**Università degli studi di Udine**

**Corso di Immagini e Multimedialità – 2016-17**

prof. Cristian Virgili

Relazione finale – Java3D

**Giovanardi Andrea**

120903

# ESERCIZIO 3.1

##### Traccia dell’esercizio

Creare una scena con un color cube.

1. **Fare le traslazioni sui singoli assi (x, y, z)**

Cosa succede quando mantenendo fisso il punto di visualizzazione e traslare:

* + 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

1. **Fare le rotazioni sui singoli assi (x, y, z)**

Cosa succede quando mantenendo fisso il punto di visualizzazione e ruotare:

* + 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

1. **Fare le scalature sui singoli assi (x, y, z)**

Cosa succede quando mantenendo fisso il punto di visualizzazione e scalare:

* + 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

BG

TG

Transform3D

Color Cube

##### Codice relativo alle traslazioni

public TransformGroup createSubGraph() {

//Creazione nuovo TransformGroup

TransformGroup transform = new TransformGroup();

//Creazione ed aggiunta del cubo

transform.addChild(new ColorCube(0.2));

//Creazione nuova trasformazione

Transform3D t3d = new Transform3D();

//TRASLAZIONE

t3d.setTranslation(new Vector3d(0.0d, 0.0d, -0.5d)); //Vector3d(x, y, z)

transform.setTransform(t3d);

return transform;

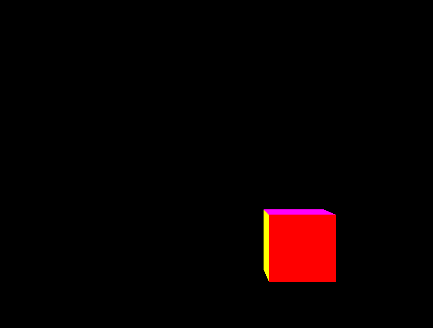
}

##### Schermate relative alle traslazioni

|  |  |  |
| --- | --- | --- |
|  | Valori positivi | valori negativi |
| X |  |  |
| Y |  |  |
| Z |  |  |

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

t3d.setTranslation(new Vector3d(0.5d, -0.5d, -1.0d));



Per valori positiva della X l’oggetto viene spostato a destra, per valori negativi viene sposta a sinistra.

Per valori positivi della Y l’oggetto viene spostato in alto, per valori negativi viene spostato in basso.

Per valori positiva della Z l’oggetto viene avvicinato, per valori negativi viene allontanato.

##### Codice relativo alle rotazioni

public TransformGroup createSubGraph() {

//Creazione nuovo TransformGroup

TransformGroup transform = new TransformGroup();

//Creazione ed aggiunta del cubo

transform.addChild(new ColorCube(0.2));

//Creazione nuova trasformazione

Transform3D t3d = new Transform3D();

//ROTAZIONE

t3d.rotX(Math.*PI*/3); //uguale per rotY e rotZ

transform.setTransform(t3d);

return transform;

}

##### Schermate relative alle rotazioni

|  |  |  |
| --- | --- | --- |
|  | Valori positivi | valori negativi |
| X |  |  |
| Y |  |  |
| Z |  |  |

##### Codice relativo alle scalature

//Funzione che crea oggetto TG e aggiungo cubo come suo figlio

public TransformGroup createSubGraph() {

//Creazione nuovo TransformGroup

TransformGroup transform = new TransformGroup();

//Creazione ed aggiunta del cubo

transform.addChild(new ColorCube(0.2));

//Creazione nuova trasformazione

Transform3D t3d = new Transform3D();

//SCALATURA

t3d.setScale(new Vector3d(2.0d, 1.0d, 1.0d)); //x positiva

//t3d.setScale(new Vector3d(-2.0d, 1.0d, 1.0d)); //x negativa

//t3d.setScale(new Vector3d(1.0d, 2.0d, 1.0d)); //y positiva

//t3d.setScale(new Vector3d(1.0d, -2.0d, 1.0d)); //y negativa

//t3d.setScale(new Vector3d(1.0d, 1.0d, 6.0d)); //z positiva

//t3d.setScale(new Vector3d(1.0d, 1.0d, -6.0d)); //z negativa

transform.setTransform(t3d);

return transform;

}

##### Schermate relative alle scalature

|  |  |  |
| --- | --- | --- |
|  | Valorei positivi | valori negativi |
| X |  |  |
| Y |  |  |
| Z |  |  |

# ESERCIZIO 3.2

##### Traccia dell’esercizio

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

##### Scenegraph

BG

Transform3D  
[dim]

Transform3D  
[0]

Color  
Cube  
[dim]

Color  
Cube  
[0]

TG  
[0]

TG  
[dim]

##### Codice dell’esercizio

//Vengono aggiunti un numero arbitrario di cubi alla scena con associata una trasformazione per ognuno

public TransformGroup createSubGraph() {

//Numero dei cubi

int dim = 10;

//Applica una trasformazione ad ogni oggetto associato

TransformGroup[] transform = new TransformGroup[dim];

TransformGroup scene = new TransformGroup();

ColorCube[] cube = new ColorCube[dim];

//Descrive una trasformazione per ogni oggetto associato

Transform3D [] t3d = new Transform3D[dim];

for (int i = 0; i < cube.length; i++) {

//Traslazione dei cubi rispetto il centro

double x = Math.*sin*((Math.*PI* \* 2 / cube.length) \* i);

double y = Math.*cos*((Math.*PI* \* 2 / cube.length) \* i);

cube[i] = new ColorCube(0.1);

transform[i] = new TransformGroup();

transform[i].addChild(cube[i]);

t3d[i] = new Transform3D();

t3d[i].setTranslation(new Vector3d(x/2, y/2, 0.0d));

transform[i].setTransform(t3d[i]);

scene.addChild(transform[i]);

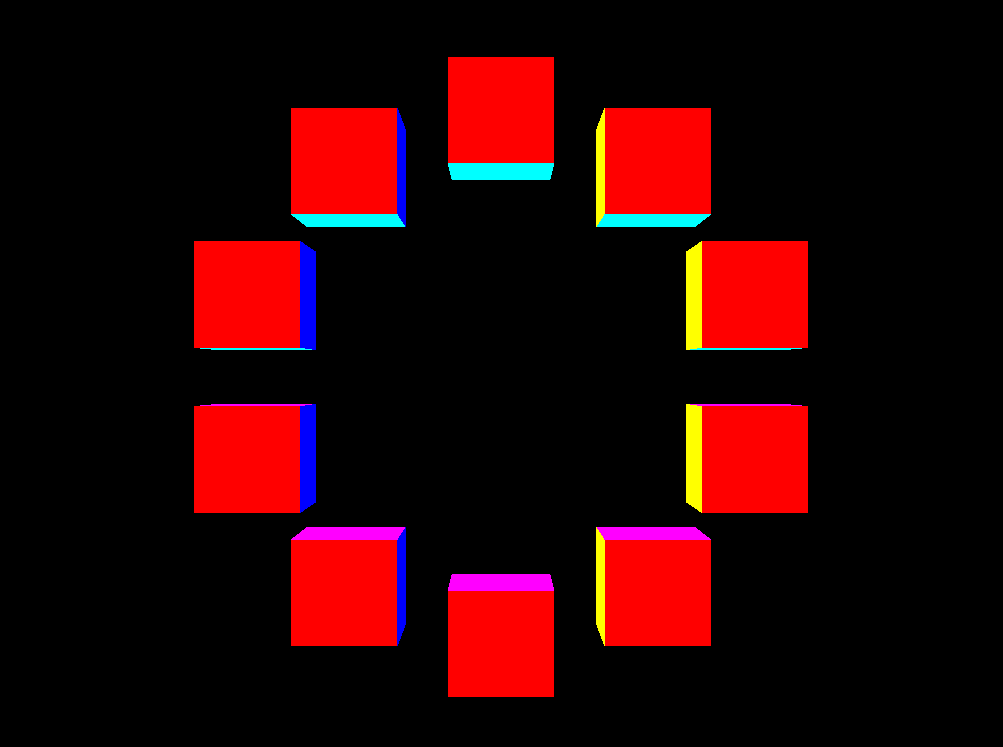
}

return scene;

}

##### Schermate

Vengono usate le funzioni *sin* e *cos* della classe Math per traslare e quindi posizionare i cubi nella scena.



# ESERCIZIO 3.3

##### Traccia dell’esercizio

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

Trovare trasformazioni da applicare al ViewPlatform per cui si hanno:

1. 1 punto di fuga;
2. 2 punti di fuga;
3. 3 punti di fuga;

##### Scenegraph

BG

TG

Color  
Cube

ViewPlatform

##### Codice dell’esercizio

//Funzione per gestire la visuale dell'intera scena

public TransformGroup createViewBranch(SimpleUniverse simpleU) {

Transform3D punto1 = new Transform3D();

Transform3D punti2 = new Transform3D();

Transform3D punti3 = new Transform3D();

punto1.lookAt(new Point3d(2.0d, 2.0d, 2.0d),

new Point3d(0.0, 0.0, 0.0),

new Vector3d(0d, 1.0d, 0d));

punto1.invert();

punti2.lookAt(new Point3d(4d, 0d, 2d),

new Point3d(0.0, 0.0, 0.0),

new Vector3d(0.0, 2.0, 0.0));

punti2.invert();

punti3.lookAt(new Point3d(2d, 1.8d, 2d),

new Point3d(0.0, -0.6, 0.0),

new Vector3d(0.0, 1.0, 0.0));

punti3.invert();

TransformGroup vtg = simpleU.getViewingPlatform().getViewPlatformTransform();

//vtg.setTransform(punto1);

//vtg.setTransform(punti2);

vtg.setTransform(punti3);

return vtg;

