

Radostaw Kamysz

FLASH I ACTIONSCRIPT

Aplikacje 3D od podstaw



Helion



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

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

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

Redaktor prowadzący: Michał Mrowiec

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

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

Kody źródłowe wybranych przykładów dostępne są pod adresem:
<ftp://ftp.helion.pl/przyklady/flacpo.zip>

ISBN: 978-83-246-3065-3

Copyright © Helion 2013

Printed in Poland.

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

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

Spis treści

Rozdział 1. Wprowadzenie	13
Dostępne biblioteki 3D	14
Away3D	14
Alternativa3D	16
Papervision3D	18
FIVE3D	20
Flare3D	21
Sandy3D	23
Sophie3D	24
Wybór biblioteki 3D	25
Pobieranie silnika Away3D	26
Away3D FP9, FP10 czy może FP11?	27
Pobieranie Away3D 3.6.0 z Away3D.com	27
Pobieranie Away3D 3.6.0 z github.com	28
Pobieranie Away3D 3.6.0 z SVN	28
Instalacja biblioteki Away3D	31
Konfiguracja w Adobe Flash CS4 i CS5	31
Konfiguracja we FlashDevelop	32
Konfiguracja w Adobe Flash Builder 4	35
Podsumowanie	37
Rozdział 2. Podstawy biblioteki Away3D	39
Podstawowe komponenty	39
View3D	40
Scene3D	43
Camera3D	45
Object3D	46
ObjectContainer3D	49
Podstawowa budowa aplikacji w Away3D	50
Położenie obiektów w przestrzeni	53
Układ współrzędnych w Away3D	54
Podsumowanie	59

Rozdział 3. Obiekty	61
Base	62
Vertex	62
Mesh	65
Segment	68
Face	72
Sprites	76
Klasa bazowa	77
Sprite3D	81
MovieClipSprite	83
DirectionalSprite	85
Primitives	87
Przykład	88
AbstractPrimitive	94
LineSegment	100
Trident	101
Triangle	102
Plane	104
GridPlane	106
RegularPolygon	107
Cube	109
RoundedCube	111
Sphere	112
GeodesicSphere	114
Torus	115
TorusKnot	116
Cylinder	118
Cone	120
SeaTurtle	122
Arrow	123
WirePlane	125
WireCube	126
WireCone	128
WireCylinder	129
WireSphere	130
WireRegularPolygon	132
WireTorus	133
Skybox	134
Skybox	135
Skybox6	138
Podsumowanie	139

Rozdział 4. Materiały	143
Przygotowanie zasobów dla aplikacji	144
Dodawanie plików do zasobów w programie Adobe Flash	144
Odwoływanie się do zasobów w kodzie źródłowym aplikacji	147
Przykład	150
Linie	164
WireframeMaterial	164
WireColorMaterial	165
Kolor	167
ColorMaterial	167
EnviroColorMaterial	168
Bitmapy	170
BitmapMaterial	170
BitmapFileMaterial	172
EnviroBitmapMaterial	174
GlassMaterial	175
TransformBitmapMaterial	178
Animowane	181
MovieMaterial	181
AnimatedBitmapMaterial	183
VideoMaterial	185
Mieszanie materiałów	187
Podsumowanie	188
Rozdział 5. Światło	191
Rodzaje świateł	192
Klasa bazowa dla przykładów	192
AmbientLight3D	194
PointLight3D	196
DirectionalLight3D	199
Materiały reagujące na światło	202
Klasa bazowa dla przykładów	202
PhongColorMaterial	204
ShadingColorMaterial	206
PhongBitmapMaterial	207
WhiteShadingBitmapMaterial	209
PhongMovieMaterial	210
Dot3BitmapMaterial	212
Dot3BitmapMaterialF10	214
Dot3MovieMaterial	216
PhongMultiPassMaterial	218
Podsumowanie	220

Rozdział 6. Modele i animacje	223
3ds Max	224
Maya	224
LightWave	225
Blender	226
MilkShape 3D	227
CharacterFX	228
Google SketchUp	229
PreFab3D	230
Low poly i High poly	231
Format plików obsługiwanych w Away3D	232
3DS	232
ASE	232
OBJ	232
DAE	232
MD2	233
AWD	233
AS	233
Konwertowanie modelu do klasy ActionScript	233
Klasa AS3Exporter	233
Eksport z programu PreFab3D	241
Eksport z programu Blender	244
Eksport z programu Autodesk 3ds Max	246
Stosowanie modeli 3D w Away3D	248
Przykładowa aplikacja	248
3DS	263
OBJ	266
ASE	268
DAE	269
MD2	271
AWD	273
Stosowanie animacji modeli	275
AnimationData i AnimationDataType	275
AnimationLibrary	277
Animator i AnimatorEvent	277
Zastosowanie animacji w przykładzie	279
Podsumowanie	280

Rozdział 7. Praca z obiektami	283
Biblioteki do animacji	284
Pobieranie i instalacja biblioteki GreenSock Tweening Platform	284
Wybór biblioteki GreenSock Tweening Platform	285
Implementacja TweenMax	285
Klasa bazowa	286
Przemieszczanie	294
Właściwości	294
Metody	295
Obracanie	301
Właściwości	302
Metody	303
Skalowanie	309
Właściwości	309
Metody	310
Macierz transformacji	311
Czym jest macierz transformacji?	311
Tworzenie macierzy transformacji	312
Modyfikowanie powierzchni obiektu	313
PathExtrusion	313
HeightMapModifier	319
Elevation i SkinExtrude	325
Explode i Merge	331
Podsumowanie	337
Rozdział 8. Interaktywność	339
Używanie klawiatury	340
Klasa KeyboardEvent	340
Klasa Keyboard	341
Klasa KeyLocation	343
Sterowanie statkiem kosmicznym w przestrzeni	344
Używanie myszy	352
MouseEvent	352
Obracanie i skalowanie statku kosmicznego	353
MouseEvent3D	359
Metody dla zdarzeń MouseEvent3D	361
Malowanie na trójwymiarowych obiektach	361
Rozmieszczanie obiektów na planszy	367
Podsumowanie	384

Rozdział 9. Kamery	387
Podstawy działania kamer w Away3D	388
Rejestrowanie sceny w Away3D	388
Podstawowe pojęcia i właściwości kamer w Away3D	389
Rodzaje soczewek	406
Przykład	406
AbstractLens	414
ZoomFocusLens i PerspectiveLens	415
SphericalLens	415
OrthogonalLens	417
Rodzaje kamer	418
Camera3D	418
TargetCamera3D	420
HoverCamera3D	421
SpringCam	422
Przykładowe zastosowania kamer	425
Kamera z perspektywy pierwszej osoby	425
Kamera z perspektywy osoby trzeciej	443
Kamera stosowana w grach typu action RPG	451
Podsumowanie	461
Rozdział 10. Dźwięk	465
Klasy obsługujące dźwięk	465
Sound3D	465
SimplePanVolumeDriver	467
Przykłady zastosowań	468
Dźwięk 3D	468
Kontrolowanie obiektów dźwiękiem	477
Podsumowanie	487
Rozdział 11. Tekst	489
Przygotowanie czcionki	490
Sposoby wyświetlania tekstu	492
Tekst płaski	492
Tekst przestrzenny	496
Materiały dla tekstu	497
Tekst jako BitmapData	498
Tekst w obiekcie MovieClip	499
Deformowanie tekstu	499
Podsumowanie	503

Rozdział 12. Optymalizacja	505
Statystyki aplikacji	506
Away3D Project stats	506
AwayStats	508
Hi-ReS-Stats	509
Wyświetlanie zawartości	510
Sposoby przycinania widoku	510
Stosowanie filtrów	516
Ogólne wskazówki	521
Obiekty	522
Stosowanie obiektów LOD	522
Stosowanie narzędzia Weld	527
Ogólne wskazówki	530
Tekstury i światło	531
Ogólne wskazówki	531
Rozdział 13. Więcej zabawy z 3D	533
Fizyka i kolizje	534
Wykrywanie kolizji w Away3D	534
Dodatkowe silniki fizyki	538
Rzeczywistość rozszerzona	541
FLARToolKit	543
FLARManager	544
IN2AR	545
Kontrolery	546
Xbox Kinect	546
Wii Remote	547
Stage3D	548
O przygotowaniu środowiska do Stage3D słów kilka	548
Co to jest Stage3D?	555
Gdzie umieszczony jest Stage3D?	556
Ograniczenia związane ze Stage3D	557
Jak korzystać ze Stage3D?	558
Podstawowe pojęcia związane ze Stage3D	559
Podstawowy szkielet dla aplikacji korzystającej ze Stage3D	563
AGAL	565
Co to jest AGAL?	566
Podstawowe komendy języka AGAL	566

Rejestry w języku AGAL	567
Zastosowanie kodu AGAL wraz z ActionScript	570
Away3D 4.x	578
Instalacja biblioteki	579
Podstawowy przykład z kulą ziemską	579
Skorowidz	591

Rozdział 8.

Interaktywność

Jedną z głównych zalet pisania aplikacji na platformę Flash jest to, że można tworzyć ciekawe interaktywne gry i strony internetowe. W tego typu projektach większość operacji wykonywanych jest za pomocą myszki, klawiatury i coraz częściej kamer. Jeżeli masz jakieś doświadczenie w realizacji tego typu projektów, z pewnością łatwo połączysz zdobytą wiedzę z informacjami zawartymi w tym rozdziale. Jeżeli są to Twoje początki z aplikacjami tworzonymi na platformę Adobe Flash Player, nie przejmuj się, na tym etapie wiesz już wystarczająco dużo.

Tym, co łączy zwykle dwuwymiarowe aplikacje z projektami bazującymi na bibliotece Away3D, jest to, że w obu przypadkach można korzystać z urządzeń takich jak mysz czy klawiatura. Stosując szereg standardowych metod, zdarzeń z ActionScript 3.0 oraz Away3D, można zaprogramować odpowiednie operacje, które wywoływane będą przez ingerencję użytkownika.

Czytając ten rozdział, dowiesz się:

- ◆ Jak obsługuje się zdarzenia wciśnięcia i zwolnienia klawisza klawiatury.
- ◆ Czym są i jakie role odgrywają klasy `KeyboardEvent`, `Keyboard` oraz `KeyLocation`.
- ◆ Jak sterować trójwymiarowym obiektem za pomocą klawiatury.
- ◆ Czym jest klasa `MouseEvent3D`.
- ◆ W jakich sytuacjach i jak stosować `MouseEvent` oraz `MouseEvent3D`.
- ◆ Jak przemieszczać trójwymiarowe obiekty przy użyciu myszki.
- ◆ Jak malować po powierzchni trójwymiarowych obiektów.

Używanie klawiatury

W większości gier i innych aplikacji niekiedy obiekty sterowane są za pomocą klawiatury. W aplikacjach Flash do wykrywania czynności wykonywanych przez użytkownika na klawiaturze stosuje się obiekt zdarzenia `KeyboardEvent`.

Klasa `KeyboardEvent`

Klasa `KeyboardEvent`, ponieważ reprezentuje obiekt zdarzenia, zlokalizowana jest w pakiecie `flash.events`. Ma ona dwie stałe: `KeyboardEvent.KEY_DOWN` i `KeyboardEvent.KEY_UP`, które określają przypadki wciśnięcia i zwolnienia klawisza na klawiaturze. Aby móc korzystać z tych zdarzeń, należy metodę `addEventListener` wywołać na obiekcie `stage`, tak jak to pokazano poniżej:

```
stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown);
stage.addEventListener(KeyboardEvent.KEY_UP, onKeyUp);
```

Samo wywołanie detektora nie wystarczy do podjęcia konkretnych akcji. W ciele metody wywoływanej podczas wystąpienia zdarzenia należy rozpoznać, który klawisz zmienił swój stan. Do tego celu służą dwie właściwości: `keyCode` i `charCode`. Pierwsza zwraca wartość numeryczną odpowiadającą numerowi klawisza, a druga określa wartość liczbową kodu znaku wciśniętego klawisza.

Istnieje zasadnicza różnica między wartościami tych właściwości. Polega ona na tym, że kod klawisza jest przypisany do konkretnego klawisza fizycznego, natomiast kod znaku jest przypisany do konkretnego znaku. Czyli jeśli na przykład chcemy wykorzystać klawisz 5 na klawiaturze numerycznej i w górnym rzędzie, należy odwołać się do wartości właściwości `charCode`.



Domyślny zestaw kodów znaków UTF-8 zawiera w swoim zbiorze identyfikatory dla każdego ze znaków z osobną, odróżnia tym samym wielkość liter.

Na klawiaturze umieszczone są również klawisze specjalne, takie jak *Ctrl*, *Alt* czy *Shift*. Do określenia, czy któryś z tych klawiszy zmienił swój stan, służą specjalne właściwości: `ctrlKey`, `altKey` i `shiftKey`. Wartości tych właściwości są typu `Boolean`, gdzie `true` oznacza, że klawisz jest wciśnięty.

Tabele 8.1, 8.2 oraz 8.3 zawierają metody, właściwości i stałe klasy `KeyboardEvent`.

Tabela 8.1. Właściwości klasy `KeyboardEvent`

Nazwa	Rodzaj	Wartość domyślna	Opis
<code>altKey</code>	Boolean	<code>false</code>	Określa, czy klawisz <i>Alt</i> jest wciśnięty
<code>ctrlKey</code>	Boolean	<code>false</code>	Określa, czy klawisz <i>Ctrl</i> jest wciśnięty
<code>controlKey</code>	Boolean	<code>false</code>	Określa, czy klawisz <i>Control</i> jest wciśnięty
<code>commandKey</code>	Boolean	<code>false</code>	Określa, czy klawisz <i>Command</i> jest wciśnięty
<code>shiftKey</code>	Boolean	<code>false</code>	Określa, czy klawisz <i>Shift</i> jest wciśnięty
<code>charCode</code>	uint		Wartość liczbowa kodu znaku wciśniętego klawisza
<code>keyCode</code>	uint		Wartość liczbowa odpowiadająca numerowi klawisza na klawiaturze
<code>keyLocation</code>	uint		Położenie klawisza na klawiaturze

Tabela 8.2. Metody klasy `KeyboardEvent`

Nazwa	Opis
<code>KeyboardEvent(type:String, bubbles:Boolean, cancelable:Boolean, charCodeValue:uint, keyCodeValue:uint, keyLocationValue:uint, ctrlKeyValue:Boolean, altKeyValue:Boolean, shiftKeyValue:Boolean, controlKeyValue:Boolean, commandKeyValue:Boolean)</code>	Konstruktor
<code>updateAfterEvent</code>	Określa, czy po zakończeniu przetwarzania zdarzenia obraz powinien być zrenderowany, jeżeli lista wyświetlania została zmodyfikowana

Tabela 8.3. Stałe klasy `KeyboardEvent`

Nazwa	Opis
<code>KEY_DOWN</code>	Definiuje zdarzenie wciśnięcia klawisza
<code>KEY_UP</code>	Definiuje zdarzenie zwolnienia wciśniętego klawisza

Klasa `Keyboard`

`Keyboard` jest klasą typu `final`, umieszczoną w pakiecie `flash.ui`. Ma ona zdefiniowane kody większości znaków w postaci stałych publicznych. Do jej metod oraz właściwości można odwoływać się bez tworzenia obiektu. Dzięki tym cechom `Keyboard` stanowi swego rodzaju słownik do stosowania w instrukcjach warunkowych takich jak `if` czy `switch`.

```

switch(event.charCode)
{
    case Keyboard.UP: trace('Wciśnięto klawisz górnej strzałki'); break;
    case Keyboard.DOWN: trace('Wciśnięto klawisz dolnej strzałki'); break;
    case Keyboard.LEFT: trace('Wciśnięto klawisz lewej strzałki'); break;
    case Keyboard.RIGHT: trace('Wciśnięto klawisz prawej strzałki'); break;
}

```

Tabele 8.4 i 8.5 zawierają metody i właściwości klasy `Keyboard`. Ważniejsze od nich są jednak stałe umieszczone w tabeli 8.6. `Keyboard` ma dużą liczbę stałych, dlatego przedstawiono tylko niektóre z nich.

Tabela 8.4. Właściwości klasy `Keyboard`

Nazwa	Rodzaj	Wartość domyślna	Opis
<code>capsLock</code>	Boolean	<code>false</code>	Określa, czy klawisz <i>Caps Lock</i> jest wciśnięty
<code>hasVirtualKeyboard</code>	Boolean	<code>false</code>	Określa, czy udostępniona jest klawiatura wirtualna
<code>numLock</code>	Boolean	<code>false</code>	Określa, czy klawisz <i>Num Lock</i> jest wciśnięty
<code>physicalKeyboardType</code>	String		Określa rodzaj klawiatury

Tabela 8.5. Metody klasy `Keyboard`

Nazwa	Opis
<code>isAccessible():Boolean</code>	Określa, czy ostatnio wciśnięty klawisz jest dostępny dla innych plików SWF

Tabela 8.6. Stałe klasy `Keyboard`

Nazwa	Opis
<code>A ... Z</code>	Klawisze alfabetyczne
<code>NUMBER_0 ... NUMBER_9</code>	Klawisze numeryczne umieszczone w górnym rzędzie
<code>NUMPAD_0 ... NUMPAD_9</code>	Klawisze numeryczne umieszczone na klawiaturze numerycznej
<code>F1 ... F12</code>	Klawisze funkcyjne
<code>ENTER</code>	Klawisz <i>Enter</i>
<code>ESCAPE</code>	Klawisz <i>Esc</i>
<code>CONTROL</code>	Klawisz <i>Ctrl</i>
<code>BACKSPACE</code>	Klawisz <i>Backspace</i>
<code>SPACE</code>	Klawisz spacji
<code>LEFT</code>	Klawisz lewej strzałki

Tabela 8.6. Stałe klasy *Keyboard* (ciąg dalszy)

Nazwa	Opis
RIGHT	Klawisz prawej strzałki
UP	Klawisz górnej strzałki
DOWN	Klawisz dolnej strzałki

Klasa *KeyLocation*

W tabeli 8.1 punktu „Klasa *KeyboardEvent*” wspomniano o tym, że właściwość *keyLocation* określa położenie wciskanego lub zwalnianego klawisza. W sytuacji gdy dany klawisz ma swoje duplikaty w innych miejscach na klawiaturze, zastosowanie tej właściwości pozwala na odróżnienie na przykład lewego i prawego klawisza *Ctrl*, co umożliwi przypisanie każdemu innych działań. Do określenia wartości właściwości *keyLocation* klasy *KeyboardEvent* służą stałe klasy *KeyLocation*.

Zlokalizowana w pakiecie *flash.ui* klasa *KeyLocation* — tak samo jak *Keyboard* — jest klasą z przypisanym atrybutem *final*. *KeyLocation* można traktować jako mały słownik, który ma w swoich zasobach określenia dostępnych pozycji na klawiaturze.

```

if(event.charCode == Keyboard.CONTROL && event.keyLocation ==
↳KeyLocation.LEFT)
{
    trace('Wciśnięto lewy Ctrl');
}
else if(event.charCode == Keyboard.CONTROL && event.keyLocation ==
↳KeyLocation.RIGHT)
{
    trace('Wciśnięto prawy Ctrl');
}
else
{
    trace('Wciśnięto inny klawisz');
}

```

Ponieważ klasa *KeyLocation* nie ma swoich metod i właściwości, tylko stałe określające położenie, w tym punkcie uwzględniono jedną tabelę 8.7.

