helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

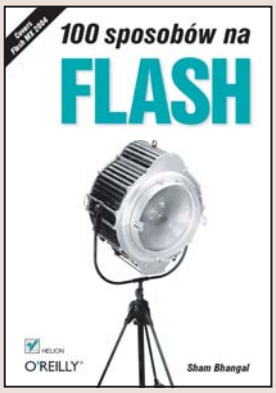
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

100 sposobów na Flash



Autor: Sham Bhangal
Tłumaczenie: Piotr Cieślak (rozdz. 7 – 12),
Konrad Mantorski (wstęp, rozdz. 1 – 6)
ISBN: 83-7361-629-2
Tytuł oryginału: Flash Hacks
Format: B5, stron: 480

„100 sposobów na Flash" to książka zawierająca opisy technik pracy z Flashem, wykorzystywanych przez najbardziej znanych projektantów. Wykorzystanie zawartych w książce sztuczek pozwoli Ci nie tylko na poszerzenie horyzontów twórczych, ale także na zmianę podejścia do projektowania. Dzięki nim stworzysz oryginalniejsze i bardziej wydajne aplikacji i jednocześnie zmniejszysz nakład pracy potrzebny do ich opracowania.

- Wizualne efekty specjalne oparte na elementach graficznych
- Maski i rysowanie
- Animacja postaci
- Zaawansowane techniki animacyjne
- Uwzględnianie fizyki w animacjach
- Formatowanie tekstu za pomocą stylów CSS
- Efekty specjalne tworzone w oparciu o tekst
- Dźwięk
- Tworzenie elementów interfejsów użytkownika
- Optymalizacja projektów
- Sztuczki z językiem ActionScript
- Integracja z przeglądarką internetową
- Zabezpieczanie prezentacji Flash

Wykorzystaj w swojej pracy sztuczki ekspertów.
Przekonasz się, jak wielu rzeczy jeszcze nie wiedziałeś o Flashu

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

Spis treści

O Autorach **7**

Słowo wstępne **11**

Wstęp **15**

Rozdział 1. Efekty wizualne **23**

1. Markowane przekształcenia pikselowe 24
2. Pikselowe efekty tekstowe 31
3. Efekt starego filmu 34
4. Tworzenie plików SWF z animowanych obrazków GIF 40
5. Animowanie plików PSD we Flashu 44
6. Drzewko na Brooklynie 50
7. Gałęzie kołysane wiatrem: imitacja ruchu drzewa 54

Rozdział 2. Efekty kolorystyczne **57**

8. Kolorystyczne efekty wideo 58
9. Ściemnienie — rozjaśnienie 63
10. Transformacje niestandardowe 67
11. Tworzenie i organizowanie niestandardowych palet barw 71
12. Wykorzystanie naturalnych schematów kolorystycznych 74
13. Efekt sepii 77

Rozdział 3. Rysowanie i maski **83**

14. Szybkie tworzenie kół w czasie rzeczywistym 84
15. Tworzenie syntetycznej grafiki 89
16. Tworzenie elementów mozaiki 91
17. Pokrywanie obszaru wzorem 94
18. Efekt ŕ la Escher 98
19. Redukcja niedokładności właściwości alfa 102
20. Wykorzystanie skomplikowanych kształtów jako masek 107
21. Wzory interferencyjne i efekt falowania 112
22. Postrzępione krawędzie bitmap 113

23. Dodawanie do bitmapy krawędzi wektorowych ... 116
24. Rozwiązanie problemu z przesunięciem bitmapy ... 119
25. Efekt odwracanej strony — symetria i maskowanie ... 123

Rozdział 4. Animacja ... 129
26. Płynna animacja skryptowa ... 130
27. Animacja ograniczona czasowo ... 133
28. Szybkie i wydajne animowanie postaci ... 137
29. Alternatywne narzędzia do tworzenia animacji ... 141
30. Animacyjne déjá vu ... 144
31. Matrix bez tajemnic ... 146
32. Generowana komputerowo animacja postaci ... 149
33. Efekty cząsteczkowe ... 155
34. Animowanie skomplikowanych kształtów ... 158

Rozdział 5. Efekty trójwymiarowe i fizyka ... 163
35. Imitacja efektu trójwymiarowego ... 164
36. Obrazy panoramiczne ... 169
37. Wydajny ploter 3D ... 176
38. Wykorzystanie przyspieszenia do imitowania efektu grawitacji i tarcia ... 180
39. Symulacja rzutu ... 183
40. Wykrywanie kolizji ... 186
41. Obracanie obiektu w określonym kierunku ... 192

Rozdział 6. Tekst ... 197
42. Wyraźny tekst ... 199
43. Autouzupełnianie ... 201
44. Tworzenie listy wszystkich wpisanych słów ... 208
45. Importowanie do Flasha złożonego formatowania tekstu ... 212
46. HTML i CSS we Flashu ... 218
47. Wykorzystanie funkcji ułatwień dostępu do utworzenia pola pomocy tekstowej ... 225
48. Modelowanie efektów tekstowych ... 231
49. Efekt pisma maszynowego ... 235
50. Czasowe efekty tekstowe ... 237
51. Efekty tekstowe na listwie czasowej ... 241

Rozdział 7. Dźwięk ... 245
52. Syntezator mowy w programie Flash ... 246
53. Synchronizacja mowy i ruchu ust animowanej postaci ... 253
54. Wymuszenie synchronizacji dźwięku ... 257
55. Jak z prostych dźwięków mono zrobić dźwięki stereo ... 260

56. Efekty dźwiękowe w czasie rzeczywistym ... 263
57. Jak szybko przygotować dźwięki do interfejsu aplikacji ... 264
58. Optymalizacja dźwięku ... 270
59. Dodatkowe informacje o dźwięku (punkty kontrolne) ... 278
60. Własna klasa obiektów umożliwiająca przekształcanie dźwięku ... 282

Rozdział 8. Elementy interfejsu użytkownika ... 285
61. Pokrętła Amita, czyli testowanie na gorąco ... 287
62. Prawy i środkowy przycisk myszy ... 292
63. Klipy filmowe jako przyciski ... 294
64. Hej, a gdzie jest mój pasek przewijania? ... 299

Rozdział 9. Wydajność i optymalizacja ... 303
65. Opanować pęczniejące projekty ... 306
66. Pomiar szybkości wczytywania złożonych witryn internetowych ... 308
67. Tuszowanie efektów włączenia trybu niskiej jakości ... 311
68. Zwiększanie wydajności poprzez optymalizację grafiki ... 316
69. Pomiar wydajności działania projektu ... 318
70. Dynamiczna zmiana stopnia skomplikowania animacji ... 320
71. Budżet zasobów systemowych ... 326
72. Zastępujemy wektory bitmapą ... 331
73. Optymalizacja pobierania i zastosowania komponentów ... 334

Rozdział 10. ActionScript ... 337
74. Zewnętrzne edytory skryptów ... 341
75. Jawne deklaracje typów zmiennych dla każdego ... 347
76. Podpowiedzi ... 351
77. Klonowanie obiektu ... 352
78. Licznik oczekujący (detektor bezczynności) ... 358
79. Szybkie wyszukiwanie w ActionScript ... 361
80. Blokowanie warstwy ze skryptem ... 364
81. Usuwanie błędów za pomocą polecenia trace() ... 366
82. Nieudokumentowane polecenia ActionScript ... 370
83. Tylnym wejściem: metoda ASnative() ... 375
84. Tajemnicze operatory ... 376
85. Importowanie plików ASC w postaci XML ... 381

Rozdział 11. Integracja z przeglądarkami ... 389
86. Strony przyjazne dla przeglądarki ... 391
87. Uniwersalny wykrywacz odtwarzacza Flash ... 395
88. Testowanie projektu w różnych wersjach odtwarzacza ... 401

89. Preferencje i standardowe opcje publikowania ... 404
90. Wyśrodkowanie pliku SWF bez skalowania ... 407
91. Wyśrodkowanie pliku SWF za pomocą CSS ... 408
92. Dynamiczne skalowanie zawartości ... 415
93. Łącza HTML we Flashu ... 418
94. Implementacja przycisku Wstecz we Flashu ... 420
95. Aktywna klawiatura w filmie SWF ... 427
96. Skróty klawiaturowe ... 428

Rozdział 12. Bezpieczeństwo ... 435
97. Odzyskiwanie zasobów z pliku SWF ... 443
98. Utrudnianie dostępu do filmów Flash ... 449
99. Film SWF melduje się posłusznie ... 452
100. Przeglądamy skompilowany ActionScript ... 456

Skorowidz ... 463

ROZDZIAŁ PIERWSZY

Efekty wizualne

Sposoby 1. – 7.

W książce przyjęto założenie, że czytelnik zna już podstawy Flasha w zakresie tworzenia efektów wizualnych i animacji za pomocą listwy czasowej. Jeśli jednak nie posługuje się on sprawnie programem, przedstawione w tym rozdziale techniki na pewno okażą się interesujące. Po poznaniu podstaw Flasha z internetowego lub książkowego podręcznika można wrócić do sposobów, które wydały się szczególnie interesujące. Na początku myślałem o tym, żeby książka zaczynała się od wskazówek dotyczących optymalizacji, zabezpieczeń itp. Uznałem jednak, że lepiej będzie z tym poczekać. Miałem też nadzieję, że rozwiązania przedstawione w tym rozdziale rozpalą ciekawość czytelników, poszerzą ich horyzonty — chciałem także pozostać wierny „hakerskiej" zasadzie zaczynania od najciekawszych rzeczy.

Dlatego też zamieściłem tu sposoby, które umożliwią osiągnięcie nieznanych wcześniej efektów lub efektów, które gdzieś się widziało, ale trudno wykonać je samodzielnie. Jak w przypadku wszystkich sposobów zawartych w tej książce, mam nadzieję, że będą one nie tylko uczyć, ale również inspirować. Liczę, że czytelnikom starczy motywacji, żeby wypróbować przedstawione tu sposoby, i że zaryzykują opracowanie własnych sztuczek.

Wszystkie sposoby zaprezentowane w tym rozdziale są w pewnym stopniu związane z efektami wizualnymi. Omówione zostaną efekty pikselowe oraz przekształcenie plików wykonanych w programie Photoshop i plików *GIF* we flashowe pliki *.fla* i *.swf* (pierwszy format oznacza plik źródłowy, drugi — dystrybucję tego pliku). Na koniec przedstawię sposób utworzenia drzewa kołyszącego gałęziami na wietrze. W następnych rozdziałach zajmiemy się dodatkowymi efektami wizualnymi wykorzystującymi przekształcenia, zmiany barw, techniki 3D, maskowanie oraz interfejs programistyczny API Drawing.

Chociaż maski są dokładniej omówione w rozdziale 3., to są tak ważnym elementem Flasha **[Sposób 1.]**, że można się z nimi zetknąć także w innych miejscach książki. Początkującym użytkownikom przyda się więc krótkie wprowadzenie.

Animacje we Flashu są tworzone poprzez nakładanie na siebie warstw animacji (są to takie same warstwy, z jakimi można zetknąć się w Photoshopie i innych programach do obróbki grafiki). W panelu *Timeline* znajduje się główna listwa czasowa, na której uporządkowane są wszystkie warstwy i na której odtwarzane są poszczególne klatki. Maski

są często używane do tworzenia efektów wizualnych, takich jak na przykład reflektor — jedna z warstw jest widoczna przez „otwór" w warstwie maskującej. Innymi słowy, warstwa maskująca określa obszar podrzędnej warstwy maskowanej, który jest widoczny (reszta zostaje zamaskowana i jest niewidoczna). Aby utworzyć taką warstwę w środowisku roboczym, należy wstawić na listwie czasowej nową warstwę (klikając *Insert/Timeline/Layer*) ponad warstwą, która ma zostać zamaskowana. Następnie w oknie dialogowym właściwości (*Modify/Timeline/Layer Properties*) trzeba zmienić typ warstwy na *Mask*. Następnie na tej warstwie należy utworzyć kształt, który będzie maską. We Flashu, te obszary warstwy maskującej, które zawierają piksele (są zamalowane), uwidoczniają warstwę leżącą poniżej, zaś te, które są puste, przesłaniają maskowaną warstwę. Na przykład aby utworzyć efekt reflektora, w którym maskowana warstwa będzie widoczna w okręgu, można skorzystać z narzędzi do rysowania (*Window/Tools*) i narysować czarne koło, które będzie maską.

We Flashu MX wprowadzono możliwość tworzenia masek skryptowych, w których maska zdefiniowana przez jeden klip filmowy jest używana do zamaskowania zawartości innego klipu filmowego. Maska skryptowa — jak wskazuje nazwa — jest maską używaną dynamicznie w czasie rzeczywistym, utworzoną przy wykorzystaniu języka ActionScript i metody *MovieClip.setMask()*. Stosowanie dynamicznej maski właściwie niczym się nie różni od tworzenia warstwy maski w środowisku roboczym, poza tym że metoda ta jest dużo bardziej elastyczna. Choć możliwe jest animowanie warstwy maski za pomocą narzędzi dostępnych w programie, to można uzyskać bardziej wyrafinowane animacje, animując maskę w czasie rzeczywistym przy wykorzystaniu skryptów.

Mam nadzieję, że to krótkie wprowadzenie umożliwi czytelnikom pełniejsze skorzystanie z różnych sposobów zawartych w tej książce, które wykorzystują maski tworzone w środowisku roboczym i w czasie rzeczywistym. Więcej informacji o maskach można znaleźć w pomocy online, w temacie *How do I/Basic Flash/Work with Layers/Add a mask Layer* (jak to zrobić/podstawy Flasha/warstwy/dodawanie maski) lub wpisać w indeksie pomocy hasło *mask*.

