Polsko-Japońska Wyższa Szkoła
Technik Komputerowych

Katedra Metod Programowania

Michał Tomaszewski

***

Tworzenie światów rzeczywistości wirtualnej przy wykorzystaniu pakietu Java3D

Praca magisterska
napisana pod kierunkiem
prof. Jana Bieleckiego

Warszawa 2001

Wstęp

W 1984 roku wyprodukowano pierwszy moduł graficzny dla komputera osobistego, przystosowany do wyświetlenia w rozdzielczości 640x350 równocześnie 4 kolorów z 16-kolorowej palety. To odległe, zarówno technologicznie jaki i czasowo osiągnięcie, było milowym krokiem w rozwoju grafiki komputerowej. Już pięć lat później dostępne były urządzenia operujące na 256 kolorach. Jednak oferowana skala barw w relatywnie krótkim czasie przestała być wyznacznikiem jakości kart, jej miejsce zajęła szybkość wyświetlania grafiki. W trakcie kolejnej dekady prace konstrukcyjne nakierowane były przede wszystkim na zwiększanie szybkości działania układów, osiągając zamierzony cel głównie przez zwiększanie prędkości wyświetlania kształtów dwuwymiarowych. Kolejnym logicznym krokiem była akceleracja obiektów 3D i na tym właśnie polu do dziś dnia trwa zażarty bój.

Obecnie większość kart graficznych wykorzystuje do akceleracji jeden z trzech standardów. Utworzony przez **Silicon Graphics** *OpenGL*, konkurencyjny *Direct3D* stworzony prze **Microsoft** oraz *QuickDraw3D* opracowany przez firmę **Apple**.

Twórca Javy3D, firma **SUN Microsystems**, starała się zawrzeć w swoim produkcie najlepsze cechy każdego z wymienionych standardów. Niestety producent udostępnia bardzo skąpe i nie zawsze rzetelne informacje o sposobie użycia tego pakietu. Dlatego też celem niniejszej pracy było przedstawienie mechanizmów jakie można wykorzystać podczas tworzenia światów wirtualnej rzeczywistości za pomocą Javy3D, w formie przystępnej dla osoby zaznajomionej z programowaniem w języku Java

Java 3D jest pakietem klas umożliwiającym implementacje i wizualizacje trójwymiarowych scen. Jednak w odróżnieniu od takich produktów jak 3D Studio MAX, Maya czy Lightwave jego przeznaczeniem nie jest tworzenie filmów, lecz interakcyjna współpraca z użytkownikiem. Wykorzystując format XGL jako standard opisu obiektów występujących w kreowanym przez siebie świecie, twórca może zmieniać położenie obiektów oraz oglądać je z każdej strony. Jednak potencjalne możliwości Javy 3D są znacznie większe niż jej poprzedników, do których zaliczyć można między innymi skryptowy język opisu rzeczywistości wirtualnej VRML. Dzieje się tak dzięki temu, że użytkownik dysponuje całą potęgą języka programowania wysokiego poziomu, jakim jest Java a nie tylko ograniczoną liczbą predefiniowanych brył i węzłów. Ponadto należy zwrócić uwagę na fakt, iż obsługa jakiegokolwiek urządzenia zewnętrznego np. rękawicy 3D, także będzie znacznie prostsza do zaimplementowania w języku niezależnym od platformy.

W tej pracy przedstawiono przykłady tworzenia i wykorzystania najważniejszych elementów pakietu Java3D. Rozpoczęto od najprostszych przykładów wykorzystujących jedynie predefiniowane bryły dostarczane z pakietem. Następnie przedstawiono bardziej skomplikowane sceny składające się z wielu komponentów takich jak sterowanie, oświetlenie czy zachowanie w przypadku kolizji. Wszystkie opisane elementy zebrano w jednym programie przedstawiającym możliwości i ograniczenia nałożone na tę metodę opisu rzeczywistości wirtualnej.

Pierwszy rozdział poświęcono takim krytycznym komponentom każdego programu, jak `SimlpeUnivers`, `BranchGroup` czy `TransformGroup`. Ponadto opisano ideowe schematy, które były wykorzystywane w kolejnych rozdziałach niniejszej pracy.

W drugim rozdziale opisano rodzaje świateł jakie mogą zostać wykorzystane w scenach. Zdefiniowanie świateł na tak wczesnym etapie było podyktowane koniecznością opisania brył predefiniowanych. Elementy te nie są widoczne w scenach, które nie zawierają źródeł światła.

Trzeci rozdział poświęcono charakterystyce brył predefiniowanych. Opisano i pokazano kulę, sześcian, stożek oraz walec.

Czwarty rozdział poświęcono procesowi tworzenia brył. Opisano tu zarówno tworzenie kształtu jak i powierzchni. Scharakteryzowano również podstawowe oraz bardziej zaawansowane elementy strukturujące jakimi dysponuje Java3D. Ponadto dostarczono kompletne informacje o nakładaniu tekstur, kolorowaniu węzłów i innych możliwościach składowych brył.

W piątym rozdziale poruszono zagadnienie interakcji programu z użytkownikiem. Przedstawiono implementację obsługi klawiatury i myszki.

Szósty rozdział poświęcono animacji obiektów (płynnym przekształceniom afinicznym i metamorficznym).

Siódmy rozdział poświęcono dźwiękowi. Opisano tam dźwięki tła, ukierunkowane oraz punktowe.

Tworzenie aplikacji wykorzystujących pakiet Java3D

Aplikacje wykorzystujące pakiet Java3D są zbudowane z co najmniej dwóch modułów. Pierwszym z nich jest okno aplikacji lub apletu. Do jego utworzenia mogą zostać wykorzystane klasy `Frame`, `JFrame` lub `MainFrame`. Drugim modułem jest scena, którą należy „skompilować", a następnie umieścić w oknie. Powyższy podział można zaobserwować na przykładzie *Prog. 1*, który przedstawia standardowe okno (kod oznaczony kolorem zielonym) i umieszczoną w nim pojedynczą scenę (kod oznaczony kolorem niebieskim).

```
import java.awt.*;
import java.awt.event.*;

import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;

import javax.media.j3d.*;
import javax.vecmath.*;

public class J3DAppl extends Frame{

    public J3DAppl(){
        Canvas3D c3d = new Canvas3D(null);
            SimpleUniverse su = new SimpleUniverse(c3d);
                BranchGroup objRoot = new BranchGroup();
                    TransformGroup objScale = new TransformGroup();

                    objScale.addChild(
                        new ColorCube()
                    );

                objRoot.addChild(objScale);
                objRoot.compile();
            su.addBranchGraph(objRoot);
            su.getViewingPlatform().setNominalViewingTransform();

        this.setSize( 400, 400);
        this.add(c3d, "Center");
        this.addWindowListener(
            new WindowAdapter() {
                public void windowClosing(WindowEvent evt)
                {
                    System.exit(0);
                }
            }
        );
        this.setVisible(true);
    }
```

```
    public static void main(String[] args){
        new J3DAppl();
    }
}
```

Prog. 1

W dalszych rozdziałach zostaną przedstawione kolejno zagadnienia związane z strukturą sceny, predefiniowanymi obiektami sceny, światłem, interakcją użytkownika z obiektami w scenie, oraz obiektami animującymi inne obiekty.

1. Struktura sceny

Pakiet Java3D, podobnie do wcześniej istniejących języków o podobnej funkcjonalności, np. VRML, wykorzystuje strukturę drzewa dla opisu obiektów występujących w scenie oraz relacji zachodzących pomiędzy nimi. Najniżej w hierarchii drzewa stoją liście. Liśćmi mogą być takie obiekty jak sześcian, kula czy stożek. Każdy z nich ma właściwości np.: rozmiar czy kolor, stanowiące wewnętrzną strukturę liścia.

Obiekty grupujące, które wiążą inne obiekty mogą zostać porównane do gałęzi i konarów. Ich zadaniem jest połączenie lub nadanie pewnych dodatkowych właściwości strukturom znajdującym się w hierarchii niżej od nich. Ostatnim komponentem drzewa jest pień, rolę tę spełniają obiekty określające położenie grup oraz rodzaj środowiska.

Aby ułatwić interpretację przedstawionych powyżej komponentów, w tabeli *Tab. 1* przestawiono graficzną reprezentację tych obiektów.

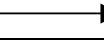
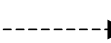
	Opis świata
	Określenie położenia
	Obiekty grupujące
	Liść
	Właściwości liści
	Inne obiekty

Tab. 1 Rodzaje obiektów – reprezentacja graficzna

Pomiędzy obiektami mogą istnieć dwa rodzaje relacji. Za ich pomocą w strukturę drzewa są wpisywane zależności pomiędzy obiektami należącymi do różnych poziomów.

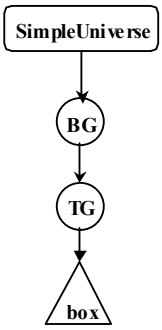
Pierwszą z nich jest relacja „rodzic-dziecko”. Jest ona tworzona pomiędzy dowolnym obiektem liścia, a dowolnym węzłem znajdującym się wyżej w hierarchii. W tej relacji parametry obiektu potomnego nie są określane względem początku układu współrzędnych sceny, lecz względem położenia obiektu-rodzica.

Drugą relacją jest „referencja", która określa parametry obiektów np. liści. Graficzna reprezentacja tych elementów została przedstawiona w *Tab. 2*

	„rodzic – dziecko”
	„referencja”

Tab. 2 Referencje – reprezentacja graficzna

Dysponując kompletnymi informacjami o reprezentacjach graficznych można dokonać graficznej analizy kodu programu *Prog. 1* co zostało przedstawione na ilustracji *Ilus. 1*.



Ilus. 1

Graficzna reprezentacja struktury programu jest również prosta. Niestety efekt, jaki uzyskano również nie jest imponujący: cała przestrzeń okna została zamalowana na kolor czerwony. Powodem tego jest odległość pomiędzy obiektem a użytkownikiem. Aby zobaczyć cały obiekt należy zmniejszyć obiekt lub oddalić od niego.

1.1 Węzeł grupujący „TransformGroup"

Najczęściej wykorzystywanym obiektem grupującym jest węzeł `TransformGroup`. Spełnia on dwie funkcje: grupującą - łączącą pewną ilość elementów i pozycjonującą – pozwalającą dokonywać przesunięć, obrotów lub przeskalowań na zgrupowanych obiektach. Opisane funkcje są realizowane przez dwie metody:

- `setTransform(Transform3D)` – odpowiadającą za przypisanie grupie zbioru parametrów, które umożliwiają orientację pod-obiektów tej grupy w scenie;
- `addChild(Node)` – pozwala na dodanie podgrup, lub brył zarówno predefiniowanych jak i nowo utworzonych.

Wykorzystanie powyższych metod pokazano w *Kod. 1*. Jako parametry metod zastosowano domyślne konstruktory obiektów.

```
TransformGroup transformgroup = new TransformGroup();
    transformgroup.setTransform(new Transform3D());
    transformgroup.addChild(new ColorCube());
```

Kod. 1

1.2 Przekształcenia afiniczne na grupach

Przekształcenie afiniczne jest to takie odwzorowanie jednego zbioru punktów na drugi, że funkcja odwzorowująca jest funkcją liniową z przesunięciem ($y = ax+b$).

Transform3D jest klasą, której obiekty są wykorzystywane przez `TransformGroup` do uzyskania informacji o przesunięciu, obrocie lub przeskalowaniu, które ma wykonać na podległych sobie obiektach.

Aby sfabrykować obiekt klasy `Transform3D` należy użyć jednego spośród trzynastu konstruktorów. Najczęściej używany jest konstruktor `Transform3D()`, tak jak pokazano to w *Kod. 2*.

```
Transform3D transform3d = new Transform3D();
    transform3d.setScale(0.1);
    transform3d.setTranslation(new Vector3d());
    transform3d.setRotation(new AxisAngle4d());
```

Kod. 2

Metoda `setScale(double)` odpowiada za wartość współczynnika przeskalowania, jakiemu zostanie poddana grupa.

Metoda `setTranslation(Vector3D)` jest odpowiedzialna za przesunięcie grupy względem środka układu współrzędnych. Ponieważ przesunięcie lub kierunek światła można opisać jako wektor, stworzono klasę `Vector3D`. Konstruktor tej klasy potrzebuje współrzędnych punktu końcowego wektora, zaś jako punkt początkowy przyjmowany jest punkt (0, 0, 0).

```
SimpleUniverse su = new SimpleUniverse(c3d);
    BranchGroup objRoot = new BranchGroup();
        TransformGroup objScale = new TransformGroup();
            Transform3D t3d = new Transform3D();
                t3d.setTranslation(
                    new Vector3d(0.0, 0.0, -5.0)
                );
                // lub
                t3d.setScale(0.1);
        objScale.setTransform(t3d);
        objScale.addChild(
            new ColorCube()
        );
     objRoot.addChild(objScale);
     objRoot.compile();
su.addBranchGraph(objRoot);
su.getViewingPlatform().setNominalViewingTransform();
```

Prog. 2

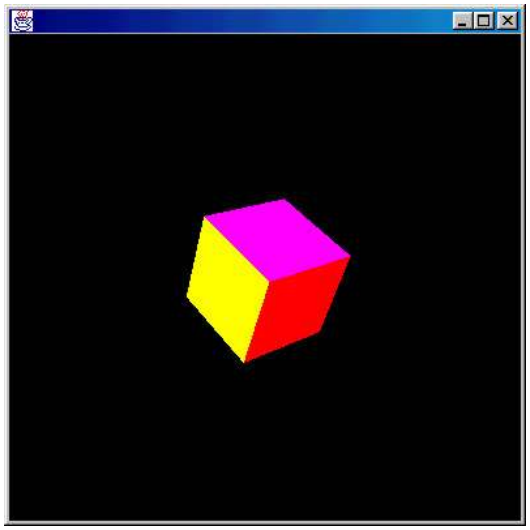
Użycie obu wymienionych metod spowodowało oddalenie obiektu oraz jego zmniejszenie. Użytkownik może teraz zobaczyć w całości jedną ze ścian obiektu.

W celu zobaczenia więcej niż jednej ściany, należy użyć metody `setRotation(AxisAngle4d)`. W konstruktorze parametru `AxisAngle4d` podawane są cztery parametry, z których pierwsze trzy opisują osie obrotu X, Y, Z; zaś czwarta kąt, o jaki należy dokonać obrotu. Implementacje opisanych metod i obiektów przedstawiono w poniższym *Prog. 3*. Efekt działania został pokazany na *Rys. 1*.

```
SimpleUniverse su = new SimpleUniverse(c3d);
    BranchGroup objRoot = new BranchGroup();
        TransformGroup objScale = new TransformGroup();
            Transform3D t3d = new Transform3D();
```

```
            t3d.setScale(0.2);

            t3d.setRotation(
                new AxisAngle4d( 1, 1, 0, 45)
            );
        objScale.setTransform(t3d);
        objScale.addChild(
            new ColorCube()
        );

    objRoot.addChild(objScale);
    objRoot.compile();
su.addBranchGraph(objRoot);
su.getViewingPlatform().setNominalViewingTransform();
```

Prog. 3



Rys. 1

2. Światło

W pakiecie Java3D zdefiniowano cztery rodzaje światła:

- wszechobecne (*Ambient*)
- ukierunkowane (*Directional*)
- punktowe (*Point*)
- reflektorowe (*Spot*)

Klasy implementujące powyższe źródła zawierają pola opisujące następujące ich właściwości:

- pole typu `boolean`, które określa czy światło jest włączone czy nie, implementuje metoda `setEnabled(bool)`
- pole kolor typu `Color3f`, określające kolor emitowanego światła jest implementowane przez metodę `setColor(Color3f)`
- zasięg, w którym obiekty będą pod wpływem tego światła został zaimplementowany przez `setInfluenceBounds(Bounding)`

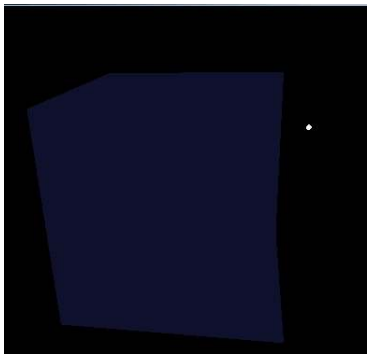
2.1 AmbientLight

Światło typu `Ambient` pozornie dobiega ze wszystkich kierunków równomiernie – nie jest możliwe zlokalizowanie jego źródła. Fabrykowanie tego obiektu przy wykorzystaniu pustego konstruktora zostało pokazane w *Kod. 3*. W przykładzie tym wykorzystano również opisane powyżej metody `setEnabled(bool)` oraz `setColor(Color3f)`

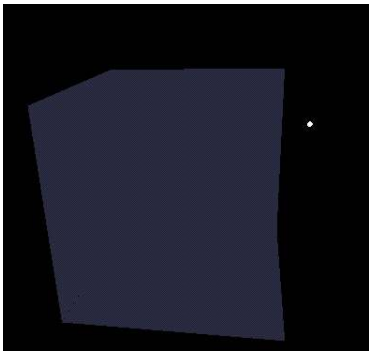
```
AmbientLight light = new AmbientLight();
    light.setEnable(
        true
    );
    light.setColor(
        new Color3f( 1.0f, 1.0f, 1.0f )
    );
```

Kod. 3

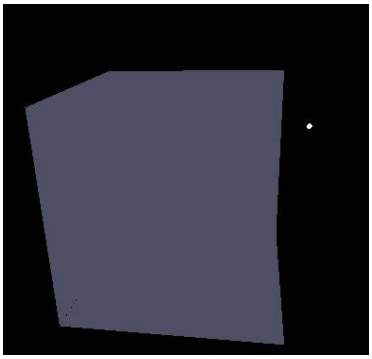
Uzyskany efekt pokazują *Rys. 2*, *Rys. 3*, *Rys. 4* na których przedstawiono światło `Ambient` w nasyceniu odpowiednio 0.2, 0.5 i 1.0.



Rys. 2



Rys. 3



Rys. 4

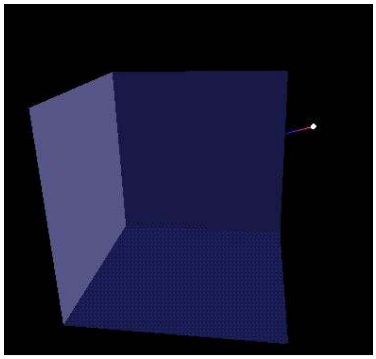
2.2 DirectionalLight

Obiektem reprezentującym światło o mniejszym zakresie oddziaływania jest `DirectionalLight`. Źródło tego światła znajduje się bardzo daleko, dlatego też zaniechano specyfikowania jego położenia. Należy natomiast zdefiniować wektor wzdłuż którego rozchodzi się światło. Przypisać dany wektor można wykorzystując metodę `setDirection(Vector3f)`. Przykład fabrykacji i przypisania wartości pól można zobaczyć w *Kod. 4*

```
DirectionalLight light = new DirectionalLight();
    light.setEnable(
        true
    );
    light.setColor(
        new Color3f( 1.0f, 1.0f, 1.0f )
    );
    light.setDirection(
        new Vector3f( 1.0f, 0.0f, 0.0f )
    );
```

Kod. 4

Efekt działania światła typu `Directional` widoczny jest na *Rys. 5*, gdzie światło rozchodzi się zgodnie z wyrysowanym wektorem.



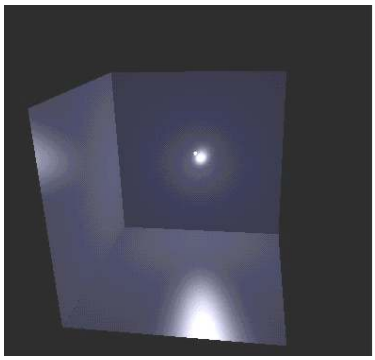
Rys. 5

2.3 PointLight

Kolejnym z dostępnych typów źródeł światła jest `PointLight`. W odróżnieniu od obu omówionych poprzednio, jest określone źródło tego światła. Służy do tego metoda `setPosition(Point3f`), która określa położenie światła w scenie. Implementację tego światła można obejrzeć w *Kod. 5*, natomiast efekt działania na *Rys. 6*.

```
PointLight light = new PointLight( );
    light.setEnable(
        true
    );
    light.setColor(
        new Color3f( 1.0f, 1.0f, 1.0f )
    );
    light.setPosition(
        new Point3f( 0.0f, 1.0f, 0.0f )
    );
    light.setAttenuation(
        new Point3f( 1.0f, 0.0f, 0.0f )
    );
```

Kod. 5



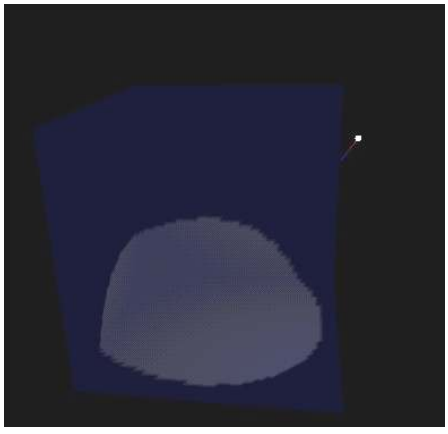
Rys. 6

2.4 SpotLight

Ostatnim z rodzajów źródeł światła, jakie pozostało do opisania jest światło reflektorowe. `SpotLight` można traktować jako połączenie świateł `Directional` i `PointLight`. Funkcje `setDirection(Vector3f)` i `setPosition(Point3f)` zostały tu wzbogacone o parametr rozpraszalności (*spreadness*), któremu odpowiada funkcja `setSpredAngle(double)`. Ponadto dodano funkcję `setConcentration(double)` która odpowiada za skupienie wiązki światła. Przykładową implementację można zobaczyć w *Kod. 6*.

```
SpotLight light = new SpotLight( );
    light.setEnable(
        true
    );
    light.setColor(
        new Color3f( 1.0f, 1.0f, 1.0f )
    );
    light.setPosition(
        new Point3f( 0.0f, 1.0f, 0.0f )
    );
    light.setAttenuation(
        new Point3f( 1.0f, 0.0f, 0.0f )
    );
    light.setDirection(
       new Vector3f( 1.0f, 0.0f, 0.0f )
    );
    light.setSpreadAngle( 0.785f );
    light.setConcentration( 0.0f );
```

Kod. 6

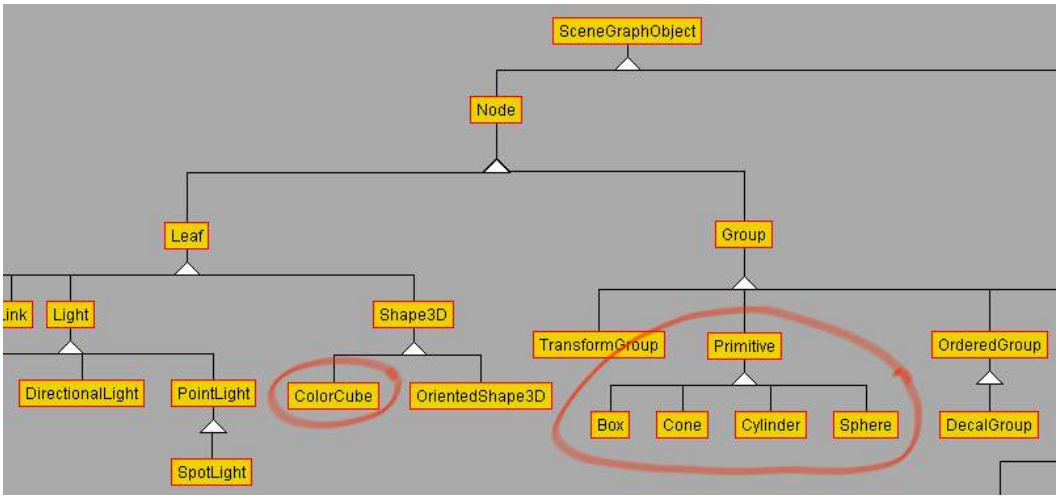


Rys. 7

3. Bryły predefiniowane

W pakiecie Java3D zostało zdefiniowane pięć elementarnych brył. Wszystkie z nich znajdują się w podpakiecie *com.sun.j3d.utils.geometry.**.

