowych, zauważyli też, że procesor (ang. *Central Processing Unit*, CPU — centralna jednostka przetwarzania) nie jest najlepszym narzędziem do renderowania tego rodzaju grafiki. Podjęli więc decyzję o dodaniu specjalnych kart graficznych, które są odpowiedzialne wyłącznie za tego rodzaju obliczenia. Często karty te są droższe od samych procesorów! W języku JavaScript w przeglądarkach nie możesz bezpośrednio uzyskać dostępu do GPU (ang. *Graphics Processing Unit* — jednostka przetwarzania grafiki) w celu bezpośredniego wykonania renderowania właśnie w niej. Jesteś ograniczony tylko do silnika języka JavaScript, który wszystkie instrukcje wykonuje w ramach zwykłego procesora (CPU). Procesory nie są przystosowane do wykonywania tego rodzaju operacji, dlatego renderowanie zajmie znacznie więcej czasu niż w przypadku GPU.

Zanim w świecie aplikacji internetowych pojawiło się WebGL, utworzenie własnego renderera 3D stanowiło jedyną metodę obsługi grafiki trójwymiarowej w przeglądarce bez użycia Flasha lub Javy. Programiści tworzyli własne silniki; niektóre z nich były naprawdę interesujące. Niestety, koniec końców zawsze czegoś brakowało, ponieważ nie da się stworzyć gry trójwymiarowej o przyzwoitej jakości w ramach kontekstu 2D. WebGL działa inaczej — dzięki niemu jesteś w stanie przekazać najgorszą część obliczeń do GPU.

Większość telefonów dostępnych na rynku nie dysponuje jeszcze odrębną jednostką GPU, niemniej jest to tylko kwestia czasu. Co zatem stanie się, jeśli uruchomisz WebGL na takim urządzeniu? Czy to API w ogóle zadziała? Tak, oczywiście. Problem w tym, że do renderowania sceny zostanie użyty CPU, przez co nie uzyskasz zbyt wielu klatek na sekundę. Gdy scena jest renderowana przez GPU, mamy do czynienia z *renderowaniem sprzętowym* (ang. *hardware rendering*) — oznacza to, że obliczenia są wykonywane w ramach GPU, który jest przystosowany do tego typu pracy. Z drugiej strony renderowanie może być też przeprowadzone *programowo* (ang. *software rendering*). Ma ono miejsce, gdy renderujesz scenę przy użyciu CPU, np. z poziomu języka JavaScript bez użycia WebGL lub na urządzeniu, które w ogóle nie zawiera osobnej jednostki GPU.

- 
- **Uwaga** Problemy z działaniem WebGL możesz mieć nawet na swoim komputerze stacjonarnym. Istnieje specjalna „czarna lista” kombinacji urządzeń – sterownik – system operacyjny, które powodują problemy z działaniem tego API. Jeśli konfiguracja Twojego komputera zakwalifikuje się na tę czarną listę, kontekst WebGL nie będzie akcelerowany (przyspieszany), a nawet może w ogóle nie działać. Jeśli chcesz zaryzykować, możesz ustawić specjalny parametr, który spowoduje zignorowanie czarnej listy w trakcie działania przeglądarki. W przypadku Chrome musisz uruchomić przeglądarkę z parametrem `ignore-gpu-blacklist`. W Firefoksie musisz przejść na stronę `about:config` i ustawić zmienną `webgl.force-enabled` na `true`.
-

# Jak działa renderowanie 3D?

Ten podrozdział stanowił dla mnie jedno z największych wyzwań podczas pisania tej książki. Renderowanie 3D to temat niezwykle skomplikowany — nietrudno byłoby wypełnić treścią jedną, a nawet dwie książki poświęcone tylko temu zagadnieniu. Spróbuję wyjaśnić tylko najistotniejsze zagadnienia, abyś był w stanie zrozumieć bardziej zaawansowane algorytmy kryjące się za całym procesem. Postaram się również ograniczyć matematyczne opisy do minimum, jednak nie da się ich zupełnie uniknąć.

## Przekształcenia matematyczne

Zacznijmy od dobrej informacji — wszystkie operacje wykonywane w przestrzeni dwuwymiarowej będą działać prawidłowo również w trójwymiarze. Wszystko, co robimy w świecie 3D, działa podobnie jak w przypadku dwuwymiarowym, jednak musimy uwzględnić jeszcze jeden wymiar, nazywany zwyczajowo „z”. W świecie 2D środek odcinka oblicza się zgodnie z wzorami:

$$x = (x_1 + x_2)/2, y = (y_1 + y_2)/2$$

Jeśli dodamy jeszcze jeden wymiar, otrzymamy wzory na środek odcinka trójwymiarowego:

$$x = (x_1 + x_2)/2, y = (y_1 + y_2)/2, z = (z_1 + z_2)/2$$

To samo stwierdzenie jest prawdziwe przy obliczaniu długości odcinka pomiędzy dwoma punktami. W przypadku dwuwymiarowym wzór jest następujący:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Po dodaniu trzeciego wymiaru otrzymujemy wzór:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Czemu jest to takie istotne? Niektóre zagadnienia łatwiej zrozumieć w przypadku dwuwymiarowym — wzory i ilustracje są znacznie prostsze. Po zrozumieniu tych zagadnień w 2D zastosowanie ich w świecie trójwymiarowym staje się proste. Jeśli jakkolwiek omawiana kwestia jest dla Ciebie trudna do zrozumienia, spróbuj przeanalizować ją w przypadku dwuwymiarowym. Przeniesienie jej potem na grunt trójwymiarowy nie będzie stanowiło dla Ciebie problemu.

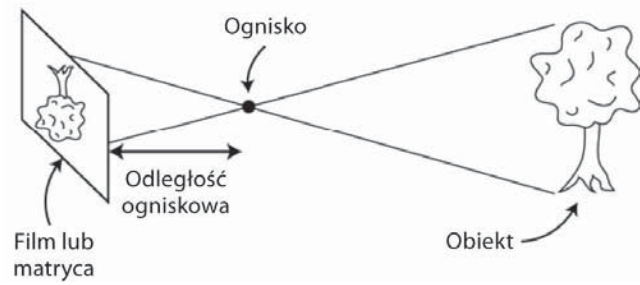
Oczywiście od każdej reguły są pewne wyjątki. W przypadku świata 2D linie albo się przecinają, albo są do siebie równoległe. W trójwymiarze linie mogą nie być równoległe ani się nie przecinać.

## Przykład w trójwymiarze

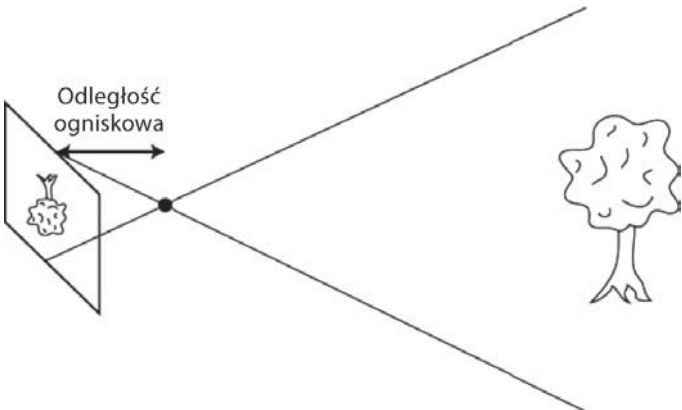
Zacznijmy od przykładu wziętego prosto z życia. Jaki jest najprostszy przykład renderowania świata trójwymiarowego na powierzchni dwuwymiarowej? Oczywiście jest to zdjęcie lub film wideo. Aparat robi zdjęcie sceny trójwymiarowej i umieszcza je na powierzchni 2D. Dawniej powierzchnią był film (taśma) — obecnie jest to cyfrowa matryca. Rysunek 8.1 przedstawia zasadę działania aparatu.

Światło odbijane przez obiekty jest przechwytywane przez soczewki i przekazywane do matrycy. Obraz jest odwrócony, co wynika ze sposobu działania soczewki.

Aparaty cyfrowe udostępniają funkcję przybliżenia (ang. *zoom*). Wciśnięcie przycisku powoduje powiększenie widocznego obiektu. Tak naprawdę wewnątrz aparatu następuje zmiana odległości ogniskowej, co przekłada się na zwiększenie lub zmniejszenie widocznego obszaru (rysunek 8.2). Większy widoczny obszar oznacza możliwość ujęcia na zdjęciu większej liczby obiektów przy jednoczesnym zmniejszeniu ich rozmiarów — innymi słowy, jest to operacja oddalenia (ang. *zoom out*). Odwrotna operacja powoduje zmniejszenie obszaru widoku i powiększenie obiektów, czyli jest to operacja powiększenia (ang. *zoom in*).

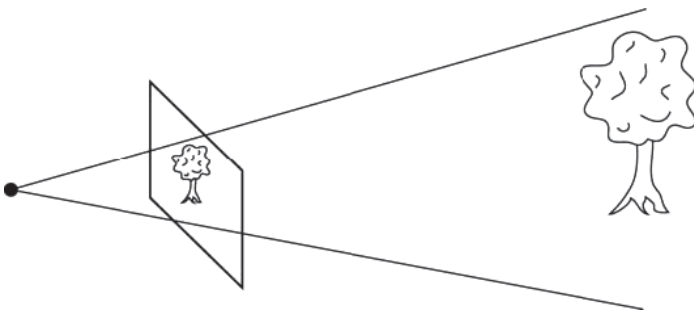


**Rysunek 8.1.** Zdjęcie z aparatu stanowi dobry przykład dwuwymiarowej projekcji używanej do przedstawienia fragmentu sceny trójwymiarowej



**Rysunek 8.2.** Zmiana ogniskowej powoduje przybliżanie lub oddalanie: im większy staje się obszar widoku, tym mniejszy jest obiekt na ekranie

Renderowanie sceny 3D przebiega niemal identycznie, przy czym nie korzystamy rzecz jasna z żadnych soczewek. Nie musimy więc umieszczać „docelowej” powierzchni dwuwymiarowej za ogniskiem. Musimy zachowywać się tak, jakbyśmy stali przed kamerą (rysunek 8.3).



**Rysunek 8.3.** W grafice komputerowej nie ma soczewek. Z tego względu płaszczyzna „przechwytyjąca” scenę musi się znaleźć przed ogniskiem. Mimo różnic zasady przybliżania i oddalania pozostają niezmienione

Przeanalizujmy teraz różnice pomiędzy renderowaniem sceny 3D a robieniem zdjęcia. W przeciwieństwie do świata rzeczywistego scena 3D jest tworzona z trójkąta (jest to do tej pory

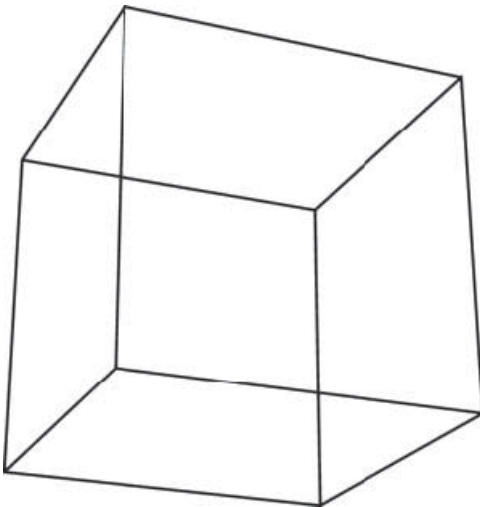
najlepszy sposób przedstawiania trójwymiarowych modeli). Trójkąty nie mają żadnych fizycznych właściwości, w przeciwieństwie do obiektów, z którymi stykasz się każdego dnia. Nie mają wagi, koloru, nie są stworzone z konkretnego materiału itd. Wszystkie właściwości musimy zasymulować, korzystając ze skomplikowanych modeli matematycznych.