Teraz możemy zabrać się za naprawdę ciekawe rzeczy.



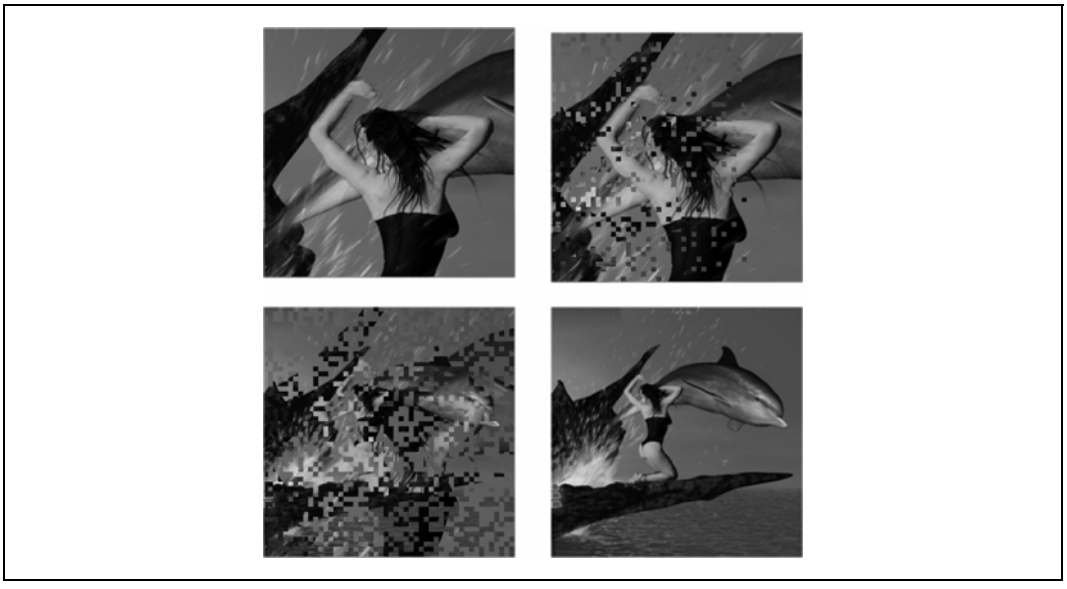
SPOSÓB 1.

Markowane przekształcenia pikselowe

Tworzenie pikselowych efektów zanikania i ścierania, przypominających efekty wykonane w programie Macromedia Director.

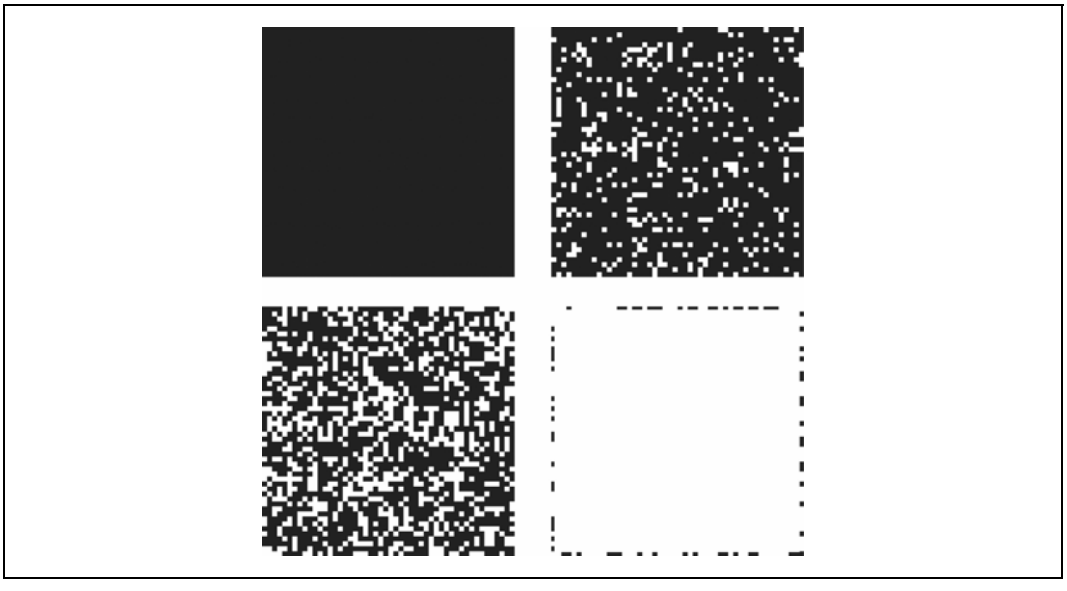
Silnik renderujący we Flashu bazuje na wektorach i przekształceniach wektorowych, nie zapewnia więc bezpośredniego dostępu do pojedynczych pikseli na ekranie. Zatem siłą rzeczy Flash nie obsługuje przekształceń pikselowych. Stosując markowane przekształcenia pikselowe, wykorzystujemy fakt, że piksele są niewielkie, a każda mała rzecz przypomina inną małą rzecz. Markowane przekształcanie pikselowe można łączyć z innymi sztuczkami wideo **[Sposób 8.]**, aby nieco urozmaicić statyczne bitmapy **[Sposób 3.]**.

Efekt przekształcenia pikselowego przedstawiono na rysunku 1.1.



Rysunek 1.1. Przekształcenie pikselowe, etapy 1 – 4

W miarę upływu czasu piksele pierwszego obrazu są ukrywane (maskowane) — obrazek znika piksel po pikselu. Usunięcie maski z pierwszego obrazka ujawnia drugi rysunek umieszczony poniżej. Stąd właśnie bierze się efekt przejścia jednego obrazka w drugi. Maski wykorzystane do utworzenia poprzedniego efektu pokazano na rysunku 1.2. Należy zauważyć, że w przypadku czarnych pikseli efekt maski wyświetla pierwszy (górny) obrazek; w przypadku białych pikseli (brak maski) wyświetlany jest obrazek spod spodu.



Rysunek 1.2. Maski przekształcenia pikselowego, etapy 1 – 4

Jak się za chwilę przekonamy, po wprowadzeniu drobnych zmian można utworzyć znacznie bardziej skomplikowane przekształcenia.

Efekt zostanie wykonany w trzech etapach:

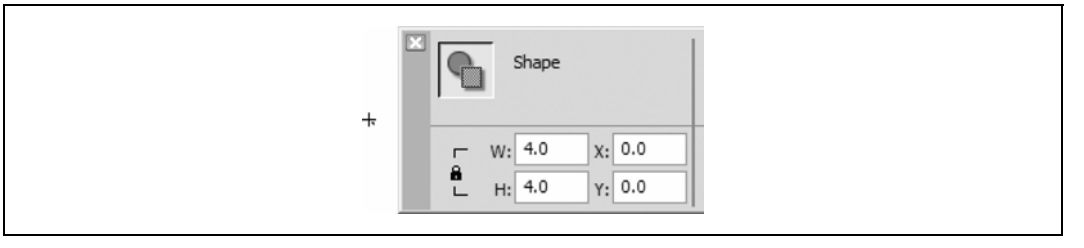
1. Utworzenie pseudopiksela. W tym ćwiczeniu utworzymy mały kwadrat o wymiarach 4×4 piksele.
2. Znalezienie sposobu na tworzenie wielu pseudopikseli. Można to łatwo zrobić, wykorzystując metodę *MovieClip.attachMovie()*.
3. Utworzenie przekształcenia poprzez napisanie skryptu, dzięki któremu każdy punkt (pseudopiksel) będzie znikać po określonym czasie. Wykorzystując wszystkie punkty jako dużą maskę, utworzymy efekt przejścia z jednego obrazka (lub klipu wideo) na drugi — tak jak na rysunku 1.1.

Pojawia się jednak pewien problem. Będziemy pracować z tysiącami pseudopikseli, ale nie jest wskazane wykorzystanie metody tak obciążającej procesor, jak tysiące skryptów *onEnterFrame()* uruchamianych w każdej klatce podczas trwania efektu. Zamiast tego posłużymy się funkcją *setInterval()*, która nie wymaga dużej mocy obliczeniowej, ponieważ kod będzie wykonywany tylko raz na jeden pseudopiksel w czasie efektu.

Tworzenie maski pikselowej

Utworzenie maski pikselowej jest tak łatwe, jak narysowanie kwadratu.

1. Otwórz nowy dokument Flash (*File/New/Flash Document*).
2. Kliknij polecenie *Document* w menu *Modify*, aby określić wymiary sceny większe niż 200×200 pikseli i wybrać białe tło (dzięki temu widoczny będzie czarny kwadrat utworzony w następnym punkcie).
3. Narysuj czarny kwadrat bez konturu (kontur nie byłby co prawda widoczny, ale spowolniłby efekt).
4. Posługując się panelem właściwości (*Window/Properties*), określ wartość wysokości i szerokości kwadratu na 4 piksele, a współrzędnych *X* i *Y* na 0. Ustawienia powinny wyglądać tak, jak w oknie po prawej stronie punktu zaczepienia na rysunku 1.3.

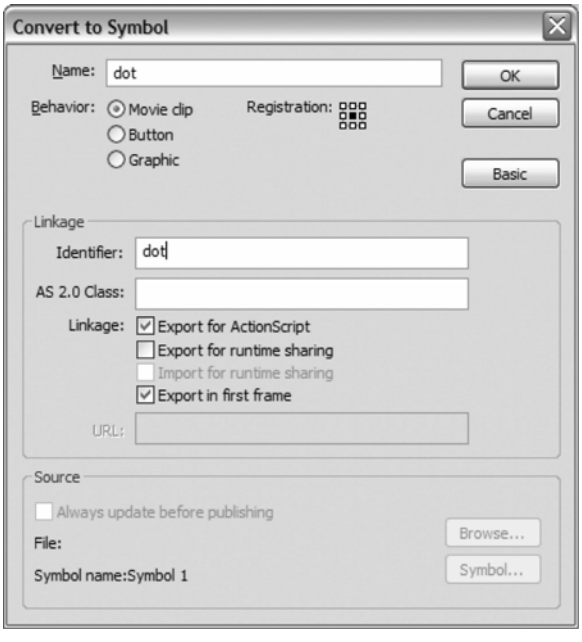


Rysunek 1.3. Maska pikselowa o wymiarach 4×4 i jej punkt zaczepienia



Wielbiciele języka ActionScipt mogą narysować maskę, wykorzystując interfejs API Drawing. Jednak dynamiczne kreślenie wszystkich kwadratów, które będą nam potrzebne, potrwałoby zbyt długo.

5. Przekształć kwadrat w klip filmowy — zaznacz go (za pomocą narzędzia *Selection*) i naciśnij klawisz F8 (lub kliknij *Modify/Convert to Symbol*), aby wyświetlić okno właściwości symbolu. Zmień nazwę symbolu na `dot` i sprawdź, czy opcje eksportu są tak ustawione, jak na rysunku 1.4 (kliknij przycisk *Advanced*, jeśli opcje *Linkage* są niewidoczne).



Rysunek 1.4. Okno dialogowe właściwości symbolu

Klip filmowy można usunąć ze sceny, ponieważ wykorzystamy metodę *MovieClip.attachMovie()* do dynamicznego powiązania symbolu w bibliotece z główną listwą czasową. Aby uniknąć tworzenia klipu, a następnie usuwania go ze sceny, można kliknąć polecenie *New Symbol* (nowy symbol) w menu *Insert* (wstaw) — lub nacisnąć klawisze *Ctrl+F8* (Windows) bądź *⌘+F8* (Mac) — aby utworzyć nowy symbol od razu w bibliotece.

Efekt wykorzystuje dużą maskę zbudowaną z kwadratów o wymiarach 4×4. Maska jest nałożona na pierwszy obrazek, efekt działa w taki sposób, że w miarę upływu czasu ukrywane są kolejne kwadraciki, w wyniku czego w wolnych miejscach zaczyna pojawiać się drugi obrazek (umieszczony poniżej pierwszego).

Jeszcze więcej pikseli

Na listwie czasowej wstaw nową warstwę o nazwie *actions* **[Sposób 80.]**. Na warstwie *actions* zaznacz klatkę 1. i osadź w niej następujący kod, posługując się panelem akcji (*F9*):

```
function drawGrid (theWidth:Number, theHeight:Number):Void {
   var initDot:Object = new Object();
   var k:Number = 0;
for (var i:Number = 0; i < theWidth; i += 4) {
     for (var j:Number = 0; j < theHeight; j +=4){
       var dotName:String = "dot" + i + "_" +j;
       initDot._x = i;
       initDot._y = j;
       this.attachMovie("dot", dotName, k, initDot);
       k++;
     }
}
}
drawGrid(200, 200);
```

Za pomocą tego kodu utworzony zostaje kwadrat o rozmiarach 200×200 pikseli, który składa się z klipów filmowych o rozmiarach 4×4 piksele (w funkcji *drawGrid()* można określić różne rozmiary, aby utworzyć siatkę innej wielkości). Każdy pseudopiksel znajduje się w określonym miejscu (`i, j`) i na określonym poziomie na scenie oraz ma nazwę instancji `dot`*`i_j`*. Nazwa pierwszej instancji (umieszczonej w górnym lewym rogu kwadratu) to `dot0_0`, a ostatniej (prawy dolny róg) to `dot199_199`. Aby wyświetlić klipy filmowe, można uruchomić kod w trybie debugowania (*Control/Debug Movie*). Należy jednak pamiętać, że programowi uruchomieniowemu (debugerowi) zajmie trochę czasu wyświetlenie wszystkich klipów (program może nawet przez chwilę przestać odpowiadać — trzeba dać mu parę sekund na wykonanie obliczeń!).



