helion.pl

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

Flash MX. Vademecum profesjonalisty



Autorzy: Jody Keating, Fig Leaf Software
Tłumaczenie: Marek Binkowski, Marek Korbecki
ISBN: 83-7197-916-9
Tytuł oryginału: Inside Flash MX
Format: B5, stron: 618

Książka omawia niezwykłą technologię. Flash MX łączy piękno projektowania graficznego z wygodą programowania zorientowanego obiektowo, wprowadzając nową jakość do świata aplikacji internetowych. W książce tej zawarto wszystkie informacje pozwalające w pełni wykorzystać możliwości Flasha MX. Jest ona adresowana do dwóch grup odbiorców: artystów i programistów. Wiadomo, że nie każdy artysta chce zostać programistą, podobnie jak nie każdy programista posiada uzdolnienia artystyczne. Treść niniejszej książki będzie jednak użyteczna zarówno dla jednych, jak i drugich. Tworzenie za pomocą Flasha wymaga bowiem połączenia obu dziedzin, zaś w przypadku Flasha MX zasada ta ma jeszcze głębsze znaczenie niż dotychczas.

„Flash MX. Vademecum profesjonalisty" to kompletne źródło informacji, począwszy od wiadomości podstawowych, poprzez omówienie twórczych technik, aż po wyczerpujące omówienie języka ActionScript.

- Dowiedz się, jakie zmiany wprowadzono w sposobie pracy i funkcjonowania interfejsu Flasha
- Użyj nowych rozwiązań Named Anchors i Shared Objects, usprawniających współpracę Flasha z przeglądarką
- Wykorzystaj animacjach Flasha materiały wideo, korzystając z nowych możliwości importowania ścieżek wideo
- Skorzystaj z komponentów Flasha, by przyspieszyć proces tworzenia rozbudowanych aplikacji internetowych
- Użyj nowych poleceń ActionScript, umożliwiających rysowanie na obrazie za pomocą skryptów
- Zastosuj w filmie Flasha prawa fizyki tak, by obiekty poruszały się w naturalny sposób

Jody Keating jest dyrektorem pomocniczym do spraw mediów interaktywnych w firmie Fig Leaf Software (jednej z czołowych firm programistycznych intensywnie korzystających z technologii Flash) oraz ekspertem i instruktorem certyfikowanym przez firmę Macromedia. Nad książką pracował zespół specjalistów pod przewodnictwem Jody, każdy z nich podzielił się wiadomościami z dziedziny, w której się specjalizuje.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

O Autorach ... 13

Wprowadzenie ... 17

Rozdział 1. Zmiany w systemie pracy Flasha ... 21
Zmiany na listwie czasowej ... 21
Przybornik i panel Properties ... 24
Zmiany na panelach ... 26
Zmiany na panelu Color Mixer ... 28
Zmiany na panelu Actions ... 29
Krótka charakterystyka komponentów ... 32
Funkcja rozmieszczania obiektów na warstwach ... 33
Podsumowanie ... 34

Rozdział 2. Jak korzystać z biblioteki i bibliotek współużytkowanych ... 35
Korzystanie z funkcji liczenia elementów ... 36
Tworzenie nowych symboli, korzystanie z folderów i rozwiązywanie konfliktów nazw ... 37
Uaktualnianie elementów zgromadzonych w bibliotece ... 38
Przyłączanie i współużytkowanie bibliotek ... 40
Tworzenie i wykorzystanie symboli czcionek ... 46
Jak uchronić się przed przykrymi niespodziankami ... 48
Tworzenie własnych bibliotek stałych ... 49
Podsumowanie ... 49

Rozdział 3. Importowanie, wykorzystanie i optymalizacja elementów graficznych ... 51
Optymalizacja map bitowych ... 53
Konwertowanie map bitowych na obrazy wektorowe ... 56
Importowanie obrazów wektorowych ... 60
Importowanie obrazów utworzonych w innych programach ... 62
Importowanie obrazów utworzonych w programie Adobe Illustrator ... 62
Importowanie obrazów utworzonych w programie Adobe Photoshop ... 63
Importowanie obrazów utworzonych w programie Macromedia FreeHand ... 64
Importowanie plików przygotowanych w programie Macromedia Fireworks ... 66
Importowanie obrazów utworzonych w programie Toon Boom Studio ... 68
Łączenie bitmap z obrazami wektorowymi ... 70
Niewłaściwe połączenie bitmapa — obraz wektorowy ... 70
Monitorowanie obciążenia procesora ... 71
Czy ten plik można zapisać? ... 72
Bandwidth Profiler ... 74
Tworzenie raportu o wielkości pliku ... 78
Rozwiązywanie problemów za pomocą narzędzi diagnostycznych ... 79
Dzielenie dużego filmu na kilka mniejszych ... 81
Podsumowanie ... 82

Rozdział 4. Wykorzystanie dźwięku we Flashu ... 83
Teoria dźwięku ... 84
Dźwięk kontra obraz ... 85
Częstotliwość próbkowania i głębia bitowa ... 85
Zachowanie niewielkiej objętości plików: mowa a muzyka ... 86
Importowanie plików dźwiękowych do filmu ... 87
Odtwarzanie strumieniowe a zdarzenia dźwiękowe ... 89
Układanie dźwięków na warstwach ... 90
Synchronizacja dźwięków na warstwach ... 91
Edycja dźwięku we Flashu ... 93
Dodawanie efektów ... 94
Kompresja dźwięku ... 97
Synchronizacja dźwięku z prostą animacją ... 99
Ścisz ten hałas ... 103
Wstępne ładowanie dźwięku i animacji ... 109
Podsumowanie ... 111

Rozdział 5. Wprowadzenie do języka ActionScript ... 113
Podstawy programowania ... 113
Zmienne ... 114
Składnia kropkowa ... 120
Składnia kropkowa i zmienne listwy czasowej ... 121
Bezpośrednie odwołania do obiektów ... 125
Zmienne globalne ... 126
Zmienne lokalne ... 127
Składnia wywołania metody ... 128
Inne kluczowe elementy programowania ... 128
Tablice ... 129
Nawiasy klamrowe ... 129
Pętle ... 130
Komentarze ... 132
Funkcje ułatwiające eliminowanie błędów ... 132
Podsumowanie ... 134

Rozdział 6. Identyfikatory klatek ... 135
Identyfikowanie klatek w filmie Flasha ... 136
Co się dzieje za kulisami ... 138
Nieliniowy dostęp do filmu ... 140
Obiekty SharedObject ... 140
Analiza kodu ActionScript ... 141
Niekompatybilność przeglądarek ... 145
Podsumowanie ... 145

Rozdział 7. Efekty maskowania ... 147
Krótki opis efektu maskowania ... 147
Klipy filmowe i symbole graficzne jako maski ... 148
Prosta maska ... 149
Zmiękczanie krawędzi maski o prostych kształtach ... 152
Zmiękczanie krawędzi masek o nieregularnych kształtach ... 154
ActionScript i maskowanie ... 157
Dodawanie akcji ... 162
Sterowanie maskowanym filmem za pomocą ActionScript ... 164
Podsumowanie ... 171

Rozdział 8. Flash i wideo ... 173
Podstawowe informacje dotyczące kompresji plików wideo ... 174
Podstawowe informacje dotyczące importu ... 175
Łączenie importowanych klipów wideo z plikami zewnętrznymi ... 176
Osadzanie klipów wideo w filmach Flasha ... 177
Jakość ... 178
Keyframe Interval ... 180
Scale ... 181
Synchronize video to Macromedia Flash document frame rate ... 182
Number of Video Frames to Encode ... 183
Import Sound ... 183
Jakie ustawienia są najwłaściwsze? ... 184
Osadzanie klipów wideo i manipulowanie nimi w klipach filmowych ... 184
Manipulowanie klipami wideo za pomocą metody loadMovie() ... 188
Wykorzystanie programu Sorenson Squeeze for Flash MX w procesie przetwarzania wideo ... 196
Podsumowanie ... 200

Rozdział 9. Wprowadzenie do programowania zorientowanego obiektowo ... 201
Podstawy programowania ... 202
Metodologia programowania ... 202
Obiekty, właściwości i metody ... 203
Języki obiektowe oparte na klasach lub prototypach ... 204
Tworzenie i używanie obiektów ... 204
Obiekty statyczne ... 209
Notacja tablicowa ... 209
Rozszerzanie możliwości obiektów ... 211
Rozszerzanie możliwości obiektów prototypowych ... 212
Przesyłanie argumentów przez wartość i przez odniesienie ... 213
Tworzenie własnych obiektów prototypowych ... 214
Projektowanie obiektów we Flashu ... 221
Zdarzenia ... 221
Zastępowanie detektorów zdarzeń ... 222
Sondy ... 223
Wyszukiwanie i usuwanie błędów ... 224
Akcja trace ... 225
Panel Debugger ... 225
Podsumowanie ... 229

Rozdział 10. Rysowanie za pomocą skryptów ... 231
Metody związane z rysowaniem ... 231
Rysowanie kwadratu ... 233
Rysowanie krzywych za pomocą skryptów ... 236
Rysowanie okręgów za pomocą skryptów ... 242
Tworzenie wypełnień za pomocą skryptów ... 244
Jednolite wypełnienia ... 244
Wypełnienia gradientowe ... 246
Rysowanie za pomocą skryptów — praktyczne zastosowanie ... 249
Podsumowanie ... 253

Rozdział 11. Elementy interfejsu ... 255
Prosta interaktywność typu „przeciągnij i upuść” ... 256
Upuszczenie obiektu na innym obiekcie ... 259
Tworzenie własnego kursora myszy ... 266

Tworzenie suwaka ... 272
Przewijalne pola tekstowe ... 278
Mechanizm wstępnego pobierania danych ... 284
Tworzenie paska postępu ... 285
Podsumowanie ... 288

Rozdział 12. Komponenty ... 289
Co to są komponenty ... 289
Do czego służą komponenty ... 290
Dlaczego komponenty ... 290
Stosowanie komponentów ... 291
Standardowe komponenty Flasha MX ... 292
Współpraca komponentów ze skryptami ... 295
Aktywacja komponentów za pomocą skryptów ... 296
Przesyłanie informacji do i z bazy danych ... 299
Modyfikowanie wyglądu komponentu ... 302
Pozyskiwanie komponentów z innych źródeł ... 303
Tworzenie własnych komponentów ... 307
Podsumowanie ... 308

Rozdział 13. Naśladowanie rzeczywistości — wprowadzenie do fizyki ... 309
Programowanie prostego ruchu ... 310
Sterowanie ruchem obiektu za pomocą przycisku ... 312
Sterowanie ruchem obiektu za pomocą klawiatury ... 317
Operacja rzucania obiektem przez użytkownika ... 319
Ruch uwzględniający bezwładność obiektu ... 323
Detekcja kolizji ... 326
Metoda hitTest() ... 327
Kolizje a odległości między obiektami ... 329
Rzucanie piłki i odbijanie jej od ścian ... 331
Definiowanie krawędzi ... 331
Przeszkody ... 336
Podsumowanie ... 344