}

##### Schermate

|  |  |
| --- | --- |
| Un punto di fuga |  |
| due punti di fuga |  |
| tre punti di fuga |  |

# ESERCIZIO 3.4

##### Traccia dell’esercizio

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:

1. Proiezione ortografica con diverse profondità.
2. Proiezione prospettica con diverse aperture angolari e distanze focali (e quindi diversi livelli di deformazione prospettica)

##### Scenegraph

BG

TG

Color  
Cube

ViewPlatform

##### Codice relativo alla proiezione ortografica/prospettica con diverse profondità e aperture angolari

public Esercizio3\_4() {

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();

//Accedo all'oggetto view del SimpleUniverse

View myView = simpleU.getViewer().getView();

//Abilitazione compatibility mode per modificare la matrice di proiezione

myView.setCompatibilityModeEnable(true);

//Creazione trasformazione

Transform3D t3d = new Transform3D();

//Impostazione matrice di proiezione ortografica

//t3d.ortho(-2, 2, -2.0, 2.0, 2.1, 10.0);

//t3d.ortho(-1, 1, -1, 1, 3, 4);

//t3d.perspective(-Math.PI/4, 1024/768, 3.1, 4);

//t3d.perspective(Math.PI/6, 1024/768, 3.1, 4);

t3d.perspective(-Math.*PI*/5, 1.0, 2.8, 4);

//t3d.perspective(-Math.PI/1.5, 1024/768, 3.1, 4);

//t3d.perspective(Math.PI/4, 1024/768, 2, 4);

myView.setLeftProjection(t3d);

createViewBranch(simpleU);

simpleU.addBranchGraph(scene);

}

##### Schermate

|  |
| --- |
| PROIEZIONE ORTOGONALE |
|  |
|  |

|  |
| --- |
| PROIEZIONE PROSPETTICA |
| -π/4 |
| π/6 |
| π/4 |

# ESERCIZIO 3.5

##### Traccia dell’esercizio

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).

##### Scenegraph

BG

TG

Color  
Cube

ViewPlatform

##### Codice relativo alle proiezioni ortografiche e prospettiche (con diverse aperture angolari)

public Esercizio3\_5() {

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();

//Accedo all'oggetto view del SimpleUniverse

View myView = simpleU.getViewer().getView();

//Impostazione della distanza dal piano sullo sfondo

myView.setBackClipDistance(10);

//Impostazione del clip dal piano frontale

myView.setFrontClipDistance(0.83);

//Impostazione del campo visivo

myView.setFieldOfView(Math.*PI*/8);

//Impostazione del tipo di proiezione

myView.setProjectionPolicy(View.*PERSPECTIVE\_PROJECTION*);

//myView.setProjectionPolicy(View.PARALLEL\_PROJECTION);

myView.setVisibilityPolicy(View.*VISIBILITY\_DRAW\_ALL*);

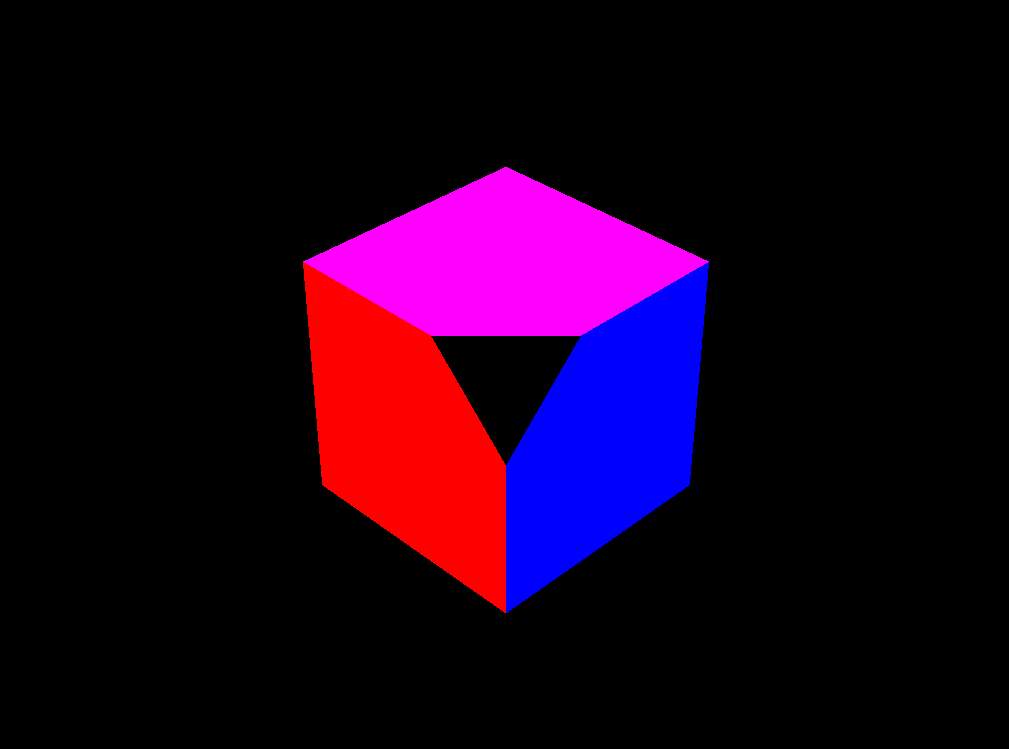
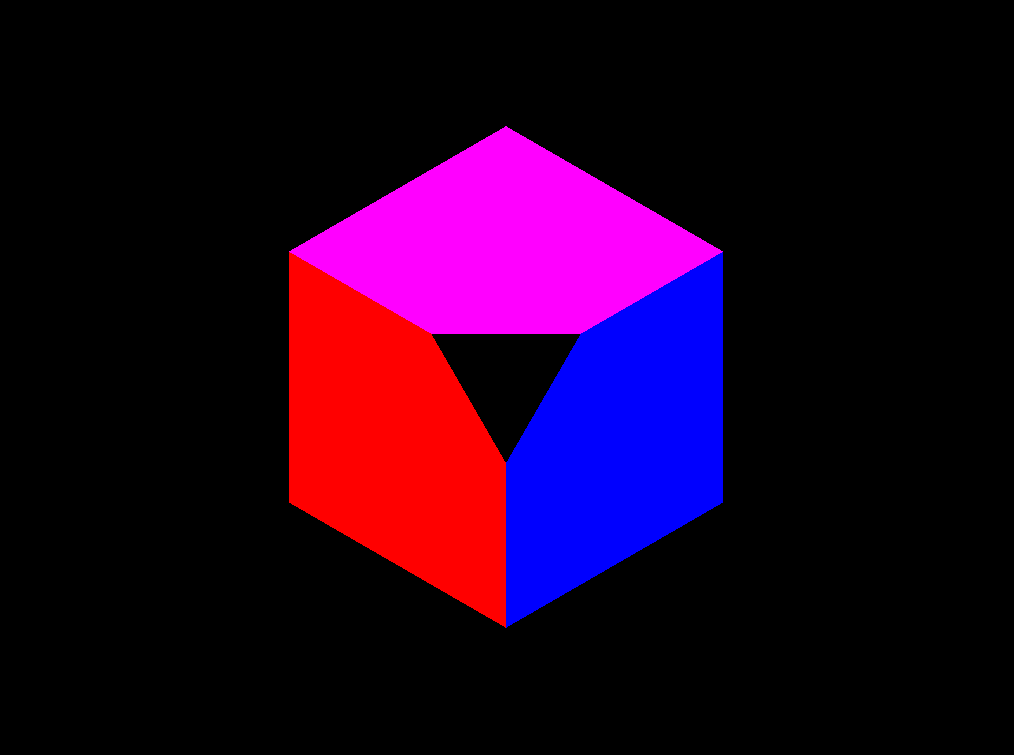
createViewBranch(simpleU);

simpleU.addBranchGraph(scene);