Efekt generuje wiele klipów filmowych: $(200/4)^2 = 2500$. Flash zaczyna działać wolniej, kiedy w tym samym czasie na ekranie wyświetlane są 3 – 4 tysiące klipów filmowych (nawet jeśli nie są one ruchome), dlatego też nie zaleca się przekraczania granicy 2500 klipów. Jeśli konieczne jest zastosowanie maski większej niż ta, nad którą obecnie pracujemy (200×200 pikseli), lepiej jest wykorzystać większe kwadraty, niż zwiększać ich liczbę.

Sterowanie pikselami

Należałoby teraz zatroszczyć się o to, by punkty znikały na żądanie. Można zastosować do tego metodę `setInterval(`*`object, "method", timer`*`)`, która wywołuje funkcję `object.method()` co określoną liczbę (`timer`) milisekund. Do napisanego wcześniej kodu dopisz następujący skrypt:

```
initDot.timer = 1000 + Math.ceil(Math.random()*800);
```

Skrypt tworzy właściwość `timer`, która dla każdego klipu pseudopiksela przechowuje liczbę całkowitą w zakresie od 1000 do 1800. Liczba 1000 określa moment pauzy przed uruchomieniem efektu, a 800 to czas trwania efektu. Obydwie wartości są podane w milisekundach, jednostce standardowo używanej w kodach ActionScript.

Ten sposób bazuje na efekcie maski, ale we Flashu można stosować tylko jedną maskę dla jednego klipu filmowego. Łatwo można obejść to ograniczenie, umieszczając wszystkie klipy filmowe pseudopikseli wewnątrz innego klipu, który posłuży za maskę. Nazwę klipu, który ma być zamaskowany, wpisujemy jako parametr do funkcji *drawGrid()* (zmiany w kodzie są wyróżnione):

```
function drawGrid (theWidth:Number, theHeight:Number,
imageClip:MovieClip):Void {
var initDot:Object = new Object();
var k:Number = 0;
//Utwórz klip maski, w którym znajdą się wszystkie punkty.
this.createEmptyMovieClip("mask", 1)
//Przypisz go jako klip maskujący.
imageClip.setMask(mask);
for (var i:Number = 0; i < theWidth; i += 4) {
  for (var j:Number = 0; j < theHeight; j +=4){
    var dotName:String = "dot" + i + "_" +j;
    initDot._x = i;
    initDot._y = j;
    initDot.timer = 1000 + Math.ceil(Math.random()*800);
    //Umieść maskujące punkty w klipie-kontenerze.
    mask.attachMovie("dot", dotName, k, initDot);
       k++;
  }
 }
}
drawGrid(200, 200, image1_mc);
```

Teraz wszystkie klipy punktów znajdują się już wewnątrz innego klipu o nazwie `mask`, którego użyjemy jako maski klipu filmowego o nazwie określonej w parametrze funkcji *drawGrid()*. W tym przypadku użyjemy pliku o nazwie `image1_mc`, który utworzymy później w punkcie „Używanie efektu”. Najpierw jednak musimy skończyć pracę nad klipami filmowymi punktów.

Tworzenie liczników

Utworzyliśmy już właściwość `timer` dla wszystkich punktowych klipów. Zabierzmy się teraz do napisania kodu odpowiedzialnego za znikanie punktów.

Przejdź w tryb edycji symbolu klipu *dot* i dodaj do niego nową warstwę o nazwie *actions* (pierwsza warstwa na listwie czasowej ma zazwyczaj nazwę *scripts* lub *actions* i służy wyłącznie do przechowywania skryptów). W pierwszej klatce na tej warstwie osadź następujący kod:

```
removeMe = function () {
  clearInterval(countDown);
  this.removeMovieClip();
};
var countDown = setInterval(this, "removeMe", timer)
```

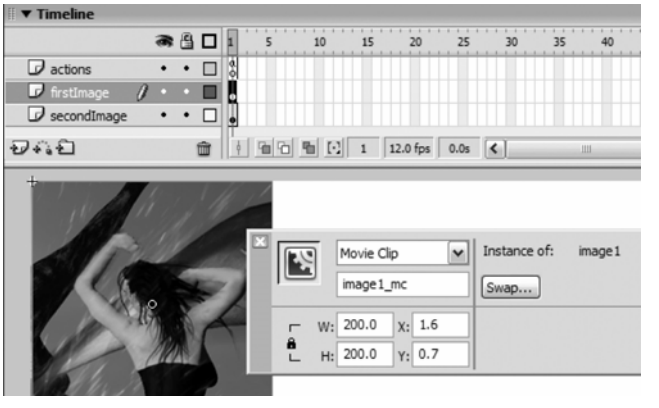
W ostatnim wierszu kodu wykorzystana została funkcja *setInterval()* w celu utworzenia dla każdego punktu licznika (`timer`) o nazwie `countDown`. Wywołuje ona funkcję *removeMe()*, gdy minie określony czas. Funkcja *removeMe()* czyści interwał, a następnie usuwa bieżący klip punktu, co tworzy efekt „znikających pikseli”.



Jeśli referencja do funkcji zostanie przekazana funkcji *setInterval()* jako pierwszy parametr, na przykład *setInterval(removeMe, timer)*, wartość słowa kluczowego `this` pozostanie niezdefiniowana (`undefined`) w funkcji *removeMe()*. Dlatego właśnie używamy alternatywnej formy wywołania funkcji *setInterval(this, "removeMe", timer)*, w której jako dwa pierwsze parametry przekazywane są obiekt i nazwa metody. W tym przypadku słowo kluczowe `this` jest obiektem przesyłanym jako pierwszy argument. Po wywołaniu funkcji *removeMe* słowo kluczowe `this` jest zdefiniowane i możliwe jest wywołanie funkcji *this.removeMovieClip()* w celu usunięcia klipu.

Używanie efektu

Do korzystania z efektu potrzebne są dwa obiekty, między którymi ma być zastosowane przejście i które należy umieścić na oddzielnych warstwach — pierwszy obrazek lub klip filmowy na warstwie górnej (jak na rysunku 1.5). Posługując się panelem właściwości, nadaj pierwszemu klipowi nazwę instancji `image1_mc`. Drugi obrazek może mieć dowolną nazwę, ponieważ i tak nie będzie ona używana w kodzie.



Rysunek 1.5. Tworzenie przejścia na dwóch warstwach

Działanie efektu można prześledzić w pliku *pixelMask.fla*, który można pobrać ze strony internetowej poświęconej książce.

Wzbogacanie efektu

Zmieniając interwał czasowy między momentami zniknięcia każdego punktu, można osiągnąć różne efekty przejścia. Na przykład zmiana wartości licznika na podstawie położenia punktów może być punktem wyjściowym wielu często spotykanych przekształceń pikselowych:

```
// Zanikanie z lewej do prawej
initDot.timer = 1000 + (Math.random()*(initDot._x)*10);
// Zanikanie poprzeczne
initDot.timer = 1000 + (Math.random()*(initDot._x + initDot._y)*5);
```

Uwagi końcowe

Maskowanie jest niedocenianą funkcją dostępną we Flashu. To jedna z tych opcji, które nie wydają się zbyt przydatne, dopóki nie pozna się ich lepiej. Nic zatem dziwnego, że wiele najbardziej interesujących efektów **[Sposób 21.]** tak obficie wykorzystuje maski!



SPOSÓB 2.

Pikselowe efekty tekstowe

Tworzenie zaawansowanych efektów i przekształceń tekstowych działających na poziomie piksela.

Ze stosowaniem efektów pikselowych we Flashu związany jest pewien problem — aby animacja nie absorbowała zbyt wiele mocy obliczeniowej procesora, konieczne jest ograniczenie liczby pseudopikseli. Problem ten można rozwiązać na dwa sposoby: użyć małych obrazków (jak w poprzednim ćwiczeniu **[Sposób 1.]**) lub zastosować efekt na obrazku, który ma wiele pikseli tła (można je zignorować, aby zmniejszyć liczbę potrzebnych pseudopikseli).

Sporo czasu zabrało mi kiedyś uświadomienie sobie, że tekst można podciągnąć pod kryterium „wiele pikseli tła". Szybka wycieczka po internecie udowadnia jednak, że nie jest to aż tak oczywiste, ponieważ chyba nikt inny nie używa tego sposobu.

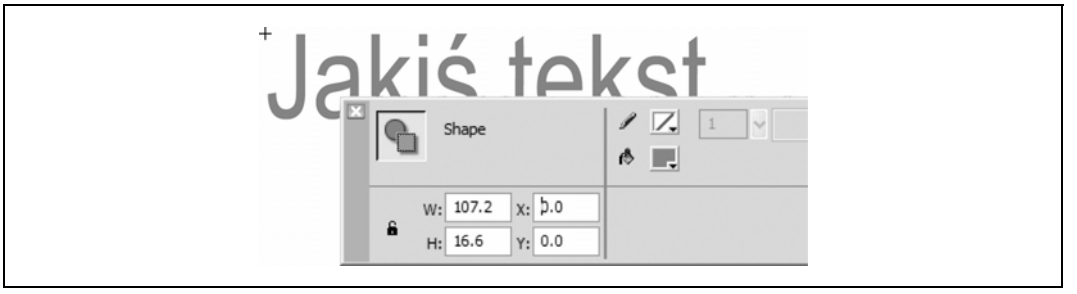
Tym razem będziemy pracować nad tekstem, który zostanie utworzony przez łączące się piksele rozrzucone po całym ekranie. Oczywiście można uzyskać różne efekty, zmieniając wartości położenia pikseli maski.

Sposób składa się z dwóch etapów:

- przekształcenie kształtu bloku tekstu w kwadraty o wymiarach 1×1 (czyli w pseudopiksele),
- animowanie pseudopikseli.

Oto kolejność działań:

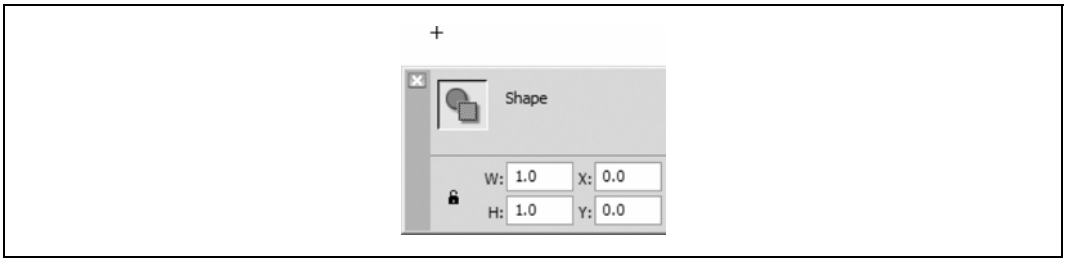
1. Utwórz pole tekstowe i wpisz dowolny tekst.
2. Naciśnij dwukrotnie klawisze *Ctrl+B* (Windows) lub *⌘+B* (Mac) lub kliknij dwukrotnie polecenie *Break Apart* w menu *Modify*, aby zmienić tekst w prosty kształt.
3. Tekst powinien być już zaznaczony; naciśnij więc klawisz *F8*, przekształcając w ten sposób napis w klip filmowy o nazwie **`text`**. Upewnij się, że zaznaczone jest pole wyboru *Export for ActionScript*, i określ identyfikator łącza jako **`text`**. Usuń instancję klipu ze sceny, ponieważ klip będzie ładowany w czasie rzeczywistym z biblioteki za pomocą metody *MovieClip.attachMovie()*.
4. Aby efekt poprawnie działał, punkt zaczepienia klipu filmowego musi znajdować się w górnym lewym rogu tekstu. Kliknij dwukrotnie klip filmowy, aby przejść w tryb *Edit in Place* (edycja w miejscu), a następnie zaznacz cały tekst, klikając *Edit/Select All* (edycja/zaznacz wszystko). W panelu właściwości wpisz 0 jako wartość współrzędnych *X* i *Y*, tak jak na rysunku 1.6.



Rysunek 1.6. Ustawianie położenia punktu zaczepienia zaznaczonego symbolu tekstowego

Aby wszystko działało, konieczne jest przekształcenie tekstu w prosty kształt za pomocą polecenia *Modify/Break Apart* (za chwilę przekonamy się dlaczego). Ma to jednak wpływ na plik — w przypadku dużej ilości tekstu powiększa dość znacznie wielkość dokumentu. Jednym ze sposobów przezwyciężenia tej niedogodności jest przekształcenie każdej litery czcionki w oddzielny klip zawierający prosty kształt i dynamiczne formowanie liter w napisy. Chociaż może się wydawać, że w ten sposób do pliku dodane zostaną dodatkowe bajty, trzeba pamiętać, że Flash wydajnie robi to samo, gdy w pliku *SWF* zapisywane są kontury czcionek (jest to konieczne, gdy litery mają być traktowane jak elementy graficzne).

Należy także utworzyć drugi klip filmowy z identyfikatorem łącza o nazwie `dot`. Klip `dot` powinien być kwadratem o wymiarach 1×1 piksel, a współrzędne *X* i *Y* — jak na rysunku 1.7 — powinny być ustawione na 0. Do określenia współrzędnych użyj panelu właściwości, ponieważ punkt będzie prawie niewidoczny.