Klasy `Box` (sześcian), `Sphere` (kula), `Cone` (stożek), `Cylinder` (walec) dziedziczą po klasie `Primitive`, zaś `ColorCube` (pokolorowany sześcian) po klasie Shape3D. Pokazuje to *Rys. 8*, na którym przedstawiono fragment hierarchii dziedziczenia klas w pakiecie Java3D.



Rys. 8

Konstruktory obiektów reprezentujących bryły i ich metody są zbliżone. Pusty konstruktor oznacza, iż wszystkim parametrom związanym z rozmiarem bryły zostanie przypisana wartość 1.0, a powierzchnia (`Appearance`) zostanie wypełniona standardowym materiałem. Najpełniejszy z konstruktorów opisuje natomiast wszystkie jego parametry.

`Box(`
`  float x,`
`  float y,`
`  float z,`
`  int primflag,`
`  Appearance ap`
`)`
	x, y, z - odpowiadają wielkościom boków w prostopadłościanie
	ap – określa rodzaj powierzchni danej bryły

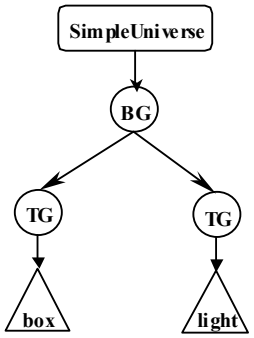
`ColorCube(`
`  double scale`
`)`
	bryła ta ma już predefiniowane rozmiary wszystkich boków dlatego też jako jej parametr podawać można jedynie skalę całego obiektu

`Cone (`
`  float radius,`
`  float height,`
`  int primflags,`
`  int xdivision,`
`  int ydivision,`
`  Appearance ap`
`)`
	radius – promień podstawy
	heigth – wysokość stożka
	xdivision – liczba segmętów w poziomie
	ydivision – liczba segmentów w pionie
	ap – określa rodzaj powierzchni danej bryły

`Cylinder(`
`  float radius,`
`  float height,`
`  int primflags,`
`  int xdivision,`
`  int ydivision,`
`  Appearance ap`
`)`
	jak w `Cone`

`Sphere(`
`  float radius,`
`  int primflags,`
`  int divisions,`
`  Appearance ap`
`)`
	jak w `Cone`

W przykładowych programach *Prog. 2* i *Prog. 3* z rozdziału „Przekształcenia afiniczne na grupach” wykorzystany został obiekt `ColorCube`. Modyfikacja programów tak, aby wykorzystywały inne z predefiniowanych obiektów zakończy się jednak fiaskiem. Program będzie kompilował się poprawnie, lecz inne obiekty niż `ColorCube` nie będą widoczne. Dzieje się tak, ponieważ wszystkie inne obiekty mają zdefiniowane parametry powierzchni, w tym i współczynnik odbicia światła, a jeżeli nie ma światła nie ma również bryły. Dlatego modyfikacje muszą odzwierciedlać poniższy model.



Ilus. 2

Jak widać na *Ilus. 2*, przekształcenia można zamknąć w dodaniu gałęzi odpowiedzialnej za światło.

4. Tworzenie własnych brył

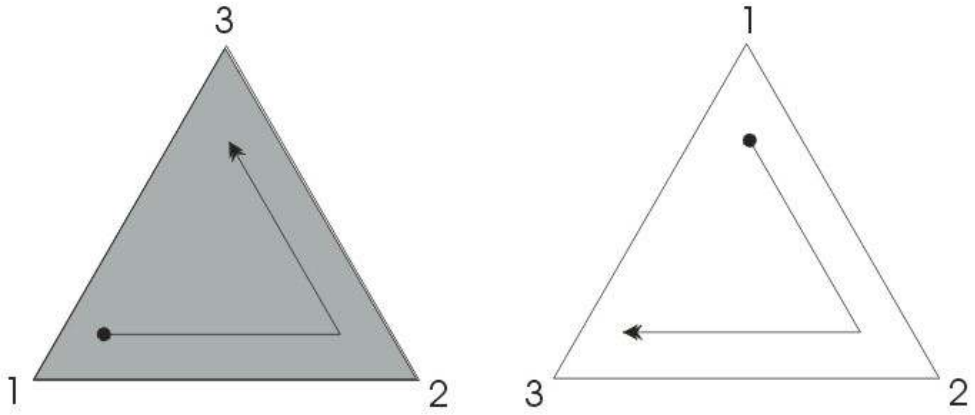
4.1 Kształt

W programach 1,2 i 3 została wykorzystana predefiniowana bryła `ColorCube`. Obiekt ten dziedziczy po klasie `Shape3D` i powstał poprzez opisanie położenia wszystkich punktów znajdujących się w tej bryle. Proces tworzenia obiektów tego typu składa się z następujących etapów :

- przygotowanie tablicy położenia wierzchołków,
- przygotowanie tablicy kolorów wierzchołków
- na podstawie o tablice wierzchołków i kolorów należy utworzyć jeden z obiektów szkieletowych jakimi mogą być `PointArray`, `LineArray`, `TriangleArray`, czy `QuadArray`

Powyższe kroki zostały zademonstrowane w klasie `Ostrosłup`, która ma za zadanie utworzyć ostrosłup o podstawie trójkąta.

Należy jednak wyjaśnić, iż w obiekcie szkieletowym każda ściana jest wykreślana (renderowana) tylko z jednej strony. Mechanizm ten ma na celu oszczędniejsze zarządzanie dostępnymi zasobami maszyny. W związku z taką implementacją użytkownik musi zwrócić uwagę na sposób, w jaki wpisuje informacje o wierzchołkach. Zgodnie z ogólną zasadą, iż kolejne punkty powinny być umieszczane przeciwnie do ruchu wskazówek zegara. Zapis ten jest ilustrowany przez Ilus. 3.



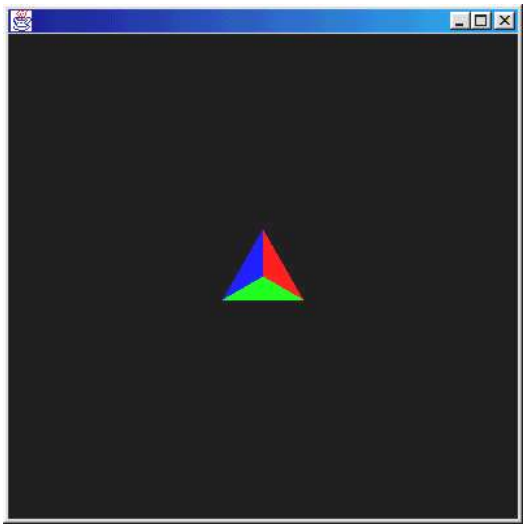
Ilus. 3

```
import javax.media.j3d.*;

public class Ostroslup extends Shape3D{
    private static final float[] wierzcholki = {
        -0.86f, -0.5f, -1.0f,
        0.86f, -0.5f, -1.0f,
        0.0f, 0.0f, 1.0f,
```

```
            0.86f, -0.5f, -1.0f,
            0.0f, 1.0f, -1.0f,
            0.0f, 0.0f, 1.0f,

            0.0f, 1.0f, -1.0f,
            -0.86f, -0.5f, -1.0f,
            0.0f, 0.0f, 1.0f,

            0.0f, 1.0f, -1.0f,
            0.86f, -0.5f, -1.0f,
            -0.86f, -0.5f, -1.0f
        };

        private static float[] kolory = {
            0.0f, 1.0f, 0.0f,
            0.0f, 1.0f, 0.0f,
            0.0f, 1.0f, 0.0f,

            1.0f, 0.0f, 0.0f,
            1.0f, 0.0f, 0.0f,
            1.0f, 0.0f, 0.0f,

            0.0f, 0.0f, 1.0f,
            0.0f, 0.0f, 1.0f,
            0.0f, 0.0f, 1.0f,

            1.0f, 0.0f, 1.0f,
            1.0f, 0.0f, 1.0f,
            1.0f, 0.0f, 1.0f
        };

        public Ostroslup()
        {
            TriangleArray pa = new TriangleArray(
                wierzcholki.length/3,
                TriangleArray.COORDINATES | TriangleArray.COLOR_3
            );

            pa.setCoordinates(0, wierzcholki);
            pa.setColors(0, kolory);

            this.setGeometry(pa);
        }
    }
```

Prog. 4

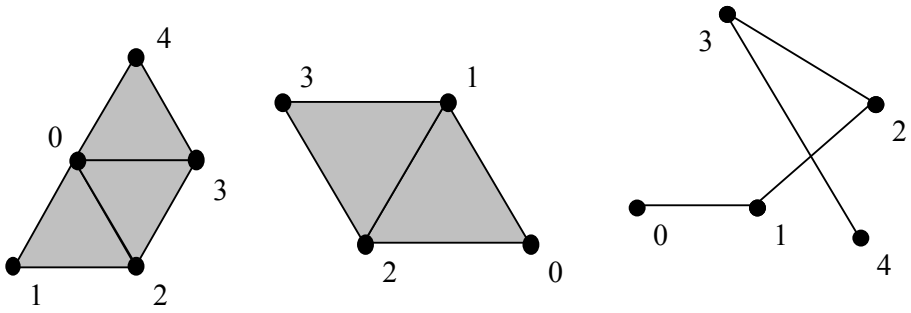
Podczas fabrykowania obiektu klasy `TriangleArray` użyto konstruktora, który wymagał dwóch parametrów. Pierwszy z nich opisuje liczbę wierzchołków ostrosłupa, drugi zaś określa jakiego rodzaju informacje zostaną przekazane. W programie *Prog. 4* przekazano współrzędne oraz kolory (drugi parametr konstruktora) w postaci alternatywy bitów dwóch zmiennych. Następnie wykorzystując metodę `setCoordinates( int, float[])` ustalono położenie wierzchołków począwszy od wierzchołka o numerze 0. Metoda `setColors(int, float[])` działa w podobny sposób z tą różnicą, że opisuje kolory. Gdy bryła jest gotowa metodą `setGeometry(Geometry)` należy przypisać utworzoną bryłę do grupy w której użyto fabrykatora. Wynik wywołania klasy `Ostrosłup` został przedstawiony na *Rys. 9*.



Rys. 9

Ponieważ specyfika ostrosłupa wymagała użycia siatki tworzonej na podstawie trzech punktów, jako obiekt szkieletowy zastosowano `TriangleArray`. W innych bryłach można wykorzystać obiekty szkieletujące innego rodzaju, takie które będą bardziej odpowiadały idei bryły. Przykładem może tu być piramida schodkowa do szkieletowania której można zalecić `QuadArray`.

Innymi typami siatek są `TriangleFunArray`, `TriangleStripArray` i `LineStripArray`. Główna różnica polega na interpretacji wprowadzonych punktów. Podane współrzędne są traktowane jako spójny w przestrzeni ciąg wielokątów. Na *Ilus. 4* przedstawiono efekty uzyskane przy pomocy wymienionych powyżej obiektów strukturujących.



Ilus. 4

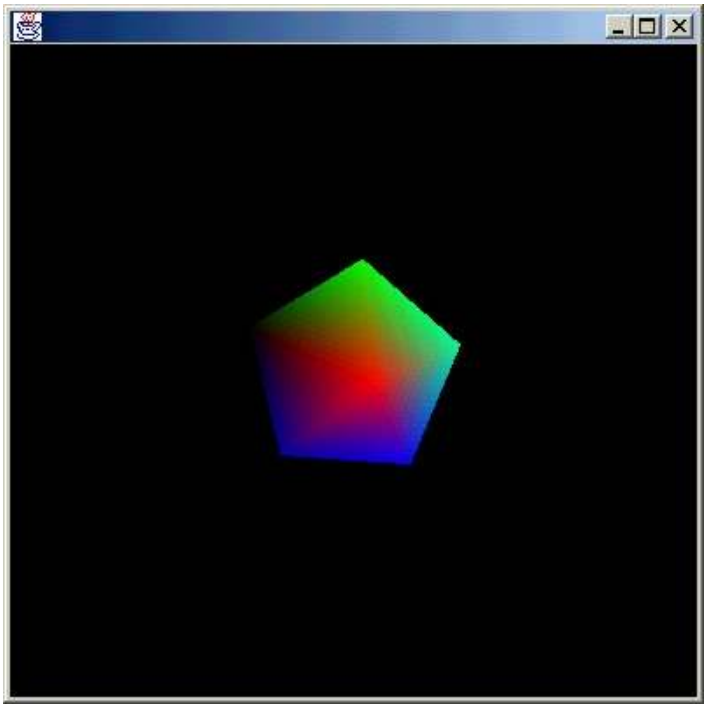
Jako przykład demonstrujący powyższe elementy zostanie utworzony ostrosłup o podstawie wielokąta foremnego, w którym ilość kątów będzie jednym z parametrów konstruktora.

```
import javax.media.j3d.*;

public class Ostroslup extends Shape3D{
    public Ostroslup(int iloscKatow, double promien, float wysokosc){
        double krokKatowy = (2*Math.PI) / iloscKatow,
               kat = 0.0;
        float punkty[] = new float[(iloscKatow+2)*3*2];
        int wyliczonePunkty = 0;

        punkty[wyliczonePunkty] = 0.0f;
        punkty[wyliczonePunkty+1] = wysokosc;
        punkty[wyliczonePunkty+2] = 0.0f;
        wyliczonePunkty+=3;

        while(wyliczonePunkty<punkty.length/2){
            punkty[wyliczonePunkty] = (float) (promien * Math.sin(kat));
            punkty[wyliczonePunkty+1] = 0.0f;
            punkty[wyliczonePunkty+2] = (float)(promien * Math.cos(kat));
            wyliczonePunkty+=3;
            kat += krokKatowy;
        }

        punkty[wyliczonePunkty] = 0.0f;
        punkty[wyliczonePunkty+1] = 0.0f;
        punkty[wyliczonePunkty+2] = 0.0f;
        wyliczonePunkty+=3;
```

```
        while(wyliczonePunkty<punkty.length){
            punkty[wyliczonePunkty] = (float) (promien * Math.sin(kat));
            punkty[wyliczonePunkty+1] = 0.0f;
            punkty[wyliczonePunkty+2] = (float) (promien * Math.cos(kat));
            wyliczonePunkty+=3;
            kat -= krokKatowy;
        }

        int tom[] = {
            (punkty.length/2)/3,
            (punkty.length/2)/3
        };

        TriangleFanArray tfa = new TriangleFanArray(
            punkty.length/3,
            TriangleFanArray.COORDINATES,
            tom
        );

        tfa.setCoordinates(0, punkty);

        this.setGeometry(tfa);
    }
}
```

Prog. 5

Sfabrykowanie tego obiektu, a następnie dodanie go do grupy zaowocuje efektem widocznym na *Rys. 10*.



Rys. 10

Warto zwrócić uwagę na trzeci z parametrów konstruktora `TriangleFanArray`. Jest on tablicą liczb całkowitych, której wielkość określa liczbę obiektów strukturujących. Każdy z elementów tablicy określa liczbę punktów, z jakiej składa się element strukturujący, w związku z tym suma wartości elementów musi być równa wielkości tablicy która zawiera punkty.

Bryły przedstawione w *Prog. 4* i *Prog. 5* mają określony kształt i kolory. Oznacza to iż wykorzystano stałe `COORDINATES` i `COLOR_3`. Pierwsza z tych wartości jest odpowiedzialna za współrzędne wierzchołków opisujących bryłę, druga za kolory przypisywane tym wierzchołkom.

Nieopisane pozostały jeszcze dwie stałe.

`TEXTURE_COORDINATES` odpowiada za określenie punktów na których zostanie rozciągnięta tekstura. W *Prog. 6* wykorzystano tę stałą, jako punktów rozpinających użyto punktów które tworzą obiekt. Efekt uzyskany został zaprezentowane na *Rys. 11*.

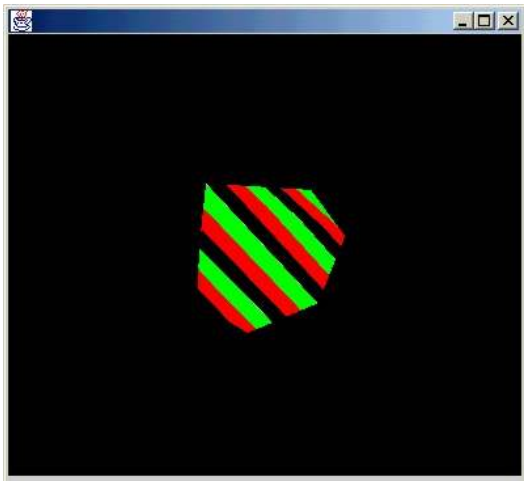
Analizując kod *Prog. 6* można zauważyć, iż wykorzystano tam nie omówiony dotąd obiekt typu `Appearance`. Wszelkie informacje na jego temat zostaną podane w rozdziale „Tworzenie własnych brył – powierzchnia”.

```
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.image.TextureLoader;

public class Ostroslup extends Shape3D{
    OptionFrame of;
    public Ostroslup(int iloscKatow, double promien, float wysokosc){
        double krokKatowy = (2*Math.PI) / iloscKatow,
               kat = 0.0;
        float punkty[] = new float[(iloscKatow+2)*3*2];
```

```
float kolory[] = new float[ punkty.length ];
int wyliczonePunkty = 0;

punkty[wyliczonePunkty] = 0.0f;
punkty[wyliczonePunkty+1] = wysokosc;
punkty[wyliczonePunkty+2] = 0.0f;

kolory[wyliczonePunkty] = 1.0f;
kolory[wyliczonePunkty+1] = 0.0f;
kolory[wyliczonePunkty+2] = 0.0f;

wyliczonePunkty+=3;

while(wyliczonePunkty<punkty.length/2){
    punkty[wyliczonePunkty] = (float) (promien * Math.sin(kat));
    punkty[wyliczonePunkty+1] = 0.0f;
    punkty[wyliczonePunkty+2] = (float)(promien * Math.cos(kat));

    kolory[wyliczonePunkty] = 0.0f;
    kolory[wyliczonePunkty+1] = punkty[wyliczonePunkty];
    kolory[wyliczonePunkty+2] = punkty[wyliczonePunkty+2];

    wyliczonePunkty+=3;
    kat += krokKatowy;
}

punkty[wyliczonePunkty] = 0.0f;
punkty[wyliczonePunkty+1] = 0.0f;
punkty[wyliczonePunkty+2] = 0.0f;

kolory[wyliczonePunkty] = 1.0f;
kolory[wyliczonePunkty+1] = 0.0f;
kolory[wyliczonePunkty+2] = 0.0f;

wyliczonePunkty+=3;

while(wyliczonePunkty<punkty.length){
    punkty[wyliczonePunkty] = (float) (promien * Math.sin(kat));
    punkty[wyliczonePunkty+1] = 0.0f;
    punkty[wyliczonePunkty+2] = (float) (promien * Math.cos(kat));

    kolory[wyliczonePunkty] = 0.0f;
    kolory[wyliczonePunkty+1] = punkty[wyliczonePunkty];
    kolory[wyliczonePunkty+2] = punkty[wyliczonePunkty+2];

    wyliczonePunkty+=3;
```

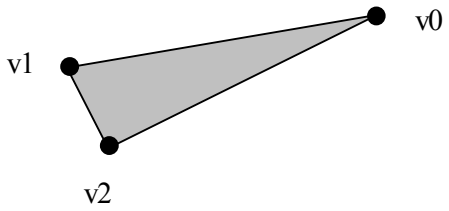
```
            kat -= krokKatowy;
        }

        int tom[] = {
            (punkty.length/2)/3,
            (punkty.length/2)/3
        };

        TriangleFanArray tfa = new TriangleFanArray(
            punkty.length/3,
            TriangleFanArray.COORDINATES |
            TriangleFanArray.COLOR_3 |
            TriangleFanArray.TEXTURE_COORDINATE_3,
            tom
        );


        tfa.setCoordinates(0, punkty);
        tfa.setTextureCoordinates(0, punkty);
        tfa.setColors(0, kolory);
// Eksperymenty z Appearance

        Appearance ap = new Appearance();
            ColoringAttributes ca = new ColoringAttributes();
                ca.setShadeModel(ColoringAttributes.SHADE_FLAT );
                ca.setColor(0.0f, 1.0f, 0.0f);
                ca.setCapability(
                    ColoringAttributes.ALLOW_SHADE_MODEL_WRITE
                );
            ap.setColoringAttributes(ca);
            TextureLoader tl = new TextureLoader( "bg.jpg", null);

            Material ma = new Material();
                ma.setLightingEnable(true);
            ap.setTexture(tl.getTexture());
            ap.setCapability(
                Appearance.ALLOW_COLORING_ATTRIBUTES_READ
            );
            ap.setCapability(
                Appearance.ALLOW_COLORING_ATTRIBUTES_WRITE
            );
        this.setAppearance(ap);
        this.setGeometry(tfa);
    }
}
```

Prog. 6

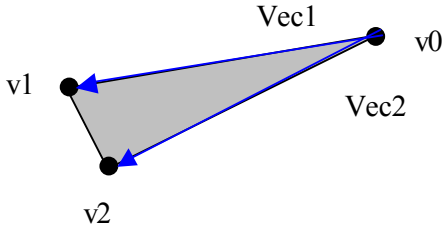


Rys. 11

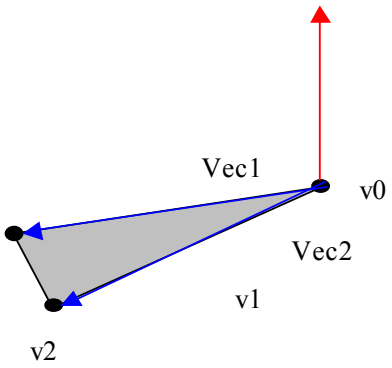
Za informacje o tym, z której strony i w jakim stopniu dana powierzchnia będzie oświetlana odpowiada parametr `NORMALS`. Metoda `setNormals()` jako argument przyjmuje tablicę wektorów definiujących które z promieni, rozchodzącego się światła wpływają na wygląd powierzchni wieloboku. Metodologia tworzenia tablicy wektorów tego typu została przedstawiona na kolejnych rysunkach poniżej.