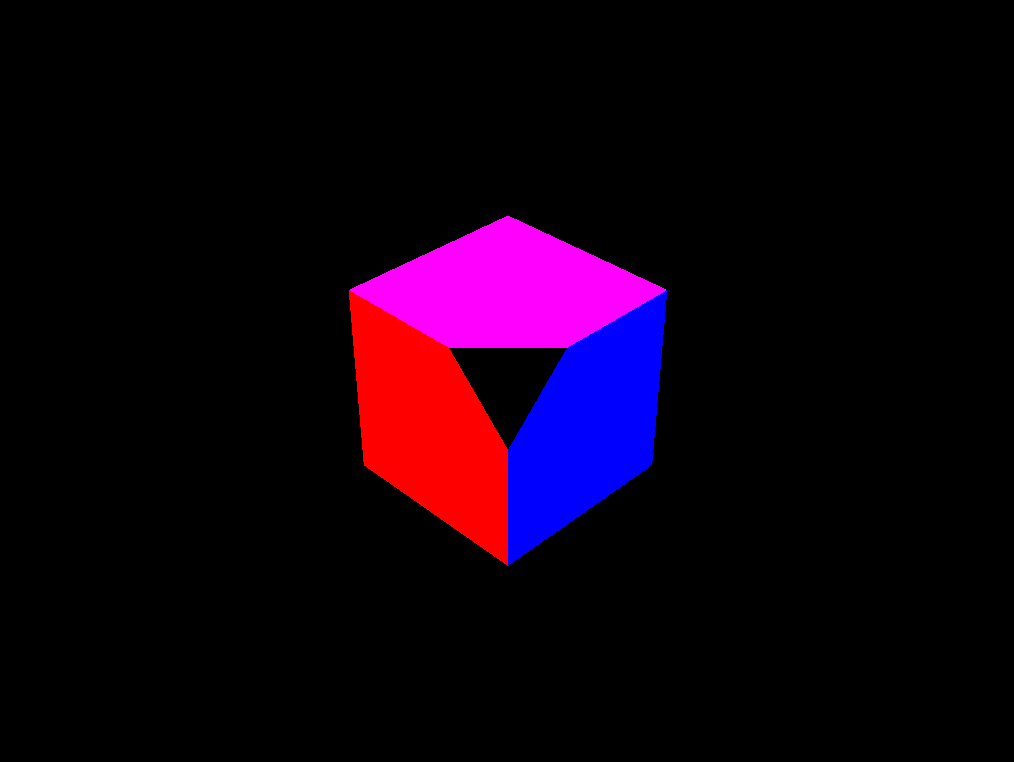
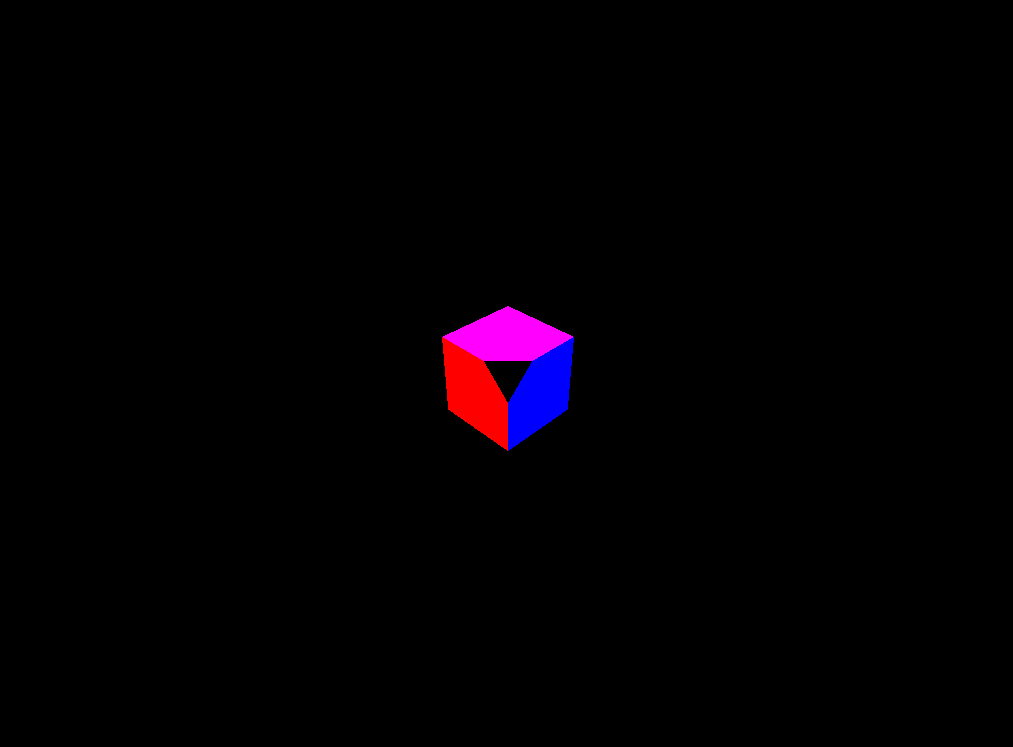
}

##### Schermate

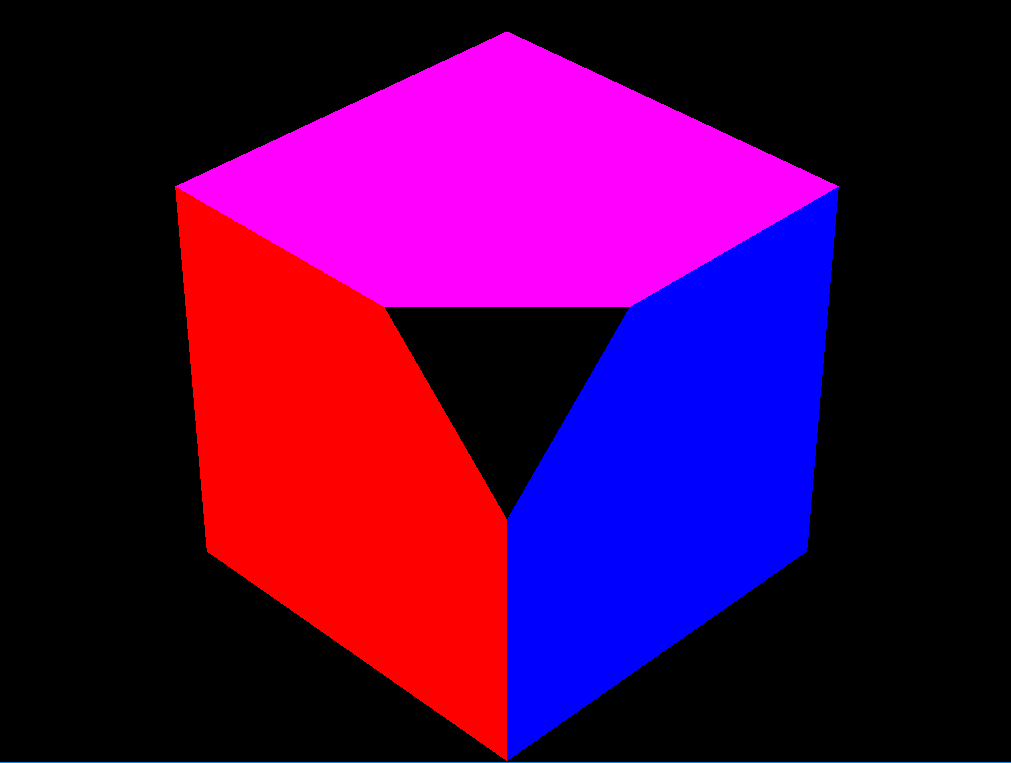
Proiezioni ortogonale Proiezione prospettica



Proiezione prospettica con apertura angolare π/2 Proiezione prospettica con apertura angolare π/4



Proiezione prospettica con apertura angolare π/8



# ESERCIZIO 3.6

##### Traccia dell’esercizio

* 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

Transform3D

Transform3D

Shape3D

Shape3D

##### Codice relativo all’implementazione di una classe per la creazione di tronchi di piramide quadrata

public class Tronco extends Shape3D {

protected Point3d vertex[] = new Point3d[10];

protected TriangleStripArray triangles = null;

protected PolygonAttributes polyAttrbutes = new PolygonAttributes();

protected Appearance appearance = new Appearance();

public Tronco(double altezza, double baseMinore, double baseMaggiore) {

vertex[0] = new Point3d(-baseMinore, -baseMinore, altezza/2);

vertex[1] = new Point3d(-baseMaggiore, -baseMaggiore, -altezza/2);

vertex[2] = new Point3d(baseMinore, -baseMinore, altezza/2);

vertex[3] = new Point3d(baseMaggiore, -baseMaggiore, -altezza/2);

vertex[4] = new Point3d(baseMinore, baseMinore, altezza/2);

vertex[5] = new Point3d(baseMaggiore, baseMaggiore, -altezza/2);

vertex[6] = new Point3d(-baseMinore, baseMinore, altezza/2);

vertex[7] = new Point3d(-baseMaggiore, baseMaggiore, -altezza/2);

vertex[8] = vertex[0];

vertex[9] = vertex[1];

int [] stripCounts = {(vertex.length)};

// n° vertici, formato vertici (coordinate), n° vertici di ogni geometria

triangles = new TriangleStripArray(vertex.length, GeometryArray.*COORDINATES*, stripCounts);

triangles.setCoordinates(0, vertex);

setGeometry(triangles);

polyAttrbutes.setPolygonMode(PolygonAttributes.*POLYGON\_FILL*);

polyAttrbutes.setCullFace(PolygonAttributes.*CULL\_NONE*);

//Applico trasparenza alla piramide per migliorare effetto visivo

appearance.setTransparencyAttributes(

new TransparencyAttributes(

TransparencyAttributes.*BLENDED*, 0.35f)

);

appearance.setPolygonAttributes(polyAttrbutes);

//Colore piramide

ColoringAttributes color = new ColoringAttributes();

color.setColor(1.0f, 0.6f, 0.2f);

appearance.setColoringAttributes(color);

setAppearance (appearance);

}

}

##### Codice relativo all’implementazione di una classe per la creazione di una piramide Maya

public class Piramide extends Group{

protected static final int *num\_tronchi* = 9; //n° tronchi

protected Tronco troncoPiramide[] = new Tronco[*num\_tronchi*+1];

//TransformGroup per troncoPiramide[]

protected TransformGroup TG\_Tronchi[] = new TransformGroup[*num\_tronchi*+1];

protected TransformGroup TG = new TransformGroup(); //TransformGroup generale

protected Transform3D t3dTronchi[] = new Transform3D[*num\_tronchi*+1];

protected Transform3D t3d = new Transform3D();

protected Box punta = new Box(0.5f,0.5f, 0.2f, *createAppearance*()); //punta

public Piramide() {

int i;

double baseMinore = 2.3;

double baseMaggiore = 2.4;

double altezza = 0.2;

for(i=0; i < troncoPiramide.length-1; i++) {

troncoPiramide[i] = new Tronco(altezza,baseMinore,baseMaggiore);

baseMinore = baseMinore-0.2;

baseMaggiore = baseMaggiore-0.2;

TG\_Tronchi[i] = new TransformGroup();

t3dTronchi[i] = new Transform3D();

t3dTronchi[i].setTranslation(new Vector3d(0, 0, i\*altezza));

TG\_Tronchi[i].setTransform(t3dTronchi[i]);

TG\_Tronchi[i].addChild(troncoPiramide[i]);

TG.addChild(TG\_Tronchi[i]);

}

//Definizione traslazione per il Box usato come punta

t3dTronchi[i] = new Transform3D();

t3dTronchi[i].setTranslation(new Vector3d(0, 0, (i\*altezza)+altezza/2));

