Università degli Studi di Udine
Corso di Laurea in Tecnologie
Web e Multimediali

Corso di Immagini e Multimedialità
A.S. 2016/2017

Java3D
Prof. Cristian Virgili

Relazione Finale

Florio Silvia Gioia

Esercizio 3.1

Creare una scena con un color cube.

(a) Effettuare le traslazioni lungo i singoli assi x,y,z

Mantenendo fisso il punto di vista, effettuare le traslazioni:

Per valori positivi della x
Per valori negativi della x
Per valori positivi della y
Per valori negativi della y
Per valori positivi della z
Per valori negativi della z

Effettuare una traslazione in modo che il cubo sia visto in basso a dx e risulti più piccolo.

(b) Effettuare le rotazioni attorno ai singoli assi x,y,z

Mantenendo fisso il punto di vista effettuare le rotazioni:

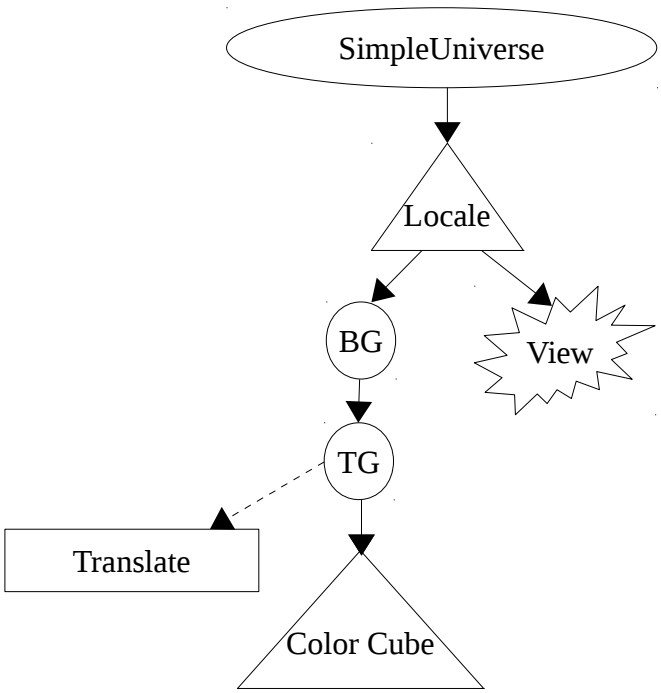
Per valori positivi della x (provare con 90˚,180˚,...)
Per valori negativi della x
Per valori positivi della y
Per valori negativi della y
Per valori positivi della z
Per valori negativi della z

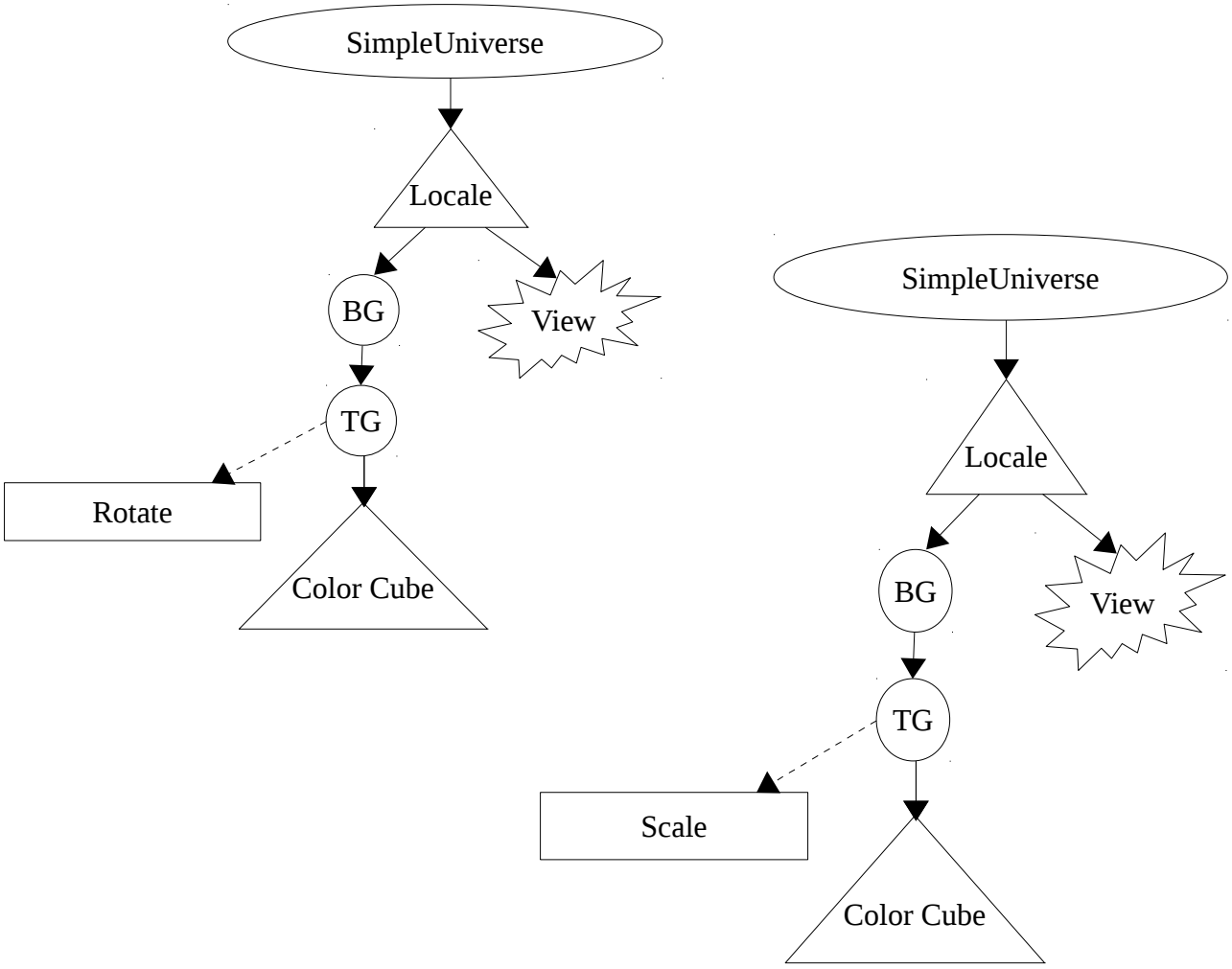
(c) Effettuare le scalature rispetto ai singoli assi x,y,z

Mantenendo fisso il punto di vista effettuare le scalature:

Per valori positivi della x
Per valori negativi della x
Per valori positivi della y
Per valori negativi della y
Per valori positivi della z
Per valori negativi della z

Scenegraph:





Codice generico:

```
public TransformGroup createSubGraph() {
	TransformGroup transform = new TransformGroup();
	double dimCube = 0.1;
	Transform3D t3d = new Transform3D();
	t3d.*; //qui si effettuano le trasformazioni

	transform.setTransform(t3d);
	transform.addChild(new ColorCube(dimCube));
	return transform;
}
```

Traslazione sull'asse x:

```
	t3d.setTranslation(new Vector3d(0.6d, 0d, 0d)); //traslazione a destra

	t3d.setTranslation(new Vector3d(-0.6d, 0d, 0d)); //traslazione a sinistra
```

Traslazione sull'asse y:

```
	t3d.setTranslation(new Vector3d(0d, 0.6d, 0d)); //traslazione in alto

	t3d.setTranslation(new Vector3d(0d, -0.6d, 0d)); //traslazione in basso
```

Traslazione sull'asse z:

```
t3d.setTranslation(new Vector3d(0.0d, 0.0d, 0.6d)); //traslazione in avanti
t3d.setTranslation(new Vector3d(0.0d, 0.0d, -0.6d)); //traslazione indietro
```

Rotazione sull'asse x:

```
t3d.rotX(Math.PI*0.2); //rotazione dall'alto verso il basso
t3d.rotX(-Math.PI*0.2); //rotazione dal basso verso l'alto
```

Rotazione sull'asse y:

```
t3d.rotY(Math.PI*0.2); //rotazione da sinistra verso destra
t3d.rotY(-Math.PI*0.2); //rotazione da destra verso sinistra
```

Rotazione sull'asse z:

```
t3d.rotZ(Math.PI*0.2); //rotazione in senso anti orario
t3d.rotZ(-Math.PI*0.2); //rotazione in senso orario
```

Scala sull'asse x:

```
t3d.setScale(new Vector3d(1.5d, 1d, 1d)); //più largo
t3d.setScale(new Vector3d(-1.5d, 1d, 1d)); //rivoltato
```

Scala sull'asse y:

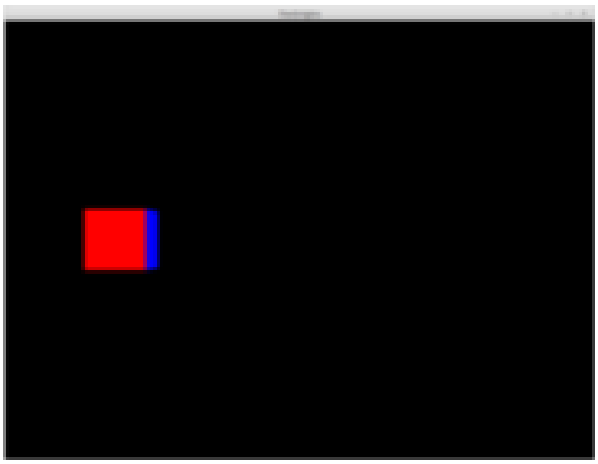
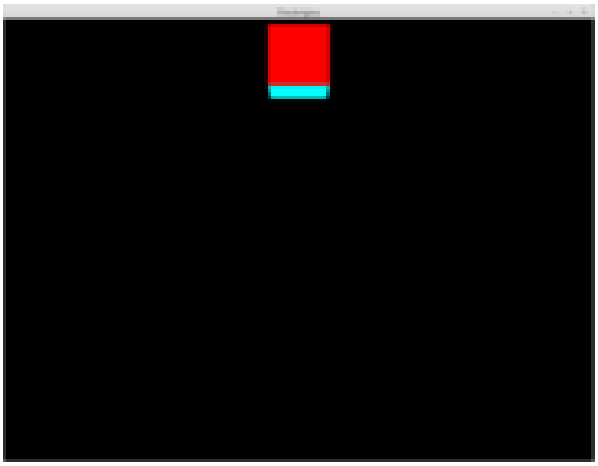
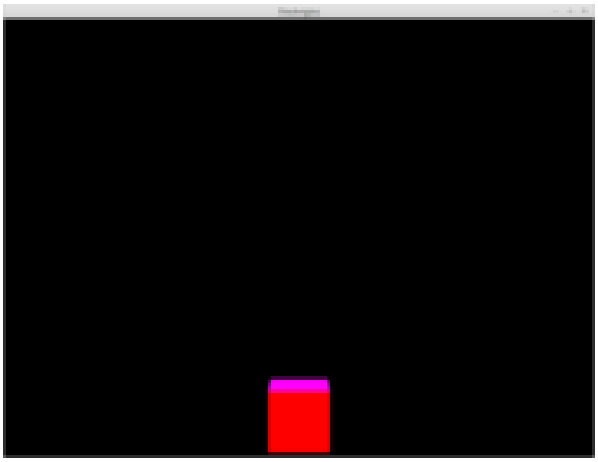
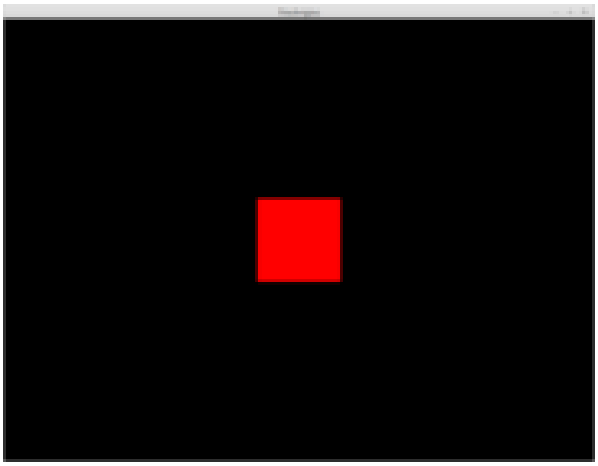
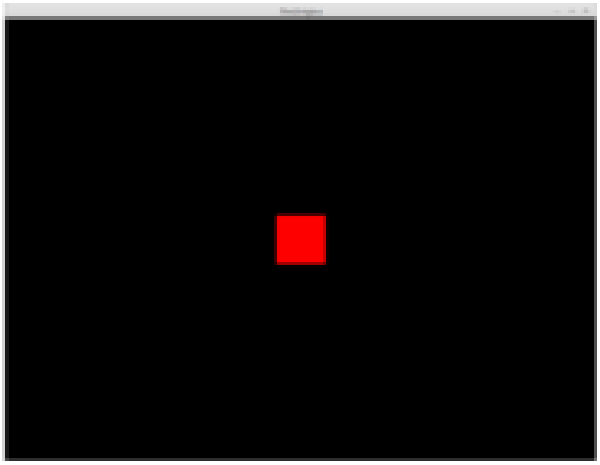
```
t3d.setScale(new Vector3d(1d, 1.5d, 1d)); //più alto
t3d.setScale(new Vector3d(1d, -1.5d, 1d)); //rivoltato
```

Scala sull'asse z:

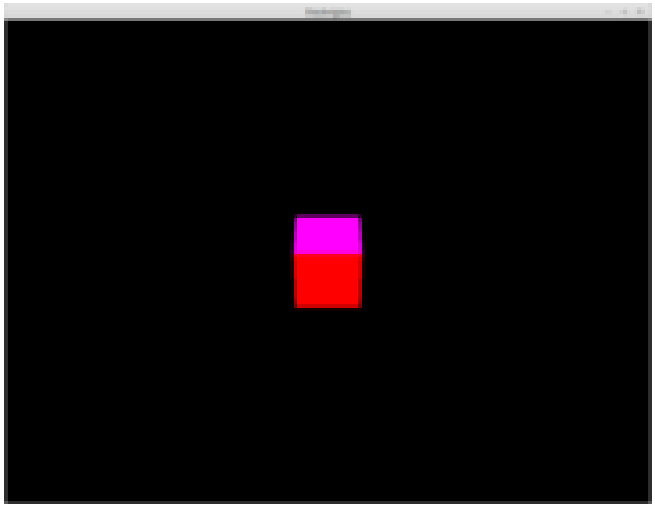
```
t3d.setScale(new Vector3d(1d, 1d, 1.5d)); //non si notano cambiamenti
t3d.setScale(new Vector3d(1d, 1d, -1.5d)); //rivoltato
```

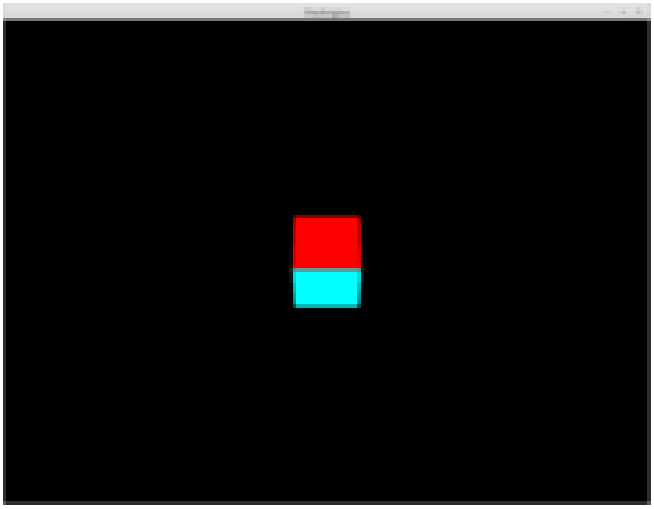
Output:

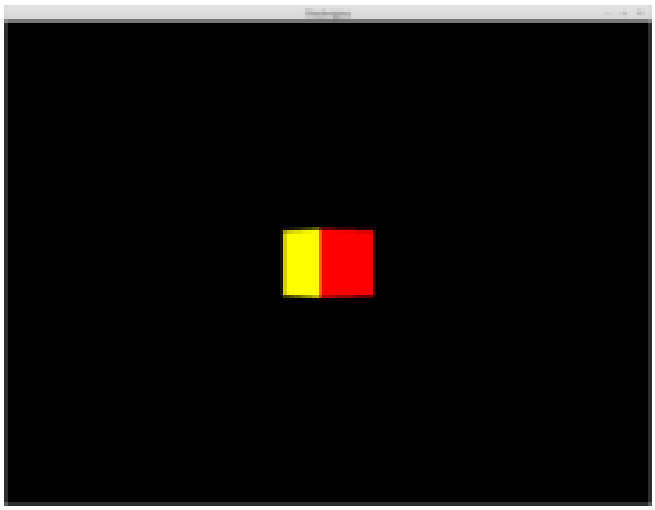
- Translate

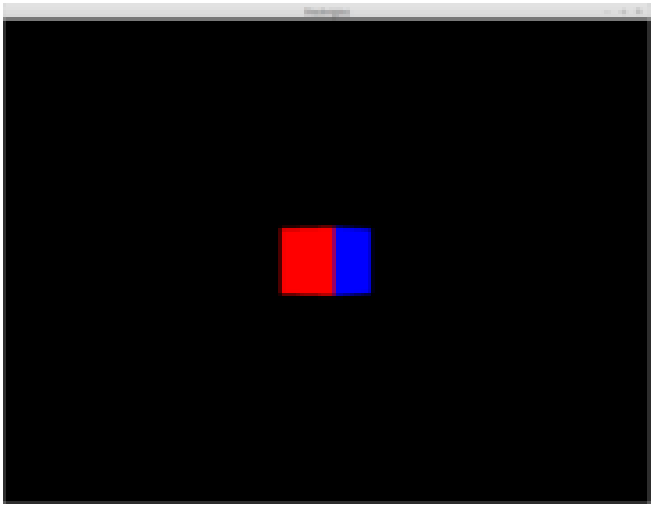







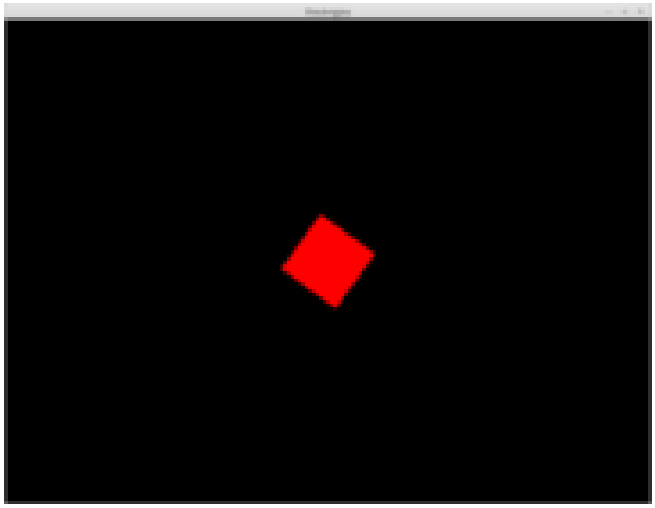
- Rotate

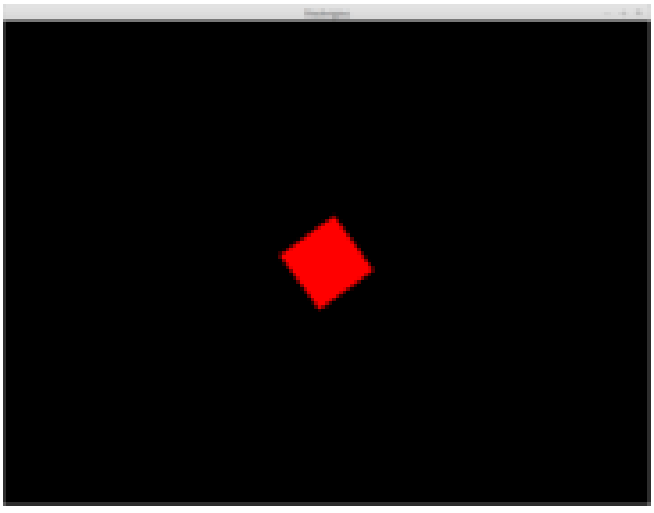




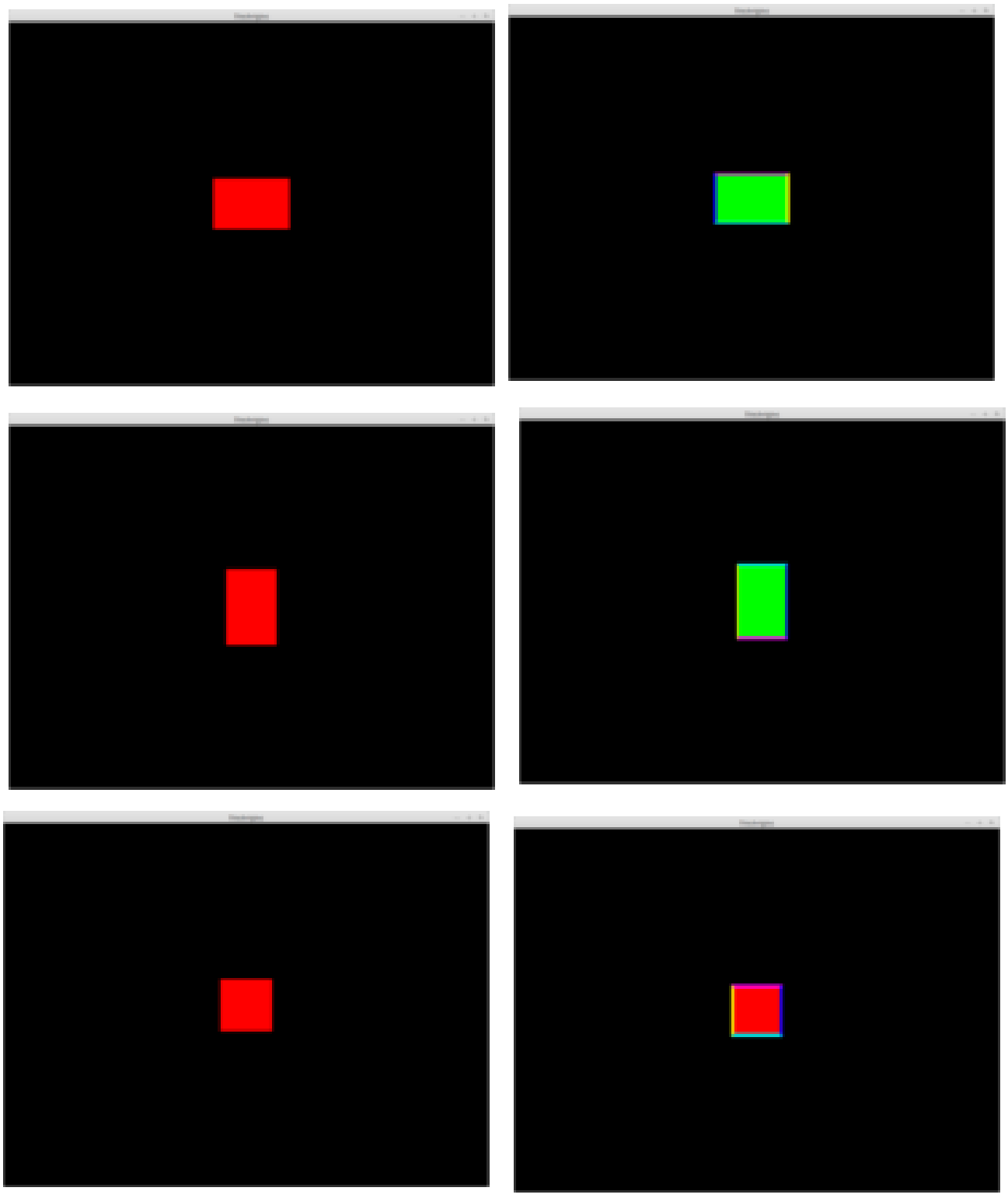








- Scale

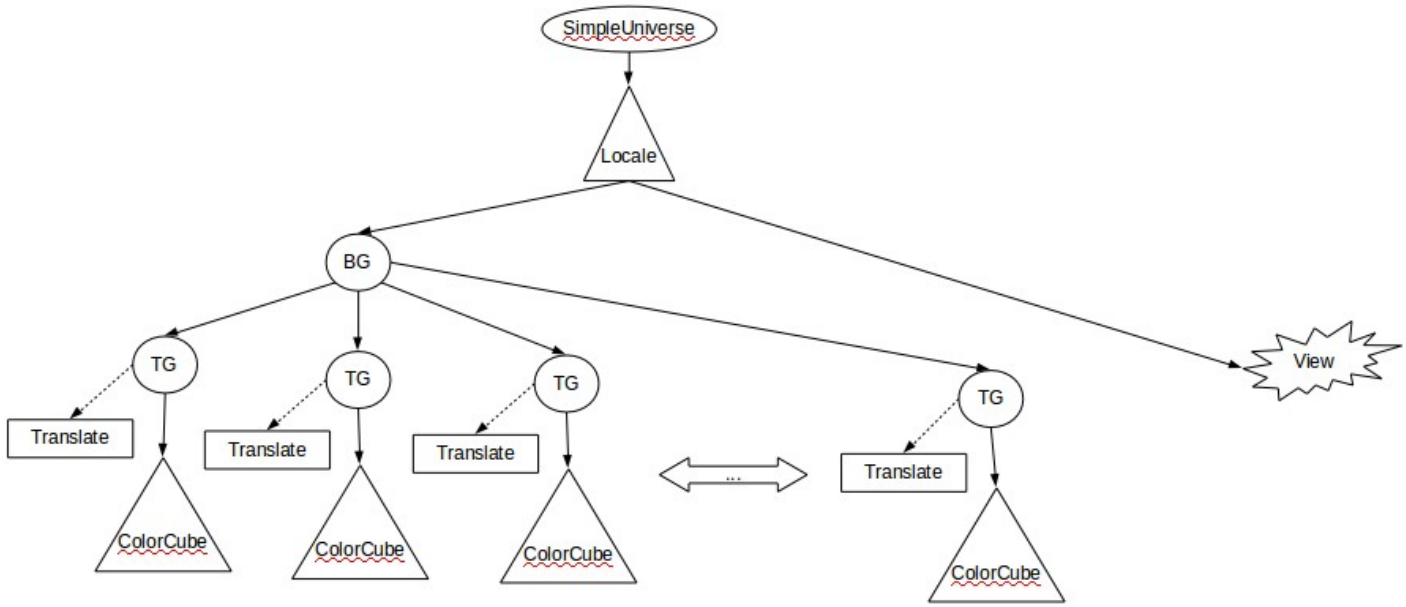


Si nota come per valori di scala negativi il cubo venga rivoltato dall'interno, come se ci fosse qualcosa che lo comprimesse e spingesse oltre il piano rosso.

Esercizio 3.2

Utilizzando le trasformazioni, creare una scena con un numero arbitrario di cubi (diversi) disposti a cerchio.

Scenegraph:



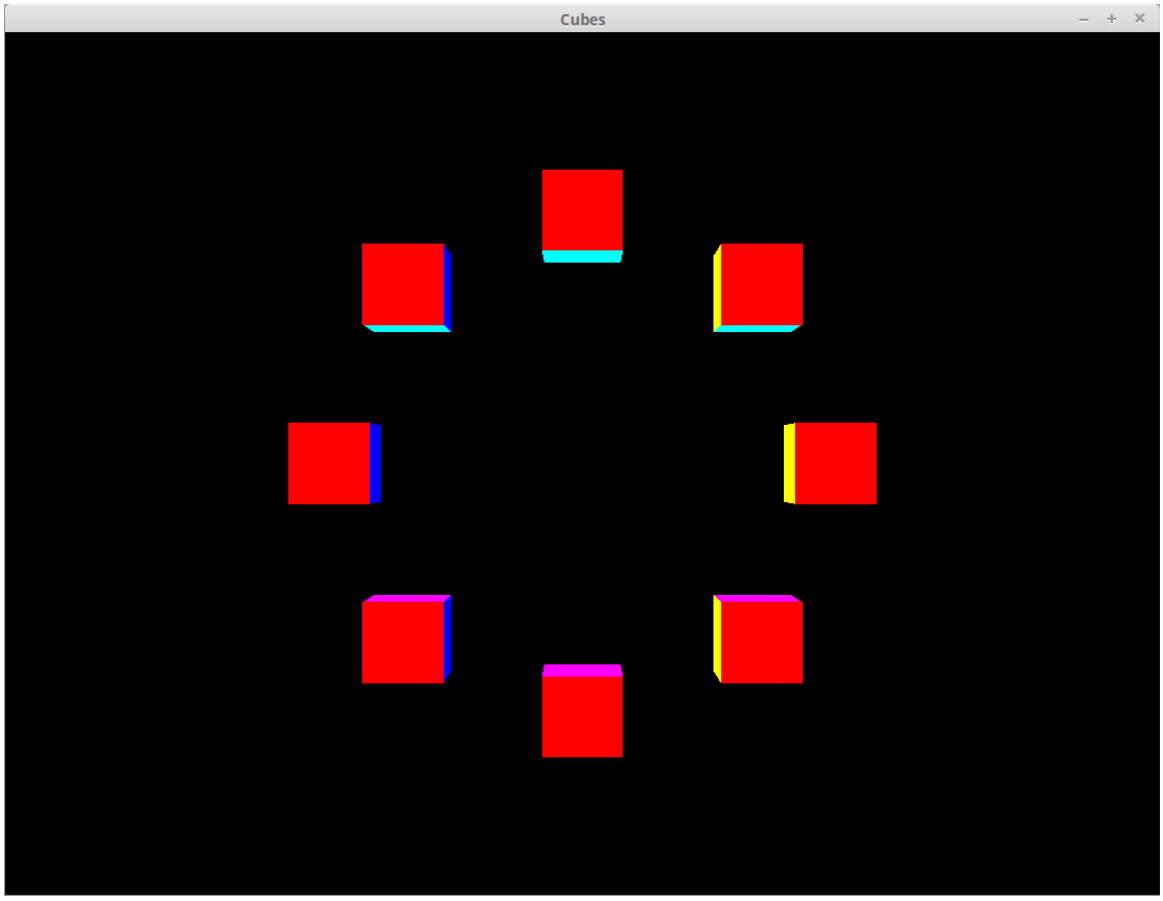
In totale nella scena ci sono 8 TransformGroup che hanno ognuno un ColorCube come figlio.

Codice:

```
public class Cubes extends Applet{

	public Cubes() {
		setLayout(new BorderLayout());
		GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
		Canvas3D canvas3D = new Canvas3D(config);
		add("Center", canvas3D);
		BranchGroup scene = createSceneGraph();
		scene.compile();

		SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
		simpleU.getViewingPlatform().setNominalViewingTransform();
		simpleU.addBranchGraph(scene);
	}

	//funzione che crea lo scene graph
	public BranchGroup createSceneGraph() {
		BranchGroup node = new BranchGroup();
		TransformGroup TG;
		double dimCube = 0.04; //dimensione del cubo
		int numCubes = 8; //numero di cubi che voglio nel cerchio
		Transform3D t3d;
		Matrix3d transformMatrix;
		Matrix3d axisVector = new Matrix3d(0,0,0,0,0,0,1,0,0);
		Vector3d translation = new Vector3d();
		/*ciclo che crea I cubi, si serve di numCubes per creare l'angolo giusto
		per distanziare ogni cubo*/
		for(double i=0; i<2*Math.PI; i= i+Math.PI/(numCubes/2)) {
			TG = new TransformGroup();
			t3d = new Transform3D();

			transformMatrix = new Matrix3d(1, 0, Math.cos(i)/4, 0, 1,
Math.sin(i)/4, 0, 0, 1);
```

```
                transformMatrix.mul(axisVector);
                transformMatrix.getColumn(0, translation);

                t3d.setTranslation(translation);
                TG.setTransform(t3d);

                TG.addChild(new ColorCube(dimCube));
                node.addChild(TG);
            }

            return node;

        }
```

Output:


Cubes

Esercizio 3.3

Utilizzare lookAt() per mostrare una scena da diversi punti di vista.

Trovare trasformazioni da applicare al ViewPlatform per cui si hanno: - 1 punto di fuga;
- 2 punti di fuga;
- 3 punti di fuga

Scenegraph:

Codice:

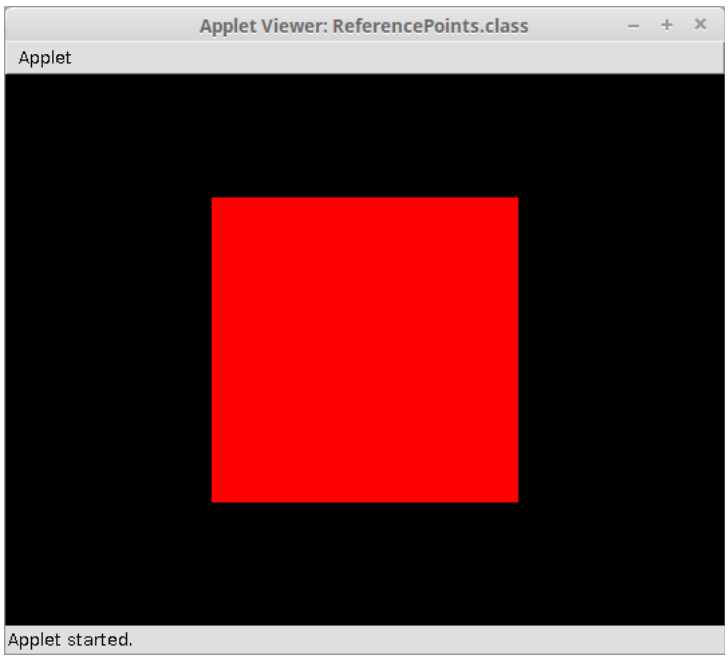
- nel costruttore:

```
ViewingPlatform vp = simpleU.getViewingPlatform();
/* one reference point */
setTransform(vp, new Point3d (0,0,2), new Point3d(0,0,0), new Vector3d(0,1,0));
/* two references points */
setTransform(vp, new Point3d (1,0,2), new Point3d(0,0,0), new Vector3d(0,1,0));
/* three references points */
setTransform(vp, new Point3d (1,1,2), new Point3d(0,0,0), new Vector3d(0,1,0));
```

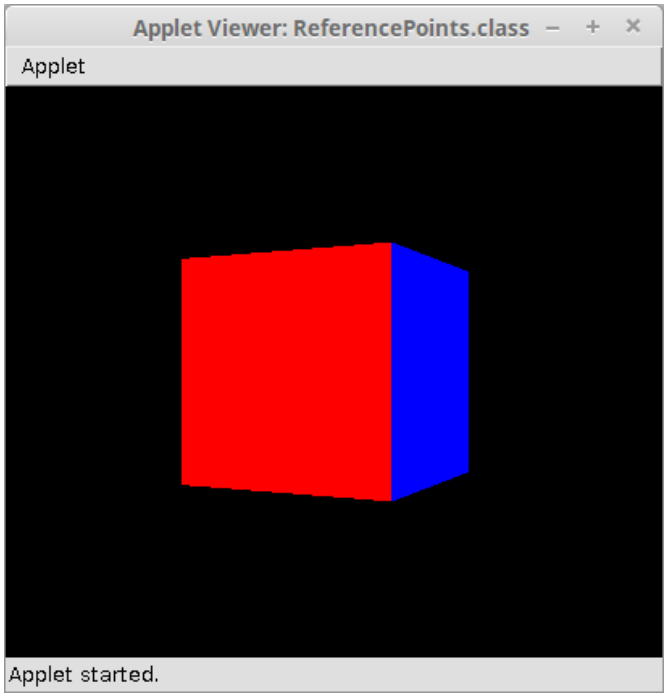
- la funzione:

```
public void setTransform(ViewingPlatform vp, Point3d p1, Point3d p2, Vector3d
                         vector) {
      TransformGroup vtg = vp.getViewPlatformTransform ( ) ;
      Transform3D t3d2 = new Transform3D();
      vtg.getTransform(t3d2); //estraggo la trasformazione della ViewPlatform

      t3d2.lookAt(p1,p2,vector);//sposto la vista
      t3d2.invert( ) ;
      vtg.setTransform(t3d2);

}
```

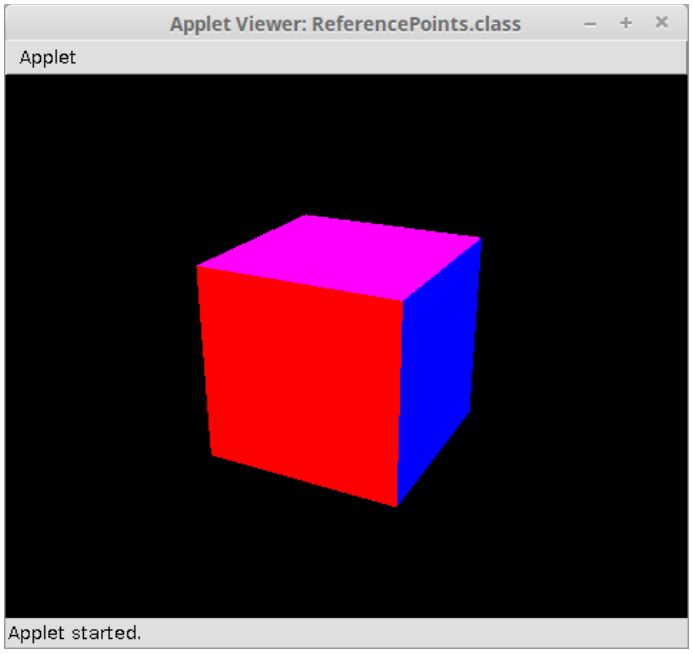
Output:



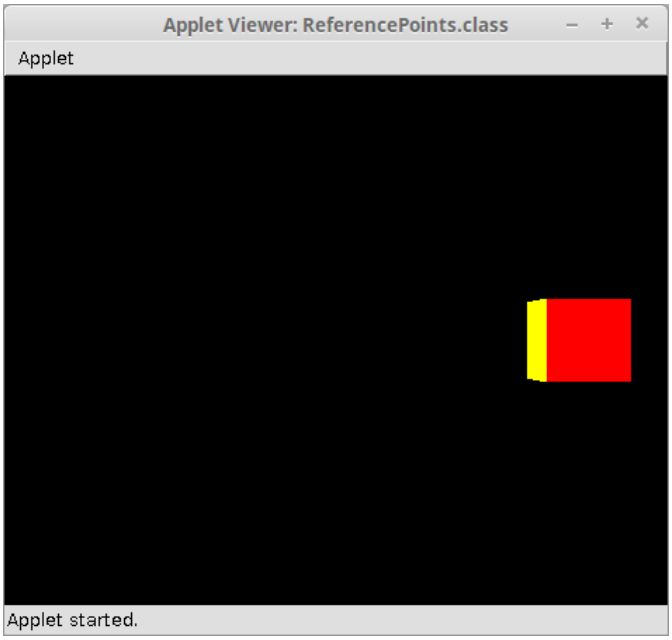
Un punto di fuga - dietro al centro



Due punti di fuga



Tre punti di fuga



Un punto di fuga - cubo traslato a destra

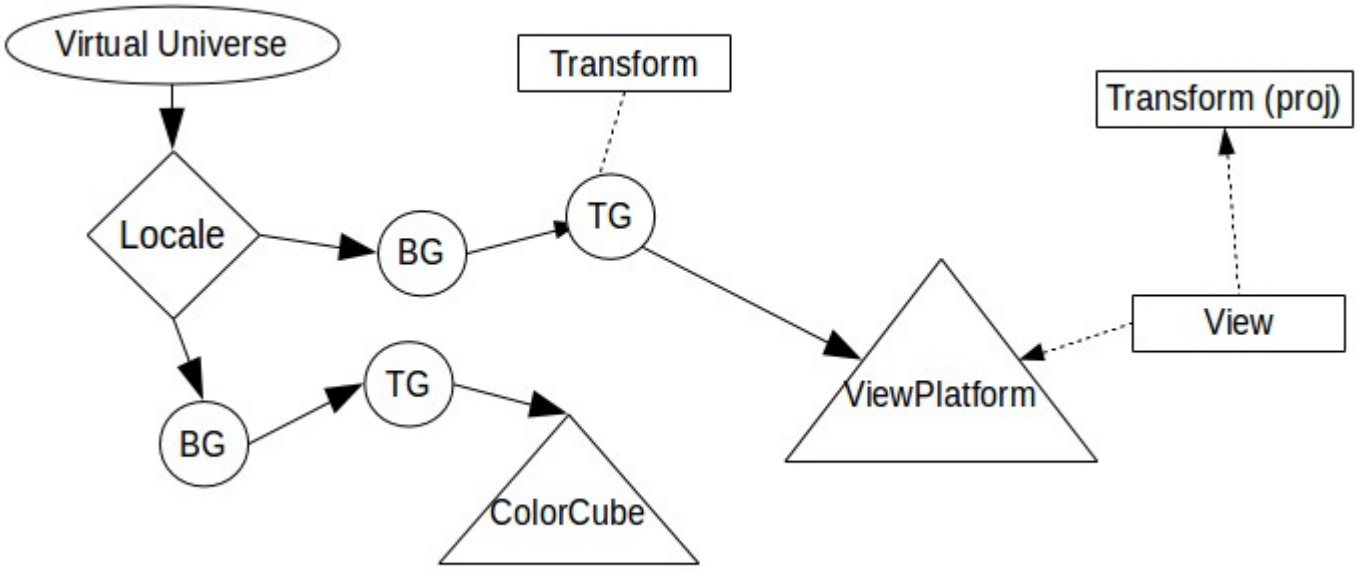
Osservazioni:
Per lavorare con la vista utilizziamo il metodo lookAt() che si può immaginare come lo spostare la testa dell'osservatore di modo che guardi da dove e verso dove vogliamo noi.
Per applicarlo, dobbiamo prima tirare fuori la trasformazione attuale della ViewPlatform.

Esercizio 3.4

Riprendendo dall'esercizio precedente una scene con 2 o 3 punti di fuga, impostare la proiezione con setLeftProjection()

in modo da sperimentare la visualizzazione con:
- Proiezione ortografica con diverse profondità.
- Proiezione prospettica con diverse aperture angolari e distanze focali (e quindi diversi livelli di deformazione prospettica)

Scenegraph:



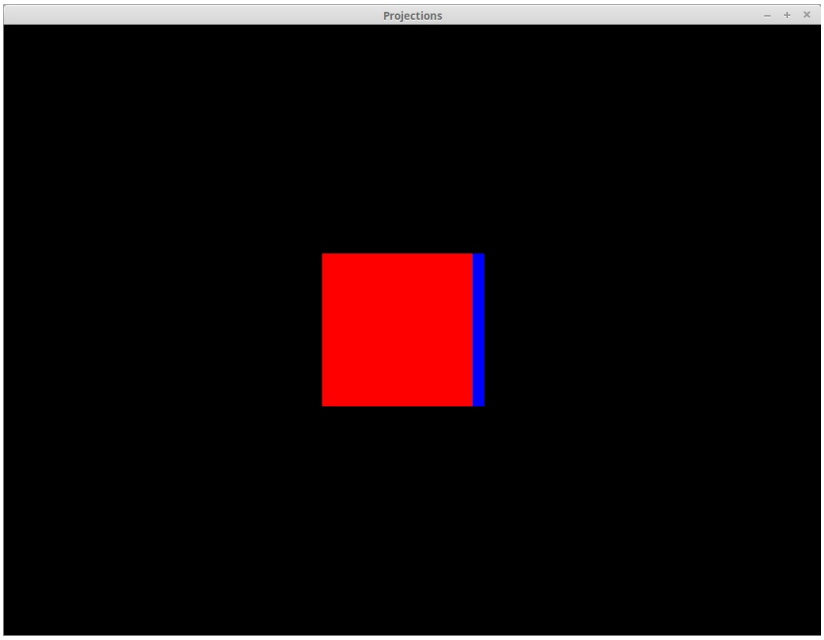
Proiezione Ortografica

Codice:

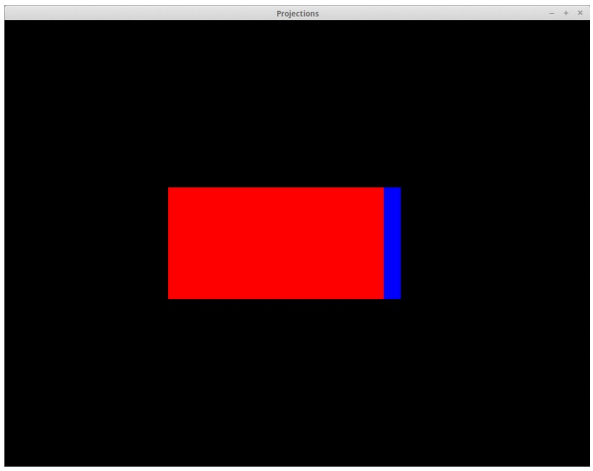
- nel costruttore:

```
ViewingPlatform vp = simpleU.getViewingPlatform();
setTransform(vp, new Point3d (2, 0, 10 ), new Point3d(0,0,0), new
             Vector3d(0,1,0)); //due punti di fuga

View myView = simpleU.getViewer().getView(); // referenza alla view
myView.setCompatibilityModeEnable(true);// posso modificare la matrice di
                                          proiezione

Transform3D proj = new Transform3D();

// ortografica
double ratio = 1024.0/768.0;
proj.ortho(-2*ratio, 2*ratio, -2.0, 2.0, 1, 10);

myView.setLeftProjection(proj);
```

Output:

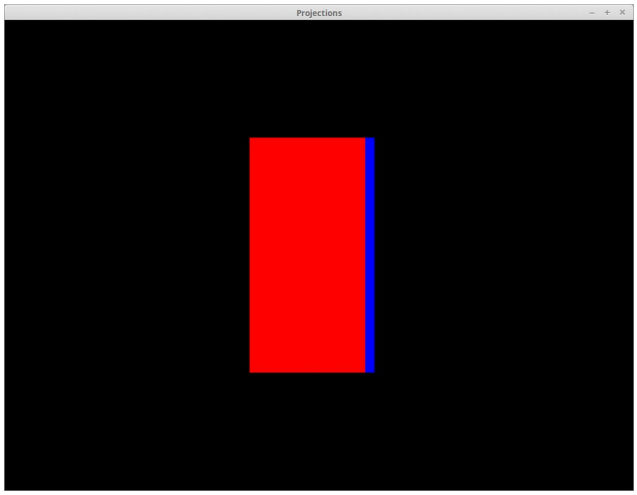


- Variazioni dei parametri left, right, top, bottom:



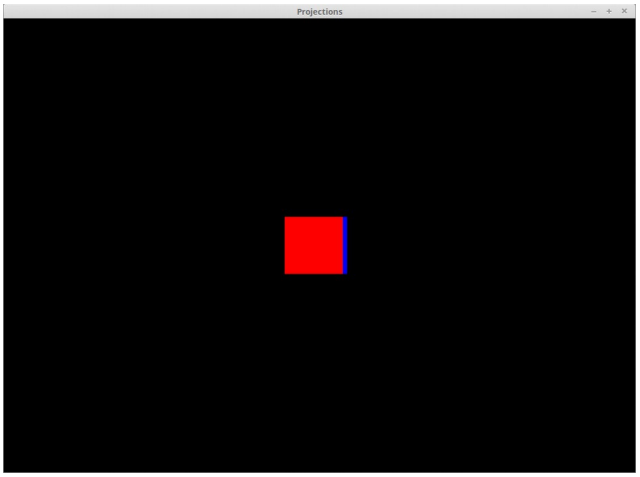
right & left: ratio;

top & bottom: 2.0



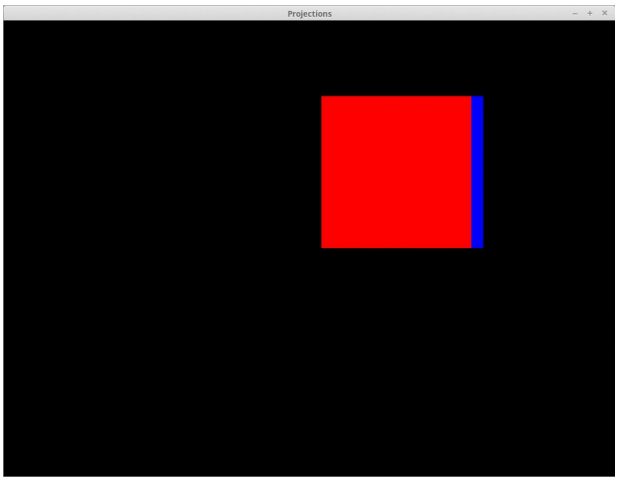
*right & left: 2*ratio;*

top & bottom: 1.0



*right & left: 4*ratio;*

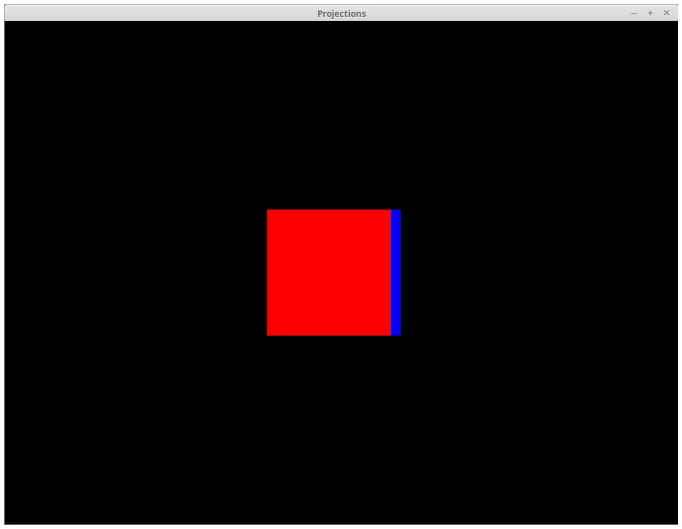
top & bottom: 4.0



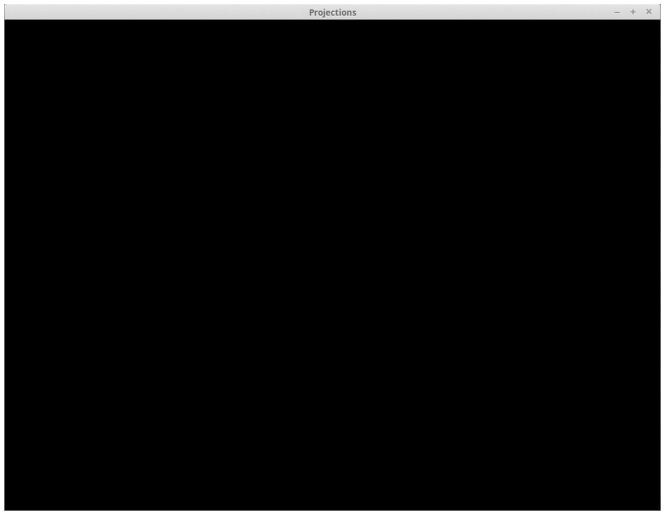
*left: 2*ratio; right: ratio*

bottom: 2.0; top: 1.0

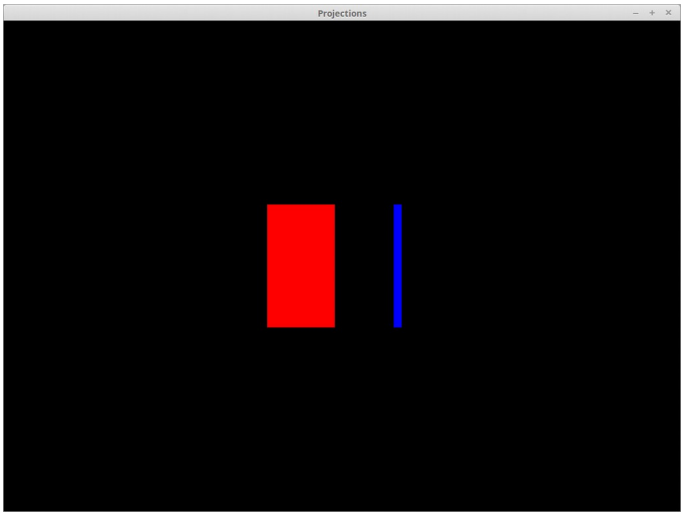
- Variazione del parametro near



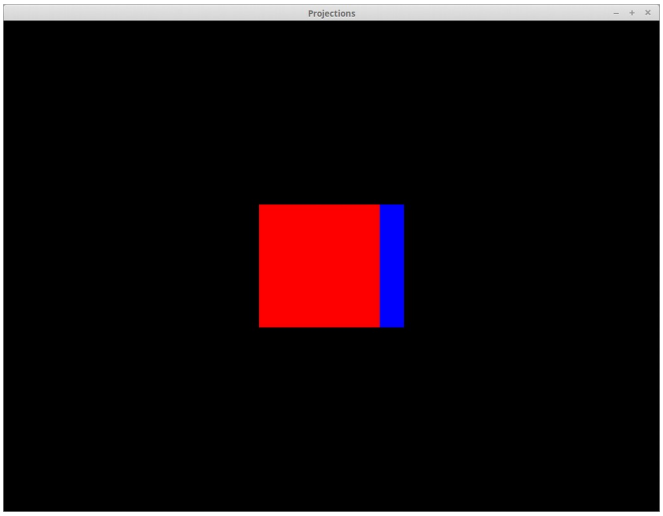
5.0 – 10.0



11.0 – 10.0



9.7 – 10.0



1.0-100.0

Osservazioni (proiezioni ortografiche)**:**
Variare i parametri *left right bottom top* significa cambiare la dimensione della finestra frontale del frostum e di conseguenza se non vengono mantenute le proporzioni si ha una deformazione dell'oggetto e se non si mantiene la specularità delle coppie left-right e bottom-top si ha un decentramento dall'origine.

Variare i parametri *near* e *far* invece allunga e restringe lo spazio prospettico, facendo aumentare o diminuire la porzione blu del cubo visibile. Ovviamente, se la differenza tra *far* e *near* è negativa, il cubo non si vede, perché il frustum è nullo. Nel momento in cui la differenza tra *far* e *near* è minore della grandezza del cubo, invece, quello che si vede è solo una porzione “tagliata” del cubo. Cosa che succede anche se il parametro *near* lo impostiamo in modo che la facciata frontale del frustum intersechi il cubo.

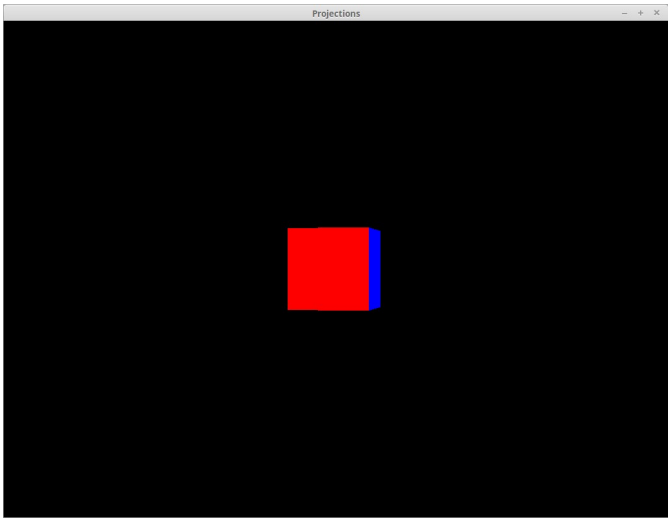
Proiezione Prospettica

Codice:

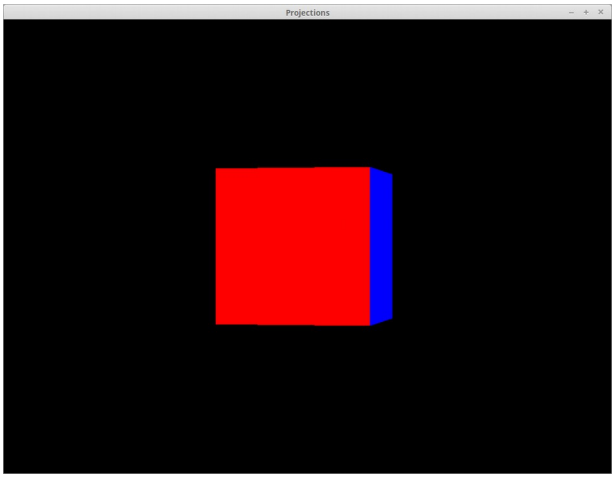
- nel costruttore:

```
ViewingPlatform vp = simpleU.getViewingPlatform();
setTransform(vp, new Point3d (2, 0, 10 ), new Point3d(0,0,0), new
             Vector3d(0,1,0)); //due punti di fuga

View myView = simpleU.getViewer().getView(); // referenza alla view

double ratio = 1024.0/768.0;
// prospettica
proj.perspective(Math.PI/4.0, ratio, 1, 30.0);

myView.setLeftProjection(proj);
```

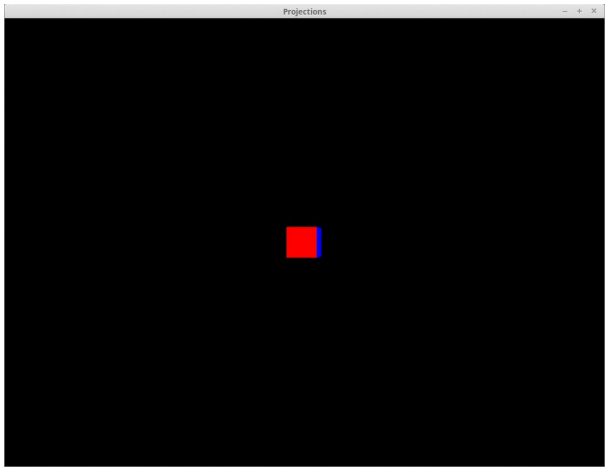
Output:



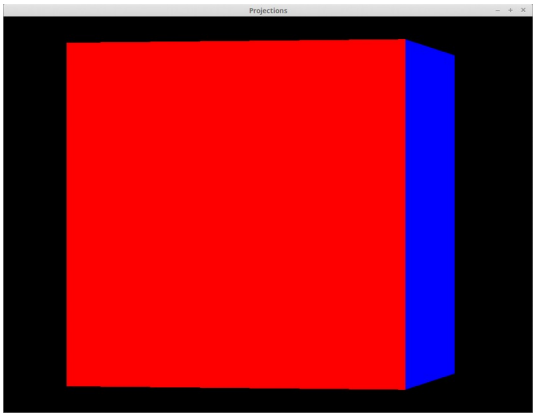
- Variazione del Field of View



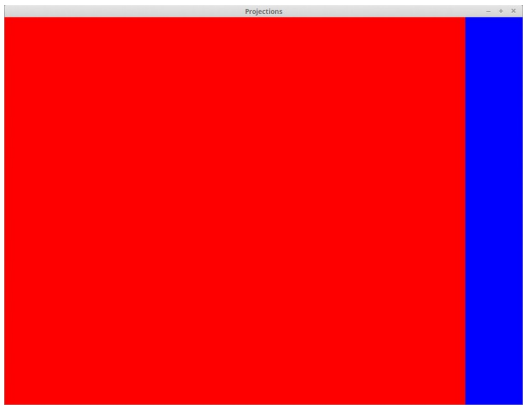
Math.PI/8.0



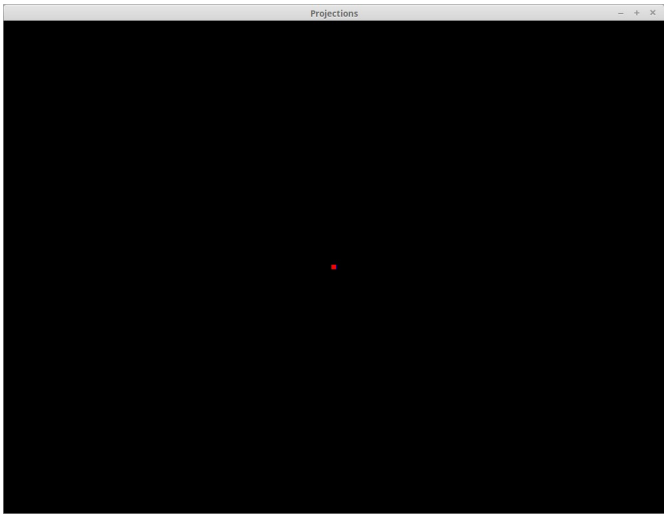
Math.PI/2.0



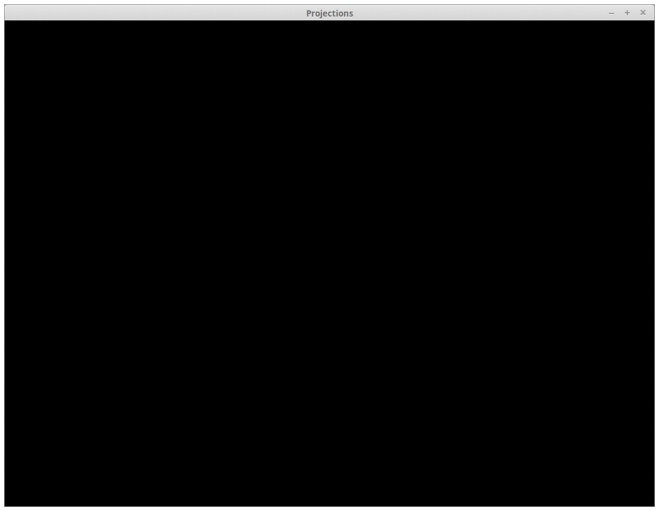
Math.PI/20.0



Math.PI/30.0



Math.PI/1.1



Math.PI

Osservazioni (proiezioni prospettiche)**:**
Con diminuire della distanza tra i due fuochi l'oggetto sembra allontanarsi sempre più fino a sparire, mentre con l'aumentarne l'oggetto si avvicina e sovrasta l'osservatore.

Per quanto riguarda il parametro near valgono osservazioni e output analoghi a quelli per le proiezioni ortografiche.

Esercizio 3.5

Sperimentare le stesse proiezioni dell'esercizio 3.4 con gli opportuni metodi di View.

Riprendendo dall'esercizio precedente una scene con 2 o 3 punti di fuga,

in modo da sperimentare la visualizzazione con:
- Proiezione ortografica con diverse profondità.
- Proiezione prospettica con diverse aperture angolari e distanze focali (e quindi diversi livelli di deformazione prospettica).

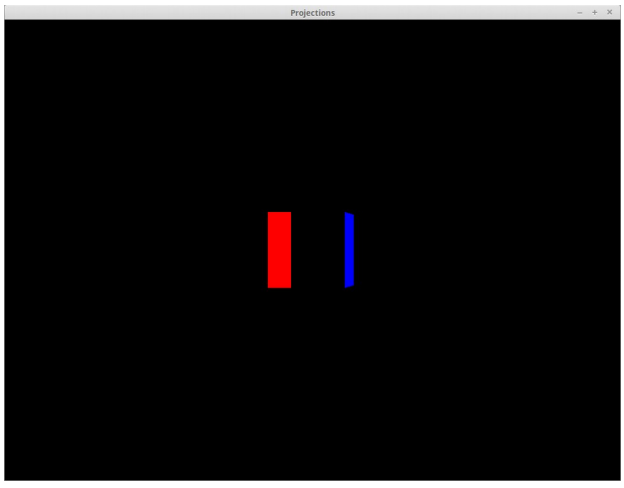
Codice:

- nel costruttore:

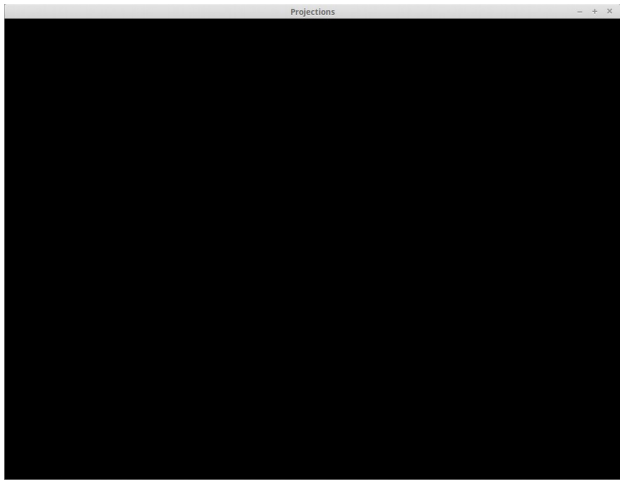
```
myView.setFrontClipDistance(1); // impostazione distanza frontale dal piano
myView.setBackClipDistance(100.0); // impostazione distanza dietro al piano

myView.setFieldOfView(Math.PI/4.0); // impostazione del campo visivo – solo
                                       prospettica

myView.setProjectionPolicy(View.PARALLEL_PROJECTION); // ortografica
myView.setProjectionPolicy(View.PERSPECTIVE_PROJECTION);// prospettica
```

Output:

- Variazione della Front Clip Distance



2.2



4.0

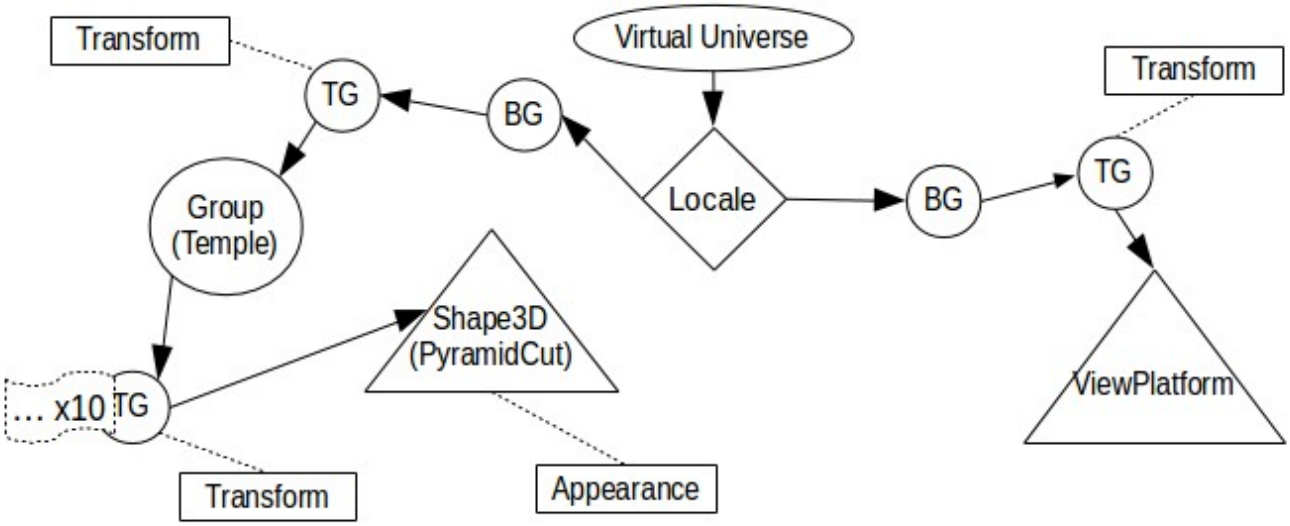
Valgono scenegraph e output analoghi all'esercizio precedente.

Esercizio 3.6

- Implementare una classe derivata da Shape3D per la creazione di tronchi di piramide quadrata.

- Implementare una classe derivata da Group per la creazione di una piramide Maya (senza scalinata e porta).