Klikając przycisk aparatu w celu zrobienia zdjęcia, masz szczęście, gdyż proste zjawiska fizyczne umożliwiają osiągnięcie pożądanego efektu: światło odbija się w określony sposób od powierzchni, będąc mniej lub bardziej zaburzone w wyniku mgły, wysokiej temperatury lub innych warunków atmosferycznych. Koniec końców dociera ono do aparatu lub siatkówki Twojego oka, tworząc obraz. W świecie trójwymiarowym musimy samodzielnie zaimplementować prawa fizyki i przeprowadzać skomplikowane obliczenia, aby uzyskać pożądaną efekt, który będzie widoczny dla użytkownika. To właśnie dlatego silniki trójwymiarowe są takie skomplikowane.

Na szczęście mam dla Ciebie dobrą wiadomość — nie musisz przeprowadzać superrealistycznych symulacji wszystkich praw optyki, aby stworzyć silnik 3D. Jeśli pominiemy symulację oświetlenia i materiałów, silnik staje się w istocie dość prosty — jego zadaniem jest jedynie wyświetlanie wielokątów na ekranie. Programem typu *Witaj, świecie!* dla grafiki trójwymiarowej jest obracający się sześcian. Dojdziemy do tego — i to bez ani jednej instrukcji z WebGL. Dlaczego? Bo w ten sposób znacznie lepiej zrozumiesz zasady działania grafiki trójwymiarowej.

## Uproszczony silnik 3D

W tym podrozdziale stworzymy najprostszy możliwy silnik trójwymiarowy, korzystając jedynie ze znanego nam API płótna. Rozpocznemy od reprezentacji obiektów 3D w języku JavaScript. Następnie przejdziemy do macierzy — matematycznych obiektów pomagających renderować modele na ekranie. Dzięki macierzom i przekształceniom będziemy w stanie obrócić sześcian na ekranie. Na koniec zajmiemy się perspektywami i rzutami (projekcjami), które sprawiają, że nasz model będzie bardziej naturalny. Naszym celem jest wyświetlenie obracającego się sześcianu (rysunek 8.4).



**Rysunek 8.4.** Ostateczna wersja aplikacji wykonywanej w tym rozdziale — renderujemy sześcian

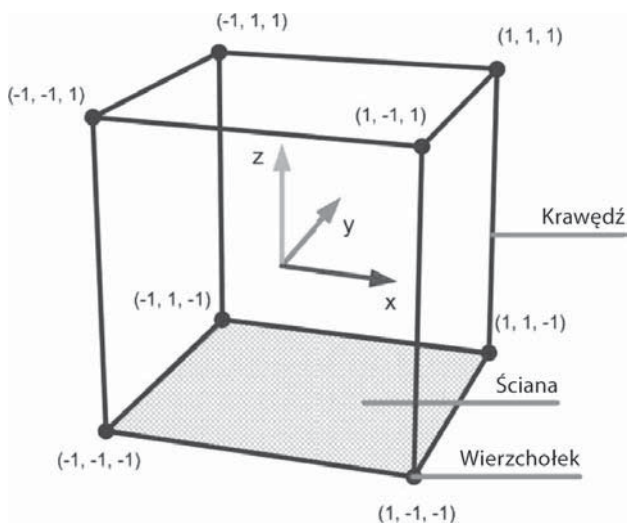
## Model i scena

Najpierw musimy określić dane niezbędne do wyrenderowania sceny. Przede wszystkim konieczne jest posiadanie listy wierzchołków tworzących sześcian lub dowolny inny model, który chcemy wyświetlić. Następnie musimy zdefiniować połączenia między punktami, czyli krawędzie. Pierwszym krokiem tworzenia tej aplikacji będzie zdefiniowanie sześcianu w języku JavaScript.

### Model

W przestrzeni trójwymiarowej model jest reprezentowany przy użyciu *wierzchołków* (ang. *vertexes*), *krawędzi* (ang. *edges*) i *ścian* (płaszczyzn — ang. *faces*). Wierzchołek to punkt w przestrzeni, krawędź to odcinek łączący dwa wierzchołki, a ściana to powierzchnia ograniczona krawędziami.

Sześcian ma osiem wierzchołków, dwanaście krawędzi i sześć ścian: każda krawędź łączy parę wierzchołków, a każda ściana jest ograniczona czterema krawędziami. Każdy punkt w przestrzeni 3D jest reprezentowany przy użyciu trzech współrzędnych:  $x$ ,  $y$  i  $z$ . Zerknijmy na rysunek 8.5, który przedstawia sześcian, kierunki układu współrzędnych i współrzędne wierzchołków sześcianu.



Rysunek 8.5. Sześcian w przestrzeni 3D

- 
- **Uwaga** W przestrzeni 3D nie używa się kierunków: góra, dół, lewo i prawo. Mamy do czynienia z trzema osiami, z których każda wskazuje inny kierunek. O tym, która oś wskazuje „górze”, decydujesz Ty — zazwyczaj przyjmuje się jednak, że wysokość jest określana na osi  $Z$ .
- 

Początek układu współrzędnych znajduje się w środku sześcianu — jest to punkt  $(0, 0, 0)$ . Dla uproszczenia przyjmujemy, że krawędzie sześcianu mają długość 2, a zatem wszystkie współrzędne wszystkich wierzchołków mają wartość  $-1$  lub  $1$ . Teraz musimy zapisać informacje o wierzchołkach w tablicy. Osiem jej elementów przechowuje wszystkie osiem wierzchołków w tablicy (listing 8.1).

#### Listing 8.1. Wierzchołki sześcianu

```
var vertices = [
  [-1, -1, -1], [-1, -1, 1], [-1, 1, -1], [-1, 1, 1],
  [1, -1, -1], [1, -1, 1], [1, 1, -1], [1, 1, 1]
];
```



Drugą część modelu stanowią krawędzie. Krawędź to odcinek, który łączy dwa wierzchołki. Nie musimy przechowywać współrzędnych wierzchołków krawędzi, ponieważ są one zdefiniowane w tablicy wierzchołków. Wystarczy więc odwoływać się do numerów wierzchołków, posługując się indeksami z pierwszej tablicy. Pierwsza krawędź na listingu 8.2 jest zapisana przy użyciu współrzędnych  $[0, 1]$ . Oznacza to, że wierzchołek o indeksie 0 łączy się z wierzchołkiem o indeksie 1. Zerknij do tablicy wierzchołków, a przekonasz się, że współrzędne tych punktów to  $(-1, -1, -1)$  i  $(-1, -1, 1)$ . Kod tablicy krawędzi został pokazany na listingu 8.2.

**Listing 8.2.** Krawędzie sześcianu — każdy element zawiera indeksy z tablicy wierzchołków

```
var edges = [
    [0, 1], [0, 2], [0, 4], [1, 3],
    [1, 5], [2, 3], [2, 6], [3, 7],
    [4, 5], [4, 6], [5, 7], [6, 7]
];
```

## Scena

Mamy już sześcián, dlatego teraz skupimy się na możliwości oglądania go pod różnymi kątami. W rzeczywistości nie możemy przemieszczać świata, dlatego zmieniamy położenie aparatu (kamery). W grafice trójwymiarowej zetknijemy się z odwrotną sytuacją — kamera jest nieruchoma, kierunek również nie ulega zmianie. W technologii WebGL kamera jest skierowana w dół, wzdłuż osi  $Z$ , przez co obiekt o mniejszej wartości współrzędnej  $Z$  wydaje się bardziej oddalony. Aby pokazać użytkownikowi żądany fragment sceny, musimy dopasować kąt i położenie kamery lub zmienić kąt i położenie świata. Zasada działania jest identyczna jak w poprzednim rozdziale: aby użytkownik odniósł wrażenie przemieszczania się po mapie, obszar widoku pozostanie nieruchomy, a będziemy poruszać właśnie mapą (światem).

Obrót to tylko jeden z przykładów przekształceń, które możesz zastosować do swojego modelu. Można także przemieszczać obiekt po scenie lub skalować go względem wybranych osi. Zaimplementowanie tych operacji samodzielnie nie jest trudne. Problemy zaczynają się, gdy zechcesz je grupować. Rozważmy przykład, w którym przesuwasz obiekt o pięć jednostek w lewo, obracasz go o  $\pi$  radianów wokół osi  $Z$  i skalujesz go trzy razy względem osi  $X$ . Zaimplementowanie takiego cyklu operacji stanowi dobre ćwiczenie matematyczne, jednak najpierw zajmiemy się innym mechanizmem o sprawdzonej skuteczności.

## Renderowanie

Po zdefiniowaniu współrzędnych modelu i zapisaniu ich w tablicy możemy wyrenderować sześcián na ekranie. Jak przekształcić te punkty w szkielet? W tym rozdziale nauczymy się wykonywać bardzo proste renderowanie 3D. Wyrenderowanie bryły trójwymiarowej wymaga obliczenia rzutu wierzchołków na ekranie. Po ich znalezieniu możemy narysować odcinki pomiędzy wszystkimi punktami, które są połączone krawędziami — i gotowe! W ten sposób otrzymujemy szkielet.

Zagadnieniem z zakresu matematyki, które pozwoli nam na znalezienie rzutów (projekcji) i wykonanie renderowania, jest *teoria macierzy*. Rozpoczniemy od najprostszego pojęcia w jej ramach, czyli samej macierzy. Po omówieniu zasad użycia macierzy będziemy mogli zastosować zdobytą wiedzę w praktyce. Wyrenderujemy sześcián, po czym zaczniemy go obracać i stosować odpowiednią perspektywę, dzięki której sześcián wyda się bardziej naturalny.

## Macierze

Czym jest macierz (ang. *matrix*)? Nie, nie chodzi nam o równoległą rzeczywistość, w której Neo walczy z agentem Smithem. Macierz to przede wszystkim elementarna struktura danych, niezbędna w każdej aplikacji trójwymiarowej. Rysunek 8.6 przedstawia przykładową macierz.

$$A = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & 8 & 3 & 1 \\ 4 & 5 & 3 & 2 \\ 1 & 0 & 0 & 2 \end{bmatrix}$$

**Rysunek 8.6.** Przykład macierzy

Macierz jest siatką liczb zbudowaną z wierszy i kolumn. Macierze mogą rzecz jasna przyjmować różne rozmiary. Rysunek 8.6 przedstawia macierz o rozmiarze 4 na 4. Naturalnym sposobem reprezentacji macierzy w języku JavaScript jest stosowanie tablic dwuwymiarowych. To, co jest naprawdę ważne w przypadku macierzy o tym rozmiarze, to fakt, że taka macierz może przechowywać dowolną liczbę przekształceń w dowolnej kolejności, jaką zastosujesz wobec swojej sceny. Jak? Otóż każda macierz reprezentuje pewne przekształcenie: obrót, skalowanie, przesunięcie (translację) lub połączenie wszystkich trzech. Rysunek 8.7 przedstawia kilka przykładów.

$$\begin{bmatrix} 0,5 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 5 & 2 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0,7 & -0,7 & 0 & 0 \\ 0,7 & 0,7 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**Rysunek 8.7.** Macierze reprezentują różne przekształcenia. Od lewej do prawej: zmniejszenie o 50% wzdłuż osi X, przesunięcie o pięć jednostek wzdłuż osi X i o dwie wzdłuż osi Y, obrót wokół osi Z o około 45 stopni

Jeśli nigdy nie korzystałeś z macierzy, rysunek 8.7 przedstawia dla Ciebie trzy siatki nieistotnych liczb. Całość stanie się bardziej czytelna, jeśli wiesz, jak mnożyć się macierze. Zasady nie są intuicyjne — znacznie łatwiej jest opisać je na macierzach o rozmiarze 2 na 2. Załóżmy, że chcesz pomnożyć macierz A przez B.