TG\_Tronchi[i] = new TransformGroup();

TG\_Tronchi[i].setTransform(t3dTronchi[i]);

TG\_Tronchi[i].addChild(punta);

TG.setTransform(t3d);

TG.addChild(TG\_Tronchi[i]);

addChild(TG);

}

static private Appearance createAppearance(){

Appearance appearance = new Appearance();

ColoringAttributes color = new ColoringAttributes();

color.setColor(1.0f, 0.6f, 0.2f);

appearance.setColoringAttributes(color);

appearance.setPolygonAttributes(

new PolygonAttributes(

PolygonAttributes.*POLYGON\_FILL*,

PolygonAttributes.*CULL\_NONE*,

0.0f)

);

//Applico trasparenza al box per migliorare effetto visivo

appearance.setTransparencyAttributes(new TransparencyAttributes(

TransparencyAttributes.*BLENDED*, 0.35f)

);

return appearance;

}

}

##### Codice finale

public class Esercizio3\_6 extends Applet{

public Esercizio3\_6() {

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();

Transform3D transform = new Transform3D();

transform.lookAt(

new Point3d(0.0, 1.5, 12.0),

new Point3d(0.0, 0.0, 0.0),

new Vector3d(0.0, 0.1, 0.0)

);

transform.invert();

TransformGroup vtg = simpleU.getViewingPlatform().getViewPlatformTransform();

vtg.setTransform(transform);

simpleU.addBranchGraph(scene);

OrbitBehavior orbit = new OrbitBehavior(canvas3D, OrbitBehavior.*REVERSE\_ROTATE*);

orbit.setSchedulingBounds(new BoundingSphere());

simpleU.getViewingPlatform().setViewPlatformBehavior(orbit);

}

/\*\*

\* Funzione che crea il sottografo (piramide e trasformazione per visualizzazione)

\*

\* @return il BranchGroup da aggiungere al SimpleUniverse

\*/

public BranchGroup createSceneGraph() {

BranchGroup branchGroup = new BranchGroup();

TransformGroup TG = new TransformGroup();

Transform3D transform3D = new Transform3D();

transform3D.rotX(-Math.*PI*/4);

TG.setTransform(transform3D);

TG.addChild(new Piramide());

branchGroup.addChild(TG);

return branchGroup;

}

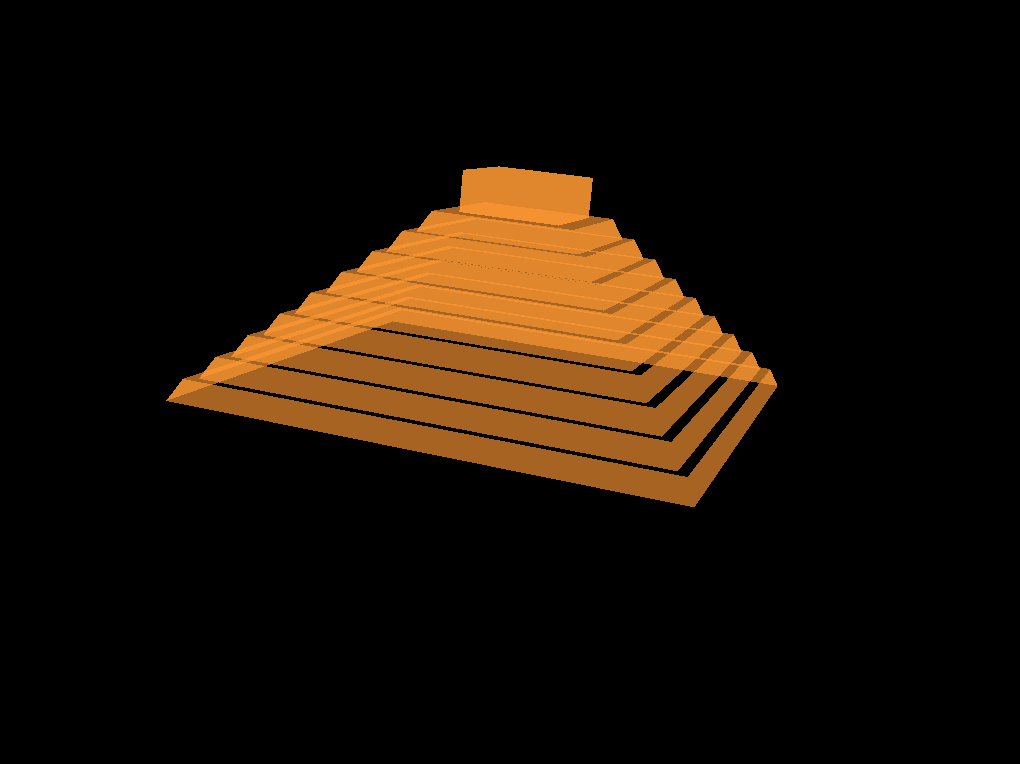
public static void main(String[] args) {

new MainFrame(new Esercizio3\_6(), 800, 800);

}

}

##### Schermata



Nella classe Tronco si è proceduto a definire i vertici che compongono i singoli tronchi, successivamente si è deciso di intervenire sugli attributi del poligono tramite *setPolygonMode* impostato a POLYGON\_FILL (ovvero il poligono viene renderizzato riempiendo lo spazio tra i vertici) e *setCullFace* impostato a CULL\_NONE (ovvero non vengono scartati eventuali poligono prima di essere convertiti in coordinate). Inoltre è stato applicato un colore consono alla piramide ed una trasparenza in modo da migliorare la visualizzazione. Nella classe Piramide si è proceduto alla creazione della piramide usando array di TransformGroup per gestire la creazione dei singoli tronchi di piramide ai quali viene applicata una traslazione per posizionarli correttamente mentre la punta della piramide è stata creata mediato l’uso della primitiva Box. Anche al Box è stata impostata la stessa appearance dei tronchi di piramide.

# ESERCIZIO 3.7

##### Traccia dell’esercizio

Testare le luci su una matrice 5x5 di sfere

##### Scenegraph

BG

BoundingSphere directBound

Dir.  
Light

Amb.  
Light

BoundingSphere ambientBound

TG

TG

TG

BoundingSphere spotBound

Amb.  
Light

Sphere

Sphere

##### Codice

public class Matrice extends Applet {

public Matrice() {

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();

//Transfomazione per visualizzazione della scena corretta

Transform3D transform = new Transform3D();

transform.lookAt(new Point3d(0.0, 0.0, 15.0),

new Point3d(0.0, 0.0, 0.0),

new Vector3d(0.0, 0.1, 0.0));

transform.invert();

TransformGroup vtg = simpleU.getViewingPlatform().getViewPlatformTransform();

vtg.setTransform(transform);

simpleU.addBranchGraph(scene);

//Possibilità ruotara visuale della scena tramite il mouse

OrbitBehavior orbit = new OrbitBehavior(canvas3D, OrbitBehavior.REVERSE\_ROTATE);

orbit.setSchedulingBounds(new BoundingSphere());

simpleU.getViewingPlatform().setViewPlatformBehavior(orbit);

}

/\*\*

\* Funzione che crea il sottografo principale

\*

\* @return branchGroup

\*/

public BranchGroup createSceneGraph() {

BranchGroup branchGroup = new BranchGroup();

branchGroup.addChild(luceDirezionale());

//branchGroup.addChild(luceAmbientale());

//branchGroup.addChild(luceSpot());

TransformGroup TG = new TransformGroup();

//Creo matrice di sfere

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

//TransformGroup e Transform3D per ogni sfera della matrice

TransformGroup Sfera = new TransformGroup();

Transform3D t3d = new Transform3D();

//Uso gli indici come base delle nuove coordinate della sfera

t3d.setTranslation(new Vector3d(i-2, j-2, 0));

Sfera.setTransform(t3d);

//Creo singola sfera

Sfera.addChild(new Sphere(0.4f,

Primitive.GEOMETRY\_NOT\_SHARED | Primitive.GENERATE\_NORMALS,

createAppearance())

);

TG.addChild(Sfera);

}

}

branchGroup.addChild(TG);

return branchGroup;

}

/\*\*

\* Funzione che crea la luce direzionale (settando anche una direzione)

\*

\* @return directLight da applicare alla matrice di sfere

\*/

public DirectionalLight luceDirezionale() {

BoundingSphere directBound = new BoundingSphere(

new Point3d(0.0, 0.0, 0.0),

50.0

);

DirectionalLight directLight = new DirectionalLight();

directLight.setInfluencingBounds(directBound);

directLight.setDirection(new Vector3f(1.5f, -1.0f, -1.0f));

return directLight;

}

/\*\*

\* Funzione che crea la luce ambientale (settando anche un colore)

\*

\* @return ambientLight da applicare alla matrice di sfere

\*/

public AmbientLight luceAmbientale() {

BoundingSphere ambientBound = new BoundingSphere(

new Point3d(0.0, 0.0, 0.0),

20.0

);

AmbientLight ambientLight = new AmbientLight();

Color3f pink = new Color3f(1.0f, 0.0f, 1.0f);

ambientLight.setColor(pink);

ambientLight.setInfluencingBounds(ambientBound);

return ambientLight;

}

/\*\*

\* Funzione che crea un cono di luce (settando posizione e angolo)

\*

\* @return spotLight da applicare alla matrice di sfere

\*/

public SpotLight luceSpot() {

BoundingSphere spotBound = new BoundingSphere(

new Point3d(0.0, 0.0, 0.0),

10.0

);

SpotLight spotLight = new SpotLight();

//Setto posizione della luce e angolo di apertura

spotLight.setPosition(new Point3f(0.5f, 0.0f, 1.8f));

spotLight.setSpreadAngle((float)Math.PI/3.3f);

spotLight.setInfluencingBounds(spotBound);

return spotLight;