Rysunek 1.7. Maska pikselowa o wymiarach 1×1

Poniższy kod replikuje efekt „zoom z boków z rozmyciem", ale tym razem tekst jest naprawdę rozmywany (zwykle efekt ten symulowany jest przy użyciu kanału alfa) — jak na rysunku 1.8 — ponieważ tekst został rozbity na piksele.



Rysunek 1.8. Pikselowy efekt tekstowy, etapy 1 – 4

```
function mover() {
  this._x -= (this._x - this.x) /4;
  this._y -= (this._y - this.y) /4;
}

function lastMover() {
  this._x -= (this._x - this.x) /4;
  this._y -= (this._y - this.y) /4;
  if ((this._x - this.x) < 0.1) {
    dotHolder.removeMovieClip();
    textClip._visible = true;
  }
}
//Umieść tekst na scenie i ukryj go.
textClip = this.attachMovie("text", "textClip", 0);
textClip._x = 200;
textClip._y = 100;
textClip._visible = false;
//Zainicjuj zmienne, łącznie z wysokością i szerokością.
var dots = 1;
var distance = 10000;
var stopDot = true;
var height = textClip._y + textClip._height;
var width = textClip._x + textClip._width;
//Utwórz klip punktowy dla każdego piksela tekstu.
var dotHolder = this.createEmptyMovieClip("holder", 1);
for (var j = textClip._y; j < height; j++) {
  for (var i = textClip._x; i < width; i++) {
    if (textClip.hitTest(i, j, true)) {
      var clip = dotHolder.attachMovie("dot", "dot" + dots, dots);
      if (stopDot) {
        clip._x = distance;
        clip.onEnterFrame = lastMover;
        stopDot = false;
      } else {
         clip._x = Math.random() * distance - distance/2;
         clip.onEnterFrame = mover;
      }
      //Zapisz pozycję, do której ma dotrzeć klip punktowy
      //(clip.x, clip.y) i przenieś go poza ekran.
      clip.x = i;
      clip.y = j;
      clip._y = j;
      dots++;
    }
  }
}
```

Pomińmy na razie omówienie funkcji *mover()* i *lastMover()*. Pozostały kod jest odpowiedzialny za umieszczenie tekstu na scenie i ukrycie go. Następnie inicjowanych jest kilka zmiennych, m.in. te, które określają wysokość i szerokość tekstu.

Pętla *for* korzysta z metody *MovieClip.hitTest()*, aby znaleźć inne niż puste piksele tekstu i utworzyć dla każdego z nich klipy filmowe dot. Każdemu punktowi przypisywany jest uchwyt zdarzenia *onEnterFrame()* w celu dodania animacji do całego efektu (zamiast tego można by użyć funkcji *setInterval()*, aby efekt był animowany **[Sposób 1.]**).

W tym kodzie pętli zostały wykorzystane dwa sposoby.

Pierwszy — *wykorzystanie hitTest()* — powoduje, że konieczne jest rozbicie tekstu na elementy. Metoda *hitTest()* zawsze zwraca wartość `false`, gdy zostanie użyta w połączeniu z dynamicznym polem tekstowym (uznaje wtedy piksele za puste).

Drugi sposób jest związany z metodą sprawdzania, czy pseudopiksele osiągnęły swoje docelowe pozycje. Punkty, rozmieszczone losowo na ekranie, są w większości sterowane uchwytem zdarzenia *mover()*. Jednak z pierwszym pseudopikselem, umieszczonym najdalej, jest skojarzony nieco bardziej złożony uchwyt — *lastMover()*. Zdarzenie zatrzymuje efekt, gdy ów pseudopiksel osiągnie swoje docelowe położenie — do tego czasu pozostałe pseudopiksele powinny już znaleźć się we wskazanych pozycjach (pod warunkiem, że mają do pokonania mniejszą odległość).



Chociaż sposób ten nie jest zbyt elegancki, to ważniejsza jest w nim poprawa wydajności niż kontrolowanie działania każdego pseudopiksela.

Uwagi końcowe

W internecie można znaleźć mnóstwo efektów tekstowych wykonanych we Flashu, ale nie widziałem nigdy, żeby w którymś z nich wykorzystano przekształcenia pikselowe. W wykorzystaniu pseudopikseli najciekawsze jest to, że można wtedy zastosować dowolny efekt cząsteczkowy (np. śnieg, wodospad lub **efekt gwiezdnej szybkości [Sposób 33.]**).



SPOSÓB 3.

Efekt starego filmu

Wykonanie efektu starej taśmy filmowej za pomocą programów Photoshop i Flash.

Wektorowy silnik graficzny Flasha ma mnóstwo zalet, ale czasem chciałoby się, żeby grafiki nie miały aż tak równych krawędzi. Efekt starego filmu jest jednym z najłatwiejszych i najszybszych sposobów nadania zwykłemu klipowi niepowtarzalnej atmosfery i surowego wyglądu. Efekt ten można dodatkowo wzbogacić o zmianę kolorów wideo **[Sposób 8.]** lub zmianę koloru na sepię **[Sposób 13.]** — projekt będzie wtedy bardziej intrygujący i atrakcyjny.

Najoczywistszym sposobem zastosowania zniekształceń charakterystycznych dla starych filmów jest dodanie wyświetlanych losowo linii i kropek. Efekt jest mniej więcej taki sam, ale atmosfera starego filmu jest jednak zupełnie inna (nie tak „gładka"). W tym ćwiczeniu wykorzystamy mapę bitową, która umożliwia uniknięcie wyrazistości grafiki wektorowej.

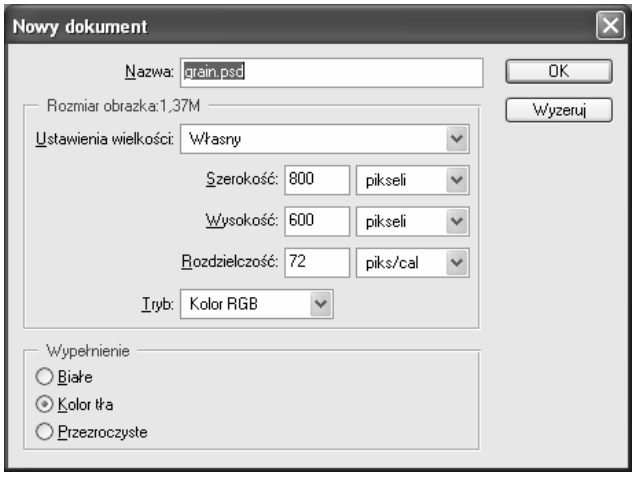
Najpierw utworzymy bitmapę z zakłóceniami w programie Photoshop, a następnie zaimportujemy ją do Flasha i poddamy dalszej obróbce. Oczywiście zamiast Photoshopa można użyć programu Fireworks — zasady pracy są podobne w obu aplikacjach.

Tworzenie bitmapy

Zabrudzenia, zadrapania i białe plamy nadają fotografii nieco bardziej autentyczny wygląd. Kurz, brud, włos lub nitka, gdy trafią na kliszę lub negatyw, wyglądają jak ciemne pasma i linie. Zadrapania są widoczne jako białe linie.

Aby przystąpić do przystosowania obrazka w programie Photoshop, należy wykonać następujące czynności:

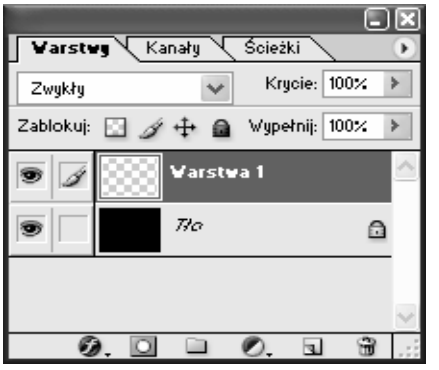
1. Otwórz program Photoshop.
2. Naciśnij klawisz D, aby ustawić domyślne kolory tła i pierwszego planu.
3. Naciśnij klawisz X, aby zamienić kolory — kolor tła będzie czarny, a pierwszego planu — biały.
4. Utwórz nowy plik o nazwie *grain.psd*, klikając polecenie *Nowy* w menu *Plik*. Określ wymiary mapy bitowej, aby obrazek był dłuższy niż szerszy. W przykładzie wybrałem 800×400 pikseli, ale obrazek może być oczywiście mniejszy (zazwyczaj będzie to 400×200 pikseli).
5. W oknie dialogowym tworzenia nowego dokumentu sprawdź, czy w polu *Wypełnienie* zaznaczona została opcja *Kolor tła* (jak na rysunku 1.9). Utworzony zostanie pusty czarny kwadrat.



Rysunek 1.9. Photoshop: zaznaczanie opcji koloru tła

6. Dodaj nową warstwę, klikając ikonę tworzenia nowej warstwy na zakładce *Warstwy*. Będziemy rysować tylko na jednej warstwie, należy więc upewnić się, że *Warstwa 1* jest zawsze zaznaczona na zakładce warstw — jak na rysunku 1.10.

Zabierzmy się do rysowania efektów. Na starych filmach można zobaczyć trzy rodzaje zniekształceń: kreski, kropki i łatki oraz rysy.



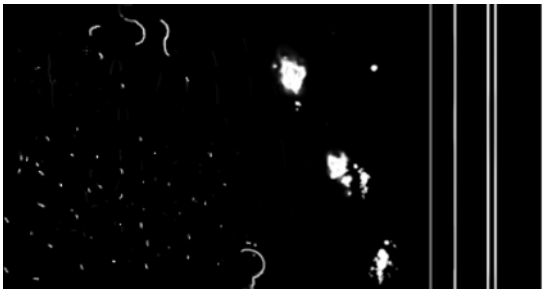
Rysunek 1.10. Photoshop: nowo utworzona warstwa

Kreski
Powstają, gdy na filmie znajdą się ciemnie, długie pasma (np. włosy).

Kropki i łatki
Ciemnie kropki powstają wskutek zabrudzenia przez kurz lub inny materiał, a białe po zarysowaniu lub uszkodzeniu filmu.

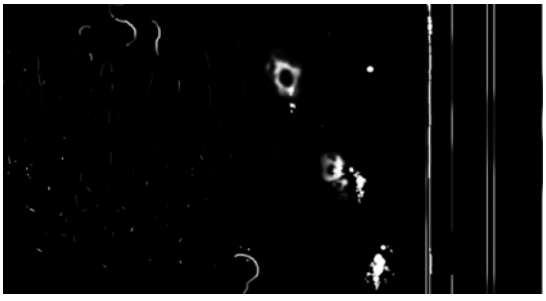
Rysy
Powstają wskutek zarysowania filmu, czyli mechanicznego usunięcia fragmentu obrazu.

Za pomocą narzędzi dostępnych w programie Photoshop (najlepiej ołówka i pędzla) dodaj trzy typy zniekształceń do obrazka na *Warstwie 1*. Jak widać na rysunku 1.11, po lewej stronie narysowałem małe kropki, w środku duże łaty, a po prawej rysy. Narysowałem też na dole i na górze kreski imitujące włosy na kliszy.



Rysunek 1.11. Photoshop: podrabiane uszkodzenia

Posługując się narzędziem *Gumka* o średniej grubości, można strawić kolor pikseli. W prawdziwych filmach głębokie zadrapania i inne uszkodzenia są widoczne na biało, ale wiele zniekształceń ma na powierzchnię kliszy filmowej jedynie częściowy wpływ — ten właśnie efekt postaramy się osiągnąć (rysunek 1.12).

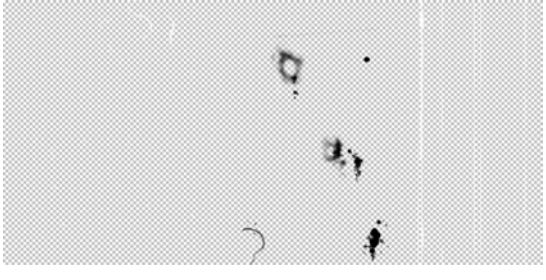


Rysunek 1.12. Płytkie zadrapania

Chociaż użyliśmy tylko koloru białego, wiele efektów na starych filmach jest czarnych i je również należałoby zastosować.

1. Zaznacz niektóre obszary białych pikseli za pomocą narzędzia *Zaznaczenie*.
2. Odwróć kolory zaznaczenia, klikając *Obrazek/Dostosuj/Odwrotność*. Może się wydawać, że zaznaczone piksele zniknęły — dzieje się tak, ponieważ utworzyliśmy właśnie czarne piksele, które znajdują się na czarnym tle (więc nie zniknęły, a są tylko niewidoczne).
3. Usuń warstwę tła (przeciągnij ją z zakładki *Warstwy* nad ikonę kosza w dolnej części zakładki).

Obraz powinien wyglądać mniej więcej tak, jak na rysunku 1.13 (tło w postaci szachownicy oznacza w Photoshopie kanał zero alfa, czyli brak pikseli).



Rysunek 1.13. Udawane dziury

Zapisz obrazek jako plik *PNG*. Nie jest konieczne dokonywanie jego optymalizacji.