Rozdział 14. Komunikacja z serwerem ... 345
Oprogramowanie pośredniczące i aplikacje serwerowe ... 345
Macromedia ColdFusion ... 346
ASP.NET (Microsoft .NET) ... 348
Java Servlet/JSP (J2EE) ... 350
PHP — procesor hipertekstu ... 351
ColdFusion czy PHP? ASP.NET czy JSP? ... 353
Modele danych udostępnianych przez serwer ... 353
Baza danych ... 354
Dostęp do bazy danych za pośrednictwem skryptu ColdFusion ... 358
Komunikacja Flasha z serwerem ... 360
Obiekty kategorii Client/Server we Flashu MX ... 362
Obiekt loadVars ... 362
Akcja loadVariables() ... 365
Przesyłanie zmiennych w adresie URL ... 366
Akcja getURL() ... 367
Tworzenie aplikacji korzystającej z obiektu loadVars ... 367
Analiza aplikacji ... 368
Interfejs utworzony we Flashu ... 368
Skrypty Flasha ... 370

Skrypt serwerowy ColdFusion ... 375
Gotowy skrypt ColdFusion ... 379
Dane XML ... 380
Definicje DTD i schematy ... 381
Flash i XML ... 382
Pobieranie i wysyłanie dokumentów XML ... 382
Przetwarzanie danych XML ... 387
Podsumowanie ... 388

Rozdział 15. Usługi Flash Remoting ... 389
Podstawowe wiadomości ... 390
Oprogramowanie serwera współpracujące z usługą Flash Remoting ... 391
Flash — usługi serwerowe ... 391
Usługi Flash Remoting w postaci komponentów CFC ... 392
Komunikacja Flasha MX z serwerem usług Flash Remoting ... 397
Tworzenie połączenia ... 399
Tworzenie detektora _Result ... 405
SSAS — serwerowe skrypty ActionScript ... 409
Technologia DataGlue ... 410
Umieszczanie usług serwerowych w edytorze skryptów Flasha MX ... 413
Narzędzie NetConnect Debugger ... 414
Podsumowanie ... 415

Rozdział 16. Komunikacja z aplikacją macierzystą ... 417
Akcje Flasha umożliwiające komunikację z przeglądarką ... 417
Akcja loadVariables() ... 418
Akcja getURL() ... 418
Model DOM ... 420
Wywoływanie funkcji JavaScript za pomocą akcji getURL() ... 421
Tworzenie cookies za pomocą akcji getURL() ... 425
Akcja FSCommand() ... 435
Projektor Flasha ... 438
Metody obiektu Flash w języku JavaScript ... 439
Pakiet Macromedia Dreamweaver JavaScript Integration Kit ... 443
Podsumowanie ... 447

Dodatek A Obiekty języka ActionScript ... 449
Obiekty ... 449
Słowniczek ... 450
Obiekt Accessibility ... 451
Obiekt Arguments ... 451
Obiekt Array ... 451
Obiekt Boolean ... 453
Obiekt Button ... 453
Obiekt Capabilities ... 455
Obiekt Color ... 456
Obiekt CustomActions ... 456
Obiekt Date ... 457
Obiekt Function ... 461
Obiekt Key ... 461
Obiekt LoadVars ... 463
Obiekt Math ... 465
Obiekt Mouse ... 467

Obiekt MovieClip ... 468
Obiekt Number ... 474
Obiekt Object ... 475
Obiekt Selection ... 477
Obiekt System ... 478
Obiekt Sound ... 479
Obiekt Stage ... 480
Obiekt String ... 481
Obiekt TextField ... 483
Obiekt TextFormat ... 488
Obiekt XML ... 490
Obiekt XMLSocket ... 494

Dodatek B Polecenia języka ActionScript ... 497
Akcje ... 497
Akcje sterujące odtwarzaniem filmu ... 497
Akcje realizujące komunikację z przeglądarką i serwerem ... 499
Akcje sterujące klipami filmowymi ... 501
Akcje zarządzające zmiennymi i obiektami ... 503
Wyrażenia warunkowe i pętle ... 504
Akcje umożliwiające drukowanie ... 507
Akcje związane z funkcjami użytkownika ... 510
Inne akcje ... 511
Operatory ... 512
Operatory standardowe ... 512
Operatory arytmetyczne ... 513
Operatory przypisania ... 514
Operatory bitowe ... 516
Operatory porównania ... 518
Operatory logiczne ... 520
Inne operatory ... 520
Funkcje ... 522
Funkcje konwertujące typy danych ... 523
Funkcje matematyczne ... 524
Stałe ... 525
Właściwości ... 525
Komponenty Flasha ... 529
Komponent CheckBox (pole wyboru opcji) ... 529
Komponent ComboBox (rozwijana lista) ... 531
Komponent ListBox (przewijana lista) ... 536
Komponent PushButton (przycisk) ... 541
Komponent RadioButton (wybór jednej z kilku opcji) ... 543
Grupa komponentów RadioButton ... 546
Komponent ScrollBar (pasek przewijania) ... 548
Komponent ScrollPane (przewijane pole) ... 551
Obiekt FStyleFormat (formatujący komponenty) ... 553

Dodatek C Skróty klawiszowe ... 559
Zestaw skrótów Macromedia Standard ... 559
Narzędzia przybornika ... 559
Polecenia menu ... 560
Skróty klawiszowe w środowisku testowym ... 564
Skróty klawiszowe w edytorze skryptów ... 565

Zestaw skrótów programu Flash 5 ... 567
Narzędzia przybornika ... 567
Skróty klawiszowe w środowisku testowym ... 572
Skróty klawiszowe w edytorze skryptów ... 573
Zestaw skrótów programu Fireworks 4 ... 575
Narzędzia przybornika ... 575
Polecenia menu ... 576
Zestaw skrótów programu FreeHand 10 ... 579
Narzędzia przybornika ... 580
Polecenia menu ... 580
Zestaw skrótów programu Illustrator 10 ... 583
Narzędzia przybornika ... 584
Zestaw skrótów programu Photoshop 6 ... 588
Narzędzia przybornika ... 588

Skorowidz ... 593

Rozdział 10.

Rysowanie za pomocą skryptów

Jedną z fascynujących nowości wprowadzonych we Flashu MX jest „rysowanie" za pomocą skryptów podstawowych kształtów wektorowych, takich jak linie czy wieloboki, w trakcie odtwarzania filmu. Udostępniony interfejs programistyczny pozwala sterować wszystkimi parametrami kształtów wektorowych, takimi jak kolor, grubość linii, wypełnienie i inne. Można je też z łatwością modyfikować. Rysowanie za pomocą skryptów otwiera nowy, ekscytujący rozdział w historii grafiki tworzonej we Flashu — po raz pierwszy możesz rysować na ekranie kształty wektorowe sterowane kodem, rezygnując tym samym z powielania setek klipów filmowych czy stosowania innych pośrednich i mało efektywnych rozwiązań. Nowy interfejs programistyczny, wykorzystujący skrypty, służy do tworzenia wszelkiego rodzaju grafiki, od wykresów po proste funkcje grafiki trójwymiarowej.

W tym rozdziale omówimy następujące zagadnienia:

- Praca z wirtualnym piórem
- Zmiany stylu rysowania
- Rysowanie linii prostych
- Rysowanie linii krzywych
- Rysowanie okręgów
- Wypełnienia
- Tworzenie gradientów
- Rysowanie i wypełnianie dynamicznie pracującego przycisku

Wszystkie te operacje wykonasz, nie dotykając ani jednego narzędzia rysowniczego Flasha!

Metody związane z rysowaniem

Wszystkie metody związane z rysowaniem za pomocą skryptów są metodami obiektu *MovieClip*. Oznacza to, że możesz tworzyć rysunki wektorowe w dowolnym klipie filmowym, łącznie z główną listwą czasową i klipami filmowymi tworzonymi dynamicznie. Należy

jednak zwrócić uwagę na to, że ze względu na specyfikę działania tych metod wszelkie utworzone za ich pomocą wektory pojawiają się na najniższej pozycji (czyli jako najniższa warstwa) w stosie warstw danego klipu filmowego.

Zanim rozpoczniemy rysowanie za pomocą skryptów, poznajmy służące do tego metody:

- `moveTo` — przemieszcza wirtualne pióro, nie rysując niczego. Metoda ta przyjmuje dwa argumenty — współrzędną X i współrzędną Y w układzie współrzędnych, którego środek leży w punkcie odniesienia klipu filmowego. Przykładem zastosowania tej metody może być polecenie `moveTo(10, 10)`.
- `lineStyle` — określa styl linii, które zostaną narysowane. Metoda ta przyjmuje trzy argumenty: `thickness` (grubość), `rgb` (kolor w kodzie RGB) oraz `alpha` (parametr krycia). Grubość linii jest podawana w punktach i argument ten może przyjmować wartość z zakresu od 0 do 255. W przypadku kiedy wartość argumentu wynosi 0, linia ma grubość włosową (`hairline`), czyli jest najcieńszą możliwą linią przy danej skali i rozdzielczości, która nigdy nie staje się grubsza przy zwiększeniu skali. Kolor linii określamy w kodzie szesnastkowym; domyślnie jest ustawiony kolor czarny o kodzie `0x000000`. Argument `alpha` może przyjmować wartości od 0 do 100, przy czym wartość 0 oznacza linię zupełnie przezroczystą, z kolei wartość 100 — linię całkowicie kryjącą. Przykładem zastosowania tej metody może być polecenie `lineStyle(2, 0x0000ff)`. (W przykładzie nie podaliśmy wartości trzeciego argumentu — jest on opcjonalny, podobnie zresztą jak drugi argument).
- `lineTo` — przesuwa wirtualne pióro w nowe miejsce, rysując linię prostą. Metoda ta przyjmuje dwa argumenty — współrzędną X i współrzędną Y w układzie współrzędnych, którego środek leży w punkcie odniesienia klipu filmowego. Są to współrzędne punktu docelowego, do którego przesuwa się wirtualne pióro, zostawiając ślad w postaci linii rozpoczynającej się od aktualnego położenia. Przykładem zastosowania tej metody może być polecenie `lineTo(10, 30)`.
- `curveTo` — przesuwa wirtualne pióro w nowe miejsce, rysując linię krzywą. Ta metoda wymaga podania czterech argumentów: `controlX`, `controlY`, `anchorX` i `anchorY`. Pierwsze dwa argumenty decydują o krzywiźnie rysowanej linii, zaś dwa kolejne określają położenie końcowego punktu krzywej względem punktu odniesienia klipu filmowego. Przykładem zastosowania tej metody może być polecenie `curveTo(30, 30, 70, 70)`.
- `clear` — usuwa z aktualnego klipu filmowego utworzoną przez skrypt grafikę wektorową. Wszelkie elementy graficzne utworzone za pomocą standardowych narzędzi graficznych Flasha pozostają nienaruszone. Przykładem zastosowania tej metody jest polecenie `clear()`.
- `beginFill` — rozpoczyna tworzenie nowego jednolitego wypełnienia. Metoda ta przyjmuje dwa argumenty: `rgb` oraz `alpha`. Pierwszy z nich określa kolor wypełnienia i jest podawany w postaci szesnastkowego kodu RGB. Na przykład kod `0x000000` to jednolite czarne wypełnienie. Drugi argument jest opcjonalny i określa parametr krycia. Jego wartość powinna mieścić się w przedziale od 0 do 100, zaś domyślnie przyjmowana jest wartość 100 (całkowite krycie). Polecenie `beginFill` powinno się pojawić przed wszelkimi poleceniami `lineTo` lub `curveTo`, rysującymi dany wypełniony kształt. Rysowanie tego kształtu powinno kończyć polecenie `endFill`. Przykładem zastosowania metody `beginFill` może być polecenie `beginFill(0xFF66FF, 80)`.