}

/\*\*

\* Funzione che imposta l'appearance delle sfere

\*

\* @return appearance impostato alla sfera

\*/

public Appearance createAppearance() {

Appearance appearance = new Appearance();

Material material = new Material();

appearance.setMaterial(material);

return appearance;

}

public static void main(String[] args) {

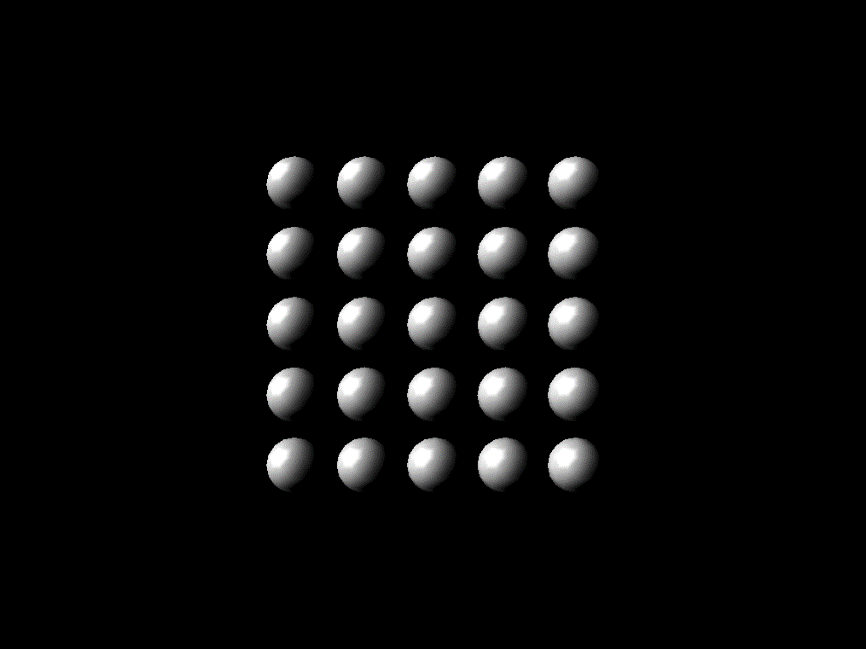
new MainFrame(new Matrice(), 1024, 768);

}

}

##### Schermate

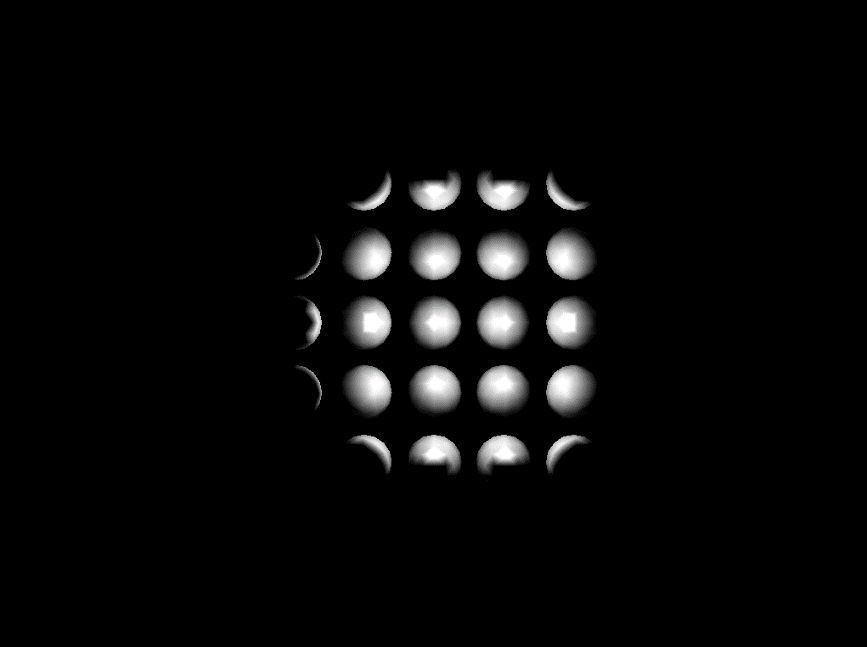
Luce Direzionale (direzione impostata tramite vettore)



Luce Ambientale (colore applicato tramite definizione di una variabile Color3f ed impostata con setColor)



Luce Spot (posizione della luce ed angolo di apertura impostati con setPosition e setSpreadAngle)



È stato scelto di applicare un TransformGroup Sfera e di associare un Transfrom3D t3d per ogni sfera della matrice 5x5. La trasformazione si occupa di traslare le coordinate del centro della sfera (facendo uso degli indici dei cicli per il calcolo) in modo da ottenere una matrice di sfere. Ad ogni sfera viene applicata una appearance tale da permettere alle luci create di creare vari effetti in base al colore, direzione e/o attenuazione della fonte luminosa. Sono state usate luci di tipo DirectLight, Ambientlight e SpotLight con i relativi metodi per impostare eventuali parametri opzionali, ogni luce viene definita in una funzione a parte per permettere un maggiore controllo e riutilizzo del codice.

# ESERCIZIO 3.8

##### Traccia dell’esercizio

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).

##### Scenegraph

Abaco

Transform3D

Transform3D

BoundingSphere directBound

Dir. Light

Echino

Fusto

##### Codice relativo alla creazione del Fusto

public class Fusto extends Cylinder {

public Fusto(float altezza) {

//Cylinder(raggio, altezza, primitive, n° divisioni lungo direzione x, n° divisioni lungo altezza, appearance)

super(altezza/9, altezza, Primitive.*GENERATE\_NORMALS*, 20, 1, *createApp*());

}

/\*\*

\* Funzione che imposta aspetto del Fusto

\*

\* @return appearance

\*/

public static Appearance createApp() {

Appearance appearance = new Appearance();

Material material = new Material();

//setAmbientColor(R, G, B) => colore luce ambientale riflessa

material.setAmbientColor(210/255f, 180/255f, 140/255f);

//setDiffuseColor(R, G, B) => colore del materiale quando viene illuminanto da una fonte luminosa

material.setDiffuseColor(210/255f, 180/255f, 140/255f);

appearance.setMaterial(material);

return appearance;

}

}

Per la creazione della classe Fusto si è deciso di estendere la classe primitiva Cylinder e impostando al costruttore della superclasse, oltre che i valori per le varie dimensioni anche un’appearance definita con il metodo createApp().

##### Codice relativo alla creazione dell’Abaco

public class Abaco extends Box {

public Abaco(float xDim, float yDim, float zDim) {

//Faccio riferimento al costruttore di Box

super(xDim, yDim, zDim, *createAppearance*());

}

/\*\*

\* Funzione che imposta aspetto dell'Abaco

\*

\* @return appearance

\*/

public static Appearance createAppearance() {

Appearance appearance = new Appearance();

Material material = new Material();

//set Color(R, G, B) => colore luce ambientale riflessa

material.setAmbientColor(210/255f, 180/255f, 140/255f);

//setDiffuseColor(R, G, B) => colore del materiale quando viene illuminanto da una fonte luminosa

material.setDiffuseColor(210/255f, 180/255f, 140/255f);

appearance.setMaterial(material);

return appearance;

}

}

Per la creazione della classe Abaco si è deciso di estendere la classe primitiva Box e impostando al costruttore della superclasse, oltre che i valori per le varie dimensioni anche un’appearance definita con il metodo createAppearance().

##### Codice relativo alla creazione dell’Echino

public class Echino extends Shape3D {

protected Point3f v[] = null;

protected TriangleStripArray triangleStrip = null;

protected PolygonAttributes polyAttributes = new PolygonAttributes();

protected Appearance appearance = new Appearance();

public Echino(int steps, float raggioS, float raggioI, float altezza) {

float top = altezza/2;

float bottom = -altezza/2;

v = new Point3f[(steps+1)\*2];

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

double angle = 2.0\*Math.*PI*\*(double)i/(double)steps;

float xI = (float)Math.*sin*(angle)\*raggioI;

float yI = (float)Math.*cos*(angle)\*raggioI;

float xS = (float)Math.*sin*(angle)\*raggioS;

float yS = (float)Math.*cos*(angle)\*raggioS;

v[i\*2] = new Point3f(xI, yI, bottom);

v[i\*2+1] = new Point3f(xS, yS, top);

}

v[steps\*2] = new Point3f(0.0f, raggioI, bottom);

v[steps\*2+1] = new Point3f(0.0f, raggioS, top);

int[] stripCounts = {(steps+1)\*2};

triangleStrip = new TriangleStripArray((steps+1)\*2,

GeometryArray.*COORDINATES*, stripCounts);

triangleStrip.setCoordinates(0, v);

GeometryInfo geometry = new GeometryInfo(triangleStrip);

NormalGenerator normal = new NormalGenerator();

normal.generateNormals(geometry);

setGeometry(triangleStrip);

setGeometry(geometry.getGeometryArray());

//Impostazione aspetto wireframe (tramite FILL riempio l’intera figura)

polyAttributes.setPolygonMode(PolygonAttributes.*POLYGON\_FILL*);

polyAttributes.setCullFace(PolygonAttributes.*CULL\_NONE*);

appearance.setPolygonAttributes(polyAttributes);

Material material = new Material();

material.setAmbientColor(210/255f, 180/255f, 140/255f);

material.setDiffuseColor(210/255f, 180/255f, 140/255f);

appearance.setMaterial(material);

setAppearance(appearance);

}

}

Per la creazione della classe Echino si è preso spunto dall’esempio di cilindro fornito a lezione, variando il raggio superiore ed inferiore in modo da creare l’aspetto di un tronco di cono. Inoltre per migliorare l’aspetto grafico è stata utilizzata la classe Geometry (anche per generare le normali) ed è stata creata una appearance.