Ogólna zasada mnożenia macierzy jest następująca: aby uzyskać wartość komórki, która znajduje się w  $m$ -tym wierszu i  $n$ -tej kolumnie wynikowej macierzy, musisz wziąć wartości z  $m$ -tego wiersza lewej macierzy i  $n$ -tej kolumny prawej macierzy, przemnożyć kolejne elementy przez siebie i je zsumować. Aby uzyskać wartość pierwszej komórki (wiersz 1., kolumna 1.), musisz wziąć wartości z pierwszego wiersza lewej macierzy i pierwszej kolumny prawej macierzy, pomnożyć ich kolejne elementy i dodać wszystkie cząstkowe iloczyny. Operacja jest przedstawiona na rysunku 8.8.

$$\begin{bmatrix} 1 & 0 \\ 3 & 2 \end{bmatrix} \cdot \begin{bmatrix} 5 & 4 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 5 & \\ & \end{bmatrix} \quad 1 \times 5 + 0 \times 2 = 5$$

A                      B

**Rysunek 8.8.** Obliczenie pierwszego elementu wynikowej macierzy

W przypadku elementu znajdującego się w pierwszym wierszu i drugiej kolumnie musisz skorzystać z pierwszego wiersza lewej macierzy i drugiej kolumny prawej macierzy, a następnie wykonać to samo: pomnożyć kolejne elementy przez siebie i dodać cząstkowe iloczyny. Rysunek 8.9 przedstawia przebieg tej operacji.

$$\begin{bmatrix} 1 & 0 \\ 3 & 2 \end{bmatrix} \cdot \begin{bmatrix} 5 & 4 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 4 \\ 19 & 14 \end{bmatrix} \quad 1 \times 4 + 0 \times 1 = 4$$

A                      B

**Rysunek 8.9.** Następnym krokiem mnożenia — drugi element w pierwszym wierszu

Rysunek 8.10 przedstawia obliczenia niezbędne od uzyskania dwóch pozostałych komórek wynikowej macierzy.

$$\begin{bmatrix} 1 & 0 \\ 3 & 2 \end{bmatrix} \cdot \begin{bmatrix} 5 & 4 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 4 \\ 19 & 14 \end{bmatrix} \quad 3 \times 5 + 2 \times 2 = 19$$

A                      B

$$\begin{bmatrix} 1 & 0 \\ 3 & 2 \end{bmatrix} \cdot \begin{bmatrix} 5 & 4 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 4 \\ 19 & 14 \end{bmatrix} \quad 3 \times 4 + 2 \times 1 = 14$$

A                      B

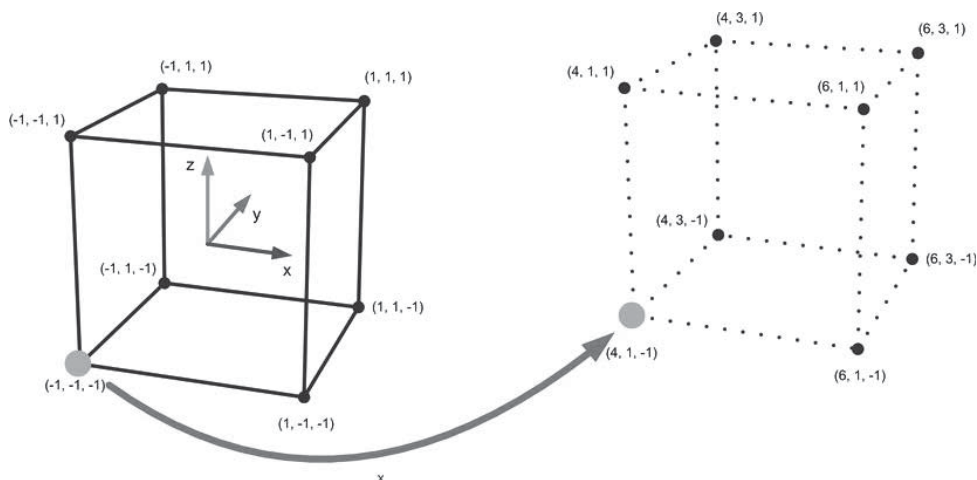
**Rysunek 8.10.** Ostateczny wynik mnożenia macierzy

Wreszcie dochodzimy do długo wyczekiwanej funkcjonalności. Aby zastosować przekształcenie opisane przez macierz, musimy pomnożyć współrzędne punktu przez macierz. Uzyskasz w ten sposób nowe współrzędne punktu — *po* transformacji. Jeśli tę samą transformację zastosujesz wobec wszystkich wierzchołków sześcianu, przekształcisz cały sześcian; w ten sposób możesz obracać, skalować lub przesuwać całe bryły. Teraz możemy omówić sam mechanizm nieco dokładniej. Przede wszystkim musisz zapisać współrzędne punktu w formie macierzy:  $[x, y, z, 1]$ . Trzeba dodać współrzędną 1, aby było możliwe przeprowadzenie mnożenia macierzy. Zobaczmy, co się stanie, gdy weźmiemy pierwszy wierzchołek sześcianu i zastosujemy do niego drugą z przykładowych macierzy przekształceń. Rysunek 8.11 przedstawia efekt mnożenia.

$$\begin{bmatrix} -1 & -1 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 5 & 2 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 1 & -1 & 1 \end{bmatrix}$$

**Rysunek 8.11.** Stosowanie przekształceń do współrzędnych wierzchołków

Punkt został przesunięty o pięć jednostek na osi  $X$  i o dwie na osi  $Y$ . Dokładnie to chcieliśmy osiągnąć! Jeśli powtórzymy tę operację dla wszystkich wierzchołków sześcianu, zostaną one przesunięte w odpowiednie miejsca. Rysunek 8.12 przedstawia zasadę działania mechanizmu. Wykropkowane krawędzie oznaczają położenie sześcianu w momencie stosowania przekształcenia do każdego wierzchołka. Podświetlony punkt oznacza punkt, który został przekształcony.



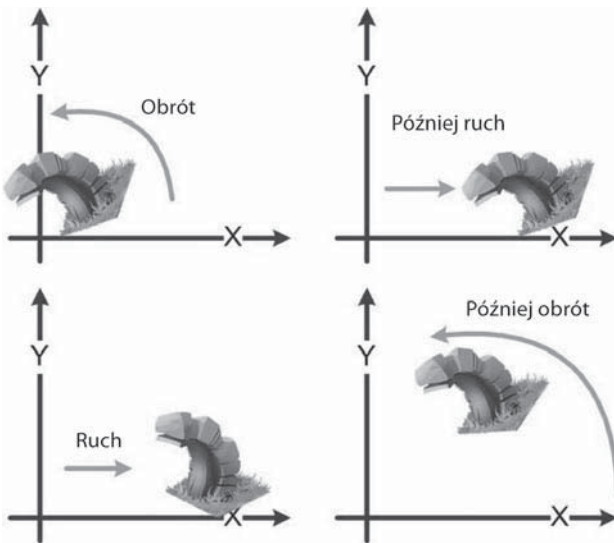
**Rysunek 8.12.** Przesunięcie sześcianu. Po przesunięciu wszystkich wierzchołków osiągniemy przesunięcie całego sześcianu

No cóż, nie jest to może porywający efekt, ponieważ mogliśmy po prostu dodać wartości 5 i 2 do odpowiednich współrzędnych bez zajmowania się macierzami. Prawdziwe możliwości macierzy ujawniają się jednak, gdy musimy wykonać szereg przekształceń na zbiorze punktów. Jeśli musisz wykonać np. dziesięć różnych przekształceń (jedno na drugim), a Twój model ma wiele wierzchołków, musisz albo zastosować przekształcenia dla modelu jedno po drugim, albo utworzyć pojedynczą macierz, która implementuje wszystkie przekształcenia, i zastosować tylko ją. Zgadnij, które podejście jest lepsze.

- 
- **Uwaga** Ta skromna jedynka, którą dodaliśmy jako czwarty element współrzędnych świata, nie znajduje się tam tylko dla formalności (aby mnożenie było w ogóle możliwe). Silniki 3D reprezentują punkty w nieco innym układzie współrzędnych, zwanym układem współrzędnych jednorodnych (ang. *homogeneous coordinates system*). Czemu są one używane w grafice komputerowej? Ponieważ stosowanie tego rodzaju współrzędnych znacznie ułatwia wykonywanie rzutów. Bardziej szczegółowe informacje na ten temat znajdziesz z łatwością w internecie lub literaturze matematycznej, jednak z pewnością nie będziesz potrzebował niczego więcej poza omówionymi w tej książce mechanizmami. Mimo to zachęcam do poszerzenia wiedzy na ten temat, gdy tylko dobrze zrozumiesz koncepty przedstawione do tej pory.
- 

Utworzenie macierzy, która łączy wiele przekształceń w jedno, jest niezwykle proste — wystarczy pomnożyć macierze, które określają pojedyncze transformacje. Jeśli pomnożysz macierz, która przesuwa wierzchołek w prawo o jedną jednostkę, przez macierz, która przesuwa wierzchołek o jedną jednostkę w górę, a następnie przez macierz, która obraca wierzchołek wokół osi  $X$ , uzyskasz macierz wykonującą wszystkie trzy operacje: przesuwać obiekt w górę i w prawo, a także obracającą model.

Operując na macierzach, musimy pamiętać o dwóch kwestiach. Po pierwsze, kolejność wykonywanych działań mnożenia ma znaczenie. Jeśli przeprowadzasz operację na pojedynczych liczbach, nie ma znaczenia, czy wykonasz operację  $5 \times 3$ , czy  $3 \times 5$  — w przypadku macierzy tak jednak nie jest. Wpływ kolejności przekształceń na efekt końcowy został przedstawiony na rysunku 8.13.



**Rysunek 8.13.** Zmiana kolejności przekształceń ma wpływ na efekt końcowy

Trzeba również pamiętać o tym, że przekształcenia są stosowane w odwrotnej kolejności, tj. ostatnie wprowadzone przekształcenie zostanie zastosowane wobec modelu jako pierwsze. Na początku jest to nieco mylące, jednak na pewno się do tego przyzwyczaisz.

Drugą kwestią, o której trzeba pamiętać, jest element neutralny. Podobnie jak pomnożenie liczby przez jeden nie zmienia wyniku mnożenia, tak pomnożenie macierzy przez *macierz jednostkową* nie zmieni danej macierzy — niezależnie od tego, czy macierz znajdzie się po lewej, czy po prawej stronie mnożenia. Macierz jednostkowa przedstawia brak jakichkolwiek przekształceń (rysunek 8.14).

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**Rysunek 8.14.** Macierz jednostkowa opisująca stan bez przekształceń

Na zakończenie na rysunku 8.15 przedstawiamy krótką listę najczęściej używanych macierzy przekształceń. Nie bój się sinusów i cosinusów pojawiających się w macierzach obrotu — nie musisz zagłębiać się w szczegóły przekształceń punktów.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x & y & z & 1 \end{bmatrix}$$

Translacja

$$\begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Skalowanie

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Obrót wokół osi X

$$\begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Obrót wokół osi Y

$$\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Obrót wokół osi Z

Rysunek 8.15. Różne rodzaje przekształceń przedstawione w formie macierzy

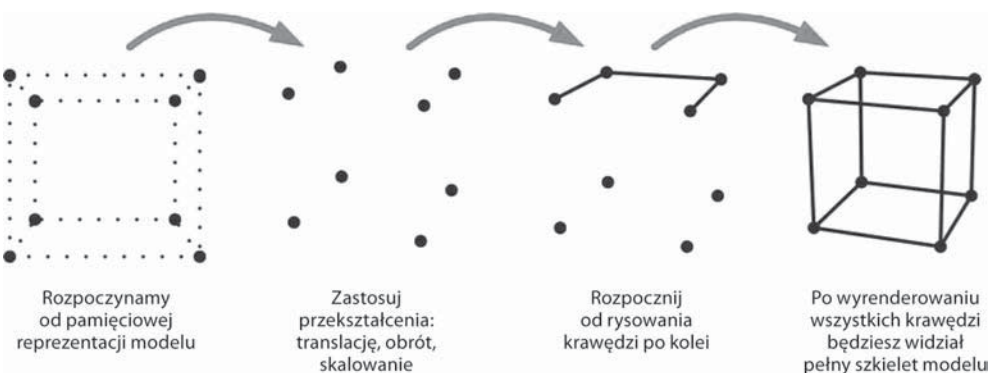
## Implementacja przekształceń

Jesteśmy wreszcie gotowi do utworzenia pierwszej wersji naszego „szczęśliwego” silnika. Na początek dobre informacje. Wszystkie operacje na macierzach wykona za nas biblioteka `gl-matrix` (<https://github.com/toji/gl-matrix>). Dzięki Brandonowi Jonesowi możemy skorzystać z doskonałego i bardzo szybkiego narzędzia. Pobierz plik `gl-matrix.js` lub `gl-matrix-min.js` i dodaj go do projektu. Zgodnie z nazwą biblioteka jest zazwyczaj używana razem z biblioteką WebGL, jednak możemy ją bez problemów wykorzystać także w innych celach.