- `beginGradientFill` — rozpoczyna tworzenie nowego wypełnienia gradientowego. Ta metoda wymaga podania aż pięciu argumentów — `fillType` (typ wypełnienia), `colors` (kolory), `alphas` (parametry krycia), `ratios` (pozycje kolorów) oraz `matrix` (macierz przekształceń gradientu). Pierwszy argument, `fillType`, może przyjąć wartość `"linear"` (gradient liniowy) lub `"radial"` (gradient radialny). Drugi, `colors`, jest tablicą szesnastkowych kodów kolorów, wchodzących w skład gradientu. Trzeci argument, `alphas`, jest tablicą wartości procentowych, reprezentujących parametry krycia poszczególnych kolorów zawartych w tablicy `colors`. Czwarty argument, `ratios`, również jest tablicą — zawarte w niej wartości określają położenia poszczególnych kolorów z tablicy `colors` w gradiencie. Wartość 0 odpowiada początkowi gradientu, zaś wartość 255 reprezentuje najbardziej oddalony punkt gradientu. Nieco trudniejsze jest funkcjonowanie argumentu `matrix`. Jedną z dopuszczalnych postaci tego argumentu jest macierz o wymiarach 3×3, określająca przekształcenia, jakim ma być poddane wypełnienie gradientowe. Jeśli miałeś do czynienia z algebrą liniową, wiesz, jak wygląda macierz przekształceń. Jeśli nie spotkałeś się wcześniej z tym terminem, nie martw się — funkcjonowanie argumentu `matrix` wyjaśnimy w dalszej części rozdziału. Przykładem zastosowania metody `beginGradientFill` może być polecenie `beginGradientFill("linear", colors, alphas, ratios, matrix)`, przy czym wartości argumentów `colors`, `alphas`, `ratios` i `matrix` są wcześniej określone.
- `endFill` — kończy tworzenie wypełnienia. Przykładem zastosowania tej metody jest polecenie `endFill()`.

Jak z pewnością zauważyłeś, kilkakrotnie pojawił się termin *wirtualne pióro*. Co to takiego? Na tym pojęciu bazuje cały mechanizm rysowania za pomocą skryptów. Przesuwając wirtualne pióro po płaszczyźnie obrazu za pomocą poleceń `lineTo` i `curveTo`, rysujesz kolejne segmenty linii i kształtów wektorowych. Domyślnym, początkowym położeniem wirtualnego pióra jest punkt odniesienia klipu filmowego. Jeśli rysujesz na głównej listwie czasowej, to punkt ten znajduje się w lewym górnym narożniku obrazu (0, 0). Aby przesunąć wirtualne pióro w inne miejsce obrazu, bez rysowania linii, możesz użyć polecenia `moveTo`. Wirtualne pióro porusza się w układzie współrzędnych klipu filmowego — gdy klip filmowy się przemieszcza, rysunek przemieszcza się wraz z nim.

Najłatwiejszym sposobem opanowania zasad działania wirtualnego pióra jest praktyczne wypróbowanie jego możliwości.

Rysowanie kwadratu

Rozpocznijmy eksperyment od narysowania kwadratu. W pierwszej kolejności ustal styl linii. Kwadrat narysujemy za pomocą czarnej linii o grubości 2 punktów i kryciu 50%. Oto definicja takiego stylu linii:

```
this.lineStyle(2, 0x000000, 50);
```

Gdy wiesz już, jaką linią będziesz rysował kwadrat, ustal położenie punktu, od którego rozpoczniesz rysowanie. Jeśli chcesz, aby był to punkt, w którym aktualnie znajduje się wirtualne pióro, mógłbyś pominąć ten krok. Domyślnie wirtualne pióro znajduje się w punkcie odniesienia klipu filmowego (w przypadku głównej listwy czasowej jest to lewy górny narożnik obrazu). Przyjmijmy jednak, że chcesz rozpocząć rysowanie w jakimś innym miejscu, zatem powinieneś przesunąć pióro za pomocą metody `moveTo()`.

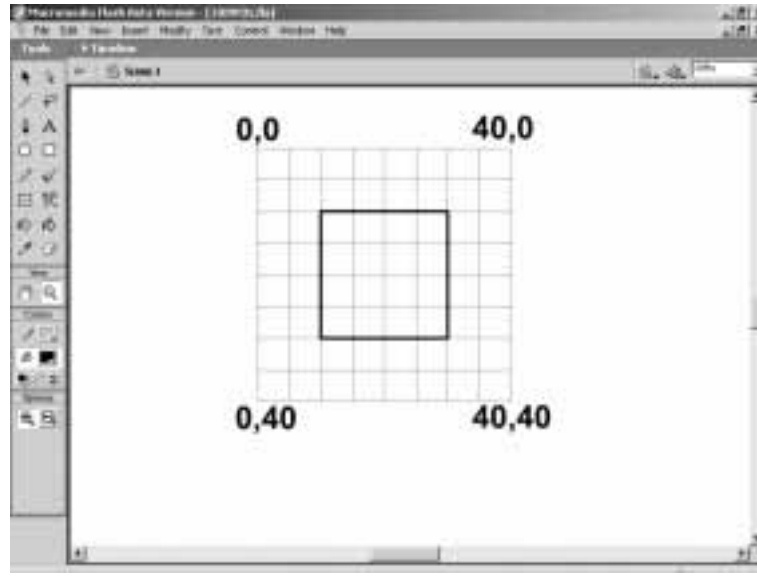
Jak wiesz, metoda ta przyjmuje dwa argumenty — współrzędne X i Y punktu, w którym ma się znaleźć wirtualne pióro. Aby umieścić pióro w punkcie oddalonym o 10 pikseli w dół i 10 pikseli w prawo względem początku układu współrzędnych, użyj takiego polecenia:

```
this.moveTo(10, 10);
```

Pióro zostało już umieszczone w pozycji startowej i ustaliliśmy styl linii, zatem możemy rozpocząć rysowanie. Działanie sprowadza się do podania w odpowiedniej kolejności współrzędnych narożników kwadratu. To nie będzie trudne. Po prostu zdecyduj, w którym kierunku chcesz obrysować kwadrat i jaka ma być długość boku kwadratu. Użyj metody `lineTo()`, przesuwając pióro do kolejnych narożników. Aby rozpocząć od poziomej linii, zwiększ wartość współrzędnej X o długość boku; współrzędną Y pozostaw niezmienioną (równą 10). Następnie narysuj pionowy bok kwadratu — nową współrzędną X pozostaw niezmienioną, zaś współrzędną Y zwiększ o długość boku. W podobny sposób narysuj pozostałe dwa boki, odejmując odpowiednie długości od współrzędnych. Jeśli przyjąłeś długość boku równą 20, sekwencja poleceń `lineTo()` powinna wyglądać tak:

```
this.lineTo(30,10);
this.lineTo(30,30);
this.lineTo(10,30);
this.lineTo(10,10);
```

Rysunek 10.1 przedstawia kwadrat narysowany za pomocą tych poleceń (dla ułatwienia nanieśliśmy siatkę).



Rysunek 10.1. *Za pomocą skryptu narysowaliśmy kwadrat*

To nie było trudne. Nie wierz mi na słowo. Poświęć kilka chwil i wykonaj ćwiczenie, aby opanować podstawowe zasady rysowania linii prostych za pomocą skryptów.

Ćwiczenie 10.1. Rysowanie kwadratu za pomocą skryptu

Wiesz już, jak można narysować kwadrat za pomocą skryptu. Teraz narysuj własny.

1. Utwórz nowy film Flasha[1]. W nim utwórz klip filmowy o nazwie `square` (kwadrat). Umieść klon tego klipu na obrazie i nadaj mu taką samą nazwę klonu, `square`.
2. Zaznacz pierwsze ujęcie na głównej listwie czasowej i wyświetl edytor skryptów (naciśnij klawisz *F9*).
3. Ustal styl linii. Tym razem niech to będzie 2-punktowa, czarna linia o parametrze krycia 100%. W edytorze skryptów wpisz taki wiersz:

```
square.lineStyle(2, 0x000000, 100);
```

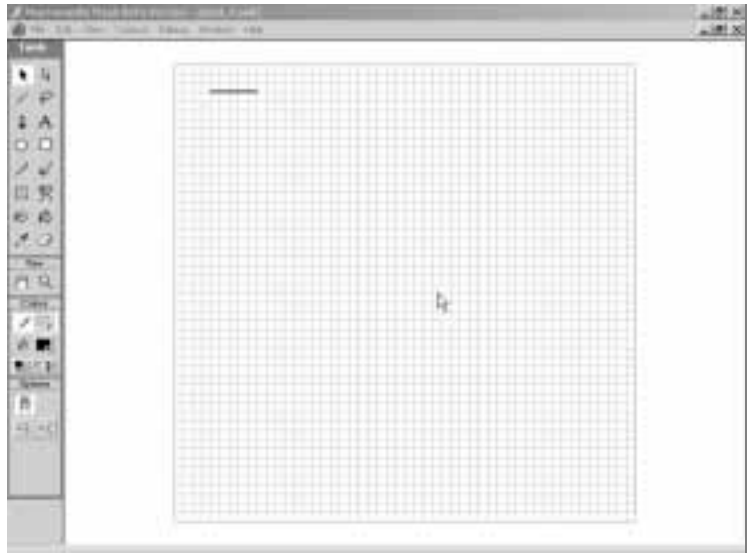
4. Ustal położenie punktu, od którego rozpoczniesz rysowanie. Niech to będzie punkt (40, 30). Dodaj następujący wiersz:

```
square.moveTo(40, 30);
```

5. Teraz zacznie się prawdziwa zabawa. Będziesz mógł sprawdzić, jak postępuje rysowanie. Utworzymy kwadrat, którego bok ma długość 50 pikseli. Rozpocznij od poziomego boku:

```
square.lineTo(90, 30);
```

6. Przetestuj film. Na obrazie powinna się pojawić czarna, pozioma linia (rysunek 10.2).



Rysunek 10.2. *Za pomocą wpisanego do tej pory kodu można narysować pojedynczą, czarną poziomą linię o długości 50 pikseli*

Teraz wystarczy, byś dodał pozostałe trzy boki kwadratu. Pamiętaj, że w tym momencie wirtualne pióro znajduje się w punkcie (90, 30).