Scenegraph:



Codice:

- PyramidCut extends Shape3D :

```
public class PyramidCut extends Shape3D{
protected Geometry geometry;
protected Appearance appearance;

private Point3d p1;
private Point3d p2;
private Point3d p3;
private Point3d p4;
private Point3d p5;
private Point3d p6;
private Point3d p7;
private Point3d p8;

public PyramidCut(double height, double sideBottom, double sideTop) {
      /* inizializzo i vertici del tronco di piramide a partire dai parametri in
      input */
      p1 = new Point3d( sideBottom/2, 0, -sideBottom/2);
      p2 = new Point3d(-sideBottom/2, 0, -sideBottom/2);
      p3 = new Point3d(-sideBottom/2, 0, sideBottom/2);
      p4 = new Point3d ( sideBottom/2, 0, sideBottom/2);
      p5 = new Point3d( sideTop/2, height, -sideTop/2);
      p6 = new Point3d(-sideTop/2, height, -sideTop/2);
      p7 = new Point3d(-sideTop/2, height, sideTop/2);
      p8 = new Point3d ( sideTop/2, height, sideTop/2);

      /* creo le facce */
      Point3d[] faces = {
                  p4,p1,p2,
```

```
                        p4,p2,p3,
                        p1,p5,p6,
                        p1,p6,p2,
                        p2,p6,p7,
                        p2,p7,p3,
                        p4,p8,p3,
                        p3,p8,p7,
                        p1,p5,p4,
                        p4,p5,p8,
                        p5,p6,p8,
                        p8,p6,p7
                };

                /* creo il tronco a partire dalle facce */
                geometry=createGeometry(faces);
                setGeometry(geometry);

                appearance=createAppearance();
                setAppearance(appearance);
        }

        protected Geometry createGeometry (Point3d[] faces) {
                /* creo il tronco a partire dalle facce */
                TriangleArray triangles ;
                triangles = new TriangleArray(faces.length, TriangleArray.COORDINATES);
                triangles.setCoordinates(0,faces );

                return triangles ;
        }

        protected Appearance createAppearance() {
                Appearance app = new Appearance () ;
                /* POLYGON_FILL per vedere le facce */
                app.setPolygonAttributes(new
        PolygonAttributes(PolygonAttributes.POLYGON_FILL, PolygonAttributes.CULL_NONE,0)
        );
                /* per la trasparenza */
                //app.setTransparencyAttributes(new
        TransparencyAttributes(TransparencyAttributes.BLENDED, 0.75f));
                /* per il colore */
                //app.setColoringAttributes(new ColoringAttributes(0f, 255f, 0f, 1));

                /* POLYGON_LINE per vedere il wireframe */
                //app.setPolygonAttributes(new
        PolygonAttributes(PolygonAttributes.POLYGON_LINE, PolygonAttributes.CULL_NONE,0)
        );

                return app;
        }
}
```

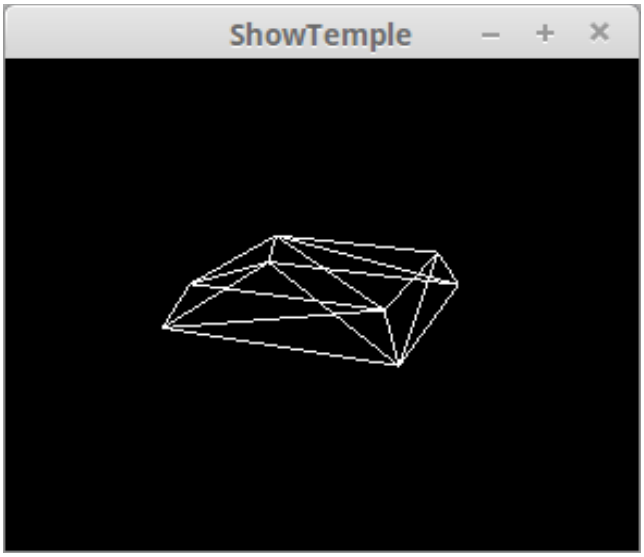
- Temple extends Group:

```
public class Temple extends Group {

	public Temple() {
		TransformGroup TG;
		Transform3D t3d;

		double height = .1;
		double trans = 0;

		for(double i=2; i>.6; i-=.2) {
			TG = new TransformGroup();
			t3d = new Transform3D();

			/* imposta la traslazione a partire dalla variabile trans che aumenta
			ad ogni iterazione */
			t3d.setTranslation(new Vector3d(0,trans,0));
			TG.setTransform(t3d);
			/* il nuovo tronco di piramide si basa sui valori della i che sono
			decrescenti */
			TG.addChild(new PyramidCut(height,i,i-.1)); // aggiungo il nuovo
			                                               tronco di piramide
			addChild(TG);

			trans += height;

		}
		/* le ultime due parti sono della stessa dimensione */
		for(double i=0; i<2; i++) {
			TG = new TransformGroup();
			t3d = new Transform3D();

			t3d.setTranslation(new Vector3d(0,trans,0));
			TG.setTransform(t3d);

			TG.addChild(new PyramidCut(height,.4,.4));
			addChild(TG);

			trans += height;
		}

	}


}
```

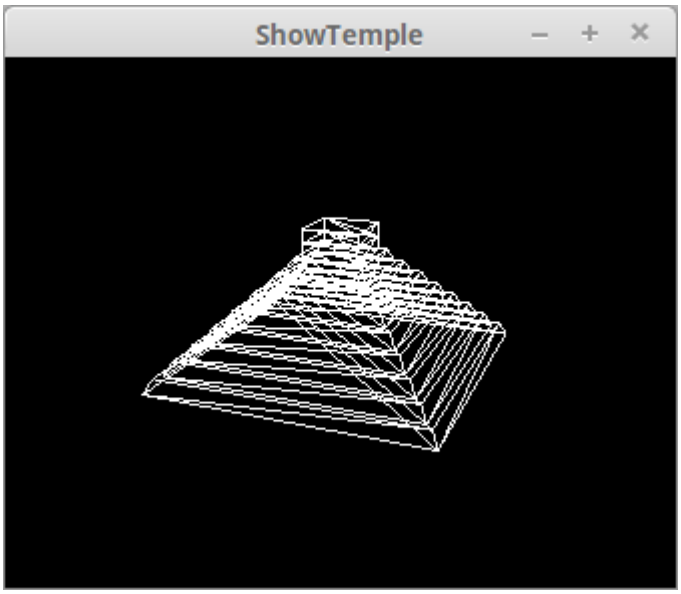
- ShowTemple:

```
	public BranchGroup createSceneGraph() {
		BranchGroup objRoot = new BranchGroup();
		TransformGroup TG = new TransformGroup();
		Transform3D t3d = new Transform3D();
		t3d.setTranslation(new Vector3d(0,-.3,0));
		TG.setTransform(t3d);
		Temple temple = new Temple();

		TG.addChild(temple);
		objRoot.addChild(TG);
		return objRoot;
	}
```

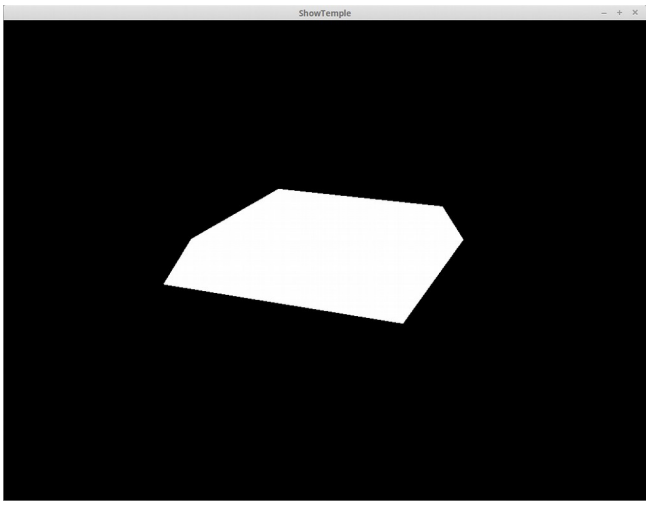
Output:



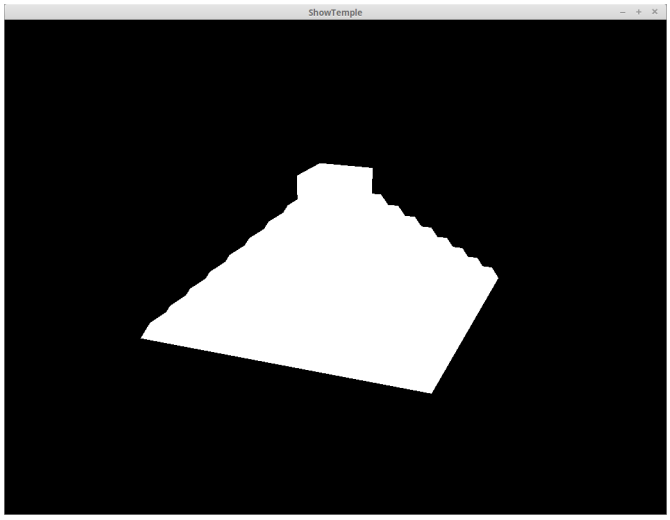
PyramidCut: Wireframe



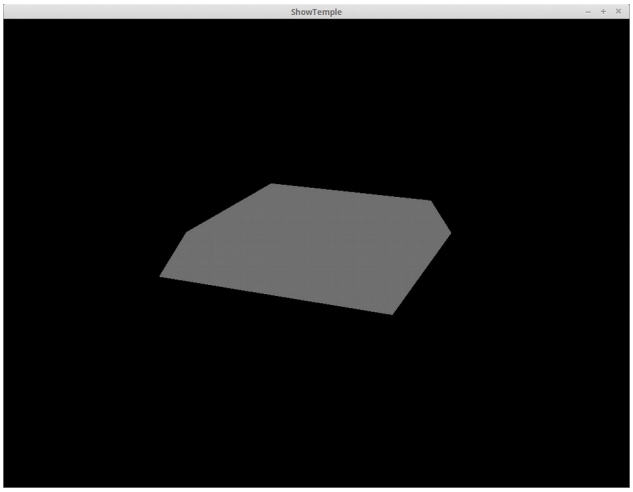
Temple: Wireframe



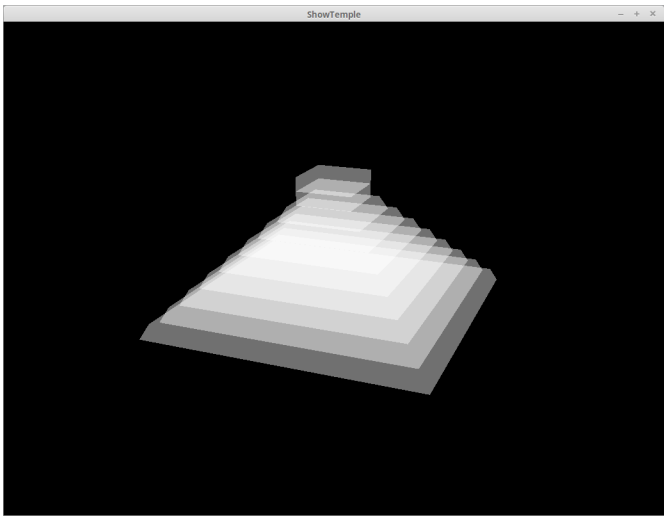
PyramidCut: Fill



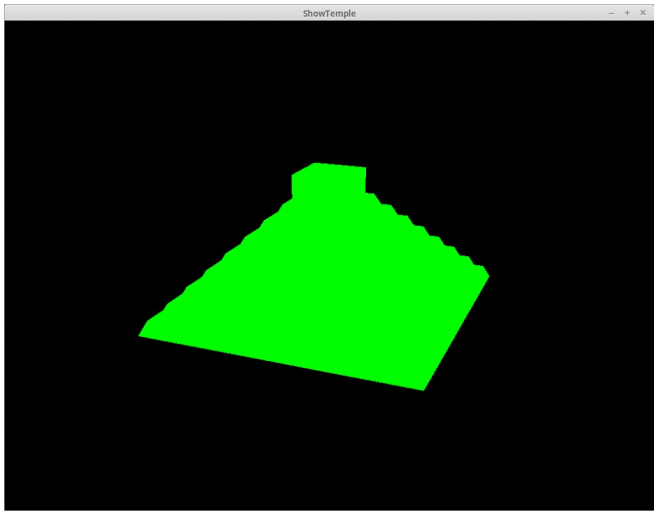
Temple: Fill



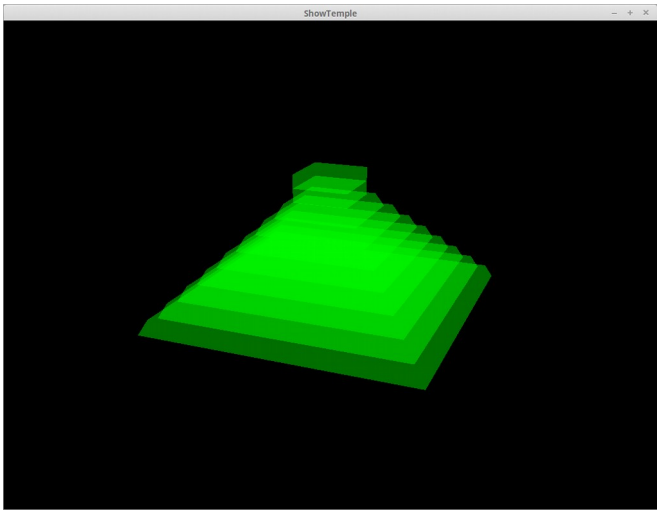
PyramidCut: Fill + Transparency .75



Temple: Fill + Transparency .75



Temple: Coloring (0, 255, 0, 1)



Temple: Coloring (0, 255, 0, 1) + Transparency .75

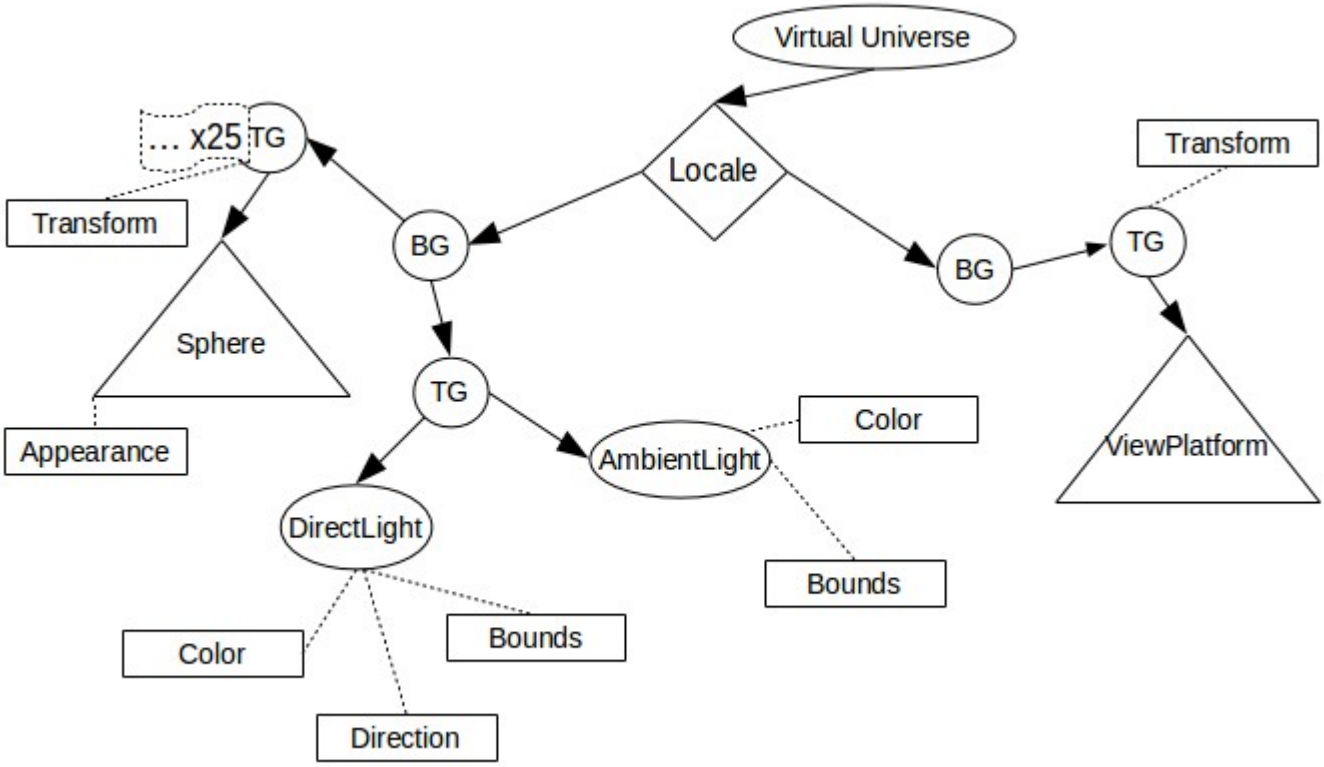
Osservazioni:
In questo esercizio abbiamo appreso la possibilità di creare forme personalizzate a partire dai suoi vertici estendendo la classe Shape3D e la possibilità di utilizzare queste forme personalizzate per creare forme più complesse estendendo la classe Group.

Abbiamo anche visto come gestire alcuni parametri della Appearance per vedere l'oggetto in wireframe piuttosto che semi-trasparente o colorato di un particolare colore RGB.

Esercizio 3.7

Testare le luci su una matrice 5x5 di sfere.

Scenegraph:



Codice:

- AmbientLight:

```
private void ambientLight(TransformGroup node){
	/* il bound definisce lo spazio dell'illuminazione */
	BoundingSphere bounds = new BoundingSphere(
	new Point3d(0.d,0.d,0.d),10.d); // dall'origine, un raggio di 10d
	AmbientLight lightP1 = new AmbientLight();// creazione di una luce
	// Color3f green = new Color3f(0.0f, 1.0f, 0.0f);
	// lightP1.setColor(green); // coloro la luce
	lightP1.setInfluencingBounds(bounds);
	node.addChild(lightP1); // aggiunta della light al BranchGroup
}
```

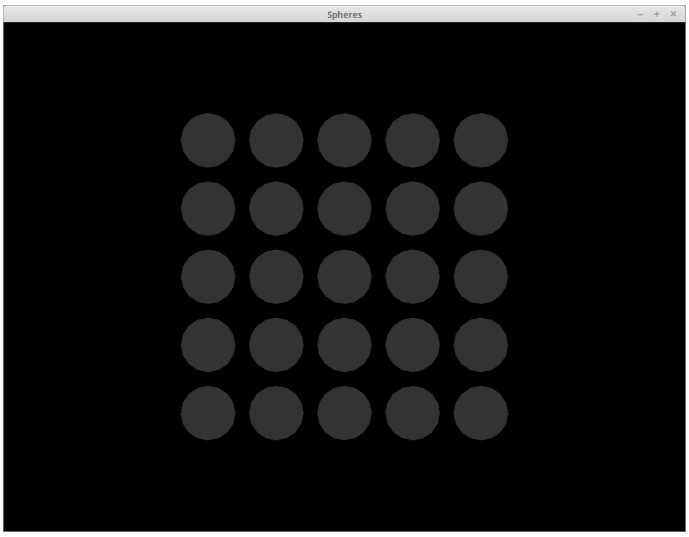
- DirectLight:

```
private void directLight(TransformGroup node) {
	BoundingSphere bounds = new BoundingSphere(new Point3d(0d,0.0d,0d),.5d); //
                                                        definizione del bound
	DirectionalLight lightD1 = new DirectionalLight(); //creazione di una luce
                                                        direzionale
	// Color3f green = new Color3f(0.0f, 1.0f, 0.0f);
	// lightP1.setColor(green); // coloro la luce
	lightD1.setInfluencingBounds(bounds);
	lightD1.setDirection(10, 10f, -10f); //definizione della direzione
	node.addChild(lightD1); //aggiunta al BG

}
```

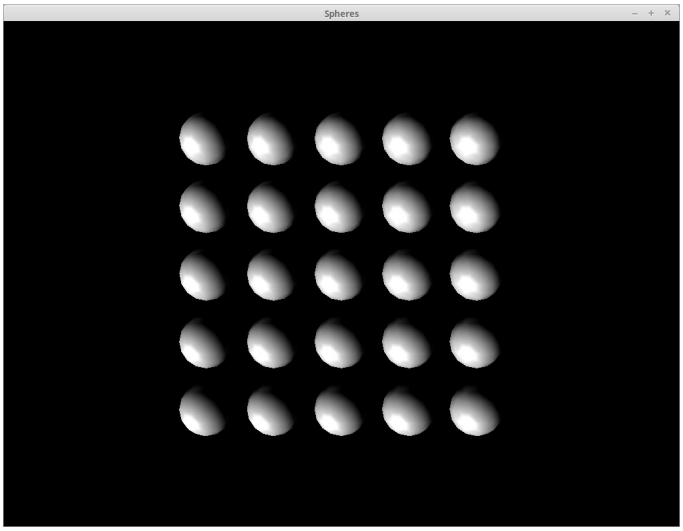
- CreateSeneGraph:

```
	public BranchGroup createSceneGraph() {
		/* inizializzo le variabili che mi serviranno */
		BranchGroup BG = new BranchGroup();
		TransformGroup transform;
		Transform3D t3d;
		Material material = new Material();
		Appearance app = new Appearance();
		app.setMaterial(material);

		/* creo la matrice di sfere */
		for(double i=-.4; i<.6; i+=.2) {
			for(double j= -.4; j<.6; j+=.2) {
				transform = new TransformGroup();
				t3d = new Transform3D();
				t3d.setTranslation(new Vector3d(i,j,0));
				transform.setTransform(t3d);
				transform.addChild(new Sphere(0.08f,
Primitive.GENERATE_NORMALS, app)); //aggiungo la sfera al TG
				BG.addChild(transform); //aggiunge l'oggetto TG come figlio del
BrachGroup
			}
		}

		transform= new TransformGroup();
		ambientLight(transform);
		directLight(transform);
		BG.addChild(transform);

		return BG;
	}
```

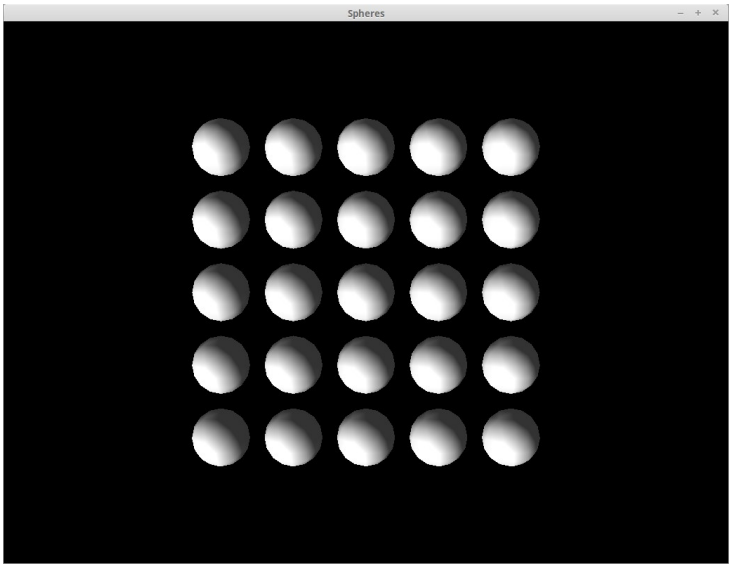
Output:



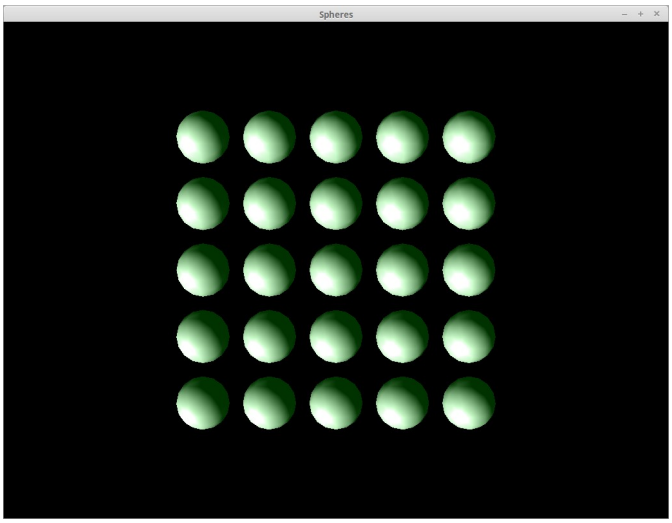
AmbientLight



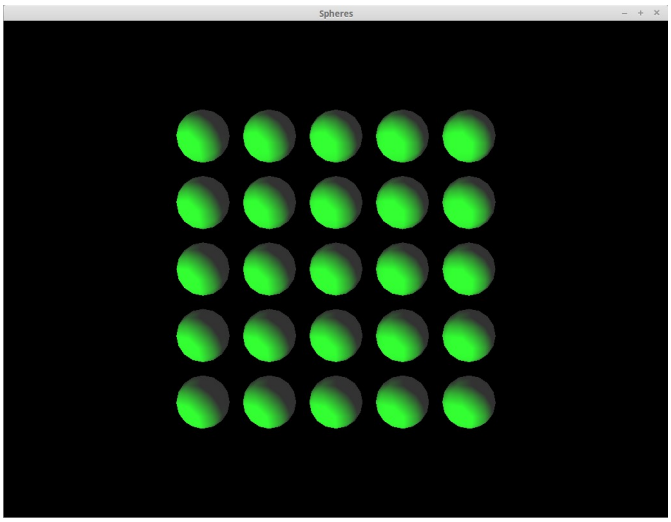
DirectLight



DirectLight & AmbientLight

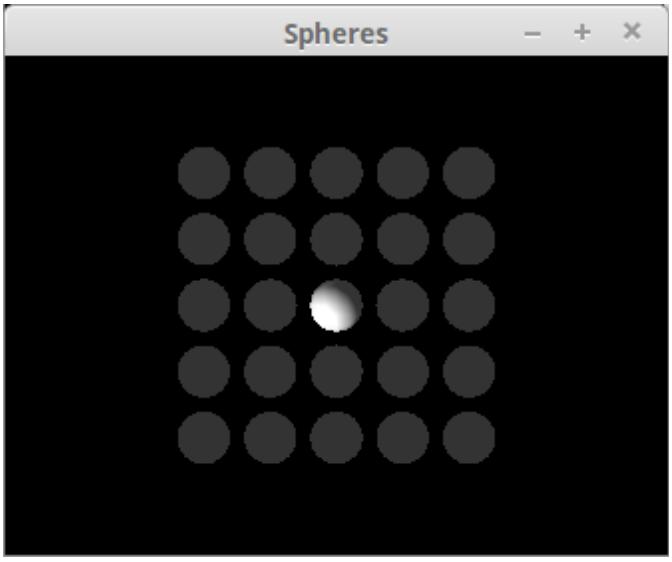


DirectLight - default - & AmbientLight - green

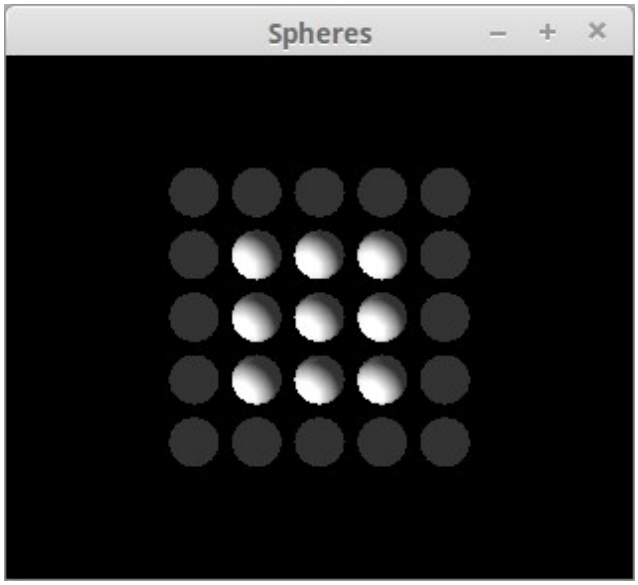


DirectLight - green - & AmbientLight - default

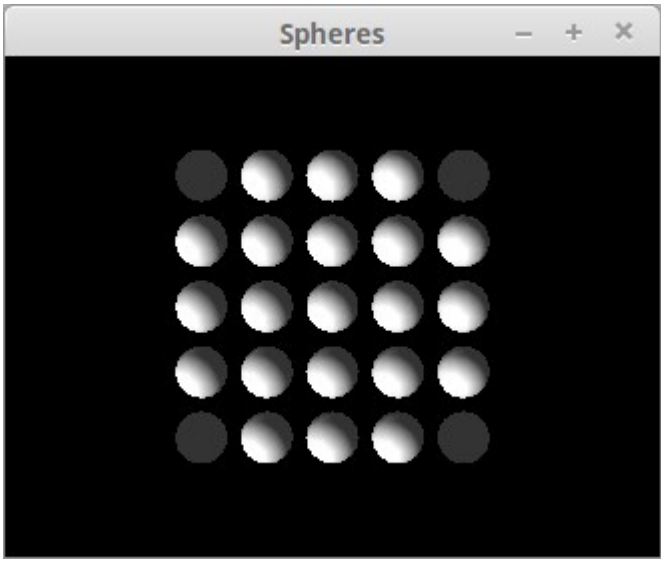
- Variazione dei bounds



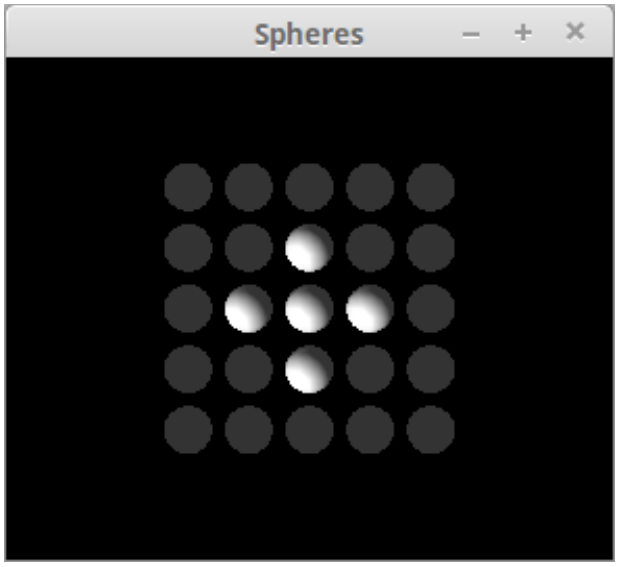
*new BoundingSphere(new Point3d(0d,0d,0d), **.1d**)*



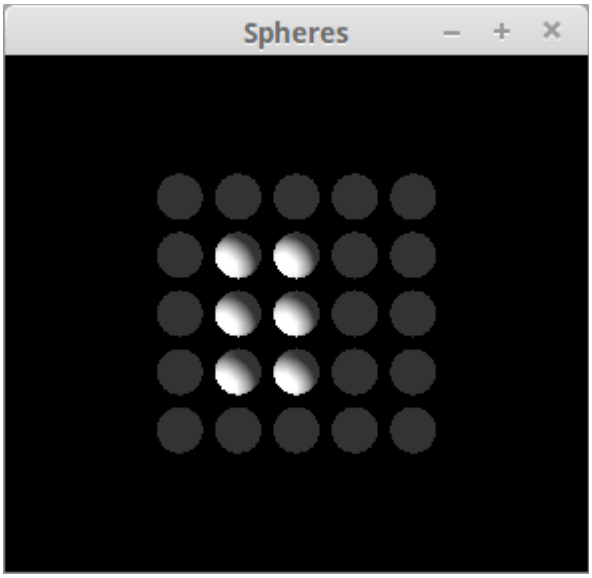
*new BoundingSphere(new Point3d(0d,0d,0d), **.2d**)*



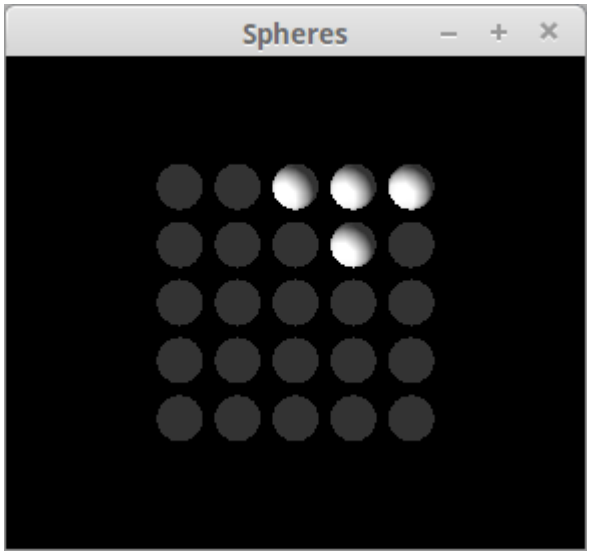
*new BoundingSphere(new Point3d(0d,0d,0d), **.4d**)*



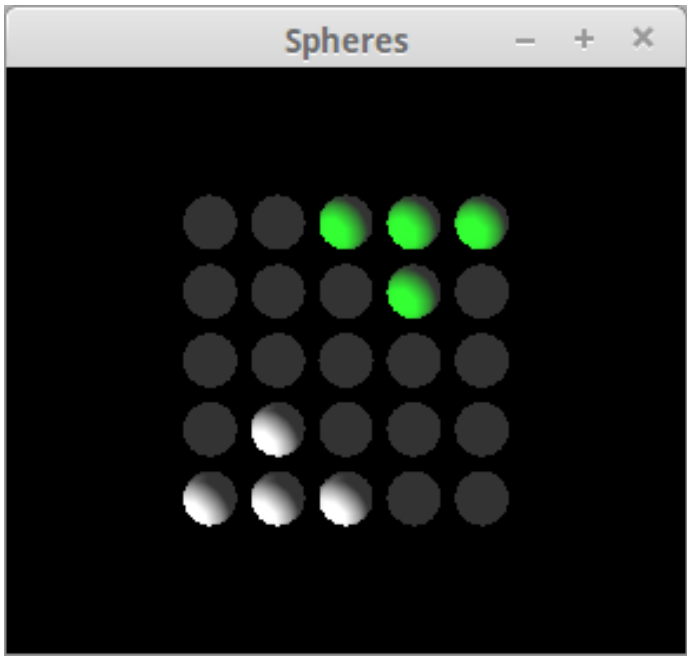
*new BoundingSphere(new Point3d(0d,0d,0d), **.15d**)*



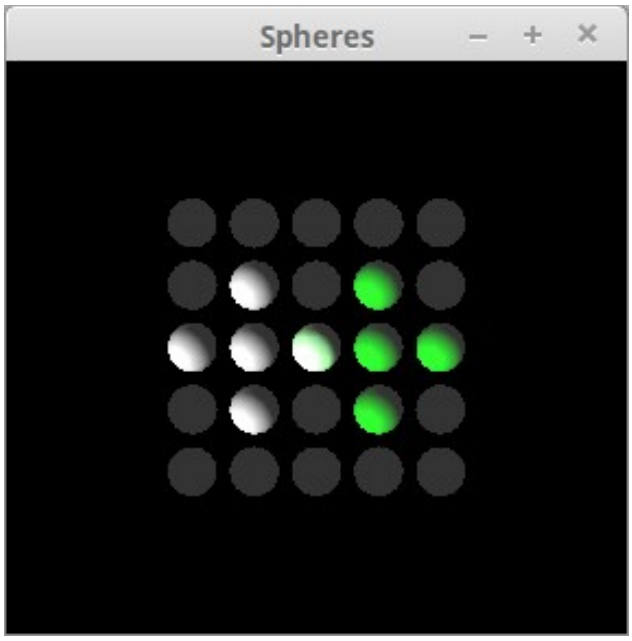
*new BoundingSphere(new Point3d(**-.1d,.0d,0d**), .15d)*



*new BoundingSphere(new Point3d(**.2d,.4d,0d**), .15d)*



Due DirectLight, una verde con centro (2,4) e una bianca con centro (-2,-4)



Due DirectLight, una verde con centro (2,0) e una bianca con centro (-2,0)

Osservazioni:
La luce ambientale è molto piatta, dimostra che ci sono degli oggetti nella scena ma non contribuisce in alcun modo a dare all'occhio umano l'illusione di tridimensionalità.

Il raggio della BoundingSphere determina quante sfere della matrice vengono illuminare, tutto ciò che è al di fuori non viene illuminato neanche parzialmente.

L'origine della BoundingSphere determina qual è il centro della sfera entro la quale gli oggetti della scena sono illuminati.

Si possono aggiungere più di una luce direzionale per evidenziare diversi oggetti all'interno della scena. Quando i bounds delle due luci si intersecano, il colore viene mischiato.

Esercizio 3.8

Realizzare la raffigurazione 3D di una colonna composta da:

- Fusto: basato su un cilindro (trascurando l'entasi, si può usare la Primitive apposita).

- Echino: un tronco di cono (si può ricavare dall'esempio di cilindro già fornito variando i raggi superiore ed inferiore).

- Abaco: un parallelepipedo.

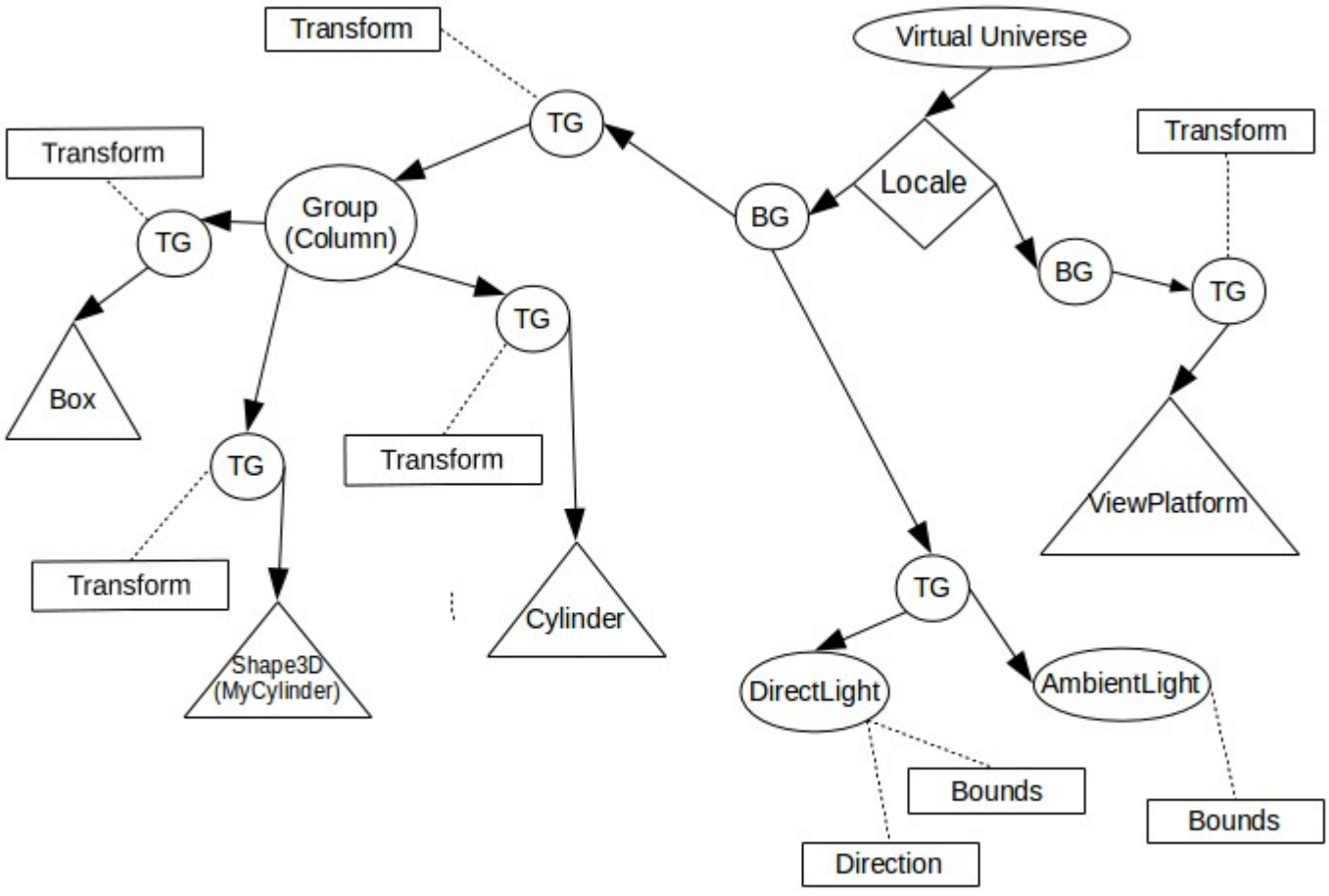
Fornire il tutto di un aspetto opportuno in modo da avere colore coerente con la pietra.

Il risultato deve essere incapsulato in una classe riutilizzabile.

È possibile scomporre in ulteriori classi e si consiglia di procedere in modo da poter collaudare i risultati ad ogni passo (primitive, geometrie, gruppi, aspetto, materiale)

(per la descrizione della colonna si faccia riferimento alle slide della lezione 3.5)

Scenegraph:



Codice:

- MyCylinder:

```
class MyCylinder extends Shape3D {

	protected float top = 1.0f;
	protected float bottom = -1.0f;
	protected Appearance appearance = new Appearance ( ) ;
	protected PolygonAttributes polyAttrbutes = new PolygonAttributes ( ) ;
	protected TriangleStripArray triangleStrip = null;
	protected Point3f v[] = null;

	public MyCylinder(int steps, float diameter, float height, Appearance app ) {

		float size = diameter;
		top = height/2;
		bottom = -height/2;
		appearance = app;

		/* creo il cilindro con le dimensioni in input */
		v = new Point3f[(steps+1)*2];
		for(int i=0; i<steps; i++) {
			/*l'angolo per il punto serve a suddividere la circonferenza in parti
uguali */
			double angle = 2.0*Math.PI*(double)i/(double)steps;
			float x = size*(float)Math.sin(angle);
			float y = size*(float)Math.cos(angle);
			v[i*2+0] = new Point3f(x*1.2f, top, y*1.2f);
			v[i*2+1] = new Point3f(x,bottom, y);
		}
		/* ultimo giro */
		v[steps*2+0] = new Point3f(0.0f ,top, size*1.2f);
		v[steps*2+1] = new Point3f(0.0f , bottom, size);
		int [ ] stripCounts ={( steps+1)*2};
		/* impostazioni per le texture e le luci */
		triangleStrip = new TriangleStripArray((steps+1)*2,
GeometryArray.COORDINATES | GeometryArray.NORMALS |
GeometryArray.TEXTURE_COORDINATE_2,stripCounts );
		triangleStrip.setCoordinates(0,v);
		/* disabilita lo scartare di alcune facce */
		polyAttrbutes.setCullFace(PolygonAttributes.CULL_NONE ) ;
		appearance.setPolygonAttributes (polyAttrbutes) ;
		GeometryInfo info = new GeometryInfo(triangleStrip);
		NormalGenerator ng = new NormalGenerator(); // per le luci
		ng.generateNormals(info);
		Geometry geom = info.getGeometryArray();
		/* per le texture*/
		TexCoordGeneration textureCoordGeneration = new
TexCoordGeneration( TexCoordGeneration.OBJECT_LINEAR,
TexCoordGeneration.TEXTURE_COORDINATE_2);
		appearance.setTexCoordGeneration(textureCoordGeneration);

		setGeometry(geom);
		setAppearance ( appearance ) ;
	}
}
```

- Column:

```
public class Column extends Group {

	static final protected Appearance appearance = new Appearance() ;

	static final protected Transform3D cylinderTransform = new Transform3D ();
	static final protected Transform3D coneTransform = new Transform3D ();
	static final protected Transform3D boxTransform = new Transform3D ();
	protected TransformGroup cylinderTG = new TransformGroup(cylinderTransform);
	protected TransformGroup coneTG = new TransformGroup(coneTransform);
	protected TransformGroup boxTG = new TransformGroup(boxTransform);
	protected Appearance app = new Appearance();
	int primflags = Primitive.GENERATE_NORMALS + Primitive.GENERATE_TEXTURE_COORDS;
	protected Cylinder cylinder = new Cylinder(.1f, .6f, primflags, app);
	protected MyCylinder cone = new MyCylinder(40, 1f, 2f, app);
	protected Box box = new Box(.15f,.02f,.15f, primflags,  app);


	public Column(float diameter, float height, Appearance appearance) {
		app = appearance;

		/* inizializzazione delle parti */
		cylinder = new Cylinder(diameter, height, primflags, app);
		cone = new MyCylinder(20, diameter, height/6, app);
		box = new Box(diameter+0.5f*diameter,.2f*diameter, diameter+0.5f*diameter,
primflags,  app);

		/* echino */
		coneTransform.setTranslation(new Vector3d(0,height+height/12,0));
		coneTG = new TransformGroup(coneTransform);

		/* cilindro */
		cylinderTransform.setTranslation(new Vector3d(0,height/2, 0));
		cylinderTG = new TransformGroup(cylinderTransform);

		/* box */
		boxTransform.setTranslation(new Vector3d(0,height+height/6, 0));
		boxTG = new TransformGroup(boxTransform);

		/* imposto l'appearance */
		cylinder.setAppearance ( app) ;
		cone . setAppearance ( app ) ;
		box.setAppearance ( app) ;
		/* aggiungo al TransformGroup */
		cylinderTG.addChild (cylinder) ;
		coneTG.addChild (cone) ;
		boxTG.addChild (box) ;
		/* aggiungo al Group */
		addChild(cylinderTG) ;
		addChild(coneTG ) ;
		addChild(boxTG);

	}

}
```

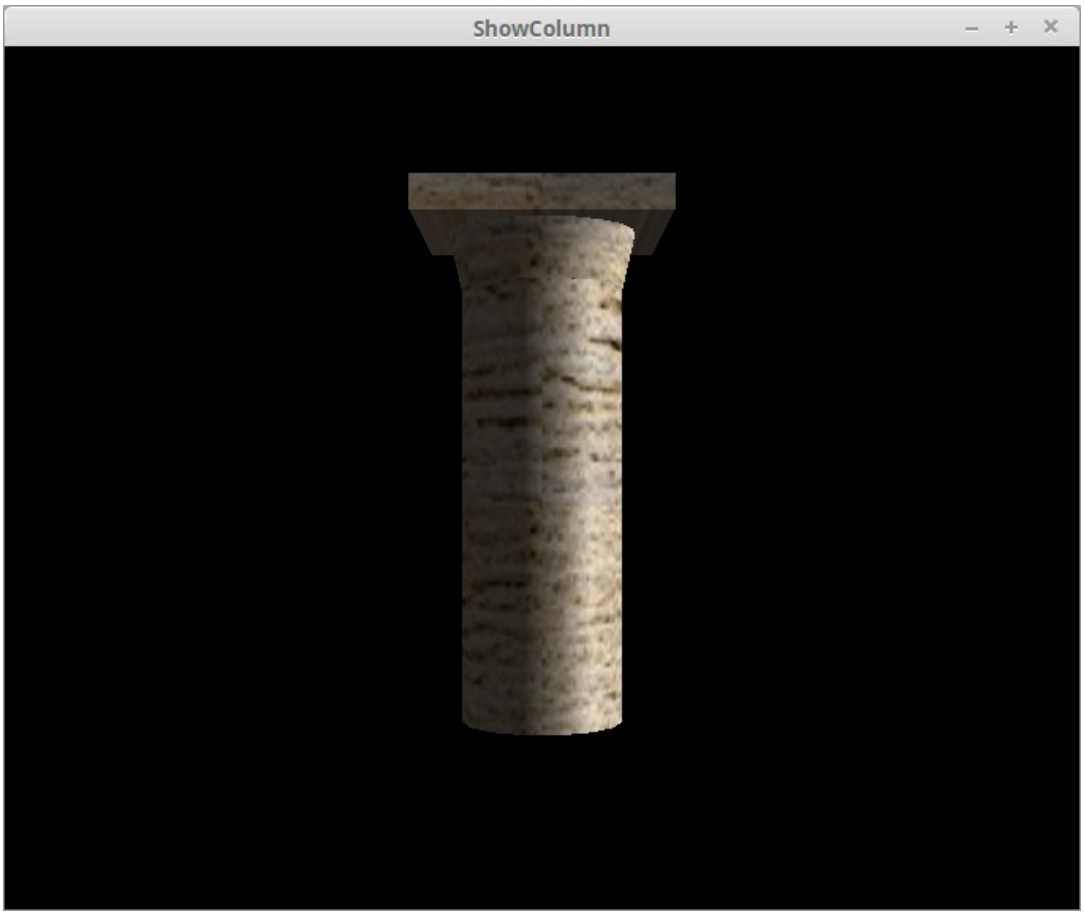
- ShowColumn – setTexture:

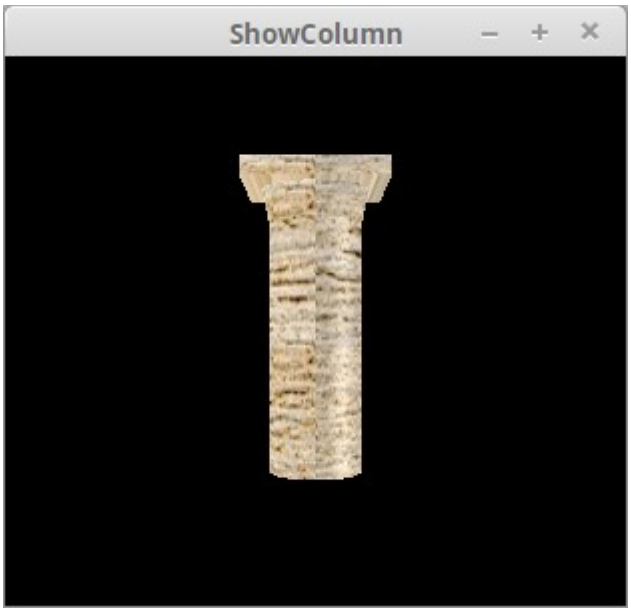
```
    public void setTexture(Appearance app, String path) {
          TextureLoader loader = new TextureLoader(path, null);

          Texture texture = loader.getTexture();
          /*comportamento ai bordi */
          texture.setBoundaryModeS(Texture.WRAP);
          texture.setBoundaryModeT(Texture.WRAP);

        TextureAttributes texAttr = new TextureAttributes();
        texAttr.setTextureMode(TextureAttributes.REPLACE  ); // il colore del
materiale è modulato con quello della texture

        app.setTexture(texture);
        app.setTextureAttributes(texAttr);
    }
```

- ShowColumn – createSceneGraph:

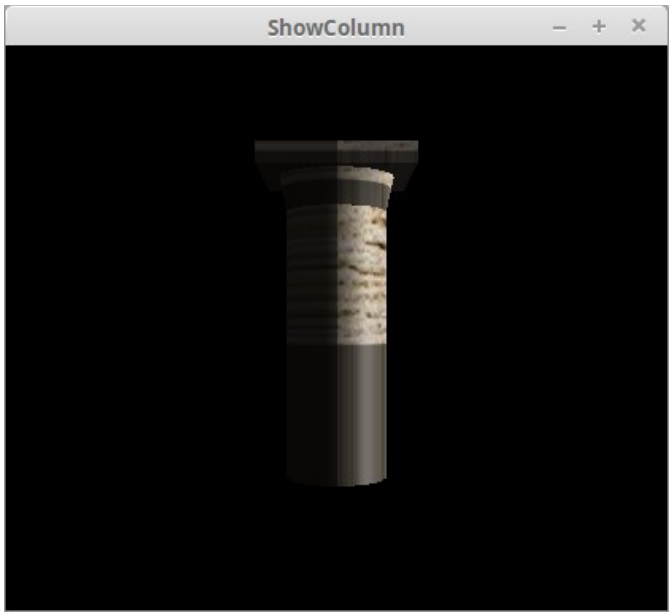
```
          Appearance app = new Appearance();
          Material m = new Material();
          app.setMaterial(m);
          /* imposto la texture */
          setTexture(app, "./textures/PietraColonna.jpg");

          Column column = new Column(.1f,.6f,app); // inizializzo la colonna

          TG.addChild(column);
          objRoot.addChild(TG);
```

Output:





TextureAttributes: REPLACE



BoundaryMode: CLAMP

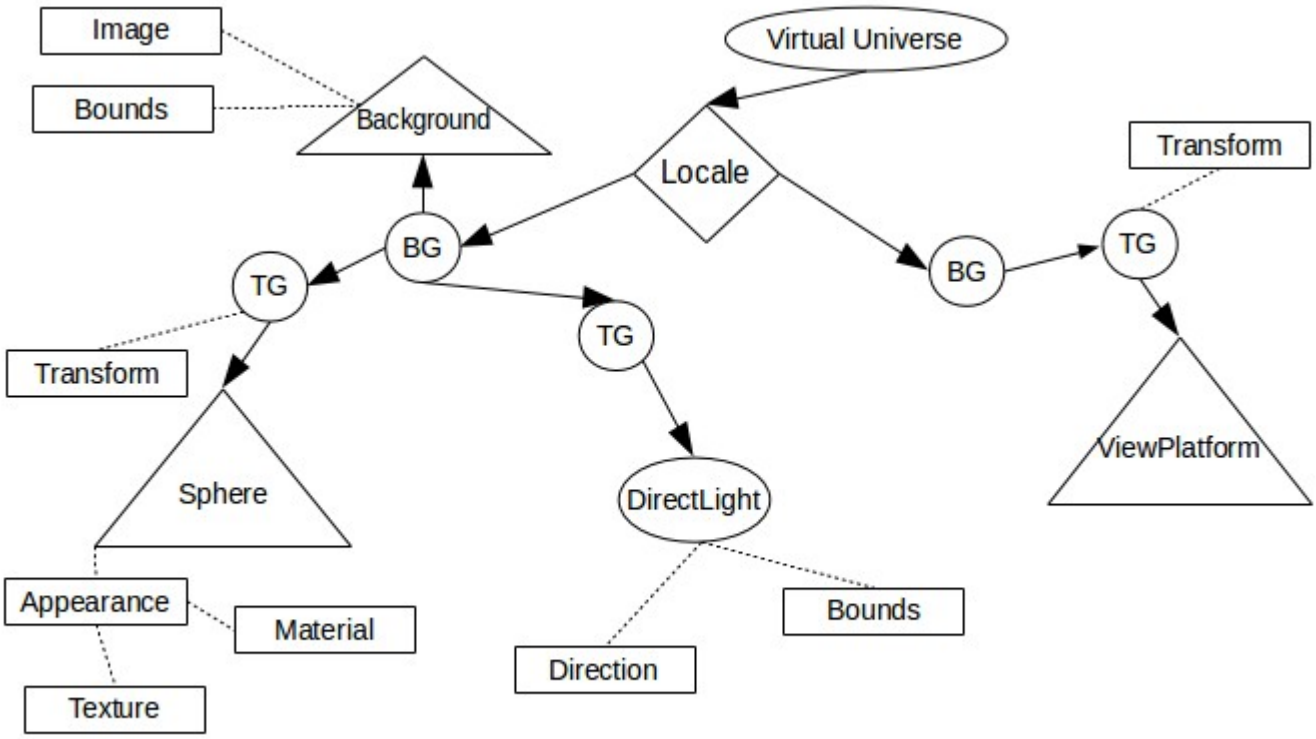
Osservazioni:
In questo esercizio abbiamo creato un tronco di cono tramite GeometryArray e l'abbiamo unito a un Box e a un Cylinder in un Group per formare una colonna.

Abbiamo inoltre imparato ad usare le textures, le quali per essere utilizzate hanno bisogno di avere delle coordinate. In più, perché la luce abbia effetto, deve esserci sotto la texture un materiale e la texture non deve sostituirlo. Abbiamo infine imparato a gestirne i comportamenti ai bordi della texture.

Esercizio 4.1

Alla “Terra” aggiungere una luce direzionale per creare l'effetto dell'illuminazione solare.

Scenegraph:



Codice:

- CreateSceneGraph:

```
	public BranchGroup createSceneGraph() {
		//Crea la radice del branch graph
		BranchGroup objRoot=new BranchGroup();

		//Texture per simulare la Terra
		Appearance appearance = new Appearance ( ) ;
		//Caricamento della texture da file.
		TextureLoader textureLoader = new TextureLoader("./textures/earth.jpg",
null);
		// Inizializzazione dell ’oggetto Texture.
		Texture texture = textureLoader.getTexture();

		TextureAttributes texAttr = new TextureAttributes();
		texAttr.setTextureMode(TextureAttributes.MODULATE);

		// Impostazione dell’aspetto .
		Material m = new Material();
		appearance.setMaterial(m);

		appearance.setTexture(texture);
		appearance.setTextureAttributes(texAttr);
		// Creazione di una primitiva completa di
		// coordinate per la texture.
		int primflags=Primitive.GENERATE_NORMALS+Primitive.GENERATE_TEXTURE_COORDS;
		Sphere earth = new Sphere( 1.0f, primflags, 40, appearance);

		// Imposto il background
		objRoot.addChild(getBackground("./textures/stars.jpg"));
```

```
        TransformGroup TG = new TransformGroup();
        // Aggiungo la luce diretta per simulare il sole
        directLight(TG);

        TG.addChild(earth);
        objRoot.addChild(TG);

        return objRoot;


    }
```

- getBackground:

```
        private Background getBackground(String path) {
                    //Inizializza la texture per lo sfondo
                    TextureLoader myLoader = new TextureLoader( path , this );
                ImageComponent2D myImage = myLoader.getImage( );
                //Inizializza il Background
                Background myBack = new Background( );

                //Applica la texture sullo sfondo
                myBack.setImage( myImage );
                //Deve coprire tutto lo sfondo
                myBack.setImageScaleMode(Background.SCALE_FIT_MAX);
                //Fino a che limiti si deve estendere
                BoundingSphere myBounds = new BoundingSphere(new Point3d( ),
                                                                1000.0 );
                myBack.setApplicationBounds( myBounds );

                return myBack;
        }
```

- directLight:

```
        private void directLight(TransformGroup node) {
                    // creazione del bound
                    BoundingSphere bounds = new BoundingSphere(new
                    Point3d(0d,0.0d,0d),.1d);
                    // creazione di una luce direzionale
                    DirectionalLight lightD1 = new DirectionalLight();
                    // impostazione del bound
                    lightD1.setInfluencingBounds(bounds);
                    lightD1.setDirection(-1f, -1f, 0f);
                    // aggiunta al TransformGroup
                    node.addChild(lightD1);

        }
```

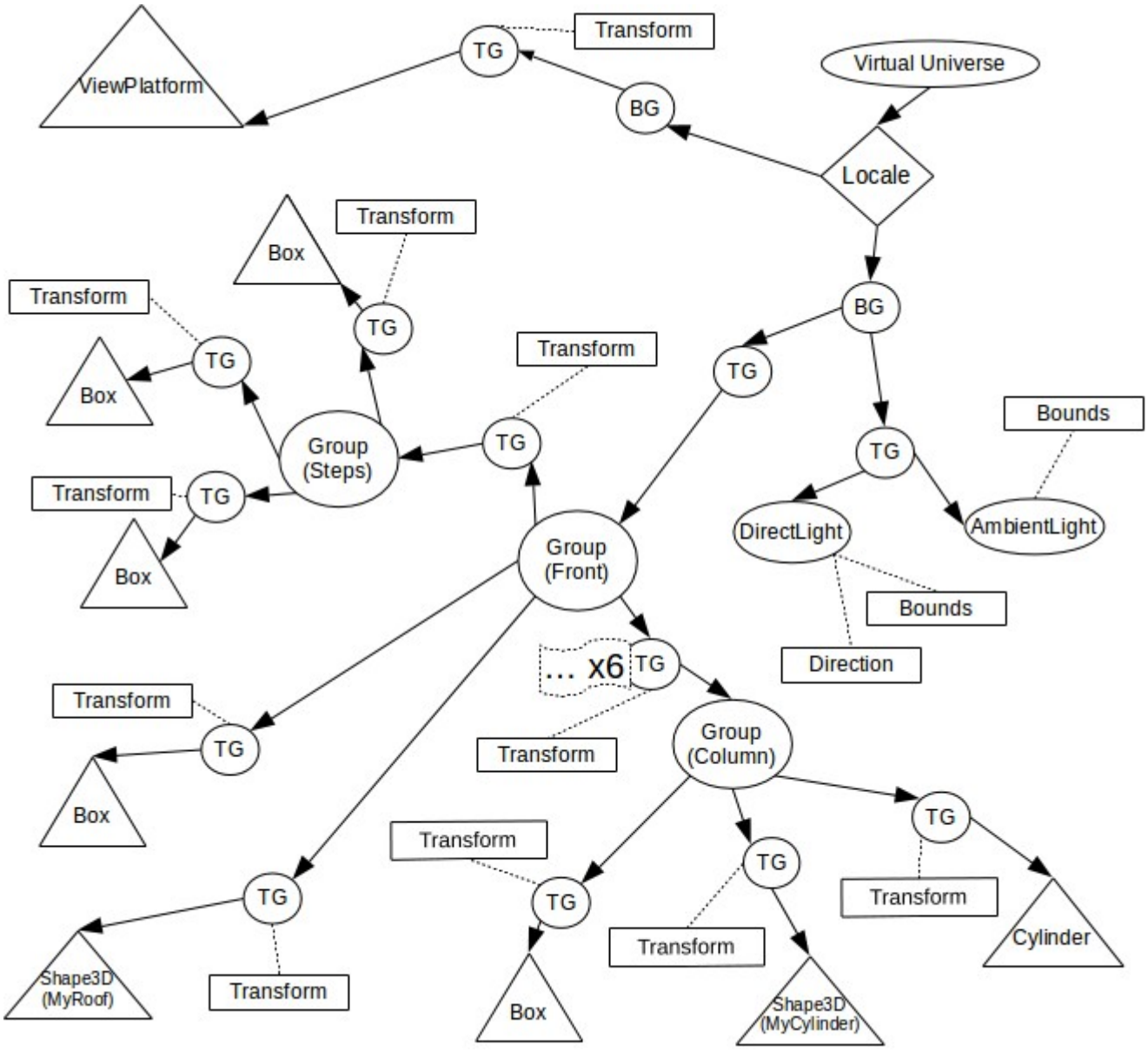
Output:

Esercizio 4.2

- Applicare una texture agli elementi della colonna dorica creata precedentemente.

- Riprodurre la facciata del tempio di Poseidone di Pæstum.

Scenegraph:



Codice:

- CreateSceneGraph:

```
// Crea la facciata
FrontPaestum face = new FrontPaestum();
TG.addChild(face);
objRoot.addChild(TG);
```

- FrontPaestum extends Group:

```
public class FrontPaestum extends Group {
      protected int numColumns = 6;

      protected Transform3D t3dBase = new Transform3D();
      protected Transform3D t3dRoof = new Transform3D();
```

```
	protected Transform3D t3dRoofSupport = new Transform3D();
	protected Transform3D[] t3dColumns = new Transform3D[numColumns];

	protected TransformGroup TGBase = new TransformGroup(t3dBase);
	protected TransformGroup TGRoof = new TransformGroup(t3dRoof);
	protected TransformGroup TGRoofSupport = new
TransformGroup(t3dRoofSupport);
	protected TransformGroup[] TGColumns = new TransformGroup[numColumns];

	protected Appearance app = new Appearance();
	int primflags = Primitive.GENERATE_NORMALS +
Primitive.GENERATE_TEXTURE_COORDS;

	protected Steps base = new Steps(3,.7f, .05f, .15f, app);
	protected Column[] columns = new Column[numColumns];
	protected MyRoof roof = new MyRoof(1.35f, .25f, app);
	protected Box roofSupport = new Box(.62f, .07f, .03f, primflags, app);

	public FrontPaestum() {

		t3dBase.setTranslation(new Vector3d(0,-.4,.05));
		TGBase.setTransform(t3dBase);
		TGBase.addChild(base);

		addChild(TGBase);


		float xAxis = 3*.05f-.7f;
		for(int i=0; i<numColumns; i++) {
			columns[i] = new Column(.05f, .3f, app);
			t3dColumns[i] = new Transform3D();
			TGColumns[i] = new TransformGroup();
			t3dColumns[i].setTranslation(new Vector3d(xAxis, -.28, 0));

			TGColumns[i].setTransform(t3dColumns[i]);
			TGColumns[i].addChild(columns[i]);

			addChild(TGColumns[i]);

			xAxis+=.22f;
		}

		t3dRoofSupport.setTranslation(new Vector3d(0,-.28+.3f+.125, 0));
		TGRoofSupport.setTransform(t3dRoofSupport);
		TGRoofSupport.addChild(roofSupport);

		addChild(TGRoofSupport);

		t3dRoof.setTranslation(new Vector3d(0,-.28+.3f+.125+.195, 0));
		TGRoof.setTransform(t3dRoof);
		TGRoof.addChild(roof);

		addChild(TGRoof);

		Material material = new Material();
		material.setShininess(128f);
		app.setMaterial(material);

		// Imposta la texture
		TextureLoader loader = new
TextureLoader("./textures/PietraColonna.jpg",
			null);
		Texture texture = loader.getTexture();
		texture.setBoundaryModeS(Texture.WRAP);
	    texture.setBoundaryModeT(Texture.WRAP);
	    texture.setBoundaryColor( new Color4f( 0.0f, 1.0f, 0.0f, 0.0f ) );
```

```
            // Imposta gli attributi della texture
                TextureAttributes texAttr = new TextureAttributes();
                texAttr.setTextureMode(TextureAttributes.MODULATE);
                app.setTexture(texture);
                app.setTextureAttributes(texAttr);

        }

    }
```

- MyRoof extends Shape3D:

```
    class MyRoof extends Shape3D {

        protected float roofHeight = .3f;
        protected float roofLength = .8f;
        protected float thickness= .05f;
        protected float border= .015f;

        protected Appearance appearance = new Appearance ( ) ;
        protected PolygonAttributes polyAttrbutes = new PolygonAttributes ( ) ;
        protected TriangleStripArray triangleStrip = null;
        protected Point3f v[] = null;

        public MyRoof(float length, float height, Appearance app) {
                roofLength = length;
                roofHeight = height;
                appearance = app;

                int size = 27;

                Point3f p1 = new Point3f(0,roofHeight/2,thickness);
                Point3f p2 = new Point3f(-roofLength/2,-roofHeight/2,thickness);
                Point3f p3 = new Point3f(roofLength/2,-roofHeight/2,thickness);
                Point3f p4 = new Point3f(0,roofHeight/2-border*1.5f,thickness);
                Point3f p5 = new Point3f(-roofLength/2+border*5f,-
    roofHeight/2+border,thickness);
                Point3f p6 = new Point3f(roofLength/2-border*5f,-
    roofHeight/2+border,thickness);
                Point3f p7 = new Point3f(0,roofHeight/2,-thickness);
                Point3f p8 = new Point3f(-roofLength/2,-roofHeight/2,-thickness);
                Point3f p9 = new Point3f(roofLength/2,-roofHeight/2,-thickness);
                Point3f p10 = new Point3f(0,roofHeight/2-border*1.5f,-thickness/2);
                Point3f p11 = new Point3f(-roofLength/2+border*5f,-
    roofHeight/2+border,-thickness/2);
                Point3f p12 = new Point3f(roofLength/2-border*5f,-
    roofHeight/2+border,-thickness/2);


                v = new Point3f[size];

                v[0] = (p7);
                v[1] = (p8);
                v[2] = (p9);
                v[3] = (p7);
                v[4] = (p1);
                v[5] = (p8);
                v[6] = (p2);
                v[7] = (p9);
                v[8] = (p3);
                v[9] = (p1);
                v[10] = (p4);
                v[11] = (p2);
                v[12] = (p5);
                v[13] = (p3);
```

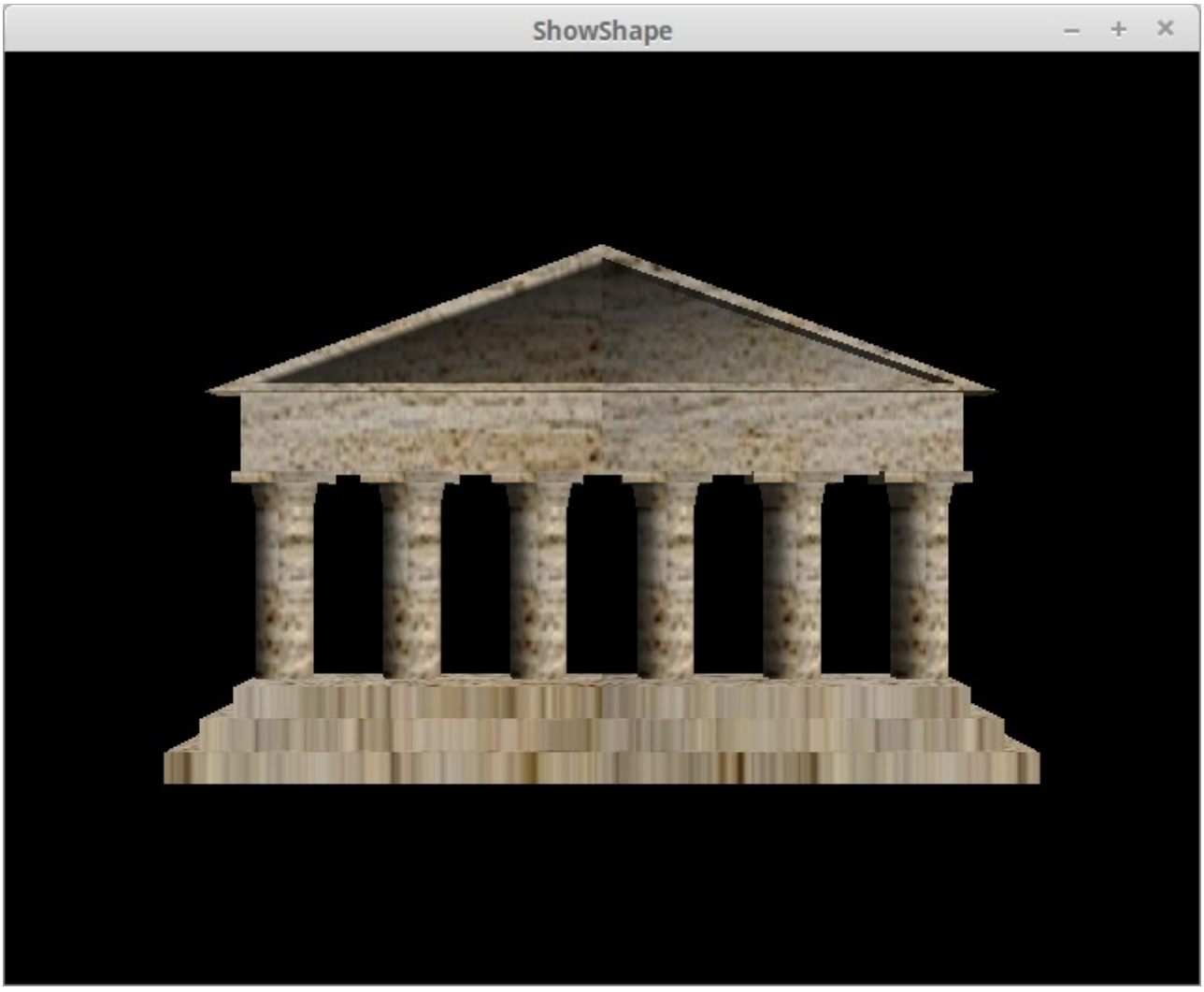
```
            v[14] = (p6);
            v[15] = (p4);
            v[16] = (p4);
            v[17] = (p5);
            v[18] = (p10);
            v[19] = (p11);
            v[20] = (p12);
            v[21] = (p5);
            v[22] = (p6);
            v[23] = (p6);
            v[24] = (p12);
            v[25] = (p4);
            v[26] = (p10);

            int [ ] stripCounts ={size};
            triangleStrip = new TriangleStripArray(size,
                        GeometryArray.COORDINATES | GeometryArray.NORMALS |
GeometryArray.TEXTURE_COORDINATE_2,stripCounts );
            triangleStrip.setCoordinates(0,v);

            polyAttrbutes.setCullFace(PolygonAttributes.CULL_NONE ) ;
            appearance.setPolygonAttributes (polyAttrbutes) ;
            GeometryInfo info = new GeometryInfo(triangleStrip);
            NormalGenerator ng = new NormalGenerator();
            ng.generateNormals(info);
            Geometry geom = info.getGeometryArray();

            TexCoordGeneration textureCoordGeneration = new
TexCoordGeneration( TexCoordGeneration.OBJECT_LINEAR,
TexCoordGeneration.TEXTURE_COORDINATE_2);
            appearance.setTexCoordGeneration(textureCoordGeneration);

            setGeometry(geom);
            setAppearance ( appearance ) ;
      }
}
```