Przekształcenie szkieletu sześcianu wymaga wykonania dwóch kroków:

1. Przekształcenia wszystkich wierzchołków (chcemy je obracać).
2. Narysowania linii pomiędzy wszystkimi wierzchołkami, pomiędzy którymi zostały zdefiniowane krawędzie.

Kolejne kroki są przedstawione na rysunku 8.16.



Rysunek 8.16. Kolejne kroki niezbędne do wyrenderowania szkieletu

Zacznijemy od pierwszej operacji. Nasz sześcian musimy dwukrotnie przekształcić — obrócić wokół osi  $X$  i wokół osi  $Y$  (możesz zastosować także inne przekształcenia). Biblioteka *gl-matrix.js* udostępnia wiele metod, które ukrywają szczegóły implementacji w ramach biblioteki. Kod przekształceń został pokazany na listingu 8.3.

**Listing 8.3.** Tworzenie macierzy przekształceń

```
var modelView = mat4.create();
mat4.identity(modelView);
mat4.rotateX(modelView, xRot);
mat4.rotateY(modelView, yRot);
```

Na początku tworzymy pustą macierz. Jest ona wypełniona zerami, dlatego musimy ją zainicjalizować — w tym celu korzystamy z macierzy jednostkowej; w przeciwnym razie wszystkie przekształcenia dawałyby wynik  $(0, 0, 0)$ . W trzecim wierszu dodajemy obrót wokół osi  $X$ . Jest to równoważne utworzeniu macierzy obrotu i pomnożeniu przez nią macierzy jednostkowej. W ostatnim wierszu kończymy przygotowania — w tym momencie jesteśmy gotowi do obrotu sześcianu.

- 
- **Uwaga** Cemu zmienna macierzy nosi nazwę `modelView`? Taka właśnie nazwa jest używana w OpenGL i oznacza ona... widok modelu, czyli wszystkie przekształcenia, które są stosowane wobec wierzchołków modelu. Cemu nie zastosujemy po prostu nazwy „matrix”? Ponieważ w trakcie całego procesu korzystamy również z innych macierzy — w dalszej części rozdziału będziemy się zajmować np. macierzą rzutu (projekcji).
- 

Proces obrotu jest bardzo prosty: stosujemy macierz po kolei do wszystkich wierzchołków. Otrzymujemy w ten sposób przekształcone współrzędne — aby wyświetlić je na ekranie, ignorujemy po prostu współrzędną  $z$ . Musimy także przeskalować sześcian, aby mieścił się na ekranie. Rozmiar sześcianu to jedynie 2 (od  $-1$  do  $1$ ) — jeśli spróbujemy go wyświetlić na ekranie, będzie on niezwykle mały. Na listingu 8.4 znajdziesz kod odpowiedzialny za obliczanie współrzędnych projekcji.

**Listing 8.4.** Obliczanie współrzędnych projekcji dla wszystkich punktów

```
// Dobrze jest utworzyć indeksy, które będą odpowiadać indeksom w tablicy.
var X = 0;
var Y = 1;
var Z = 2;

// Współrzędne ekranowe.
var points = [];

// Współczynnik skalowania — w tym miejscu możesz wpisać wybraną wartość.
var scaleFactor = canvas.width/8;

for (var i = 0; i < vertices.length; i++) {
  // Przekształć punkt do współrzędnych jednorodnych — przygotuj się do mnożenia.
  var point = [vertices[i][X], vertices[i][Y], vertices[i][Z], 1];

  // Zastosuj przekształcenie.
  mat4.multiplyVec4(modelView, point);

  // Dodaj obliczone współrzędne do tablicy.
  points[i] = [
    Math.round(canvas.width/2 + scaleFactor*point[X]),
    Math.round(canvas.height/2 - scaleFactor*point[Y]);
  ]
}
```

- **Uwaga** Rozmiar wynosi dwa? Dwa co? Piksele, cale, metry, melony? W grafice trójwymiarowej nie ma to znaczenia. To od Ciebie zależy, jaką jednostkę przyjmiesz. Musisz jedynie zachować spójność w ramach całego projektu. Jeśli Twoje „dwa” będzie oznaczać „dwa metry”, musisz o tym pamiętać — inaczej może się zdarzyć, że utworzysz model królika, który będzie miał wymiary małego samochodu. Nie jest to jednak tak naprawdę problem — zawsze możesz wprowadzić dodatkowe skalowanie. W tym rozdziale we wszystkich odwołaniach do rozmiarów stosuję słowo „jednostki”.

Jak widać, obliczenia nie stanowią problemu, gdy tylko przyzwyczaisz się do korzystania z macierzy.

Na zakończenie musimy narysować krawędzie. Dysponujemy tablicą krawędzi i tablicą przekształceń punktów. Łącząc obie tablice, możemy otrzymać obracający się sześcian (nie zapomnij zaktualizować wartości obrotu). Listing 8.5 przedstawia pełny kod dla tego projektu. Spróbuj uruchomić go na swoim telefonie lub w przeglądarce stacjonarnej.

**Listing 8.5.** Pełny kod źródłowy dla dema obracającego się sześcianu

```
<script src="js/utils.js"></script>
<script src="js/gl-matrix-min.js"></script>

<script>

    var canvas = null;
    var ctx = null;

    var X = 0;
    var Y = 1;
    var Z = 2;

    var vertices = [
        [-1, -1, -1], [-1, -1, 1], [-1, 1, -1], [-1, 1, 1],
        [1, -1, -1], [1, -1, 1], [1, 1, -1], [1, 1, 1]
    ];

    var edges = [
        [0, 1], [0, 2], [0, 4], [1, 3],
        [1, 5], [2, 3], [2, 6], [3, 7],
        [4, 5], [4, 6], [5, 7], [6, 7]
    ];

    var xRot = 0;
    var yRot = 0;

    function init() {
        canvas = initFullScreenCanvas("mainCanvas");
        ctx = canvas.getContext("2d");
        animate(0);
    }

    function animate(t) {
        ctx.clearRect(0, 0, canvas.width, canvas.height);
        renderCube();
        requestAnimationFrame(arguments.callee);
    }

    function renderCube() {

        var modelView = mat4.create();
```



```

mat4.identity(modelView); // Ustaw macierz jednostkową
mat4.rotateX(modelView, xRot);
mat4.rotateY(modelView, yRot);

var points = [];
var scaleFactor = canvas.width/8;

for (var i = 0; i < vertices.length; i++) {
    var point = [vertices[i][X], vertices[i][Y], vertices[i][Z], 1];
    mat4.multiplyVec4(modelView, point);

    points[i] = [
        Math.round(canvas.width/2 + scaleFactor*point[X]),
        Math.round(canvas.height/2 + scaleFactor*point[Y]);
    ]
}

ctx.strokeStyle = "black";

// Narysuj szkielet
ctx.beginPath();
for (i = 0; i < edges.length; i++) {
    ctx.moveTo(points[ edges[i][0] ][X], points[ edges[i][0] ][Y]);
    ctx.lineTo(points[ edges[i][1] ][X], points[ edges[i][1] ][Y]);
}

ctx.stroke();
ctx.closePath();
xRot += 0.01;
yRot += 0.01;
}

function initFullScreenCanvas(canvasId) {
    var canvas = document.getElementById(canvasId);
    resizeCanvas(canvas);
    window.addEventListener("resize", function() {
        resizeCanvas(canvas);
    });
    return canvas;
}

function resizeCanvas(canvas) {
    canvas.width = document.width || document.body.clientWidth;
    canvas.height = document.height || document.body.clientHeight;
}
</script>

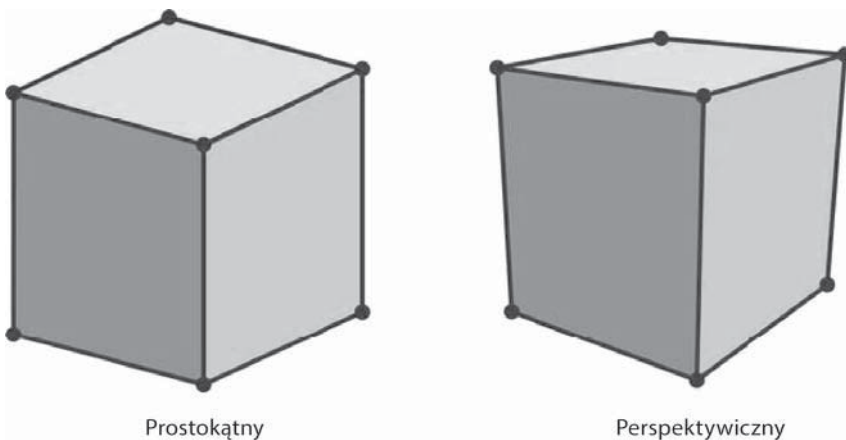
```

Po uruchomieniu dema przekonasz się, że sześcian nie wygląda naturalnie: tak naprawdę w ogóle nie wygląda on jak prawdziwy sześcian. Problem w tym, że skorzystaliśmy z rzutu prostokątnego — w prawdziwym świecie sytuacja wygląda inaczej. Pełny kod źródłowy z demem do tego rozdziału jest dostępny w pliku *01.spinning\_cube.html* wraz z innymi zasobami.

## Rzuty

W prawdziwym świecie obiekty, które znajdują się dalej, są mniejsze od obiektów znajdujących się bliżej. W naszym programie nie zastosowaliśmy się do tej zasady — to właśnie dlatego sześcian nie wygląda dobrze. Jak mogłeś się przekonać w kodzie, pominieliśmy po prostu wartość  $z$ , przez co nie uczestniczy ona wcale w obliczaniu rzutu.

W naszym demie korzystamy z rzutu *prostokątnego*, w którym podczas renderowania modeli 3D nie stosujemy żadnej perspektywy. W przypadku takiego rzutu samochód znajdujący się pięć metrów od aparatu ma taki sam rozmiar jak samochód znajdujący się dwadzieścia metrów dalej. Nie jest to naturalny widok. *Rzut perspektywiczny* funkcjonuje bardziej naturalnie: obiekty znajdujące się dalej są mniejsze niż te znajdujące się bliżej kamery. Rysunek 8.17 przedstawia różnice pomiędzy obydwoma rzutami.



**Rysunek 8.17.** Obraz po lewej stronie przedstawia rzut prostokątny, po prawej — rzut perspektywiczny

Jak widać, obraz po prawej stronie wygląda znacznie bardziej realistycznie niż ten po lewej. Jak możemy poprawić nasz silnik, aby perspektywa również była brana pod uwagę? Oczywiście musimy wprowadzić kolejną macierz.