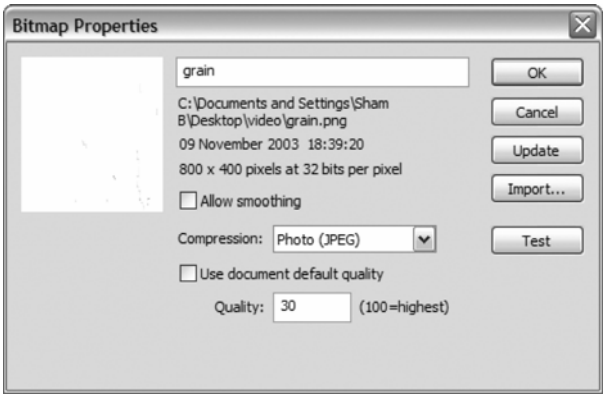
Na tym etapie wielu projektantów optymalizuje grafiki, o ile mają one zostać załadowane na stronę WWW wykonaną we Flashu. W naszym przypadku nie ma jednak takiej potrzeby. Poczekanie z optymalizacją do momentu utworzenia pliku *SWF* pozwala zachować większą elastyczność.

Jeśli na przykład klient zechce, aby przygotować wersję witryny dla łączy o dużej przepustowości, wystarczy zmienić w programie Flash ustawienia eksportu map bitowych. Jeśli bitmapy zostałyby zoptymalizowane przed zaimportowaniem do programu, konieczne byłoby ponowne otwarcie Photoshopa i wyeksportowanie obrazków z nowymi ustawieniami. Następnie należałoby zamienić nowymi obrazkami wszystkie instancje starych grafik na listwie czasowej Flasha. Jasno więc widać, że zaimportowanie do Flasha grafik pełnej jakości i późniejsze ich zoptymalizowanie pozwala zaoszczędzić trochę czasu. Użytkownicy, którzy wolą pracować w programie Fireworks, mogą wyciągnąć wiele korzyści ze sposobu, w jaki ten program integruje się z Flashem (tzn. z funkcji ładowania i edycji).

Obróbka animacji we Flashu

Po wyeksportowaniu z Photoshopa bitmapy w formacie *PNG* nadszedł czas na użycie Flasha.

1. Zaimportuj plik *PNG*, klikając *File/Import/Import to Library* (plik/importuj/importuj do biblioteki).
2. Wybierz bitmapę z biblioteki.
3. W panelu *Library* (biblioteka) kliknij tę bitmapę prawym przyciskiem myszy (Windows) lub przytrzymując klawisz ⌘ (Mac), a następnie na wyświetlonym menu kontekstowym (zwanym także wysuwanym menu opcji) kliknij polecenie *Properties* (właściwości).
4. Zmień właściwości mapy bitowej, tak jak na rysunku 1.14 — wybierz niski współczynnik kompresji *JPEG* i wyłącz opcję wygładzania (Flash szybciej przetwarza grafikę, jeśli ta opcja jest wyłączona).



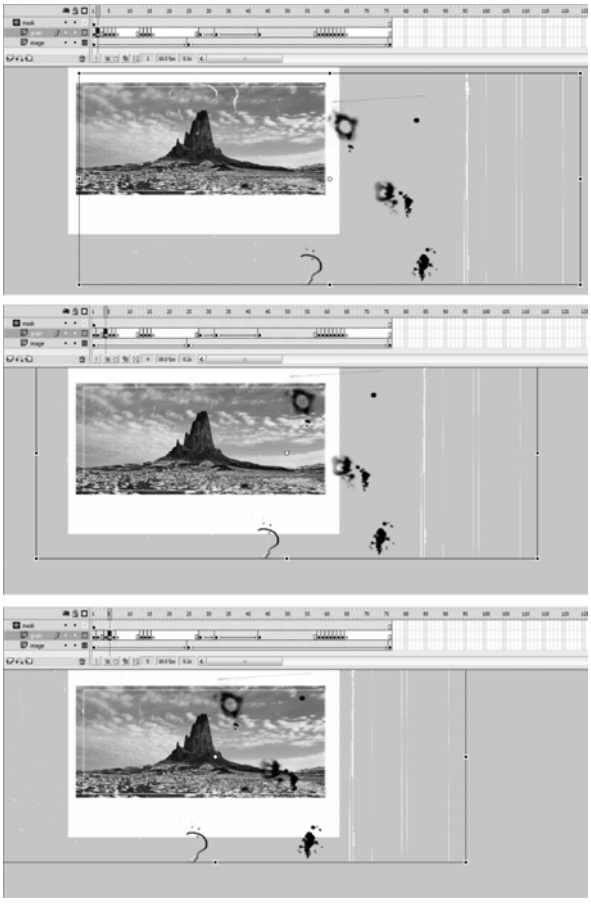
Rysunek 1.14. Właściwości bitmapy w bibliotece Flasha



Utworzyliśmy właśnie obrazek, który ma zarówno kompresję *JPEG*, jak i kanał alfa! Zazwyczaj nie jest możliwa praca nad grafiką w formacie *JPEG* ze skojarzonym kanałem alfa, ale Flashowi zdaje się to nie robić różnicy — fakt ten okazuje się bardzo przydatny, gdy zachodzi potrzeba zastąpienia grafiki wektorowej mapą bitową.

Przeciągnij obrazek na scenę, naciśnij klawisz *F8* (*Modify/Convert to Symbol*) i przekształć bitmapę w symbol klipu filmowego o nazwie *grain*.

Pozostaje tylko nałożyć szybko się poruszającą wersję naszego klipu na film wideo, bitmapę lub animację wektorową. Ja nałożyłem grafikę na statyczny obraz, dzięki czemu wygląda on, jakby był wycięty z taśmy filmowej (rysunek 1.15).



Rysunek 1.15. Efekt starego filmu, etapy 1 – 3

Wykorzystałem także maskę, aby ukryć te elementy stylizowanego klipu, które nie są widoczne na obrazku. Efekt jest widoczny na rysunku 1.16 (można również zapoznać się z plikiem *grain.fla*, który można pobrać ze strony poświęconej książce).



Rysunek 1.16. Użycie maski do wykończenia efektu

Uwagi końcowe

Zaprezentowana technika może nie tylko urozmaicić krótki klip, który nie ma dużego znaczenia, ale także:

- zamaskować niedoskonałości wideo (np. pikselację spowodowaną wysoką kompresją),
- ożywić statyczne obrazki, dzięki czemu będą one wydawać się ruchome,
- ukryć ewentualne przerwy. Jeśli łączone są pliki wideo lub nieruchome grafiki (np główna sekcja filmu i wektorowy wstęp lub napisy końcowe), można zatuszować połączenia, dodając do całości ten efekt.

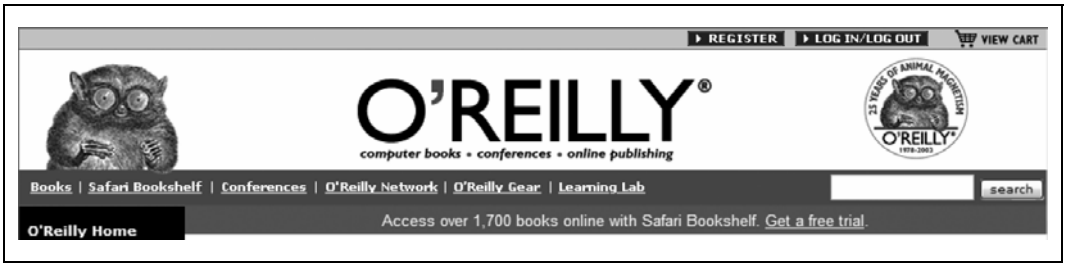


SPOSÓB 4.

Tworzenie plików SWF z animowanych obrazków GIF

Szybkie przekształcanie animacji GIF i wykorzystanie ich we Flashu.

Czytelników na pewno zainteresuje rozszerzenie możliwości plików *GIF* poprzez zaimportowanie ich do Flasha. Udałem się więc na stronę wydawnictwa O'Reilly (*http://www.oreilly.com*), na której powitała mnie mała futrzana istota — powiedziano mi, że jest to wyrak[1] (rysunek 1.17). Nazwa pliku, *oreilly_header1.gif*, jest typowa dla obrazków umieszczanych w tabelach *HTML*. Miałem więc plik *GIF*, nad którym mogłem popracować. Wyrak wyglądał tak sympatycznie, że przyglądałem mu się przez jakiś czas. Nagle mrugnąłem powiekami, a po chwili futrzak zrobił to samo. Zdziwiłem się i poczekałem chwilę, żeby zobaczyć to jeszcze raz — miałem więc do czynienia z animowanym obrazkiem *GIF*.



Rysunek 1.17. Maskotka wydawnictwa O'Reilly

Pomyślałem, że zrobienie flashowej wersji animowanego futrzaka byłoby dobrym sposobem przedstawienia różnic między Flashem a standardowym *HTML*-em. Ta sztuczka pokazuje, jak zredukować wielkość pliku, używając Flasha zamiast animowanych obrazków *GIF*. Gdy dwuwymiarowa animacja zostanie już zaimportowana do Flasha, można przekształcić ją w symulację trójwymiarowego mrugającego powiekami wyraka **[Sposób 35.]**.

[1] Zwany także tarsjuszem lub tarsjuszem wyrakiem (łac. *Tarsius syrichta*). Mała nadrzewna małpiatka prowadząca nocny tryb życia i żyjąca na wyspach południowo-wschodniej Azji. Gatunek objęty całkowitą ochroną — *przyp. tłum.*

Animowany obrazek GIF

Kopię naszego animowanego przyjaciela możemy uzyskać, wykorzystując opcję *Zapisz* przeglądarki. Na przykład w windowsowym programie Internet Explorer wystarczy kliknąć wyraka prawym przyciskiem myszy i z menu kontekstowego wybrać polecenie *Zapisz obraz jako*, aby pobrać obrazek na dysk twardy komputera.

Kolejną zaletą animacji wykonanych we Flashu jest zabezpieczenie pliku przed kradzieżą — nie jest on wtedy tak łatwo dostępny, jak plik *GIF*. Robi się to poprzez ukrycie pliku *SWF* **[Sposób 98.]** w pamięci podręcznej przeglądarki, gdzie użytkownicy najczęściej szukają pobranych z internetu plików wykonanych we Flashu.

Po otwarciu pliku *GIF* zamieszczonego w witrynie wydawnictwa O'Reilly w programie do edycji grafiki (Fireworks, Photoshop/ImageReady) można przekonać się, że animacja mrugnięcia jest uruchamiana co 12,5 sekundy (czas trwania pierwszej klatki bez mrugnięcia wynosi 12 sekund, zaś sama animacja mrugnięcia trwa pół sekundy), a całkowity czas jej trwania to również 12,5 sekundy. Warte uwagi jest także to, że większość pikseli składających się na obrazek — poza oczami — nie zmienia się w poszczególnych klatkach animacji. Dlatego też przekształcenie obrazka do formatu Flash powinno natychmiast zaowocować znacznym ograniczeniem wielkości pliku.

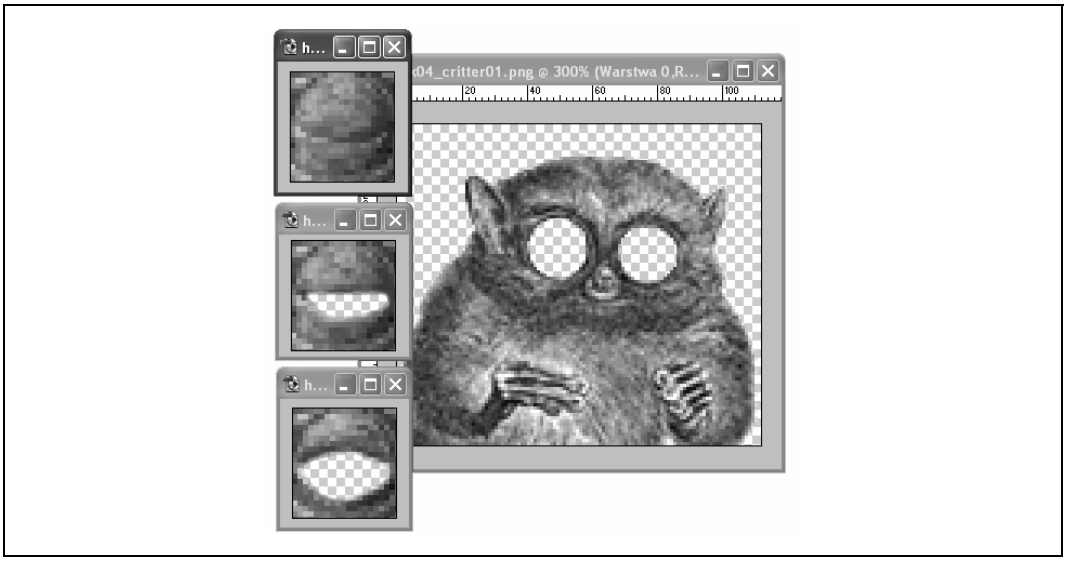
Trzeba też zauważyć, że animacja nie jest interaktywna. W tym przypadku podejście takie jest ze wszech miar odpowiednie — animacja nie powinna rozpraszać czytelników witryny. My jednak (dla zabawy) dodamy do niej funkcje interaktywne, a przy okazji przekonamy się, że mimo to plik *SWF* jest mniejszy niż plik *GIF*.

Tworzenie elementów animacji

Najlepszym sposobem zaimportowania wszystkich elementów pliku *GIF* jest pobranie ich jako serii obrazków *PNG*, ponieważ nie spowoduje to pogorszenia jakości oryginału (zwłaszcza jeśli zostanie wykorzystany format *PNG-24*) i łatwe będzie dodanie kanału alfa. Chociaż pliki *GIF* obsługują przezroczystość, w rzeczywistości zawierają jedynie maskę, określającą, które piksele są widoczne, a które nie. Format *PNG-24* obsługuje prawdziwy kanał alfa (ang. *true alpha*), gdzie przezroczystość jest określana w procentach. Na rysunku 1.18 pokazano obrazki *PNG* (w programie Photoshop) gotowe do wyeksportowania do Flasha.