7. Aby narysować pierwszy pionowy bok kwadratu, nie zmieniaj współrzędnej X wirtualnego pióra, a jego współrzędną Y zwiększ o 50 pikseli. Dopisz do skryptu poniższy wiersz:

```
square.lineTo(90, 80);
```

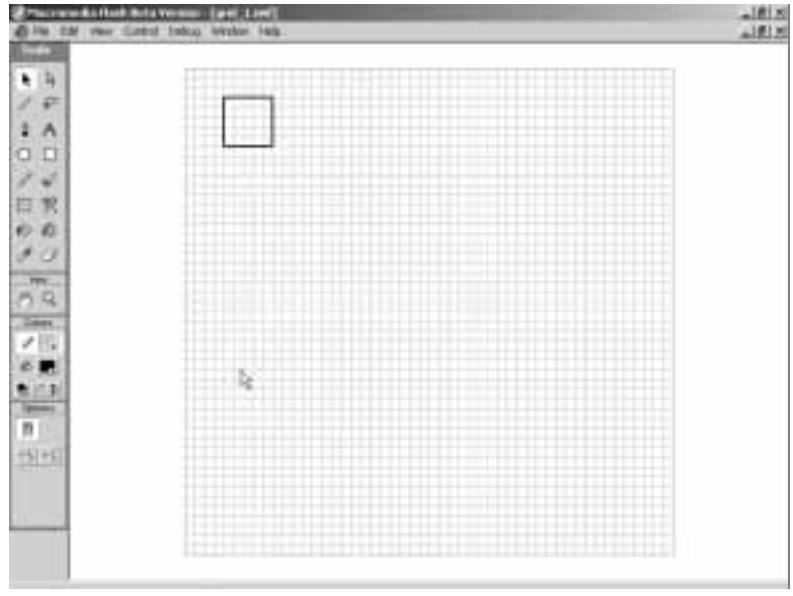
8. Zapisz i ponownie przetestuj film.

[1] Jeśli chcesz, by kwadrat był rysowany na tle siatki, możesz otworzyć plik *grid.fla*, zapisany na płycie CD-ROM w katalogu *Przyklady/Rozdzial_10 — przyp. tłum.*

Teraz zastanów się, jak powinny wyglądać dwa pozostałe wiersze. Śmiało, eksperymentuj. Jeśli pojawią się problemy, zajrzyj do listingu 10.1. Film powinien wyglądać tak, jak pokazano na rysunku 10.3. Pamiętaj, by zapisać plik.

Listing 10.1. *Kompletny kod rysujący kwadrat*

```
square.lineStyle(2, 0x000000, 100);
square.moveTo(40, 30);
square.lineTo(90, 30);
square.lineTo(90, 80);
square.lineTo(40, 80);
square.lineTo(40, 30);
```



Rysunek 10.3. *Używając wyłącznie kodu, możesz narysować na obrazie kwadrat*

Nie poprzestawaj na narysowaniu kwadratu. Poeksperymentuj, zmieniając parametry w skrypcie. Sprawdź, jak wpływają na wygląd rysunku.

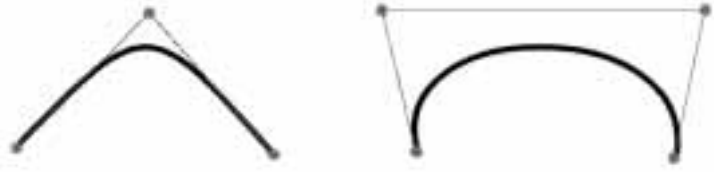
Oprócz linii prostych możesz również rysować linie krzywe, korzystając z polecenia `curveTo`. Aby zrozumieć, jak za pomocą skryptu rysować krzywe, najpierw powinieneś zapoznać się z kilkoma ogólnymi informacjami na temat krzywych we Flashu.

Rysowanie krzywych za pomocą skryptów

W większości wektorowych programów ilustracyjnych, takich jak FreeHand czy Illustrator, występują krzywe nazywane *krzywymi Beziera drugiego stopnia*. Ich kształt jest określany za pomocą czterech punktów — dwa z nich to wierzchołki rysowanej krzywej (linii), zaś dwa pozostałe są wierzchołkami stycznych. Wierzchołki krzywej znajdują się oczywiście na obu jej końcach, z kolei za pomocą stycznych można regulować jej krzywiznę (styczne zwykle są wyposażone w przeciągalne uchwyty, będące zarazem wierzchołkami stycznych).

Tego typu krzywe można kształtować intuicyjnie, zwłaszcza gdy są wyświetlane dynamicznie na ekranie komputera. Istnieje jeszcze inny typ krzywych, tak zwane *krzywe Beziera drugiego stopnia*, które są wewnętrznie wykorzystywane przez Flasha. Rysując krzywe za pomocą skryptów, posługujemy się właśnie krzywymi Beziera drugiego stopnia.

W uproszczeniu można powiedzieć, że krzywe Beziera drugiego stopnia są uproszczoną postacią krzywych trzeciego stopnia — uproszczoną w ten sposób, że wierzchołki obu stycznych są połączone w jeden wierzchołek, czyli obie styczne łączą się w jednym punkcie. Z tego uproszczenia wynika, że krzywe drugiego stopnia nie pozwalają uzyskać niektórych krzywizn — na przykład wypukłości. Z drugiej strony, opis krzywej drugiego stopnia wymaga o 25% mniej danych niż opis krzywej trzeciego stopnia (rysunek 10.4). Właśnie dlatego Flash wewnętrznie przechowuje wszystkie informacje o krzywych w postaci krzywych drugiego stopnia. Również dlatego metoda `curveTo()` korzysta z tego rodzaju krzywych.



Rysunek 10.4. *W krzywych Beziera drugiego stopnia obie styczne mają wspólny wierzchołek; w krzywych Beziera trzeciego stopnia każda ze stycznych posiada własny wierzchołek. Flash potrafi pracować z reprezentacją krzywych Beziera trzeciego stopnia, lecz wewnętrznie opisuje je jako krzywe drugiego stopnia*

Stosowanie metody `curveTo()` jest proste — przyjmuje ona cztery argumenty: współrzędne X i Y wierzchołka stycznych oraz współrzędne X i Y końcowego wierzchołka krzywej (pozycję początkowego wierzchołka krzywej wyznacza aktualna pozycja wirtualnego pióra). Składnię metody `curveTo` można zatem zapisać tak:

```
curveTo(<styczna_x>, <styczna_y>, <koniec_krzywej_x>, <koniec_krzywej_y>);
```

Wykonując ćwiczenie 10.2, zobaczysz, jak wyglądają krzywe rysowane za pomocą poleceń `curveTo()`. Utworzymy mechanizm, który pozwoli użytkownikowi przeciągać zarówno oba wierzchołki krzywej drugiego stopnia, jak i wierzchołek jej stycznych — wszystko wyłącznie za pomocą kodu ActionScript!

Ćwiczenie 10.2. Tworzenie krzywych za pomocą skryptu

To ćwiczenie jest bardziej skomplikowane od poprzedniego, więc uzbrój się w cierpliwość i zaparz sobie kawę.

1. Rozpocznij od utworzenia nowego, pustego dokumentu Flasha. Otwórz edytor skryptów (panel *Actions*).
2. Utwórz trzy puste klipy filmowe — dwa dla wierzchołków krzywych oraz jeden dla uchwytu stycznej. W edytorze skryptów wpisz następujący kod:

```
this.createEmptyMovieClip("anchor0_mc", 1);
this.createEmptyMovieClip("anchor1_mc", 2);
this.createEmptyMovieClip("handle_mc", 3);
```

Aby narysować uchwyty, zastosujemy pewną sztuczkę. Dla każdego z klipów filmowych zdefiniujemy styl linii o grubości 10 pikseli i korzystając z tego stylu, narysujemy linię o długości 1 piksela. Uzyskamy w ten sposób kropkę, która kształtem przypomina koło.

3. Pod kodem, który znajduje się w 2. punkcie, dopisz następne wiersze (zwróć uwagę, że wierzchołki krzywej będą czarne, zaś wierzchołek stycznych niebieski):

```
anchor0_mc.lineStyle(10, 0x000000);
anchor1_mc.lineStyle(10, 0x000000);
handle_mc.lineStyle(10, 0x0000FF);

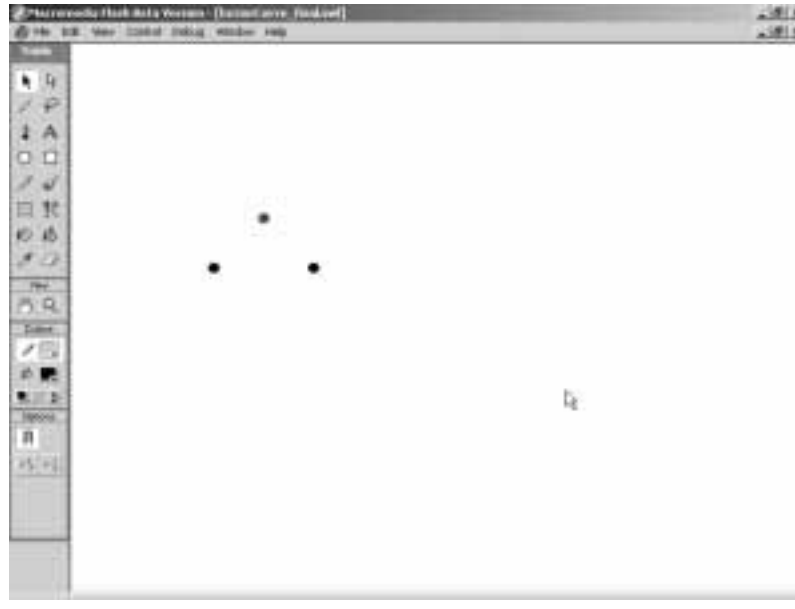
anchor0_mc.lineTo(1, 0);
anchor1_mc.lineTo(1, 0);
handle_mc.lineTo(1, 0);
```

Jeśli w tym momencie przetestujesz film, zobaczysz na obrazie tylko pojedynczą niebieską kropkę. To dlatego że pozostałe dwie czarne kropki znajdują się w tym samym miejscu i są zasłaniane przez niebieską.

4. Przemieść klipy filmowe:

```
handle_mc._x = 100;
handle_mc._y = 100;
anchor0_mc._x = 50;
anchor0_mc._y = 150;
anchor1_mc._x = 150;
anchor1_mc._y = 150;
```

5. Zapisz i przetestuj film. Powinien wyglądać tak jak na rysunku 10.5.



Rysunek 10.5. *Po rozmieszczeniu klipów filmowych za pomocą skryptu, niebieski uchwyt powinien się znajdować w górnym wierzchołku trójkąta, zaś czarne uchwyty u jego podstawy*

W następnych krokach zdefiniujemy funkcję rysującą krzywą w głównym filmie i korzystającą przy tym z pozycji klipów filmowych (uchwytów).