Tabela 8.7. Stałe klasy *KeyLocation*

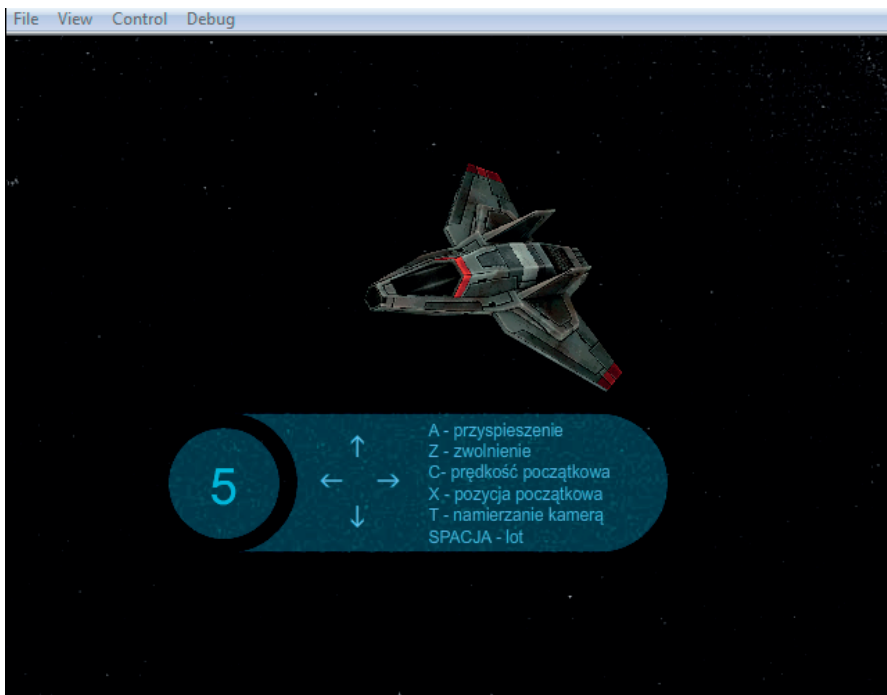
Nazwa	Opis
D_PAD	Określa, czy wciśnięty klawisz znajduje się na panelu kierunkowym
LEFT	Określa, czy wciśnięty klawisz znajduje się po lewej stronie klawiatury
RIGHT	Określa, czy wciśnięty klawisz znajduje się po prawej stronie klawiatury

Tabela 8.7. Stałe klasy *KeyLocation* (ciąg dalszy)

Nazwa	Opis
NUM_PAD	Określa, czy wciśnięty klawisz znajduje się na klawiaturze numerycznej lub w jej wirtualnym odpowiedniku
STANDARD	Określa, czy pozycja klawisza nie została określona w konkretnych pozycjach

Sterowanie statkiem kosmicznym w przestrzeni

Skoro poznaliśmy klasy obsługujące interakcje użytkownika z użyciem klawiatury, przyszedł czas na zastosowanie zdobytej wiedzy w praktyce. Do tego celu wykorzystamy możliwość sterowania futurystycznym pojazdem w przestrzeni kosmicznej. Aplikacja ta składa się z dwóch warstw. Pierwsza to przestrzeń trójwymiarowa, w której porusza się pojazd, druga warstwa to element interfejsu użytkownika, w którym widoczne są aktualna prędkość pojazdu oraz spis klawiszy i akcji, jakie wywołują. Całość przedstawiono na rysunku 8.1.

**Rysunek 8.1.** Zrzut ekranu z aplikacji sterowania statkiem kosmicznym

Widoczny statek kosmiczny to jeden z darmowych modeli w formacie 3DS, które są dostępne w sieci. W pozycji początkowej pojazd ten skierowany jest dziobem w stronę osi Z. Gdy wciskamy klawisze lewej i prawej strzałki, obiekt obraca się

wokół osi Z. Natomiast klawiszami strzałki górnej i dolnej odpowiednio podnosi się i opuszcza dziób. Wprawienie w ruch statku kosmicznego następuje po wciśnięciu klawisza spacji. Zwroty i skręty uzależnione są od kąta nachylenia pojazdu i prędkości, z jaką się porusza. Standardowo prędkość jest równa 50, lecz można ją zwiększyć, wciskając klawisz A, lub zmniejszyć klawiszem Z. Aby szybko zredukować prędkość do jej standardowej wartości, wystarczy użyć klawisza C. W sytuacji gdy stracimy pojazd z pola widzenia, można przywrócić go do pozycji początkowej, wciskając klawisz X, lub namierzyć kamerą jego pozycję, używając klawisza T.

Ponieważ kamery zostaną omówione dopiero w rozdziale 9. „Kamery”, skorzystamy ze standardowej, utworzonej wraz z widokiem.

Teraz przepisz i skompiluj kod źródłowy tego przykładu, a następnie zajmiemy się omawianiem jego poszczególnych fragmentów.

```
package
{
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;
    import flash.display.MovieClip;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.KeyboardEvent;
    import flash.geom.Vector3D;
    import flash.ui.Keyboard;
    //
    import away3d.containers.ObjectContainer3D;
    import away3d.containers.View3D;
    import away3d.loaders.Loader3D;
    import away3d.loaders.Max3DS;
    import com.greensock.TweenMax;
    public class KeyboardEventsExample extends Sprite
    {
        private var view:View3D;
        private var speed:Number = 5;
        private var shipLoader:Loader3D;
        private var ship:ObjectContainer3D;
        private var leftArrowDown:Boolean = false;
        private var rightArrowDown:Boolean = false;
        private var upArrowDown:Boolean = false;
        private var downArrowDown:Boolean = false;
        private var spaceDown:Boolean = false;
        private var trackShip:Boolean = false;
        private var speedMonitor:Monitor;

        public function KeyboardEventsExample():void
        {
            if (stage) init();
            else addEventListener(Event.ADDED_TO_STAGE, init);
        }
    }
}
```

```

private function init(e:Event = null):void
{
    stage.scaleMode = StageScaleMode.NO_SCALE;
    stage.align = StageAlign.TOP_LEFT;
    removeEventListener(Event.ADDED_TO_STAGE, init);
    addEventListener(Event.ENTER_FRAME, onEnterFrame);
    stage.addEventListener(Event.RESIZE, onResize);
    stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown);
    stage.addEventListener(KeyboardEvent.KEY_UP, onKeyUp);

    view = new View3D();
    view.x = stage.stageWidth * .5;
    view.y = stage.stageHeight * .5;
    addChild(view);
    view.camera.z = -100;

    speedMonitor = new Monitor();
    speedMonitor.txt.text = String(speed * 10);
    addChild(speedMonitor);

    loadShipModel();
    onResize();
}

private function onResize(e:Event = null):void
{
    universe.x = stage.stageWidth * .5;
    universe.y = stage.stageHeight * .5;
    view.x = stage.stageWidth * .5;
    view.y = stage.stageHeight * .5;
    speedMonitor.y = stage.stageHeight - 100;
    speedMonitor.x = 100;
}

private function loadShipModel():void
{
    ship = new ObjectContainer3D();
    shipLoader = Max3DS.load('../resources/models/max3ds/spaceship/
↳spaceship.3DS');
    shipLoader.rotationY = -90;
    shipLoader.rotationX = 90;
    shipLoader.position = new Vector3D(0, 0, 0);
    ship.addChild(shipLoader);
    view.scene.addChild(ship);
}

private function onKeyDown(e:KeyboardEvent):void
{
    if (e.keyCode == Keyboard.C)
    {
        speed = 5;
        speedMonitor.txt.text = String(speed * 10);
    }
}

```

```

    }
    if (e.keyCode == Keyboard.A)
    {
        speed += .5;
        speedMonitor.txt.text = String(speed * 10);
    }
    if (e.keyCode == Keyboard.Z)
    {
        speed -= .5;
        speedMonitor.txt.text = String(speed * 10);
    }
    if (e.keyCode == Keyboard.LEFT) leftArrowDown = true;
    if (e.keyCode == Keyboard.RIGHT) rightArrowDown = true;
    if (e.keyCode == Keyboard.UP) upArrowDown = true;
    if (e.keyCode == Keyboard.DOWN) downArrowDown = true;
    if (e.keyCode == Keyboard.SPACE) spaceDown = true;
}

private function onKeyUp(e:KeyboardEvent):void
{
    if (e.keyCode == Keyboard.LEFT) leftArrowDown = false;
    if (e.keyCode == Keyboard.RIGHT) rightArrowDown = false;
    if (e.keyCode == Keyboard.UP) upArrowDown = false;
    if (e.keyCode == Keyboard.DOWN) downArrowDown = false;
    if (e.keyCode == Keyboard.SPACE) spaceDown = false;
    if (e.keyCode == Keyboard.X)
    {
        trackShip = false;
        view.camera.lookAt(new Vector3D(0, 0, 0));
        TweenMax.to(ship, 1, { x:0, y:0, z:0, rotationY:0, rotationX:0,
        ↪rotationZ:0 } );
    }

    if (e.keyCode == Keyboard.T)
    {
        trackShip = !trackShip;
    }
}

private function onEnterFrame(e:Event):void
{
    if (spaceDown) ship.moveForward(speed);
    if (leftArrowDown) ship.roll(-5);
    if (rightArrowDown) ship.roll(5);
    if (upArrowDown) ship.pitch(5);
    if (downArrowDown) ship.pitch( -5);
    if (trackShip) view.camera.lookAt(ship.scenePosition);
    view.render();
}
}
}

```

W tym przykładzie skorzystaliśmy z podstawowych klas języka ActionScript 3.0 między innymi do obsługi zdarzeń użycia klawiatury, słownika kodów klawiszy oraz wyświetlenia interfejsu.

```
import flash.display.MovieClip;
import flash.events.KeyboardEvent;
import flash.ui.Keyboard;
```

Z Away3D poza widokiem użyliśmy klas służących do ładowania i umieszczania na scenie zewnętrznych obiektów 3D. W tym konkretnym przypadku modeli w formacie 3DS.

```
import away3d.containers.ObjectContainer3D;
import away3d.containers.View3D;
import away3d.loaders.Loader3D;
import away3d.loaders.Max3DS;
```

Dodatkowo do ustawienia statku w pozycji zerowej sceny skorzystaliśmy z biblioteki TweenMax.

```
import com.greensock.TweenMax;
```

W klasie KeyboardEventsExample na początku zdefiniowaliśmy potrzebne obiekty i zmienne. Jako obiekt reprezentujący model 3D statku kosmicznego wykorzystaliśmy obiekt klasy Loader3D i nazwaliśmy go shipLoader. Wewnętrzne ustawienia pozycji modelu uzależnione są od tego, jak został on wyeksportowany. Dlatego właśnie zawartość obiektu shipLoader umieścimy w kontenerze o nazwie ship, aby w jego wnętrzu dokonywać zmian.

Obiekt o nazwie speedMonitor klasy Monitor to reprezentant obiektu interfejsu. Kolejne linijki to zmienne stosowane do obsługi pojazdu. Zmienna speed to stopień zmiany prędkości statku, z kolei trackShip określa, czy kamera ma śledzić obiekt statku, czy nie. Zmienne: leftArrowDown, rightArrowDown, upArrowDown, downArrowDown oraz spaceDown określają, czy wybrany klawisz został wciśnięty.

```
private var view:View3D;
private var shipLoader:Loader3D;
private var ship:ObjectContainer3D;
private var speedMonitor:Monitor;
private var speed:Number = 5;
private var trackShip:Boolean = false;
private var leftArrowDown:Boolean = false;
private var rightArrowDown:Boolean = false;
private var upArrowDown:Boolean = false;
private var downArrowDown:Boolean = false;
private var spaceDown:Boolean = false;
```

W konstruktorze uruchamiamy metodę init(), z chwilą gdy obiekt stage zostanie zainicjowany.

```
if (stage) init();
else addEventListener(Event.ADDED_TO_STAGE, init);
```

W metodzie `init()` w pierwszych dwóch liniijkach usunęliśmy detektor zdarzenia `Event.ADDED_TO_STAGE` i utworzyliśmy nowy dla `Event.ENTER_FRAME`. W kolejnych liniijkach ustawiliśmy właściwości dla okna aplikacji i dodaliśmy kolejne rejestratory zdarzeń dla obiektu `stage`. Pierwszy z nich uruchamia metodę `onResize()` z każdą zmianą rozmiarów okna. Kolejne dwa słuchają zdarzeń wciśnięcia i zwolnienia klawiszy klawiatury.

```
removeEventListener(Event.ADDED_TO_STAGE, init);
addEventListener(Event.ENTER_FRAME, onEnterFrame);
stage.scaleMode = StageScaleMode.NO_SCALE;
stage.align = StageAlign.TOP_LEFT;
stage.addEventListener(Event.RESIZE, onResize);
stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown);
stage.addEventListener(KeyboardEvent.KEY_UP, onKeyUp);
```

W dalszej części metody `init()` stworzyliśmy widok `Away3D` i zmieniliśmy pozycję kamery na osi `Z`, tak aby przybliżyć ją do obiektu statku kosmicznego.

```
view = new View3D();
addChild(view);
view.camera.z = -100;
```

Po utworzeniu widoku dodaliśmy obiekt `speedMonitor`, który przedstawia listę dostępnych klawiszy i prędkość pojazdu. Wyświetloną w panelu użytkownika prędkość początkową uzyskaliśmy z iloczynu wartości zmiennej `speed` i liczby 10.

```
speedMonitor = new Monitor();
speedMonitor.txt.text = String(speed * 10);
addChild(speedMonitor);
```

Na końcu `init()` wywołujemy metodę `loadShipModel()` i `onResize()`, aby ustawić wszystkie obiekty w odpowiednim miejscu.

```
loadShipModel();
onResize();
```

W metodzie `onResize()` centrujemy widok `Away3D` oraz obiekt typu `movieClip` o nazwie `universe`. Obiekt `universe` to gwiazdziste tło, które zostało bezpośrednio umieszczone w głównym oknie aplikacji. Z kolei obiekt interfejsu użytkownika umieszczamy w lewym dolnym rogu okna.

```
universe.x = stage.stageWidth * .5;
universe.y = stage.stageHeight * .5;
view.x = stage.stageWidth * .5;
view.y = stage.stageHeight * .5;
speedMonitor.y = stage.stageHeight - 100;
speedMonitor.x = 100;
```

Metoda `loadShipModel()` służy do pobrania modelu z konkretnej lokalizacji i umieszczenia go na scenie. W pierwszej kolejności stworzyliśmy obiekt klasy `ObjectContainer3D` o nazwie `ship`, który będzie służył jako kontener dla pobranego modelu. Następnie pobierany jest model w formacie 3DS za pomocą klasy `Max3DS` i jej metody `load`.

Po załadowaniu zmieniliśmy kąt nachylenia statku na osi Y oraz X i ustawiliśmy jego pozycję w miejscu zerowym kontenera. Na końcu umieściliśmy obiekt na scenie.

```
ship = new ObjectContainer3D();
shipLoader = Max3DS.load('models/spaceship/spaceship.3ds');
shipLoader.rotationY = -90;
shipLoader.rotationX = 90;
shipLoader.position = new Vector3D(0, 0, 0);
ship.addChild(shipLoader);
view.scene.addChild(ship);
```



Wskazówka

W sytuacji gdy model pochodzi z nieznanego źródła, warto wcześniej sprawdzić w `Away3D`, jak prezentuje się on na scenie. Może dojść do tego, że jego skala i pozycja nie odpowiadają konkretnym potrzebom. W takiej sytuacji, jeżeli nie mamy możliwości edytowania modelu, można umieścić go w kontenerze `ObjectContainer3D` i w jego wnętrzu skorygować skalę bądź kąty nachylenia, na końcu ustawiając jego pozycję. Dzięki temu unikniemy konieczności przestawiania całej sceny i kamery, aby uzyskać oczekiwany efekt.

Metoda `onKeyDown()` w całości składa się z instrukcji warunkowych. W tym miejscu sprawdzane są wartości klawiszy i przypisywane odpowiednie akcje bądź wartości zmiennym określającym stan używanych klawiszy.

W sytuacji gdy wartość `keyCode` zdarzenia `KeyboardEvent` równa jest `Keyboard.C`, prędkość zredukowana jest do wartości początkowej.

```
if (e.keyCode == Keyboard.C)
{
    speed = 5;
    speedMonitor.txt.text = String(speed * 10);
}
```

Z kolei gdy wartość `keyCode` zdarzenia `KeyboardEvent` równa jest `Keyboard.A`, prędkość statku zwiększa się o 0.5.

```
if (e.keyCode == Keyboard.A)
{
    speed += .5;
    speedMonitor.txt.text = String(speed * 10);
}
```

Zmniejszenie prędkości następuje po wciśnięciu klawisza Z, którego wartość `keyCode` równa jest `Keyboard.Z`.

```
if (e.keyCode == Keyboard.Z)
{
    speed -= .5;
    speedMonitor.txt.text = String(speed * 10);
}
```

Aby zachować ciągłość akcji sterowania statkiem, nie modyfikujemy w tym miejscu jego pozycji, lecz przypisujemy wartość typu `Boolean` właściwościom, które w metodzie `onEnterFrame()` będą wskazywały na wciskane przez użytkownika klawisze.

Zdarzenie `KeyboardEvent.KEY_DOWN` uruchamiane jest jedynie przy zmianie stanu, w jakim znajduje się klawisz, dlatego zapisanie zmiany pozycji w metodzie `onKeyDown()` spowodowałoby pojedyncze przejście dla każdego wciśniętego klawisza.

```
if (e.keyCode == Keyboard.LEFT) leftArrowDown = true;
if (e.keyCode == Keyboard.RIGHT) rightArrowDown = true;
if (e.keyCode == Keyboard.UP) upArrowDown = true;
if (e.keyCode == Keyboard.DOWN) downArrowDown = true;
if (e.keyCode == Keyboard.SPACE) spaceDown = true;
```

W metodzie `onKeyUp()` również zawarliśmy instrukcje warunkowe `if`, ale tutaj sprawdzaliśmy stan klawisza i przypisywaliśmy odpowiedniej zmiennej wartość `false`. Nie mogliśmy wszystkim naraz przypisać wartości `false`, ponieważ mogłoby to przerwać ciągłość innej akcji.

```
if (e.keyCode == Keyboard.LEFT) leftArrowDown = false;
if (e.keyCode == Keyboard.RIGHT) rightArrowDown = false;
if (e.keyCode == Keyboard.UP) upArrowDown = false;
if (e.keyCode == Keyboard.DOWN) downArrowDown = false;
if (e.keyCode == Keyboard.SPACE) spaceDown = false;
```

W przypadku wystąpienia warunku, w którym wciśnięty jest klawisz `X`, zapisaliśmy zmianę wartości zmiennej `trackShip` na `false` oraz skierowaliśmy kamerę w stronę środka sceny. Poza tym korzystając z klasy `TweenMax`, stworzyliśmy animację, w której statek kosmiczny powraca do swojej pierwotnej pozycji.

```
if (e.keyCode == Keyboard.X)
{
    trackShip = false;
    view.camera.lookAt(new Vector3D(0, 0, 0));
    TweenMax.to(ship, 1, { x:0, y:0, z:0, rotationY:0, rotationX:0,
rotationZ:0 } );
}
```

W warunku wciśnięcia klawisza `T` zapisaliśmy zmianę wartości `trackShip` na przeciwną do aktualnie ustawionej.

```
if (e.keyCode == Keyboard.T)
{
    trackShip = !trackShip;
}
```

W metodzie `onEnterFrame()`, która jest stale wywoływana, użyliśmy wszystkich zmiennych, których wartości zmieniają się w metodach `onKeyDown()` oraz `onKeyUp()`. W zależności od tego, która ze zmiennych równa jest wartości `true`, wykonywana jest akcja obrotu bądź przesunięcia. Dodatkowo jeżeli wartość `trackShip` równa jest `true`, kamera przy każdym odświeżeniu ustawia się w kierunku pozycji statku.

```
if (spaceDown) ship.moveForward(speed);
if (leftArrowDown) ship.roll(-5);
if (rightArrowDown) ship.roll(5);
if (upArrowDown) ship.pitch(5);
if (downArrowDown) ship.pitch(-5);
if (trackShip) view.camera.lookAt(ship.scenePosition);
view.render();
```

Używanie myszy

W poprzednim podrozdziale pokazaliśmy, jak z użyciem klawiatury manipulować trójwymiarowymi obiektami umieszczonymi na scenie. Poparliśmy to przykładem sterowania pojazdem kosmicznym. Jak jednak wiadomo, w aplikacjach Flash częściej stosuje się mysz komputerową niż klawiaturę. Dlatego w tym podrozdziale przyjrzymy się możliwym sposobom zastosowania tego urządzenia wraz z biblioteką `Away3D`.