Dla osiągnięcia efektu skorzystałem z kilku ciekawych sposobów:

- przyciąłem obrazek, aby oddzielić animowaną część (tzn. oczy) od reszty;
- chociaż oryginalna animacja trwa sześć klatek, trzy spośród nich są powtarzane (sekwencja zamykania oka jest po prostu odwrotną powtórką sekwencji otwierania) — można więc znacznie zmniejszyć liczbę pojedynczych klatek, które będzie trzeba odtworzyć we Flashu (tym samym zmniejszając wielkość pliku);



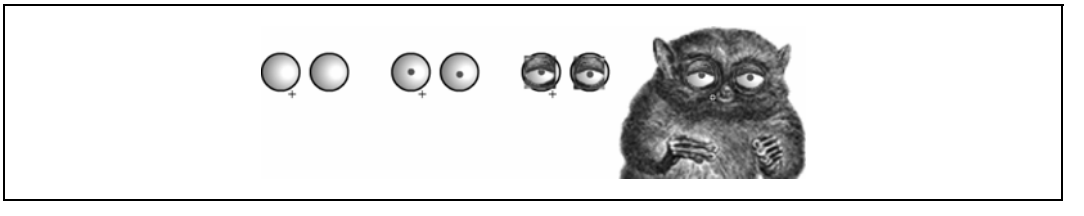
Rysunek 1.18. Animacja GIF w postaci plików PNG

- animację można wyeksportować w formacie *PNG-24* i z kanałem true alpha, co umożliwi wygładzenie konturów obrazka. Jeśli futrzak zostanie umieszczony na tle dżungli (poczułby się jak w domu), jego kontur wtopi się w tło, nie tworząc nierównych krawędzi lub poświaty, co ma zwykle miejsce przy standardowych plikach *GIF*.

Co zrobić, żeby zaimportowany do Photoshopa obrazek był przezroczysty?

1. Skopiuj warstwę *Background* na zakładce *Layers*.
2. Usuń oryginalną warstwę tła. Powinna pozostać jedna warstwa o nazwie *Background Layer Copy*.
3. Przy użyciu narzędzia *Eraser* (gumka) można utworzyć na tej warstwie kanał alfa. Warstwa *Background* jest już usunięta i po wymazaniu pikseli nie istnieje tło, które mogłoby przeświwitywać przez wytarte miejsca. Zamiast tego w programie powstaje efekt przezroczystości.

Jeśli obrazki *PNG* zostaną zaimportowane do Flasha (za pomocą polecenia *Import* w menu *File*), możliwe będzie odbudowanie obrazka maskotki w zwyczajowy sposób — począwszy od najniższej warstwy, a skończywszy na najwyższej: po kolei gałki oczne, źrenice, powieki i wreszcie ciało (rysunek 1.19).



Rysunek 1.19. Gałki, źrenice, powieki i ciało

Proszę zwrócić uwagę, że w tej wersji pliku gałki oczne są grafiką wektorową. Sekwencja mrugania oczami jest klipem filmowym składającym się z trzech map bitowych z oczami zamykającymi się stopniowo i trzech bitmap, na których sytuacja jest odwrócona i oczy się otwierają. Źrenice to dwa punkty o nazwach `leftEye_mc` i `righEye_mc`. Po zastosowaniu poniższego skryptu źrenice maskotki będą podążać za wskaźnikiem myszy (rysunek 1.20). Kod ten można standardowo osadzić w pierwszej klatce warstwy *actions* na głównej listwie czasowej.



Rysunek 1.20. Śledzenie kursora

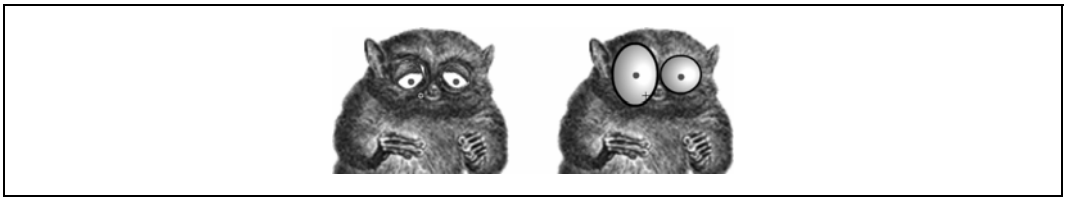
```
followMouse = function () {
  this.startX = this._x;
  this.startY = this._y;
  this.onEnterFrame = animateEye;
};

animate eye = funtion () {
  var distX = (_xmouse - this._x) / 50;
  var distY = (_ymouse - this._y) / 50;
  if (Math.abs(distX) < 5) {
    this._x = this.startX+distX;
  }
}:

  leftEye_mc.onEnterFrame = followMouse;
  rightEye_mc.onEnterFrame = followMouse;
```

Funkcja *followMouse()* została określona dla obydwu oczu jako uchwyt zdarzenia *onEnterFrame*. Po wywołaniu wczytuje pozycję startową źrenic i zmienia uchwyt zdarzenia *onEnterFrame* na *animateEye()* w celu wyświetlenia następnych klatek. Funkcja porusza źrenicami o określoną odległość od pozycji wyjściowej, opierając się na położeniu kursora myszy — dzięki temu maskotka zdaje się śledzić wzrokiem kursor myszy.

Oczywiście maskotka może być dużo bardziej wyrazista niż animowany obrazek *GIF* (jak na rysunku 1.21). Nie jest to jednak żadna sztuczka — na tym właśnie polega Flash.



Rysunek 1.21. Wyrak melancholik i wyrak alkoholik

Uwagi końcowe

Choć jest to raczej standardowa animacja, szybkość, z jaką została przeniesiona z istniejącego pliku *GIF*, pokazuje, jak łatwe jest tworzenie we Flashu ciekawych animacji. Bitmapy z przezroczystością można nakładać na siebie w celu utworzenia wielowarstwowych animacji przy wykorzystaniu animowanych obrazków *GIF*. Jest to także dobry sposób eksportowania plików *PSD* wykonanych w Photoshopie do Flasha — wystarczy wyeksportować każdą warstwę jako oddzielny plik *PNG-24*, w który będzie wbudowana przezroczystość. Następnie można ułożyć pliki *PNG* w określonej kolejności, posługując się warstwami Flasha. Jedyna (oczywista) różnica polega na tym, że można potem przystąpić do animowania warstw!



SPOSÓB 5.

Animowanie plików PSD we Flashu

Import plików wykonanych w programie Photoshop i animowanie ich we Flashu.

Tym razem zajmiemy się odtwarzaniem we Flashu wielowarstwowych plików wykonanych w programie Photoshop. Omówimy dokładnie wszystkie szczegóły, ponieważ są one bardzo pouczające (i nic nas to nie kosztuje!). Sprawę ułatwi znacznie dodatek do Photoshopa o nazwie *PSD2FLA* (*http://www.medialab.com/psd2fla*) opracowany przez firmę Media Lab. Programiści pracujący z programem Director znają firmę Media Lab jako twórcę PhotoCastera, cenionego i popularnego narzędzia umożliwiającego importowanie plików *PSD* do programu Director.

Jeśli w systemie zainstalowany jest program QuickTime 4.0 lub nowszy, możliwe jest zaimportowanie pliku *PSD* bezpośrednio do Flasha. Flash najprawdopodobniej wyświetli informację, że nie może zaimportować pliku, ale umożliwi wybór opcji importu za pośrednictwem programu QuickTime. Po kliknięciu przycisku potwierdzenia obrazek zostanie zaimportowany do programu.

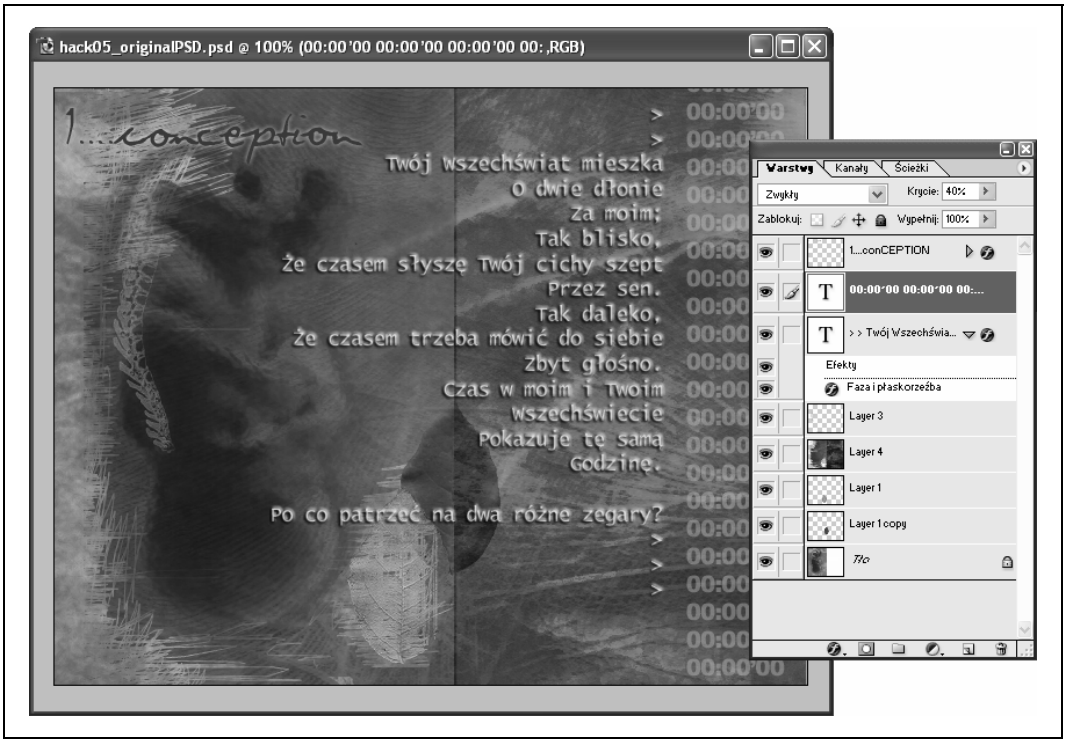


Narzędzie importu plików w programie Flash rozpozna plik *.psd* jako dokument programu Photoshop w wersji 2.5 lub 3. Jednak przy wykorzystaniu QuickTime'a Flash może obsługiwać pliki wykonane w nowszych wersjach Photoshopa.

Z importowaniem plików za pomocą programu QuickTime związany jest pewien problem — plik zostaje spłaszczony, uniemożliwiając dostęp do oddzielnych warstw (rysunek 1.22). Zakładamy, że jedynym powodem zaimportowania pliku *PSD* (w przeciwieństwie do takich formatów wykorzystywanych na stronach WWW, jak *JPEG*) jest uzyskanie dokładnych informacji o jego warstwach, co powoduje, że QuickTime poprzez swoje ograniczenie nie jest narzędziem do końca idealnym.

W tym ćwiczeniu zajmiemy się zaimportowaniem pliku *PSD* wraz ze wszystkimi istotnymi informacjami o warstwach, aby możliwe było edytowanie go i animowanie we Flashu.

Zacznijmy od przycięcia obrazka *PSD* lub zmiany jego wymiarów w Photoshopie, aby miał on wymiary odpowiednie do umieszczenia na stronie WWW (czyli nie był większy niż 500×500 pikseli).



Rysunek 1.22. Photoshop: obrazek z warstwami

Lepsze efekty uzyska się, zmniejszając rozmiary pliku Photoshopa stopniowo. Na przykład dzięki krokowym zmianom z 1000×1000 do 900×900, 800×800 aż do 500×500 pikseli obrazek będzie miał lepszą jakość i dostępnych będzie więcej opcji podczas późniejszej kompresji.

Kolejnym powodem zmniejszenia rozmiarów obrazka jest to, że Flash nie został zaprojektowany do obsługi bitmap w taki sposób, w jaki zamierzamy ich użyć — jest przede wszystkim narzędziem do obróbki grafiki wektorowej. Można oczywiście obejść to ograniczenie, jeśli ograniczymy rozmiary map bitowych, a Flash nie będzie musiał dokonywać zbyt wielu zmian na ekranie w każdej klatce.

Następnie należy połączyć (spłaszczyć) możliwie wiele warstw, aby maksymalnie zredukować ich liczbę. Flash będzie działał naprawdę sprawnie, jeśli uda się uzyskać maksymalnie 5 – 6 warstw. Należy także zastanowić się nad usunięciem warstw tekstu lub innych warstw, które można wykonać we Flashu (napisy można odtworzyć za pomocą narzędzi do rysowania i pisania).

Zdecydowałem się usunąć wszystkie napisy poza głównym nagłówkiem (górny lewy róg). Usunięty tekst można zastąpić wyraźniejszym tekstem wektorowym podczas obrabiania obrazka we Flashu. Postanowiłem zachować główny nagłówek bez zmian, ponieważ

zastosowano w nim efekty dostępne w programie Photoshop, których nie uda się łatwo odtworzyć narzędziem do tworzenia grafiki wektorowej. Rysunek 1.23 przedstawia uproszczoną wersję grafiki z usuniętymi napisami.