6. Pod wprowadzonym już kodem rozpocznij definicję funkcji o nazwie `drawCurve()`:

```
function drawCurve(){
}
```

7. Ustal styl linii dla głównej listwy czasowej — grubość włosowa i kolor czerwony. W tym celu pomiędzy nawiasami klamrowymi funkcji wpisz poniższy kod:

```
_root.clear();
_root.lineStyle(0, 0xFF0000);
```

8. W definicji funkcji dopisz wiersz, który umieszcza wirtualne pióro w miejscu pierwszego wierzchołka (klipu filmowego `anchor0_mc`):

```
_root.moveTo(_root.anchor0_mc._x, _root.anchor0_mc._y);
```

9. Następnie, korzystając z polecenia `curveTo`, narysuj krzywą, biegnącą do drugiego klipu filmowego, `anchor1`, o krzywiźnie określonej przez położenie klipu filmowego `handle_mc`. Wreszcie wywołaj akcję `updateAfterEvent()`, aby wymusić odświeżenie obrazu Flasha. Dopisz w definicji funkcji następujący kod:

```
_root.curveTo(_root.handle_mc._x, _root.handle_mc._y, _root.anchor1_mc._x,
_root.anchor1_mc._y);
updateAfterEvent();
```

Gdy użytkownik przeciągnie jeden z uchwytów na obrazie, zostanie wywołana funkcja `drawCurve`, która z kolei wywoła akcję `updateAfterEvent()`. Chcemy, aby przeciąganie uchwytów i aktualizacja kształtu krzywej było działaniem jak najbardziej płynnym, dlatego za pomocą tej akcji wymuszamy odświeżanie obrazu pomiędzy klatkami. Listing 10.2 przedstawia kompletną definicję funkcji `drawCurve()`.

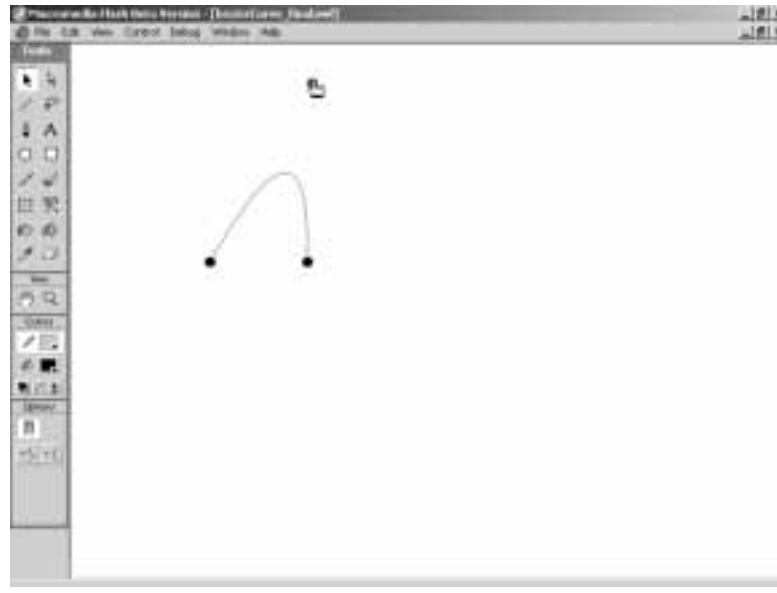
Listing 10.2. *Kompletna definicja funkcji drawCurve()*

```
function drawCurve(){
   _root.clear();
   _root.lineStyle(0, 0xFF0000);
   _root.moveTo(_root.anchor0_mc._x, _root.anchor0_mc._y);
   _root.curveTo(_root.handle_mc._x, _root.handle_mc._y, _root.anchor1_mc._x,
_root.anchor1_mc._y);
   updateAfterEvent();
}
```

10. Funkcja jest już gotowa, jednak film jeszcze nie. Musimy zdefiniować metodę `onPress()` (reagującą na wciśnięcie klawisza myszy) dla każdego z klipów filmowych reprezentujących uchwyty. Gdy użytkownik umieści kursor nad klipem `handle_mc` i wciśnie klawisz myszy, powinno rozpocząć się przeciąganie klipu. Wówczas każdy ruch myszy (powodujący przeciągnięcie uchwytu) powinien wywoływać funkcję `drawCurve()`, która od nowa narysuje krzywą. W tym celu metodzie `onMouseMove()` klipu `handle_mc` przypiszemy zdefiniowaną przed chwilą funkcję `drawCurve()`. Pod definicją funkcji `drawCurve()` umieść dalszy ciąg kodu:

```
handle_mc.onPress = function(){
   this.startDrag();
   this.onMouseMove = _root.drawCurve;
}
```

11. Przetestuj film. Możesz kliknąć niebieski uchwyt i przeciągnąć go. Gdy to zrobisz, zmieni się kształt krzywej (rysunek 10.6).



Rysunek 10.6. *Po dodaniu funkcji obsługującej zdarzenie onPress, możesz kliknąć i przeciągnąć niebieski uchwyt, aby zmienić kształt krzywej*

Istnieje jednak pewien problem — gdy klikniesz i przeciągniesz uchwyt, nie możesz go już upuścić. Teraz zajmiemy się tą kwestią. Musimy zdefiniować, co się ma wydarzyć po zwolnieniu klawisza myszy. Innymi słowy, musimy zdefiniować metodę `onRelease()`.

12. Przede wszystkim trzeba zakończyć przeciąganie klipu filmowego przycisku. Następnie wywołamy metodę `onMouseMove()`, aby odświeżyła krzywą. Jeśli nie zrobilibyśmy tego, krzywa mogłaby się dziwnie zachować w przypadku energicznego przeciągania, a następnie szybkiego zwolnienia klawisza myszy. I wreszcie, przypiszemy metodzie `onMouseMove()` wartość `undefined` (niezdefiniowana) — gdy nie jest przeciągany żaden uchwyt, krzywa nie powinna być modyfikowana. Gdybyś nie zlikwidował definicji metody `onMouseMove()`, Flash nadal śledziłby ruchy myszy, mimo że nie byłoby to konieczne i jedynie obciążałoby procesor.

    Aby zrealizować opisane zadania, dopisz w edytorze skryptów następujący kod:

```
handle_mc.onRelease = function(){
   this.stopDrag();
   this.onMouseMove();
   this.onMouseMove = undefined;
}
```

13. W czasach, gdy zdarzenia myszy mogły być powiązane wyłącznie z przyciskami, projektanci zawsze sprawdzali zdarzenia `on(release)` i `on(releaseOutside)` (czyli odpowiednio: zwolnienie klawisza myszy nad przyciskiem i zwolnienie go poza przyciskiem). Również w tym przypadku musimy to zrobić. Te same operacje, które są wykonywane w reakcji na zdarzenie `onRelease`, powinny być wykonane również po wykryciu zdarzenia `onReleaseOutside`. Można to osiągnąć, przypisując metodzie `onReleaseOutside` metodę `onRelease`, w taki sposób:

```
handle_mc.onReleaseOutside = handle_mc.onRelease;
```

14. Zapisz i przetestuj film.

    Utworzyłeś zgrabny film, który w elegancki sposób demonstruje funkcjonowanie krzywych Beziera drugiego stopnia. Mógłbyś na tym poprzestać, lecz byłoby to mało twórcze. W prosty sposób możesz skopiować utworzone metody i przypisać je pozostałym dwóm klipom filmowym, reprezentującym początek i koniec krzywej. Dzięki temu zabiegowi użytkownik będzie mógł przeciągać każdy z trzech uchwytów.

15. Skopiujemy teraz metody klipu `handle_mc`, przypisane detektorom poszczególnych zdarzeń myszy i przypiszemy je pozostałym dwóm klipom. Zrealizujemy to za pomocą takiego prostego kodu:

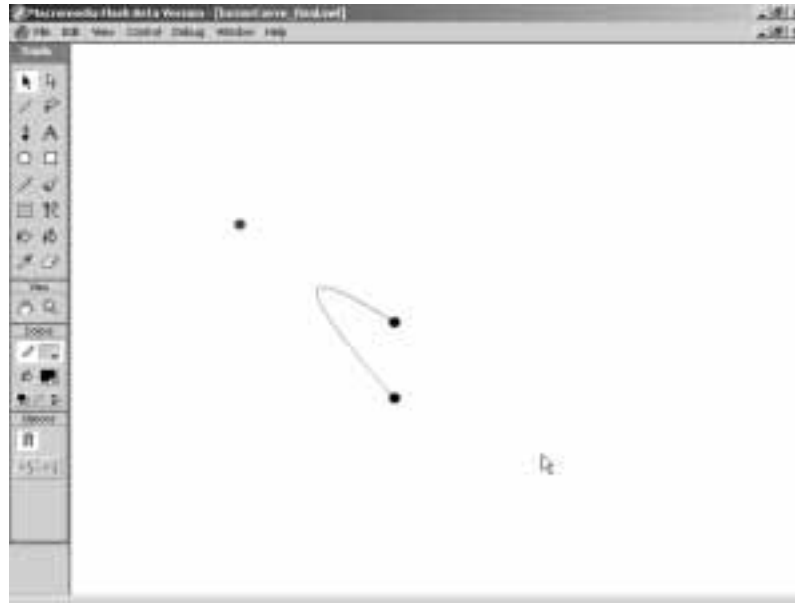
```
anchor0_mc.onPress = anchor1_mc.onPress = handle_mc.onPress;
anchor0_mc.onRelease = anchor1_mc.onRelease = handle_mc.onRelease;
anchor0_mc.onReleaseOutside = anchor1_mc.onReleaseOutside = handle_mc.onRelease;
```

16. Na końcu skryptu umieść wywołanie funkcji `drawCurve()`, aby krzywa pojawiła się od razu po uruchomieniu filmu, a nie dopiero po przeciągnięciu któregoś z uchwytów:

```
drawCurve();
```

17. Zapisz i przetestuj film.

Teraz film jest rzeczywiście gotowy i w pełni funkcjonalny (rysunek 10.7). I co najciekawsze, żadnego elementu nie narysowaliśmy na obrazie za pomocą tradycyjnych narzędzi ilustracyjnych Flasha. W przypadku jakichkolwiek wątpliwości lub problemów porównaj swój kod z listingiem 10.3[2].