Procedurę należy rozpocząć od utworzenia dwóch wektorów, niech będą to wektory utworzone pomiędzy parami punktów (v0, v1) i (v0, v2). Wykorzystując konstruktor `Vector3f(Point, Point)` utworzono dwa wektory o nazwach `Vec1` i `Vec2`.

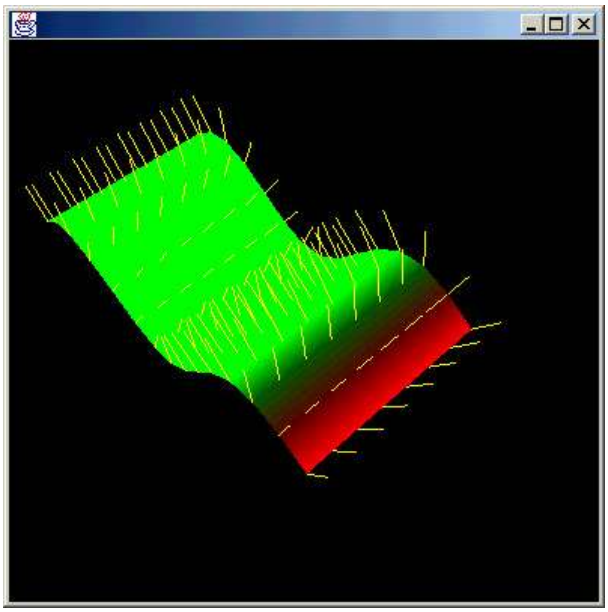


Następnym krokiem powinno być utworzenie wektora wypadkowego, powstałego przy wykorzystaniu metody `cross(Vector3f, Vector3f)`. Wektor ten oznaczono `VecW`.



Ostatnim krokiem jest znormalizowanie otrzymanego wyniku za pomocą metody `normalize()`.

Rezultat uzyskany w wyniku powyższych kroków należy przypisać odpowiednio punktom `v0`, `v1` i `v2`. Opisany algorytm znalazł swoją implementację w *Prog. 7*, w którym można zobaczyć płaszczyznę powstałą na bazie sinusoidy. Obiekt ten dodatkowo pokazuje wszystkie wyprowadzone wektory normalne, tak jak to pokazano na *Rys. 12*.



Rys. 12

J3Dappl.java

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
import com.sun.j3d.utils.applet.MainFrame;

import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
```

```
import com.sun.j3d.utils.behaviors.mouse.*;
import com.sun.j3d.utils.behaviors.picking.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

import javax.media.j3d.*;
import javax.vecmath.*;

public class J3DAppl extends Frame{

    public J3DAppl(){
        Canvas3D c3d = new Canvas3D(null);
            SimpleUniverse su = new SimpleUniverse(c3d);
                BranchGroup objRoot = new BranchGroup();

                    TransformGroup tg = new TransformGroup();
                        TransformGroup group = new TransformGroup();
                            TransformGroup objScale1 = new TransformGroup();
                                Transform3D trans1 = new Transform3D();
                                    trans1.setScale(0.2);

                                    trans1.setTranslation(
                                        new Vector3d(0.0, -0.5, 0.0)
                                    );

                                    trans1.setRotation(
                                        new AxisAngle4d( 1, 0, 0, -1.5)
                                    );

                                objScale1.setTransform(
                                    trans1
                                );
                                Plaszczyzna1 p = new Plaszczyzna1(0.0);
                                objScale1.addChild(
                                    p
                                );

                                objScale1.setCapability(
                                    TransformGroup.ALLOW_TRANSFORM_READ
                                );
                                objScale1.setCapability(
                                    TransformGroup.ALLOW_TRANSFORM_WRITE
                                );

                            TransformGroup objScale2 = new TransformGroup();
                                objScale2.setTransform(
                                    trans1
```

```
                        );

                        objScale2.addChild(
                            new Wektor(p)
                        );

                    MouseRotate beh1 = new MouseRotate(
                        group
                    );

                    beh1.setSchedulingBounds(
                        new BoundingSphere(
                            new Point3d(0.0, 0.0, 0.0),
                            1
                        )
                    );
                group.setCapability(
                    TransformGroup.ALLOW_TRANSFORM_READ
                );
                group.setCapability(
                    TransformGroup.ALLOW_TRANSFORM_WRITE
                );
                group.addChild(beh1);
                group.addChild(objScale1);
                group.addChild(objScale2);
            tg.addChild(group);

            objRoot.addChild(tg);
            objRoot.compile();
        su.addBranchGraph(objRoot);
        su.getViewingPlatform().setNominalViewingTransform();
    this.setSize( 400, 400);
    this.add(c3d, "Center");
    this.addWindowListener(
        new WindowAdapter(){
            public void windowClosing(WindowEvent evt){
                System.exit(0);
            }
        }
    );
    this.setVisible(true);
}

public static void main(String[] args){
    System.out.println("Tomasz");
    new J3DAppl();
```

```
    }
}
```

Plaszczyzna1.java

```
import javax.media.j3d.*;
import javax.vecmath.*;

public class Plaszczyzna1 extends Shape3D{
    int iloscSegX = 14,
        iloscSegY = 14;
    float tablicaPunktow[] = new float[ ((1+(3+((iloscSegX-1)*2)))*iloscSegY*3)];
    float kolory[] = new float[tablicaPunktow.length];
    double przesuniecie = 0;
    TriangleStripArray tsa;

    public Plaszczyzna1(double przesuniecie){
        this.przesuniecie = przesuniecie;
        int rozmieszczenie[] = new int[iloscSegY];

        for(int i=0; i<iloscSegY;i++){
            rozmieszczenie[i] = (1+(3+((iloscSegX-1)*2)));
        }

        generatorPunktow();

        tsa = new TriangleStripArray(
            tablicaPunktow.length /3,
            GeometryArray.COORDINATES |
            GeometryArray.NORMALS |
            GeometryArray.COLOR_3
           ,
            rozmieszczenie
        );

        // Process zwany normalizacja

        int licznik = 0,
            poz = 0;
        for(int i=0; i<iloscSegY; i++){
            Vector3f normal = new Vector3f();
            int j;

            for(j=0; j<(3+(iloscSegX-1)*2)-1;j+=2){
```

```
            Vector3f v1 = new Vector3f(),
                     v2 = new Vector3f();

            v1.sub(
                new Point3f(
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)],
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+1],
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+2]
                ),
                new Point3f(
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+3],
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+4],
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+5]
                )
            );

            v2.sub(
                new Point3f(
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)],
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+1],
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+2]
                ),
                new Point3f(
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+6],
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+7],
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+8]
                )
            );

            normal.cross(v1,v2);
            normal.normalize();

            tsa.setNormal((i*(4+(iloscSegX-1)*2))+j, normal);
            tsa.setNormal((i*(4+(iloscSegX-1)*2))+j+1, normal);
            tsa.setNormal((i*(4+(iloscSegX-1)*2))+j+2, normal);

        }
        tsa.setNormal((i*(4+(iloscSegX-1)*2))+j+1, normal);
    }

    tsa.setCoordinates(0, tablicaPunktow);
    tsa.setColors(0, kolory);
    this.setCapability(Geometry.ALLOW_INTERSECT);
    this.setGeometry(tsa);
}
```

```
public Vector3f usrednij(Vector3f normal, int numer){
    Vector3f v3f = new Vector3f();
    tsa.getNormal(numer, v3f);
    v3f.x = (normal.x+v3f.x)/2;
    v3f.y = (normal.y+v3f.y)/2;
    v3f.z = (normal.z+v3f.z)/2;
    return v3f;
}

public void generatorPunktow(){
    double dlugoscX = 10.0,
           dlugoscY = 10.0,
           krokX = dlugoscX / (iloscSegX+1),
           krokY = dlugoscY / (iloscSegY+1),
           punktStartuX = dlugoscX/2,
           punktStartuY = dlugoscY/2;

    double kat = 0,
           x = 1.0,
           y =  -1.0,
           z = Math.sin( ((kat*Math.PI)/180)+przesuniecie);

    double krokKatowy = ((360.0*1.5) / (iloscSegX+1));

    int wstawianyPunkt = 0;
    int punkt = 0;

    for(int i=0; i < iloscSegY; i++){
        tablicaPunktow[wstawianyPunkt  ] = (float) x;
        tablicaPunktow[wstawianyPunkt+1] = (float) y;
        tablicaPunktow[wstawianyPunkt+2] = (float) z;

        kolory[wstawianyPunkt  ] = (float) x;
        kolory[wstawianyPunkt+1] = (float) (x*-1);
        kolory[wstawianyPunkt+2] = 0.0f;

        wstawianyPunkt+=3;
        punkt++;
        y+=krokY;

        for(int j=0; j<(3+(iloscSegX-1)*2);j++){

            tablicaPunktow[wstawianyPunkt  ] = (float) x;
            tablicaPunktow[wstawianyPunkt+1] = (float) y;
            tablicaPunktow[wstawianyPunkt+2] = (float) z;
```

```
                kolory[wstawianyPunkt  ] = (float)x;
                kolory[wstawianyPunkt+1] = (float) (x*-1);
                kolory[wstawianyPunkt+2] = 0.0f;

                if( (j%2)==0){
                    wstawianyPunkt+=3;
                    punkt++;
                    kat += krokKatowy;

                    x-=krokX;
                    y-=krokY;
                    z = Math.sin( ((kat*Math.PI)/180)+przesuniecie);
                }else{
                    wstawianyPunkt+=3;
                    punkt++;
                    y+=krokY;
                }
            }
            kat = 0;
            x = 1.0;
            y += krokY;
            z = Math.sin( ((kat*Math.PI)/180)+przesuniecie);
        }


       for(int i = tablicaPunktow.length/2, j = (tablicaPunktow.length/2)-1; i<
tablicaPunktow.length;i++){
            tablicaPunktow[i] = tablicaPunktow[j];
       }

   }
}
```

Wektor.java

```
import javax.media.j3d.*;
import javax.vecmath.*;

public class Wektor extends Shape3D{
    float punktyWektora[];

    public Wektor(Plaszczyzna1 p){
        punktyWektora = new float[ (p.tablicaPunktow).length * 2];
        int punktPlaszczyzny = 0,
            punktWektora = 0;
```

```
        for(int i=0; i < (p.tablicaPunktow).length/3; i++){
            Vector3f v3f = new Vector3f();
            (p.tsa).getNormal(i, v3f);

            punktyWektora[punktWektora] = p.tablicaPunktow[punktPlaszczyzny];
            punktyWektora[punktWektora+1] = p.tablicaPunktow[punktPlaszczyzny+1];
            punktyWektora[punktWektora+2] = p.tablicaPunktow[punktPlaszczyzny+2];

            punktyWektora[punktWektora+3] = punktyWektora[punktWektora]+v3f.x;
            punktyWektora[punktWektora+4] = punktyWektora[punktWektora+1]+v3f.y;
            punktyWektora[punktWektora+5] = punktyWektora[punktWektora+2]+v3f.z;
            punktPlaszczyzny+=3;
            punktWektora+=6;
        }

        LineArray la = new LineArray(
            punktyWektora.length/3,
            LineArray.COORDINATES
        );
        la.setCoordinates(0, punktyWektora);

        Appearance app = new Appearance();
            ColoringAttributes ca = new ColoringAttributes();
                ca.setColor(
                    new Color3f(1.0f, 1.0f, 0.0f)
                );
            app.setColoringAttributes(ca);
        this.setAppearance(app);
        this.addGeometry(la);
    }
}
```

Prog. 7

4.2 Powierzchnia

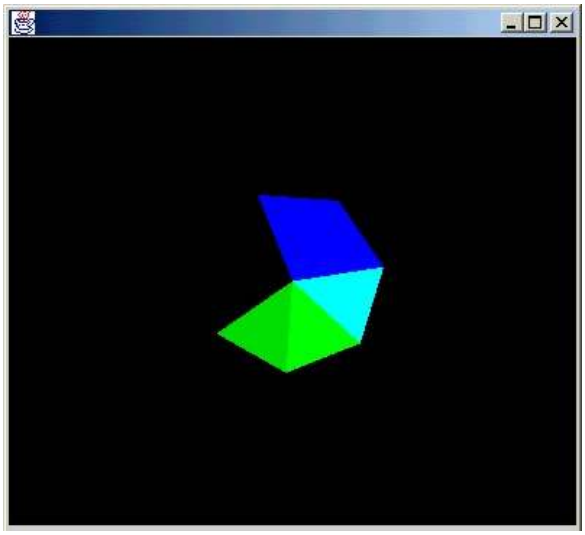
W pakiecie Java3D powierzchnie są opisane klasą `Appearance`. Kolor, materiał, przezroczystość oraz inne parametry są przypisywane danemu obiektowi przez odpowiednie metody.

Metoda `setColoringAttributes( ColoringAttributes coloringAttributes )` wywoływana na rzecz `Appearance` jest odpowiedzialna za przypisanie obiektu typu `ColoringAttributes` polu powierzchni poświęconemu kolorowaniu.

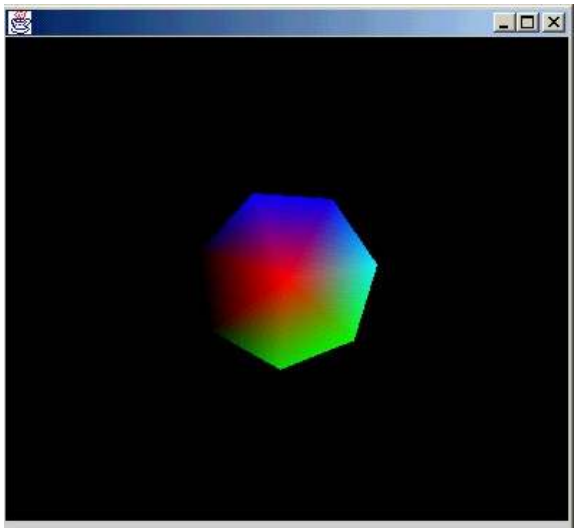
Klasa `ColoringAttributes` przechowuje atrybuty koloru.

Metoda `setShadeModel(int model)` odpowiada za sposób w jaki będą wykreślane powierzchnie w obiekcie. Można wybrać tu cztery rodzaje parametrów:

- `SHADE_FLAT` – ściana będzie miała kolor zgodny z kolorem pierwszego punktu tworzącego tę ścianę (*Rys. 13*)
- `SHADE_GOURAUD` – przejście pomiędzy kolorami kolejnych punktów będzie stopniowe (*Rys. 14*)
- `NICEST` – najładniejszy
- `FASTEST` - najszybszy.



Rys. 13



Rys. 14

Metoda `setColor(Color3f color3f)` jest odpowiedzialna za kolor wewnętrzny bryły (`intrinsic color`). Był on ustawiany w *Prog. 5* i *Prog. 8* przez co możliwe było uzyskanie efektu wielokolorowych powierzchni przy jednoczesnym braku jakiegokolwiek oświetlenia.

Metoda `setMaterial(Material material)` wywoływana na rzecz `Appearance` opisuje parametry powiązane z materiałem. Klasa `Material` zawiera sześć kluczowych pól, które opisują zachowanie światła padającego na daną powierzchnię. Dlatego też dane zapisane w tych polach wchodzą w interakcje tylko gdy w scenie zostało zdefiniowane światło.

Metoda `setAmbientColor (Color3f color3f)` jest składową w równaniu które definiuje w jaki sposób powstaje odbite światło rozproszone. (UWAGA: parametr ten wchodzi w interakcje tylko z światłem rozproszonym (ang. *Ambient*)).

Metoda `setDifussColor(Color3f color3f)` określa kolor obiektu w każdych warunkach oświetleniowych.

Metoda `setEmissivColor(Color3f color3f)` określa rodzaj i natężenie światła emanującego z obiektu, niezależnie od liczby zewnętrznych źródeł oświetlenia istniejących w scenie. Można w ten sposób uzyskać np. efekt obiektu jarzącego się w ciemności.

Metoda `setSpecularColor(Color3f color3f)` w przypadku, gdy kąt pomiędzy źródłem światła a powierzchnią odbijającą jest bliski kątowi pomiędzy tą powierzchnią a osobą patrzącą, ma wpływ na tworzenie efektu połysku gładkiej powierzchni oraz na stopień rozproszenia koloru

Metoda `setShiness(float shiness)` określa stopień wygładzenia powierzchni oraz stopień jej połysku / matowości

Najlepszym sposobem, aby zaprezentować opisane powyżej właściwości, będzie stworzenie płaszczyzny.

5. Interakcja

W rozdziale „Węzeł grupujący *TransformGroup*” spotkano się z problemem pokazania bryły z innej perspektywy. Użyto wówczas przypisania tej bryle określonego kąta lub skali. Przedstawione tam rozwiązanie problemu zmuszało użytkownika do określenia położenia oraz kąta obroty bryły już na etapie tworzenia sceny. Wykorzystanie sterowania umożliwi przełamanie opisanych barier, wynikiem czego będzie scena z obiektami które mogą być dynamicznie obracane lub poruszane.

Istnieją dwa rodzaje sterowania: wykorzystujące klawiaturę oraz wykorzystujące mysz. Obie metody zostały zdefiniowane w pakiecie *com.sun.j3d.utils.behaviors.**, który musi zostać zaimportowany. Ponieważ operacja translacji lub rotacji wymaga dokonania odczytu i zapisu informacji o relacji grupy z sceną, dlatego też konieczne jest wyrażenie zgody na te operacje. Dokonuje się tego poprzez metodę `setCapability(int)`, gdzie jako parametr dostarczana jest zmienna statyczna klasy `TransformGroup`. Przykładowo zezwolenie na odczyt przedstawiona *Kod. 7*, a analogiczne zezwolenie zapisu *Kod. 8*.

```
objScale1.setCapability(
    TransformGroup.ALLOW_TRANSFORM_READ
);
```

Kod. 7

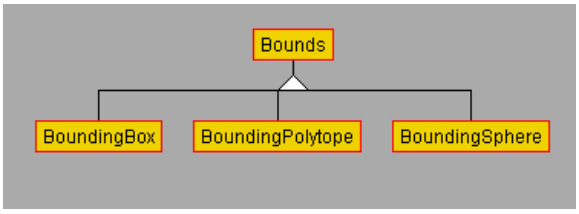
```
objScale1.setCapability(
    TransformGroup.ALLOW_TRANSFORM_WRITE
);
```

Kod. 8

Interakcja w scenie przy wykorzystaniu klawiatury polega na wykonywaniu operacji przy wykorzystaniu klawiszy sterowania, którymi są cztery strzałki. Aby scena została wzbogacona o taki element, trzeba do niej dodać obiekt klasy `KeyNavigatorBehavior`. Fabrykacja tego obiektu wymaga wykorzystania konstruktora w następującej postaci :