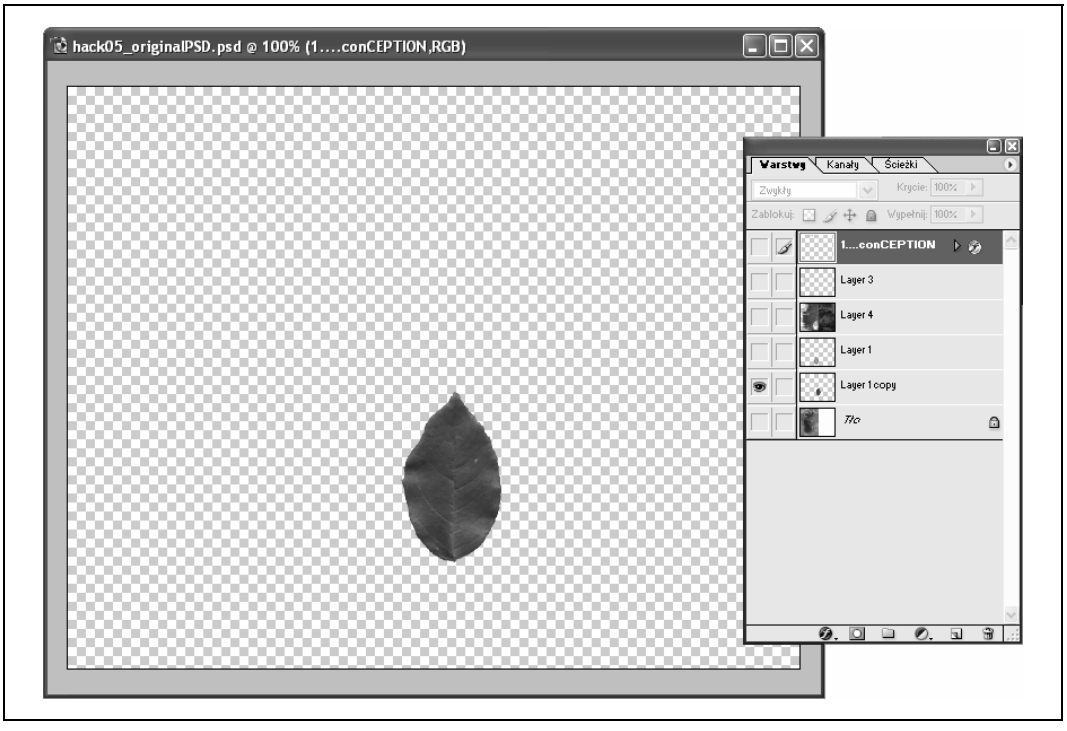
Rysunek 1.23. Obrazek wykonany w programie Photoshop bez warstw, które można odtworzyć we Flashu

Następnym etapem jest wyeksportowanie każdej warstwy jako pliku *PNG*. Pracując w Photoshopie, należy:

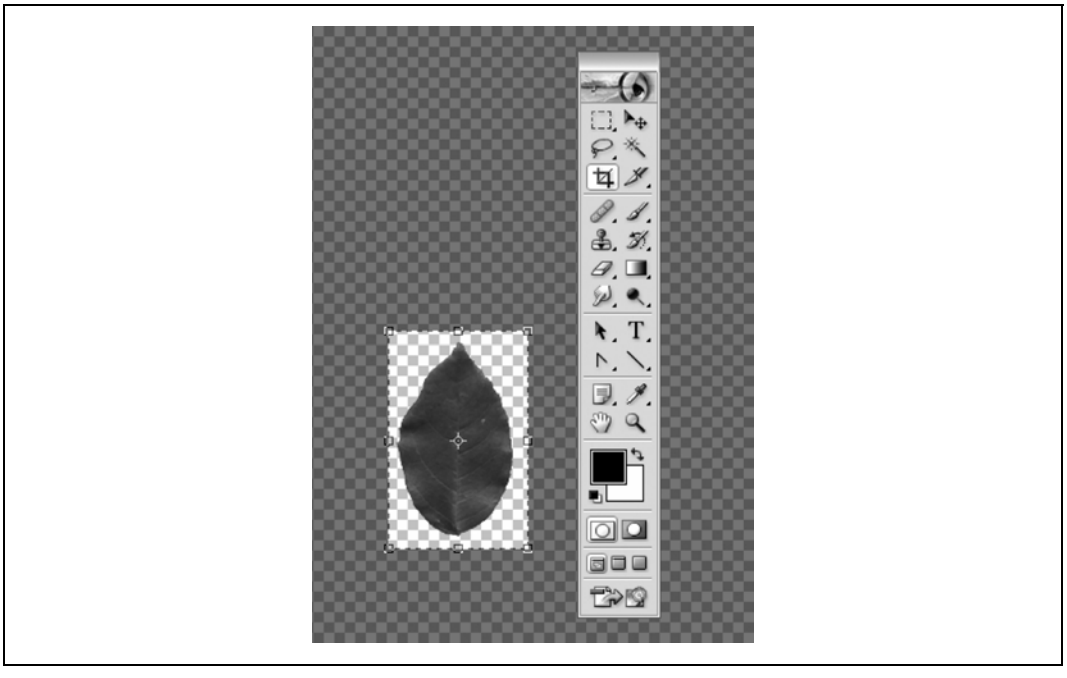
1. Ukryć wszystkie warstwy poza tą, która ma zostać wyeksportowana.
2. Zapisać plik w formacie *PNG*, klikając polecenie *Zapisz jako* w menu *Plik*.
3. Po wyeksportowaniu wszystkich warstw ponownie załadować pliki *PNG*, na których jest dużo wolnego miejsca — na przykład jak wokół liścia na rysunku 1.24 — i przyciąć je, aby pozbyć się zbędnych pikseli alfa (rysunek 1.25).

Wróćmy teraz do oryginalnego pliku *PSD*, nie zamykając Photoshopa — można wykonać zrzut obrazka, jeśli nie dysponuje się wystarczającą ilością pamięci RAM, umożliwiającą jednoczesne działanie obydwu programów. Obrazek będzie jeszcze potrzebny.

We Flashu określ takie same rozmiary sceny, jakie ma grafika *PSD*, i zaimportuj wszystkie pliki *PNG* do biblioteki, klikając *File/Import/Import to Library* (Flash MX 2004) lub *File/Import to Library* (Flash MX).



Rysunek 1.24. Warstwa, która aż się prosi o przycięcie

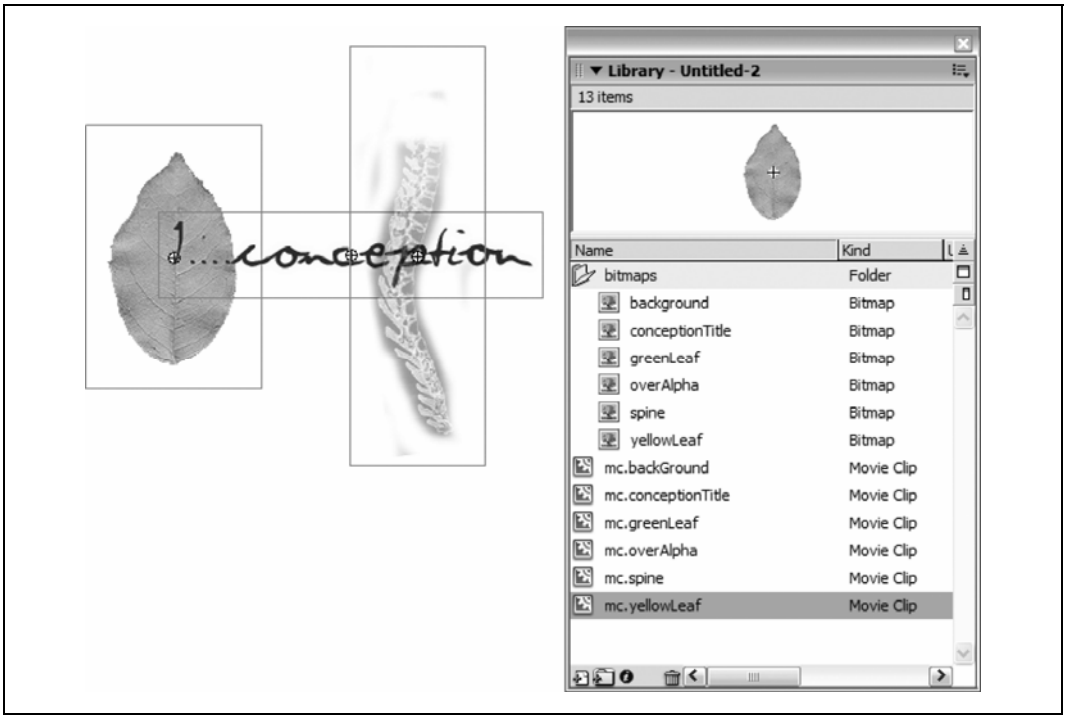


Rysunek 1.25. Przycięty obrazek

Aby animowanie bitmap było wygodne, zalecane jest przekształcenie ich w klipy filmowe (tzn. umieszczenie ich w klipach filmowych, ponieważ można później korzystać z metod i efektów animacyjnych dostępnych dla klipów):

1. Przeciągnij po kolei każdą bitmapę z biblioteki na scenę.
2. Zaznacz bitmapę.
3. Naciśnij klawisz *F8*, aby utworzyć symbol klipu filmowego. Dla wygody dobrze jest zachować nazwy obrazków, ale dodać do nich przyrostek `_mc`.

Mamy więc teraz kilka bitmap o przezroczystych tłach, które można umieszczać na scenie Flasha tak samo, jak warstwy w oryginalnych obrazkach *PSD*. Na rysunku 1.26 pokazano mapy bitowe w bibliotece Flasha.



Rysunek 1.26. Import map bitowych do biblioteki

Umieść ręcznie klipy filmowe na listwie czasowej (lub w innym klipie — w zależności od tego, jak będą prezentowane) w tym samym porządku i położeniu, co w oryginalnym pliku *PSD*. Efekty dostępne w programie Photoshop (np. rozjaśnienie, ściemnienie itp.) można emulować we Flashu, posługując się efektami kolorystycznymi. Oczywiście można wprowadzić zmiany, aby flashowa wersja przeznaczona na stronę WWW (rysunek 1.27) różniła się od wersji drukowanej wykonanej w Photoshopie.



Rysunek 1.27. Odtwarzanie kompozycji we Flashu

Na koniec należy jeszcze zoptymalizować każdą bitmapę z osobna, klikając ją prawym przyciskiem myszy (Windows) lub przytrzymując klawisz ⌘ (Mac) w bibliotece i określając odpowiednie ustawienia eksportu. Ustawienia zapewniające największą równowagę między wielkością pliku a jakością obrazu to zazwyczaj:

- fotografia (*JPEG*),
- bez wygładzania,
- jakość w granicach 30 – 50%.

Określając jakość każdej bitmapy, trzeba mieć na uwadze to, że obrazki stanowią część wielowarstwowej kompozycji i wiele niedoskonałości spowodowanych przez wysoki współczynnik kompresji zostanie zamaskowanych przez warstwy i przezroczystość. Dlatego też należy przyjrzeć się całości i ocenić, jaka jakość jest zadowalająca. Grafiki na niższych warstwach skompresowane na poziomie 20% mogą wyglądać całkiem dobrze.



We Flashu możliwe jest wyeksportowanie bitmapy z kanałem alfa w momencie zastosowania kompresji JPEG!

Wszystko jest już przygotowane do tworzenia animacji. Każdy element kompozycji można teraz:

- animować przy wykorzystaniu animacji uzupełnianej (określenie wartości *Motion* dla opcji *Tween* w panelu właściwości),
- animować dynamicznie za pomocą kodów ActionScript po uprzednim nadaniu nazwy instancji.

Można także dodać napisy i obrazki wektorowe analogiczne do tych, które istniały w pierwotnej wersji (lub wzbogacić projekt o nowe elementy wektorowe).

Jak już wspominałem wcześniej, Flash może działać nieco ociężale podczas animowania map bitowych, choć doświadczenie pokazuje, że można osiągnąć dobre efekty, jeśli w określonym czasie animowana jest tylko jedna warstwa lub jeśli rozmiary elementów są niewielkie i niewiele zmienia się w każdej klatce.

Uwagi końcowe

Póki planowane animacje nie wpływają znacząco na wydajność procesora lub nie wymagają zastosowania dużych grafik (należałoby wtedy pomyśleć o użyciu programu Director), można skorzystać z przedstawionego w tym rozdziale sposobu przekształcenia plików *PSD* w obrazki *PNG* i zaimportowania ich do Flasha. Sposób ten okaże się także pomocny, jeśli witryna ma zostać urozmaicona o wykonane w Photoshopie wizualizacje lub dokumenty przygotowane do druku. Można wtedy posłużyć się narzędziem autorów *PSD2FLA* — programem PhotoWebber (*http://www.photowebber.com*).

Bazując na swoim doświadczeniu, mogę powiedzieć, że wielkość ostatecznego pliku *SWF* może być znacznie mniejsza — zwłaszcza w porównaniu z oryginalnym plikiem *PSD*. Zazwyczaj plik *SWF* ma taką samą wielkość, jak statyczny obrazek *JPEG* średniej lub wysokiej jakości.



SPOSÓB 6.

Drzewko na Brooklynie[2]

Tworzenie generatora drzew.

W tym ćwiczeniu za pomocą standardowych metod (fraktali, rekurencji, algorytmu powtarzania i skalowania) utworzymy wyglądające naturalnie drzewo. Następna sztuczka będzie polegać na utworzeniu animacji przy wykorzystaniu wbudowanej hierarchii klipów filmowych. Mówiąc po ludzku: chodzi o to, że zasadzimy drzewko, które będzie kołysać gałęziami na wietrze **[Sposób 7.]**. Zrobimy to, naśladując w kodzie zjawiska natury.