MouseEvent

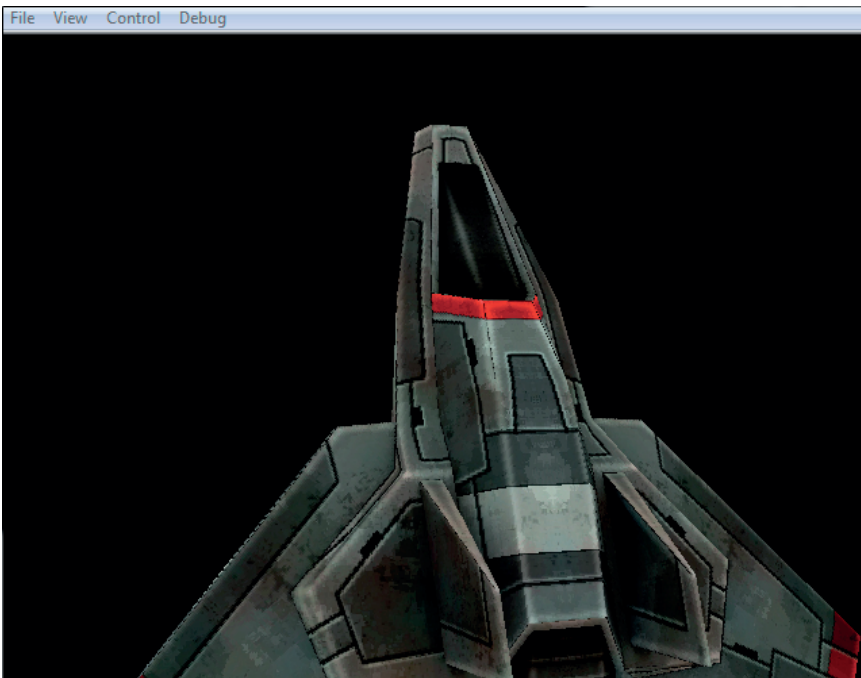
Jak wiemy, w języku ActionScript 3.0 dostępna jest klasa `MouseEvent`, której obiekt obsługuje zdarzenia związane z myszą. W zwykłych aplikacjach dwuwymiarowych zdarzeń myszy można nasłuchiwać na obiektach potomnych względem klasy `InteractiveObject`, czyli prawie na każdym widocznym obiekcie. Niestety, na obiektach trójwymiarowych umieszczonych na scenie `Away3D` nie ma możliwości bezpośredniego stosowania zdarzeń klasy `MouseEvent`. Mimo to są sytuacje, w których można połączyć stosowanie klasy `MouseEvent` z obiektami trójwymiarowymi. Pierwszy sposób polega na zastosowaniu klasy `MovieClipSprite`, którą poznaliśmy w rozdziale 3. „Obiekty”. Jak pamiętamy, wyświetlana w obiekcie klasy `MovieClipSprite` tekstura ma postać obiektu klasy `DisplayObject`. Skutkiem tego na powierzchni tej tekstury można korzystać ze zdarzeń `MouseEvent`. Nie tylko na obiekcie klasy `MovieClipSprite` można stosować zdarzenia `MouseEvent`. Otóż w rozdziale 4. „Materiały” poznaliśmy klasy `MovieMaterial`, `PhongMovieMaterial` oraz `Dot3MovieMaterial`, dla których źródłem wyświetlanej grafiki są obiekty klas `Sprite` oraz `MovieClip`. Jak wiemy, bezpośrednio na obiekcie zarówno klasy `Sprite`, jak i `MovieClip` można wywołać zdarzenia `MouseEvent`, ale gdy używamy ich wewnątrz

któregoś z wyżej wymienionych materiałów, są one niedostępne dla wskaźnika myszy. Jest to normalny stan, który można zmienić, nadając właściwości interactive wybranego materiału wartość true.

Inny sposób wykorzystania zdarzeń MouseEvent do modyfikowania obiektów 3D polega na pobieraniu współrzędnych wskaźnika myszy i przekazywaniu ich do sceny Away3D. Wykrywając zmianę pozycji kursora myszki w oknie aplikacji, można na obiekcie umieszczonym w przestrzeni trójwymiarowej wykonać czynności takie jak skalowanie, przesunięcie czy też obrót. W przykładzie umieszczonym w kolejnym punkcie tego podrozdziału odniesiemy się do tego sposobu interakcji z użyciem myszki komputerowej.

Obracanie i skalowanie statku kosmicznego

Widocznym elementem w tym przykładzie będzie jedynie model statku kosmicznego zastosowany wcześniej w punkcie „Sterowanie statkiem kosmicznym w przestrzeni”. Pozycją początkową dla pojazdu będzie środek przestrzeni trójwymiarowej. Sam model zostanie skierowany dziobem wzdłuż osi Z. Poruszając kursorem, będzie można doprowadzić model do różnych pozycji i wymiarów. Przykładowe ustawienia zaprezentowano na rysunku 8.2.



Rysunek 8.2. Przykładowe ustawienia pozycji i wymiarów modelu

Aby zmienić pozycję modelu statku kosmicznego, należy wcisnąć lewy przycisk myszy i trzymając go, poruszać kursorem w różne strony okna aplikacji. Wciśnięcie i trzymanie przycisku powoduje zmianę wartości jednej ze zmiennych, która jest warunkiem podejmowania działań obrotu. Bez tego każde poruszenie kursorem, nawet nieplanowe, powodowałoby modyfikacje.

Aby dodatkowo przy poruszaniu kursorem kontrolować wymiary modelu, należy użyć klawisza *Ctrl*. Cały mechanizm wyjaśnimy przy omawianiu kodu źródłowego tego przykładu. Teraz przepisz i skompiluj następujący kod.

```
package
{
    import flash.display.StageAlign;
    import flash.display.StageQuality;
    import flash.display.StageScaleMode;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.MouseEvent;
    import flash.events.KeyboardEvent;
    import flash.ui.Keyboard;
    import flash.geom.Vector3D;
    //
    import away3d.containers.ObjectContainer3D;
    import away3d.containers.View3D;
    import away3d.loaders.Loader3D;
    import away3d.loaders.Max3DS;
    public class MouseEventExample extends Sprite
    {
        private var view:View3D;
        private var shipLoader:Loader3D;
        private var ship:ObjectContainer3D;
        private var mouseDown:Boolean = false;
        private var ctrlDown:Boolean = false;
        private var xPos:Number;
        private var yPos:Number;
        private var ease:Number = .3;

        public function MouseEventExample():void
        {
            if (stage) init();
            else addEventListener(Event.ADDED_TO_STAGE, init);
        }

        private function init(e:Event = null):void
        {
            stage.quality = StageQuality.LOW;
            stage.scaleMode = StageScaleMode.NO_SCALE;
            stage.align = StageAlign.TOP_LEFT;
            removeEventListener(Event.ADDED_TO_STAGE, init);
            addEventListener(Event.ENTER_FRAME, onEnterFrame);
            stage.addEventListener(Event.RESIZE, onResize);
        }
    }
}
```

```
stage.addEventListener(MouseEvent.CLICK, onMouseClick);
stage.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
stage.addEventListener(MouseEvent.MOUSE_UP, onMouseUp);
stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown);
stage.addEventListener(KeyboardEvent.KEY_UP, onKeyUp);
//
view = new View3D();
addChild(view);
view.camera.z = -50;
//
ship = new ObjectContainer3D();
shipLoader = Max3DS.load(' ../../resources/models/max3ds/spaceship/
↳spaceship.3DS');
shipLoader.rotationY = -90;
shipLoader.rotationX = 90;
shipLoader.position = new Vector3D(0, 0, 0);
ship.addChild(shipLoader);
view.scene.addChild(ship);
//
onResize();
}

private function onMouseDown(e:MouseEvent):void
{
    mouseDown = true;
}

private function onMouseUp(e:MouseEvent):void
{
    mouseDown = false;
}

private function onKeyDown(e:KeyboardEvent):void
{
    if (e.keyCode == Keyboard.CONTROL) ctrlDown = true;
}

private function onKeyUp(e:KeyboardEvent):void
{
    if (e.keyCode == Keyboard.CONTROL) ctrlDown = false;
}

private function onResize(e:Event = null):void
{
    view.x = stage.stageWidth * .5;
    view.y = stage.stageHeight * .5;
}

private function onEnterFrame(e:Event):void
{
    xPos = stage.mouseX - stage.stageWidth * .5;
    yPos = stage.mouseY - stage.stageHeight * .5;
```

```

        if (mouseDown && !ctrlDown)
        {
            ship.rotationY += (xPos * .5 - ship.rotationY) * ease;
            ship.rotationX += (yPos * .5 - ship.rotationX) * ease;
        }
        else if (mouseDown && ctrlDown)
        {
            ship.scale(Math.sqrt(xPos * xPos + yPos * yPos)*.01);
        }

        view.render();
    }
}
}

```

W tym przykładzie najważniejszą dla nas klasą z pakietu `flash` jest oczywiście `MouseEvent`. Z `Away3D`, tak samo jak w poprzednich przykładach tego rozdziału, skorzystaliśmy z modelu 3DS. Dlatego potrzebowaliśmy klas `Loader3D`, `Max3DS` i `ObjectContainer3D` do umieszczenia modelu statku na scenie.

Na początku ciała klasy `MouseEventExample` zdefiniowaliśmy obiekt widoku `Away3D`, obiekty służące do pobrania i wyświetlenia modelu 3DS oraz kilka zmiennych. Pierwsza w kolejności `mouseDown` przyjmuje wartości typu `Boolean` i określa, czy lewy przycisk myszy jest wciśnięty. Jeżeli jest, to wartość tej właściwości równa się `true`. Kolejną zmienną, którą dodaliśmy, jest `ctrlDown`. Jej wartość również jest typu `Boolean` i określa, czy został wciśnięty klawisz *Ctrl*. Właściwości `xPos` oraz `yPos` to zmienne liczbowe, które określają odległość pozycji kursora od środka okna aplikacji. Ich wartości są modyfikowane przy każdym odświeżeniu stołu montażowego. Ostatnia zmienna `ease` odpowiada za złagodzenie przejścia między pozycjami bądź wymiarami modelu pojazdu. Jej wartość w trakcie uruchamiania oraz używania programu nie zmienia się, dlatego można zdefiniować ją jako wartość stałą.

```

private var view:View3D;
private var shipLoader:Loader3D;
private var ship:ObjectContainer3D;
private var mouseDown:Boolean = false;
private var ctrlDown:Boolean = false;
private var xPos:Number;
private var yPos:Number;
private var ease:Number = .3;

```

Podobnie jak w poprzednich konstruktorach dla klas przykładów, i tutaj, gdy obiekt `stage` jest gotowy, uruchamiamy metodę `init()`.

```

if (stage) init();
else addEventListener(Event.ADDED_TO_STAGE, init);

```

Na początku metody `init()` ustawiliśmy jakość generowanego obrazu na poziomie `LOW`. W następnej kolejności wyłączyliśmy skalowanie okna oraz wyrównaliśmy stół montażowy do lewego górnego rogu. Dzięki temu aplikacja będzie działała płynniej i przy powiększeniu okna zwiększymy obszar działania kursora.

```
stage.quality = StageQuality.LOW;
stage.scaleMode = StageScaleMode.NO_SCALE;
stage.align = StageAlign.TOP_LEFT;
```

W kolejnych liniach kodu metody `init()` usunęliśmy detektor zdarzenia `Event.ADDED_TO_STAGE` i utworzyliśmy szereg nowych. Kolejność, z jaką je dodaliśmy, jest przypadkowa. Pierwszy detektor odpowiada za nasłuchiwanie zdarzenia `Event.ENTER_FRAME` i wywołanie metody `onEnterFrame()`. W drugiej kolejności dodaliśmy detektor dla zmiany rozmiarów okna aplikacji `Event.RESIZE`. Kolejne dwa detektory nasłuchują zdarzeń `MouseEvent.MOUSE_DOWN` oraz `MouseEvent.MOUSE_UP`, które odpowiadają za kliknięcie i puszczenie przycisku myszki. Podobne zadanie przydzieliliśmy detektorom zdarzeń `KeyboardEvent.KEY_DOWN` i `KeyboardEvent.KEY_UP`, z tą różnicą, że one wskazują na kliknięcie bądź zwolnienie klawisza na klawiaturze.

```
removeEventListener(Event.ADDED_TO_STAGE, init);
addEventListener(Event.ENTER_FRAME, onEnterFrame);
stage.addEventListener(Event.RESIZE, onResize);
stage.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
stage.addEventListener(MouseEvent.MOUSE_UP, onMouseUp);
stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown);
stage.addEventListener(KeyboardEvent.KEY_UP, onKeyUp);
```

Po ustaleniu detektorów zdarzeń stworzyliśmy i dodaliśmy obiekt widoku biblioteki `Away3D`. Dodatkowo przybliżyliśmy kamerę do pozycji `-50` na osi `Z`, tak aby statek w swojej początkowej pozycji był dobrze widoczny.

```
view = new View3D();
addChild(view);
view.camera.z = -50;
```

Gdy widok został dodany, stworzyliśmy obiekt statku kosmicznego, korzystając z ustawień z poprzednich przykładów tego rozdziału. Po pobraniu modelu za pomocą klasy `Max3DS` obrócić go tak, aby dziobem, zwrócony był w stronę osi `Z`. Następnie umieściliśmy go w kontenerze o nazwie `ship` i dodaliśmy go do sceny.

```
ship = new ObjectContainer3D();
shipLoader = Max3DS.load('models/spaceship/spaceship.3ds');
shipLoader.rotationY = -90;
shipLoader.rotationX = 90;
shipLoader.position = new Vector3D(0, 0, 0);
ship.addChild(shipLoader);
view.scene.addChild(ship);
```

Na końcu bloku kodu metody `init()` odwołał się do metody `onResize()`.

Kolejnymi metodami, które zapisaaliśmy w klasie `MouseEventExample`, są `onMouseDown()` oraz `onMouseUp()`. Gdy wciśnięty jest lewy przycisk myszki, wywołwana jest metoda `onMouseDown()`, w której właściwości `mouseDown` przypisywana jest wartość `true`.

Z kolei po puszczeniu przycisku myszy wartość `mouseDown()` zostaje zmieniona na `false` w metodzie `onMouseUp()`.

Idąc tym torem, zapisaaliśmy metody `onKeyDown()` oraz `onKeyUp()`, które uruchamiane są z każdym wciśnięciem i zwolnieniem klawisza na klawiaturze. Jednak nie każde wywołanie tych metod powoduje zmiany. Wewnątrz obu metod zastosowaliśmy instrukcję warunkową `if`. Obie filtrują kody wciskanych i zwalnianych klawiszy w celu wychwycenia wartości numerycznej odpowiadającej numerowi klawisza `Ctrl`. Jeżeli klawisz ten zostanie wciśnięty, to w metodzie `onKeyDown()` zmienna `ctrlDown` zmieni swoją wartość na `true`. Z kolei po puszczeniu przycisku `Ctrl` w metodzie `onKeyUp()` zmienna `ctrlDown` zmieni swoją wartość na `false`.

Przy każdej zmianie rozmiaru okna uruchamiana jest metoda `onResize()`. W jej ciele zapisaaliśmy wyśrodkowanie widoku `view` w oknie programu.

```
view.x = stage.stageWidth * .5;
view.y = stage.stageHeight * .5;
```

Na końcu klasy `MouseEventExample` zapisaaliśmy metodę `onEnterFrame()`, wywołwaną przy każdym wystąpieniu zdarzenia `Event.ENTER_FRAME`. W pierwszych dwóch liniach wpisaliśmy formuły wyliczające różnicę między współrzędnymi kursora i współrzędnymi środka okna aplikacji.

```
xPos = stage.mouseX - stage.stageWidth * .5;
yPos = stage.mouseY - stage.stageHeight * .5;
```

Następnie zapisaaliśmy instrukcje `if`, które sprawdzają wartości zmiennych `mouseDown` oraz `ctrlDown`. W pierwszym przypadku sprawdzamy, czy wartość `mouseDown` jest równa `true`, a wartość `ctrlDown` równa `false`. Jeżeli warunek zachodzi, to wcześniej zdefiniowane odległości `xPos` i `yPos` zostaną użyte do zmiany kątów nachylenia modelu na osiach X i Y. Parametr `ease` służy jako mnożnik łagodzący przejście między pozycjami.

```
if (mouseDown && !ctrlDown)
{
    ship.rotationY += (xPos * .5 - ship.rotationY) * ease;
    ship.rotationX += (yPos * .5 - ship.rotationX) * ease;
}
```

Jeżeli wartości właściwości `mouseDown` i `ctrlDown` są równe `true`, to każda zmiana współrzędnych kursora spowoduje zmianę wymiarów obiektu. Im dalej od środka będzie skierowany kursor, tym większą skalę będzie miał model statku kosmicznego. Aby obliczyć tę odległość, musieliśmy użyć właściwości `xPos` i `yPos`. Bezpośrednio

w metodzie `scale()` obiektu `ship` obydwie te wartości podnieśliśmy do kwadratu i dodaliśmy. Następnie wyliczyliśmy pierwiastek z tej sumy i pomnożyliśmy go przez 0.01. W ten sposób, bazując na współrzędnych pozycji kursora, uzyskamy długość wektora, którą będziemy skalowali obiekt `ship`.

```
else if (mouseDown && ctrlDown)
{
    ship.scale(Math.sqrt(xPos * xPos + yPos * yPos)*.01);
}
```

Na końcu metody `onEnterFrame()` odświeżamy widok metodą `render()`.

MouseEvent3D

We wcześniejszych przykładach tego podrozdziału pokazaliśmy, że można za pomocą `MouseEvent` stworzyć interakcje użytkownika ze środowiskiem `Away3D`. Problem polega na tym, że przedstawione sposoby nie odnosiły się bezpośrednio do obiektów trójwymiarowych, tylko do obiektów klas pochodnych względem `InteractiveObject`. Na szczęście biblioteka `Away3D` ma swoją implementację pełnej obsługi zdarzeń myszki w postaci klasy `MouseEvent3D`.