##### Codice relativo alla creazione della Colonna

public class Colonna extends Group{

private TransformGroup TGAbaco;

private TransformGroup TGFusto;

private TransformGroup TGEchino;

private Transform3D t3dEchino;

private Transform3D t3dEchinoRotazione;

private Transform3D t3dAbaco;

public Colonna(float altezza) {

Fusto fusto = new Fusto(altezza);

Echino echino = new Echino(18, altezza/5, altezza/9, altezza/12);

Abaco abaco = new Abaco(altezza/5, altezza/30, altezza/5);

TGFusto = new TransformGroup();

TGFusto.addChild(fusto);

TGEchino = new TransformGroup();

TGEchino.addChild(echino);

TGAbaco = new TransformGroup();

TGAbaco.addChild(abaco);

//Trasformazione relative al posizionamento dell’echino

t3dEchino = new Transform3D();

t3dEchino.setTranslation(new Vector3f(0f, altezza/2+altezza/24, 0f));

t3dEchinoRotazione = new Transform3D();

t3dEchinoRotazione.rotX(-Math.*PI*/2);

t3dEchino.mul(t3dEchinoRotazione);

TGEchino.setTransform(t3dEchino);

//Trasformazioni relative al posizionamento dell’abaco

t3dAbaco = new Transform3D();

t3dAbaco.setTranslation(new

Vector3f(0,altezza/2+altezza/12+altezza/30,0f)

);

TGAbaco.setTransform(t3dAbaco);

addChild(TGFusto);

addChild(TGEchino);

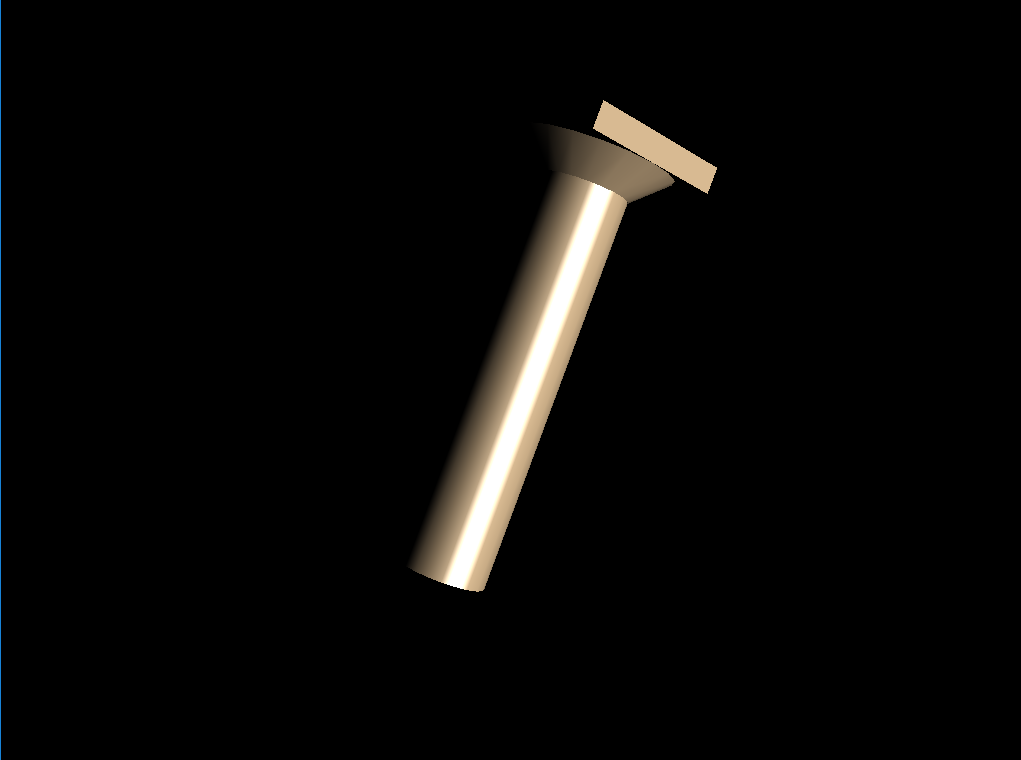
addChild(TGAbaco);

}

}

Per creare la colonna nella sua interezza è stato deciso di creare un TransformGroup per ogni singola figura che la compone oltre che due Transform3D necessari per posizionare correttamente l’abaco e l’echino sopra il fusto tramite traslazione e rotazioni. In questo modo l’oggetto Colonna è riutilizzabile oltre che essere facilmente modificabile.

##### Schermata



# ESERCIZIO 4.1

##### Traccia dell’esercizio

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

##### Scenegraph

BounginSphere bounds

Sphere

BoundingSphere directBound

BackGround

Directional Light

##### Codice

public BranchGroup createSceneGraph() {

BranchGroup branchGroup = new BranchGroup();

TransformGroup TG = new TransformGroup();

branchGroup.addChild(luceSolare());

/\*CREAZIONE TERRA\*/

Sphere terra = new Sphere(

1.0f,

Primitive.GENERATE\_NORMALS | Primitive.GENERATE\_TEXTURE\_COORDS,

200,

createAppearance()

);

TG.addChild(terra);

branchGroup.addChild(TG);

/\*CREAZIONE BACKGROUND\*/

//1) Carico texture per il background

String localPath = new File("").getAbsolutePath() + "\\src/Texture\\";

TextureLoader loader = new TextureLoader(localPath + "stars.jpg", "RGB", new Container());

ImageComponent2D image = loader.getImage();

//2) Impostazione background

Background background = new Background();

background.setImage(image);

//Stira l'immagine per coprire tutta l'ampiezza della finestra

background.setImageScaleMode(Background.SCALE\_FIT\_MAX);

//3) Raggio d'influenza del background sulla scena

BoundingSphere bounds = new BoundingSphere

(new Point3d(0.0d, 0.0d, 0.0d), 1000.0

);

background.setApplicationBounds(bounds);

branchGroup.addChild(background);

return branchGroup;

}

/\*\*

\* Funzione che modella l'aseptto della terra

\*

\* @return appearance da applicare alla terra

\*/

public Appearance createAppearance() {

Appearance appearance = new Appearance();

Material material = new Material();

//Carico texture, impostazione attributi materiale (riflessione)

material.setShininess(80.0f);

material.setSpecularColor(new Color3f(0.0f, 0.0f, 0.0f));

appearance.setMaterial(material);

//Caricamento texture

String localPathTerra = new File("").getAbsolutePath() + "\\src/Texture\\";

TextureLoader loaderTerra = new TextureLoader(localPathTerra + "earth.jpg", "RGB", new Container());

Texture texture = loaderTerra.getTexture();

//Impostazione texture e suoi attributi

TextureAttributes attributes = new TextureAttributes();

attributes.setTextureMode(TextureAttributes.COMBINE);

appearance.setTexture(texture);

appearance.setTextureAttributes(attributes);

return appearance;

}

/\*\*

\* Funzione che crea la luce del sole (settando anche una direzione)

\*

\* @return directLight da applicare alla terra

\*/

public DirectionalLight luceSolare() {

BoundingSphere bounds = new BoundingSphere(

new Point3d(0.0, 0.0, 0.0), 10.0

);

DirectionalLight directLight = new DirectionalLight();

directLight.setDirection(new Vector3f(-5.0f, -1.0f, -1.5f));

directLight.setInfluencingBounds(bounds);

return directLight;

}

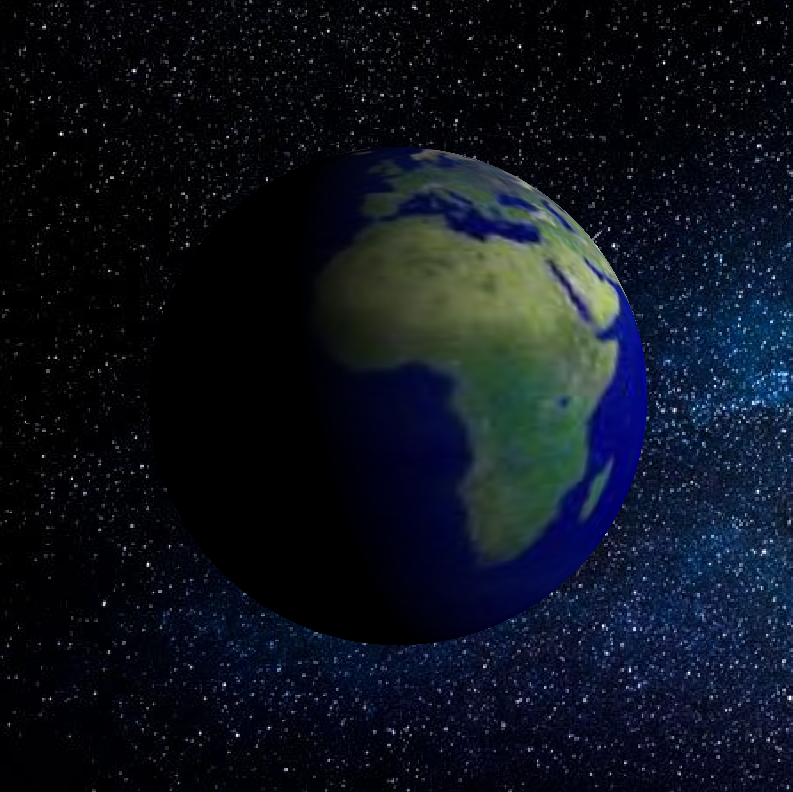
public static void main(String[] args) {

new MainFrame(new Terra(), 800, 800);

}

}

##### Schermata



Sono stati impostati alcuni attributi del materiale in modo da rendere migliore li risultato della riflessione della luce sulla terra andando ad agire sulla lucidità (1 non lucido – 128 lucido) e colore speculare.

# ESERCIZIO 4.2

##### Traccia dell’esercizio

Applicare una texture agli elementi della colonna dorica creata precedentemente.