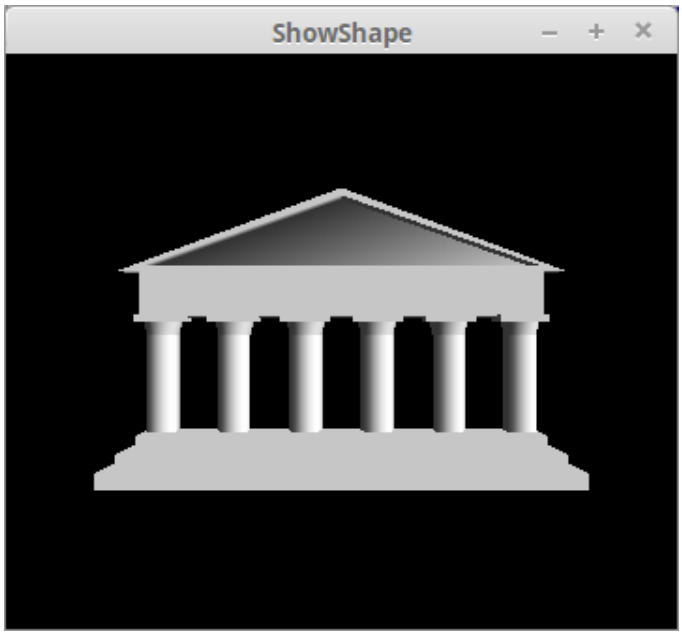
- Steps extends Group:

```
public class Steps extends Group {
      protected int stepsNumber = 3;
      protected float baseLength = .8f;
      protected float stepHeight = .1f;
      protected float baseDeep = .2f;

      int primflags = Primitive.GENERATE_NORMALS +
Primitive.GENERATE_TEXTURE_COORDS;
      protected Box[] steps = new Box[stepsNumber];


      public Steps(int stepsN, float lengthIn, float height, float deep,
Appearance app){
            stepsNumber = stepsN;
            baseLength = lengthIn;
            stepHeight = height;
            baseDeep = deep;

            steps = new Box[stepsNumber];
            TransformGroup[] TG = new TransformGroup[stepsNumber];
            Transform3D[] t3d = new Transform3D[stepsNumber];
            float yOffset = 0;
            float length = baseLength;
            for(int i=0; i<stepsNumber; i++) {
                  steps[i] = new Box(length, baseDeep-yOffset/2, stepHeight/2,
primflags, app);
```

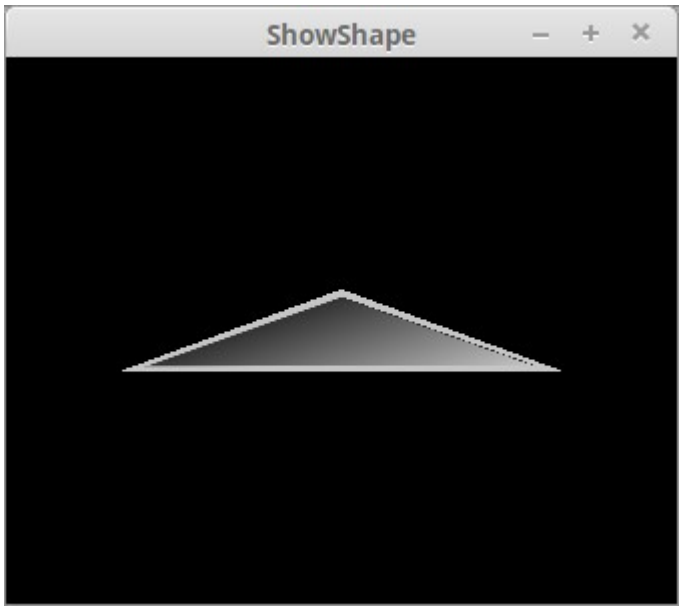
```
                    t3d[i] = new Transform3D();
                    TG[i] = new TransformGroup();
                    t3d[i].rotX(-Math.PI/2);
                    t3d[i].setTranslation(new Vector3d(0,yOffset,0));
                    TG[i].setTransform(t3d[i]);
                    TG[i].addChild(steps[i]);
                    addChild(TG[i]);

                    length-=stepHeight;
                    yOffset+=stepHeight;

                }

            }

        }
```

Output:

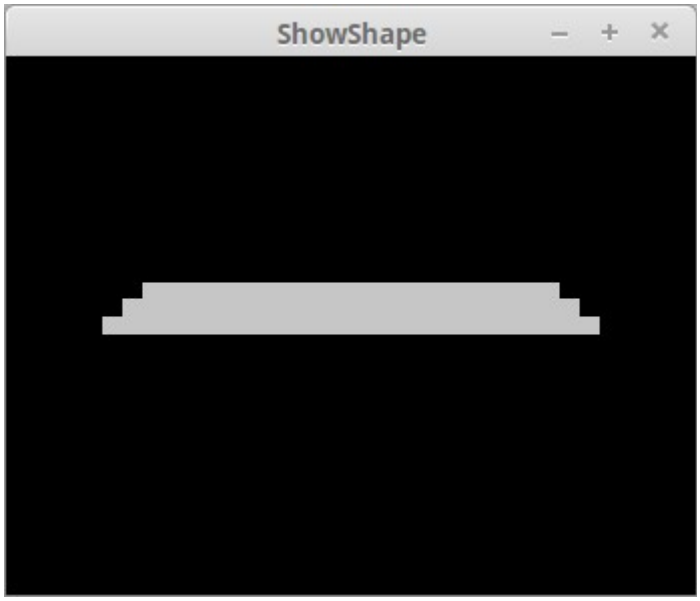




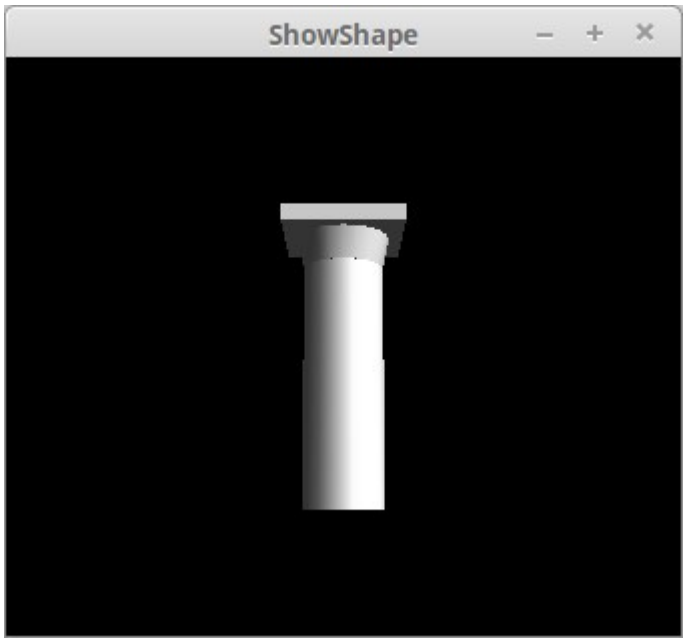
Senza texture



MyRoof



Steps



Column

Osservazioni:
Tramite l'unione di forme personalizzate e gruppi si può creare oggetti sempre più complessi mantenendo il codice pulito e comprensibile, scomponendo l'oggetto nelle sue parti principali, a loro volta scomposte in forme geometriche di base.

Esercizio 4.3

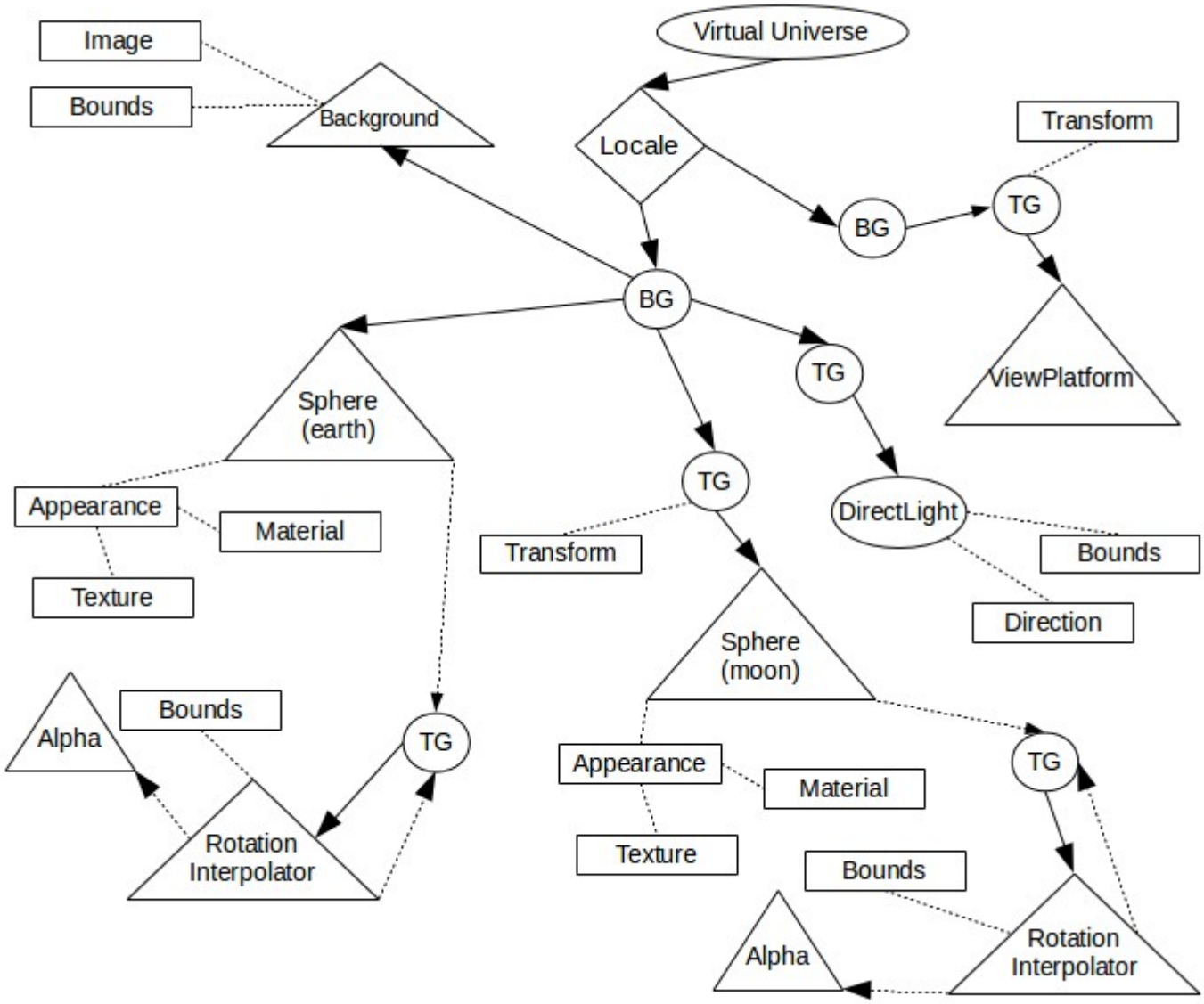
Riprendendo l'esercizio riproduzione della Terra:

- Aggiungere la Luna (texture fornita sul sito di e-learning).

- Trascurando la rotazione della Terra attorno al Sole, far orbitare la Luna intorno alla Terra.

Si può assumere che l'eclittica sia sullo stesso piano dell'equatore e non è necessario rispettare la scala nella distanza orbitale lunare.

Scenegraph:

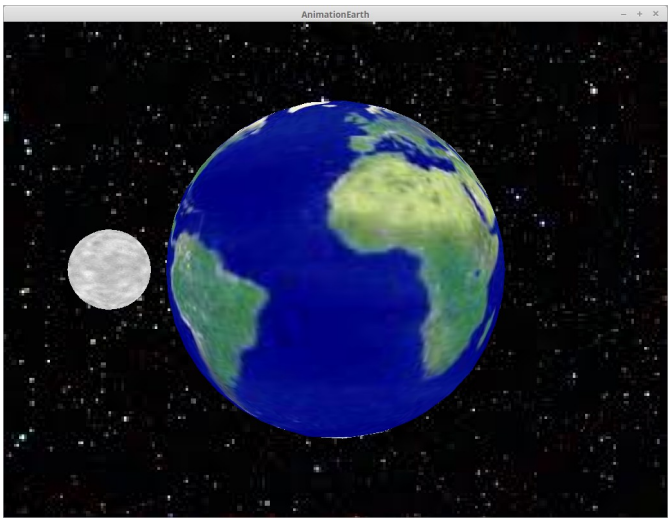


Codice:

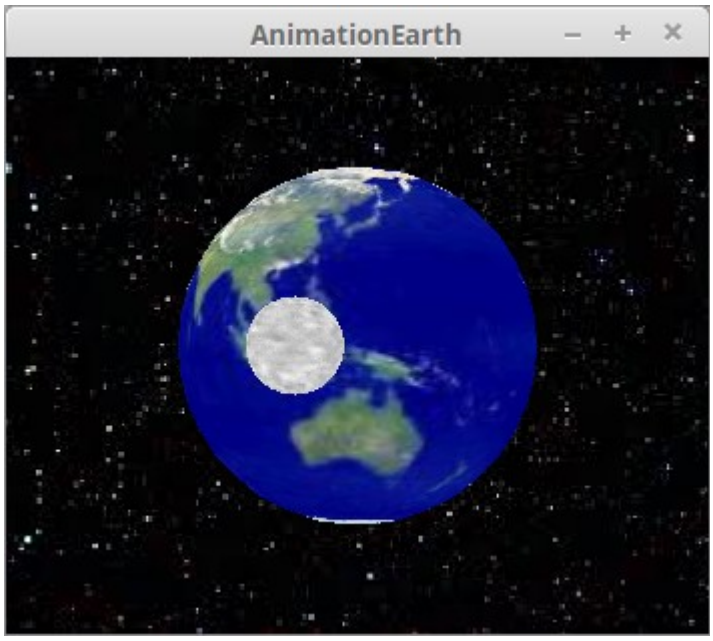
```
public class AnimationEarth extends Applet{

	public AnimationEarth() {
		setLayout(new BorderLayout());
		GraphicsConfiguration config =
SimpleUniverse.getPreferredConfiguration();
		Canvas3D canvas3D = new Canvas3D(config);
		add("Center", canvas3D);
		BranchGroup scene = createSceneGraph();
		scene.compile();

		SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
		simpleU.getViewingPlatform().setNominalViewingTransform();
		simpleU.addBranchGraph(scene);
	}


	//Crea la scena
	private BranchGroup createSceneGraph() {

		//Crea la radice del branch graph
		BranchGroup objRoot=new BranchGroup();

		//texture per simulare la Terra
		Appearance appearance = new Appearance ( ) ;
		//Caricamento della texture da file.
		TextureLoader textureLoader = new
TextureLoader("./textures/earth.jpg", null);
		// Inizializzazione dell 'oggetto Texture.
		Texture texture = textureLoader.getTexture();
		// Impostazione dell'aspetto .
		appearance.setTexture(texture);
		// Creazione di una primitiva completa di
		// coordinate per la texture.
		Sphere earth = new Sphere( .5f, Primitive.GENERATE_TEXTURE_COORDS,
40, appearance);

		// Gruppo di trasformazione per la luna
		Transform3D t3dmoon = new Transform3D();
		t3dmoon.setTranslation(new Vector3d(-.7,0,0));
		TransformGroup TGmoon = new TransformGroup(t3dmoon);


		//texture per fare una Luna
		Appearance appMoon = new Appearance ( ) ;
		//Caricamento della texture da file.
		TextureLoader loaderMoon = new TextureLoader("./textures/moon.jpg",
null);
		// Inizializzazione dell 'oggetto Texture.
		Texture texMoon = loaderMoon.getTexture();
		// Impostazione dell'aspetto .
		appMoon.setTexture(texMoon);
		// Creazione di una primitiva completa di
		// coordinate per la texture.
		Sphere moon = new Sphere( .1f, Primitive.GENERATE_TEXTURE_COORDS, 40,
appMoon);

		TGmoon.addChild(moon);

		// Imposto il background
		objRoot.addChild(getBackground("./textures/stars.jpg"));
```

```
            // Imposto le rotazioni
            objRoot.addChild(setRotatorInterpolator(TGmoon, -1, 30000));
            objRoot.addChild(setRotatorInterpolator(earth, -1, 1000));

            return objRoot;
      }

      private Background getBackground(String path) {
            //Inizializza la texture per lo sfondo
            TextureLoader myLoader = new TextureLoader( path , this );
            ImageComponent2D myImage = myLoader.getImage( );
            //Inizializza il Background
            Background myBack = new Background( );

            //Applica la texture sullo sfondo
            myBack.setImage( myImage );
            //Deve coprire tutto lo sfondo
            myBack.setImageScaleMode(Background.SCALE_FIT_MAX);
            //Fino a che limiti si deve estendere
            BoundingSphere myBounds = new BoundingSphere(new Point3d( ),
1000.0 );
            myBack.setApplicationBounds( myBounds );

            return myBack;
      }

      private TransformGroup setRotatorInterpolator(Group node, int loopCount,
long timerInterval ) {
            //Crea un gruppo per le trasformazioni affini della terra
            TransformGroup objRotate=new TransformGroup();
            objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
            objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
            //Aggiunge al gruppo un
            objRotate.addChild(node);

            //Crea un timer per la rotazione della terra
            Alpha rotationAlpha=new Alpha(loopCount,timerInterval);
            //Crea un interpolatore per le rotazioni collegato con il gruppo di
trasformazione
            RotationInterpolator rotator=new
RotationInterpolator(rotationAlpha,objRotate);
            //Imposta un raggio d'azione all'interpolatore
            BoundingSphere bounds=new BoundingSphere();
            rotator.setSchedulingBounds(bounds);
            //aggiunge l'interpolatore alla gruppo di trasformazione
            objRotate.addChild(rotator);

            return objRotate;
      }

      public static void main(String[] args) {
            new MainFrame(new AnimationEarth(), 1024, 768);
      }

}
```

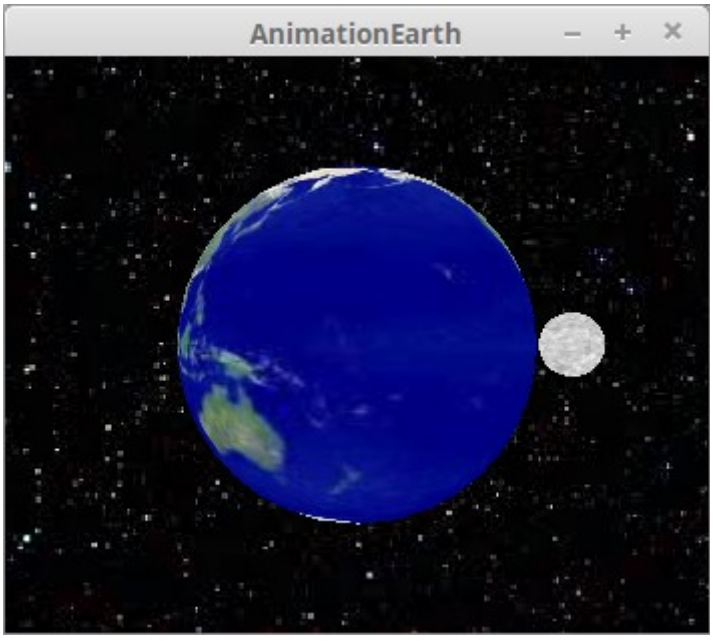
Output:



Situazione iniziale



Un frame



Un altro frame

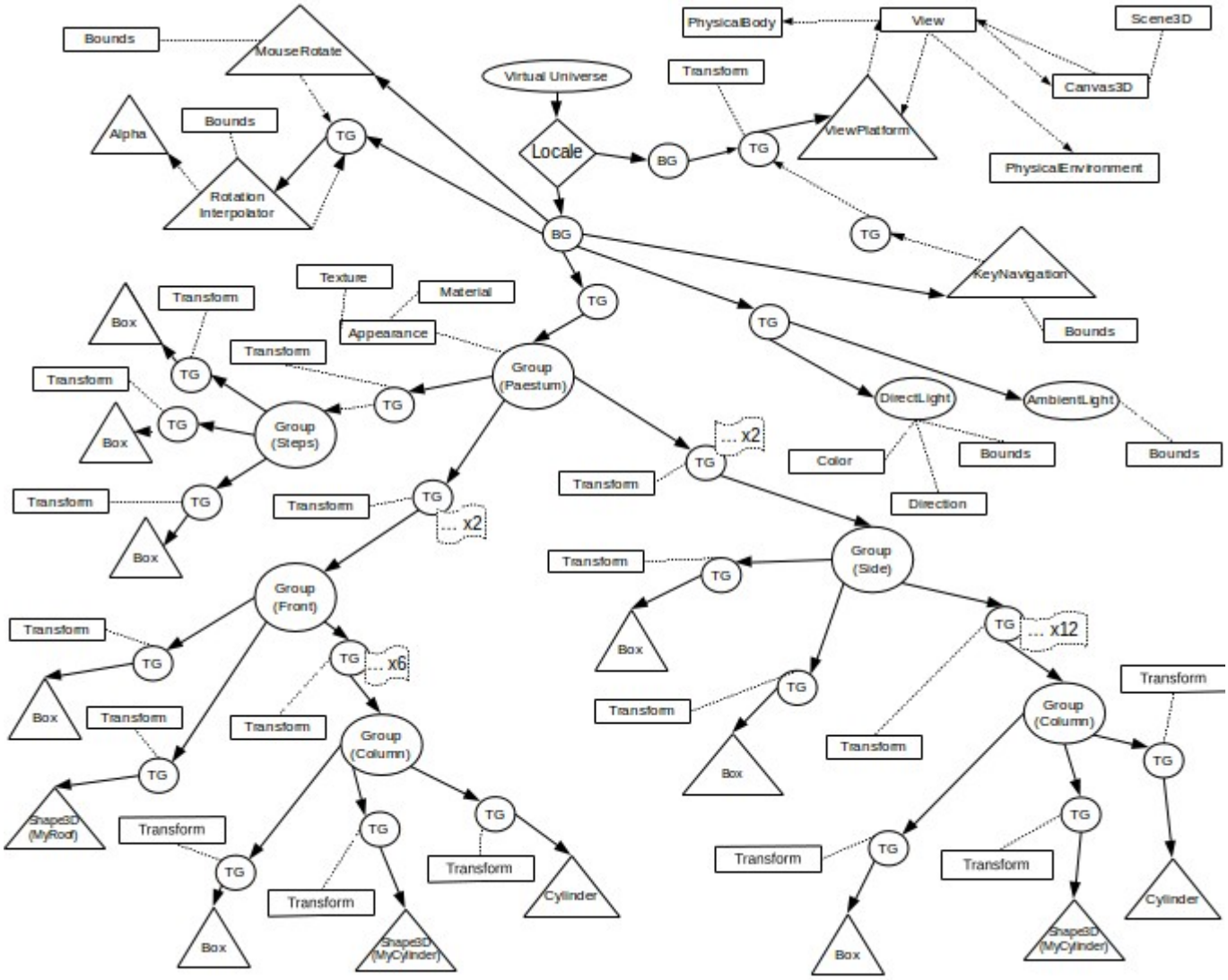
Un ultimo frame

Osservazioni:
Utilizzando un interpolatore è possibile creare una semplice animazione per un oggetto. Si possono impostare il numero di rotazioni e il tempo che l'oggetto impiega a fare una rotazione (nel caso dell'esercizio ho fatto in modo che la terra girasse circa 28 volte in un giro della luna).