Zlokalizowana w pakiecie `away3d.events` klasa `MouseEvent3D` jest odpowiednikiem standardowego `MouseEvent`, tylko że w tym przypadku ma ona zastosowanie bezpośrednio na obiektach umieszczonych w przestrzeni trójwymiarowej, niezależnie od ich rodzaju i materiału, którym są pokryte. W swoich zasobach klasa `MouseEvent3D` ma stałe, które definiują większość zdarzeń znanych z `MouseEvent`, między innymi `MouseEvent3D.MOUSE_DOWN`, `MouseEvent3D.MOUSE_UP` czy też `MouseEvent3D.MOUSE_MOVE`. Każde ze zdarzeń `MouseEvent3D` wywołuje się bezpośrednio na obiekcie, na którym ma ono być rejestrowane. Działa to na tych samych zasadach jak w przypadku zwykłych obiektów `DisplayObject`.

```
Obj3d.addEventListener(MouseEvent3D.MOUSE_DOWN, onMouseDown);
```

Oczywiście zamiast odwoływać się do obiektu zdarzenia, można wpisać samą postać `String`, którą reprezentuje stała obiektu `MouseEvent3D`. Powyższy przykład dodania detektora w tym przypadku wyglądałby następująco:

```
Obj3d.addEventListener("mouseDown3d", onMouseDown);
```

Tabele 8.8 i 8.9 zawierają metody i właściwości klasy `MouseEvent3D`. W tabeli 8.10 wypisane zostały stałe odpowiadające konkretnym zdarzeniom.

Tabela 8.8. Właściwości klasy *MouseEvent3D*

Nazwa	Rodzaj	Wartość domyślna	Opis
ctrlKey	Boolean	false	Określa, czy klawisz <i>Ctrl</i> jest wciśnięty
elementV0	ElementV0		Obiekt typu <i>ElementV0</i> , na którym wywołano zdarzenie
material	Material		Materiał obiektu 3D, na którym wywołano zdarzenie
Object	Object3D		Obiekt 3D, na którym wywołano zdarzenie
sceneX	Number		Współrzędna <i>x</i> na scenie
sceneY	Number		Współrzędna <i>y</i> na scenie
sceneZ	Number		Współrzędna <i>z</i> na scenie
screenX	Number		Współrzędna <i>x</i> w widoku
screenY	Number		Współrzędna <i>y</i> w widoku
screenZ	Number		Współrzędna <i>z</i> w widoku
shiftKey	Number	false	Określa, czy klawisz <i>Shift</i> jest wciśnięty
uv	UV		Współrzędne UV wewnątrz obiektu, na którym wywołano zdarzenie
view	View3D		Obiekt widoku, w którym wywołano zdarzenie

Tabela 8.9. Metody klasy *MouseEvent3D*

Nazwa	Opis
<i>MouseEvent3D</i> (type:String)	Konstruktor
clone():Event	Tworzy kopię obiektu <i>MouseEvent3D</i>

Tabela 8.10. Stałe klasy *MouseEvent3D*

Nazwa	Wartość domyślna	Opis
MOUSE_DOWN	'mouseDown3d'	Zdarzenie wciśnięcia przycisku myszy na trójwymiarowym obiekcie
MOUSE_MOVE	'mouseMove3d'	Zdarzenie poruszania kursorem na trójwymiarowym obiekcie
MOUSE_OUT	'mouseOut3d'	Zdarzenie opuszczenia kursora z trójwymiarowego obiektu lub któregośkolwiek z obiektów w nim umieszczonych
MOUSE_OVER	'mouseOver3d'	Zdarzenie najechania kursorem na trójwymiarowy obiekt lub którykolwiek z obiektów w nim umieszczonych
MOUSE_UP	'mouseUp3d'	Zdarzenie puszczenia przycisku myszy na trójwymiarowym obiekcie

Tabela 8.10. Stałe klasy `MouseEvent3D` (ciąg dalszy)

Nazwa	Wartość domyślna	Opis
<code>ROLL_OVER</code>	<code>'rollOut3d'</code>	Zdarzenie najechania kursorem na trójwymiarowy obiekt
<code>ROLL_OUT</code>	<code>'rollOver3d'</code>	Zdarzenie opuszczenia kursora z trójwymiarowego obiektu

Metody dla zdarzeń `MouseEvent3D`

Obiekty klasy `Object3D` mają własne metody inicjujące detektory zdarzeń dla `MouseEvent3D`. Każda z tych metod ma inną nazwę, która wskazuje na rodzaj zdarzenia. Jako argument dla takiej metody należy podać jedynie nazwę metody, która ma być wywołana po zajściu zdarzenia.

Tabela 8.11 zawiera spis nazw metod oraz zdarzeń, którym one odpowiadają.

Tabela 8.11. Metody obsługi zdarzeń `MouseEvent3D`

Nazwa metody	Zdarzenie
<code>addOnMouseDown</code>	<code>MouseEvent3D.MOUSE_DOWN</code>
<code>addOnMouseMove</code>	<code>MouseEvent3D.MOUSE_MOVE</code>
<code>addOnMouseOut</code>	<code>MouseEvent3D.MOUSE_OUT</code>
<code>addOnMouseOver</code>	<code>MouseEvent3D.MOUSE_OVER</code>
<code>addOnMouseUp</code>	<code>MouseEvent3D.MOUSE_UP</code>
<code>addOnRollOut</code>	<code>MouseEvent3D.ROLL_OUT</code>
<code>addOnRollOver</code>	<code>MouseEvent3D.ROLL_OVER</code>

Malowanie na trójwymiarowych obiektach

W tym punkcie wyjaśnimy, jak napisać aplikację umożliwiającą malowanie po powierzchni obiektów 3D.

Aby wykonać to zadanie, w pierwszej kolejności stworzymy obiekt ściany i wypełnimy go teksturą z ceglaną powierzchnią. Następnie do tego obiektu dodamy detektory zdarzeń poruszania kursorem, wciśnięcia i puszczenia przycisku myszy. Dodatkową zmienną typu `Boolean` sprawimy, że malowanie będzie trwało od momentu wciśnięcia do zwolnienia przycisku myszy.

Sam proces rysowania umieścimy w metodzie obsługi zdarzenia `Event.ENTER_FRAME`.

Aby na powierzchni ściany były widoczne zmiany, skorzystamy z materiału typu `MovieMaterial`. Jego pierwszą warstwę wypełnimy wcześniej wspomnianą ceglaną

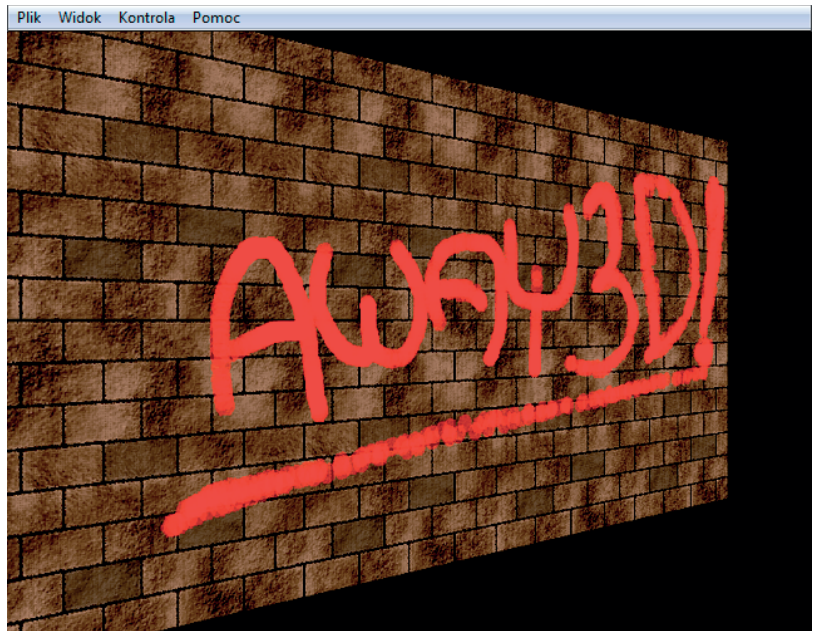
teksturą. Kolejne rysunki będą skutkiem tworzenia nowych obiektów Sprite, w których metodą `drawCircle()` będziemy rysowali pojedyncze, półprzezroczyste czerwone koła.

Aby ustalić współrzędne kursora na obiekcie, skorzystamy ze wspomnianego w tabeli 8.8 obiektu UV.

Efekt, który uzyskamy po napisaniu i skompilowaniu przykładu, ilustruje rysunek 8.3.

Rysunek 8.3.

Malowanie graffiti na trójwymiarowej ścianie



Teraz przepisz i skompiluj następujący kod. Następnie zajmiemy się omawianiem jego poszczególnych fragmentów.

```
package
{
    import flash.geom.Vector3D;
    import flash.events.Event;
    import flash.display.StageQuality;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;
    import flash.display.Sprite;
    //
    import away3d.containers.View3D;
    import away3d.primitives.Cube;
    import away3d.core.utils.Cast;
    import away3d.materials.MovieMaterial;
    import away3d.primitives.data.CubeMaterialsData;
    import away3d.events.MouseEvent3D;
```

```
public class GraffitiExample extends Sprite
{
    private var view:View3D;
    private var wall:Cube;
    private var cubeMaterials:CubeMaterialsData;
    private var drawingEnabled:Boolean = false;

    public function GraffitiExample()
    {
        if (stage) init();
        else addEventListener(Event.ADDED_TO_STAGE, init);
    }

    private function init(e:Event = null):void
    {
        stage.quality = StageQuality.LOW;
        stage.scaleMode = StageScaleMode.NO_SCALE;
        stage.align = StageAlign.TOP_LEFT;
        removeEventListener(Event.ADDED_TO_STAGE, init);
        stage.addEventListener(Event.RESIZE, onResize);
        stage.addEventListener(Event.ENTER_FRAME, onEnterFrame);
        //
        view = new View3D();
        addChild(view);
        //
        cubeMaterials = new CubeMaterialsData();
        cubeMaterials.front = new MovieMaterial(new brickTexture(),
        ↪{ interactive:true } );
        cubeMaterials.back = new MovieMaterial(new brickTexture(),
        ↪{ interactive:true } );
        cubeMaterials.top = new MovieMaterial(new brickTexture(),
        ↪{ interactive:true } );
        cubeMaterials.bottom = new MovieMaterial(new brickTexture(),
        ↪{ interactive:true } );
        cubeMaterials.left = new MovieMaterial(new brickTexture(),
        ↪{ interactive:true } );
        cubeMaterials.right = new MovieMaterial(new brickTexture(),
        ↪{ interactive:true } );
        wall = new Cube( { width:1024, height:512, depth:50,
        ↪cubeMaterials:cubeMaterials } );
        wall.addOnMouseDown(onMouseDown);
        wall.addOnMouseUp(onMouseUp);
        wall.addEventListener(MouseEvent3D.MOUSE_MOVE, onMouseMove);
        view.scene.addChild(wall);
        //
        onResize();
    }

    private function onMouseDown(e:MouseEvent3D):void
    {
        drawingEnabled = true;
    }
}
```

```

private function onMouseUp(e:MouseEvent3D):void
{
    drawingEnabled = false;
}

private function onMouseMove(e:MouseEvent3D):void
{
    if (drawingEnabled)
    {
        var graff:Sprite = new Sprite();
        graff.graphics.beginFill(0xFF0000, .5);
        graff.graphics.drawCircle(e.uv.u * (e.material as MovieMaterial).
↳movie.width, (e.material as MovieMaterial).movie.height -
↳e.uv.v * (e.material as MovieMaterial).movie.height, 10);
        graff.graphics.endFill();
        (e.material as MovieMaterial).movie.addChild(graft);
    }
}

private function onResize(e:Event = null):void
{
    view.x = stage.stageWidth * .5;
    view.y = stage.stageHeight * .5;
}

private function onEnterFrame(e:Event):void
{
    wall.rotationY -= .1;
    view.render();
}
}
}
}

```

Na początku bloku klasy `GraffitiExample` zdefiniowaliśmy obiekt widoku `View3D`, a następnie obiekt klasy `Cube` o nazwie `wall`. Jak wspomnieliśmy na początku tego punktu, materiałem pokrywającym ścianę jest obiekt klasy `MovieMaterial`. W naszym przykładzie nazwaliśmy go `wallTexture`. Jako źródło dla tego materiału posłużył obiekt klasy `brickTexture`. Jest to element `MovieClip` dodany do biblioteki pliku *fla*. Po zdefiniowaniu wszystkich potrzebnych obiektów utworzyliśmy jeszcze jedną zmienną o nazwie `drawingEnabled`, której wartość określa, czy można rysować po powierzchni ściany.

```

private var view:View3D;
private var wall:Cube;
private var wallTexture:MovieMaterial;
private var textureMC:brickTexture;
private var isDrawing:Boolean = false;

```

W konstruktorze uruchamiamy metodę `init()`, z chwilą gdy obiekt `Stage` zostanie zainicjowany.

```
if (stage) init();
else addEventListener(Event.ADDED_TO_STAGE, init);
```

Gdy metoda `init()` zostanie wywołana, obiekt nasłuchujący zdarzenia `Event.ADDED_TO_STAGE` będzie zbędny, dlatego można go usunąć, stosując metodę `removeEventListener()`.

Dla szybszego działania aplikacji ustawiliśmy niską jakość wyświetlanego obrazu, stosując w tym celu wartość `StageQuality.LOW`. W kolejnych liniach usunęliśmy skalowanie obiektów i wyrównaliśmy stół montażowy do lewego górnego rogu. Ostatnimi operacjami, jakie wykonaliśmy na obiekcie `stage`, było przypisanie im detektorów zdarzeń reagujących na zmianę rozmiarów okna i odświeżanie wyświetlanego obrazu.

```
removeEventListener(Event.ADDED_TO_STAGE, init);
stage.quality = StageQuality.LOW;
stage.scaleMode = StageScaleMode.NO_SCALE;
stage.align = StageAlign.TOP_LEFT;
stage.addEventListener(Event.RESIZE, onResize);
stage.addEventListener(Event.ENTER_FRAME, onEnterFrame);
```

Po ustaleniu wszystkich opcji i dodaniu obiektów nasłuchujących zdarzeń `Event.RESIZE` i `Event.ENTER_FRAME` stworzyliśmy i dodaliśmy widok bez ustalania jakichkolwiek dodatkowych zmian wartości jego właściwości. Położenie zmieniamy w metodzie `onResize()`.

```
view = new View3D();
addChild(view);
```

W kolejnych liniach metody `init()` stworzyliśmy i uzupełniliśmy zawartość obiektu klasy `CubeMaterialsData` o nazwie `cubeMaterials`. Każdemu z boków sześcianu przypisaliliśmy nowy materiał typu `MovieMaterial`. W każdym z tych materiałów źródłem wyświetlanego obrazu jest obiekt klasy `brickTexture`. Dodatkowo aby umożliwić malowanie na powierzchni tego materiału, ustawiliśmy wartość właściwości `interactive` jako `true`.

```
cubeMaterials = new CubeMaterialsData();
cubeMaterials.front = new MovieMaterial(new brickTexture(),
    { interactive:true });
cubeMaterials.back = new MovieMaterial(new brickTexture(),
    { interactive:true });
cubeMaterials.top = new MovieMaterial(new brickTexture(),
    { interactive:true });
cubeMaterials.bottom = new MovieMaterial(new brickTexture(),
    { interactive:true });
cubeMaterials.left = new MovieMaterial(new brickTexture(),
    { interactive:true });
cubeMaterials.right = new MovieMaterial(new brickTexture(),
    { interactive:true });
```

Po wykonaniu tych czynności stworzyliśmy obiekt klasy `Cube` o nazwie `wall`, w którym właściwości `cubeMaterials` przypisaliśmy nasz obiekt `cubeMaterials`. Następnie korzystając z wcześniej poznanych metod `addOnMouseDown()`, `addOnMouseUp()` oraz zwykłego `addEventListener()` dodaliśmy obiekt nasłuchujący zdarzeń `MouseEvent3D`.

```
wall = new Cube( { width:1024, height:512, depth:50,
↳cubeMaterials:cubeMaterials } );
wall.addOnMouseDown(onMouseDown);
wall.addOnMouseUp(onMouseUp);
wall.addEventListener(MouseEvent3D.MOUSE_MOVE, onMouseMove);
```

Na końcu metody `init()` dodaliśmy obiekt ściany do sceny `Away3D` i wywołaliśmy metodę `onResize()` w celu ustawienia widoku na środku okna aplikacji.

```
view.scene.addChild(wall);
onResize();
```

Metody `onMouseDown()` oraz `onMouseUp()` wywoływane są w sytuacji zajścia zdarzenia `MouseEvent3D`. Po wciśnięciu przycisku myszki uruchamia się metoda `onMouseDown()`, w której właściwości `mouseDown` przypisywana jest wartość `true`. Z kolei przy puszczeniu przycisku myszy wywoływana jest metoda `onMouseUp()`, w której `mouseDown` zmienia swoją wartość na `false`.

Najważniejszą metodą w klasie `GraffitiExample` jest `onMouseMove()`. Kod zapisany w jej ciele odpowiada za rysowanie wzorów na powierzchni tekstury obiektu `wall`. Aby malowanie było możliwe jedynie przy wciśniętym przycisku myszy, musieliśmy w pierwszej kolejności zastosować instrukcję warunkową `if`, w której sprawdzana jest wartość właściwości `drawingEnabled`.

Jeżeli wartość tej właściwości równa jest `true`, to wewnątrz instrukcji tworzony jest nowy obiekt klasy `Sprite`.

```
var graff:Sprite = new Sprite();
```

Stosując odwołanie do obiektu `graphics` wewnątrz obiektu `graff`, można korzystać z jego metod rysowania.

Do narysowania półprzezroczystego czerwonego koła użyliśmy metod `beginFill()` oraz `drawCircle()`. W metodzie `beginFill()` jako argument podaliśmy szesnastkowy kod koloru czerwonego.

```
graff.graphics.beginFill(0xFF0000, .5);
```

Metoda `drawCircle()` przyjmuje trzy właściwości: `x`, `y`, oraz `radius`. Jako `x` podaliśmy wartość właściwości `u` z obiektu `UV` i pomnożyliśmy ją przez szerokość tekstury.

```
e.uv.u * (e.material as MovieMaterial).movie.width;
```

Z kolei aby określić wartość argumentu y , musieliśmy od wysokości tekstury odjąć iloczyn tej wysokości i wartości właściwości v z obiektu uv . Dzięki temu współrzędne tekstury będą zgodne z ruchem kursora.

```
(e.material as MovieMaterial).movie.height - e.uv.v * (e.material as  
MovieMaterial).movie.height;
```

Etap rysowania zakończyliśmy metodą `endFill()`, po czym umieściliśmy nowo utworzony obiekt `graff` w wyświetlanym obiekcie klasy `brickTexture`.

```
graff.graphics.endFill();  
(e.material as MovieMaterial).movie.addChild(graff);
```

Warto wspomnieć o tym, że zapis `(e.material as MovieMaterial).movie` odwołuje się do źródła materiału, na którym aktualnie znajduje się kursor.

Rozmieszczanie obiektów na planszy

Przyjmijmy, że Twoim zadaniem jest napisanie strategicznej gry czasu rzeczywistego. Jednymi z kluczowych elementów takiej gry są:

- ♦ możliwość zmiany pozycji obiektów strategicznych,
- ♦ tworzenie oraz przemieszczanie jednostek bojowych.

Tymi konkretnymi zagadnieniami zajmiemy się w tym punkcie, a efekt, który uzyskamy, kompilując kod źródłowy przykładu, przedstawia rysunek 8.4.