Riprodurre la facciata del tempio di Poseidone di Pӕstum.

##### Scenegraph

BoundingSphere

DirectLight

Timpano

TIMPANO

Scale [0]

Scale [3]

Transform3D

Transform3D

Col. [0]

Col. [5]

Transform3D

Abaco

Transform3D

SCALA

COLONNA

ABACO

##### Codice relativo alla creazione del Fusto

public class Fusto extends Cylinder {

public Fusto(float altezza) {

//Cylinder(raggio, altezza, primitive, n° divisioni lungo direzione x, n° divisioni lungo altezza, appearance)

super(altezza/9, altezza, Primitive.*GENERATE\_TEXTURE\_COORDS*, 20, 1, *createAppearance*());

}

/\*\*

\* Funzione che imposta aspetto del Fusto

\*

\* @return appearance

\*/

public static Appearance createAppearance() {

Appearance appearance = new Appearance();

Material material = new Material();

//Caricamento texture

String localPath = new File("").getAbsolutePath() + "\\src/Texture\\";

TextureLoader loader = new TextureLoader(localPath + "PietraColonna.jpg", "RGB", new Container());

Texture texture = loader.getTexture();

//Impostazione texture e suoi attributi

/\*TextureAttributes attributes = new TextureAttributes();

attributes.setTextureMode(TextureAttributes.COMBINE);\*/

appearance.setTexture(texture);

//appearance.setTextureAttributes(attributes);

appearance.setMaterial(material);

return appearance;

}

}

##### Codice relativo alla creazione dell’Abaco

public class Abaco extends Box {

public Abaco(float xDim, float yDim, float zDim) {

super(xDim, yDim, zDim, Box.*GENERATE\_TEXTURE\_COORDS*, *createAppearance*());

}

/\*\*

\* Funzione che imposta aspetto dell'Abaco

\*

\* @return appearance

\*/

public static Appearance createAppearance() {

Appearance appearance = new Appearance();

Material material = new Material();

//Caricamento texture

String localPath = new File("").getAbsolutePath() + "\\src/Texture\\";

TextureLoader loader = new TextureLoader(localPath + "PietraColonna.jpg", "RGB", new Container());

Texture texture = loader.getTexture();

//Impostazione texture e suoi attributi

/\*TextureAttributes attributes = new TextureAttributes();

attributes.setTextureMode(TextureAttributes.COMBINE);\*/

appearance.setTexture(texture);

//appearance.setTextureAttributes(attributes);

appearance.setMaterial(material);

return appearance;

}

##### Codice relativo alla creazione dell’Echino

public class Echino extends Shape3D {

protected Point3f v[] = null;

protected TriangleStripArray triangleStrip = null;

protected PolygonAttributes polyAttributes = new PolygonAttributes();

protected Appearance appearance = new Appearance();

public Echino(int steps, float raggioS, float raggioI, float altezza) {

float top = altezza/2;

float bottom = -altezza/2;

v = new Point3f[(steps+1)\*2];

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

double angle = 2.0\*Math.*PI*\*(double)i/(double)steps;

float xI = (float)Math.*sin*(angle)\*raggioI;

float yI = (float)Math.*cos*(angle)\*raggioI;

float xS = (float)Math.*sin*(angle)\*raggioS;

float yS = (float)Math.*cos*(angle)\*raggioS;

v[i\*2] = new Point3f(xI, yI, bottom);

v[i\*2+1] = new Point3f(xS, yS, top);

}

v[steps\*2] = new Point3f(0.0f, raggioI, bottom);

v[steps\*2+1] = new Point3f(0.0f, raggioS, top);

int[] stripCounts = {(steps+1)\*2};

triangleStrip = new TriangleStripArray((steps+1)\*2, GeometryArray.*COORDINATES*, stripCounts);

triangleStrip.setCoordinates(0, v);

GeometryInfo geometry = new GeometryInfo(triangleStrip);

NormalGenerator normal = new NormalGenerator();

normal.generateNormals(geometry);

setGeometry(triangleStrip);

setGeometry(geometry.getGeometryArray());

//Impostazione aspetto wireframe

polyAttributes.setPolygonMode(PolygonAttributes.*POLYGON\_FILL*);

polyAttributes.setCullFace(PolygonAttributes.*CULL\_NONE*);

appearance.setPolygonAttributes(polyAttributes);

Material material = new Material();

//Caricamento texture

String localPath = new File("").getAbsolutePath() + "\\src/Texture\\";

TextureLoader loader = new TextureLoader(localPath + "PietraColonna.jpg", "RGB", new Container());

Texture texture = loader.getTexture();

//Impostazione texture e suoi attributi

TextureAttributes attributes = new TextureAttributes();

attributes.setTextureMode(TextureAttributes.*MODULATE*);

TexCoordGeneration texCoordinates = new TexCoordGeneration(TexCoordGeneration.*OBJECT\_LINEAR*, TexCoordGeneration.*TEXTURE\_COORDINATE\_3*);

appearance.setTexCoordGeneration(texCoordinates);

appearance.setTexture(texture);

appearance.setTextureAttributes(attributes);

appearance.setMaterial(material);

setAppearance(appearance);

}

}

##### Codice relativo alla creazione della Colonna

public class Colonna extends Group{

private TransformGroup TGAbaco;

private TransformGroup TGFusto;

private TransformGroup TGEchino;

private Transform3D t3dEchino;

private Transform3D t3dEchinoRotazione;

private Transform3D t3dAbaco;

public Colonna(float altezza) {

Fusto fusto = new Fusto(altezza);

Echino echino = new Echino(18, altezza/5, altezza/9, altezza/12);

Abaco abaco = new Abaco(altezza/5, altezza/30, altezza/5);

TGFusto = new TransformGroup();

TGFusto.addChild(fusto);

TGEchino = new TransformGroup();

TGEchino.addChild(echino);

TGAbaco = new TransformGroup();

TGAbaco.addChild(abaco);

t3dEchino = new Transform3D();

t3dEchino.setTranslation(new Vector3f(0f, altezza/2+altezza/24, 0f));

t3dEchinoRotazione = new Transform3D();

t3dEchinoRotazione.rotX(-Math.*PI*/2);

t3dEchino.mul(t3dEchinoRotazione);

TGEchino.setTransform(t3dEchino);

t3dAbaco = new Transform3D();

t3dAbaco.setTranslation(new Vector3f(0, altezza/2+altezza/12+altezza/30, 0f));

TGAbaco.setTransform(t3dAbaco);

addChild(TGFusto);

addChild(TGEchino);

addChild(TGAbaco);

}

}

##### Codice relativo alla creazione del Timpano

public class Timpano extends Shape3D{

protected Point3f p[] = null;

protected TriangleStripArray triangleStrip = null;

protected PolygonAttributes polyAttributes = new PolygonAttributes();

protected Appearance appearance = new Appearance();

public Timpano(float altezza, float larghezza, float distanza) {

Point3f p0 = new Point3f(-larghezza/2, -altezza/2, distanza/2);

Point3f p1 = new Point3f(0, altezza/2, distanza/2);

Point3f p2 = new Point3f(larghezza/2, -altezza/2, distanza/2);

Point3f p3 = new Point3f(0, altezza/2, -distanza/2);

Point3f p4 = new Point3f(larghezza/2, -altezza/2, -distanza/2);

Point3f p5 = new Point3f(-larghezza/2, -altezza/2, -distanza/2);

//Creazione punti per le coordinate della figura

p = new Point3f[24];

p[0] = p0; p[1] = p2; p[2] = p1;

p[3] = p1; p[4] = p2; p[5] = p3;

p[6] = p2; p[7] = p4; p[8] = p3;

p[9] = p0; p[10] = p1; p[11] = p5;

p[12] = p1; p[13] = p3; p[14] = p5;

p[15] = p3; p[16] = p4; p[17] = p5;

p[18] = p0; p[19] = p2; p[20] = p5;

p[21] = p5; p[22] = p2; p[23] = p4;

//N° facce (in base ai punti appena definiti)

int[] stripCounts = {3,3,3,3,3,3,3,3};

triangleStrip = new TriangleStripArray(p.length, GeometryArray.*COORDINATES*, stripCounts);

triangleStrip.setCoordinates(0, p);

GeometryInfo geometry = new GeometryInfo(triangleStrip);

NormalGenerator normal = new NormalGenerator();

normal.generateNormals(geometry);

setGeometry(triangleStrip);

setGeometry(geometry.getGeometryArray());

//Impostazione aspetto wireframe

polyAttributes.setPolygonMode(PolygonAttributes.*POLYGON\_FILL*);

polyAttributes.setCullFace(PolygonAttributes.*CULL\_NONE*);

appearance.setPolygonAttributes(polyAttributes);

Material material = new Material();

//Caricamento texture

String localPath = new File("").getAbsolutePath() + "\\src/Texture\\";

TextureLoader loader = new TextureLoader(localPath + "PietraColonna.jpg", "RGB", new Container());

Texture texture = loader.getTexture();

//Impostazione texture e suoi attributi

TextureAttributes attributes = new TextureAttributes();

attributes.setTextureMode(TextureAttributes.*MODULATE*);

TexCoordGeneration texCoordinates = new TexCoordGeneration(TexCoordGeneration.*OBJECT\_LINEAR*, TexCoordGeneration.*TEXTURE\_COORDINATE\_3*);

appearance.setTexCoordGeneration(texCoordinates);

appearance.setTexture(texture);

appearance.setTextureAttributes(attributes);

appearance.setMaterial(material);

setAppearance(appearance);

}

}

Per creare il timpano (triangolo superiore della facciata) è stato necessario definire i punti e successivamente le coordinate per la creazione della figura, inoltre (come per le altre classi presenti nell’esercizio) è stata impostata una texture dopo aver impostato attributi e materiale.