Nie będę się zagłębiał w szczegóły matematyczne implementacji macierzy projekcji. Aby tworzyć świetne gry 3D, musisz o niej wiedzieć tylko dwie rzeczy. Po pierwsze, jeśli pomnożysz wynik przekształcenia `modelView` przez tę macierz, uzyskasz przekształcenie rzutu, które weźmie projekcję pod uwagę. Po drugie, Brandon to naprawdę fajny gość, który dołączył do swojej biblioteki mechanizm tworzenia macierzy perspektywy po podaniu parametrów przyjaznych dla użytkownika (listing 8.6).

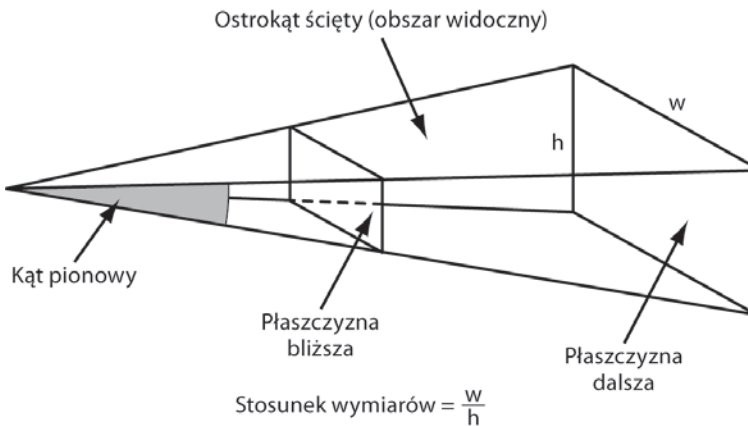
**Listing 8.6.** Tworzenie macierzy projekcji

```
var persp = mat4.create();
mat4.perspective(45, canvas.width/canvas.height, 0.1, 100, persp);
```

Wspomnianymi parametrami są pionowy kąt widoku, stosunek wymiarów (ang. *aspect ratio*) oraz odległość od bliższej i dalszej płaszczyzny. Płaszczyzny bliższa i dalsza określają granice sceny. Dzięki nim możesz stwierdzić, że chcesz widzieć tylko obiekty znajdujące się w odległości od 0,1 do 100 jednostek od kamery. Ostatni parametr określa macierz, która jest inicjalizowana przy użyciu tych wartości (listing 8.7). Rysunek 8.18 przedstawia wpływ parametrów perspektywy na końcowy efekt.

**Listing 8.7.** Stosowanie macierzy projekcji

```
for (var i = 0; i < vertices.length; i++) {
    var point = [vertices[i][X], vertices[i][Y], vertices[i][Z], 1];
    mat4.multiplyVec4(modelView, point);
    mat4.multiplyVec4(persp, point);
    ...
}
```



**Rysunek 8.18.** Określenie granic sceny przy użyciu perspektywy

Współrzędne otrzymane w wyniku stosowania macierzy perspektywy muszą zostać znormalizowane przed narysowaniem wierzchołków na scenie. Mówiąc wprost, przekształcamy współrzędne jednorodnie (te zawierające jeden dodatkowy parametr) na zwykłe współrzędne ekranowe zawierające dwa elementy:  $x$  i  $y$  (listing 8.8).

**Listing 8.8.** Normalizowanie współrzędnych

```
var ndcPoint = [
    point[X]/point[W],
    point[Y]/point[W]
];
```

Powstałe współrzędne  $x$  i  $y$  mieszczą się w zakresie od  $-1$  do  $1$ . „NDC” w zmiennej `ndcPoint` oznacza znormalizowane współrzędne urządzenia (ang. *Normalized Device Coordinates*). Teraz musimy jedynie „rozciągnąć” je na całe płótno, co pokazano na listingu 8.9. Pełny kod dema z rzutem perspektywicznym jest dostępny w pliku `02.perspective.html` dołączonym do kodów źródłowych do tego rozdziału.

**Listing 8.9.** Translacja współrzędnych ekranowych

```
points[i] = [
    Math.round(canvas.width/2*(1 + ndcPoint[0])),
    Math.round(canvas.height/2*(1 - ndcPoint[1]));
```

Listing 8.10 przedstawia pełny kod funkcji `renderCube()`, która obsługuje perspektywę (nowy kod jest pogrubiony).

**Listing 8.10.** Ostateczna wersja funkcji `renderCube()`

```
function renderCube() {

    var modelView = mat4.create();
    mat4.identity(modelView); // Macierz jednostkowa
    mat4.translate(modelView, [0, 0, -10]);
    mat4.rotateX(modelView, xRot);
    mat4.rotateY(modelView, yRot);

    var persp = mat4.create();
    mat4.perspective(45, canvas.width/canvas.height, 0.1, 100, persp);
```

```

var points = [];

for (var i = 0; i < vertices.length; i++) {
    var point = [vertices[i][X], vertices[i][Y], vertices[i][Z], 1];
    mat4.multiplyVec4(modelView, point);
    mat4.multiplyVec4(persp, point);

    var ndcPoint = [
        point[X]/point[W], // W = 3, położenie czwartej współrzędnej
        point[Y]/point[W] // w tablicy
    ];

    points[i] = [
        Math.round(canvas.width/2*(1 + ndcPoint[0])),
        Math.round(canvas.height/2*(1 - ndcPoint[1]))];
}

ctx.strokeStyle = "black";

// Narysuj szkielet
ctx.beginPath();
for (i = 0; i < edges.length; i++) {
    ctx.moveTo(points[ edges[i][0] ][X], points[ edges[i][0] ][Y]);
    ctx.lineTo(points[ edges[i][1] ][X], points[ edges[i][1] ][Y]);
}

ctx.stroke();
ctx.closePath();
xRot += 0.01;
yRot += 0.01;
}

```

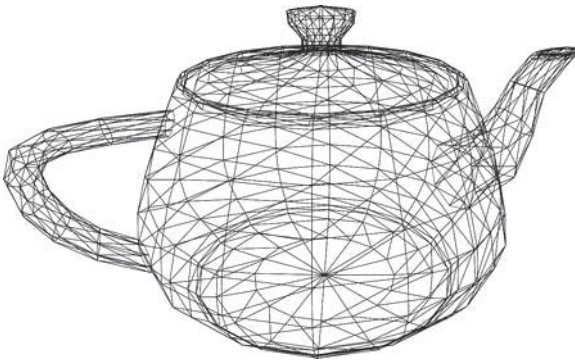
Świetnie! Utworzyłeś bardzo prosty silnik 3D! To naprawdę duże osiągnięcie. I wiesz co? Właśnie zduplikowaliśmy fragment mechanizmu renderującego biblioteki OpenGL — serce tego API. Przeanalizujmy, co właściwie udało nam się osiągnąć.

1. Skorzystaliśmy ze współrzędnych wierzchołków i krawędzi jako danych wejściowych.
2. Zastosowaliśmy przekształcenia wobec wierzchołków, obracając model wokół dwóch osi.
3. Zastosowaliśmy macierz perspektywy na wyniku poprzedniej operacji.
4. Znormalizowaliśmy współrzędne.
5. Przekształciliśmy współrzędne źródłowych wierzchołków na współrzędne płótna 2D.
6. Narysowaliśmy krawędzie obracającego się sześcianu.

Oczywiście cały proces w bibliotece WebGL jest bardziej skomplikowany, jednak jego główne części — przekształcenie wierzchołków, zastosowanie tablic, perspektywy i inne omówione w tym rozdziale zagadnienia — pozostają identyczne. Teraz możemy powiedzieć, że rozumiemy matematyczne podstawy kryjące się za operacjami wykonywanymi w WebGL.

Na koniec pozostaje nam jedno pytanie. Czy jesteśmy w stanie rysować cokolwiek poza prostymi sześcianami, prostokątami i kołami? Oczywiście. Rysunek 8.19 przedstawia bardzo złożony model, który udało mi się znaleźć w internecie. Nie będę wklejał teraz współrzędnych kilkuset wierzchołków, jednak cały kod możesz znaleźć w źródłach dołączonych do tego rozdziału. Rysunek 8.19 przedstawia dzbanek.

Pełny kod dzbanka jest dostępny w pliku *03.teapot.html*.



*Rysunek 8.19. Dzbanek do herbaty wyrenderowany przy użyciu tego samego mechanizmu co sześcian*

## Podsumowanie

W tym rozdziale skupiliśmy się całkowicie na zrozumieniu podstaw teoretycznych odpowiedzialnych za działanie silników 3D. Obliczenie położenia i zastosowanie perspektywy stanowią jedynie pierwszy krok w tworzeniu prawdziwego silnika, który renderuje kolejne ramki wewnątrz gry. Następne kroki są bardziej skomplikowane: usuwanie wielokątów, które nie są widoczne, aby zaoszczędzić czas, zastosowanie tekstur do ścian modeli, obliczanie efektów oświetleniowych itd. Silniki 3D zazwyczaj wykorzystują znacznie bardziej skomplikowane wzory i zależności matematyczne w porównaniu do silników 2D, a do tego silniki trójwymiarowe cały czas się zmieniają. Programiści i matematycy pracują wspólnie w celu uzyskania jeszcze lepszych algorytmów, które pozwolą na renderowanie coraz bardziej rzeczywistych obrazów.

Nawet najbardziej skomplikowana matematyka trójwymiarowa zaczyna się od współrzędnych wierzchołków i obliczeń przekształceń modelu. Jeśli nie pracowałeś nigdy z silnikiem 3D, to teraz możesz się pochwalić, że jesteś jednym z nielicznych programistów, którzy rozumieją, co się dzieje w ramach tego typu silników.

W tym rozdziale utworzyliśmy najprostsze możliwe demo 3D — obracający się sześcian. Podczas tego kursu nauczyliśmy się, jak:

- reprezentować model 3D w pamięci,
- korzystać z macierzy do przemieszczania, obracania i skalowania modeli,
- renderować szkielet,
- implementować rzut perspektywiczny w ramach sceny.

Teraz jesteśmy gotowi, aby zająć się bardziej zaawansowanym tematem — WebGL API.



# Skorowidz

## A

ADB, Android Debug Bridge, 387  
adresy URL danych, 125, 128  
AI, Artificial Intelligence, 415  
Ajax, 370  
akcesor modyfikujący, 151  
aktualizacja  
aplikacji, 479  
  planszy, 108  
  prototypu, 46  
  shaderów, 321  
  treści, 378  
algorytm  
  A\*, 424–427  
  brudnych prostokątów, 259  
  wykonania ruchu, 403  
aliasing, 61  
aliasy, 121  
animacja, 141, 143  
  na płótnie, 136  
  w Crafty, 450  
animowany rycerz, 118  
antialiasing, 61  
Apache Ant, 460  
Apache Cordova, 459  
API, Application Programming  
  Interface, 53  
  graficzne, 438  
  XMLHttpRequest, 372  
aplikacja DDMS, 477  
aplikacje  
  natywne, 157, 457  
  Node.js, 337  
Aptana Studio, 28, 31  
architektura  
  API obsługi zdarzeń, 172  
  gry Cztery kule, 98, 409  
  gry sieciowej, 389  
  modułów, 343  
arkusz sprite'ów, 118, 130–133

atrybuty  
  DOM, 158  
  shaderów, 314  
audio API, 486  
AVD, Android Virtual Device, 36

## B

bąbelkowanie, bubbling, 160  
bezpieczeństwo, 391  
biblioteka, 341  
  BinaryFile.js, 329  
  Connect, 347  
  jQuery, 372  
  jQuery UI Effects, 151  
  Node.js, 331–367  
  OpenGL, 293  
  Prototype.js, 372  
  RequireJS, 344  
  Socket.IO, 394  
  SoundManager, 490  
  SoundManager2, 487  
  WebGL, 298  
biblioteki graficzne, 439  
Blender, 328  
blokada przepływu serwera, 334  
blokowanie orientacji ekranu, 198  
błąd ReferenceError, 498  
błędy, 360, 374, 496  
breakpoint, 495  
brudne prostokąty, 257, 262, 265  
B-splajn, 71  
bufor pozaekranowy, 200, 205, 241