Rysunek 10.7. *Ostateczna postać filmu rysującego interaktywną krzywą Beziera drugiego stopnia*

[2] **Możesz też porównać swój plik z plikiem** *bezierCurve_final.fla*, **zapisanym na płycie CD-ROM w katalogu** *Przyklady/Rozdzial_10 — przyp. tłum.*

Listing 10.3. ***Kompletny skrypt rysujący krzywe***

```
this.createEmptyMovieClip("anchor0_mc", 1);
this.createEmptyMovieClip("anchor1_mc", 2);
this.createEmptyMovieClip("handle_mc", 3);

anchor0_mc.lineStyle(10, 0x000000);
anchor1_mc.lineStyle(10, 0x000000);
handle_mc.lineStyle(10, 0x0000FF);

anchor0_mc.lineTo(1, 0);
anchor1_mc.lineTo(1, 0);
handle_mc.lineTo(1, 0);

handle_mc._x = 100;
handle_mc._y = 100;

anchor0_mc._x = 50;
anchor0_mc._y = 150;

anchor1_mc._x = 150;
anchor1_mc._y = 150;

function drawCurve(){
   _root.clear();
   _root.lineStyle(0, 0xFF0000);
   _root.moveTo(_root.anchor0_mc._x, _root.anchor0_mc._y);
   _root.curveTo(_root.handle_mc._x, _root.handle_mc._y, _root.anchor1_mc._x,
_root.anchor1_mc._y);
   updateAfterEvent();
}

handle_mc.onPress = function(){
   this.startDrag();
   this.onMouseMove = _root.drawCurve;
}

handle_mc.onRelease = function(){
   this.stopDrag();
   this.onMouseMove();
   this.onMouseMove = undefined;
}

handle_mc.onReleaseOutside = handle_mc.onRelease;
anchor0_mc.onPress = anchor1_mc.onPress = handle_mc.onPress;
anchor0_mc.onRelease = anchor1_mc.onRelease = handle_mc.onRelease;
anchor0_mc.onReleaseOutside = anchor1_mc.onReleaseOutside = handle_mc.onRelease;
drawCurve();
```

Już wiesz, jak rysować krzywe za pomocą skryptów. Czy potrafiłbyś jednak narysować okrąg? Jak myślisz, czy to trudne?

Rysowanie okręgów za pomocą skryptów

Często używanym rodzajem krzywych są okręgi. Flash nie udostępnia bezpośredniej metody rysowania okręgów za pomocą skryptów, dlatego operacja ta może być utrudniona. Prawdę mówiąc, ze względu na naturę krzywych Beziera drugiego stopnia nie jest możliwe narysowanie idealnego okręgu. Jednak możesz narysować kształt na tyle zbliżony do okręgu, że nikt nie dostrzeże różnicy. Co więcej, okręgi rysowane za pomocą narzędzia ilustracyjnego *Oval Tool* Flasha tak naprawdę również są jedynie przybliżeniami.

Wyprowadzanie wzorów matematycznych, umożliwiających przybliżenie okręgu serią krzywych, byłoby żmudne i zniechęcające. Dla ułatwienia zdefiniujemy ostateczną postać funkcji, która pozwala narysować okrąg o dowolnym promieniu w dowolnym miejscu obrazu. Definiując tę funkcję jako wewnętrzną metodę obiektu `MovieClip.prototype`, będziesz mógł skorzystać z niej w każdym klipie filmowym w filmie.

W funkcji korzystamy jedynie z metod `moveTo()` i `curveTo()` oraz z dwóch współczynników o wartościach wynikających ze skomplikowanych obliczeń, których nie będziemy tu przytaczać. Prototypowa funkcja `drawCircle` przyjmuje trzy argumenty — `x` i `y` (współrzędne środka okręgu) oraz `r` (promień okręgu).

Listing 10.4 przedstawia definicję funkcji `drawCircle`. Przepisz ten kod do filmu lub skopiuj go ze skryptu zapisanego na płycie CD-ROM w pliku *circle.as*, w katalogu *Przyklady/Rozdzial_10*.

Listing 10.4. *Funkcja prototypowa rysująca okręgi*

```
movieClip.prototype.drawCircle = function(x, y, r){
   var a = r * 0.414213562;
   var b = r * 0.707106781;
   this.moveTo(x+r, y);
   this.curveTo(x+r, y-a, x+b, y-b);
   this.curveTo(x+a, y-r, x, y-r);
   this.curveTo(x-a, y-r, x-b, y-b);
   this.curveTo(x-r, y-a, x-r, y);
   this.curveTo(x-r, y+a, x-b, y+b);
   this.curveTo(x-a, y+r, x, y+r);
   this.curveTo(x+a, y+r, x+b, y+b);
   this.curveTo(x+r, y+a, x+r, y);
}
```

Gdy funkcja jest zdefiniowana, wystarczy, byś określił styl linii (`lineStyle`) dla danego klipu filmowego i wywołał metodę `drawCircle()`, podając konkretne wartości argumentów `x`, `y` i `r`.

Ćwiczenie 10.3. Tworzenie okręgu za pomocą skryptu

Zdefiniowaliśmy już funkcję rysującą okrąg, wystarczy więc jej użyć w klipie filmowym, aby w dowolnym miejscu narysować okrąg o dowolnym promieniu.

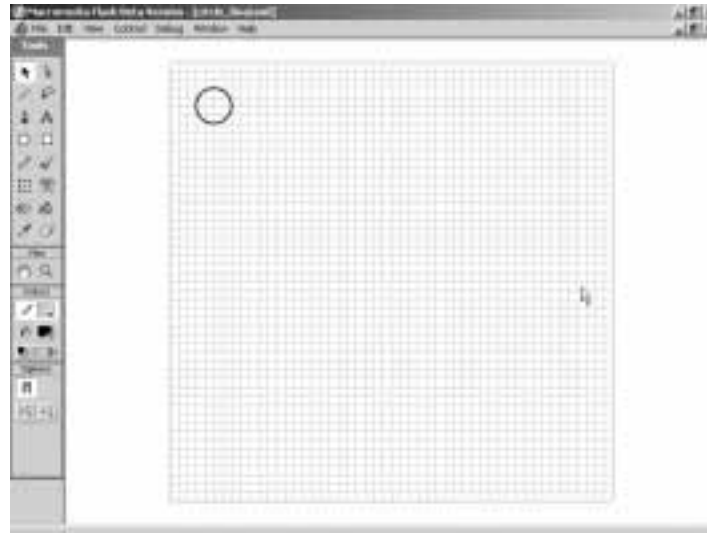
1. Utwórz nowy film Flasha i otwórz edytor skryptów.
2. W pierwszym ujęciu filmu umieść prototypową funkcję rysującą okrąg.
3. Utwórz na obrazie nowy pusty klip filmowy, umieszczając w skrypcie wiersz:

   ```
   createEmptyMovieClip("circle_mc_1", 1);
   ```

4. Kiedy utworzysz film, możesz przystąpić do rysowania. Pierwszym etapem jest jak zwykle ustalenie stylu linii. Następnie możesz wywołać metodę `drawCircle()`, podając współrzędne środka okręgu i jego promień. Aby narysować okrąg za pomocą 2-punktowej, czarnej linii, o środku w punkcie (50, 50) i promieniu 20, wpisz kod:

   ```
   circle_mc_1.lineStyle(2, 0x000000);
   circle_mc_1.drawCircle(50, 50, 20);
   ```

5. Zapisz i przetestuj film. Powinieneś uzyskać rezultat taki jak na rysunku 10.8.



Rysunek 10.8. *Okrąg (niemalże idealny) narysowany za pomocą skryptu*

W podobny sposób narysuj jeszcze kilka okręgów. Za każdym razem powtórz 3. i 4. krok, zmieniaj nazwę klonu (`circle_mc_2`, `circle_mc_3` itd.), zwiększaj numer poziomu warstwy i zmieniaj argumenty metody `lineStyle()`. Masz do dyspozycji 16 000 poziomów warstwy — wykorzystaj przynajmniej ich niewielką część!

W następnym podrozdziale dowiesz się, jak za pomocą skryptów można tworzyć wypełnienia.

Tworzenie wypełnień za pomocą skryptów

Gdy się nad tym zastanowić, tworzenie wypełnień za pomocą skryptów powinno być łatwe — i rzeczywiście tak jest! Używając skryptów, możesz tworzyć dwa rodzaje wypełnień — jednolite i gradientowe. Rozpoczniemy od pierwszego, prostszego rodzaju, a następnie przejdziemy do bardziej skomplikowanego, czyli gradientowego.

Jednolite wypełnienia

Aby rozpocząć tworzenie wypełnienia, wystarczy umieścić w skrypcie akcję `beginFill`:

```
beginFill(<kolor>, <parametr_krycia>);
```

Tak jak w przypadku innych metod rysujących wektory parametr `<kolor>` jest szesnastkowym kodem koloru, zaś parametr `<parametr_krycia>` — wartością procentową, określającą stopień krycia koloru (czyli odwrotność przezroczystości), z przedziału od 0 (całkowita przezroczystość) do 100 (pełne krycie).

Aby utworzyć wypełniony kształt, umieść wirtualne pióro w miejscu, w którym chcesz rozpocząć rysowanie. Następnie użyj polecenia `beginFill`, po którym umieść w skrypcie sekwencję poleceń `lineTo()` i `curveTo()`, rysujących kształt. Na końcu sekwencji umieść polecenie `endFill()`, zamykające wypełniony kształt. Spróbujmy zastosować to w praktyce.

Ćwiczenie 10.4. Tworzenie jednolitego wypełnienia

To ćwiczenie będzie dla Ciebie wyjątkowo łatwe — niemal wszystkie polecenia występujące w kodzie poznałeś już wcześniej.

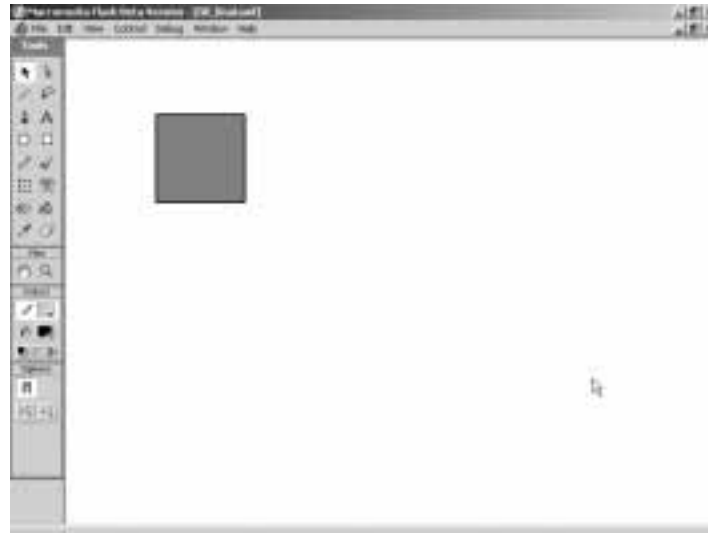
1. Utwórz nowy dokument Flasha i zapisz go na dysku twardym w pliku *fill.fla*.
2. W pierwszym ujęciu filmu umieść skrypt rysujący pusty kwadrat:

```
this.lineStyle(2, 0x000000, 100);
this.moveTo(10, 10);
this.lineTo(110, 10);
this.lineTo(110, 110);
this.lineTo(10, 110);
this.lineTo(10, 10);
```

3. Aby skrypt rysował kwadrat wypełniony jednolitym kolorem czerwonym, dodaj jeden wiersz kodu bezpośrednio po poleceniu `moveTo` i jeden na końcu skryptu. Dla ułatwienia dodane wiersze umieściliśmy pomiędzy komentarzami z gwiazdek:

```
this.lineStyle(2, 0x000000, 100);
this.moveTo(10, 10);
//*****************************
this.beginFill(0xFF0000);
//*****************************
this.lineTo(110, 10);
this.lineTo(110, 110);
this.lineTo(10, 110);
this.lineTo(10, 10);
//*****************************
this.endFill();
//*****************************
```

4. Zapisz i przetestuj film (rysunek 10.9).



Rysunek 10.9. *Dodając dwa proste wiersze do kodu rysującego kwadrat, możesz z łatwością wypełnić go jednolitym kolorem*

Jeśli chciałbyś uzyskać jednolity kwadrat bez krawędzi, po prostu pomiń wiersz definiujący styl linii, `lineStyle`.