Esercizio 4.4

Riprendendo l'esercizio della lezione scorsa (riproduzione della facciata del tempio di Poseidone di Pæstum), procedere nella ricostruzione dell'intero edificio con la possibilità di navigare nel mondo virtuale.

Scenegraph:



Codice:

- ShowPaestum:

```
public class ShowPaestum extends Applet {
	public ShowPaestum() {
		setLayout(new BorderLayout()); // layout manager del container
		// trova la miglior configurazione grafica per il sistema
		GraphicsConfiguration config = SimpleUniverse
				.getPreferredConfiguration();
		// Canvas3D: si occupa del rendering 3D on-screen e off-screen
		Canvas3D canvas3D = new Canvas3D(config);
		add("Center", canvas3D);
		BranchGroup scene = createSceneGraph();
		SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
		simpleU.getViewingPlatform().setNominalViewingTransform();

		ViewingPlatform vp = simpleU.getViewingPlatform();
		TransformGroup vtg = vp.getViewPlatformTransform();
		Transform3D t3d2 = new Transform3D();
		vtg.getTransform(t3d2);
```

```
            /* three reference points */
            t3d2.lookAt(new Point3d(3, 4, 12), new Point3d(0, .4, 0),
                        new Vector3d(0, 1, 0));
            t3d2.invert();
            vtg.setTransform(t3d2);

            // Creo un behavior per la navigazione da tastiera
            KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vtg);
            // Imposto il bound del behavior
            keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),
10000.0));
            // Aggiungo il behavior alla scena
            scene.addChild(keyNavBeh);
            // creazione del sottografo principale
            scene.compile();
            simpleU.addBranchGraph(scene);

      }

      public BranchGroup createSceneGraph() {
            BranchGroup objRoot = new BranchGroup();
            TransformGroup TG = new TransformGroup();
            Transform3D t3d = new Transform3D();
            t3d.setScale(2);
            TG.setTransform(t3d);
            Appearance app = new Appearance();
            app.setMaterial(new Material());
            // create temple to show
            Paestum paestum =  new Paestum();
            TG.addChild(paestum);

            /* Gruppo di trasformazione per le luci */
            TransformGroup TG1 = new TransformGroup();
            ambientLight(TG1);
            directLight(TG1);
            objRoot.addChild(TG1);

            /* Gruppo di trasformazione per le trasformazioni affini */
            TransformGroup objRotate = new TransformGroup();
            objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
            objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
            // Aggiunge al gruppo il gruppo di trasformazione del tempio
            objRotate.addChild(TG);
            // Crea il behavior per ruotarlo
            MouseRotate myMouseRotate = new MouseRotate(objRotate);
            // Imposta un raggio d'azione del behavior
            myMouseRotate.setSchedulingBounds(new BoundingSphere());
            // assembla la scena
            objRoot.addChild(myMouseRotate);
            objRoot.addChild(objRotate);

            return objRoot;

      }

      private void ambientLight(TransformGroup node) {
            /*
             * reazione del bound definisce lo spazio dell'illuminazione mi dice
             * quali sono gli oggetti che posso illuminare
             */
            BoundingSphere bounds = new BoundingSphere(new Point3d(0.d, 0.d,
0.d),
                        10.d);
            // creazione di una sorgente di luce
            AmbientLight lightP1 = new AmbientLight();
```

```
		Color3f green = new Color3f(0.0f, 1.0f, 0.0f);
		// lightP1.setColor(green);
		lightP1.setInfluencingBounds(bounds);
		node.addChild(lightP1); // aggiunta della light al BranchGroup
	}

	private void directLight(TransformGroup node) {
		// creazione del bound
		BoundingSphere bounds = new BoundingSphere(new Point3d(0d, 0.0d, 0d),
				10.d);
		// creazione di una luce direzionale
		DirectionalLight lightD1 = new DirectionalLight();
		// impostazione del bound
		lightD1.setInfluencingBounds(bounds);
		Color3f green = new Color3f(.7f, .7f, .1f);
		//lightD1.setColor(green);
		lightD1.setDirection(-1f, -1f, -1f);
		// aggiunta al TransformGroup
		node.addChild(lightD1);

	}

	public static void main(String[] args) {
		new MainFrame(new ShowPaestum(), 1024, 768);
	}
}
```

- Paestum:

```
public class Paestum extends Group {

	protected Transform3D t3dBase = new Transform3D();
	protected Transform3D t3dFront = new Transform3D();
	protected Transform3D t3dBack = new Transform3D();
	protected Transform3D t3dLeft = new Transform3D();
	protected Transform3D t3dRight = new Transform3D();

	protected TransformGroup TGBase = new TransformGroup(t3dBase);
	protected TransformGroup TGFront = new TransformGroup(t3dFront);
	protected TransformGroup TGBack = new TransformGroup(t3dBack);
	protected TransformGroup TGLeft = new TransformGroup(t3dLeft);
	protected TransformGroup TGRight = new TransformGroup(t3dRight);

	protected Appearance app = new Appearance();
	int primflags = Primitive.GENERATE_NORMALS +
Primitive.GENERATE_TEXTURE_COORDS;

	protected Steps base = new Steps(3,.7f, .05f, 1.5f, app);
	protected FrontFacePaestum front = new FrontFacePaestum(app);
	protected FrontFacePaestum back = new FrontFacePaestum(app);
	protected SidePaestum left = new SidePaestum(app);
	protected SidePaestum right = new SidePaestum(app);


	public Paestum(Appearance appearance) {
		app = appearance;
		Material material = new Material();
		material.setShininess(127);
		app.setMaterial(material);

		Steps base = new Steps(3,.7f, .05f, 1.5f, app);
		FrontFacePaestum front = new FrontFacePaestum(app);
		FrontFacePaestum back = new FrontFacePaestum(app);
		SidePaestum left = new SidePaestum(app);
		SidePaestum right = new SidePaestum(app);
```

```
            /* Gradini */
            t3dBase.setTranslation(new Vector3d(0,-.4,0));
            TGBase.setTransform(t3dBase);
            TGBase.addChild(base);

            addChild(TGBase);

            /* Faccia frontale */
            t3dFront.setTranslation(new Vector3d(0,0, 1.4));
            TGFront.setTransform(t3dFront);
            TGFront.addChild(front);

            addChild(TGFront);

            /* Faccia posteriore */
            t3dBack.rotY(Math.PI);
            t3dBack.setTranslation(new Vector3d(0,0,-1.4));
            TGBack.setTransform(t3dBack);
            TGBack.addChild(back);

            addChild(TGBack);

            /* Lato sinistro */
            t3dLeft.rotY(Math.PI/2);
            t3dLeft.setTranslation(new Vector3d(-.55,0,-.02));
            TGLeft.setTransform(t3dLeft);
            TGLeft.addChild(left);

            addChild(TGLeft);

            /* Lato destro */
            t3dRight.rotY(Math.PI/2);
            t3dRight.setTranslation(new Vector3d(.55,0,-.02));
            TGRight.setTransform(t3dRight);
            TGRight.addChild(right);

            addChild(TGRight);

            /* Texture */
            TextureLoader loader = new
TextureLoader("./textures/PietraColonna.jpg", null);
            Texture texture = loader.getTexture();
            texture.setBoundaryModeS(Texture.WRAP);
            texture.setBoundaryModeT(Texture.WRAP);
            texture.setBoundaryColor( new Color4f( 0.0f, 1.0f, 0.0f, 0.0f ) );

            TextureAttributes texAttr = new TextureAttributes();
            texAttr.setTextureMode(TextureAttributes.MODULATE);
            app.setTexture(texture);
            app.setTextureAttributes(texAttr);


      }

}
```

- FrontFacePaestum:

```
public class FrontFacePaestum extends Group {
      protected int numColumns = 6;

      protected Transform3D t3dRoof = new Transform3D();
      protected Transform3D t3dRoofSupport = new Transform3D();
      protected Transform3D[] t3dColumns = new Transform3D[numColumns];
```

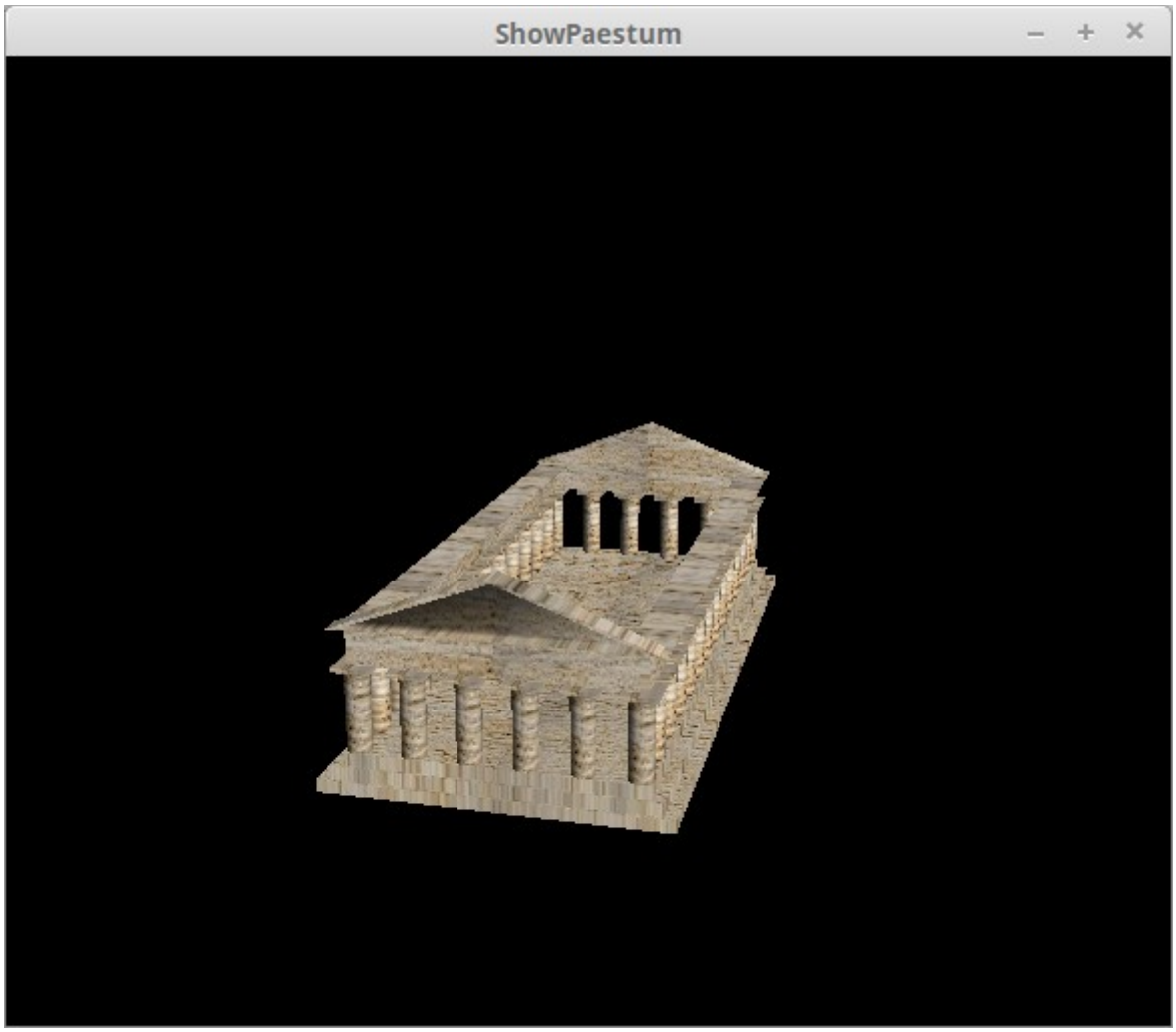
```
	protected TransformGroup TGRoof = new TransformGroup(t3dRoof);
	protected TransformGroup TGRoofSupport = new
TransformGroup(t3dRoofSupport);
	protected TransformGroup[] TGColumns = new TransformGroup[numColumns];

	protected Appearance app = new Appearance();
	int primflags = Primitive.GENERATE_NORMALS +
Primitive.GENERATE_TEXTURE_COORDS;

	protected Column[] columns = new Column[numColumns];
	protected MyRoof roof = new MyRoof(1.3f, .25f, app);
	protected Box roofSupport = new Box(.55f, .07f, .03f, primflags, app);

	public FrontFacePaestum(Appearance appearance) {
		app = appearance;
		float xAxis = 3*.05f-.7f; // spazio tra le colonne
		/* Colonne */
		for(int i=0; i<numColumns; i++) {
			columns[i] = new Column(.05f, .3f, app);
			t3dColumns[i] = new Transform3D();
			TGColumns[i] = new TransformGroup();
			t3dColumns[i].setTranslation(new Vector3d(xAxis, -.28, 0));

			TGColumns[i].setTransform(t3dColumns[i]);
			TGColumns[i].addChild(columns[i]);

			addChild(TGColumns[i]);

			xAxis+=.22f;
		}

		/* Tetto */
		t3dRoofSupport.setTranslation(new Vector3d(0,-.28+.3f+.13, 0));
		TGRoofSupport.setTransform(t3dRoofSupport);
		roofSupport.setAppearance(app);
		TGRoofSupport.addChild(roofSupport);

		addChild(TGRoofSupport);

		t3dRoof.setTranslation(new Vector3d(0,-.28+.3f+.13+.195, 0));
		TGRoof.setTransform(t3dRoof);
		roof.setAppearance(app);
		TGRoof.addChild(roof);

		addChild(TGRoof);

	}

}
```

- SidePaestum:

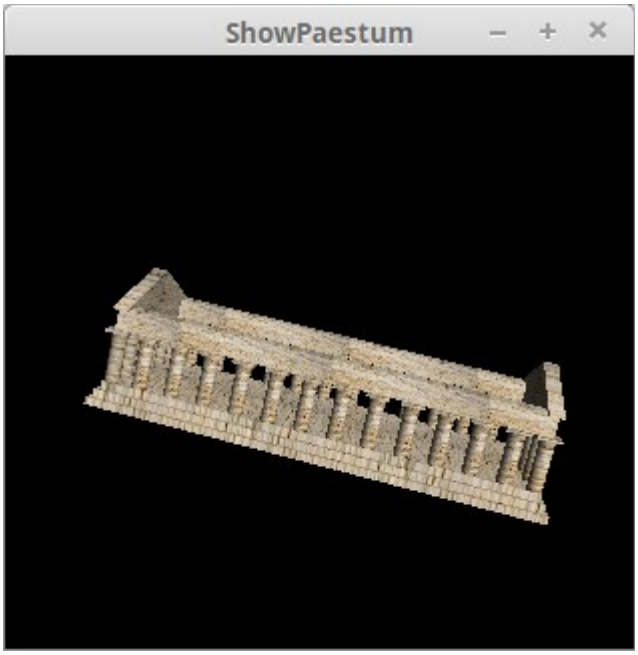
```
public class SidePaestum extends Group {
	protected int numColumns = 12;

	protected Transform3D t3dRoof = new Transform3D();
	protected Transform3D t3dRoofSupport = new Transform3D();
	protected Transform3D[] t3dColumns = new Transform3D[numColumns];

	protected TransformGroup TGRoof = new TransformGroup(t3dRoof);
	protected TransformGroup TGRoofSupport = new
TransformGroup(t3dRoofSupport);
	protected TransformGroup[] TGColumns = new TransformGroup[numColumns];
```

```
	protected Appearance app = new Appearance();
	int primflags = Primitive.GENERATE_NORMALS +
Primitive.GENERATE_TEXTURE_COORDS;

	protected Column[] columns = new Column[numColumns];
	protected Box roof = new Box(1.429f, .005f, .08f, primflags, app);
	protected Box roofSupport = new Box(1.429f, .07f, .03f, primflags, app);

	public SidePaestum(Appearance appearance) {
		app = appearance;
		float xAxis = -1.45f +.22f; // spazio tra le colonne
		/* Colonne */
		for(int i=0; i<numColumns; i++) {
			columns[i] = new Column(.05f, .3f, app);
			t3dColumns[i] = new Transform3D();
			TGColumns[i] = new TransformGroup();
			t3dColumns[i].setTranslation(new Vector3d(xAxis, -.28, 0));

			TGColumns[i].setTransform(t3dColumns[i]);
			TGColumns[i].addChild(columns[i]);

			addChild(TGColumns[i]);

			xAxis+=.22f;
		}

		/* Tetto */
		t3dRoofSupport.setTranslation(new Vector3d(-.02,-.28+.3f+.13, 0));
		TGRoofSupport.setTransform(t3dRoofSupport);
		roofSupport.setAppearance(app);
		TGRoofSupport.addChild(roofSupport);

		addChild(TGRoofSupport);

		t3dRoof.setTranslation(new Vector3d(0,-.28+.3f+.205, 0.02));
		TGRoof.setTransform(t3dRoof);
		roof.setAppearance(app);
		TGRoof.addChild(roof);

		addChild(TGRoof);

	}

}
```

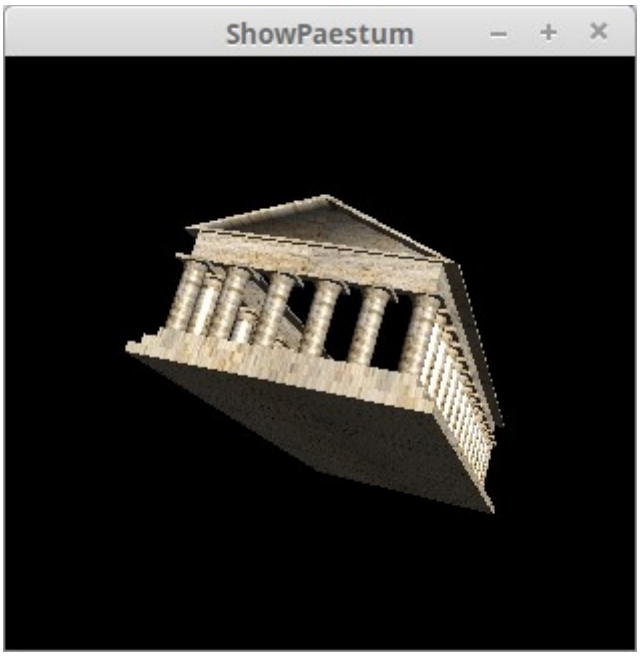
Output:



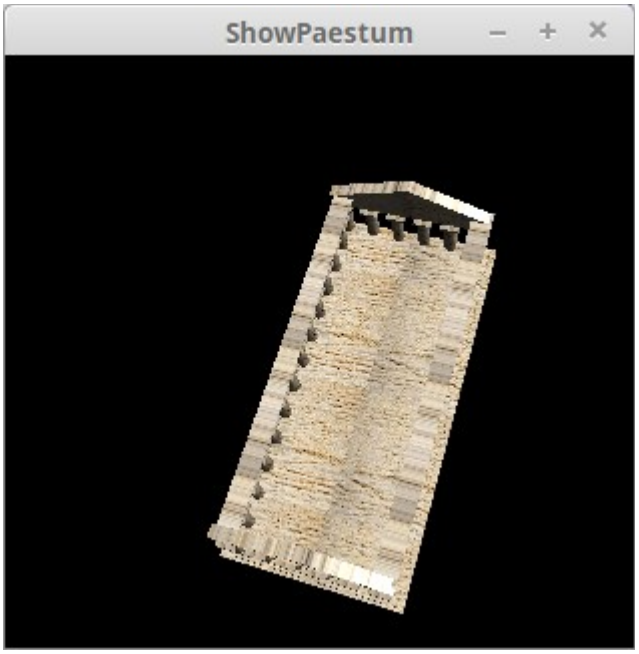
Retro



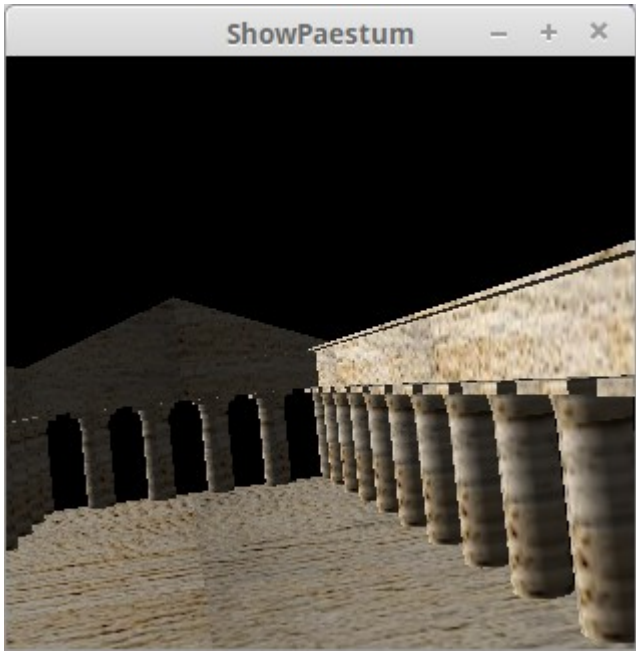
Lato



Da Sotto



Da Sopra



Dall'interno

Osservazioni:
Come ultimo progetto ci è stata assegnata la realizzazione dell'intero tempio di Paestum. La figura complessa è stata suddivisa in 5 parti: la facciata frontale, la facciata posteriore, i due lati e i gradini. Ogni elemento è stato modellato indipendentemente, anche riutilizzando elementi già creati in precedenza.

Sono stati poi aggiunti un rotatore per muovere il tempio e guardarlo a 360° e la possibilità di utilizzare la tastiera per navigare nel mondo virtuale come se si fosse al suo interno.