## C

canvas, 53, 55  
cel, destination, 130  
ciągły IFRAME, Forever IFRAME, 385  
Cordova, 460

CPU, Central Processing Unit, 282  
Crafty, 441, 445  
CSS, 93  
cykliczne wysyłanie żądań, 381  
czas pomiędzy ramkami, 142  
czujki, watches, 336

## D

dane binarne, 328, 376  
DDMS, 386, 477  
debugowanie, 38  
  aplikacji, 495–498  
  skryptów, 336  
  urządzeń mobilnych, 499  
  USB, 499  
  w IntelliJ, 336  
  w przeglądarce, 496  
  w WebStorm, 337  
  weinre, 503  
dekorowanie tła, 71  
deskryptor modułu, 344  
długa czynność, 42  
dodawanie  
  ekranu lobby, 397  
  gry do szkieletu HTML, 113  
  klasy IsometricTileLayer, 243  
  klasy ObjectLayer, 246  
  obiektów do klastrów, 250  
  obsługi dżojstika, 187  
  obsługi płótna, 201  
  obsługi rozgrywki, 402  
  stylu CSS, 401  
  warstwy do gry, 243  
dokumentacja  
  Connect, 364  
  Express, 356, 358  
  JSDoc, 102  
dokumentowanie kodu w Javie, 103  
DOM, Document Object Model, 54

domknięcie, closures, 340  
dostęp do  
  natywnego API, 459  
  parametrów zapytania, 355  
  prototypu, 45  
  stanu gry, 107  
dostępność zasobów, 42  
drzewo  
  przestrzenne, spatial tree, 245  
  decyzyjne, 432  
dyspozytor zdarzeń, 378  
dżbanek do herbaty, 299  
działanie  
  brudnych prostokątów, 258  
  map kafelkowych, 190  
  szablonów, 352  
dziedziczenie, 47, 49, 169  
  w encji gier, 442  
  w modułach, 343  
dziennik zdarzeń, 364  
dźwięki, 484  
dźwięki w grze, 493  
dżoistik wirtualny, 186

## E

Eclipse, 28, 458  
ECS, Entity Component System, 441  
edytor 3D, 328  
efekt paralaksy, 361  
eksport zmiennej, 342  
element  
  canvas, 53, 55  
  undefined, 360  
  xStats, 197  
elementy warstwy pośredniej, 358  
emulacja zablokowanego ekranu, 96  
emulator  
  systemu Android, 34  
  urządzenia mobilnego, 24  
encje, 41, 441

## F

File API, 377  
filtrowanie, 359  
Firebug, 302  
Flash Player, 487  
fonty rastrowe, 135  
format  
  Collada, 328  
  JAR, 340  
  JSON, 327, 344  
  MD2, 328  
  MP3, 489  
  PNG, 131  
  SVG, 55  
  Wavefront, 328  
fps, 196, 197

framework  
  Connect, 347  
  Crafty.js, 41  
  Express, 347, 360  
  Timing, 146  
funkcja  
  \_checkWinDirection(), 109  
  \_clearCanvas(), 112  
  \_drawMapRegion(), 237  
  \_getBoardRect(), 111  
  \_getGameState(), 109  
  \_getInputCoordinates(), 173  
  \_loadItem(), 123  
  \_onDownDomEvent(), 176  
  \_onItemLoaded(), 124  
  \_onMoveDomEvent(), 174  
  \_onObjectMove(), 253  
  \_onUpDomEvent(), 175  
  \_redrawOffscreen(), 202  
  \_reset(), 112  
  \_throwIfStarted(), 152  
  \_updateOffscreenBounds(), 204  
  addConnection(), 422  
  addEventListener(), 160  
  addObject(), 251  
  addUser(), 401  
  animate(), 137, 227  
  app.get(), 355  
  apply(), 227  
  arc(), 65  
  beginPath(), 64, 71  
  bind(), 124, 224–226  
  broadcast(), 396  
  call(), 228  
  canHandleMyself(), 362  
  clearInterval(), 141  
  clearTimeout(), 141  
  closePath(), 65, 71  
  configure, 366  
  console.log, 500  
  create(), 49  
  Crafty.e(), 446  
  Crafty.scene(), 445  
  createPattern(), 78  
  createServer(), 348  
  draw(), 201, 205, 241, 253  
  drawImage(), 129  
  drawPlayer(), 130  
  drawRect(), 62  
  drawScene(), 79, 303, 315  
  emit(), 396  
  extend(), 50  
  extends, 169  
  fillRect(), 62  
  foo(), 502  
  get, 122  
  getBounds(), 212  
  getContext(), 58

globalCompositeOperation, 183  
handleResize(), 111  
init(), 56, 79, 114  
load(), 123  
loadImage(), 79  
makeTurn, 108  
markDirty(), 263  
Math.round(), 132  
move(), 204  
moveTo(), 71  
next(), 359  
Object.defineProperty(), 401  
Object.freeze, 41  
Object.preventExtensions, 41  
Object.seal, 41  
onCanvasTouchStart(), 160  
onDone(), 124  
onItemLoaded(), 124  
overrideMimeType(), 376  
preventDefault(), 160  
render(), 355  
renderCube(), 297  
reorient(), 97  
repaint(), 97, 105  
requestAnimationFrame(), 139–  
  143, 149  
reset(), 107  
resizeCanvas(), 94, 114  
restore(), 85  
rotate(), 83  
save(), 85  
scale(), 83  
set(), 355  
setInterval(), 139  
setTile(), 242  
setTileAt(), 266  
setTimeout(), 140  
setTransform(), 83  
setViewportSize(), 204  
startGame(), 406  
stopPropagation(), 160  
stroke(), 71  
toDataURL, 126  
toString(), 351  
transform(), 83  
trigger(), 452  
wrapWithLogging, 501  
xhr.abort(), 374  
funkcje  
  gry Cztery kule, 98  
  haków do klasy Animatora, 152  
  klasy BoardRenderer, 105  
  klasy InputHandlerBase, 173  
  matematyczne, 228  
  nasłuchujące, 159  
  przejścia, 71, 151  
  w GLSL, 312  
funkcjonalność lobby, 395



**G**

generator zdarzeń, event emitter, 155  
 geometria sześcianu, 304  
 GLSL, 308  
 głośność dźwięku, 491  
 główny plik  
   HTML, 411  
   serwera, 405  
 gniazda, sockets, 332, 394  
 gniazdo Flash, Flash Socket, 385  
 GPU, Graphics Processing Unit, 282  
 gra, 53  
   Angry Birds, 133  
   Contra, 117  
   Cztery kule, 53, 91, 390  
   Fallout, 219  
   Quake II, 328  
   SimCity, 219  
   Warcraft 2, 190  
   XCOM, 219  
 gradienty, 73  
   liniowe, 73  
   radialne, 75  
 graf, 418  
   skierowany, 420  
   tworzony ręcznie, 430  
   ważony, 420  
 grafika  
   rastrowa, 118  
   wektorowa, 118  
 granice mapy, 238, 239  
 grupowanie transformacji, 82  
 gry  
   RPG, 189  
   sieciowe, 378, 390

**I**

IDE, Integrated Development Environment, 23, 27  
 identyfikator środowiska, 366  
 ikony, 468, 469  
 implementacja  
   algorytmu A\*, 427, 430  
   animacji, 142  
   dziedziczenia, 48  
   grafów, 420  
   klastrów obiektów, 248  
   klasy InputHandlerBase, 173  
   klasy WorldObjectRenderer, 208  
   logiki gry, 277  
   mapy kafelków, 191  
   przekształceń, 292  
   wirtualnego dżoystika, 187  
 informacje  
   o błędach, 365  
   o modułach, 346

inicjalizacja  
   biblioteki SoundManager2, 488  
   drzewa decyzyjnego, 434  
   dźwięku, 484  
   frameworku Crafty, 444  
   MouseEventHandler, 176  
   WebGL, 302  
 instalacja  
   Node.js, 335  
   SDK, 35  
 instalowanie frameworków, 345  
 instrukcja if-else, 311  
 IntelliJ Idea, 28, 337, 458  
 interakcja, 274  
 interfejs  
   CustomEvent, 165  
   użytkownika, 271  
 interpolacja liniowa, 322  
 interwał odpytywania, polling interval, 380  
 izometryczny silnik gry, 220

**J**

JAR, Java ARchive, 340  
 JDK, Java Development Kit, 23  
 język  
   GLSL, 309  
   JavaScript, 24, 38

**K**

kafelki, tiles, 189, 192  
   izometryczne, 215, 233  
   tła, 192  
 katalog  
   bin, 27  
   conf, 33  
   node\_modules, 344  
 klasa  
   Animator, 145–151  
   Array, 231  
   BoardModel, 99, 348–351, 403, 409  
   BoardRenderer, 99, 105, 409  
   DirtyRectangleManager, 260, 263, 266  
   EventEmitter, 166–169  
   Game, 99, 110, 224, 409  
   GameObject, 229, 262  
   GameSession, 407  
   ImageManager, 121, 126, 169  
   InputHandlerBase, 173, 175  
   IsometricTileLayer, 237, 241, 257, 266, 276  
   LayerManager, 222, 268  
   LobbyUsersList, 398  
   MapRenderer, 194, 198  
   MouseEventHandler, 172, 176  
   ObjectLayer, 245–247, 255, 265, 276  
   PropertyAnimator, 151  
   Rect, 228  
   RoundStateButton, 271  
   SpriteSheet, 134, 212  
   StaticImage, 246  
   TouchInputHandler, 172, 178  
   UILayer, 272, 276  
   WorldObject, 208  
   WorldObjectRenderer, 208  
 klastry, 245, 248  
 klasy gry Cztery kule, 99  
 klient serwera Socket.IO, 396  
 klucz  
   debugowania, 474  
   RSA, 474  
 kod shaderów, 313  
 kodowanie base64, 125  
 kolejność  
   mnożenia, 290  
   przekształceń, 83, 291  
   renderowania, 235  
 kolor, 73, 319  
 kolor piksela, 181  
 komentarze JSDoc, 102  
 kompensacja opóźnienia, 392  
 kompilacja shadera wierzchołków, 314  
 komponent  
   lotnika, 443  
   mechanika, 443  
   WebView, 458  
 komponenty gry Cztery kule, 392  
 komunikacja  
   klient-serwer, 394  
   w czasie rzeczywistym, 379  
 komunikat o błędzie, 360, 366  
 konfiguracja  
   Apache Ant, 460  
   Apache Cordova, 459  
   AVD, 36  
   serwera, 371  
   serwera nginx, 33  
   timerów, 139  
   WebGL, 302  
   zmiennych środowiskowych, 25  
 konfiguracje serwera, 366  
 konsola programisty, 477, 478  
 konstruktor, 48  
   grafu, 422  
   klasy  
     BoardModel, 106  
     BoardRenderer, 100  
     Game, 111  
     ImageManager, 122  
     InputHandlerBase, 173  
     MapRenderer, 194  
     PropertyAnimator, 152

kontekst, 57, 83  
 3D, 58  
 globalCompositeOperation, 182  
 kontrola przepływu zdarzeń, 160  
 kontrolki mobilne, 156  
 kontur, 72  
 konwencja wielbłądzia, camel case, 468  
 koszt drogi, 425  
 krawędzie, edges, 286  
 krawędzie sześcianu, 287  
 krzywe, 64, 68  
 krzywe Béziera, 69, 71  
 książka adresowa, 472  
 kwalifikator pamięciowy, 309