Wypełnienia gradientowe

Rozumiesz już, na czym polega tworzenie wypełnień za pomocą skryptów. Zajmijmy się teraz bardziej skomplikowanymi wypełnieniami — gradientowymi. Wypełnienie gradientowe za pomocą skryptu wymaga zastosowania kilku tablic. Jeśli nie oswoiłeś się jeszcze z pojęciem tablicy we Flashu, niezbędne informacje znajdziesz w rozdziale 5.

Oto składnia metody `startGradientFill`, za pomocą której rozpoczyna się rysowanie wypełnienia gradientowego:

```
startGradientFill(<typ_gradientu>, <kolory>, <parametry_krycia>, <pozycje>,
<macierz_przekształceń>);
```

Metoda ta wymaga podania wszystkich argumentów: jeśli któryś z nich pominiesz lub jego postać będzie niepoprawna, gradient nie pojawi się na obrazie.

Pierwszy argument, `<typ_gradientu>`, może przyjąć wartość tekstową `"linear"` (gradient liniowy) lub `"radial"` (gradient radialny). W gradiencie liniowym kolejne kolory pojawiają się w postaci równoległych, płynnie przechodzących pasm, w gradiencie radialnym kolory rozchodzą się w postaci koncentrycznych kół.

Argumenty `<kolory>`, `<parametry_krycia>` i `<pozycje>` są tablicami, które muszą mieć taką samą długość (czyli wszystkie trzy muszą zawierać tyle samo wartości). Tablica `<kolory>` zawiera szesnastkowe kody kolorów, które znajdą się w gradiencie. Tablica `<parametry_krycia>` zawiera wartości z przedziału od 0 do 100, reprezentujące krycie poszczególnych kolorów gradientu — poszczególne parametry krycia z tej tablicy są przyporządkowane kolorom z tablicy `<kolory>` o takim samym indeksie. Tablica `<pozycje>` określa pozycje poszczególnych kolorów w gradiencie. Jej komórki mogą przyjmować wartości z przedziału od 0 (początek gradientu) do 255 (koniec gradientu).

Przypuśćmy, że chcesz utworzyć gradient rozpoczynający się od koloru czerwonego, następnie przechodzący w biel i wreszcie kończący się kolorem niebieskim. Poszczególne kolory gradientu mają być równomiernie rozmieszczone i półprzezroczyste. W takim przypadku omawiane argumenty przyjmą następującą postać:

```
kolory = [0xFF0000, 0xFFFFFF, 0x0000FF];
parametry_krycia = [50, 50, 50];
pozycje = [0, 127, 255];
```

Najbardziej skomplikowany jest ostatni argument, `<macierz_przekształceń>`. Może on przyjmować jedną z dwóch postaci — złożoną lub prostszą. W skomplikowanej postaci argument jest obiektem o dziewięciu właściwościach, noszących odpowiednio nazwy: `a`, `b`, `c`, `d`, `e`, `f`, `g`, `h`, `i`. Właściwości te reprezentują macierz przekształceń o wymiarach 3×3:

a	b	c
d	e	f
g	h	i

Jeśli jest Ci znana algebra liniowa, wiesz, jak funkcjonują i jak powinny wyglądać macierze przekształceń. W przeciwnym razie powinieneś użyć prostszej postaci argumentu `<macierz_przekształceń>`.

W prostszej postaci argument <macierz_przekształceń> również jest obiektem, lecz tym razem o sześciu właściwościach, noszących następujące nazwy: `matrixType`, `x`, `y`, `w`, `h` oraz `r`. Właściwość `matrixType` zawsze powinna przybierać wartość tekstową `"box"` — w ten sposób Flash pozna, że korzystasz z prostszej postaci argumentu <macierz_przekształceń>. Właściwości `x` i `y` określają współrzędne lewego górnego narożnika wypełnienia gradientowego. Właściwości `w` i `h` to odpowiednio szerokość i wysokość wypełnienia gradientowego. Wreszcie właściwość `r` określa kąt obrotu wypełnienia gradientowego, jej wartość jest podawana w radianach.



Aby kąt podany w stopniach przeliczyć na radiany, zastosuj taki wzór:

```
radiany = (stopnie * Math.PI)/180;
```

Zmienna `stopnie` przechowuje kąt podany w stopniach, zmienna `radiany` — kąt podany w radianach, zaś właściwość `Math.PI` reprezentuje stałą matematyczną π.

W ćwiczeniu wypełnimy gradientem dynamicznie rysowany okrąg, a za pomocą metody `startGradientFill` uzyskamy iluzję podświetlenia sfery.

Ćwiczenie 10.5. Tworzenie wypełnienia gradientowego

To ćwiczenie jest tylko nieco bardziej skomplikowane od poprzedniego — i tylko dlatego, że korzystamy w nim ze złożonego kształtu wektorowego i kilku tablic.

1. Utwórz nowy dokument Flasha i zapisz go w pliku o nazwie *gradient.fla*.
2. Otwórz edytor skryptów i zaimportuj skrypt z pliku *circle.as*, zapisanego na płycie CD-ROM w katalogu *Przyklady/Rozdzial_10*.
3. Aby uzyskać iluzję podświetlonej sfery, wypełnimy okrąg gradientem radialnym, rozpoczynającym się od bieli (w centralnej części gradientu), szybko przechodzącym w kolor sfery, który następnie ciemnieje i wreszcie przy samym końcu rozjaśnia się, symulując odbicie światła od podłoża. Taki gradient naśladuje grę światła i cienia na powierzchni sfery o określonym kolorze.

   W gradiencie użyjemy trzech kolorów: białego (`0xFFFFFF`), niebieskiego (`0x006699`) oraz ciemnoniebieskiego (`0x003366`). Każdy z kolorów będzie w 100% kryjący.

   Wpisz kod, tworzący opisany gradient:

   ```
   colors = [0xFFFFFF, 0x006699, 0x003366, 0x006699];
   alphas = [100, 100, 100, 100];
   ratios=[0,160,255,200];
   ```

4. Okrąg będzie miał promień 50 pikseli i jego środek znajdzie się w punkcie (100, 100). Aby odbicie światła pojawiło się w lewej górnej części „sfery", gradient umieścimy w punkcie (50, 50). Zarówno szerokość, jak i wysokość gradientu wyniesie 70, tak aby był on w całości widoczny wewnątrz okręgu.

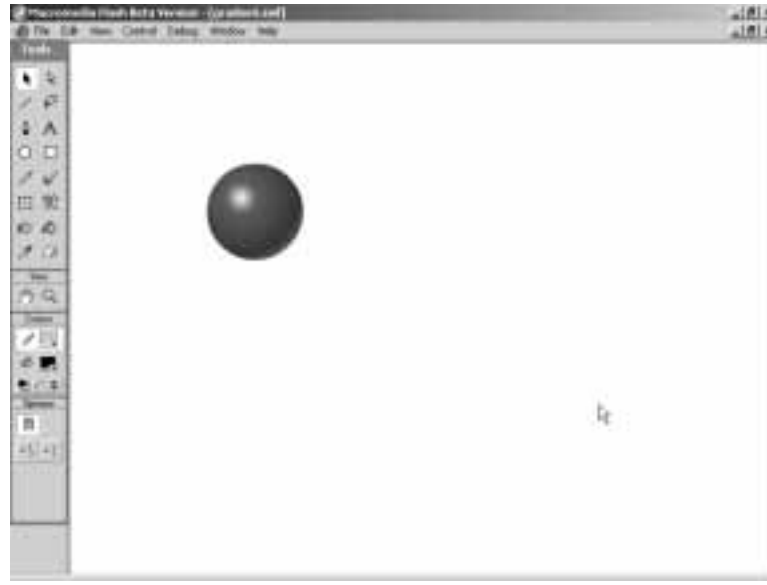
   Wpisz kod:

   ```
   matrix = {matrixType:"box", x:50, y:50, w:70, h:70, r:0};
   ```

5. Należy jeszcze rozpocząć wypełnianie, narysować okrąg i zakończyć wypełnianie. Dopisz następujący kod:

```
this.beginGradientFill("radial", colors, alphas, ratios, matrix);
this.drawCircle(100, 100, 50);
this.endFill();
```

Gdy zapiszesz i przetestujesz film, powinieneś zobaczyć ładnie oświetloną niebieską sferę (rysunek 10.10). W przypadku problemów porównaj swój kod z listingiem 10.5.



Rysunek 10.10. *Podświetlona sfera utworzona wyłącznie za pomocą skryptu*

Listing 10.5. *Kompletny kod tworzący podświetloną sferę*

```
movieClip.prototype.drawCircle = function(x, y, r){
   var a = r * 0.414213562;
   var b = r * 0.707106781;
   this.moveTo(x+r, y);
   this.curveTo(x+r, y-a, x+b, y-b);
   this.curveTo(x+a, y-r, x, y-r);
   this.curveTo(x-a, y-r, x-b, y-b);
   this.curveTo(x-r, y-a, x-r, y);
   this.curveTo(x-r, y+a, x-b, y+b);
   this.curveTo(x-a, y+r, x, y+r);
   this.curveTo(x+a, y+r, x+b, y+b);
   this.curveTo(x+r, y+a, x+r, y);
}
colors = [0xFFFFFF, 0x006699, 0x003366, 0x006699];
alphas = [100, 100, 100, 100];
ratios=[0,160,255,200];
matrix = {matrixType:"box", x:50, y:50, w:70, h:70, r:0};
this.beginGradientFill("radial", colors, alphas, ratios, matrix);
this.drawCircle(100, 100, 50);
this.endFill();
```

Poznałeś kilka podstawowych przykładów rysowania kształtów wektorowych za pomocą skryptów. W następnym podrozdziale zajmiemy się praktycznym zastosowaniem omawianych możliwości skryptów.

Rysowanie za pomocą skryptów — praktyczne zastosowanie

Rozdział ten zakończymy przykładem praktycznego wykorzystania możliwości rysowania za pomocą skryptów. Często opracowując różnego rodzaju projekty, potrzebujemy prostego mechanizmu tworzenia „standardowych" przycisków, przypominających te znane z systemu operacyjnego — szare, trójwymiarowe przyciski, do których od dawna jesteśmy przyzwyczajeni.