Rysunek 8.4.

Rozmieszczanie obiektów na planszy



Na rysunku 8.4 widzimy kawałek planszy pokryty teksturą ze sztucznym ukształtowaniem terenu tworzącym pole podzielone wyschniętym korytem rzeki. Na tej planszy umieszczony jest jeden budynek, który w naszym scenariuszu pełni funkcję fabryki pojazdów latających. Poza nim w powietrzu wiszą cztery wyprodukowane statki kosmiczne.

Tworzenie tych latających obiektów następuje po kliknięciu na obiekt fabryki — pojazd pojawia się na jej platformie, po czym przemieszcza na losowo wybraną pozycję.

Najechanie kursorem na którykolwiek z modeli powoduje pojawienie się czerwonej poświaty. Oznacza to, że na wybranym elemencie można wykonać akcję zmiany pozycji.

Żeby zmienić pozycję wybranego obiektu, należy kliknąć na niego, trzymając wciśnięty klawisz *Ctrl*.

Na podłożu planszy pojawi się niebieski celownik, tak jak to pokazano na rysunku 8.4. Celownik ten, podobnie jak w grach tego typu, wyznacza nowe miejsce docelowe.

Aby wybrany obiekt zmienił swoją pozycję na nową, należy kliknąć lewy przycisk myszy, trzymając wciśnięty klawisz *Ctrl*. W przypadku obiektu fabryki pozycja zmieniona zostanie natychmiastowo, ponieważ — w przeciwieństwie do obiektów statków — nie stworzyliśmy animacji płynnego przejścia wygenerowanej za pomocą bibliotek *TweenMax* i *TimelineMax*. Wszystkie procesy, które zachodzą w kodzie źródłowym tego przykładu, omówimy za chwilę. Teraz przepisz ten kod i załącz go jako główną klasę programu. Do poprawnego działania aplikacji niezbędne są zasoby umieszczone w pliku *fla*.

```
package
{
    import flash.display.StageAlign;
    import flash.display.StageQuality;
    import flash.display.StageScaleMode;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.KeyboardEvent;
    import flash.filters.GlowFilter;
    import flash.geom.Vector3D;
    import flash.ui.Keyboard;
    //
    import away3d.core.base.Object3D;
    import away3d.core.utils.Cast;
    import away3d.core.render.Renderer;
    import away3d.primitives.Plane;
    import away3d.materials.MovieMaterial;
    import away3d.materials.BitmapMaterial;
```



```
import away3d.events.MouseEvent3D;
import away3d.events.Loader3DEvent;
import away3d.containers.ObjectContainer3D;
import away3d.containers.View3D;
import away3d.loaders.Loader3D;
import away3d.loaders.Max3DS;
//
import com.greensock.TweenMax;
import com.greensock.TimelineMax;

public class MouseEvent3DExample extends Sprite
{
    private var view:View3D;
        private var pointer:Plane;
    private var pointerMC:Crosshair = new Crosshair();
    private var board:ObjectContainer3D;
    private var ground:Plane;
    private var texture:BitmapMaterial;
    private var modelLoader:Loader3D;
    private var landingDock:ObjectContainer3D;
    private var fighter:ObjectContainer3D;
    private var animations:TimelineMax;
    private var leftArrowDown:Boolean = false;
    private var rightArrowDown:Boolean = false;
    private var upArrowDown:Boolean = false;
    private var downArrowDown:Boolean = false;
    private var selectedBuilding:Object3D;
    private var selectedVehicle:Object3D;
    private var ctrlDown:Boolean = false;
    private var zoomInDown:Boolean = false;
    private var zoomOutDown:Boolean = false;
    private var fighters:Vector.<Object3D> = new Vector.<Object3D>;

    public function MouseEvent3DExample():void
    {
        if (stage) init();
        else addEventListener(Event.ADDED_TO_STAGE, init);
    }

    private function init(e:Event = null):void
    {
        stage.quality = StageQuality.LOW;
        stage.scaleMode = StageScaleMode.NO_SCALE;
        stage.align = StageAlign.TOP_LEFT;
        removeEventListener(Event.ADDED_TO_STAGE, init);
        addEventListener(Event.ENTER_FRAME, onEnterFrame);
        stage.addEventListener(Event.RESIZE, onResize);
        stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown);
        stage.addEventListener(KeyboardEvent.KEY_UP, onKeyUp);
        view = new View3D();
        view.renderer = Renderer.BASIC;
        view.x = stage.stageWidth * .5;
```

```

        view.y = stage.stageHeight * .5;
        addChild(view);
        view.camera.zoom = 15;
        view.camera.y = 800;
        view.camera.lookAt(new Vector3D(0, 0, 0));
        view.scene.addOnMouseMove(overScene);

        pointer = new Plane( { width:34, height:34,
        ↪material:new MovieMaterial(pointerMC),pushfront:true } );
        initBoard();
        initBuildings();
        onResize();
    }

private function initBoard():void
{
    board = new ObjectContainer3D();
    texture = new BitmapMaterial(Cast.bitmap('terrain'));
    ground = new Plane();
    ground.width = 512;
    ground.height = 512;
    ground.pushback = true;
    ground.material = texture;
    ground.addOnMouseDown(onGroundClick);
    board.addChild(ground);
    view.scene.addChild(board);
}

private function overScene(e:MouseEvent3D):void
{
    pointer.x = e.sceneX;
    pointer.z = e.sceneZ;
}

private function onGroundClick(e:MouseEvent3D):void
{
    trace(e.sceneX, e.sceneZ);
    if (ctrlDown && selectedBuilding)
    {
        selectedBuilding.alpha = 1;
        selectedBuilding.x = pointer.x;
        selectedBuilding.z = pointer.z;
        selectedBuilding = null;
        view.scene.removeChild(pointer);
    }
    else if (ctrlDown && selectedVehicle)
    {
        view.scene.removeChild(pointer);
        TweenMax.to(selectedVehicle, 1, { x:pointer.x, z:pointer.z,
        ↪onComplete:function() { selectedBuilding = null; } } );
    }
    else{}
}

```

```
}
private function initBuildings():void
{
    modelLoader = Max3DS.load('../resources/models/max3ds/
↳landingDock/LandingDock.3DS');
    modelLoader.addOnSuccess(onLandingDockSuccess);
}

private function onLandingDockSuccess(e:Loader3DEvent):void
{
    landingDock = new ObjectContainer3D();
    landingDock.ownCanvas = true;
    landingDock.name = 'RED_landingDock';
    landingDock.addChild(modelLoader.handle);
    landingDock.scale(.2);
    landingDock.x = 71;
    landingDock.z = 196;
    landingDock.y = 15;
    landingDock.rotationX = 90;
    landingDock.filters = [ new GlowFilter(0xFF0000, 0, 7, 7, 1, 1)];
    view.scene.addChild(landingDock);
    landingDock.addOnMouseDown(landingDockOnMouseDown);
    landingDock.addOnRollOver(landingDockOnMouseRollOver);
    landingDock.addOnRollOut(landingDockOnMouseRollOut);
}

private function landingDockOnMouseDown(e:MouseEvent3D):void
{
    trace('landingDock:CLICK');
    if (ctrlDown)
    {
        selectedBuilding = landingDock;
        selectedBuilding.alpha = .5;
        view.scene.addChild(pointer);
        pointerMC.gotoAndStop('move');
    }else
    {
        modelLoader = Max3DS.load('../resources/models/max3ds/
↳spaceFighter01/spaceFighter01.3ds');
        modelLoader.addOnSuccess(onModelLoaderSuccess);
    }
}

private function landingDockOnMouseRollOver(e:MouseEvent3D):void
{
    trace('landingDock:RollOver');
    TweenMax.to(landingDock.filters[0], .5, { alpha:1 } );
}

private function landingDockOnMouseRollOut(e:MouseEvent3D):void
{
    trace('landingDock:RollOut');
```

```

        TweenMax.to(landingDock.filters[0], .5, { alpha:0 } );
    }

    private function randomPosition(min:Number, max:Number):Number
    {
        return Math.floor(Math.random() * (1 + max - min)) + min;
    }

    private function onModelLoaderSuccess(e:Loader3DEvent):void
    {
        var randomX:Number = landingDock.x - randomPosition(-150, 150);
        var randomZ:Number = landingDock.z - randomPosition(-150, 150);
        fighter = new ObjectContainer3D();
        fighter.ownCanvas = true;
        fighter.addOnMouseDown(fighterOnMouseDown);
        fighter.addOnRollOver(fighterOnMouseRollOver);
        fighter.addOnRollOut(fighterOnMouseRollOut);
        fighter.name = fighter + fighters.length;
        fighter.filters = [new GlowFilter(0xFF0000, 0, 7, 7, 1, 1)];
        fighter.addChild(modelLoader.handle);
        fighter.scale(.001);
        fighter.rotationX = 90;
        fighter.x = landingDock.x;
        fighter.y = 20;
        fighter.z = landingDock.z;
        view.scene.addChild(fighter);
        fighters.push(fighter);
        animations = new TimelineMax( { onComplete:onFighterReady } );
        animations.append(TweenMax.to(fighter, 1, { scaleX:.1,
        ↪scaleY:.1, scaleZ:.1 } ));
        animations.append(TweenMax.to
        ↪(fighter, 2, { x:randomX, y: 60, z:randomZ } ));
        animations.play();
    }

    private function onFighterReady():void
    {
        trace('Fighter ready!');
    }

    private function fighterOnMouseDown(e:MouseEvent3D):void
    {
        trace('Fighter:CLICK');
        if (ctrlDown)
        {
            selectedVehicle = e.target as ObjectContainer3D;
            view.scene.addChild(pointer);
            pointerMC.gotoAndStop('move');
        }
    }

    private function fighterOnMouseRollOver(e:MouseEvent3D):void

```

```
{
    trace('Fighter:RollOver');
    TweenMax.to(e.currentTarget.filters[0], .5, { alpha:1 } );
}

private function fighterOnMouseRollOut(e:MouseEvent3D):void
{
    trace('Fighter:RollOut');
    TweenMax.to(e.currentTarget.filters[0], .5, { alpha:0 } );
}

private function onResize(e:Event = null):void
{
    view.x = stage.stageWidth * .5;
    view.y = stage.stageHeight * .5;
}

private function onKeyDown(e:KeyboardEvent):void
{
    if (e.keyCode == Keyboard.LEFT)    leftArrowDown = true;
    if (e.keyCode == Keyboard.RIGHT)   rightArrowDown = true;
    if (e.keyCode == Keyboard.UP)      upArrowDown = true;
    if (e.keyCode == Keyboard.DOWN)    downArrowDown = true;
    if (e.keyCode == Keyboard.CONTROL) ctrlDown = true;
    if (e.keyCode == 187) zoomInDown = true;
    if (e.keyCode == 189) zoomOutDown = true;
}

private function onKeyUp(e:KeyboardEvent):void
{
    if (e.keyCode == 187) zoomInDown = false;
    if (e.keyCode == 189) zoomOutDown = false;
    if (e.keyCode == Keyboard.LEFT) leftArrowDown = false;
    if (e.keyCode == Keyboard.RIGHT) rightArrowDown = false;
    if (e.keyCode == Keyboard.UP) upArrowDown = false;
    if (e.keyCode == Keyboard.DOWN) downArrowDown = false;
    if (e.keyCode == Keyboard.CONTROL)
    {
        ctrlDown = false;
        if (selectedBuilding)
        {
            selectedBuilding.alpha = 1;
            selectedBuilding = null;
            view.scene.removeChild(pointer);
        }
    }
}

private function onEnterFrame(e:Event):void
{
    if (zoomInDown)
    {
        if (ctrlDown) view.camera.y += 5;
    }
}
```

```

        else view.camera.zoom++;
    }
    if (zoomOutDown)
    {
        if (ctrlDown) view.camera.y -= 5;
        else view.camera.zoom--;
    }
    if (leftArrowDown)
    {
        if (ctrlDown) view.camera.rotationY++;
        else view.camera.x -= 10;
    }
    if (rightArrowDown)
    {
        if (ctrlDown) view.camera.rotationY--;
        else view.camera.x += 10;
    }
    if (upArrowDown)
    {
        if (ctrlDown) view.camera.rotationX++;
        else view.camera.z += 10;
    }
    if (downArrowDown)
    {
        if (ctrlDown) view.camera.rotationX--;
        else view.camera.z -= 10;
    }
    view.render();
}
}
}

```

Jak w większości kodów źródłowych zawartych w tej książce, ustawiliśmy poziom jakości wyświetlania elementów, brak skalowania oraz wyrównanie stołu montażowego. Użyliśmy do tego celu klas `StageAlign`, `StageQuality` i `StageScaleMode`.

Jednym z istotnych elementów tego przykładu jest możliwość zmiany położenia obiektów znajdujących się na planszy. Aby móc wyznaczyć nową pozycję na trzech osiach równocześnie, musieliśmy skorzystać z klasy `Vector3D`. Do samego przemieszczenia użyliśmy zdarzeń `MouseEvent3D` i animacji generowanych klasami `TweenMax` i `TimelineMax`.

Ponieważ do zrealizowania tego przykładu skorzystaliśmy z modeli zapisanych w formacie 3DS, musieliśmy użyć kilku klas, aby wyświetlić je na scenie. W pierwszej kolejności użyliśmy ogólnej klasy `Loader3D`, przechowującej model. Aby pobrać obiekt w formacie 3DS, skorzystaliśmy z klasy `Max3DS`. Skala oraz położenie zawartości modelu uzależnione są od zastosowanych podczas eksportu ustawień. Pobrany z określonej lokalizacji obiekt modelu umieściliśmy w kontenerze `ObjectContainer3D`, aby w jego wnętrzu dostosować obiekt modelu do ustawień sceny.

Wspomniane otoczenie stworzyliśmy za pomocą klasy `Plane` i nadaliśmy odpowiedni charakter powierzchni, stosując klasę `Cast` oraz `BitmapMaterial`.

Przy wybieraniu nowej pozycji dla obiektów za śladem kursora podąża niebieski celownik. Podobnie jak w grach strategicznych, pokazuje on nowe docelowe położenie na planszy. Do jego wykonania zastosowaliśmy obiekt klasy `Plane` oraz teksturę `MovieMaterial`. Źródłem dla tego materiału jest osadzony w zasobach pliku `fla` element `MovieClip`.

Wewnątrz klasy `MouseEvent3DExample` zadeklarowaliśmy kilkanaście obiektów i zmiennych potrzebnych do prawidłowego funkcjonowania przykładu. Jak zwykle zaczęliśmy od zdefiniowania widoku `Away3D`.

Wskaźnikiem nowych pozycji jest obiekt klasy `Plane` o nazwie `pointer`. Dla źródła materiału wykorzystaliśmy obiekt klasy `Crosshair`. `Crosshair` nie zawiera w sobie żadnych dodatkowych metod poza konstruktorem. Służy jedynie jako łącznik z elementem `MovieClip` z biblioteki pliku `fla`.

Tego typu połączenia mogą wydawać się zbędne. Jednak gdyby projekt w swoich założeniach wymagał dodatkowych operacji wykonywanych na wskaźniku, dobrze byłoby umieścić je w osobnej klasie, takiej jak `Crosshair`, i odwoływać się do nich poprzez obiekt tej klasy.

W następnej kolejności zadeklarowaliśmy obiekty potrzebne do wyświetlenia planszy. Do stworzenia powierzchni zastosowaliśmy obiekt klasy `Plane` o nazwie `ground`, a do pokrycia jej teksturą użyliśmy obiektu `texture` klasy `BitmapMaterial`. Przyjmując, że do samego otoczenia moglibyśmy dodać jeszcze inne elementy, wygenerowaną powierzchnię umieściliśmy w kontenerze `board`. W razie konieczności modyfikowania środowiska wystarczyłoby odwołanie się do tego obiektu.

W tym przykładzie za pobieranie modeli odpowiada obiekt o nazwie `modelLoader` klasy `Loader3D`. W przypadku budynku fabryki pobrana zawartość umieszczana jest w kontenerze o nazwie `landingDock`. Z kolei każdy utworzony statek kosmiczny przypisywany jest do obiektu `fighter` i dodawany do wektora `Vector.<Object3D>` o nazwie `fighters`.

Aby wyselekcjonować konkretny element z całej grupy dodanych modeli, zdefiniowaliśmy specjalne obiekty klasy `Object3D` o nazwach `selectedBuilding` i `selectedVehicle`.

Do opanowania animacji złożonych z więcej niż jednej sekwencji posłużyliśmy się obiektem `animations` klasy `TimelineMax`.

Poza samymi obiektami zadeklarowaliśmy również szereg zmiennych potrzebnych do wykonywania operacji w konkretnych przypadkach. Wszystkie te właściwości

przyjmują wartości typu Boolean i odpowiadają za kliknięcie poszczególnych klawiszy klawiatury.

Do określania stanu klawiszy strzałek zastosowaliśmy zmienne: `leftArrowDown`, `rightArrowDown`, `upArrowDown` oraz `downArrowDown`, których wartości domyślnie równe są `false`. W zależności od tego, który klawisz został wciśnięty, odpowiadającej zmiennej przypisujemy nową wartość równą `true`.

Za zmianę wartości właściwości `ctrlDown` odpowiada wciśnięcie lub zwolnienie klawisza *Ctrl*. Z kolei klawisze `+` i `-` umieszczone w bloku klawiszy alfanumerycznych regulują wartości właściwości `zoomInDown` oraz `zoomOutDown`.

W konstruktorze uruchamiamy metody `init()`, z chwilą gdy obiekt `Stage` zostanie zainicjowany.

```
if (stage) init();
else addEventListener(Event.ADDED_TO_STAGE, init);
```

W metodzie `init()` w pierwszej kolejności wykonaliśmy wszystkie potrzebne operacje na obiekcie stołu montażowego. Ustawiliśmy jakość generowanego obrazu, wyrównanie oraz wyłączyliśmy skalowanie. Następnie dodaliśmy detektory dla zdarzeń zmiany rozmiaru, odświeżenia obrazu i użycia klawiatury.

```
stage.quality = StageQuality.LOW;
stage.scaleMode = StageScaleMode.NO_SCALE;
stage.align = StageAlign.TOP_LEFT;
removeEventListener(Event.ADDED_TO_STAGE, init);
addEventListener(Event.ENTER_FRAME, onEnterFrame);
stage.addEventListener(Event.RESIZE, onResize);
stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown);
stage.addEventListener(KeyboardEvent.KEY_UP, onKeyUp);
```

W dalszej części metody `init()` stworzyliśmy widok `Away3D` i zmieniliśmy ustawienia kamery tak, aby obejmowała całą powierzchnię planszy.

Do renderowania widoku skorzystaliśmy z klasy `BasicRenderer`, której obiekt utworzyliśmy, stosując odwołanie `Renderer.BASIC`. Przed umieszczeniem widoku na liście wyświetlanych obiektów wywołaliśmy metodę `addOnMouseMove()`, która przy zdarzeniu poruszania kursorem ma wywołać metodę `overScene()`.

```
view = new View3D();
view.renderer = Renderer.BASIC;
view.scene.addOnMouseMove(overScene);
addChild(view);
view.camera.zoom = 15;
view.camera.y = 800;
view.camera.lookAt(new Vector3D(0, 0, 0));
```


Po dodaniu widoku w celach testowych umieściliśmy obiekt klasy `Trident`. Długość jego ramion ustawiliśmy na poziomie 100 pikseli i zmieniliśmy wartość właściwości `showLetters` na `true`.