Na pierwszej konferencji Flash Forward (*http://www.flashforward2004.com*), w której uczestniczyłem, Josh Davis opowiadał o swoich inspiracjach. Wykład trwał 45 minut, ale jego sens zawiera się w jednym zdaniu: „Przyjrzyjcie się naturze — zobaczcie, czym was obdarowuje i jak można to wykorzystać."

[2] Tytuł książki Betty Smith (*A Tree Grows in Brooklyn*, 1943) przeniesionej na duży ekran przez Elię Kazana w 1945 roku — *przyp. tłum.*

W internecie można znaleźć wiele tego rodzaju eksperymentów, a w książce takiej, jak ta nie możemy przejść obok nich obojętnie.

Fraktale

Aby zdobyć potrzebne informacje przyrodnicze, przeprowadziłem krótką rozmowę z moją dziewczyną, Karen. Obowiązuje nas jasny podział obowiązków: ona zajmuje się ogrodem, ja — komputerem.

Oto, czego się dowiedziałem, nie wystawiając nosa za drzwi. Drzewa rosną według prostego, zazwyczaj regularnego wzorca. Gałąź jest prosta do określonego punktu, później odłączają się od niej inne gałęzie. Grubość głównej gałęzi jest zazwyczaj proporcjonalna do grubości gałęzi, które z niej wyrosły — zazwyczaj obwód jest porównywalny (całkowita grubość pnia odpowiada w takim samym lub zbliżonym stopniu grubości gałęzi, wypuszczonych z tego pnia). Oznacza to, że młoda gałązka rośnie i rozwija się zupełnie tak samo, jak główna gałąź: wymiary są porównywalne. Wiadomo, że istnieje zależność między drzewem a gałęzią, ponieważ jeśli zasadzi się sadzonkę (a raczej: jeśli Karen to zrobi; moje zawsze usychają), to wyrośnie drzewo.

Uzbrojony w tę wiedzę, zabrałem się do opracowania generatora drzew. Dwa przykładowe wyniki moich prac można zobaczyć na rysunku 1.28.



Rysunek 1.28. Chyba nigdy nie zobaczę ładniejszego zrzutu ekranu

Obydwa drzewa zostały utworzone przy wykorzystaniu tego samego kodu. Oto kod z pliku *treeGen.fla*, który można pobrać ze strony poświęconej książce:

```
function counter() {
  if (branchCounter == undefined) {
    branchCounter = 0;
  }
  return (branchCounter++);
}

function grow() {
  //Wypuść gałąź…
  this.lineStyle(trunkThickness, 0×0, 100);
  this.moveTo(0.0);
  this.lineTo(0, trunkLenth);
  //Jeśli to nie jest pień, zmień kąt i grubość gałęzi.
  if (this._name ! = "trunk") {
  this._rotation = (Math.random()*angle – angle/2;
  this._xscale *= branchSize;
  this._yscale *= branchSize;
}
//Puść pędy…
var seed = math.ceil(Math.random()*branch);
for (var i = 0; i < seed; i++) {
  if (counter() < 3000) {
  var segment = this.createEmptyMovieClip("segment" + i, i);
  segment.onEnterFrame = grow;
  segment._y = trunkLength;    }
  }
  delete (this.onEnterFrame);
}

//Zdefiniuj położenie pnia i ustaw uchwyt onEnterFrame na grow().
this.createEmptyMovieClip("trunk", 0);
trunk._x = 200;
trunk._y = 400;
trunk.onEnterFrame = grow;

//Parametry drzewa
var angle = 100;
var branch = 5;
var trunkThickness = 8;
var trunkLength = -100;
var branchSize = 0,7;
```

Podstawowy kształt drzewa jest zdefiniowany przez parametry w ostatnich kilku liniach kodu:

`Angle` (kąt)
: Maksymalny kąt tworzony przez małą i dużą gałąź.

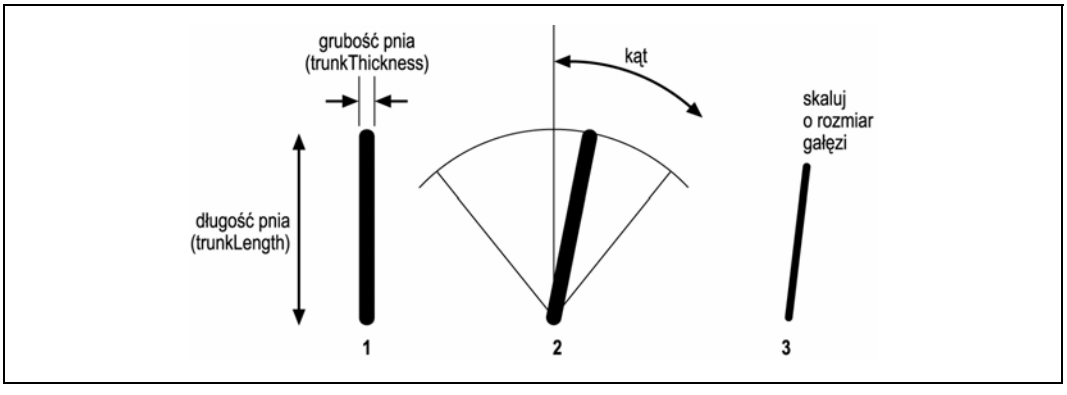
`Branch` (gałąź)
: Maksymalna liczba pędów (młodych gałązek), jakie może mieć jedna gałąź.

`TrunkThickness` (grubość pnia)
: Grubość pnia drzewa.

`TrunkLength` (długość pnia)
Długość pnia drzewa.

`BranchSize` (rozmiar gałęzi)
Stosunek gałęzi-córki do gałęzi-matki (im dalej od pnia, tym mniejsze stają się gałęzie).

Najpierw zostaje utworzony pień i określane jest jego położenie. Następnie podłączana jest do niego funkcja *grow()* jako uchwyt zdarzenia *onEnterFrame*. Jak wskazuje nazwa, funkcja *grow()* — rośnij — sprawia, że klip filmowy „rośnie" z dwóch powodów. Po pierwsze tworzona jest gałąź poprzez narysowanie pionowej linii o długości `trunkLength` i grubości `trunkThickness`. Jeśli rysowana gałąź jest pniem drzewa, zostawiamy ją bez zmian, i efekt będzie taki, jak w scenie 1. Jeśli nie rysujemy pnia, obracamy gałąź o kąt `+/- angle` — jak w scenie 2. — i skalujemy o rozmiar `branchSize` — jak w scenie 3. (wszystkie etapy uwidoczniono na rysunku 1.29).



Rysunek 1.29. Rośnij maleńkie, rośnij

Następnie utworzone zostają nowe gałązki między `1` a `branch`. Najciekawsze jest to, że do pędów przypisany jest ten sam uchwyt zdarzenia *onEnterFrame*, co do bieżącej gałęzi — *grow()* — i w następnej klatce pędy wypuszczą kolejne pędy, i tak dalej, i tak dalej. Oto fragment funkcji *grow()*, który tworzy nowy klip filmowy dla każdego pędu i przypisuje do niego uchwyt zdarzenia *onEnterFrame*. Drzewo mogłoby wypuszczać nowe gałęzie bez końca, ale musimy przecież ustalić jakiś koniec. W przeciwnym razie Flash będzie pracować coraz wolniej, aż w końcu przestanie odpowiadać. Do tego właśnie służy funkcja *counter()* — ogranicza ona całkowitą liczbę gałęzi do 3000.

```
var seed = Math.ceil(Math.random()*branch);
for (var i = 0; i < seed; i++) {
  if (counter() < 3000) {
    var segment = this.createEmptyMovieClip("segment" + i, i);
  segment.onEnterFrame = grow;
  segment._y = trunkLength; }
}
```

Na koniec funkcja *grow()* usuwa się sama, ponieważ wystarczy, że zostanie uruchomiona raz dla jednej gałęzi.

Do utworzenia fraktalowego drzewa wykorzystywana jest więc funkcja samowywołująca (lub raczej funkcja tworząca kopie gałęzi, do których przypisana jest taka sama funkcja). W rezultacie powstaje nie tylko drzewo zbudowane z gałęzi i podgałęzi, ale hierarchia ta zostaje odzwierciedlona na listwach czasowych klipów filmowych. Można przyjrzeć się temu zjawisku, posługując się debugerem (choć wtedy konieczne będzie określenie maksymalnej liczby gałęzi poniżej 3000, chyba że komuś nie zależy na czasie!).

Otrzymana w rezultacie grafika, pomimo prostoty, ma nieco orientalny charakter. Efekt nie zawiera jednak animacji, a statycznych generatorów Java jest bardzo wiele. Animacją zajmiemy się w następnym ćwiczeniu.



SPOSÓB 7.

Gałęzie kołysane wiatrem: imitacja ruchu drzewa

Tworzenie filmu przy wykorzystaniu wbudowanych klipów filmowych.

W poprzednim ćwiczeniu dowiedzieliśmy się, jak losowo wygenerować drzewo, a teraz przejdziemy do jego animacji. Gdy drzewa poruszają się, zdają się przestrzegać tego samego porządku gałąź-pęd, który wykorzystaliśmy do wygenerowania drzewa. Dojście do poniższych wniosków zajęło mi trochę czasu, choć wydają się one zupełnie oczywiste:

- gdy porusza się pień, porusza się całe drzewo,
- gdy porusza się gałąź, poruszają się wszystkie odgałęzienia.

Znacznie trudniej jest jednak opracować właściwy zakres ruchu dla poszczególnych części drzewa. Początkowo wydawało mi się, że pień średniej wielkości drzewa odchyla się na wietrze w znacznie mniejszym stopniu, niż drobne gałązki w koronie, lecz okazało się, że to nieprawda. Aby się o tym przekonać, wystarczy wetknąć w ziemię obok drzewa średniej grubości gałązkę (pozbawioną wszelkich odgałęzień) i przyjrzeć się jej w wietrzny dzień. Okaże się, że gałąź wcale nie wychyla się u swego szczytu bardziej, niż pień drzewa. Gałązka jest wprawdzie słabsza i bardziej elastyczna, lecz jej powierzchnia (zakładając, że brutalnie pozbawiliśmy ją liści) jest znacznie mniejsza niż powierzchnia drzewa, co w znaczący sposób osłabia działanie siły wiatru.

Gałązka na samej górze porusza się energiczniej niż pień tylko dlatego, że poruszają się także wszystkie gałęzie stojące od niej wyżej w hierarchii (tzn. grubsze od niej) i ich ruch zostaje skumulowany, gdy porusza się drzewo. Widzicie? Codziennie można nauczyć się czegoś nowego.

To odkrycie dnia ułatwi nam znacznie dodanie efektu wiatru — wystarczy potraktować każdą gałąź tak samo, jak pień i inne gałęzie!

Zamiast usuwania każdego uchwytu *onEnterFrame* (jak zrobiliśmy to wcześniej podczas tworzenia drzewa **[Sposób 5.]**), możemy zmienić wiersz kodu, który usuwał funkcję *grow()*:

```
delete (this.onEnterFrame);
```

— i zastąpić go funkcją *sway()*, gdy gałąź już wyrośnie:

```
this.onEnterFrame = sway;
```

Aby utworzyć efekt wiatru, należy po prostu napisać funkcję *sway()*, która określa wartość `wind` (wiatr) dodawaną do każdej gałęzi w każdej klatce animacji za pomocą tejże funkcji:

```
function sway() {
  this._rotation += wind;
}
```

Konieczne jest urozmaicenie wartości kołysania. Można to zrobić następująco:

```
function sway()
  wind +=windEffect;
  if (wind > windStrength) {
    wind = -wind;
  }
  this._rotation += wind;
}
```

Oczywiście należy określić wartości początkowe parametrów wiatru. Oto ostateczne wartości, które zastosowałem w tym przykładzie (nowe parametry zaznaczone są tłustym drukiem):

```
//Parametry drzewa
var angle = 100;
var branch = 5;
var trunkThickness = 8;
var trunkLength = -100;
var branchSize = 0.7;
//Parametry wiatru
var windEffect = 0.05;
var windStrength = 1;
var wind = 0;
```

Do animacji można dodać suwaki, aby użytkownik mógł ręcznie regulować natężenie wiatru **[Sposób 61.]**.

Na rysunku 1.30 przedstawiono drzewo kołyszące się na wietrze..



Rysunek 1.30. Drzewo na wietrze

Uwagi końcowe

Chociaż w kodzie wykorzystanych zostało kilka sztuczek, to największa jego zaletą jest zastosowany sposób myślenia. Kopiowanie natury i otaczającej nas rzeczywistości jest sprawdzoną metodą odkrywania nowych możliwości Flasha.

Ponieważ program ten jest graficznym środowiskiem programistycznym, po napisaniu kodu można natychmiast zapoznać się z efektem graficznym. Jeśli postanowicie „zasadzić" więcej niż jedno drzewo, sprawdźcie, czy efekt jest realistyczny, gdy wiatr wieje z różnym natężeniem. Wydaje mi się, że dzięki drobnym nierównościom całość będzie bliższa rzeczywistości — prawdziwy wiatr wiejący z określonego kierunku najmocniej atakuje drzewa skrajne, które wyhamowują jego siłę. Kolejne drzewa poruszają się już mniej energicznie.

Moją muzą nie jest jednak natura, ale gry wideo. Myślę, że wszystkie problemy, przed jakimi stają projektanci animacji, zostały już rozwiązane przez zespoły zajmujące się produkcją gier wideo. To już jednak zupełnie inna bajka…