```
KeyNavigatorBehavior(TransformGroup target)
```

w którym parametrem jest odnośnik do grupy, która będzie podlegała modyfikacjom. Warunkiem koniecznym poprawnego funkcjonowania tego obiektu jest określenie obszaru ograniczającego. Można tego dokonać wykorzystując metodę `setSchedulingBounds` ,parametrem której będzie jeden z obiektów ograniczających. Hierarchię dziedziczenia tych obiektów przedstawia *Rys. 15*.



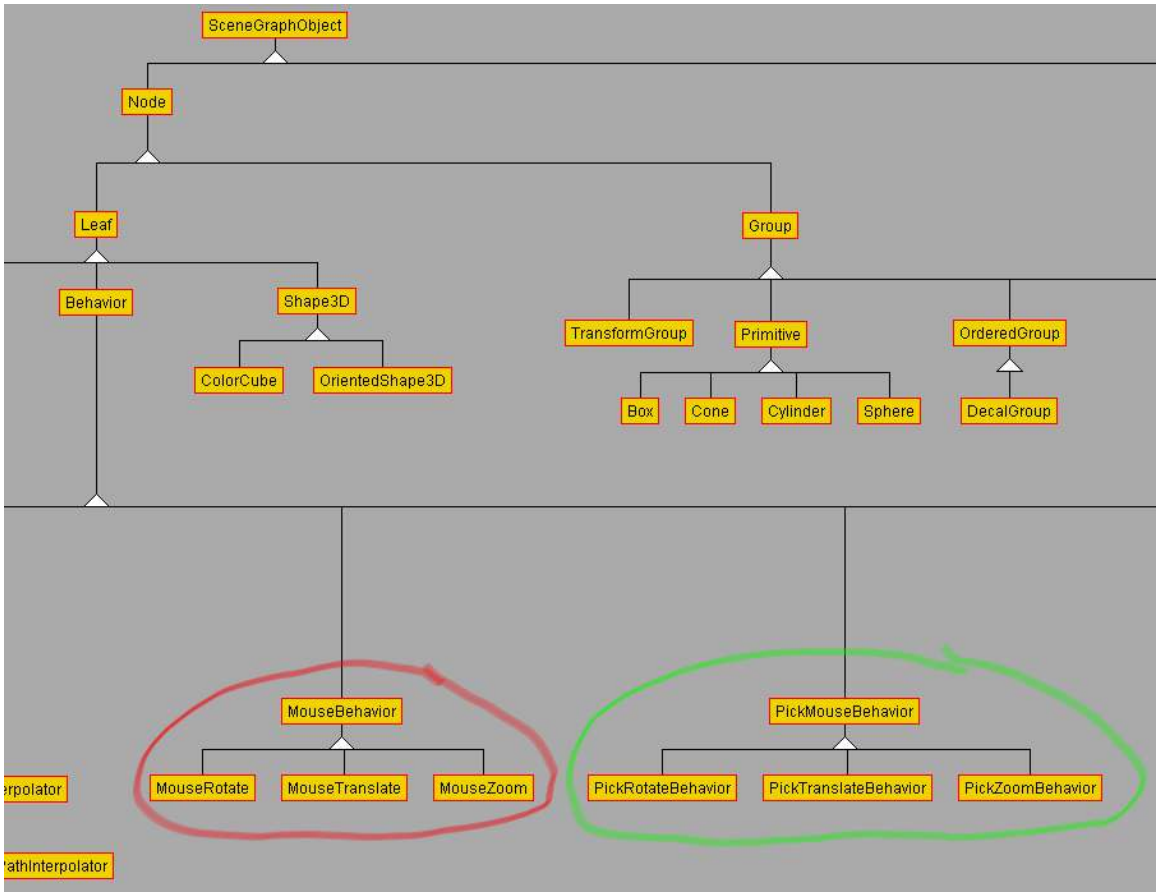
Rys. 15

Główne konstruktory klas przedstawionych na *Rys. 15* pokazano w poniższej tabeli.

`BoundingBox( Point3d lower, Point3d upper )`	parametry lower i upper odpowiadają położeniu dwóch przeciwległych wierzchołków sześcianu ograniczającego
`BoundingPolytope( Vector4d[] planes )`	parametr planes jest tablicą wektorów które opisują przestrzeń ograniczającą
`BoundingSphere( Point3d center, double range )`	parametr center określa środek kuli a paramet range określa promień kuli ograniczającej

Sfabrykowany obiekt z ustalonym obszarem oddziaływania należy dodać do sceny. Przykładową implementację powyższego sterowania można oglądać w *Prog. 8,* gdzie wyróżniono nowo zdefiniowane fragmenty kodu.

Jak już wspomniano, sterowanie wykorzystujące klawiaturę nie jest jedynym sposobem manipulacji obiektami sceny. Alternatywą jest wykorzystanie myszy. Aby wykorzystać to urządzenie należy wykorzystać obiekt klas dziedziczących po klasie `Behavior`, hierarchię dziedziczenia której przedstawia *Rys. 16*.



Rys. 16

Kolorem czerwonym zostały zaznaczone trzy obiekty: `MouseRotate`, `MouseTranslate` i `MouseZoom`, które odpowiadają odpowiednio za obrót, przesunięcie i skalę. Ich konstruktory, podobnie jak w omówionym obiekcie sterowania klawiaturą, przyjmują jeden parametr jakim jest obiekt klasy `TransformGroup` który definiuje grupę na której będą dokonywane operacje. Również podobnie do nawigacji z klawiatury konieczne jest zdefiniowanie obszaru ograniczającego.

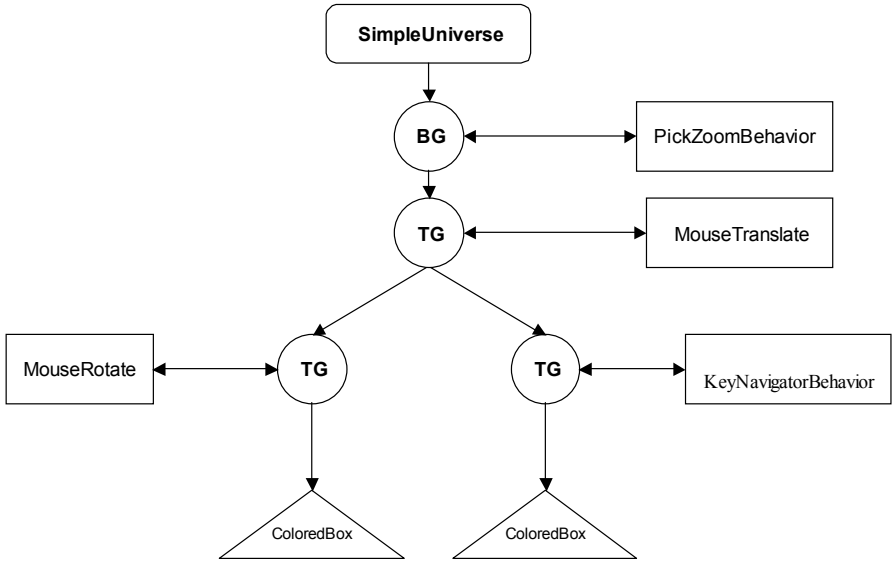
Natomiast fragment rysunku obwiedziony kolorem zielonym obejmuje klasy `PickRotateBehavior`, `PickTranslationBehavior`, oraz `PickZoomBehavior`. Wszystkie one dziedziczą po klasie `PickMouseBehavior`. W odróżnieniu od opisanych już obiektów zasięg działania tego sterowania różni się znacznie. W poprzednich przypadkach obiekt podlegający manipulacji był definiowany i trwale związany z obiektem sterowania na poziomie kodu. Użytkownik miał wpływ na scenę, lecz był ograniczony ideą twórcy. Klasy które zostały zaimplementowane w pakiecie Java3D i dziedziczą po klasie `PickMouseBehavior` pozwalają na rozszerzenie uprawnień użytkownika, przez co może on dokonać wyboru obiektu który będzie podlegał manipulacji. Użycie któregokolwiek z obiektów wiąże się z jego fabrykacją, do której wymagany jest konstruktor. Ponieważ konstruktory wszystkich klas opisanych w niniejszym paragrafie maja takie same parametry, omówione zostaną łącznie na przykładzie klasy `PickZoomBehavior` przedstawionej w *Tab. 3*

`PickZoomBehavior(`
`BranchGroup bg,`
`Canvas3D c3d,`
	bg – jest nadrzędnym obiektem grupującym w scenie
	c3d – to miejsce wyrysowywania

`Bounds bounds` `)`	bounds – to obiekt ograniczający zasięg działania manipulacji

Tab. 3

Aby zademonstrować użycie opisanych powyżej klas, utworzono scenę, której schemat blokowy jest widoczny na poniższej ilustracji .



Ilus. 5

Jak widać każda z utworzonych brył została zaimplementowana w oddzielnej grupie, ponadto utworzono dodatkową grupę która łączy wszystkie elementy sceny. Zaplanowanym efektem jest możliwość zmiany kąta nachylenia lewego sześcianu poprzez wciśnięcie lewego przycisku myszy i jednoczesnego ruchu kursorem, zmiany położenia prawego obiektu poprzez wciskanie klawiszy sterujących (↑, ↓, ←, →), zmiany położenia wszystkich obiektów w scenie w wyniku wciśnięcia prawego przycisku myszy i jednoczesnego ruchu kursorem, oraz zmiany skali wszystkich obiektów znajdujących się wewnątrz grupy `BranchGroup` poprzez przytrzymanie klawiszy ALT i lewego przycisku myszy podczas poruszania myszą. Powyższy schemat jest implementowany przez *Prog. 8*, w którym zaznaczono kolorami nowo zdefiniowane fragmenty kodu.

```
import java.awt.*;
import java.awt.event.*;

import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.mouse.*;
import com.sun.j3d.utils.behaviors.picking.*;
import com.sun.j3d.utils.behaviors.keyboard.*;


import javax.media.j3d.*;
import javax.vecmath.*;
```

```
public class J3DAppl extends Frame{

    public J3DAppl(){
        Canvas3D c3d = new Canvas3D(null);
            SimpleUniverse su = new SimpleUniverse(c3d);
                BranchGroup objRoot = new BranchGroup();

                    PickZoomBehavior pzb = new PickZoomBehavior(
                        objRoot,
                        c3d,
                        new BoundingSphere(
                            new Point3d(0.0,0.0,0.0),
                            100.0
                        )
                    );

                    TransformGroup tg = new TransformGroup();
                        tg.setCapability(TransformGroup.ENABLE_PICK_REPORTING);
                        tg.addChild(pzb);
                        TransformGroup objScale1 = new TransformGroup();
                            Transform3D trans1 = new Transform3D();
                            trans1.setScale(0.2);
                            trans1.setTranslation(
                                new Vector3d(-0.5, 0, 0)
                            );

                        objScale1.setTransform(trans1);
                        objScale1.addChild(
                            new ColorCube()
                        );

                        MouseRotate beh1 = new MouseRotate(
                            objScale1
                        );
                        beh1.setSchedulingBounds(
                            new BoundingSphere(
                                new Point3d(0.0, 0.0, 0.0),
                                0.5
                            )
                        );

                        objScale1.setCapability(
                            TransformGroup.ALLOW_TRANSFORM_READ
                        );
```

```
        objScale1.setCapability(
            TransformGroup.ALLOW_TRANSFORM_WRITE
        );

        objScale1.addChild(beh1);

        TransformGroup objScale2 = new TransformGroup();
            Transform3D trans2 = new Transform3D();
            trans2.setScale(0.2);
            trans2.setTranslation(
                new Vector3d(0.5, 0.0, 0.0)
        );
        objScale2.setTransform(trans2);
        objScale2.addChild(
            new ColorCube()
        );

        KeyNavigatorBehavior knb = new KeyNavigatorBehavior(
            objScale2
        );
        knb.setSchedulingBounds(
            new BoundingSphere(
                new Point3d( 0.0, 0.0, 0.0),
                100.0
            )
        );

        objScale2.setCapability(
            TransformGroup.ALLOW_TRANSFORM_READ
        );
        objScale2.setCapability(
            TransformGroup.ALLOW_TRANSFORM_WRITE
        );

        objScale2.addChild(knb);

    tg.addChild(objScale1);
    tg.addChild(objScale2);

    tg.setCapability(
        TransformGroup.ALLOW_TRANSFORM_READ
    );
    tg.setCapability(
        TransformGroup.ALLOW_TRANSFORM_WRITE
    );
    MouseTranslate beh0 = new MouseTranslate(tg);
```

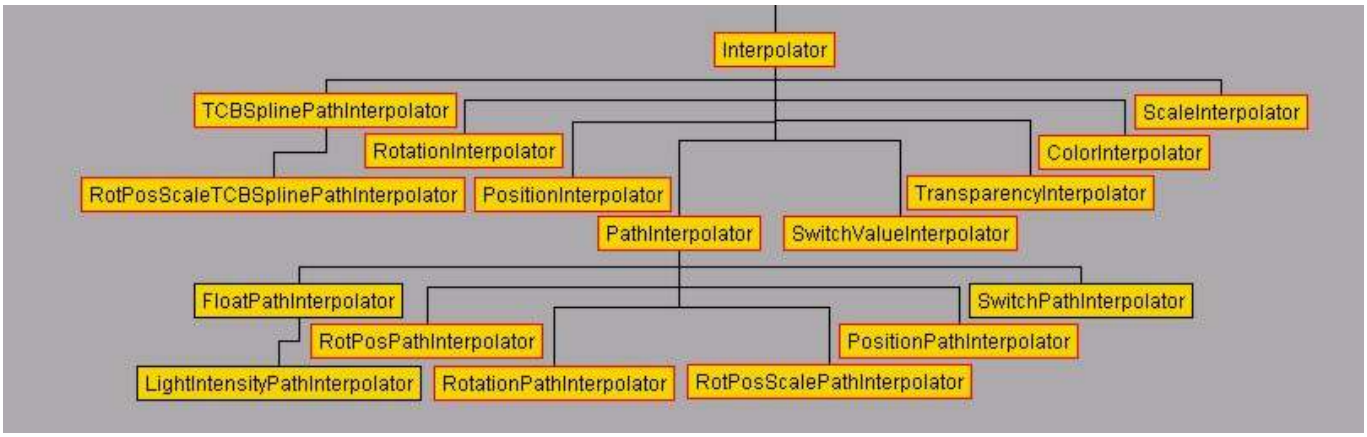
```
                    tg.addChild(beh0);
                        BoundingSphere bs = new BoundingSphere(
                            new Point3d(0.0, 0.0, 0.0),
                            100.0
                        );
                    beh0.setSchedulingBounds(bs);

                objRoot.addChild(tg);
                objRoot.compile();
            su.addBranchGraph(objRoot);
            su.getViewingPlatform().setNominalViewingTransform();
        this.setSize( 400, 400);
        this.add("Center", c3d);
        this.addWindowListener(
            new WindowAdapter() {
                public void windowClosing(WindowEvent evt)
                {
                    System.exit(0);
                }
            }
        );
        this.setVisible(true);
    }

    public static void main(String[] args){
        new J3DAppl();
    }
}
```

Prog. 8

6. Animacja

Animacja – jest procesem zmiany przynajmniej jednego parametru obiektu, w określonym czasie.

Pakiet Java3D wykorzystuje technikę interpolacji stanów pośrednich pomiędzy wartościami parametru w dwóch punktach czasu. Klasy które realizują ten proces zostały nazwane interpolatorami. Ponieważ jeden interpolator nie może sterować polami wszystkich typów, utworzono kilkanaście klas nazwy których zawierają informację o nazwie pola jakie będzie podlegać zmianom i sufiks *interpolator*. Przykładowo klasą odpowiedzialną za zmianę położenia obiektu będzie `PositionInterpolator`, a za zmianę skali będzie odpowiedzialny `ScaleInterpolator`. Wykaz wszystkich interpolatorów przedstawiono na *Rys. 17*.



Rys. 17

Ze względu na liczbę wykonywanych operacji interpolatory można podzielić na proste i złożone. Prostym interpolatorem nazwano klasę która dokonuje tylko jednego rodzaju zmian np. tylko jednego obrotu obiektu względem jednej osi w jednym cyklu czasowym. Złożony może natomiast przechowywać i wykonywać wiele takich operacji np. przez połowę cyklu dokonywany jest obrót względem osi X, a przez drugą połowę cyklu wykonywany jest obrót względem osi Y. Przykłady konstruktorów prostych i złożonych interpolatorów przedstawiono w *Tab. 4*

```
PositionInterpolator(
    Alpha alpha,
    TransformGroup target
)
```

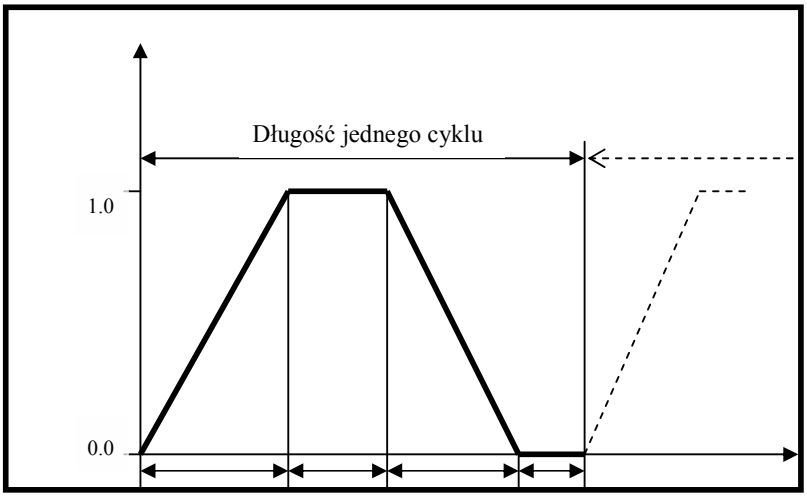
```
PositionPathInterpolator(
    Alpha alpha,
    TransformGroup target,
    Transform3D axisOfTranslation,
    float[] knots,
    Point3f[] positions
)
```

Tab. 4 Konstruktorów obiektów pochodnych od klasy Interpolator

Ponieważ nie można łączyć interpolacji tj. w obrębie jednej grupy nie można wykorzystać dwóch interpolatorów utworzono oddzielne klasy poświęcone animacjom łączonym. Przykładem może być klasa `RotPosPathInterpolator` która obsługuje obrót i pozycję.

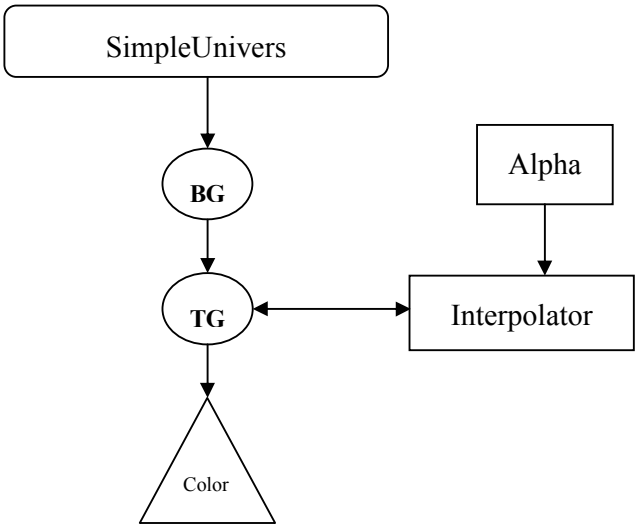
Klasy implementujące interpolatory odpowiadają za przesunięcie grupy, jej obrót lub inna akcje. Nie są jednak w stanie określić ani czasu jaki jest przeznaczony na dokonanie obrotu, ani który ze stanów pośrednich powinien być wyświetlony w danym momencie. Dlatego też utworzono klasę `Alpha`, której celem jest określenie czasu, rodzaju i dynamiki animacji. Obiekty tej klasy wykonywane są

cyklicznie, za ilość powtórzeń odpowiada metoda `setLoopCount(int)`. Wartość parametru typu `int` jaką może przyjąć ta funkcja określa ilość cykli które zostaną wykonane. Wprowadzenie wartości -1 spowoduje wykonanie nieskończonej ilości pętli. Czas trwania jednego cyklu określi suma czterech parametrów: `increasingAlphaDuration` (czas zwiększania Alphy), `alphaAtOneDuration` (przerwa pomiędzy zwiększaniem a zmniejszaniem), `decreasingAlphaDuration` (czas zmniejszania Alphy), `alphaAtZeroDuration` (przerwa pomiędzy zmniejszaniem a zwiększaniem). Informacja o zaawansowaniu danego cyklu można uzyskać wywołując metodę `value()`. Zwróci ona wartość typu `float` z przedziału [0, 1], która będzie prezentować procentowy postęp cyklu. Istnieją trzy tryby pracy klasy `Alpha` określane metodą `setMode(int)`. Parametr może wówczas przyjąć wartości INCREASING_ENABLE, DECTISING_ENABLE lub sumy tych dwóch parametrów. W zależności od wybranego trybu wartości liczone będą od 0 do 1 gdy parametrem będzie tylko włączenie zwiększania, od 1 do 0 gdy włączone zostanie tylko zmniejszanie lub od 0 przez 1 do 0 gdy parametrem będzie suma wcześniej wymienionych parametrów. Istnieje również możliwość wpływu na dynamikę zmian. W tym celu należy użyć metod `setIncreasingAlphaRampDuration(int)` lub `setDecreasingAlphaRampDuration(int)`, które umożliwią odpowiednio zwiększanie prędkości animacji przez czas określony w parametrze tych metod. Należy jednak zwrócić uwagę na fakt iż przyspieszenie animacji podczas zwiększania pól klasy `Alpha` wymusi powrót do wejściowej prędkości czasu przed upływem czasu zwiększania. Dlatego też przyspieszanie dłuższe niż połowa czasu zwiększania pól klasy `Alpha` może zaowocować niepokojącymi efektami np.: niechcianymi cofnięciami animacji. Schemat działania opisanych metod został przedstawiony na *Ilus. 6*.



Ilus. 6

W oparciu o powyższy rozdział utworzono schemat aplikacji przedstawiony na *Ilus. 7*, a następnie został zaimplementowany w *Prog. 9* i *Prog. 10*.



Ilus. 7

```
import java.awt.*;
import java.awt.event.*;

import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;

import javax.media.j3d.*;
import javax.vecmath.*;

public class J3DAppl extends Frame{

    public J3DAppl(){
        Canvas3D c3d = new Canvas3D(null);
            SimpleUniverse su = new SimpleUniverse(c3d);
                BranchGroup objRoot = new BranchGroup();
                    TransformGroup tg = new TransformGroup();
                        TransformGroup objScale1 = new TransformGroup();
                            Transform3D trans1 = new Transform3D();
                            trans1.setScale(0.2);
                            trans1.setTranslation(
                                new Vector3d(0, 0, 0)
                            );

                        objScale1.setTransform(trans1);

                        objScale1.addChild(
                            new ColorCube()
                        );
```

```
                tg.setCapability(
                    TransformGroup.ALLOW_TRANSFORM_READ
                );
                tg.setCapability(
                    TransformGroup.ALLOW_TRANSFORM_WRITE
                );

            tg.addChild(objScale1);

            Alpha upRamp = new Alpha( );
            upRamp.setMode(Alpha.INCREASING_ENABLE);
            upRamp.setPhaseDelayDuration(1000);
            upRamp.setIncreasingAlphaDuration( 5000 );
            upRamp.setIncreasingAlphaRampDuration(2500);
            upRamp.setAlphaAtOneDuration(1000);
            upRamp.setDecreasingAlphaDuration(5000);
            upRamp.setDecreasingAlphaRampDuration(2500);
            upRamp.setLoopCount( 1 );


            PositionInterpolator pi =
                new PositionInterpolator( upRamp, tg);

            pi.setAxisOfTranslation(new Transform3D());
            pi.setEndPosition(1.0f);
            pi.setStartPosition(-1.0f);

            pi.setSchedulingBounds(
                new BoundingSphere(
                    new Point3d(0.0, 0.0, 0.0),
                    100
                )
            );

            tg.addChild(pi);

        objRoot.addChild(tg);
        objRoot.compile();
    su.addBranchGraph(objRoot);
    su.getViewingPlatform().setNominalViewingTransform();
this.setSize( 400, 400);
this.add("Center", c3d);
this.addWindowListener(
    new WindowAdapter() {
        public void windowClosing(WindowEvent evt)
        {
```

```
                    System.exit(0);
                }
            }
        );
        this.setVisible(true);

    }

    public static void main(String[] args){
        new J3DAppl();
    }
}
```

Prog. 9

Poniższy program wykorzystuje bardziej skomplikowany obiekt transformacji.

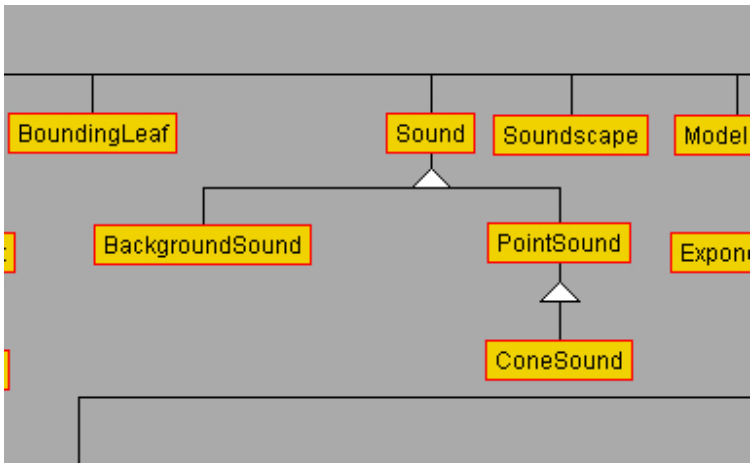
```
import java.awt.*;
import java.awt.event.*;

import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;

import javax.media.j3d.*;
import javax.vecmath.*;

public class J3DAppl extends Frame{

    public J3DAppl(){
        Canvas3D c3d = new Canvas3D(null);
            SimpleUniverse su = new SimpleUniverse(c3d);
                BranchGroup objRoot = new BranchGroup();
                    TransformGroup tg = new TransformGroup();
                        TransformGroup objScale1 = new TransformGroup();
                            Transform3D trans1 = new Transform3D();
                            trans1.setScale(0.2);
                            trans1.setTranslation(
                                new Vector3d(0, 0, 0)
                            );

                        objScale1.setTransform(trans1);

                        objScale1.addChild(
                            new ColorCube()
                        );
```

```
    objScale1.setCapability(
        TransformGroup.ALLOW_TRANSFORM_READ
    );
    objScale1.setCapability(
        TransformGroup.ALLOW_TRANSFORM_WRITE
    );

tg.addChild(objScale1);

Alpha upRamp = new Alpha( );
upRamp.setMode(Alpha.INCREASING_ENABLE);
upRamp.setPhaseDelayDuration(1000);
upRamp.setIncreasingAlphaDuration( 5000 );
upRamp.setIncreasingAlphaRampDuration(2500);
upRamp.setAlphaAtOneDuration(1000);
upRamp.setDecreasingAlphaDuration(5000);
upRamp.setDecreasingAlphaRampDuration(2500);
upRamp.setLoopCount( 1 );

float[] procenty = {
    0.0f,
    0.5f,
    1.0f
};

Quat4f [] katy = {
    new Quat4f(0.0f, 0.0f, 0.0f, 0.5f),
    new Quat4f(0.0f, 1.0f, 0.0f, 0.0f),
    new Quat4f(0.0f, 0.0f, 1.0f, 0.5f)
};

Point3f [] pozycje = {
    new Point3f( -10.0f, 0.0f, -50.0f),
    new Point3f( 0.0f, 0.0f,  -50.0f),
    new Point3f( 10.0f, 0.0f,  -50.0f)
};