Linijkę kodu dodającą oś `Trident` do sceny `Away3D` umieściliśmy w komentarzu tak, aby nie wyświetlać obiektu `trident` bez potrzeby.

```
trident = new Trident(100, true);  
//view.scene.addChild(trident);
```

Dalej w metodzie `init()` stworzyliśmy wskaźnik pozycji, który wcześniej zdefiniowaliśmy jako obiekt `pointer`. Jego wysokość oraz szerokość ustawiliśmy tak, aby zgadzały się z wymiarami źródła grafiki `pointerMC`. Ze względu na swoją rolę wskaźnik jest zawsze widoczny na powierzchni planszy. Aby uzyskać taki efekt, nie podnieśliśmy obiektu na osi `Y`, tylko zastosowaliśmy właściwość `pushfront`. Przypisanie jej wartości `true` powoduje wyświetlenie siatki wybranego obiektu nad pozostałymi znajdującymi się w tej samej pozycji `Y`.

```
pointer = new Plane( { width:34, height:34, material:new  
MovieMaterial(pointerMC),pushfront:true } );
```

Na końcu metody `init()` odwołaliśmy się do dwóch metod tworzących potrzebne modele oraz metody ustawiającej pozycję widoku.

```
initBoard();  
initBuildings();  
onResize();
```

W metodzie `initBoard()` poza stworzeniem planszy i nadaniem jej odpowiednich ustawień ważne było zastosowanie metody uruchamiającej metodę `onGroundClick()` przy każdym kliknięciu na powierzchni planszy.

```
ground.addOnMouseDown(onGroundClick);
```

Metoda `overScene()` wywoływana jest w sytuacji poruszania kursorem w przestrzeni sceny. W tym przypadku interesowały nas jedynie współrzędne osi `X` oraz `Z`, które przypisaliśmy do pozycji obiektu `pointer`.

```
pointer.x = e.sceneX;  
pointer.z = e.sceneZ;
```

W metodzie `onGroundClick()` zapisaliśmy proces przemieszczania wybranego obiektu. Wewnątrz ciała metody w pierwszej kolejności znajdują się instrukcje warunkowe, które sprawdzają, czy został wciśnięty klawisz `Ctrl` oraz jakiego rodzaju jest kliknięty obiekt. Zdefiniowaliśmy dwie możliwe sytuacje. Pierwsza dotyczy kliknięcia obiektu `landingDock` reprezentującego budynek fabryki. W tym przypadku przejście między pozycjami jest natychmiastowe, ponieważ nie zastosowaliśmy tutaj klasy `TweenMax`, tylko od razu przypisaliśmy właściwościom `x` i `z` nowe

wartości pozycji wskaźnika pointer. Poza tym przywróciliśmy pierwotny stan przezroczystości obiektu landingDock, wyczyściliśmy zawartość obiektu selected ↪Building i usunęliśmy ze sceny wskaźnik.

```
if (ctrlDown && selectedBuilding)
{
    selectedBuilding.alpha = 1;
    selectedBuilding.x = pointer.x;
    selectedBuilding.z = pointer.z;
    selectedBuilding = null;
    view.scene.removeChild(pointer);
}
```

Druga sytuacja określona w metodzie onGroundClick() dotyczy pojazdów latających, czyli obiektów fighter. W ich przypadku aby zmienić pozycję, stosuje się klasę TweenMax, w której właściwościom x i z przypisaliśmy nowe wartości położenia wskaźnika. Dodatkowo wewnątrz metody to() klasy TweenMax zapisaliśmy metodę, która zostanie wywołana na koniec animacji. Celem tej metody jest usunięcie ze sceny obiektu pointer oraz wyczyszczenie odwołania do klikniętego pojazdu w obiekcie selectedVehicle.

```
else if (ctrlDown && selectedVehicle)
{
    TweenMax.to(selectedVehicle, 1, { x:pointer.x, z:pointer.z,
    onComplete:function()
    {
        view.scene.removeChild(pointer);
        selectedVehicle = null;
    }
    } );
}
```

Aby pobrać z wybranej lokalizacji model fabryki w metodzie initBuildings(), skorzystalismy z obiektu modelLoader oraz klasy Max3DS. Dodatkowo użyliśmy detektora zdarzeń Loader3DEvent, aby przy zakończeniu pobierania wywołać metodę onLandingDockSuccess().

```
modelLoader =
Max3DS.load('../resources/models/max3ds/landingDock/LandingDock.3DS');
modelLoader.addOnSuccess(onLandingDockSuccess);
```

Dopiero w metodzie onLandingDockSuccess() zapisaliśmy proces dodawania modelu do sceny Away3D. W pierwszej kolejności nadaliśmy mu nazwę oraz umieściliśmy zawartość obiektu modelLoader w kontenerze landingDock. Ponieważ w swoich pierwotnych ustawieniach wymiary modelu są znacznie większe od wymiarów dodanej planszy, musieliśmy zastosować metodę scale() do zmniejszenia skali obiektu landingDock.

```
landingDock = new ObjectContainer3D();
landingDock.name = 'RED_landingDock';
```

```
landingDock.addChild(modelLoader.handle);
landingDock.scale(.2);
```

W kolejnych liniijkach ustawiliśmy model w wybranym miejscu i przypisaliśmy mu filtr czerwonej poświaty, który będzie się pojawiał po każdym najechaniu kursora na obiekt.

```
landingDock.x = 71;
landingDock.z = 196;
landingDock.y = 15;
landingDock.filters = [ new GlowFilter(0xFF0000, 0, 7, 7, 1, 1)];
```



Aby poświatą była widoczna na obiekcie 3D, należy przypisać jego właściwość `ownCanvas` wartość `true`.

Do rozpoznawania akcji myszy na powierzchni obiektu `landingDock` zastosowaliśmy trzy metody zdarzeń `MouseEvent3D`.

```
landingDock.addOnMouseDown(landingDockOnMouseDown);
landingDock.addOnRollOver(landingDockOnMouseRollOver);
landingDock.addOnRollOut(landingDockOnMouseRollOut);
```

Kliknięcie na obiekt `landingDock` powoduje uruchomienie metody `landingDockOnMouseMouseDown()`. W jej ciele sprawdzamy, czy wartość właściwości `ctrlDown` jest równa `true`. Jeżeli warunek jest spełniony, to metoda ta spowoduje zmianę pozycji obiektu fabryki i wyświetlenie wskaźnika.

```
if (ctrlDown)
{
    selectedBuilding = landingDock;
    selectedBuilding.alpha = .5;
    view.scene.addChild(pointer);
    pointerMC.gotoAndStop('move');
}
```

W sytuacji gdy wartość właściwości `ctrlDown` jest równa `false`, celem metody `landingDockOnMouseDown()` jest stworzenie nowego obiektu statku kosmicznego. Proces ten zaczyna się podobnie jak tworzenie modelu fabryki. W pierwszej kolejności za pomocą klasy `Max3DS` pobierany jest z określonej lokalizacji model `SpaceFighter01.3DS`. Następnie z chwilą zakończenia pobierania wywołujemy metodę `onModelLoaderSuccess()`.

```
else
{
    modelLoader = Max3DS.load('./models/sf/spaceships/SpaceFighter01.3DS');
    modelLoader.addOnSuccess(onModelLoaderSuccess);
}
```

W metodzie `onModelLoaderSuccess()` zapisaliśmy cały proces tworzenia i pokazania modelu statku kosmicznego. Według scenariusza obiekt ma się pojawić na powierzchni budynku, zwiększyć swoją pozycję na osi Y i przenieść w przypadkowe miejsce wokół fabryki.

Żeby uzyskać taki efekt, w pierwszej kolejności zdefiniowaliśmy dwie zmienne numeryczne: `randomX` i `randomY`, których wartości są różnicą pozycji modelu fabryki i losowej liczby z przedziału. Do wyznaczenia przypadkowej liczby napisaliśmy osobną metodę `randomPosition()`, którą omówimy później.

```
var randomX:Number = landingDock.x - randomPosition(-150, 150);  
var randomZ:Number = landingDock.z - randomPosition(-150, 150);
```

W dalszej części metody `onModelLoaderSuccess()` stworzyliśmy obiekt `fighter`, który przechowuje pobraną zawartość obiektu `modelLoader`.

```
fighter = new ObjectContainer3D();  
fighter.addChild(modelLoader.handle);
```

Kolejność definiowania właściwości nie ma tutaj większego znaczenia, dlatego zaczęliśmy od dodania detektorów zdarzeń `MouseEvent3D`. Tak samo jak w przypadku obiektu `landingDock`, na każdy ze statków kosmicznych będzie można kliknąć, najechać i opuścić kursorem powierzchnię modelu.

```
fighter.addOnMouseDown(fighterOnMouseDown);  
fighter.addOnRollOver(fighterOnMouseRollOver);  
fighter.addOnRollOut(fighterOnMouseRollOut);
```

Następnie dodaliśmy filtr czerwonej poświaty, który będzie się pojawiał, gdy kursor znajdzie się na powierzchni modelu.

```
fighter.filters = [new GlowFilter(0xFF0000, 0, 7, 7, 1, 1)];
```

Ponieważ pojazd ma się wylaniać z hangaru, dla animacji ustawiliśmy jego pozycję zgodną z położeniem fabryki oraz skalę na poziomie jednej tysięcznej.

```
fighter.scale(.001);  
fighter.x = landingDock.x;  
fighter.y = 20;  
fighter.z = landingDock.z;
```

Po utworzeniu statku i przypisaniu jego właściwościom nowych współrzędnych dodaliśmy go do sceny oraz obiektu `fighters`.

```
view.scene.addChild(fighter);  
fighters.push(fighter);
```

Na końcu metody `onModelLoaderSuccess()` stworzyliśmy animację złożoną z dwóch etapów. W pierwszym powiększyliśmy skalę obiektu, co ma symbolizować jego powstawanie, w drugiej zaś przenieśliśmy go do losowych pozycji `x` i `z` na wysokości 100 pikseli.

```

animations = new TimelineMax( { onComplete:onFighterReady } );
animations.append(TweenMax.to(fighter, 1, { scaleX:.4, scaleY:.4,
↳scaleZ:.4 } ));
animations.append(TweenMax.to(fighter, 2, { x:randomX, y: 100,
↳z:randomZ } ));
animations.play();

```



Do tworzenia animacji złożonej z kilku etapów najlepiej stosować klasę `TimelineMax`. Poszczególne fazy animacji dodaje się metodą `append()`, w której argumentem jest odwołanie do klasy `TweenMax`. Gotową sekwencję animacji można uruchomić, stosując metody `play()` lub `restart()`.

Szczegóły dotyczące klasy `TweenMax` znajdują się na stronie:
<http://www.greensock.com/timelinemax/>.

Metoda `randomPosition()`, jak wspomnieliśmy wcześniej, służy do wygenerowania losowej liczby. Do wyliczenia tej wartości posłużyliśmy się metodami klasy `Math` oraz przedziału liczbowego podanego w argumentach `min` i `max`.

```
return Math.floor(Math.random() * (1 + max - min)) + min;
```

W metodach `landingDockOnMouseRollOver()` i `landingDockOnMouseRollOut()` zapisaliśmy animację pokazywania i chowania poświaty. Aby płynnie przejść od jednego stanu do drugiego, skorzystaliśmy z klasy `TweenMax`, w której odwołujemy się do pierwszej pozycji z tablicy `filters` obiektu `landingDock` i zmieniamy wartość właściwości `alpha`.

```

//landingDockOnMouseRollOver
TweenMax.to(landingDock.filters[0], .5, { alpha:1 } );

//landingDockOnMouseRollOut
TweenMax.to(landingDock.filters[0], .5, { alpha:0 } );

```

Na tej samej zasadzie zakodowaliśmy pojawianie się poświaty wokół obiektów `fighter` w metodach `fighterOnMouseRollOver()` i `fighterOnMouseRollOut()`. Jediną różnicą jest odwołanie do wybranego obiektu. Ponieważ w tym przypadku liczba pojazdów latających może być większa niż jeden, aby animacja dotyczyła konkretnego obiektu, musieliśmy w klasie `TweenMax` użyć zapisu `e.currentTarget.filters[0]`.

Metoda `fighterOnMouseDown()` odpowiada za pojawienie się wskaźnika i rozpoczęcie procesu przemieszczania obiektu latającego. Żeby sprawdzić, czy użytkownik, klikając na obiekt, chce go przenieść w inne miejsce, zastosowaliśmy instrukcję `if` sprawdzającą, czy zmienna `ctrlDown` jest równa `true`. Jeżeli ma ona wartość `true`, na scenie w pozycji kursora pojawia się wskaźnik. Żeby przejście dotyczyło konkretnego pojazdu, przypisaliśmy obiektowi `selectedVehicle` odwołanie do klikniętego pojazdu.

```

if (ctrlDown)
{

```

```

        selectedVehicle = e.target as ObjectContainer3D;
        view.scene.addChild(pointer);
        pointerMC.gotoAndStop('move');
    }

```

Przy każdej zmianie rozmiaru okna uruchamiana jest metoda `onResize()`. W jej ciele zapisaaliśmy wyśrodkowanie widoku `view` w oknie programu.

```

view.x = stage.stageWidth * .5;
view.y = stage.stageHeight * .5;

```

Znane nam z wcześniejszych przykładów metody `onKeyDown()` oraz `onKeyUp()` służą do sprawdzania, czy zaszły warunki zapisane w instrukcjach `if`, i przypisania odpowiedniej wartości poszczególnym właściwościom. W obu metodach tego przykładu sprawdzaliśmy akcje wciśnięcia i zwolnienia klawiszy strzałek, *Ctrl* oraz znaków plusa i minusa znajdujących się w górnym rzędzie klawiatury. W metodzie `onKeyUp()` przy zwolnieniu klawisza *Ctrl* dodatkowo sprawdziliśmy, czy kursor znajdował się na obiekcie `landingDock`. Jeżeli tak, to musieliśmy zapisać powrót do pierwotnego stanu poziomu jego przezroczystości, wyczyścić zawartość obiektu `selectedBuilding` i usunąć wskaźnik ze sceny.

```

if (e.keyCode == Keyboard.CONTROL)
{
    ctrlDown = false;
    if (selectedBuilding)
    {
        selectedBuilding.alpha = 1;
        selectedBuilding = null;
        view.scene.removeChild(pointer);
    }
}

```

Na końcu zapisaaliśmy metodę `onEnterFrame()`, która wywoływana jest przy każdym wystąpieniu zdarzenia `Event.ENTER_FRAME`. Zawiera ona w sobie kilka instrukcji warunkowych. W każdej z nich dodatkowo sprawdzamy, czy został wciśnięty klawisz *Ctrl*.

Zastosowanie kombinacji klawiszy pozwoliło nam uniknąć korzystania z większej ich liczby.

Pierwsza instrukcja `if` sprawdza, czy wartość zmiennej `zoomInDown` równa jest `true`. Jeżeli warunek jest spełniony, to w zależności od stanu zmiennej `ctrlDown` wykonywana jest odpowiednia operacja. Gdy klawisz *Ctrl* jest wciśnięty, to kamera zmienia swoją pozycję na osi *Y*. W przeciwnym razie zwiększana jest wartość właściwości `zoom`.

```

if (zoomInDown)
{
    if (ctrlDown) view.camera.y += 5;
}

```

```
    else view.camera.zoom++;  
}
```

W przypadku drugiej instrukcji `if` sprawdzana jest wartość zmiennej `zoomOutDown`. Gdy wartość tej właściwości jest równa `true`, to kamera może zmieniać swoją pozycję na osi `Y` lub zmniejszać wartość właściwości `zoom`.

```
if (zoomOutDown)  
{  
    if (ctrlDown) view.camera.y -= 5;  
    else view.camera.zoom--;  
}
```

Trzecia instrukcja `if` dotyczy zdarzenia wciśnięcia klawisza lewej strzałki. Jeżeli warunek jest spełniony, to kamera może wykonywać obrót w lewą stronę lub zmniejszyć pozycję na osi `X`.

```
if (leftArrowDown)  
{  
    if (ctrlDown) view.camera.rotationY--;  
    else view.camera.x -= 10;  
}
```

Działania czwartej instrukcji warunkowej `if` są odwrotne do poprzedniej. Dotyczy ona bowiem wciśnięcia klawisza prawej strzałki. Gdy wartość właściwości `rightArrowDown` jest równa `true`, to kamera może obracać się w prawą stronę bądź zwiększać pozycję na osi `X`.

```
if (rightArrowDown)  
{  
    if (ctrlDown) view.camera.rotationY++;  
    else view.camera.x += 10;  
}
```

W piątej instrukcji `if` sprawdzana jest wartość zmiennej `upArrowDown`. Jeżeli warunek jest spełniony, to kamera może wykonywać obrót w górę lub zwiększyć pozycję na osi `Z`.

```
if (upArrowDown)  
{  
    if (ctrlDown) view.camera.rotationX++;  
    else view.camera.z += 10;  
}
```

W ostatniej instrukcji `if` warunkiem jest wciśnięcie klawisza dolnej strzałki. Gdy wartość właściwości `downArrowDown` jest równa `true`, to kamera może wykonać obrót w dół bądź zmniejszyć pozycję na osi `Z`.

```
if (downArrowDown)  
{
```

```
        if (ctrlDown) view.camera.rotationX--;  
        else view.camera.z -= 10;  
    }
```

Na końcu metody `onEnterFrame()` wywołaliśmy odświeżenie wygenerowanego obrazu metodą `render()`.

Podsumowanie

- ◆ Do kontrolowania obiektów można korzystać zarówno z myszki, jak i klawiatury.
- ◆ Klasa `KeyboardEvent` reprezentuje zdarzenia związane z używaniem klawiatury. Rozpoznaje procesy wciśnięcia i zwolnienia klawisza.
- ◆ Aby wykrywać akcje klawiatury, należy dodać detektor bezpośrednio do obiektu klasy `Stage`.
- ◆ Do rozpoznawania wciśniętego klawisza służą właściwości `keyCode` i `charCode`. Pierwsza właściwość zwraca wartość numeryczną położenia klawisza na klawiaturze, a druga określa kod znaku wciśniętego klawisza.
- ◆ Przy kontrolowaniu zdarzeń użycia klawiatury przydatna jest klasa `Keyboard`, która między innymi ma zdefiniowane stałe określające kody standardowych znaków klawiszy.
- ◆ Klasa `KeyLocation` służy do określenia lokalizacji klawisza, ma stałe określające strefy standardowej klawiatury.
- ◆ W metodach wywołanych zdarzeniami klawiatury warto stosować dodatkowe zmienne o wartościach typu `Boolean`, określające status poszczególnego klawisza. Jeśli stosujemy te zmienne w metodach wywołujących animację, będzie możliwe tworzenie konkretnych efektów dla różnych kombinacji wciśniętych klawiszy.
- ◆ Zdarzenia `MouseEvent` nie działają bezpośrednio na obiekcie trójwymiarowym.
- ◆ Na materiałach interaktywnych można stosować zdarzenia klasy `MouseEvent`.
- ◆ Właściwość `interactive` w materiałach służy do określenia, czy mają być stosowane zdarzenia myszki.
- ◆ Zdarzenia `MouseEvent` wywoływane na obiekcie klasy `Stage` można wykorzystać do zmiany kątów nachylenia poszczególnych obiektów na scenie.