Co powinieneś zrobić, aby utworzyć taki przycisk? Przyciski zwykle mają napisy, z tego wniosek, że powinna istnieć możliwość podania tekstu, jaki pojawi się na przycisku. Każdy przycisk znajdzie się w oddzielnym klipie filmowym, zatem powinieneś mieć możliwość określenia nazwy klonu i poziomu warstwy, na którym się znajdzie. Wreszcie, tworzona metoda powinna być dostępna we wszystkich klipach filmowych w filmie, dlatego warto dodać ją do obiektu `movieClip.prototype`.

Widać, że mamy dużo do zrobienia. Najlepiej od razu zabierzmy się do pracy.

Ćwiczenie 10.6. Tworzenie przycisku za pomocą skryptu

Rozpoczniemy od zdefiniowania funkcji tworzącej przycisk.

1. Utwórz nowy dokument Flasha i zapisz go na dysku twardym pod nazwą *button.fla*.
2. Otwórz edytor skryptów (panel *Actions*) i zaznacz pierwsze ujęcie na głównej listwie czasowej.
3. Rozpocznij definicję funkcji, wpisując wiersz:

   ```
   MovieClip.prototype.createButton = function(text, name, depth){
   ```

   Jak widać, definicja funkcji zawiera trzy argumenty — `text`, `name` oraz `depth`. Pierwszy z nich będzie tekstem wyświetlanym na przycisku, drugi — nazwą nowego klipu filmowego, zaś trzeci będzie określał numer poziomu warstwy, na którym znajdzie się przycisk.
4. Na początku definicji funkcji wpisz kod, który utworzy nowy klip filmowy:

   ```
   this.createEmptyMovieClip(name, depth);
   var btn_mc = this[name];
   ```

   Pierwszy z wierszy tworzy nowy, pusty klip filmowy o podanej nazwie, umieszczając go na wskazanym poziomie warstwy. Drugi przypisuje zmiennej `btn_mc` odnośnik do utworzonego klipu.
5. Teraz zajmijmy się utworzeniem pola tekstowego, w którym wyświetlimy napis, jaki pojawi się na przycisku. Do klipu przycisku odniesiemy się za pomocą odnośnika zawartego w zmiennej `btn_mc`. Wewnątrz nawiasów klamrowych definicji funkcji dopisz następujący kod:

   ```
   btn_mc.createTextField("btn_txt", 1, 0, 0, 1, 1);
   btn_mc.btn_txt.autoSize = true;
   btn_mc.btn_txt.selectable = false
   btn_mc.btn_txt.text = text;
   ```

Jaką rolę pełni ten kod? Pierwszy wiersz tworzy wewnątrz klipu `btn_mc` pole tekstowe o nazwie `btn_txt`, umieszcza je na poziomie 1., w pozycji o współrzędnych (0, 0) i przypisuje jego szerokości i wysokości wartość 1. W następnych wierszach ustalamy wartości kilku właściwości pola tekstowego: `autoSize`, `selectable` i `text`.

Gdy właściwość `autoSize` pola tekstowego ma wartość `true`, wymiary pola są automatycznie dopasowywane do wyświetlanego tekstu. Wartość `false` właściwości `selectable` powoduje, że użytkownik nie może zaznaczyć tekstu w polu. Wreszcie właściwość `text` pola tekstowego reprezentuje tekst wyświetlany w tym polu.

6. Następnym naszym krokiem jest sformatowanie tekstu w polu. W definicji funkcji dopisz następujące wiersze:

```
var btn_tf = new TextFormat();
btn_tf.font = "_sans";
btn_mc.btn_txt.setTextFormat(btn_tf);
```

Powyższy kod tworzy obiekt *TextFormat* o nazwie `btn_tf`. Obiekt tego typu umożliwia formatowanie tekstu w polu tekstowym. Za pomocą właściwości `font` wybieramy systemową czcionkę bezszeryfową `_sans`. Ostatni wiersz kodu oznacza, że formatowanie za pomocą obiektu `btn_tf` zostało zastosowane wobec pola tekstowego `btn_txt`.

7. Musimy także ustalić, jakie są wymiary pola tekstowego (automatycznie dopasowującego się do tekstu), aby dobrać odpowiednie wymiary samego przycisku. Wpiszemy je do dwóch zmiennych — `txtWidth` (szerokość tekstu) i `txtHeight` (wysokość tekstu). Wpisz kod:

```
btn_mc.txtWidth = btn_mc.btn_txt._width + 4;
btn_mc.txtHeight = btn_mc.btn_txt._height;
```

8. Nadszedł wreszcie czas, aby narysować przycisk. Szerokość przycisku będzie o 4 piksele większa od szerokości pola tekstowego, aby napisowi w przycisku nie było zbyt „ciasno". Utwórz funkcję, która narysuje przycisk w położeniu górnym:

```
btn_mc.drawUp = function(){
   this.btn_txt._x = 2;
   this.btn_txt._y = 1;
   this.clear();
   this.moveTo(this.txtWidth, 0);
   this.beginFill(0xCCCCCC);
   this.lineStyle(1, 0x000000);
   this.lineTo(this.txtWidth, this.txtHeight);
   this.lineTo(0, this.txtHeight);
   this.lineStyle(1, 0xFFFFFF);
   this.lineTo(0, 0);
   this.lineTo(this.txtWidth, 0);
   this.endFill();
}
```

Dwa pierwsze polecenia powyższej funkcji ustalają położenie pola tekstowego, tak aby było wyśrodkowane w stosunku do przycisku. Następnie za pomocą polecenia `clear` oczyszczamy klip filmowy z grafiki i rysujemy przycisk zgodnie z ustaloną wcześniej szerokością i wysokością pola tekstowego. Lewa i górna krawędź przycisku jest biała, zaś prawa i dolna — czarna, w efekcie powstaje wrażenie trójwymiarowości przycisku. Wnętrze przycisku zostało wypełnione kolorem jasnoszarym, do którego jesteśmy przyzwyczajeni przez system operacyjny.

9. Narysowaliśmy przycisk w położeniu górnym. Przydałby się również rysunek przycisku w położeniu dolnym. Funkcja `drawDown` jest niemal identyczna z funkcją `drawUp`, jedynie krawędzie przyjmują kolory na odwrót, dzięki czemu przycisk sprawia wrażenie wciśniętego. Wpisz poniższy kod:

```
btn_mc.drawDown = function(){
   this.btn_txt._x = 3;
   this.btn_txt._y = 2;
   this.clear();
   this.moveTo(this.txtWidth, 0);
   this.beginFill(0xCCCCCC);
   this.lineStyle(1, 0xFFFFFF);
   this.lineTo(this.txtWidth, this.txtHeight);
   this.lineTo(0, this.txtHeight);
   this.lineStyle(1, 0x000000);
   this.lineTo(0, 0);
   this.lineTo(this.txtWidth, 0);
   this.endFill();
}
```

10. Teraz umieść w skrypcie polecenia, które uruchomią odpowiednią funkcję po wciśnięciu lub zwolnieniu przycisku:

```
btn_mc.onPress = btn_mc.drawDown;
btn_mc.onRelease = function(){
   this.drawUp();
   this.onButton();
}
btn_mc.onReleaseOutside = btn_mc.drawUp;
```

Zwróć uwagę na to, że oprócz rysowania odpowiedniej postaci przycisku metoda `onRelease` wywołuje metodę o nazwie `onButton()`. Za jej pomocą będziesz mógł później wykonywać kod w reakcji na kliknięcie przycisku.

11. Narysuj przycisk w początkowym stanie i zamknij definicję funkcji:

```
   btn_mc.drawUp();
}
```

12. Aby uruchomić świeżo utworzoną funkcję, umieść poza nią następujący wiersz kodu:

```
this.createButton("Mój przycisk", "myButton_mc", 1);
```

Jeśli chcesz prześledzić reakcję przycisku na kliknięcie, dodaj następujący kod:

```
myButton_mc.onButton = function(){
   trace("Ktoś kliknął mój przycisk!");
}
```

Zapisz i przetestuj film. Po jego uruchomieniu ekran Flasha powinien wyglądać tak jak na rysunku 10.11. Jeśli coś się nie zgadza, porównaj swój kod z listingiem 10.6.

Listing 10.6. ***Kompletny kod rysujący przycisk***

```
MovieClip.prototype.createButton = function(text, name, depth){
   this.createEmptyMovieClip(name, depth);
   var btn_mc = this[name];
```



Rysunek 10.11. *Trójwymiarowy przycisk utworzony wyłącznie za pomocą skryptu*

```
btn_mc.createTextField("btn_txt", 1, 0, 0, 1, 1);
btn_mc.btn_txt.autoSize = true;
btn_mc.btn_txt.selectable = false
btn_mc.btn_txt.text = text;

var btn_tf = new TextFormat();
btn_tf.font = "_sans";
btn_mc.btn_txt.setTextFormat(btn_tf);

btn_mc.txtWidth = btn_mc.btn_txt._width + 4;
btn_mc.txtHeight = btn_mc.btn_txt._height;

btn_mc.drawUp = function(){
   this.btn_txt._x = 2;
   this.btn_txt._y = 1;
   this.clear();
   this.moveTo(this.txtWidth, 0);
   this.beginFill(0xCCCCCC);
   this.lineStyle(1, 0x000000);
   this.lineTo(this.txtWidth, this.txtHeight);
   this.lineTo(0, this.txtHeight);
   this.lineStyle(1, 0xFFFFFF);
   this.lineTo(0, 0);
   this.lineTo(this.txtWidth, 0);
   this.endFill();
}

btn_mc.drawDown = function(){
   this.btn_txt._x = 3;
   this.btn_txt._y = 2;
   this.clear();
   this.moveTo(this.txtWidth, 0);
   this.beginFill(0xCCCCCC);
   this.lineStyle(1, 0xFFFFFF);
   this.lineTo(this.txtWidth, this.txtHeight);
```

```
        this.lineTo(0, this.txtHeight);
        this.lineStyle(1, 0x000000);
        this.lineTo(0, 0);
        this.lineTo(this.txtWidth, 0);
        this.endFill();
    }

    btn_mc.onPress = btn_mc.drawDown;
    btn_mc.onRelease = function(){
        this.drawUp();
        this.onButton();//wywołuje funkcję zawierającą akcję trace, znajdującą się poniżej
    }
    btn_mc.onReleaseOutside = btn_mc.drawUp;

    btn_mc.drawUp();
}
this.createButton("Mój przycisk", "myButton_mc", 1);
myButton_mc.onButton = function(){
trace("Ktoś kliknął mój przycisk!");
}
```

Podsumowanie

W tym rozdziale poznałeś wprowadzoną we Flashu MX możliwość rysowania na ekranie za pomocą skryptów. Wprawdzie udostępniane przez program metody ograniczają się do rysowania linii prostych i krzywych oraz jednolitych i gradientowych wypełnień, jednak rozwiązanie to daje projektantowi — programiście ogromne możliwości i w znacznym stopniu zwiększa swobodę jego działania.