RotPosPathInterpolator rppi =
    new RotPosPathInterpolator(
        upRamp,
        objScale1,
        new Transform3D(),
        procenty,
        katy,
        pozycje
    );
```

```
                    rppi.setSchedulingBounds(
                        new BoundingSphere(
                            new Point3d(0,0,0),
                            100
                        )
                    );

                    tg.addChild(rppi);

                objRoot.addChild(tg);
                objRoot.compile();
            su.addBranchGraph(objRoot);
            su.getViewingPlatform().setNominalViewingTransform();
        this.setSize( 400, 400);
        this.add("Center", c3d);
        this.addWindowListener(
            new WindowAdapter() {
                public void windowClosing(WindowEvent evt)
                {
                    System.exit(0);
                }
            }
        );
        this.setVisible(true);

        (new Tom(upRamp)).start();
    }

    public static void main(String[] args){
        new J3DAppl();
    }
}
```

Prog. 10

7. Dźwięk



Proces tworzenia sceny, w której będzie aktywny dźwięk, można podzielić na trzy etapy:

- fabrykacja obiektu urządzenia odtwarzającego dźwięk
- załadowanie dźwięku
- określenie rodzaju dźwięku jaki będzie wykorzystywany w scenie
- ustawienie zasięgu tego dźwięku

Wszystkie z tych etapów są niezbędne aby usłyszeć w głośnikach dźwięk. Fabrykacja obiektu urządzenia odtwarzającego dźwięk sprowadza się do fabrykacji obiektu klasy `AudioDevice`. Następnie dzięki metodom tej klasy można ustalić jakiego rodzaju głośniki są wykorzystywane: stereo, mono czy słuchawki. Ponadto można dokładnie określić odległość odbiorcy od głośników. Wszystko to aby jak najlepiej dostosować warunki zewnętrzne panujące w pomieszczeniu do sprzętu jakim dysponuje użytkownik.

Java3D jest przystosowana do odtwarzania dźwięków w trzech formatach:

- .wav - jest to standardowy format systemu Windows
- .au - standard opracowany przez firmę SUN
- .aif - standard dla różnych platform

Referencje do plików dźwiękowych o powyższych formatach są przechowywane w obiektach klasy `MediaConteiner`.

Kolejnym punktem jest określenie jakiego źródła dźwięku będzie wykorzystywany w scenie. Do dyspozycji mamy trzy rodzaje źródeł:

- `BackgroundSound` – dźwięk w tle, dochodzi zewsząd, wykorzystuje się go przede wszystkim do stworzenia nastroju w cenie np. jeśli scena przedstawia stolik w kawiarni, to niewątpliwie w tle powinna grać jakaś romantyczna melodia
- `PointSound` – dźwięk którego źródło znajduje się w ściśle określonym punkcie Dźwięk wyemitowany z tego punktu będzie rozchodził się we wszystkich kierunkach, w miarę oddalania się od źródła intensywność będzie malała. Właściwość te można wykorzystać tworząc scenę w której do użytkownika dobiegają różne odgłosy w zależności od kierunku w którym porusza się użytkownik niektóre dźwięki zanikają a intensywność innych wzrasta.

- `ConeSound` – dźwięk ukierunkowany, źródło tego dźwięku znajduje się w określonym punkcie ale rozchodzi się tylko kierunku który zdefiniuje użytkownik. Ponadto konieczne jest określenie kąta z jakim będzie następowało rozszerzanie dźwięku.

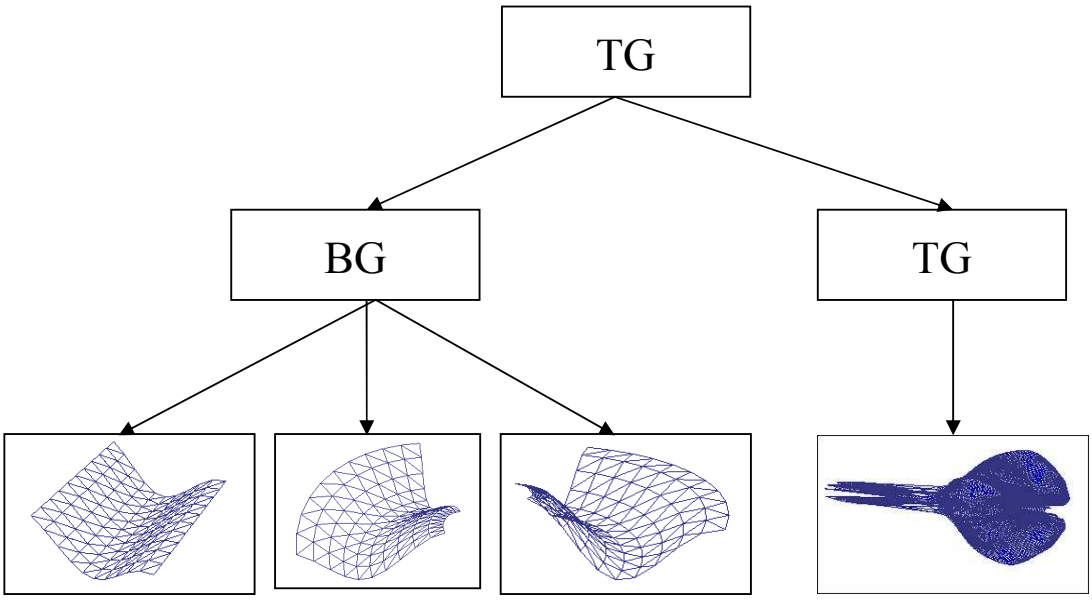
```
AudioDevice au = new AudioDevice();

MediaContainer mc = new MediaContainer("file:/E:/1.wav");
    mc.setCacheEnable(true);

TransformGroup sound = new TransformGroup();
    BackgroundSound bs = new BackgroundSound();
        bs.setSoundData(mc);
        bs.setEnable(true);
        bs.setInitialGain(1.0f);
        bs.setLoop(-1);
        bs.setSchedulingBounds(
            new BoundingSphere(
                new Point3d(0.0, 0.0, 0.0),
                1000.0
            )
        );
sound.addChild(bs);
```

Prog. 11

Projekt

Celem projektu było pokazanie możliwości pakietu Java3D. Uznano, iż najlepszym sposobem ich zaprezentowania będzie utworzenie trójwymiarowej, interakcyjnej gry. Fabuła gry została oparta na wyścigu, zadaniem gracza jest pokonanie jak największej odległości bez uszkodzenia swego pojazdu. Całość projektu powinna zostać zaimplementowana przy wykorzystaniu języka Java.

Przed rozpoczęciem pracy nad aplikacją było konieczne stworzenie modelu ideowego gry. Określono iż podmiotem całej gry będzie statek kosmiczny. Obiekt ten został umieszczony w klasie `Ship.java`. Ponieważ Java3D jest pakietem pozbawionym jakichkolwiek narzędzi do tworzenia grafiki 3D dlatego też kształt całego statku powstał w programie 3D Studio MAX. Następnie, dzięki odpowiedniemu programowi, został załadowany do sceny w aplikacji. Określono, iż statek może zostać uszkodzony w wyniku zderzenia, np. ze ścianą tunelu lub inną przeszkodą. Dlatego też konieczne było utworzenie klasy nadzorującej takie zdarzenia. Nazwano ją `ShipCollisionDetector` i umieszczono w jednej grupie z klasą `Ship`. Druga gałąź drzewa opisuje tunel w którym będzie poruszać się gracz. Nie mógł on być prostą rurą gdyż gracz po prostu nie widziałby dokąd leci. Ponadto zachodziła uzasadniona obawa iż wprawny gracz będzie grał bardzo długi okres czasu, powstanie wiec problem skończoności toru i rozmiaru jaki może zająć on w pamięci. Podjęto zatem decyzję, iż w każdym momencie gry znane będą tylko trzy elementy toru – fragment w którym aktualnie przebywa statek oraz dwa kolejne. Usunięcie jednego elementu będzie się wiązało z utworzeniem nowego. Kolejna decyzja dotyczyła elementów składowych toru. Każdy tor powinien składać się przynajmniej z trzech elementów: odcinka prostego, zakrętu w lewo oraz zakrętu w prawo. Ponieważ jednak gracz ma trójwymiarowy świat to mógłby również robić beczki fikołki itp. Na przeszkodzie stoi jednak widoczność toru i samego statku. Ostatecznie wszystkie opisane komponenty zebrano i przedstawiono na rys *Rys. 18*.



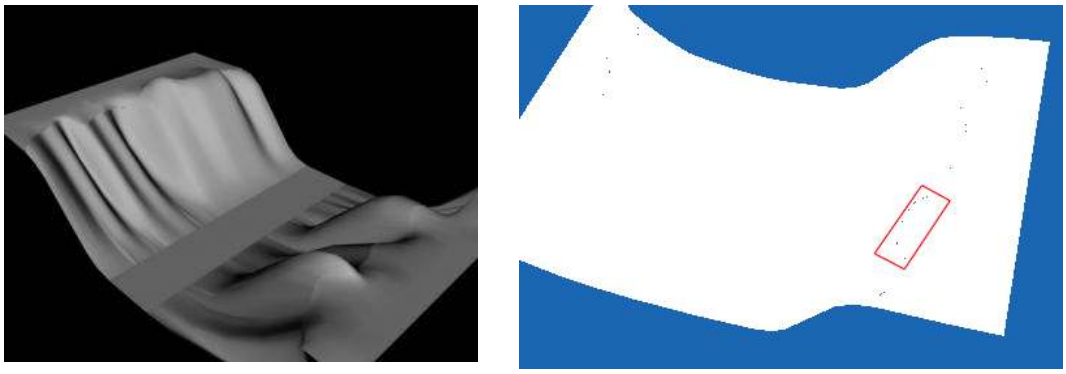
Rys. 18

8. Analiza toru

Na wstępie odrzucono tor w postaci rury, w związku z tym powstały dwa konkurencyjne rozwiązania:

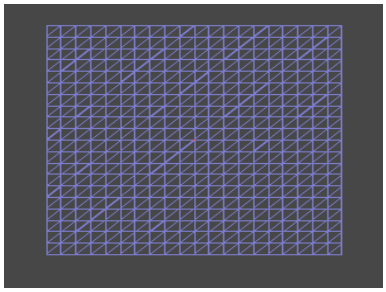
- identyczną techniką jak w przypadku statku utworzyć kanion a następnie ładować trzy prefabrykowane elementy
- zaimplementować własny algorytm kanionu który na bieżąco będzie generował potrzebne elementy

Na podstawie o pierwszy z tych punktów powstał kanion widoczny na *Rys. 19*.



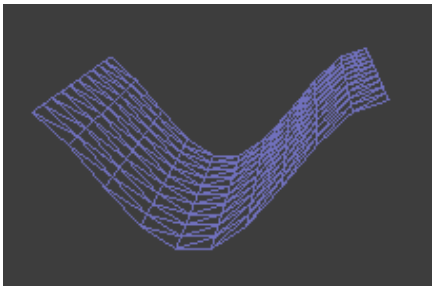
Rys. 19

Załadowanie tego rysunku trwało ok. 1 minuty. Składał się on z 9913 wieloboków i 5589 vertikali. Pomimo złożoności jaką prezentuje ten obiekt widoczne są niedokładności i zaokrąglenia w dokładności obliczeń (czerwona ramka) . Czas ładowania i wygląd końcowy tego obiektu przyczyniły się od przesunięcia tego punktu na dalszą płaszczyznę. Alternatywą było utworzenie własnego kanionu na podstawie jednego z elementów strukturujących. Wybrano `QuadArray` i utworzono płaszczyznę którą przedstawiono na *Rys. 20*.



Rys. 20

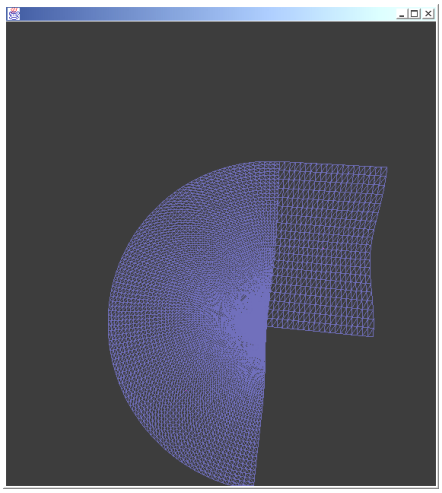
Utworzenie własnej płaszczyzny okazało się znacznie mniej kosztowne z punktu widzenia nakładów obliczeniowych, czasu procesora oraz ilości pamięci. Do zalet tak utworzonej płaszczyzny należy doliczyć elastyczność, gdyż na poziomie programu określa się długość oraz liczbę segmentów jaka jest potrzebna do konkretnych zastosowań. Nierozwiązanym problemem pozostał kształt kanionu. Początkowo rozważano możliwość utworzenia ścian które pod kątem prostym łączyłyby się z podłożem. Rozwiązanie to miałoby niewątpliwie wiele zalet, ale i jedną bardzo poważną wadę - było zbyt prymitywne. Dlatego też pomyślano na wykorzystaniem funkcji matematycznych do opisania tej powierzchni.



Rys. 21

Najbliższą funkcją która przypominała swoim kształtem kanion był fragment sinusoidy. Wynikiem była płaszczyzna widoczna na *Rys. 21*. Kolejnym krokiem było utworzenie płaszczyzn wygiętych w lewo i w prawo.

Przy okazji tworzenia, i sprawdzania wyglądu zakrętów utworzono program, którego celem było przetestowanie „z bezpiecznej odległości" jak będzie wyglądać w praktyce idea generowania tunelu.



Rys. 22

Generowanie tunelu przebiegało bez większych problemów, natomiast usuwanie niepotrzebnych już jego kawałków było znacznie bardziej skomplikowane. Początkowo program działał bez zarzutu, ale po ok. 20 do 30 wyświetlonych elementach powstawał błąd polegający na próbie usunięcia komponentu. Ponieważ wszystkie elementy toru znajdowały się w grupie `OrderedGroup` dlatego wymuszane było renderowanie wszystkich komponentów w kolejności ich włożenia. Jeżeli usunięty został zerowy element grupy to próba renderingu tego elementu kończyła się fiaskiem. Dlatego też zaimplementowano obejście: skoro ważna jest kolejność renderingu to dlaczego nie podmienić by elementu o położonego na pozycji 0 nowo utworzonym elementem. Wszelkie odwołania do starego elementu zostaną usunięte a gdy zostanie uruchomiony garbagecollector usunięty zostanie stary element. Implementację tego rozwiązania przedstawia *Prog. 12*.

```
import javax.media.j3d.*;
import javax.vecmath.*;

import com.sun.j3d.utils.geometry.*;

public class Generowanie extends Thread{
```

```
OrderedGroup og;
double inc = 0.0;
public Generowanie(OrderedGroup og){
    this.og = og;
}

public void run(){
    Vector3d v3d = new Vector3d(0.0, 0.0, 0.0);
    AxisAngle4d a4d = new AxisAngle4d(0.0, 1.0, 0.0, 0.0);
    BranchGroup temp[] = new BranchGroup[4];
    int kierunek = 0,
        next = (int)(Math.random()*2.99);

    for(int i=0; i<500; i++){

        BranchGroup bg = new BranchGroup();
            TransformGroup tmp;
                Transform3D t3d;
        switch( next ){

            case 0:
                switch(kierunek){
                    case 0:
                        v3d.z -= Math.PI/2;
                        break;
                    case 1:
                        v3d.x += Math.PI/2;
                        break;
                    case 2:
                        v3d.z += Math.PI/2;
                        break;
                    case 3:
                        v3d.x -= Math.PI/2;
                        break;
                }
                bg = new BranchGroup();
                bg.setCapability(
                    BranchGroup.ALLOW_DETACH
                );
                    tmp = new TransformGroup();
                        t3d = new Transform3D();
                            t3d.setTranslation(
                                v3d
                            );
                            t3d.setRotation(
                                a4d
```

```
                );
            tmp.setTransform(t3d);

            tmp.addChild(
                new Plaszczyzna1(0.0)
            );
            TransformGroup tg1 = new TransformGroup();
                Transform3D t3d1 = new Transform3D();
                t3d1.setScale(0.01);
                tg1.setTransform(
                    t3d1
                );
                tg1.addChild(
                    new ColorCube()
                );
            tmp.addChild(tg1);

        switch(kierunek){
            case 0:
                v3d.z -= Math.PI/2;
                break;
            case 1:
                v3d.x += Math.PI/2;
                break;
            case 2:
                v3d.z += Math.PI/2;
                break;
            case 3:
                v3d.x -= Math.PI/2;
                break;
        }

        bg.addChild(tmp);
        break;

    case 1:

        switch(kierunek){
            case 0:
                v3d.z -= 2.5;
                break;
            case 1:
                v3d.x += 2.5;
                break;
            case 2:
                v3d.z += 2.5;
```

```
        break;
    case 3:
        v3d.x -= 2.5;
        break;
}

bg = new BranchGroup();
bg.setCapability(
    BranchGroup.ALLOW_DETACH
);

    tmp = new TransformGroup();
        t3d = new Transform3D();
            t3d.setTranslation(
                v3d
            );
            t3d.setRotation(
                a4d
            );
    tmp.setTransform(t3d);

    tmp.addChild(
        new PlaszczyznaZ1(0.0, 0.0)
    );
    tg1 = new TransformGroup();
        t3d1 = new Transform3D();
        t3d1.setScale(0.01);
        tg1.setTransform(
            t3d1
        );
        tg1.addChild(
            new ColorCube()
        );
    tmp.addChild(tg1);
a4d.angle -= Math.PI/2;

switch(kierunek){
    case 0:
        v3d.x += 2.5;
        kierunek = 1;
        break;
    case 1:
        v3d.z += 2.5;
        kierunek = 2;
        break;
    case 2:
```

```
            v3d.x -= 2.5;
            kierunek = 3;
            break;
        case 3:
            v3d.z -= 2.5;
            kierunek = 0;
            break;
    }
    bg.addChild(tmp);
    break;

case 2:

    switch(kierunek){
        case 0:
            v3d.z -= 2.5;
            break;
        case 1:
            v3d.x += 2.5;
            break;
        case 2:
            v3d.z += 2.5;
            break;
        case 3:
            v3d.x -= 2.5;
            break;
    }

    bg = new BranchGroup();
    bg.setCapability(
        BranchGroup.ALLOW_DETACH
    );

        tmp = new TransformGroup();
            t3d = new Transform3D();
                t3d.setTranslation(
                    v3d
                );
                t3d.setRotation(
                    a4d
                );
        tmp.setTransform(t3d);

        tmp.addChild(
            new PlaszczyznaZ2(0.0, 0.0)
        );
```

```
            tg1 = new TransformGroup();
                t3d1 = new Transform3D();
                t3d1.setScale(0.01);
                tg1.setTransform(
                    t3d1
                );
                tg1.addChild(
                    new ColorCube()
                );
            tmp.addChild(tg1);
        a4d.angle += Math.PI/2;

        switch(kierunek){
            case 0:
                v3d.x -= 2.5;
                kierunek = 3;
                break;
            case 1:
                v3d.z -= 2.5;
                kierunek = 0;
            case 2:
                v3d.x -= 2.5;
                kierunek = 1;
                break;
            case 3:
                v3d.z -= 2.5;
                kierunek = 2;
                break;
        }
        bg.addChild(tmp);
        break;

}

System.out.println(next);

int tab1[] = {0,1,2};

switch(next){

    case 0:
        next = tab1[(int)(Math.random()*2)];
        break;
    case 1:
        next = tab1[(int)(Math.random()*2)];
        break;
```

```
                case 2:
                    next = tab1[(int)(Math.random()*2)];
                    break;
            }


            try{
               Thread.sleep(1000);
            }catch(Exception ex){
            }

            temp[3] = bg;

            if(i>2){
                for(int j=0; j<og.numChildren();j++)
                    if( temp[0] == og.getChild(j)){
                        og.setChild(bg, j);
                    }
            }else{
                og.addChild(bg);
            }

            for(int j=0; j<3; j++){
                temp[j] = temp[j+1];
            }

        }


    }
}
```

Prog. 12

9. Obsługa zdarzeń

Problem drugi wiązał się z procesem wykrywania, czy dany obiekt zderza się z innym obiektem czy tez nie. Proces takiej detekcji realizowany jest przy wykorzystaniu klasy pochodnej od `Behavior`. Obsługa samego zdarzenia pozostaje w rękach programisty. Zasada działania jest prosta, jednak jeśli chodzi o praktykę jest znacznie gorzej. Otóż jako parametr wyjściowy można podać dowolną klasę dziedziczącą po `Node` np. `TransformGroup`. Jednak nie potrafimy określić rozmiaru tej grupy, a co za tym idzie nie można określić w którym miejscu nastąpi przecięcie się obiektów. Dlatego też jako parametr obiektu `ShipCollisionDetector` należy podać obiekt klasy Shape3D (statek). Takie zastosowanie ograniczy nam kolizje do obrębu pojedynczego kształtu. Niestety jak pokazała praktyka program, który importował obiekty z pliku o rozszerzeniu 3DS, wykorzystywał grupy klas `SharedGroup`, które ze względu na swoją nieokreśloność (w każdej chwili zawartość takiej grupy może ulec zmianie) , nie mogą określać kolizji obiektów. W poszukiwaniu rozwiązania tego problemu przetestowano kilka innych programów. Niestety żaden z nich nie spełnił pokładanych w nim nadziei. W zaistniałej sytuacji podjęto decyzję o napisaniu dedykowanego programu do konwersji plików *.3DS na obiekty pochodne od klasy Shape3D. Szybka analiza problemu ujawniła złożoność tego procesu: nieznany format pliku, zapisany binarnie skutecznie utrudniał pracę. Okazało się jednak iż 3D Studio może wyeksportować poszczególne komponenty statku do pliku ASCII. W wyniku tej operacji szybko powstał prosty i skuteczny konwerter *Prog. 13*.

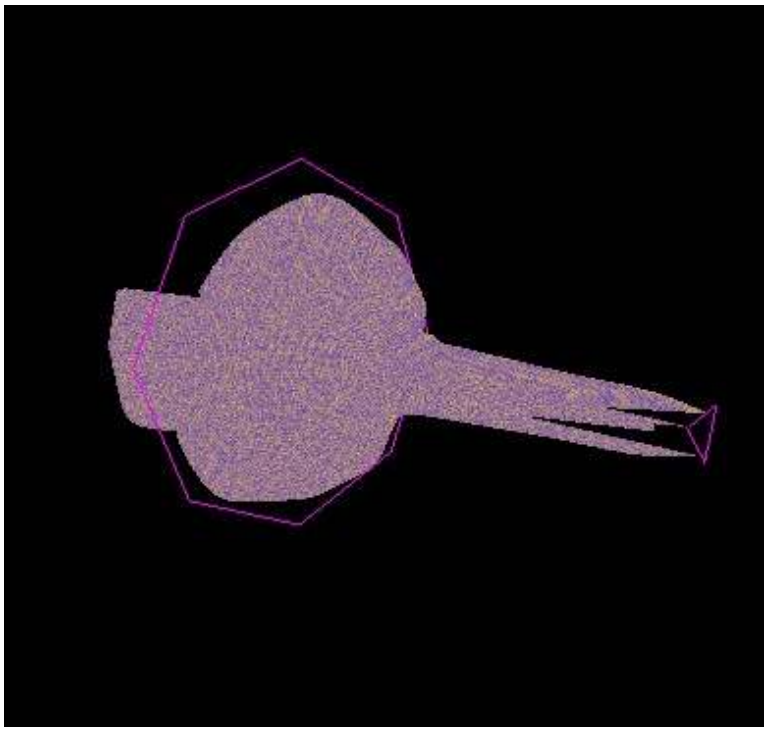
```
import javax.media.j3d.*;
import javax.vecmath.*;
import java.io.*;
import java.util.*;
public class Converter extends Shape3D{