- ♦ Biblioteka Away3D ma własną klasę do obsługi zdarzeń myszy o nazwie `MouseEvent3D`.
- ♦ Stosując współrzędne UV, można umieszczać elementy w konkretnym miejscu tekstury typu `MovieMaterial`.
- ♦ Aby filtry typu `Blur` lub `Glow` były widoczne na elementach, trzeba zastosować właściwość `ownCanvas` na wybranym obiekcie i przypisać mu wartość `true`.

Skrowidz

A

Adobe Flash
import pliku, 144
okno Properties, 146
tworzenie odwołania, 147

Adobe Flash Builder, 35

Adobe Flash Player, 16

AGAL, Adobe Graphics Assembly Language, 566

- Attribute Registers, 568
- Constant Registers, 568
- komendy, 567
- Output Registers, 568
- Temporary Registers, 568
- Texture Samplers Registers, 569
- Varying Registers, 569

Alternativa3D, 16

animacja, 41, 181, 280, 284

- biegu, 452
- modeli, 275
 - DAE, 275
 - MD2, 275
- obrotu, 53

animowane tekstury, 216

API Stage3D, 550, 556, 590

aplikacje typu RIA, 35

AR, Augmented Reality, 541

argumenty konstruktora

- AwayStats, 509
- BitmapCubeTexture, 586

Away3D, 14

- budowa aplikacji, 50
- dźwięk, 465
- filtry, 516

- formaty plików, 232
- instalacja, 31
- integracja z Adobe Flash, 31, 35
- integracja z FlashDevelop, 33
- kamery, 387
- komponenty, 39–50
- materiały, 143–187
- modele, 223
- modele 3D, 248
- obiekty, 61–138
- światło, 191–219
- układ współrzędnych, 54
- źródła, 28

Away3D 3.6.0, 27

Away3D 4.x, 578, 585–590

B

biblioteka

- Adobe Tween, 284
- Alternativa3D, 16
- AS3Dmod, 24
- Away3D, 14
- FIVE3D, 20
- Flare3D, 21
- FLARManager, 24, 544
- FLARToolkit, 20, 22, 543
- GoASAP, 284
- GreenSock Tweening Platform, 284
- GTwen, 284
- IN2AR, 545
- JiglibFlash, 20–22
- Minko, 37
- ND3D, 37
- Papervision3D, 18

biblioteka

- Sandy3D, 23
- Sophie3D, 24
- Stardust, 22
- Tweener, 284
- TweenLite, 284
- TweenMax, 24, 284
- Tweensy, 284
- WOW, 24
- Yogurt3D, 37

biblioteki

- 3D, 14
- bezpłatne, 37
- do animacji, 284

bitmapy, 170

błąd wgrzywania tekstur, 264

C

Camera3D, 39, 46, 59, 387, 418

cieniowanie

- Phong, 220
- płaskie, 220
- z symulacją zniekształceń, 220

Context3D, 559

CPU, Central Processing Unit, 15

czas reakcji kamery, 424

czcionki, 490

czułość obrotu kamery, 433

D

detektor zdarzeń, 93, 234, 399

dodawanie

- obsługi Flash Playera, 551
- plików, 144

dokładność odwzorowania, 320, 325

dźwięk, 465

- 3D, 468
- SimplePanVolumeDriver, 467
- Sound3D, 465

E

Eclipse, 35

edytor kodu, 32

efekt

- deformacji tekstu, 504
- głębi ostrości, 405
- rybiego oka, 415

eksportowanie modelu, 241

- z Autodesk 3ds Max, 246
- z Blendera, 244
- z PreFab3D, 241

F

Face, 559

filtr, 385, 516

- FogFilter, 517
- MaxPolyFilter, 520
- ZDepthFilter, 519

FIVE3D, 20

Flare3D, 21

Flash Player 11, 548

Flex SDK, 35

focus, 391, 461

format COLLADA, 232

formaty plików, *Patrz* pliki

formaty modeli, 15, 24

fov, Field of View, 389, 461

FPP, First Person Perspective, 422

funkcja

- collisionDetected(), 535
- hit(), 535
- trace(), 56

G

generowanie mapy normalnych, 213

głębia ostrości, 402, 405, 461

głośność dźwięku, 487

GPU, Graphics Processing Unit, 15, 27

gra

Almax Race, 21
BattleCell, 422
Cafe World, 15
Half-Life, 227
Logoleptic, 23
Pepsi Music Challenge, 19
Quake 2, 233
Tanki Online, 16
Yellow Planet, 22

gry typu

FPP, 422
RPG, 451

H

HD, High Definition, 215

High poly, 231

I

implementacja

kamery FPP, 435, 441
kamery HoverCamera3D, 457
kamery SpringCam, 449
TweenMax, 285

importowanie

klas, 160
modelu, 241
plików, 145

informacje o stanie aplikacji, 506

inicjalizacja sceny, 43

instalowanie

Away3D, 31
Away3D 4.x, 578
GreenSock Tweening Platform, 284

interfejs programu

3ds Max, 224
Blender, 227
CharacterFX, 229
Google SketchUp, 230
Lightwave Modeler, 226
Maya, 225
MilkShape 3D, 228
PreFab3D, 231

J

język

ActionScript 3.0, 14, 284, 570
AGAL, 565
haXe, 32
MXML, 35

K

kamera, 45, 59, 387

Camera3D, 45, 418
focus, 391
FPP, 426
głębia ostrości, 402
HoverCamera3D, 421, 451, 460
lens, 406
perspektywa osoby trzeciej, 443
perspektywa pierwszej osoby, 425
pole widzenia, 389
SpringCam, 422, 443, 463
TargetCamera3D, 420, 462
w action RPG, 451
Xbox Kinect, 546
zoom, 390

kąt nachylenia, 432

klasa

AbstractLens, 414, 462
AbstractLight, 220
AbstractPrimitive, 94, 140
ActionScript, 233
AGALMiniAssembler, 570, 576
AmbientLight3D, 194, 220
AnimatedBitmapMaterial, 184
AnimationData, 258, 276, 281
AnimationDataType, 277, 281
AnimationLibrary, 277, 281
Animator, 278, 281
AnimatorEvent, 279
Arrow, 123
arrowhead, 96
AS3Exporter, 233, 281
AS3ExporterExample, 235
Ase, 268
AwayStats, 508

klasa

- AWData, 274
- BasicExample, 51
- bazowa, 77, 192, 286
- BitmapFileMaterial, 172, 281
- BitmapMaterial, 170, 207
- Building, 473
- Car, 444, 449
- Camera3D, 46, 418
- CameraPropertiesPanel, 392
- Cast, 149, 188
- Clipping, 512
- ClippingExample, 511
- ColorMaterial, 167, 504
- Collada, 270
- CompositeMaterial, 187
- Cone, 120
- Context3D, 559, 565
- Context3DProgramType, 564
- Context3DVertexBufferFormat, 564
- coordinateSystemExample, 58
- Cube, 109
- CubeMaterialsData, 109
- Cylinder, 118
- DepthOfFieldSprite, 402
- DirectionalLight3D, 199, 220
- DirectionalSprite, 85, 139
- DisplayObject, 352
- DofCashe, 461
- DoFExample, 403
- Dot3BitmapMaterial, 221
- Dot3BitmapMaterialF10, 214
- Dot3MovieMaterial, 216
- Elevation, 325, 338
- ElevationExample, 327
- ElevationReader, 536
- EnviroBitmapMaterial, 174
- EnviroColorMaterial, 168
- Event, 52
- Explode, 331, 338
- ExplosionExample, 332
- Face, 72, 139
- faceExample, 73
- FiltersExample, 516
- FogFilter, 517
- FPP, 426, 431
- fppExample, 441
- FrustumClipping, 514
- GeodesicSphere, 114, 140
- GlassMaterial, 175
- GraffitiExample, 364, 366
- GridPlane, 106
- HeightMapModifier, 319, 338
- HeightMapModifierExample, 322
- Hi-ReS-Stats, 509
- HoverCamera3D, 421
- IndexBuffer3D, 561
- Init, 99, 140
- Keyboard, 341
- KeyboardEvent, 340, 384
- KeyboardEventsExample, 345
- KeyLocation, 343, 384
- LensExample, 409
- LensPanel, 407
- Light3DExample, 192
- LineSegment, 100, 140
- Loader3D, 268, 529
- LODObject, 526
- LODObjectExample, 523
- MaterialsExample, 150, 160, 203
- Matrix, 311
- Matrix3D, 337
- Max3DS, 263, 529
- MaxPolyFilter, 520
- Md2, 272
- Merge, 331, 338
- Mesh, 65, 70, 139
- ModelsExample, 252, 258
- ModelsPanel, 249
- MouseEvent, 352
- MouseEvent3D, 359
- MouseEvent3DExample, 375, 369
- MouseEventExample, 354, 358
- MoveRotateScaleExample, 287
- MovieClip, 52, 499
- MovieClipSprite, 83, 139, 352
- MovieMaterial, 181
- NearfieldClipping, 514

Obj, 266
Object3D, 46, 60
ObjectContainer3D, 43, 49, 60
OrthogonalLens, 417
Path, 337
PathAlignModifier, 500, 504
PathExtrusion, 313, 337
PathExtrusionExample, 316
PerspectiveLens, 415
PerspectiveMatrix3D, 572
PhongColorMaterial, 205, 207
PhongMovieMaterial, 210
PhongMultiPassMaterial, 218, 221
PointLight3D, 196, 220
Plane, 104
Player, 435, 441
primitivesExample, 89
PrimitivesPanel, 88
Program3D, 562
Ray, 535
RectangleClipping, 513
RegularPolygon, 107, 140
RoundedCube, 111
rpgExample, 457
Scene3D, 43, 49
SeaTurtle, 122, 141
Segment, 68, 139
segmentExample, 69
ShadingColorMaterial, 206
SimplePanVolumeDriver, 487
Skeleton, 452, 457
SkinExtrude, 325, 338
Skybox, 135, 141
Skybox6, 138, 141
skyboxExample, 137
Sound, 478
Sound3D, 465, 472, 487
Sound3DExample, 470
SoundChannel, 478
SoundControlExample, 478
Sphere, 112, 140, 482
SphericalLens, 415
Sprite, 52
Sprite3D, 81, 139
spritesExample, 78
Stage3D, 27, 548, 555–558, 565
Stage3DExample, 573
StageQuality, 521
Stats, 506
TargetCamera3D, 420
Tekstura, 148
TextExtrusion, 496, 503
TextField3D, 492, 503
TextField3DExample, 493
TextureMaterial, 587
textureMC, 217
TimelineLite, 285
TimelineMax, 285
Timer, 41, 284
Torus, 115, 140
TorusKnot, 117
tppExample, 449
Triangle, 103
Trident, 57, 101, 140
Transform, 311
TransformBitmapMaterial, 178
TweenLite, 285
TweenMax, 285, 521
TweenNano, 285
URLLoader, 495
URLLoaderDataFormat, 495
URLRequest, 472
Vector3D, 70
VectorText, 491, 494
Vertex, 62, 139
VertexBuffer3D, 561
vertexExample, 62
View3D, 40
VideoMaterial, 185
Weld, 527
WeldExample, 528
WhiteShadingBitmapMaterial, 209
WireColorMaterial, 165, 504
WireCone, 128
WireCube, 126
WireCylinder, 129
WireframeMaterial, 164, 504
WirePlane, 125

- klasa
 - WireRegularPolygon, 132
 - WireSphere, 131
 - WireTorus, 133
 - ZDepthFilter, 519
 - ZoomFocusLens, 415
 - klasy w away3d.primitives, 589
 - klawiatura, 340
 - kolor, 167
 - kolor piksela, 212
 - komendy języka AGAL, 566
 - komponent
 - ComboBox, 88
 - Camera3D, 45
 - Scene3D, 21, 43
 - View3D, 40
 - komunikat o błędzie, 263
 - konfiguracja Flex SDK, 36
 - konsola
 - Nintendo Wii, 547
 - Xbox 360, 546
 - konstruktor
 - Arrowhead, 98
 - BasicExample, 52
 - CameraPropertiesExample, 399
 - DoFExample, 404
 - faceExample, 74
 - FPP, 432
 - GlassMaterial, 176
 - GraffitiExample, 364
 - LensExample, 413
 - LODObjectExample, 525
 - MaterialsExample, 161
 - ModelsExample, 261
 - PathAlignModifier, 501
 - Player, 439
 - PrimitiveExample, 92
 - rpgExample, 460
 - segmentExample, 70
 - Skeleton, 455
 - skyboxExample, 137
 - Sound3DExample, 472
 - SoundControlExample, 482
 - spritesExample, 80
 - tppExample, 451
 - TransformBitmapMaterial, 179
 - vertexExample, 63
 - WeldExample, 529
 - kontroler, 546
 - Wii Remote, 547
 - Xbox Kinect, 546
 - konwertowanie modelu, 233
 - kreator Mini, 25
- L**
- lens, 406, 461
 - limity dotyczące zasobów, 558
 - linie, 164
 - LOD, Level of Detail, 15, 522
 - Low poly, 231
 - lustro, 22
- Ł**
- ładowanie modeli 3DS, 264
- M**
- macierz transformacji, 311
 - malowanie, 361
 - malowanie graffiti, 362
 - mapa normalnych, 588
 - mapowanie normalnych, 212
 - Material, 559
 - material, 143
 - AnimatedBitmapMaterial, 183, 189, 531
 - BitmapFileMaterial, 86, 136, 172, 189
 - BitmapMaterial, 170, 189, 504
 - ColorMaterial, 167, 188, 497
 - CompositeMaterial, 189
 - Dot3BitmapMaterial, 212
 - Dot3BitmapMaterialF10, 214
 - Dot3MovieMaterial, 216
 - EnviroBitmapMaterial, 174, 189
 - EnviroColorMaterial, 168, 189
 - GlassMaterial, 175, 189
 - MovieMaterial, 181, 189, 504, 531

- PhongBitmapMaterial, 207
- PhongColorMaterial, 204
- PhongMovieMaterial, 210
- PhongMultiPassMaterial, 218
- ShadingColorMaterial, 70, 206, 413
- TransformBitmapMaterial, 178, 189, 477
- VideoMaterial, 185, 189, 531
- WhiteShadingBitmapMaterial, 209, 329
- WireColorMaterial, 165, 497
- WireframeMaterial, 164, 188, 497
- materialy
 - dla tekstu, 497
 - reagujące na światło, 202, 220
- Matrix3D, 559
- Mesh, 559
- metadane, 149
- metoda
 - addChild(), 40, 44, 59
 - addFace(), 99
 - addLight(), 191
 - addLights(), 198, 205
 - addMaterial(), 187
 - addModel(), 271
 - addObjects(), 198
 - addOnMouseMove(), 376
 - addPanel(), 93
 - addSegment(), 68
 - addSounds(), 473
 - addSprite(), 82
 - assemble(), 571
 - buildPrimitive(), 99
 - clearMaterials(), 187
 - clone(), 82
 - context3DCreated(), 577
 - createTexture(), 576
 - createVertex(), 99
 - curveTo(), 68
 - defineSingleCurve(), 68
 - destroyDeathstar(), 335
 - disableDof(), 402
 - distance(), 535
 - drawCircle(), 362, 366
 - drawTriangles(), 578
 - enableDof(), 402, 461
 - endFill(), 367
 - export(), 234, 239
 - exportModel(), 239
 - extractFont(), 492
 - getArray(), 99
 - getBitmap(), 99
 - getBoolean(), 99
 - getColor(), 99
 - getInt(), 99
 - getMaterial(), 99
 - getNumber(), 99
 - getString(), 99
 - initCamera(), 460
 - initClouds(), 590
 - initDirectionalSpriteObject(), 86
 - initEarth(), 587
 - initFilter(), 518
 - initMaterial(), 205
 - initMaterials(), 162
 - initObjects(), 162
 - initPanelListeners(), 194–197
 - initPlayer(), 460
 - initPrimitives(), 93, 110, 117, 121
 - initSprite3DObjects(), 82
 - landingDockOnMouseDown(), 379
 - lineTo(), 68
 - loadShipModel(), 350
 - lookAt(), 305, 337
 - moveBackward(), 297, 337
 - moveDown(), 299, 337
 - moveForward(), 297, 337
 - moveLeft(), 300, 337
 - movePivot(), 303
 - moveRight(), 300, 337
 - moveTo(), 296, 337
 - moveUp(), 298, 337
 - normalize(), 485
 - onEnterFrame(), 94, 180, 324, 382, 485
 - onFontLoaded(), 497
 - onGroundClick(), 378, 460
 - onKeyDown(), 350, 358
 - onKeyUp(), 358
 - onLandingDockSuccess(), 378
 - onMaterialChangeEventHandler(), 161

metoda

- onModelLoaded(), 235
- onModelLoaderSuccess(), 380
- onModelSelected(), 239
- onPanelEvent(), 177, 259
- onResize(), 137, 382
- overScene(), 376, 377
- parse(), 266
- perspectiveFieldOfViewLH(), 577
- pitch(), 306, 337
- play(), 183
- playAnimations(), 280
- primitiveChange(), 88
- primitiveChangeEventHandler(), 93, 100, 105, 113
- randomPosition(), 381
- removeMaterial(), 187
- render(), 41, 53, 424, 521
- roll(), 307, 337
- rotate(), 305
- rotateTo(), 304, 337
- scale(), 310, 337, 359
- setVisibility(), 186
- toDEGS(), 434
- toRADS(), 434
- traceLevels(), 537
- translate(), 295, 337
- update(), 456
- updateCamera(), 432, 440
- updateLight(), 198
- wrapText(), 502
- yaw(), 308, 337

metody klasy

- AbstractLens, 415
- AbstractPrimitive, 95
- AGALMiniAssembler, 571
- AmbientLight3D, 196
- AnimatedBitmapMaterial, 184
- AnimationData, 276
- AnimationLibrary, 277
- Animator, 278
- AnimatorEvent, 279
- AS3Exporter, 240
- Ase, 269

- AwayStats, 509
- AWData, 275
- BitmapFileMaterial, 173
- BitmapMaterial, 172
- Camera3D, 419
- Cast, 149
- Clipping, 513
- Collada, 271
- ColorMaterial, 168
- CompositeMaterial, 188
- Cone, 122
- Context3D, 560
- Cube, 110
- Cylinder, 120
- DirectionalLight3D, 202
- DirectionalSprite, 87
- Dot3BitmapMaterial, 214
- Dot3BitmapMaterialF10, 216
- Dot3MovieMaterial, 217
- Elevation, 331
- ElevationReader, 537
- EnviroBitmapMaterial, 175
- EnviroColorMaterial, 170
- Explode, 335
- Face, 76
- FogFilter, 519
- GeodesicSphere, 115
- GlassMaterial, 178
- GridPlane, 107
- HeightMapModifier, 325
- HoverCamera3D, 422
- IndexBuffer3D, 561
- Keyboard, 342
- KeyboardEvent, 341
- LineSegment, 101
- LODObject, 527
- Matrix3D, 312
- Max3DS, 265
- Md2, 273
- Mesh, 66
- MouseEvent3D, 360
- MovieClipSprite, 85
- MovieMaterial, 182
- Obj, 267