## L

łagi, 392  
 liczba klatek na sekundę, 142, 196  
 linia, 60, 64  
 lista  
 kontaktów, 473  
 krawędzi, 426  
 listy algorytmu A\*, 426  
 liście drzewa decyzyjnego, 434  
 lobby, 401  
 logika gry, 277, 414  
 logo, 446

## Ł

ładowanie  
 modeli, 327  
 obrazka, 124, 128  
 zasobów, 448  
 łańcuch prototypów, 44  
 łączenie  
 komponentów, 442  
 przekształceń, 290  
 łuk, 65, 256

## M

macierz, matrix, 287  
 jednostkowa, 291  
 modelView, 305  
 projection, 305  
 projekcji, 296  
 transformacji, 292  
 macierzowa reprezentacja grafu, 419  
 mapa  
 izometryczna, 216  
 kafelków, 189–191  
 maski obrazków, 181  
 mechanizm zdarzeń, 158  
 menedżer  
 pakietów NMP, 345  
 warstw, 268

metoda, *Patrz* funkcja  
 metody  
 przechwytywania zawartości  
 konsoli, 48  
 tworzenia grafu, 430  
 metryka Manhattan, 429  
 mikroruchy, 171  
 mnożenie macierzy, 288  
 model, 286  
 asynchroniczny, 335  
 DOM, 54, 157  
 high-polygon, 327  
 low-polygon, 327  
 rozszerzania obiektów, 50  
 wątków, 138  
 modele trójwymiarowe, 327  
 moduł Web Developer Tools, 337  
 moduły, modules, 341  
 modyfikacja  
 klasy ImageManager, 126  
 shaderów, 326  
 mostek urządzeń Androida, 499

## N

narzędzie  
 Ant, 458  
 Apache Cordova, 157, 459  
 DDMS, 386  
 DummyNet, 387  
 JavaDoc, 103  
 keytool, 474  
 MP3Loop, 490  
 weinre, 502  
 xStats, 196  
 nasłuchiwanie  
 własnych zdarzeń, 166  
 zdarzeń, 162, 452  
 natywne  
 API, 471  
 funkcjonalności, 472  
 nazwy zmiennych, 106  
 Node.js, 331–367  
 dziedziczenie, 343  
 klasa BoardModel, 349  
 moduły, 341  
 NPM, 345  
 sesje, 357  
 szablony, 352  
 udostępnianie plików, 351  
 warstwa pośrednia, 358  
 normalizowanie współrzędnych, 297  
 NPM, 345

## O

obiekt  
 \_dirtyRectangleManager, 264  
 BoardModel, 348

canvas, 53, 89  
 event, 158  
 itemCount, 123  
 płótna, 53  
 respond, 406  
 XMLHttpRequest, 372  
 zdarzenia, 158  
 obiektowy model dokumentu, 54  
 obiekty złożone, 255  
 obliczanie  
 długości odcinka, 283  
 liczby kul, 110  
 promienia pionka, 101  
 przesunięć gradientów, 101  
 współrzędnych projekcji, 293  
 obrót, 82  
 obrót sześcianu, 294  
 obsługa  
 animacji, 451  
 błędów, 120, 360, 362  
 błędów w XHR, 374  
 danych binarnych, 376  
 dziennika zdarzeń, 364  
 dźwięku, 483, 490  
 ekranów, 101  
 ekranów dotykowych, 162  
 ekranu, 111  
 fizyki gry, 133  
 kliknięć, 112  
 kolorów, 321  
 logiki, 404  
 płótna, 56  
 rozgrywki, 402  
 ruchu, 253  
 ruchu obiektów, 255  
 sesji, 357  
 sieci, 371, 467  
 Socket.IO, 395  
 warstw, 270  
 własnych zdarzeń, 166  
 zdarzeń, 158, 172, 275  
 zdarzeń serwera, 400  
 żądania HTTP, 373  
 obszar widoku, viewport, 93, 198, 238  
 oddalanie, zoom out, 283  
 odległość pomiędzy punktami, 417  
 odpytywanie  
 cykliczne, 380  
 długotrwałe, 381, 382  
 JSONP, 385  
 odświeżanie  
 bufora pozaekranowego, 241  
 płótna, 204  
 odtwarzanie  
 dźwięku, 486, 491  
 w pętli, 489  
 odwrotny Ajax, reverse Ajax, 378  
 ograniczenia Ajaksa, 370

okno DDMS, 387  
 OpenGL, 293, 301  
 OpenGL ES 2.0, 305  
 operacja  
   source-atop, 183  
   source-over, 182  
 operacje  
   wejściowe, user input, 155, 164  
   wejściowe zaawansowane, 179  
 opóźnienia sieci, 392  
 optymalizacja renderowania mapy, 203  
 optymalizacje, 213  
 orientacja ekranu, 96  
 oznaczanie brudnych prostokątów, 266

## P

pamięć podręczna, cache, 128  
 pamięć podręczna obiektów, 250  
 paradygmat komponentów encji, 441  
 parametry  
   kafelka, 235  
   sesji, 357  
 parsowanie wejścia użytkownika, 355  
 pasek stanu, 470  
 pętla  
   do-while, 312  
   for, 312  
   while, 312  
 plansza do gry, 87  
 plik  
   .bash\_profile, 26  
   1.skeleton.html, 56  
   AndroidManifest.xml, 465, 468  
   animal.js, 342  
   Animator.js, 147  
   BoardModel.js, 105, 404  
   BoardRenderer.js, 100  
   cordova-2.0.0.js, 472  
   error.ejs, 362  
   final.html, 86  
   FourBalls.java, 464, 470  
   Game.js, 113, 224  
   gl-matrix.js, 292  
   Graph.js, 423  
   hamster.js, 342  
   index.html, 195, 411, 413  
   InputHandler.js, 413  
   Level.js, 243  
   logo.png, 446  
   nginx.conf, 33  
   package.json, 344  
   Rect.js, 228  
   skeleton.html, 98  
   socket.io.js, 397  
   strings.xml, 468  
   transform-demo.html, 80  
   utils.js, 141, 231

weinre.jar, 502  
 world.js, 193  
 pliki  
   .bat, 458  
   .java, 464  
   APK, 478  
   z opisem modułu, 344  
 płótno, 53, 56  
 płótno pozaekranowe, 238  
 pływające  
   div-y, 55  
   ramki, floating frame, 370  
 pobieranie granic mapy, 239  
 podejmowanie decyzji, 432  
 podpisywanie aplikacji, 474  
 podścieżki, 71  
 podział  
   na klastry, 248  
   siatki, 240  
   sprite'ów na kategorie, 128  
 polecenie logcat, 499  
 położenie obiektu, 146  
 poprawianie adresów URL, 467  
 port 80, 33  
 porządek renderowania, 211  
 postaci niegrywalne, 485  
 potok, pipeline, 305  
 potok OpenGL ES 2.0, 305  
 powiadomienia push, 378  
 powierzchnie NURBS, 71  
 proces renderowania, 307  
 program Blender, 328  
 programowanie  
   po stronie serwera, 348  
   zorientowane obiektowo, 43  
 projekt  
   Cordova, 463  
   Socket.IO, 394  
   Three.js, 330  
   WANem, 387  
 propagowanie zdarzeń, 275, 278  
 prostokąt, 62  
 prototypy, 44  
 próg  
   brudnego obszaru, 260  
   ruchu, movement threshold, 171  
 przechwytywanie wejścia  
   użytkownika, 157  
 przeciągnij i upuść, 179  
 przeglądarki mobilne, 51  
 przekazywanie  
   danych koloru, 322  
   danych tekstur, 327  
   zdarzenia, 275  
 przekierowanie użytkownika, 160  
 przekształcanie  
   koloru, 184  
   sześciannu, 292

przekształcenia kontekstu, 85  
 przekształcanie, *Patrz* transformacja  
 przełączanie pomiędzy ekranami, 411  
 przemieszczanie rycerza, 451  
 przemieszczenie pocisku, 150  
 przepływ  
   sterowania, 445  
   w grze sieciowej, 404  
   w Node.js, 334  
 przestrzeń nazw, namespace, 338, 461  
 przestrzeń nazw com.juriy.math, 340  
 przestrzeń robocza, workspace, 31  
 przesunięcie, slide, 80, 156  
 przesunięcie sześciannu, 290  
 przesuwanie  
   mapy, 195  
   na stronie, 56  
 przetwarzanie żądania, 358  
 przezroczystość, 131  
 przybliżenie, zoom, 283  
 przygotowywanie tekstur, 325  
 przypisywanie koloru, 320  
 przyspieszenie, 146  
 przywracanie kontekstu stanu, 84  
 publikowanie  
   aplikacji, 473, 478  
   w serwisie Google Play, 476  
 punkt  
   kontrolny, 68  
   orientacyjny, way point, 417  
   początkowy gry, 223  
   zaczeplenia, anchor point, 133

## R

radiany, 66  
 raport o błędzie, 362  
 referencja do samej funkcji, 137  
 rejestracja w Google Play, 477  
 rejestrowanie  
   informacji, 500  
   zdarzeń, 158  
 renderer kodu HTML, 351  
 renderowanie  
   3D, 281, 283  
   animacji, 133  
   aplikacji, 315  
   kafelków, 232  
   kafelków map, 194  
   mapy, 195, 203  
   obiektów, 209, 211, 244  
   OpenGL, 305  
   pionków, 76  
   planszy, 100  
   programowe, 282  
   sceny, 98  
   sceny 3D, 284

renderowanie

- siatki kafelków, 194
- sprzętowe, 282
- stron internetowych, 351
- sześcianu, 313
- ścieżek, 63
- terenu, 449
- tła, 449
- trójkątów, 72
- warstw silnika, 221

REPL, read-eval-print loop, 344

resetowanie

- pamięci podręcznej, 250
- plótka pozaekranowego, 241
- stanu kontekstu, 84
- statusu przycisku, 177

rodzaje

- błędów, 374
- węzłów, 433

rozdzielczość emulatora, 36

rozmiar

- ekranu, 101
- obszaru widoku, 252
- plótka pozaekranowego, 201

rozszerzenia, extensions, 341

RPG, Role-Playing Games, 189

ruch

- kuli, 256
- obiektu, 254

rysowanie

- elementów gry, 67
- kafelków, 192, 195
- kół, 66
- krzywych, 68, 70
- linii, 59, 64
- łuków, 65
- maski obrazu, 183
- na płótnie, 57
- obiektów świata gry, 209
- obrazka, 129
- obszaru ograniczonego, 236
- pionowej linii, 60
- pojedynczych ramek, 134
- postaci, 492
- prostokąta, 62
- przy użyciu współrzędnych, 132
- siatki, 64
- sprite'ów, 135
- sześcianu, 315
- ścieżek, 64
- tła planszy, 63
- udekorowanego sprite'a, 236
- udekorowanych kafelków, 236
- w przeglądarce, 55
- widocznych kafelków, 199

rzut

- izometryczny, 215
- perspektywiczny, 296
- prostokątny, 296

rzutowanie promieni, 431

## S

scena, 287

SDK Manager, 35

SDK systemu Android, 34

Server-Sent Events, 379

serwer

- Ajax, 370
- Apache, 33
- dla gry, 347
- nginx, 33, 338
- stron internetowych, 24, 33

serwis

- Apache Incubator, 460
- Google Play, 477
- W3Techs, 33

sesje, 357

setter, 151

shader, 308

- fragmentów, 313, 321
- wierzchołków, vertex shader, 306, 321, 326

siatka

- izometryczna, 216
- kafelków, 194
- nawigacyjna, 431

sieć Wi-Fi, 34

silnik

- Crafty, 437, 441
- Engine X, 437
- ejs, 355
- Node.js, 50

silniki

- 3D, 285
- izometryczne, 219
- szablonów, 352
- gier, 437, 440
- trójwymiarowe, 281