    Converter(){
        Point3f[] tab = null;
        Point3f[] points = null;
        Color3f[] colors = null;

        try{
            FileReader fr = new FileReader("korp.ase");
            BufferedReader br = new BufferedReader(fr);

            String line = "";

            while((line = br.readLine()) != null){
                StringTokenizer st = new StringTokenizer(line);

                while(st.hasMoreTokens()){
                    String token = st.nextToken();

                    if( token.compareTo("*MESH_NUMVERTEX")==0){
                        tab = new Point3f[(new Integer(st.nextToken())).intValue() ];
                    }

                    if(token.compareTo("*MESH_VERTEX")==0){
```

```
                        tab[Integer.parseInt(st.nextToken())] = new Point3f(
                            (new Float(st.nextToken())).floatValue(),
                            (new Float(st.nextToken())).floatValue(),
                            (new Float(st.nextToken())).floatValue()
                        );
                    }

                    if(token.compareTo("*MESH_NUMFACES")==0){
                        points = new Point3f[(new Integer(st.nextToken())).intValue()*3 ];
                        colors = new Color3f[points.length];
                    }

                    if(token.compareTo("*MESH_FACE")==0){
                        String num = st.nextToken();
                        num = num.substring(
                            0,
                            num.length()-1
                        );
                        int i = Integer.parseInt(num);
                        st.nextToken();
                        points[(i*3)+0] = tab[Integer.parseInt(st.nextToken())];
                        colors[(i*3)+0] = new Color3f(1.0f, 1.0f, 0.0f);
                        st.nextToken();
                        points[(i*3)+1] = tab[Integer.parseInt(st.nextToken())];
                        colors[(i*3)+1] = new Color3f(1.0f, 1.0f, 0.0f);
                        st.nextToken();
                        points[(i*3)+2] = tab[Integer.parseInt(st.nextToken())];
                        colors[(i*3)+2] = new Color3f(1.0f, 1.0f, 0.0f);
                    }
                }
            }
        }catch(Exception ex){
            System.out.println(ex);
        }

        TriangleArray ta = new TriangleArray(points.length, GeometryArray.COORDINATES |
GeometryArray.COLOR_3);
        ta.setCoordinates(0, points);
        ta.setColors(0, colors);
        this.setGeometry(ta);
    }

}
```

Prog. 13

Jak na ironię pojawił się kolejny problem. Liczba punktów w statku wynosiła 1300. Gdy komputer przystąpił do analizy, czy nie następuje zderzenie, w każdym takcie musiał sprawdzić, czy żaden z

punktów nie koliduje z punktami płaszczyzny. W efekcie procesor AMD Tunderbird 900 MHz nie nadążał z przetwarzaniem danych. W tej sytuacji nie było innego wyjścia, jak uprościć budowę całego statku. Konieczne było jednak poznanie pułapu liczby punktów pozwalającej na sensowne funkcjonowanie programu. Test przeprowadzono kolejno dla 1000, 500, 250, 100 i 50 punktów, na których był rozpięty kształt. Program zaczął działać poprawnie dopiero przy 50 punktach. Niestety taka liczba punktów absolutnie nie wystarczała do opisania „ładnego" statku, dlatego też rozpoczęto poszukiwanie rozwiązania jednocześnie taniego i efektywnego. Najbardziej efektywne okazało się dołączenie do statku obiektów o bardzo malej ilości węzłów, przez co statek zyskał jak gdyby tarcze które pokazane są na *Rys. 23*.



Rys. 23

Zakończenie

Pakiet Java3D niewątpliwie jest bardzo ciekawym przedsięwzięciem programistycznym. Jego klasy i metody umożliwiają tworzenie animacji i interaktywnych scen trójwymiarowych. Pakiet został zoptymalizowany pod kątem dwóch najpowszechniejszych standardów: OpenGL i DirectX. Dlatego też wykorzystanie go w komputerach pozbawionych sprzętowej akceleracji trójwymiarowej grafiki wymusza zastosowanie nadzwyczaj szybkich procesorów, które będą w stanie jednocześnie przetwarzać grafikę i obsługiwać maszynę wirtualną Javy.

Przeprowadzone doświadczenia pokazały, że procesor AMD 900 MHz nie wystarcza dla wyświetlania bardziej skomplikowanych scen. Jednak ponieważ produkowane obecnie karty graficzne wyposażone są w układy implementujące przynajmniej jeden z wymienionych standardów, w związku z tym problem został uznany za drugorzędny.

Znacznie większe obawy budzi nie do końca przemyślana implementacja metody wykrywania kolizji zawarta w pakiecie. Dlaczego podczas sprawdzania, czy dwie siatki przecinają się, obciążenie procesora wzrasta do 100%? Problem zapewne powiązany jest z obliczeniem czy wszystkie krawędzie jednej płaszczyzny nie przecinają się z krawędziami innej płaszczyzny. Uważam jednak, że można ograniczyć liczbę sprawdzeń co niewątpliwie zaowocować powinno zwiększeniem szybkości działania.

Pomimo tych mankamentów uważam, iż Java3D jest wartościowym produktem, który bez wątpienia znajdzie licznych zwolenników. Niewątpliwie już wkrótce powstaną graficzne środowiska, które w znaczny sposób uproszczą tworzenie wirtualnych światów, zaś możliwości definiowania własnych brył i efektów zachęcą programistów do tworzenia nowych komponentów.

Należy również wspomnieć, iż Java3D jest przygotowana do obsługiwania hełmów i okularów 3D. Tak więc być może już w niedalekiej przyszłości zaistnieje możliwość czytania książek w wirtualnej bibliotece?

Wymienione w ostatnich dwóch paragrafach, potencjalne możliwości pakietu nie doczekały się jak dotąd implementacji. Dlatego też uważam iż jest to doskonały materiał dla dalszych prac badawczych.

Środowisko

Dodatek 1: Wymagane elementy

Pakiet klas Java3D wymaga zainstalowania dodatkowego oprogramowania niezależnie od tego czy użytkownik tylko ogląda applet w przeglądarce, czy też chce tworzyć własne wirtualne światy. W pierwszym z wymienionych przypadków należy zainstalować produkt Java PlugIn. W drugim jest wymagane zainstalowanie większej liczby dodatków. Wymagany jest :

- kompilator
- Java Runtime Environment (JRE)
- Java3D
- w pewnych przypadkach może być potrzebny OpenGL
- HTML Converter - jeżeli użytkownik chce tworzyć applety

Pełna instalacja kompilatora Java Development Kit, potocznie nazywanego JDK, zainstaluje jednocześnie również drugi z wymaganych składników tj. JRE. W przypadku gdy używany jest inny kompilator należy dodatkowo zainstalować JRE. Po ukończeniu tego procesu należy upewnić się, że program instalacyjny zmodyfikował plik *autoexec.bat* dodając ścieżkę dostępu do wybranego przez użytkownika katalogu instalacyjnego.

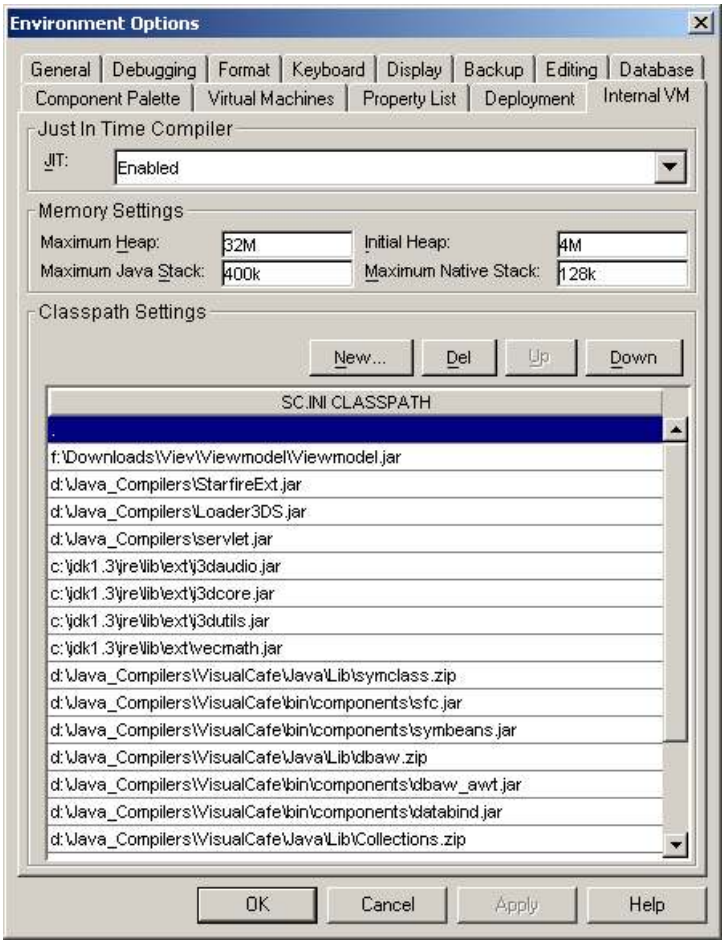
Podczas instalacji pakietu klas Java3D należy zwrócić uwagę na to, aby katalog w którym ma być zainstalowany pakiet był tym samym katalogiem, w którym znajduje się już JRE. Dzięki temu unika się niepotrzebnego dublowania plików.

Jeżeli używany jest inny produkt niż JDK należy pamiętać, aby zbiory klas: *j3daudio*, *j3dcore*, *j3dutils* i *vecmath* zostały dopisane w opcjach konfiguracyjnych kompilatora.

Dodatek 2: Ustawienia Symantec Cafe 4.0

Niezbędne zmiany zostaną omówione na przykładzie kompilatora *Visual Cafe 4.0* firmy Symantec. Aby program mógł zostać skompilowany poprawnie, należy:

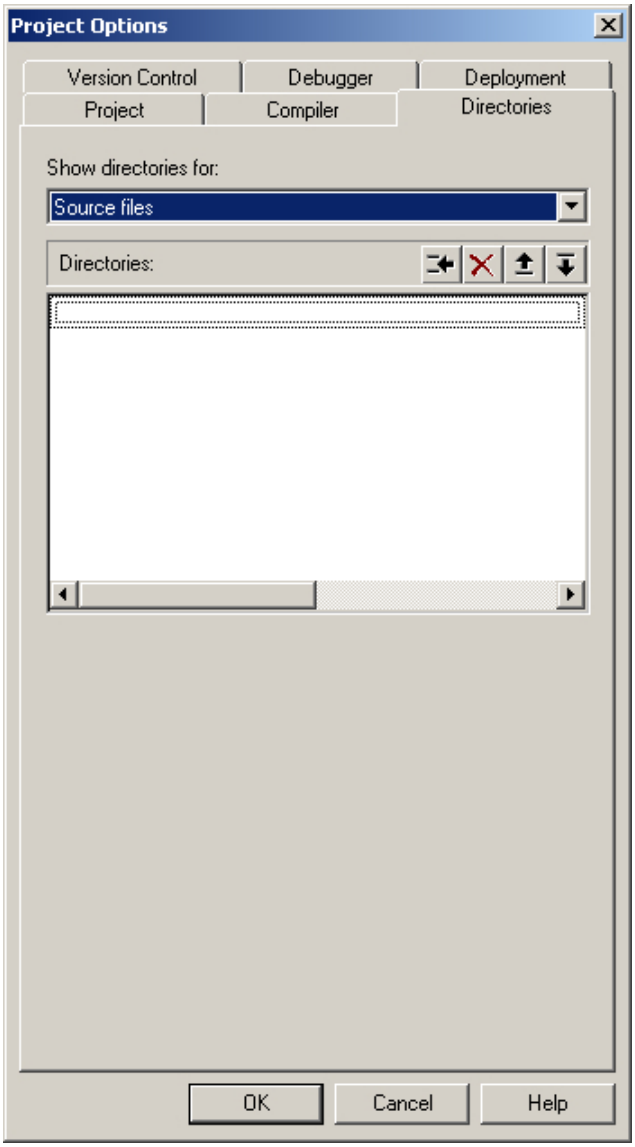
1. Z menu *Tools* wybrać opcję *Environment Options* ...
2. W oknie *Environment Options* wybrać zakładkę *Internal VM*
3. Po naciśnięciu przycisku *New*... wybrać jeden z wyżej wymienionych pakietów klas.
4. Po zatwierdzeniu dokonanego wyboru powtarzać krok 3 aż do wyczerpania klas.



Rys. 24

W celu rozszerzenia kontekstowej listy metod, które mogą zastać użyte należy:

1. Z menu *Project* należy wybrać opcję *Options ...*
2. W otwartym oknie wybrać zakładkę *Directories*
3. W rozwijanym menu pod nazwą *Show Directories* należy wybrać *Source File*
4. Następnie klikając przycisk *New* wprowadzić katalog gdzie znajdują się źródła klas Java3D.



Rys. 25

Powyższa metoda pozwala modyfikować listę metod jedynie dla aktualnie otwartego projektu - w innych nowo tworzonych projektach dodane metody nadal nie będą dostępne.

Aby uniknąć tej niedogodności należy utworzyć pusty szkielet appletu, aplikacji lub appleto-aplikacji i na jego bazie tworzyć nowe projekty. Najbardziej użytecznym z punktu widzenia zarówno programisty jak i użytkownika jest ostatni szkielet. Dlatego też został on przedstawiony jako *Kod. 9*.

```
import com.sun.j3d.utils.geometry.*;

import java.applet.*;
import java.awt.BorderLayout;
import java.awt.event.*;

import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;
```

```
import java.io.*;
import java.util.*;

public class J3DAppl extends Applet{

  public BranchGroup createSceneGraph(Canvas3D c){
    BranchGroup objRoot = new BranchGroup();

    objRoot.compile();
    return objRoot;
  }

  public J3DAppl(){
    setLayout(new BorderLayout());
    Canvas3D c = new Canvas3D(null);
    add("Center", c);

    BranchGroup scene = createSceneGraph(c);
    SimpleUniverse u = new SimpleUniverse(c);

    u.getViewingPlatform().setNominalViewingTransform();
    u.addBranchGraph(scene);
  }

  public void init(){
    setLayout(new BorderLayout());
    Canvas3D c = new Canvas3D(null);
    add("Center", c);

    BranchGroup scene = createSceneGraph(c);
    SimpleUniverse u = new SimpleUniverse(c);

    u.getViewingPlatform().setNominalViewingTransform();
    u.addBranchGraph(scene);
  }

  public static void main(String argv[]){
    BranchGroup group;
    new MainFrame(new J3DAppl(), 250, 250);
  }
}
```

Kod. 9

Listing programu

Converter.java

```
import javax.media.j3d.*;
import javax.vecmath.*;
import java.io.*;
import java.util.*;
import com.sun.j3d.utils.image.TextureLoader;

public class Converter extends Shape3D{

    Converter(){
        Point3f[] tab = null;
        Point3f[] points = null;
        Color3f[] colors = null;

        try{
            FileReader fr = new FileReader("korp.ase");
            BufferedReader br = new BufferedReader(fr);

            String line = "";

            while((line = br.readLine()) != null){
                StringTokenizer st = new StringTokenizer(line);

                while(st.hasMoreTokens()){
                    String token = st.nextToken();

                    if( token.compareTo("*MESH_NUMVERTEX")==0){
                        tab = new Point3f[(new Integer(st.nextToken())).intValue() ];
                    }

                    if(token.compareTo("*MESH_VERTEX")==0){
                        tab[Integer.parseInt(st.nextToken())] = new Point3f(
                            (new Float(st.nextToken())).floatValue(),
                            (new Float(st.nextToken())).floatValue(),
                            (new Float(st.nextToken())).floatValue()
                        );
                    }

                    if(token.compareTo("*MESH_NUMFACES")==0){
                        points = new Point3f[(new Integer(st.nextToken())).intValue()*3 ];
                        colors = new Color3f[points.length];
                    }

                    if(token.compareTo("*MESH_FACE")==0){
                        String num = st.nextToken();
```

```
                num = num.substring(
                    0,
                    num.length()-1
                );
                int i = Integer.parseInt(num);
                st.nextToken();
                points[(i*3)+0] = tab[Integer.parseInt(st.nextToken())];
                colors[(i*3)+0] = new Color3f(1.0f, 1.0f, 0.0f);
                st.nextToken();
                points[(i*3)+1] = tab[Integer.parseInt(st.nextToken())];
                colors[(i*3)+1] = new Color3f(1.0f, 1.0f, 0.0f);
                st.nextToken();
                points[(i*3)+2] = tab[Integer.parseInt(st.nextToken())];
                colors[(i*3)+2] = new Color3f(1.0f, 1.0f, 0.0f);
            }
        }
    }
}catch(Exception ex){
    System.out.println(ex);
}

TriangleArray ta = new TriangleArray(
    points.length,
    GeometryArray.COORDINATES |
    GeometryArray.COLOR_3 |
    GeometryArray.TEXTURE_COORDINATE_3
);
ta.setCoordinates(0, points);
ta.setTextureCoordinates(0, points);
ta.setColors(0, colors);
this.setGeometry(ta);

Appearance ap = new Appearance();
    ColoringAttributes ca = new ColoringAttributes();
        ca.setShadeModel(ColoringAttributes.SHADE_FLAT );
        ca.setColor(0.0f, 1.0f, 0.0f);
        ca.setCapability(
            ColoringAttributes.ALLOW_SHADE_MODEL_WRITE
        );
    ap.setColoringAttributes(ca);

    TextureLoader tl = new TextureLoader( "tekstura.jpg", null);

    Material ma = new Material();
        ma.setLightingEnable(true);
    ap.setTexture(tl.getTexture());
```

```
            ap.setCapability(
                Appearance.ALLOW_COLORING_ATTRIBUTES_READ
            );
            ap.setCapability(
                Appearance.ALLOW_COLORING_ATTRIBUTES_WRITE
            );
        this.setAppearance(ap);

        float x = 0.0f,
              y = 0.0f,
              z = 0.0f;

        for(int i=0; i<tab.length;i++){
            if(tab[i].x < x)
                x = tab[i].x;
            if(tab[i].y < y)
                y = tab[i].y;
            if(tab[i].z < z)
                z = tab[i].z;
        }

        System.out.println(x+" "+y+" "+z);
    }

}
```

DetNextElement.java

```
import javax.media.j3d.*;
import javax.vecmath.*;
import java.io.*;
import java.util.*;
public class DetNextElement extends Shape3D{

    DetNextElement(){
        Point3f[] tab = new Point3f[6];

        Color3f[] c3f = {
            new Color3f( 1.0f, 0.0f, 1.0f),
            new Color3f( 1.0f, 0.0f, 1.0f),
            new Color3f( 1.0f, 1.0f, 0.0f),
            new Color3f( 1.0f, 1.0f, 0.0f),
            new Color3f( 1.0f, 0.0f, 1.0f),
            new Color3f( 1.0f, 0.0f, 1.0f),
        };

        tab[0] = new Point3f( 0.0f, 0.0f, 0.0f);
        tab[1] = new Point3f( 0.0f, -0.95f, 0.0f);
        tab[2] = new Point3f( 0.0f, 0.0f, 0.0f);
        tab[3] = new Point3f( -2.1f, 0.95f, 0.0f);
        tab[4] = new Point3f( 0.0f, 0.0f, 0.0f);
        tab[5] = new Point3f( 2.1f, 0.95f, 0.0f);

        int t[] ={
            2,
            2,
            2
        };
        LineStripArray pa = new LineStripArray(
             tab.length,
             GeometryArray.COORDINATES |
             GeometryArray.COLOR_3, t
        );
        pa.setCoordinates(0, tab);
        pa.setColors(0, c3f);

        this.addGeometry(pa);
    }

}
```

DetPoint.java

```
import javax.media.j3d.*;
import javax.vecmath.*;
import java.io.*;
import java.util.*;
```

```
public class DetPoint extends Shape3D{

    DetPoint(){
        Point3f[] tab = new Point3f[4+4];

        Color3f[] c3f = {
            new Color3f( 1.0f, 0.0f, 1.0f),
            new Color3f( 1.0f, 0.0f, 1.0f),
            new Color3f( 1.0f, 0.0f, 1.0f),
            new Color3f( 1.0f, 0.0f, 1.0f),
            new Color3f( 1.0f, 0.0f, 1.0f),
            new Color3f( 1.0f, 0.0f, 1.0f),
            new Color3f( 1.0f, 0.0f, 1.0f),
            new Color3f( 1.0f, 0.0f, 1.0f),
        };

        double krok = (Math.PI*2)/3;
        int j=0;

        for(double i = 0; i<=(Math.PI*2); i += krok){
            tab[j++] = new Point3f(
                (float)(Math.sin(i)*140),
                -50.0f,
                (float)(Math.cos(i)*140)
            );

            System.out.println(
                tab[j-1].x+" "+tab[j-1].y+" "+tab[j-1].z
            );
        }

        krok = (Math.PI*2)/3;

        for(double i = 0; i<=(Math.PI*2); i += krok){
            tab[j++] = new Point3f(
                (float)(Math.sin(i)*10),
                135.0f,
                (float)(Math.cos(i)*10)
            );

            System.out.println(
                tab[j-1].x+" "+tab[j-1].y+" "+tab[j-1].z
            );
        }

        int t[] ={
            4,
            4
        };
        LineStripArray pa = new LineStripArray(
```