Object3D, 48
 ObjectContainer3D, 50
 PathAlignModifier, 503
 PathExtrusion, 318
 PhongBitmapMaterial, 209
 PhongColorMaterial, 206
 PhongMovieMaterial, 211
 PhongMultiPassMaterial, 220
 Plane, 105
 Ray, 536
 RegularPolygon, 109
 RoundedCube, 112
 Scene3D, 45
 SeaTurtle, 123
 Segment, 71
 ShadingColorMaterial, 207
 SimplePanVolumeDriver, 468
 Skybox, 138
 Skybox6, 139
 Sound3D, 467
 Sphere, 113
 SpringCam, 425
 Sprite3D, 83
 TargetCamera3D, 420
 TextField3D, 496
 Torus, 116
 TorusKnot, 118
 TransformBitmapMaterial, 180
 Triangle, 104
 Trident, 102
 Vertex, 65
 VertexBuffer3D, 562
 View3D, 42
 Weld, 530
 WhiteShadingBitmapMaterial, 210
 WireColorMaterial, 167
 WireCube, 128
 WireCylinder, 130
 WireframeMaterial, 165
 WirePlane, 126
 WireRegularPolygon, 133
 WireSphere, 132
 WireTorus, 134

metody obsługi zdarzeń MouseEvent3D, 361
 mieszanie materiałów, 187
 mnożenie macierzy, 572
 model, 223

- High poly, 231
- Low poly, 231
- typu MD2, 454

 modele

- cieniowania, 587
- formaty, 232, 281
- statyczne, 232

 modyfikator private, 52
 modyfikowanie powierzchni, 313
 Molehill, 16
 możliwości

- Alternativa3D, 17
- Away3D, 14
- FIVE3D, 20
- Flare3D, 22
- Papervision3D, 19
- Sandy3D, 23
- Sophie3D, 24

 możliwości kamery, 425
 MVC, Model View Controller, 41
 myszka, 352

N

narzędzie Weld, 527
 normalna, 212

O

obiekt klasy

- AbstractPrimitive, 94
- Arrow, 123
- Cone, 120
- Cube, 109
- Cylinder, 118
- DirectionalSprite, 85
- Face, 72
- GeodesicSphere, 114
- GlassMaterial, 176
- GridPlane, 77, 106

- obiekt klasy
 - LineSegment, 100
 - Mesh, 65
 - Metody klasy Arrow, 125
 - Metody klasy WireCone, 129
 - MovieClipSprite, 83
 - Plane, 104
 - RegularPolygon, 107
 - RoundedCube, 111
 - SeaTurtle, 122
 - Segment, 68
 - Skybox, 135
 - Skybox6, 138
 - Sphere, 112
 - Sprite3D, 81
 - Torus, 115
 - TorusKnot, 116
 - Triangle, 102, 103
 - Trident, 101
 - Vertex, 62
 - WireCone, 128
 - WireCube, 126
 - WireCylinder, 129
 - WirePlane, 125
 - WireRegularPolygon, 132
 - WireSphere, 130
 - WireTorus, 133
- obiekty, 61, 522
 - animations, 280
 - BitmapData, 216, 319, 531
 - DisplayObject, 18
 - Graphics, 20
 - LayerMaterial, 187
 - LOD, 15, 18, 522
 - MovieClip, 20
 - Path, 313, 317
 - Sound3D, 469
 - speedMonitor, 349
 - Sprite, 221
 - TextField3D, 500
 - Vector3D, 53, 313
 - VertexBuffer3D, 575
 - w Base, 62
 - w Primitives, 87
 - w Sprites, 76
- obiektyw, 415
 - AbstractLens, 414
 - OrthogonalLens, 417
 - PerspectiveLens, 415
 - SphericalLens, 415
 - ZoomFocusLens, 415
- Object3D, 559
- obracanie, 301
 - kamery wokół obiektu, 590
 - metody, 303
 - właściwości, 302
- obrazy wysokiej rozdzielczości, 215
- obsługa
 - Flash Playera, 551, 553
 - tekstur, 455
 - zdarzeń klawiatury, 340
 - zdarzeń MouseEvent3D, 361
 - zdarzeń myszy, 352, 385
- odbicie Fresnela, 587
- odwołania, 188
- odwoływanie się do zasobów, 147
- ograniczanie widoczności, 522
- określanie odległości, 534
- opcja
 - Away3D Project stats, 506
 - Flash WMODE Direct, 550
- osadzanie
 - aplikacji Flash, 550
 - czcionki, 491
 - zasobów, 149
- oświetlenie typu DirectionalLight3D, 413

P

- pakiet
 - away3d.animators, 278
 - away3d.cameras, 431
 - away3d.cameras.lenses, 439
 - away3d.cameras.lenses, 414
 - away3d.containers, 40, 522
 - away3d.core.base, 62
 - away3d.core.clip, 510
 - away3d.core.stats, 506
 - away3d.core.utils, 99

- away3d.debug, 508
- away3d.events, 279, 359
- away3d.exporters, 233
- away3d.extrusions, 313
- away3d.loaders, 238
- away3d.loaders.utils, 277
- away3d.materials, 143, 188, 281
- away3d.materials.methods, 587
- away3d.modifiers, 319
- away3d.primitives, 88, 134, 140, 589
- away3d.primitives.data, 109
- away3d.sprites, 77
- away3d.textures, 587
- away3d.tools, 527
- com.greensock.easing, 398
- flash.events, 356
- flash.display3D, 564
- flash.display3D.textures, 575
- flash.events, 340
- panel Hi-ReS-Stats, 510
- panel użytkownika, 89, 197, 201, 262
- panoramy, 134
- Papervision3D, 18
- parametry klasy
 - AGALMiniAssembler, 571
 - Context3D, 560
 - ElevationReader, 537
- perspektywa
 - osoby pierwszej, 422, 425
 - osoby trzeciej, 422, 443
- pilot Wii, 547
- platforma
 - Adobe Flash Player, 13
 - Eclipse, 35
- plik
 - AS3ExporterExample.as, 235
 - AS3GeomExporter.ms, 247
 - Building.as, 474
 - CameraPropertiesPanel.as, 392
 - LensExample.as, 409
 - LensPanel.as, 406
 - MaterialsExample.as, 153, 202
 - MaterialsPanel.as, 150, 153
 - ModelsExample.as, 252
 - ModelsPanel.as, 249, 252
 - monster.swf, 404
 - playerglobal.swc, 552
 - PrimitivesExample.as, 89
 - PrimitivesPanel.as, 88
 - spaceship.3DS, 529
- pliki
 - 3DS, 232, 263, 374
 - AGALMiniAssembler, 570
 - AS, 233
 - ASE, 232, 268
 - AWD, 15, 233, 273, 439
 - COLLADA, 232
 - DAE, 232, 239
 - FLV, 185
 - GIF, 183
 - graficzne, 188
 - KML, 229
 - KMZ, 229
 - MD2, 233, 271
 - ms, 247
 - OBJ, 15, 232, 266, 447
 - PSD, 213
 - swf, 149, 491
 - TGA, 455
 - XML, 232
- plugin AS3GeomExporter, 247
- płaszczyzna plane, 186
- pobieranie
 - Away3D, 28
 - modelu, 281
 - tekstury, 281
- pokrywanie teksturą, 326
- pole tekstowe, 493
- pole widzenia, 389, 461
- Polygon, 559
- położenie
 - kamery, 432
 - w przestrzeni, 53
- poziom szczegółowości, 522
- poziomy obiektów, 523
- pozycja kamery, 463
- prędkość przemieszczania, 432

procesor, 15
 CPU, 556
 GPU, 556
 profil ścieżki, 314
 program
 3D Studio Max, 224
 Adobe Flash, 26, 144
 Adobe Flash Player, 27
 Adobe Flash Professional CS5, 31
 Adobe Photoshop, 221
 Autodesk 3ds Max, 224, 246, 232
 Autodesk Maya, 224
 Blender, 226, 244
 CharacterFX, 228, 281
 Flash Builder, 26
 FlashDevelop, 26, 32
 Google Earth, 229
 Google SketchUp, 229
 LightWave, 225
 MilkShape 3D, 227
 PreFab3D, 230, 241
 TortoiseSVN, 28
 Program3D, 562
 przemieszczanie, 294
 metody, 295
 postaci, 454
 właściwości, 294
 przycinanie widoku, 510
 przycisk togglePause, 186
 publikowanie projektów, 549
 punkt
 obserwacji, 462
 początkowy pozycji kamery, 462

R

reakcje kamery, 463
 regulacja Depth of Field, 462
 rejestrator zdarzeń, 53
 rejestrowanie
 obiektów, 439
 sceny, 388
 rejestry w języku AGAL, 567
 repozytorium SVN, 28
 RGB, 212

rodzaje
 akceleracji, 549
 kamer, 418
 modeli, 223, 283
 rejestrów, 567
 rzutowania, 388
 silników Away3D, 26
 soczewek, 406
 światel, 192
 AmbientLight3D, 220
 DirectionalLight3D, 220
 PointLight3D, 220
 rozmiar tekstury, 178
 rozmieszczanie obiektów, 367
 rysowanie, 361, 367
 rzeczywistość
 rozszerzona, 541
 wirtualna, 541
 rzutnia, 388, 461
 rzutowanie, 388

S

Sandy3D, 23
 Scene3D, 40, 43, 59
 dodawanie obiektów, 44
 usuwanie obiektów, 44
 schemat
 mapowania wypukłości, 212
 wyświetlania obrazu, 389
 Shader, 562
 Fragment Shader, 563
 Vertex Shader, 563
 silnik
 AwayPhysics, 539
 Box2DFlash, 539
 Bullet Physics Engine, 540
 JigLibFlash, 538
 ND2D, 558
 Starling, 558
 silniki fizyki, 538
 skalowanie, 309
 metody, 310
 właściwości, 309

skrypt AS3GeomExporter, 246
 soczewki, 406, 462
 Sophie3D, 24
 Sophie3D PRO, 24
 Sophie3D PRO FULL, 24
 Sprite3D, 559
 Stage3D, 548, 555, 563
 Face, 559
 Material, 559
 Matrix3D, 559
 Mesh, 559
 Object3D, 559
 Polygon, 559
 Sprite3D, 559
 Texture, 559
 Vector3D, 559
 Vertex, 559
 Stage3D API, 15
 stałe klasy
 AbstractLens, 415
 AnimationDataType, 277
 AnimationEvent, 279
 Camera3D, 419
 Keyboard, 342
 KeyLocation, 343
 MouseEvent3D, 360
 statystyki, 507
 sterowanie
 dźwiękiem, 487
 statkiem, 344
 SVN, 28
 symulowanie fizyki, 533, 538

Ś

ścieżka, 313
 środowisko
 Adobe Flash Builder, 35
 Flex SDK, 33, 149, 552
 światło, 191
 AmbientLight3D, 194
 DirectionalLight3D, 199
 PointLight3D, 196

T

tablica
 pathVectors, 501
 vertices, 141
 technologia
 Adobe Pixel Bender, 214
 Flash, 14, 533
 Kinect, 546
 tekst, 489
 jako BitmapData, 498
 modyfikowany, 499
 płaski, 492
 przestrzenny, 496
 w obiekcie MovieClip, 499
 tekstura
 bazowa, 216
 cieni, 588
 mapy odbijania światła, 588
 skały, 216
 Specular, 218
 Texture, 559
 tła, 134
 tor, 314
 TPP, Third Person Perspective, 422
 transformacja
 obróć, 311
 pochylenie, 311
 skalowanie, 311
 translacja, 311
 tryb wolny kamery, 435
 twierdzenie Pitagorasa, 456
 tworzenie
 animacji, 189, 281
 aplikacji dla przeglądarek, 14
 kontenerów, 49
 macierzy transformacji, 312
 obiektów 3D, 224, 281
 projektu ActionScript 3.0, 31
 projektu Flex, 36
 rzeczywistości rozszerzonej, 542
 widoku, 41

U

układ współrzędnych
 globalny, 55
 lokalny, 55
umieszczanie obiektów na scenie, 44
ustawienia swfobject, 551
usuwanie obiektów ze sceny, 44

V

Vector3D, 559
Vertex, 559
View3D, 40, 59

W

warstwy Stage3D, 557
wartość FPS, 41, 509
wektor
 kierunkowy, 535
 normalny, 212
 przesunięcia, 432
widok, 40, 59
Wii Remote, 547
wirtualna klawiatura, 20
wizualizacja
 dźwięku, 478
 kuli ziemskiej, 580
 obiektu ObjectContainer3D, 49
właściwości klasy
 AbstractPrimitive, 95
 AmbientLight3D, 195
 AnimatedBitmapMaterial, 184
 AnimationData, 276
 AnimationDataType, 277
 AnimationEvent, 279
 Animator, 278
 Arrow, 124
 AS3Exporter, 240
 Ase, 269
 AWData, 275
 BitmapFileMaterial, 173
 BitmapMaterial, 171

Camera3D, 419
Clipping, 512
Collada, 271
ColorMaterial, 168
CompositeMaterial, 188
Cone, 121
Cube, 110
Cylinder, 119
DirectionalLight3D, 202
DirectionalSprite, 87
Dot3BitmapMaterial, 214
Dot3BitmapMaterialF10, 216
Dot3MovieMaterial, 217
Elevation, 331
EnviroBitmapMaterial, 175
EnviroColorMaterial, 169
Explode, 335
Face, 76
FogFilter, 519
GeodesicSphere, 115
GlassMaterial, 178
GridPlane, 107
HeightMapModifier, 324
HoverCamera3D, 422
Keyboard, 342
KeyboardEvent, 341
LineSegment, 101
LODObject, 526
Matrix3D, 312
Max3DS, 265
Md2, 273
Merge, 336
Mesh, 66
MouseEvent3D, 360
MovieClipSprite, 84
MovieMaterial, 182
Obj, 267
Object3D, 46
ObjectContainer3D, 49
PathAlignModifier, 502
PathExtrusion, 318
PhongBitmapMaterial, 209
PhongColorMaterial, 205
PhongMovieMaterial, 211

- PhongMultiPassMaterial, 219
 - Plane, 105
 - PointLight3D, 199
 - Ray, 536
 - RegularPolygon, 108
 - RoundedCube, 112
 - Scene3D, 44
 - Segment, 71
 - ShadingColorMaterial, 207
 - SimplePanVolumeDriver, 468
 - Sound3D, 467
 - Sphere, 113
 - SpringCam, 424
 - Sprite3D, 83
 - TargetCamera3D, 420
 - TextField3D, 496
 - Torus, 116
 - TorusKnot, 118
 - TransformBitmapMaterial, 180
 - Triangle, 104
 - Vertex, 65
 - VideoMaterial, 186
 - View3D, 41
 - Weld, 530
 - WhiteShadingBitmapMaterial, 210
 - WireColorMaterial, 166
 - WireCone, 129
 - WireCube, 127
 - WireCylinder, 130
 - WireframeMaterial, 165
 - WirePlane, 126
 - WireRegularPolygon, 133
 - WireSphere, 131
 - WireTorus, 134
 - właściwości pola tekstowego, 490
 - właściwość
 - camera, 46, 59, 418
 - focus, 390, 392
 - fov, 389, 390
 - freeCamera, 443
 - interactive, 384
 - maxblur, 402
 - mimeType., 150
 - ownCanvas, 379, 521
 - pivotPoint, 302, 337
 - position, 53, 295, 337
 - rotationX, rotationY, rotationZ, 303
 - scaleX, scaleY, scaleZ, 309
 - scene, 43, 59
 - smooth, 531
 - statsPanel, 506
 - subdivisionX, 325
 - subdivisionY, 325
 - view, 424
 - visible, 521
 - volume, 466
 - x, y, z, 294
 - zoom, 390, 391
 - współrzędne obiektów, 56, 59
 - wtyczka Adobe Flash Player, 27, 553
 - wydajność aplikacji, 506
 - wykorzystanie kart graficznych, 556
 - wykrywanie
 - kolizji, 534
 - nierówności podłoża, 536
 - wysoka rozdzielczość obrazu, 215
 - wyświetlanie
 - modelu, 281
 - obiektów, 542, 562
 - statystyk, 509
 - tekstu, 489, 492
 - zawartości sceny, 510
- X
- Xbox Kinect, 546
- Z
- zdarzenia typu modelSelectEvent, 259
 - zdarzenie
 - Event.ADDED_TO_STAGE, 565
 - Event.ENTER_FRAME, 41, 93, 284, 361, 461
 - Event.RESIZE, 93, 357
 - Event.SELECT, 259
 - Event.SOUND_COMPLETE, 469
 - ExporterEvent.COMPLETE, 234

zdarzenie

focusAnimationEvent, 400

KeyboardEvent, 340

KeyboardEvent.KEY_, 401

KeyboardEvent.KEY_DOWN, 357

KeyboardEvent.KEY_UP, 357

lessSegmentsEvent, 180

Loader3DEvent, 238

MouseEvent, 189, 352

MouseEvent.MOUSE_DOWN, 357

MouseEvent.MOUSE_UP, 357

MouseEvent3D.MOUSE_DOWN, 359

MouseEvent3D.MOUSE_MOVE, 359

MouseEvent3D.MOUSE_UP, 359

znacznik [Embed], 149, 160

zniekształcenia obrazu, 415

zoom, 390, 461

cyfrowy, 390

optyczny, 390

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄZKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

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

GRUPA WYDAWNICZA

 **Helion SA**

WKRO CZ Z FLASHEM W ŚWIAT INTERAKTYWNYCH APLIKACJI 3D!

Technologia Flash na dobre zagościła w świecie interaktywnych animacji i gier komputerowych 2D. Stała się również (obok HTML5) standardem prezentacji tego rodzaju treści na stronach WWW. Taki stan rzeczy to efekt połączenia samego Flasha z Actionscriptem, językiem programowania ułatwiającym sterowanie pracą aplikacji i reagowanie na działania podejmowane przez użytkownika. Jednak użytkownicy internetu oczekują obecnie czegoś więcej niż tylko zwykłe animacje i gry 2D – oczekują interaktywnych treści 3D, ponieważ możliwości odtwarzania tego typu materiałów są coraz większe. Do tej pory brakowało narzędzi do tworzenia takich aplikacji, a także technologii, które je wspomagają. Na szczęście z odsieczą przychodzi Flash, który śmiało wkracza w świat 3D, oferuje ciekawe biblioteki i nie ma przy tym praktycznie żadnej konkurencji.

Jeśli znasz już trochę Flasha, a teraz chcesz poszerzyć swoją wiedzę o znajomość szybko rozwijającej się technologii 3D, sięgnij po książkę *Flash i Actionscript. Aplikacje 3D od podstaw*. Prosty, lecz precyzyjnym językiem przedstawiono w niej najważniejsze kwestie związane z projektowaniem i tworzeniem aplikacji 3D z wykorzystaniem technologii Flash i języka Actionscript, zaprezentowano sposoby używania bezpłatnej biblioteki Away3D oraz pokazano, jak za jej pomocą budować i optymalizować własne rozwiązania.

- Wybór, pobieranie i instalacja właściwej biblioteki 3D dla technologii Flash
- Podstawowe komponenty biblioteki Away3D i budowanie aplikacji za jej pomocą
- Używanie materiałów i korzystanie z różnych rodzajów światła
- Tworzenie, importowanie i stosowanie modeli oraz animacji 3D w aplikacjach Flash
- Zapewnianie interakcji aplikacji z użytkownikiem i obsługa urządzeń wejściowych
- Definiowanie kamer z perspektywy pierwszej i trzeciej osoby oraz ich używanie
- Obsługa dźwięku i elementów tekstowych

**Interaktywne aplikacje 3D to przyszłość.
Zacznij ją tworzyć już dziś!**

helion.pl
księgarnia
internetowa

Nr katalogowy: 7864



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



Zamówienia telefoniczne:
0 801 339900



0 601 339900



Helion

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

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

sięgnij po WIĘCEJ



KOD KORZYŚCI

ISBN 978-83-246-3065-3



Cena: 99,00 zł

Informatyka w najlepszym wydaniu