skalowanie, 81, 119

sklepy, 459

składnia adresu URL, 125

skrypt

- build.xml, 462
- Node.js, 333

słowo kluczowe

- private, 39
- this, 225

sortowanie, 211

sortowanie obiektów, 212

specyfikacja

- ECMA, 49
- GLSL, 309, 313
- WebGL, 58

sprawdzanie

- pamięci podręcznej, 252
- typu urządzenia, 162
- warunków zwycięstwa, 110

sprite'y, 118

stała Math.PI, 66

stan kontekstu, 84

standard REST, 345, 356

stany

- gry, 274
- XHR, 373

stos

- warstwy pośredniej, 358
- wywołań, 338

stosowanie

- API, 459
- atrybutów shaderów, 314
- buforów, 307
- sesji, 357
- wzorców, 78, 79

strategia powtarzania, repetition strategy, 78

strona

- aplikacji, 479
- błędu, 361
- struktura projektu, 393, 462, 464
- strumieniowanie XHR, 383
- klient, 383
- serwer, 384

strumień, stream, 365

strumień do zapisu, 365

stuknięcie, tap, 156

styl background-image, 55

SVG, Scalable Vector Graphics, 55

symulowanie

- dźwojstka, 185
- złych warunków sieci, 387

system

- komponentów encji, 441, 444
- kontroli wersji, 344

szablon

- EJS, 352
- Jade, 353

szacunkowy koszt drogi, 425

sześcian

- w 3D, 286
- z nałożoną teksturą, 327

szkielet

- gry, 92
- strony HTML5, 56, 93, 95

sztuczna inteligencja, AI, 415

## Ś

ściany, faces, 286

ścieżka, 26

- systemowa PATH, 25
- wirtualna, 63

śledzenie

- ruchu obiektów, 254
- stanu przycisku, 176
- użytkowników, 400

środowisko  
 deweloperskie, 41  
 produkcyjne, 41

## T

tablica, 231  
 query, 355  
 targetTouches, 160  
 tablice typowane, 377  
 technologia  
 Ajax, 370  
 WebSocket, 371  
 tekstury, 322, 324  
 teren izometryczny, 221, 231, 247  
 testowanie  
 aplikacji Android, 462  
 projektu, 466  
 zdarzeń, 178  
 timery, 137, 143  
 transformacja  
 obrót, 82  
 skalowanie, 81  
 translacja, 80  
 translacja współrzędnych  
 ekranowych, 297  
 transporty, 386  
 trójkąt, 304, 306  
 tryb  
 LAN Game, 389  
 pełnoekranowy, 459, 470  
 portretowy, 97  
 REPL, 344  
 rysowania, 57  
 tutorial  
 Blendera, 328  
 DummyNet, 387  
 WebGL, 330  
 tworzenie  
 animacji, 136, 137  
 aplikacji natywnej, 461  
 aplikacji WebGL, 316, 317, 318,  
 319  
 aplikacji webowych, 347  
 buforów, 307  
 cyfrowego klucza, 474  
 grafu, 422  
 grafu wyszukiwania ścieżek, 430  
 gry Cztery kule, 91, 99  
 gry sieciowej, 389  
 gry w Crafty, 447  
 kodu w JavaScript, 38, 42  
 lobby, 399  
 macierzy, 290  
 macierzy projekcji, 296  
 macierzy przekształceń, 293  
 modułu, 341  
 natywnych aplikacji, 457

nowego AVD, 37  
 obiektów, 43  
 obrazka, 126  
 podścieżki, 71  
 pustego projektu, 461  
 silnika izometrycznego, 219  
 szablonu, 353  
 typ  
 Float32Array, 308  
 NumericDecision, 433  
 typy danych w GLSL, 310

## U

udostępnianie plików statycznych, 351  
 układ współrzędnych  
 biegunowy, 185  
 kartezjański, 58  
 unoszenie, hover, 156  
 uprawnienie  
 ACCESS\_NETWORK\_STATE,  
 472  
 INTERNET, 473  
 READ\_CONTACTS, 473  
 urządzenia dotykowe, 156  
 usługa Perfecto Mobile, 35  
 ustawianie  
 funkcji przejścia, 152  
 sceny, 445  
 ustawienia strony HTML, 55  
 usuwanie obiektu z klastra, 252

## W

W3C, 372  
 wachlarze trójkątne, 304  
 walidacja, 411  
 warstwa  
 kliencka, 408  
 obiektów, 257  
 pośrednia, middleware, 358  
 pośrednia errorHandler, 361  
 serwerowa, 404  
 session, 358  
 static, 359  
 wątki, 137  
 wczytywanie  
 danych binarnych, 329  
 dźwięków, 491  
 obrazka, 120  
 opóźnione sceny, 448  
 pliku w Node.js, 333  
 sprite'ów, 447  
 tekstury, 324  
 wielu obrazków, 121  
 Web Audio API, 485, 486  
 Web Module, moduł webowy, 28  
 WebGL, 281, 298–330  
 WebSocket, 371, 379

WebStorm, 337  
 węzły, nodes, 433  
 węzły drzewa decyzyjnego, 433  
 wiązanie shaderów, 313  
 widok  
 izometryczny, 214, 219  
 z lotu ptaka, 207, 215  
 wielodotyk, multitouch, 156  
 wielokąt, 327  
 wierzchołki, vertexes, 286  
 otwarte, 426  
 sześcianu, 286  
 wirtualne urządzenie, 36  
 własna właściwość, own property, 44  
 właściwości, 44  
 właściwości niewyliczalne, 401  
 właściwość  
 \_\_proto\_\_, 45  
 background-repeat, 78  
 canvas.height, 94  
 canvas.width, 94  
 constructor, 48  
 document.body.clientWidth, 95  
 document.width, 95  
 exports, 344  
 module.exports, 342  
 prototype, 45  
 readyState, 373  
 responseType, 377  
 socket, 401  
 src, 125  
 window.orientation, 96  
 współdzielenie logiki gry, 403  
 współrzędne  
 ekranowe, 232  
 siatki izometrycznej, 232  
 świata, 232  
 świata gry, 206  
 tekstury, 325  
 zdarzenia, 163  
 wtyczka, plug-in, 341  
 Firebug, 302  
 Flash, 489  
 wybór IDE, 28, 32  
 wyciek pamięci, memory leaks, 166  
 wydajność  
 renderowania, 198  
 rysowania, 131  
 wyjątek, exception, 160, 338  
 wykonywanie ruchów, 107  
 wykrywanie  
 dotknięć, 161  
 modułów, 344  
 wymuszanie orientacji, 96  
 wypełnienie, 63, 77  
 wyrażenie  
 arguments.callee, 137  
 foo, 502

wyrównanie, alignment, 458  
 wysyłanie e-maili, 363  
 wyszukiwanie ścieżek, 416, 424, 430  
 wyświetlanie  
   grafu, 423  
   planszy, 103, 350  
 wywołania  
   asynchroniczne, 41  
   synchroniczne, 41  
   zwrotne, 120, 169, 226, 408  
 wywoływanie  
   metody setTimeout, 139  
   XHR, 467  
   zdarzeń, 452  
 wyzwanie, challenge, 404  
 wzorce, patterns, 77  
 wzór na długość odcinka, 283

## X

XHR, XMLHttpRequest, 329, 370  
 XHR streaming, 383  
 XMLHttpRequest Level 2, 375

## Z

zadania  
   klienta, 391  
   serwera, 391  
 zakończenie gry, 407  
 zapętlenie dźwięku, 493  
 zapisywanie  
   dziennika, 365  
   informacji o błędach, 365  
   kontekstu stanu, 84  
 zarządzanie płótnem  
   pozaekranowym, 240  
 zasięg  
   globalny, 340  
   w JavaScript, 341  
 zasoby natywne, 471

zatrzymywanie  
   przeglądarki, 137  
   timerów, 141  
   zdarzeń, 175, 278  
 zbiór kluczy, keystore, 474  
 zdarzenia  
   down, 170  
   move, 171  
   myszy, 163  
   przeglądarki, 157  
   serwera, 400  
   up, 170  
   własne, 165, 170  
 zdarzenie, event, 160  
   \_onDownDomEvent, 174  
   \_onUpDomEvent, 175  
 connection, 396  
 deviceready, 472  
 finished, 166  
 loaded, 166  
 MouseDown, 453  
 onclick, 356  
 onerror, 120  
 onLoaded, 121  
 onProgress, 123, 124  
 orientationchange, 96  
 touchstart, 165, 453  
 zgłaszanie błędów, 360  
 zintegrowane środowisko  
   programistyczne, IDE, 23, 27  
 zmiana  
   kontekstu stanu, 84  
   obszaru widoku, 199  
   ogniskowej, 284  
   pikseli, 185  
   położenia, 252  
   położenia obrazka, 57  
   rozmiaru ekranu, 111  
   rozmiaru planszy, 111  
   rozmiaru płótna, 94

zmienna  
   ANDROID\_HOME, 38  
   modelView, 293  
   NODE\_DEV, 366  
   NODE\_PATH, 344  
 zmiennie  
   środowiskowe, 25, 38  
   typu varying, 322  
 znacznik  
   app\_name, 468  
   audio, 484, 485, 487  
   iframe, 370  
   img, 487  
   Meta Viewport, 93  
 znak  
   krzyżyka, 34  
   ucieczki, 26  
   ukośnika, 26  
 znaki \n, 26

## Ź

źródło, source, 130  
 źródło danych dla obrazka, 125

## Ż

żądania  
   GET, 356  
   HTTP, 373  
   POST, 356  
   uprawnień, 472  
   XHR, 373

# 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>



Potencjał HTML5, CSS3 i JavaScriptu pozwala na tworzenie niesamowitych gier i aplikacji. Dzięki tym technologiom powstało oprogramowanie działające w mobilnej przeglądarce internetowej. Jeżeli dodamy do tego moc współczesnych telefonów i tabletów działających z systemem Android, może się okazać, że to trio stanowi niezastąpiony zestaw narzędzi.

Dzięki tej książce rozwiniesz swoje umiejętności programistyczne. W trakcie lektury dowiesz się, jak najszybciej rozpocząć przygodę z grami dla platformy Android. W kolejnych rozdziałach nauczysz się korzystać z grafiki i animacji w przeglądarce, obsługiwać zdarzenia i operacje wykonywane przez użytkownika oraz stosować różne sposoby renderowania świata gry. Zbudujesz też własny silnik izometryczny oraz wykorzystasz WebGL do stworzenia zaawansowanej grafiki 3D. Dodatkowo zdobędziesz wiedzę na temat programowania sztucznej inteligencji oraz zapewniania komunikacji aplikacji z serwerem. W tej chwili tylko krok dzieli Cię od tworzenia gier w trybie multiplayer! Książka ta jest doskonałym i kompletnym źródłem informacji dla wszystkich osób chcących wykorzystać platformę Android i przeglądarkę internetową do pisania zaawansowanych i atrakcyjnych gier.

#### Stwórz grę:

- działającą na platformie Android
- wykorzystującą mistrzowskie trio: HTML5, CSS3 i JavaScript
- zawierającą zaawansowane elementy graficzne 3D
- dla wielu graczy
- i odnieść sukces!

**Niezastąpiony podręcznik dla każdego programisty tworzącego gry!**

**helion.pl**  
księgarnia  
internetowa

Nr katalogowy: 14664



Księgarnia internetowa  
<http://helion.pl>



Zamówienia telefoniczne:

**0 801 339900**



**0 601 339900**

Apress®



**Helion**

Sprawdź najnowsze promocje:

• <http://helion.pl/promocje>

Książki najchętniej czytane:

• <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

• <http://helion.pl/nowości>

Helion SA  
ul. Kościuszki 1c, 44-100 Gliwice  
tel.: 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
<http://helion.pl>

sięgnij po **WIECEJ**



KOD KORZYŚCI

ISBN 978-83-246-7401-5



9 788324 674015

Cena: 79,00 zł

Informatyka w najlepszym wydaniu