```
            tab.length,
            GeometryArray.COORDINATES |
            GeometryArray.COLOR_3,
            t
        );
        pa.setCoordinates(0, tab);
        pa.setColors(0, c3f);

        this.addGeometry(pa);
    }

}
```

J3DAppl.java

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
import java.io.*;

import com.sun.j3d.utils.applet.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.mouse.*;
import com.sun.j3d.utils.behaviors.picking.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

import javax.media.j3d.*;
import javax.vecmath.*;
import javax.swing.*;

public class J3DAppl extends JFrame{



    public J3DAppl(){
        Canvas3D c3d = new Canvas3D(null);
            SimpleUniverse su = new SimpleUniverse(c3d);
                AudioDevice audioDev = su.getViewer().createAudioDevice();
                audioDev.setAudioPlaybackType(AudioDevice.STEREO_SPEAKERS);

                BranchGroup objRoot = new BranchGroup();
                    TransformGroup tg = new TransformGroup();
                        TransformGroup group = new TransformGroup();
                            TransformGroup ship = new TransformGroup();
                                Transform3D t3d = new Transform3D();
                                t3d.setScale(0.0025);
                                t3d.setRotation(
                                    new AxisAngle4d(
                                        1,
                                        0,
                                        0,
                                        -Math.PI/2
                                    )
                                );
                                ship.setTransform(t3d);

                                Shape3D sc = new Converter();
                            ship.addChild(sc);

                                Shape3D sp = new DetPoint();
                            ship.addChild(sp);
```

```
                MediaContainer mc[] = {
                    new MediaContainer("file:/E:/skrach.wav"),
                    new MediaContainer("file:/E:/blow.wav")
                };
                mc[0].setCacheEnable(true);


                BackgroundSound bs = new BackgroundSound();
                bs.setSoundData(mc[0]);
                bs.setLoop(-1);
                bs.setEnable(false);

                BackgroundSound bs1 = new BackgroundSound();
                bs1.setSoundData(mc[1]);
                bs1.setLoop(1);
                bs1.setEnable(false);

    ship.addChild(bs);
    ship.addChild(bs1);
            ShipCollisionDetector scd = new ShipCollisionDetector(sp, bs, bs1);
    ship.addChild(scd);
group.addChild(ship);

    TransformGroup ref = new TransformGroup();
        Transform3D rett = new Transform3D();
        rett.setTranslation(
            new Vector3d(0.0,-1.0,0.5)
        );
        ref.setTransform(rett);

        DirectionalLight sl = new DirectionalLight();
        sl.setDirection(
            new Vector3f( 0.0f, -1.0f, 0.0f)
        );

        sl.setColor(
            new Color3f(0.0f, 0.0f, 1.0f)
        );
        sl.setEnable(true);
        sl.setInfluencingBounds(
            new BoundingSphere(
                new Point3d(),
                10.0
            )
        );

    ref.addChild(sl);

    ref.addChild(
        new ColorCube(0.1)
```

```
    );

group.addChild(ref);


    TransformGroup anim = new TransformGroup();

        MyPositionInterpolator mpi = new MyPositionInterpolator(anim);
        scd.setMPI(mpi);
        Thread thread = new Thread(mpi);
        thread.start();

        TransformGroup t1 = new TransformGroup();

        t1.setCapability(
            TransformGroup.ALLOW_TRANSFORM_WRITE
        );

            TransformGroup tunel = new TransformGroup();
            tunel.setCapability(
                TransformGroup.ALLOW_TRANSFORM_WRITE
            );

                NewElement ne = new NewElement();
            tunel.addChild(
                ne
            );

        t1.addChild(
            tunel
        );
    anim.addChild(
        t1
    );
group.addChild(anim);

    MyKeyBehavior mkb = new MyKeyBehavior(
        group,
        anim,
        t1,
        tunel,
        ship,
        scd,
        mpi
    );
    mkb.setSchedulingBounds(
        new BoundingSphere(
            new Point3d(0,0,0),
            100.0
        )
```

```
            );
        group.addChild(mkb);

            AmbientLight al = new AmbientLight(
                true,
                new Color3f(0.3f, 0.3f, 0.3f)
            );

            al.setInfluencingBounds(
                new BoundingSphere(
                    new Point3d(0,0,0),
                    100.0
                )
            );
        group.addChild(al);

            ExponentialFog fog = new ExponentialFog();
            fog.setColor(
                new Color3f(
                    1.0f, 1.0f, 1.0f
                )
            );

            fog.setDensity( 2.0f );

            fog.setInfluencingBounds(
                new BoundingSphere(
                    new Point3d(),
                    100
                )
            );
        group.addChild(fog);

        Background background = new Background();
               background.setColor(
            new Color3f(
                1.0f, 1.0f, 1.0f
            )
               );
               background.setApplicationBounds(
            new BoundingSphere(
                new Point3d(),
                100
            )
               );
             tg.addChild(background);
    tg.addChild(group);
objRoot.addChild(tg);

    MouseRotate mr = new MouseRotate(tg);
```

```
            mr.setSchedulingBounds(
                new BoundingSphere(
                    new Point3d(0,0,0),
                    100.0
                )
            );

            MouseTranslate mt = new MouseTranslate(tg);
            mt.setSchedulingBounds(
                new BoundingSphere(
                    new Point3d(0,0,0),
                    100.0
                )
            );

            MouseZoom mz = new MouseZoom(tg);
            mz.setSchedulingBounds(
                new BoundingSphere(
                    new Point3d(0,0,0),
                    100.0
                )
            );
            objRoot.addChild(mr);
            objRoot.addChild(mt);
            objRoot.addChild(mz);
            tg.setCapability(
                TransformGroup.ALLOW_TRANSFORM_READ
            );
            tg.setCapability(
                TransformGroup.ALLOW_TRANSFORM_WRITE
            );

        su.getViewingPlatform().setNominalViewingTransform();
        su.addBranchGraph(objRoot);

    this.setSize( 400, 400);
    this.getContentPane().add(c3d, "Center");
    GridBagLayout gbl = new GridBagLayout();
    GridBagConstraints gbc = new GridBagConstraints();
    JPanel jp = new JPanel(gbl);

        gbc.gridwidth = 1;
        gbc.gridheight = 2;
        gbc.weighty = 1.0;
        gbl.setConstraints(mpi, gbc);
        jp.add(mpi);


        JLabel jl1 = new JLabel("Speed", new ImageIcon("sr.gif"), JLabel.RIGHT );
```

```
            gbc.gridwidth = GridBagConstraints.REMAINDER;
            gbc.gridheight = 1;
            gbl.setConstraints(jl1, gbc);
            jp.add(jl1);


            JLabel jl1a = new JLabel("");
            gbl.setConstraints(jl1a, gbc);
            jp.add(jl1a);


            JLabel jl2 = new JLabel("Shilds",new ImageIcon("sl.gif"), JLabel.LEFT );
            gbc.gridwidth = 1;
            gbc.gridheight = 1;
            gbl.setConstraints(jl2, gbc);
            jp.add(jl2);

            gbc.gridwidth = GridBagConstraints.REMAINDER;
            gbc.gridheight = 2;
            gbc.weighty = 1.0;

            gbl.setConstraints(scd.getPanel(), gbc);
            jp.add(scd.getPanel());

            jp.add(
                NewElement.jp
            );

        this.getContentPane().add(jp, BorderLayout.SOUTH);
        this.addWindowListener(
            new WindowAdapter(){
                public void windowClosing(WindowEvent evt){
                    System.exit(0);
                }
            }
        );
        this.setVisible(true);
        this.pack();
    }

    public static void main(String[] args){
        new J3DAppl();
    }
}
```

MyKeyBehavior.java

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;

import javax.media.j3d.*;
import javax.vecmath.*;

import com.sun.j3d.utils.geometry.*;

public class MyKeyBehavior extends Behavior{
    TransformGroup group;
    TransformGroup pozycjaObracania;
    TransformGroup elementObracany;
    TransformGroup elementObracany1;
    TransformGroup move;
    ShipCollisionDetector scd;
    MyPositionInterpolator mpi;

    Point3f p3f = new Point3f();

    WakeupCondition wc = new WakeupOnAWTEvent(
        KeyEvent.KEY_PRESSED
    );

    public MyKeyBehavior(
        TransformGroup group,
        TransformGroup pozycjaObracania,
        TransformGroup elementObracany,
        TransformGroup elementObracany1,
        TransformGroup move,
        ShipCollisionDetector scd,
        MyPositionInterpolator mpi
    ){

        this.group = group;
        this.pozycjaObracania = pozycjaObracania;
        this.elementObracany = elementObracany;
        this.elementObracany1 = elementObracany1;
        this.mpi = mpi;
        this.move = move;
        this.scd = scd;

        group.setCapability(
            TransformGroup.ALLOW_TRANSFORM_READ
        );
        group.setCapability(
            TransformGroup.ALLOW_TRANSFORM_WRITE
        );
```

```
    group.setCapability(
        Group.ALLOW_CHILDREN_EXTEND
    );
    pozycjaObracania.setCapability(
        TransformGroup.ALLOW_TRANSFORM_READ
    );
    pozycjaObracania.setCapability(
        TransformGroup.ALLOW_TRANSFORM_WRITE
    );
    elementObracany.setCapability(
        TransformGroup.ALLOW_TRANSFORM_READ
    );
    elementObracany.setCapability(
        TransformGroup.ALLOW_TRANSFORM_WRITE
    );
    elementObracany1.setCapability(
        TransformGroup.ALLOW_TRANSFORM_READ
    );
    elementObracany1.setCapability(
        TransformGroup.ALLOW_TRANSFORM_WRITE
    );
    move.setCapability(
        TransformGroup.ALLOW_TRANSFORM_READ
    );
    move.setCapability(
        TransformGroup.ALLOW_TRANSFORM_WRITE
    );
}

public void initialize(){
    wakeupOn(wc);
}

public void processStimulus(Enumeration enum){
    AWTEvent[] evt = null;
    while(enum.hasMoreElements()){
        WakeupCriterion wakeupInstance = (WakeupCriterion) enum.nextElement();
        if(wakeupInstance instanceof WakeupOnAWTEvent){
            evt = ((WakeupOnAWTEvent)wakeupInstance).getAWTEvent();
            processEvent( (KeyEvent)evt[0]);
        }
    }
    wakeupOn(wc);
}

double rot = 0.0;
double poz = 0.0;
int last = 0;

boolean czy = true;
```

```
public void processEvent(KeyEvent event){

    //anim
        Transform3D t3d1 = new Transform3D();
    pozycjaObracania.getTransform(t3d1);
            Vector3d v3d1 = new Vector3d();
        t3d1.get(v3d1);

        //System.out.println("Plaszczyzna anim    : "+v3d1.x+" "+v3d1.y+" "+v3d1.z);
    //t1
        Transform3D t3d2 = new Transform3D();
    elementObracany.getTransform(t3d2);
            Vector3d v3d2 = new Vector3d();
        t3d2.get(v3d2);
        //System.out.println("Plaszczyzna t1      : "+v3d2.x+" "+v3d2.y+" "+v3d2.z);
    //tunel
        Transform3D t3d3 = new Transform3D();
    elementObracany1.getTransform(t3d3);
            Vector3d v3d3 = new Vector3d();
        t3d3.get(v3d3);
        //System.out.println("Plaszczyzna tunel   : "+v3d3.x+" "+v3d3.y+" "+v3d3.z+"\n\n");

    switch(event.getKeyCode()){
        case(KeyEvent.VK_UP):
            //System.out.println("Up");
                    v3d1.y += 0.01;
                t3d1.setTranslation(v3d1);
            pozycjaObracania.setTransform(t3d1);
            break;
        case(KeyEvent.VK_DOWN):
            //System.out.println("Down");
                    v3d1.y -= 0.1;
                t3d1.setTranslation(v3d1);
            pozycjaObracania.setTransform(t3d1);
            break;
        case(KeyEvent.VK_LEFT):
            //System.out.println("Left");

                t3d1.setTranslation(
                    new Vector3d(
                        0.0, v3d1.y, 0.0
                    )
                );
            pozycjaObracania.setTransform(t3d1);

                    v3d3.x -= v3d1.z*Math.sin(rot);
                    v3d3.z += v3d1.z*Math.cos(rot);
                    rot -= Math.PI/124;
                    //System.out.println(rot+"\t"+Math.sin(rot)+"\t"+Math.cos(rot));
```

```
            t3d3.setTranslation(v3d3);
        elementObracany1.setTransform(t3d3);


                AxisAngle4d a4d = new AxisAngle4d(
                    0, 1, 0,
                    rot
                );
            t3d2.setRotation(a4d);
        elementObracany.setTransform(t3d2);
        break;
    case(KeyEvent.VK_RIGHT):
        //System.out.println("Right");
            t3d1.setTranslation(
                new Vector3d(
                    0.0, v3d1.y, 0.0
                )
            );
        pozycjaObracania.setTransform(t3d1);

                v3d3.x -= v3d1.z*Math.sin(rot);
                v3d3.z += v3d1.z*Math.cos(rot);

                rot += Math.PI/124;
                //System.out.println(rot+"\t"+Math.sin(rot)+"\t"+Math.cos(rot));

            t3d3.setTranslation(v3d3);
        elementObracany1.setTransform(t3d3);


                a4d = new AxisAngle4d(
                    0, 1, 0,
                    rot
                );
            t3d2.setRotation(a4d);
        elementObracany.setTransform(t3d2);
        break;

    case(KeyEvent.VK_ADD):
        mpi.inceraseSpeed();
        break;
    case(KeyEvent.VK_SUBTRACT):
        mpi.deceraseSpeed();
        break;
    case(KeyEvent.VK_SPACE):
            BranchGroup bg = new BranchGroup();
                ShootInterpolator si = new ShootInterpolator();
            bg.addChild(si);
```

```
                    (new Thread(si)).start();
                group.addChild(bg);
                break;

            default:
                System.out.println(event.getKeyCode());
                break;
        }
    }

}
```

MyPositionInterpolator.java

```
import javax.media.j3d.*;
import javax.vecmath.*;

import java.awt.*;
import javax.swing.*;

class MyPositionInterpolator extends JPanel implements Runnable{

    TransformGroup tg;
    Transform3D t3d = new Transform3D();
    Vector3d v3d = new Vector3d();

    public boolean kolizja = false;

    private int sleep = 50;
    private Image img1, img2;
    boolean end = true;

    public MyPositionInterpolator(TransformGroup tg){
        this.tg = tg;

        tg.setCapability(
            TransformGroup.ALLOW_TRANSFORM_READ
        );
        tg.setCapability(
            TransformGroup.ALLOW_TRANSFORM_WRITE
        );

        // Panel prep

        setPreferredSize(
            new Dimension(120, 35)
        );
        setBackground(Color.lightGray);

        Toolkit t = this.getToolkit();
        img1 = t.getImage("speed.gif");
        img2 = t.getImage("speed-pasek.gif");
    }

    public void toDelete(){
        Frame f = new Frame();
        f.add(this);
        f.setVisible(true);
        f.pack();
    }

    public void run(){
```

```
        while(end){
            tg.getTransform(t3d);
                t3d.get(v3d);
                    if(kolizja)
                        v3d.z += 0.005;
                    else
                        v3d.z += 0.01;
                t3d.setTranslation(v3d);
            tg.setTransform(t3d);
            try{
                Thread.sleep(sleep);
            }catch(Exception ex){
            }
        }
    }

    public void paintComponent(Graphics g){
        super.paintComponent(g);
        g.drawImage(img1, 0, 0, this);
        g.drawImage(img2, 10, 10, 100-sleep, 15, this);
    }

    public void inceraseSpeed(){
        if(sleep>1)
            sleep -= 1;
            System.out.println(sleep);
        repaint();
    }
    public void deceraseSpeed(){
        if(sleep<100)
            sleep += 2;
        repaint();
    }

    public void collision(){
        sleep = 100;
        repaint();
    }
}
```

NewElement.java

```
import javax.media.j3d.*;
import javax.vecmath.*;

import com.sun.j3d.utils.geometry.*;

import java.awt.*;
import javax.swing.*;

public class NewElement extends OrderedGroup{

    Vector3d v3d = new Vector3d(0.0, 0.0, 0.0);
    AxisAngle4d a4d = new AxisAngle4d(0.0, 1.0, 0.0, 0.0);
    BranchGroup[] temp = new BranchGroup[4];
    int kierunek = 0,
        next = 0;

    public static JPanel jp = new JPanel(new FlowLayout());
    public static JLabel jlp = new JLabel("Point: 0");
    public static JLabel jlt = new JLabel("Time: 0");
    public static Thread t;
    public static int timer = 0;

    public NewElement(){

        jp.add(jlp);
        jp.add(jlt);

        t = new Thread(){
           public void run(){
            while(true){
                try{
                    timer+=1;
                    jlt.setText("Time: "+timer);
                    sleep(1000);
                }catch(Exception ex){
                }
            }
           }
        };

        t.start();

        for(int i=0; i<3;i++){
            this.process();
        }
        setCapability(
            Group.ALLOW_CHILDREN_EXTEND
        );
```

```
    setCapability(
        Group.ALLOW_CHILDREN_READ
    );
    setCapability(
        Group.ALLOW_CHILDREN_WRITE
    );
}



public void process(){

    BranchGroup bg = new BranchGroup();
        TransformGroup tmp;
            Transform3D t3d;
    switch( next ){

        case 0:
            switch(kierunek){
                case 0:
                    v3d.z -= 2.5;
                    break;
                case 1:
                    v3d.x += 2.5;
                    break;
                case 2:
                    v3d.z += 2.5;
                    break;
                case 3:
                    v3d.x -= 2.5;
                    break;
            }
            bg = new BranchGroup();
            bg.setCapability(
                BranchGroup.ALLOW_DETACH
            );
                tmp = new TransformGroup();
                    t3d = new Transform3D();
                        t3d.setTranslation(
                            v3d
                        );
                        t3d.setRotation(
                            a4d
                        );
                tmp.setTransform(t3d);

                tmp.addChild(
                    new Plaszczyzna1(0.0)
                );
```

```
    switch(kierunek){
        case 0:
            v3d.z -= 2.5;
            break;
        case 1:
            v3d.x += 2.5;
            break;
        case 2:
            v3d.z += 2.5;
            break;
        case 3:
            v3d.x -= 2.5;
            break;
    }

        TransformGroup tg1 = new TransformGroup();
            Transform3D t3d1 = new Transform3D();

            t3d1.setTranslation(
                new Vector3d(
                    0.0,
                    0.0,
                    -1.5
                )
            );

            tg1.setTransform(
                t3d1
            );

            DetNextElement dne = new DetNextElement();

            tg1.addChild(
                dne
            );

            tg1.addChild(
                new NextComponentDetector(dne, this)
            );
        tmp.addChild(tg1);
    bg.addChild(tmp);
    break;

case 1:

    switch(kierunek){
        case 0:
            v3d.z -= 2.5;
            break;
        case 1:
```

```
        v3d.x += 2.5;
        break;
    case 2:
        v3d.z += 2.5;
        break;
    case 3:
        v3d.x -= 2.5;
        break;
}

bg = new BranchGroup();
bg.setCapability(
    BranchGroup.ALLOW_DETACH
);

    tmp = new TransformGroup();
        t3d = new Transform3D();
            t3d.setTranslation(
                v3d
            );
            t3d.setRotation(
                a4d
            );
    tmp.setTransform(t3d);

    tmp.addChild(
        new PlaszczyznaZ1(0.0, 0.0)
    );

a4d.angle -= Math.PI/2;

switch(kierunek){
    case 0:
        v3d.x += 2.5;
        kierunek = 1;
        break;
    case 1:
        v3d.z += 2.5;
        kierunek = 2;
        break;
    case 2:
        v3d.x -= 2.5;
        kierunek = 3;
        break;
    case 3:
        v3d.z -= 2.5;
        kierunek = 0;
        break;
}
```

```
    tg1 = new TransformGroup();
            t3d1 = new Transform3D();

            t3d1.setTranslation(
                new Vector3d(
                    2.5,
                    0.0,
                    0.0
                )
            );

            t3d1.setRotation(
                a4d
            );
            tg1.setTransform(
                t3d1
            );

            dne = new DetNextElement();

            tg1.addChild(
                dne
            );

            tg1.addChild(
                new NextComponentDetector(dne, this)
            );
        tmp.addChild(tg1);

    bg.addChild(tmp);

    break;

case 2:

    switch(kierunek){
        case 0:
            v3d.z -= 2.5;
            break;
        case 1:
            v3d.x += 2.5;
            break;
        case 2:
            v3d.z += 2.5;
            break;
        case 3:
            v3d.x -= 2.5;
            break;
    }
```

```
bg = new BranchGroup();
bg.setCapability(
    BranchGroup.ALLOW_DETACH
);

    tmp = new TransformGroup();
        t3d = new Transform3D();
            t3d.setTranslation(
                v3d
            );
            t3d.setRotation(
                a4d
            );
    tmp.setTransform(t3d);

    tmp.addChild(
        new PlaszczyznaZ2(0.0, 0.0)
    );
a4d.angle += Math.PI/2;

switch(kierunek){
    case 0:
        v3d.x -= 2.5;
        kierunek = 3;
        break;
    case 1:
        v3d.z -= 2.5;
        kierunek = 0;
    case 2:
        v3d.x -= 2.5;
        kierunek = 1;
        break;
    case 3:
        v3d.z -= 2.5;
        kierunek = 2;
        break;
}

tg1 = new TransformGroup();
        t3d1 = new Transform3D();

        t3d1.setTranslation(
            new Vector3d(
                -2.5,
                0.0,
                0.0
            )
        );

        t3d1.setRotation(
```

```
                    a4d
                );
                tg1.setTransform(
                    t3d1
                );

                dne = new DetNextElement();

                tg1.addChild(
                    dne
                );

                tg1.addChild(
                    new NextComponentDetector(dne, this)
                );
            tmp.addChild(tg1);
        bg.addChild(tmp);

        break;

}

System.out.println(next);

int tab1[] = {0,1,2};

switch(next){

    case 0:
        next = tab1[(int)(Math.random()*2)];
        break;
    case 1:
        next = tab1[(int)(Math.random()*2)];
        break;
    case 2:
        next = tab1[(int)(Math.random()*2)];
        break;
}

temp[3] = bg;

if(numChildren()>2){
    for(int j=0; j< numChildren();j++)
        if( temp[0] == getChild(j)){
            setChild(bg, j);
        }
}else{
    addChild(bg);
}
```

```
            for(int j=0; j<3; j++){
                temp[j] = temp[j+1];
            }
        }
    }
```

NextComponentDetector.java

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;

import javax.media.j3d.*;
import javax.vecmath.*;

public class NextComponentDetector extends Behavior{

    Shape3D s3d;
    TransformGroup tg;
    TransformGroup tg1;
    NewElement og;
    WakeupCriterion wc[] = new WakeupCriterion[2];
    WakeupCondition wuc;

    public static Vector db = new Vector();
    public static int points = 0;

    boolean collision = false;

    public NextComponentDetector(DetNextElement s3d, NewElement og){
        this.og = og;
        this.s3d = s3d;
        this.wc[0] = new WakeupOnCollisionEntry(s3d, WakeupOnCollisionEntry.USE_GEOMETRY);
        this.wc[1] = new WakeupOnCollisionExit(s3d, WakeupOnCollisionEntry.USE_GEOMETRY);


        wuc = new WakeupOr(wc);

        this.setSchedulingBounds(
            new BoundingSphere(
                new Point3d(),
                100
            )
        );
    }

    public void initialize(){
        this.wakeupOn(wuc);
        db.addElement(this);
    }

    Boolean b = new Boolean(true);

    public void processStimulus(Enumeration enum){

        if(wc[0].hasTriggered() && !ShootComponentDetector.b.booleanValue()){
```

```
            System.out.println("Poczatek kolizji");
            collision = true;
            og.process();
            points++;
        }else{
            System.out.println("Not ship");
            this.wakeupOn(wuc);
        }

        if(wc[1].hasTriggered()){
            System.out.println("Koniec kolizji");
        }
    }
                                                    }
```

Plaszczyzna1.java

```
import javax.media.j3d.*;
import javax.vecmath.*;

public class Plaszczyzna1 extends Shape3D{
    int iloscSegX = 20,
        iloscSegZ = 20;
    float tablicaPunktow[] = new float[ ((1+(3+((iloscSegX-1)*2)))*iloscSegZ*3)];
    float kolory[] = new float[tablicaPunktow.length];
    double przesuniecie = 0;
    TriangleStripArray tsa;

    public Plaszczyzna1(double przesuniecie){
        this.przesuniecie = przesuniecie;
        int rozmieszczenie[] = new int[iloscSegZ];

        for(int i=0; i<iloscSegZ;i++){
            rozmieszczenie[i] = (1+(3+((iloscSegX-1)*2)));
        }

        generatorPunktow();

        tsa = new TriangleStripArray(
            tablicaPunktow.length /3,
            GeometryArray.COORDINATES |
            GeometryArray.NORMALS |
            GeometryArray.COLOR_3
            ,
            rozmieszczenie
        );

        // Process zwany normalizacja ???

        int licznik = 0,
            poz = 0;
        for(int i=0; i<iloscSegZ; i++){
            Vector3f normal = new Vector3f();
            int j;

            for(j=0; j<(3+(iloscSegX-1)*2)-1;j+=2){

                Vector3f v1 = new Vector3f(),
                         v2 = new Vector3f();

                Point3f p3f1 = new Point3f(
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)],
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+1],
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+2]
                );
```

```
        Point3f p3f2 = new Point3f(
            tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+3],
            tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+4],
            tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+5]
        );

        v1.sub(
            p3f1,
            p3f2
        );

        v2.sub(
            p3f1,
            p3f2
        );

        normal.cross(v1,v2);
        normal.normalize();

        tsa.setNormal((i*(4+(iloscSegX-1)*2))+j, normal);
        tsa.setNormal((i*(4+(iloscSegX-1)*2))+j+1, normal);
        tsa.setNormal((i*(4+(iloscSegX-1)*2))+j+2, normal);

    }
    tsa.setNormal((i*(4+(iloscSegX-1)*2))+j+1, normal);
}

tsa.setCoordinates(0, tablicaPunktow);
tsa.setColors(0, kolory);

Appearance app = new Appearance();
    PolygonAttributes patt = new PolygonAttributes();
        //patt.setPolygonMode(patt.POLYGON_LINE);
                patt.setCullFace(patt.CULL_FRONT);
             ColoringAttributes ca = new ColoringAttributes();
                ca.setShadeModel(
                    ColoringAttributes.FASTEST
                );
             Material mat = new Material();
                mat.setAmbientColor(0.2f, 0.2f, 0.3f);
                mat.setDiffuseColor(0.9f, 0.9f, 0.1f);
                mat.setEmissiveColor( 0.3f, 0.0f, 0.0f);
                mat.setSpecularColor( 1.0f, 1.0f, 1.0f);
                mat.setShininess(0.0f);

                mat.setLightingEnable(true);
             app.setMaterial(mat);
    app.setPolygonAttributes(patt);
this.setAppearance(app);
```

```
    this.setCapability(Geometry.ALLOW_INTERSECT);
    this.setGeometry(tsa);
}

public Vector3f usrednij(Vector3f normal, int numer){
    Vector3f v3f = new Vector3f();
    tsa.getNormal(numer, v3f);
    v3f.x = (normal.x+v3f.x)/2;
    v3f.y = (normal.y+v3f.y)/2;
    v3f.z = (normal.z+v3f.z)/2;
    return v3f;
}

public void generatorPunktow(){
    double dlugoscX = 5.0,
           dlugoscZ = 5.0,
           krokX = dlugoscX / iloscSegX,
           krokZ = dlugoscZ / iloscSegZ,
           punktStartuX = dlugoscX/2,
           punktStartuZ = dlugoscZ/2;

    double kat = 90,
           x = punktStartuX,
           y = Math.sin( ((kat*Math.PI)/180)+przesuniecie),
           z = -punktStartuZ,
           krokKatowy = 360.0 / iloscSegX;

    int wstawianyPunkt = 0;

    for(int i=0; i < iloscSegZ; i++){
        tablicaPunktow[wstawianyPunkt  ] = (float) x;
        tablicaPunktow[wstawianyPunkt+1] = (float) y;
        tablicaPunktow[wstawianyPunkt+2] = (float) z;

        kolory[wstawianyPunkt  ] = (float) x;
        kolory[wstawianyPunkt+1] = (float) (x*-1);
        kolory[wstawianyPunkt+2] = 0.0f;

        wstawianyPunkt+=3;

        z+=krokZ;

        for(int j=0; j<(3+(iloscSegX-1)*2);j++){

            tablicaPunktow[wstawianyPunkt  ] = (float) x;
            tablicaPunktow[wstawianyPunkt+1] = (float) y;
            tablicaPunktow[wstawianyPunkt+2] = (float) z;

            kolory[wstawianyPunkt  ] = (float) x;
```

```
                kolory[wstawianyPunkt+1] = (float) (x*-1);
                kolory[wstawianyPunkt+2] = 0.0f;

                if( (j%2)==0){
                    kat += krokKatowy;

                    x-=krokX;
                    y = Math.sin( ((kat*Math.PI)/180)+przesuniecie);
                    z-=krokZ;
                }else{
                    z+=krokZ;
                }
                wstawianyPunkt+=3;
            }
            kat = 90;
            x = punktStartuX;
            y = Math.sin( ((kat*Math.PI)/180)+przesuniecie);
            z += krokZ;
        }
    }
}
```

PlaszczyznaZ1.java

```
import javax.media.j3d.*;
import javax.vecmath.*;

public class PlaszczyznaZ1 extends Shape3D{
    int iloscSegX = 20,
        iloscSegZ = 20;
    float tablicaPunktow[] = new float[ ((1+(3+((iloscSegX-1)*2)))*iloscSegZ*3)];
    float kolory[] = new float[tablicaPunktow.length];
    double przesuniecie = 0;
    TriangleStripArray tsa;
    double promien;

    public PlaszczyznaZ1(double przesuniecie, double promien){
        this.przesuniecie = przesuniecie;
        this.promien = promien;
        int rozmieszczenie[] = new int[iloscSegZ];

        for(int i=0; i<iloscSegZ;i++){
            rozmieszczenie[i] = (1+(3+((iloscSegX-1)*2)));
        }

        generatorPunktow();

        tsa = new TriangleStripArray(
            tablicaPunktow.length /3,
            GeometryArray.COORDINATES |
            GeometryArray.NORMALS |
            GeometryArray.COLOR_3,
            rozmieszczenie
        );

        // Process zwany normalizacja ???

        int licznik = 0,
            poz = 0;
        for(int i=0; i<iloscSegZ; i++){
            Vector3f normal = new Vector3f();
            int j;

            for(j=0; j<(3+(iloscSegX-1)*2)-1;j+=2){

                Vector3f v1 = new Vector3f(),
                         v2 = new Vector3f();

                Point3f p3f1 = new Point3f(
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)],
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+1],
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+2]
```

```
        );

        Point3f p3f2 = new Point3f(
            tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+3],
            tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+4],
            tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+5]
        );

        v1.sub(
            p3f1,
            p3f2
        );

        v2.sub(
            p3f1,
            p3f2
        );

        normal.cross(v1,v2);
        normal.normalize();

        tsa.setNormal((i*(4+(iloscSegX-1)*2))+j, normal);
        tsa.setNormal((i*(4+(iloscSegX-1)*2))+j+1, normal);
        tsa.setNormal((i*(4+(iloscSegX-1)*2))+j+2, normal);

    }
    tsa.setNormal((i*(4+(iloscSegX-1)*2))+j+1, normal);
}

tsa.setCoordinates(0, tablicaPunktow);
tsa.setColors(0, kolory);

Appearance app = new Appearance();
    PolygonAttributes patt = new PolygonAttributes();
        //patt.setPolygonMode(patt.POLYGON_LINE);
                patt.setCullFace(patt.CULL_BACK);
             ColoringAttributes ca = new ColoringAttributes();
                ca.setShadeModel(
                    ColoringAttributes.FASTEST
                );
             Material mat = new Material();
                mat.setEmissiveColor( 0.2f, 0.2f, 0.5f);
                mat.setShininess(0.2f);
                mat.setDiffuseColor(0.2f, 0.0f, 0.0f);
                mat.setLightingEnable(true);
             app.setMaterial(mat);
    app.setPolygonAttributes(patt);
this.setAppearance(app);

this.setCapability(Geometry.ALLOW_INTERSECT);
```

```
    this.setGeometry(tsa);
}

public Vector3f usrednij(Vector3f normal, int numer){
    Vector3f v3f = new Vector3f();
    tsa.getNormal(numer, v3f);
    v3f.x = (normal.x+v3f.x)/2;
    v3f.y = (normal.y+v3f.y)/2;
    v3f.z = (normal.z+v3f.z)/2;
    return v3f;
}

public void generatorPunktow(){
    double dlugoscX = 5.0,
           dlugoscZ = 5.0,
           krokX = dlugoscX / iloscSegX,
           krokZ = (Math.PI*0.5) / iloscSegZ,
           punktStartuX = dlugoscX / 2,
           punktStartuZ = dlugoscZ / 2;

    double kat = 90,
           x = 0.0,
           y = Math.sin( ((kat*Math.PI)/180)+przesuniecie),
           z = Math.PI*1.5;


    double krokKatowy = (360.0*1) / iloscSegX;

    int wstawianyPunkt = 0;


    for(int i=0; i < iloscSegZ; i++){
        tablicaPunktow[wstawianyPunkt  ] = (float) (punktStartuX - (Math.sin(z)*(x-promien)));
        tablicaPunktow[wstawianyPunkt+1] = (float) y;
        tablicaPunktow[wstawianyPunkt+2] = (float) (punktStartuZ + (Math.cos(z)*(x-promien)));

        kolory[wstawianyPunkt  ] = (float) x;
        kolory[wstawianyPunkt+1] = (float) (x*-1);
        kolory[wstawianyPunkt+2] = 0.0f;

        wstawianyPunkt+=3;

        z+=krokZ;

        for(int j=0; j<(3+(iloscSegX-1)*2);j++){

            tablicaPunktow[wstawianyPunkt  ] = (float) (punktStartuX - (Math.sin(z)*(x-promien)));
            tablicaPunktow[wstawianyPunkt+1] = (float) y;
            tablicaPunktow[wstawianyPunkt+2] = (float) (punktStartuZ + (Math.cos(z)*(x-promien)));
```

```
                kolory[wstawianyPunkt  ] = (float)x;
                kolory[wstawianyPunkt+1] = (float) (x*-1);
                kolory[wstawianyPunkt+2] = 0.0f;

                if( (j%2)==0){
                    kat += krokKatowy;

                    x-=krokX;
                    z-=krokZ;
                    y = Math.sin( ((kat*Math.PI)/180)+przesuniecie);
                }else{
                    z+=krokZ;
                }
                wstawianyPunkt+=3;
            }
            kat = 90;
            x = 0.0;
            z += krokZ;
            y = Math.sin( ((kat*Math.PI)/180)+przesuniecie);
        }
    }
}
```

PlaszczyznaZ2.java

```
import javax.media.j3d.*;
import javax.vecmath.*;

public class PlaszczyznaZ2 extends Shape3D{
    int iloscSegX = 20,
        iloscSegZ = 20;
    float tablicaPunktow[] = new float[ ((1+(3+((iloscSegX-1)*2)))*iloscSegZ*3)];
    float kolory[] = new float[tablicaPunktow.length];
    double przesuniecie = 0,
           promien = 0;
    TriangleStripArray tsa;

    public PlaszczyznaZ2(double przesuniecie, double promien){
        this.promien = promien;
        this.przesuniecie = przesuniecie;
        int rozmieszczenie[] = new int[iloscSegZ];

        for(int i=0; i<iloscSegZ;i++){
            rozmieszczenie[i] = (1+(3+((iloscSegX-1)*2)));
        }

        generatorPunktow();

        tsa = new TriangleStripArray(
            tablicaPunktow.length /3,
            GeometryArray.COORDINATES |
            GeometryArray.NORMALS |
            GeometryArray.COLOR_3,
            rozmieszczenie
        );

        // Process zwany normalizacja ???

        int licznik = 0,
            poz = 0;
        for(int i=0; i<iloscSegZ; i++){
            Vector3f normal = new Vector3f();
            int j;

            for(j=0; j<(3+(iloscSegX-1)*2)-1;j+=2){

                Vector3f v1 = new Vector3f(),
                         v2 = new Vector3f();

                Point3f p3f1 = new Point3f(
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)],
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+1],
                    tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+2]
```

```
        );

        Point3f p3f2 = new Point3f(
            tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+3],
            tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+4],
            tablicaPunktow[(((i*(4+(iloscSegX-1)*2))+j)*3)+5]
        );

        v1.sub(
            p3f1,
            p3f2
        );

        v2.sub(
            p3f1,
            p3f2
        );

        normal.cross(v1,v2);
        normal.normalize();

        tsa.setNormal((i*(4+(iloscSegX-1)*2))+j, normal);
        tsa.setNormal((i*(4+(iloscSegX-1)*2))+j+1, normal);
        tsa.setNormal((i*(4+(iloscSegX-1)*2))+j+2, normal);

    }
    tsa.setNormal((i*(4+(iloscSegX-1)*2))+j+1, normal);
}

tsa.setCoordinates(0, tablicaPunktow);
tsa.setColors(0, kolory);

Appearance app = new Appearance();
    PolygonAttributes patt = new PolygonAttributes();
        //patt.setPolygonMode(patt.POLYGON_LINE);
                patt.setCullFace(patt.CULL_FRONT);
             ColoringAttributes ca = new ColoringAttributes();
                ca.setShadeModel(
                    ColoringAttributes.FASTEST
                );
             Material mat = new Material();
                mat.setEmissiveColor( 0.2f, 0.2f, 0.5f);
                mat.setShininess(0.2f);
                mat.setDiffuseColor(0.2f, 0.0f, 0.0f);
                mat.setLightingEnable(true);
             app.setMaterial(mat);
    app.setPolygonAttributes(patt);
this.setAppearance(app);

this.setCapability(Geometry.ALLOW_INTERSECT);
```

```
    this.setGeometry(tsa);
}

public Vector3f usrednij(Vector3f normal, int numer){
    Vector3f v3f = new Vector3f();
    tsa.getNormal(numer, v3f);
    v3f.x = (normal.x+v3f.x)/2;
    v3f.y = (normal.y+v3f.y)/2;
    v3f.z = (normal.z+v3f.z)/2;
    return v3f;
}

public void generatorPunktow(){
    double dlugoscX = 5.0,
           dlugoscZ = 5.0,
           krokX = dlugoscX / iloscSegX,
           krokZ = (Math.PI*0.5) / iloscSegZ,
           punktStartuX = dlugoscX / 2,
           punktStartuZ = dlugoscZ / 2;

    double kat = 90,
           x = 0.0,
           y = Math.sin( ((kat*Math.PI)/180)+przesuniecie),
           z = Math.PI*1.5;


    double krokKatowy = 360.0 / iloscSegX;

    int wstawianyPunkt = 0;
    int punkt = 0;

    for(int i=0; i < iloscSegZ; i++){
        tablicaPunktow[wstawianyPunkt  ] = (float) (-punktStartuX + (Math.sin(z)*(x-promien)));
        tablicaPunktow[wstawianyPunkt+1] = (float) y;
        tablicaPunktow[wstawianyPunkt+2] = (float) (punktStartuZ + (Math.cos(z)*(x-promien)));

        kolory[wstawianyPunkt  ] = (float) x;
        kolory[wstawianyPunkt+1] = (float) (x*-1);
        kolory[wstawianyPunkt+2] = 0.0f;

        wstawianyPunkt+=3;
        punkt++;
        z+=krokZ;

        for(int j=0; j<(3+(iloscSegX-1)*2);j++){

            tablicaPunktow[wstawianyPunkt  ] = (float) (-punktStartuX + (Math.sin(z)*(x-promien)));
            tablicaPunktow[wstawianyPunkt+1] = (float) y;
            tablicaPunktow[wstawianyPunkt+2] = (float) (punktStartuZ + (Math.cos(z)*(x-promien)));
```

```
                kolory[wstawianyPunkt  ] = (float)x;
                kolory[wstawianyPunkt+1] = (float) (x*-1);
                kolory[wstawianyPunkt+2] = 0.0f;

                if( (j%2)==0){
                    kat += krokKatowy;

                    x-=krokX;
                    z-=krokZ;
                    y = Math.sin( ((kat*Math.PI)/180)+przesuniecie);
                }else{
                    z+=krokZ;
                }
                wstawianyPunkt+=3;

            }

            kat = 90;
            x = 0.0;
            y = Math.sin( ((kat*Math.PI)/180)+przesuniecie);
            z += krokZ;
        }
    }
}
```

ShipCollisionDetector.java

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;

import javax.media.j3d.*;
import javax.vecmath.*;
import javax.swing.*;

public class ShipCollisionDetector extends Behavior implements Runnable{

    Shape3D s3d;
    TransformGroup tg;
    MyPositionInterpolator mpi;
    WakeupCriterion wc[] = new WakeupCriterion[2];
    WakeupCondition wuc;

    BackgroundSound bs, bs1;

    boolean collision = false;

    public JPanel panel;
    private int live = 100;
    private Image img1, img2;
    private Thread thread;

    public ShipCollisionDetector(Shape3D s3d, BackgroundSound bs, BackgroundSound bs1){
        this.s3d = s3d;
        this.bs = bs;
        this.bs1 = bs1;

        bs.setCapability(
            Sound.ALLOW_ENABLE_READ
        );
        bs.setCapability(
            Sound.ALLOW_ENABLE_WRITE
        );
        bs1.setCapability(
            Sound.ALLOW_ENABLE_READ
        );
        bs1.setCapability(
            Sound.ALLOW_ENABLE_WRITE
        );



        this.wc[0] = new WakeupOnCollisionEntry(s3d, WakeupOnCollisionEntry.USE_GEOMETRY);
        this.wc[1] = new WakeupOnCollisionExit(s3d, WakeupOnCollisionEntry.USE_GEOMETRY);
```

```
    wuc = new WakeupOr(wc);

    this.setSchedulingBounds(
        new BoundingSphere(
            new Point3d(),
            100
        )
    );



    Toolkit t = Toolkit.getDefaultToolkit();
    img1 = t.getImage("speed.gif");
    img2 = t.getImage("helth-pasek.gif");


    panel = new JPanel(){
        public void paintComponent(Graphics g){
            super.paintComponent(g);
            g.drawImage(img1, 0, 0, panel);
            g.drawImage(img2, 10, 10, live, 15, panel);
            //System.out.println("repaint "+live);
        }
    };
    panel.setDoubleBuffered(true);
    panel.setPreferredSize(
        new Dimension(120, 35)
    );

    thread = new Thread(this);
}

public void initialize(){
    this.wakeupOn(wuc);
}

public void setMPI(MyPositionInterpolator mpi){
    this.mpi = mpi;
}

public void processStimulus(Enumeration enum){
    if(wc[0].hasTriggered()){
        System.out.println("SPoczatek kolizji");
        collision = true;
        bs.setEnable(true);

        if(mpi != null){
            mpi.collision();
            mpi.kolizja = true;
        }
```

```
        if(thread.isAlive()){
            try{
                thread.notify();
            }catch(Exception ex){
                System.out.println(ex);
            }
        }else{
            thread.start();
        }
    }

    if(wc[1].hasTriggered()){
        System.out.println("SKoniec kolizji");
        collision = false;
        bs.setEnable(false);
        if(mpi != null){
            mpi.kolizja = false;
        }
        try{
            thread.wait();
        }catch(Exception ex){
            System.out.println(ex);
        }
        panel.repaint();
    }

    this.wakeupOn(wuc);
}

public JPanel getPanel(){
    return panel;
}

int num = 0;

public void run(){
    while(true && live >= 1){
        System.out.println("Uwaga kolizja"+(num++));
        live -= 1;
        panel.repaint();
        synchronized(this){
            try{
                Thread.sleep(100);
            }catch(Exception ex){
            }
        }
    }
    if(live <= 0){
        mpi.end = false;
        bs.setEnable(false);
```

```
            bs1.setEnable(true);
            panel.repaint();
        }
    }
}
```

ShootComponentDetector.java

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;

import javax.media.j3d.*;
import javax.vecmath.*;

public class ShootComponentDetector extends Behavior{

    NewElement og;

    WakeupCriterion wc[] = new WakeupCriterion[2];
    WakeupCondition wuc;

    private static Boolean tab[] = {
        new Boolean(true),
        new Boolean(false)
    };

    public static Boolean b = tab[1];
    ShootInterpolator tg;

    public ShootComponentDetector(Shape3D s3d, ShootInterpolator tg){
        this.tg = tg;
        this.wc[0] = new WakeupOnCollisionEntry(s3d, WakeupOnCollisionEntry.USE_GEOMETRY);
        this.wc[1] = new WakeupOnCollisionExit(s3d, WakeupOnCollisionEntry.USE_GEOMETRY);
        wuc = new WakeupOr(wc);

        this.setSchedulingBounds(
            new BoundingSphere(
                new Point3d(),
                100.0
            )
        );
    }



    public void initialize(){
        this.wakeupOn(wuc);
        b = tab[1];
    }

    public void processStimulus(Enumeration enum){
        if(wc[0].hasTriggered()){
            System.out.println("Shoot collision enter");
            b = tab[0];
```

```
            tg.sleep(10);
        }

        if(wc[1].hasTriggered()){
            System.out.println("Shoot collision exit");
            b = tab[1];
        }

        this.wakeupOn(wuc);
    }
}
```

ShootInterpolator.java

```
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;

public class ShootInterpolator extends TransformGroup implements Runnable{

    public ShootInterpolator(){

        MediaContainer mc = new MediaContainer("file:/E:/1.wav");
        mc.setCacheEnable(true);

            Transform3D t3dtg = new Transform3D();
            t3dtg.setTranslation(
                new Vector3d(0.0, 0.0, -1.0)
            );
        setTransform(t3dtg);

        setCapability(
            TransformGroup.ALLOW_TRANSFORM_READ
        );
        setCapability(
            TransformGroup.ALLOW_TRANSFORM_WRITE
        );

            Point2f[] distanceGain = {
                      new Point2f( 2.00f, 1.0f ),    // Full volume
                      new Point2f( 4.0f, 0.5f ),    // Half volume
                      new Point2f( 6.00f, 0.25f ),   // Quarter volume
                      new Point2f( 8.0f, 0.0f ),    // Zero volume
                };

            PointSound ps = new PointSound();
                ps.setInitialGain(1.0f);
                ps.setDistanceGain(distanceGain);
                ps.setSoundData(mc);
                ps.setEnable(true);
                ps.setLoop(-1);
                ps.setPosition(0.0f, 0.0f, 0.0f);
                ps.setSchedulingBounds(
                    new BoundingSphere(
                        new Point3d(0.0, 0.0, 0.0),
                        1000.0
                    )
                );
        addChild(ps);

        ColorCube c = new ColorCube(0.1);
        ShootComponentDetector scd = new ShootComponentDetector(c, this);
```

```
        this.addChild(
            scd
        );

        addChild(
            c
        );
    }

    public void run(){
            Transform3D t3d = new Transform3D();
        getTransform(t3d);
                Vector3d v3d = new Vector3d();
            t3d.get(v3d);
        while(true){
            v3d.z -= 0.002;
            t3d.setTranslation(v3d);
            setTransform(t3d);
            sleep(1);
        }
    }

    public void sleep(int wrt){
        try{
            Thread.sleep(wrt);
        }catch(Exception ex){
        }
    }
}
```