##### Codice relativo alla creazione della scalinata

public class Scale extends Box {

public Scale(float xDim, float yDim, float zDim) {

super(xDim, yDim, zDim, Primitive.*GENERATE\_NORMALS* | Primitive.*GENERATE\_TEXTURE\_COORDS*, *createAppearance*());

}

/\* Funzione che imposta aspetto dell'Abaco

\*

\* @return appearance

\*/

public static Appearance createAppearance() {

Appearance appearance = new Appearance();

Material material = new Material();

//Caricamento texture

String localPath = new File("").getAbsolutePath() + "\\src/Texture\\";

TextureLoader loader = new TextureLoader(localPath + "PietraColonna.jpg", "RGB", new Container());

Texture texture = loader.getTexture();

//Impostazione texture

appearance.setTexture(texture);

appearance.setMaterial(material);

return appearance;

}

}

Per creare la scalinata si è deciso di estendere la classe primitiva Box impostando al costruttore della superclasse, oltre che i valori per le varie dimensioni e le primitive per la creazione delle normali e delle coordinate per la texture anche un’appearance (ed associata texture) definita con il metodo createAppearance().

##### Codice relativo alla creazione della facciata

public class Facciata extends Group {

private float lunghezza;

private float altezza;

private TransformGroup[] TgScala;

private Transform3D[] t3dScala;

private TransformGroup[] TgColonna;

private Transform3D[] t3dColonna;

private Colonna[] colonna;

private Scale[] scala;

private Abaco abaco;

private TransformGroup TgAbaco;

private Transform3D t3dAbaco;

private Timpano timpano;

private TransformGroup TgTimpano;

private Transform3D t3dTimpano;

public Facciata() {

lunghezza = 0.6f;

altezza = 0.4f;

//Definizionen colonne e trasformazioni necessarie

colonna = new Colonna[6];

TgColonna = new TransformGroup[6];

t3dColonna = new Transform3D[6];

//Definizione scala e trasformazioni necessarie

scala = new Scale[3];

TgScala = new TransformGroup[3];

t3dScala = new Transform3D[3];

//Creazione e posizionamento Scalinata (Pavimento)

float scalaY = -0.3f;

float scalaturaScala = 0.0f;

for (int i = 0; i < scala.length; i++) {

scala[i] = new Scale(lunghezza-scalaturaScala, lunghezza/30, 0.1f);

//Imposto TG e transform3D per ogni parte del scala

TgScala[i] = new TransformGroup();

t3dScala[i] = new Transform3D();

t3dScala[i].setTranslation(new Vector3f(0.0f, scalaY, 0.78f));

TgScala[i].setTransform(t3dScala[i]); //imposto trasformazione

TgScala[i].addChild(scala[i]); //Aggiungo scala al TG

//Cambio valori per avere le trasformazioni corrette

scalaY = scalaY+lunghezza/15f;

scalaturaScala = scalaturaScala+lunghezza/30f;

}

scalaY = -0.3f;

//Creazione e posizionamento Colonne

float xColonna = -lunghezza/1.3f;

float yColonna = scalaY+2\*lunghezza/15+altezza/2;

float zColonna = lunghezza\*1.4f-scalaturaScala;

for (int i = 0; i < colonna.length; i++) {

colonna[i] = new Colonna(altezza);

TgColonna[i] = new TransformGroup();

t3dColonna[i] = new Transform3D();

t3dColonna[i].setTranslation(new Vector3f(xColonna, yColonna, zColonna));

TgColonna[i].setTransform(t3dColonna[i]);

TgColonna[i].addChild(colonna[i]);

if (i < 6) {

xColonna = xColonna + lunghezza/3.2f;

}

}

//Creazione e posizionamento Abaco

float xAbaco = lunghezza/60f;

//float zAbaco = lunghezza\*1.4f-scalaturaScala-lunghezza/11.5f;

float grandezzaXArc = lunghezza-scalaturaScala-2\*lunghezza/30f;

float grandezzaZArc = altezza/5;

abaco = new Abaco(grandezzaXArc, lunghezza/15, grandezzaZArc);

TgAbaco = new TransformGroup();

t3dAbaco = new Transform3D();

t3dAbaco.setTranslation(new Vector3f(xAbaco, yColonna+altezza/2+altezza/15+altezza/6, 0.80f));

TgAbaco.setTransform(t3dAbaco);

TgAbaco.addChild(abaco);

//Creazione e posizionamento Timpano

float yTimpano = yColonna+altezza/2+altezza/15+altezza/6;

float zTimpano = lunghezza\*1.4f-scalaturaScala-lunghezza/11.5f;;

timpano = new Timpano(2\*(lunghezza-scalaturaScala-2\*lunghezza/30f)/4f,2\*(lunghezza-scalaturaScala-2\*lunghezza/30f),altezza/3);

TgTimpano = new TransformGroup();

t3dTimpano = new Transform3D();

t3dTimpano.setTranslation(new Vector3f(lunghezza/60f,yTimpano+lunghezza/3.6f,0.78f));

TgTimpano.setTransform(t3dTimpano);

TgTimpano.addChild(timpano);

zTimpano = -zTimpano;

//Aggiungo alla scena i vari elementi

for(int i=0;i<scala.length;i++){ //aggiungo gli scalini uno a uno

addChild(TgScala[i]);

}

for(int y=0;y<colonna.length;y++){ //aggiungo le colonne una a una

addChild(TgColonna[y]);

}

addChild(TgAbaco); //Aggiungo abaco

addChild(TgTimpano); //Aggiungo timpano

}

}

Per creare la facciata sono stati usati due array TransformGroup per creare le colonne e la scalinata (dovendo gestire più elementi) e altri due TransformGroup per creare il timpano e l’abaco. I transform3D sono stati utilizzati per gestire la posizione dei vari oggetti, nel caso di colonne e scalinata ne viene applicato uno ad ogni elemento. Inoltre ogni elementi è fornito di una stessa texture opportuna per rendere il colore coerente con quello della pietra.

La classe “CreaFacciata”, non riportata nel codice ma utilizzata per creare l’intera scena, si occupa solamente di creare un oggetto TransformGroup TG a cui viene aggiunto come figlio la Facciata e di aggiungere il TransformGroup come figlio del nodo principale della scena ovvero il branchGroup.

##### Schermata



# ESERCIZIO 4.3

##### Traccia dell’esercizio

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.

##### Scenegraph

Transform3D

Background

Interpolator

BoundingSphere

Earth

BoundingSphere

Directional Light

Interpolator

BoundingSphere

BoundingSphere lightBoundingSphere

Transform3D

Luna

##### Codice

public class TerraLunaRotazione extends Applet {

public TerraLunaRotazione() {

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();

//Cambio punto di vista per visualizzazione corretta della scena

Transform3D transform = new Transform3D();

transform.lookAt(new Point3d(0.0, 0.0, 4.5),

new Point3d(0.0, 0.0, 0.0),

new Vector3d(0.0, 1.0, 0.0)

);

transform.invert();

TransformGroup vtg = simpleU.getViewingPlatform().getViewPlatformTransform();

vtg.setTransform(transform);

simpleU.addBranchGraph(scene);

}

/\*\*

\* Funzione che crea il sottografo principale

\*

\* @return branchGroup da aggiungere alla scena

\*/

public BranchGroup createSceneGraph() {

BranchGroup branchGroup = new BranchGroup();

/\*\*\*\*\*\*\*\*\*\*\*\*CREAZIONE BACKGROUND\*\*\*\*\*\*\*\*\*\*\*\*/

branchGroup.addChild(createBackGround());

/\*\*\*\*\*\*\*\*\*\*\*\*CREAZIONE TERRA\*\*\*\*\*\*\*\*\*\*\*\*/

Sphere terra = new Sphere(

0.8f,

Primitive.*GENERATE\_NORMALS* | Primitive.*GENERATE\_TEXTURE\_COORDS*,

200, //Livello di definizione della texure

terraAppearance()

);

rotazioneTerra(branchGroup, terra); //Gestisce la rotazione della terra

/\*\*\*\*\*\*\*\*\*\*\*\*CREAZIONE LUNA\*\*\*\*\*\*\*\*\*\*\*\*/

Sphere luna = new Sphere(

0.45f,

Primitive.*GENERATE\_NORMALS* | Primitive.*GENERATE\_TEXTURE\_COORDS*,

200, //Livello di definizione della texure

lunaAppearance()

);

rotazioneLuna(branchGroup, luna); //Gestisce la rotazione della luna

//Aggiungo luce del sole

branchGroup.addChild(luceSolare());

return branchGroup;

}

/\*\*

\* Funzione che imposta il background tramite una texture

\*

\* @return background da applicare scena

\*/

public Background createBackGround() {

String localPathStelle = new File("").getAbsolutePath()+\\src/Texture\\";

TextureLoader loaderStelle = new TextureLoader(localPathStelle+"spazio.jpg", "RGB", new Container());

ImageComponent2D image = loaderStelle.getImage();

//2) Impostazione background

Background background = new Background();

background.setImage(image);

//Stira l'immagine per coprire tutta l'ampiezza della finestra

background.setImageScaleMode(Background.*SCALE\_FIT\_MAX*);

//3) Raggio d'influenza del background sulla scena

BoundingSphere bounds = new BoundingSphere(new Point3d(), 1000.0);

background.setApplicationBounds(bounds);

return background;

}

/\*\*

\* Funzione che imposta textura alla sfera usata come terra

\*

\* @return appearance da applicare alla terra

\*/

public Appearance terraAppearance() {

Appearance appearance = new Appearance();

Material material = new Material();

//Dimunuzione lucentezza della terra (riflette meno la luce)

material.setShininess(80.0f);

material.setSpecularColor(new Color3f(0.0f, 0.0f, 0.0f));

appearance.setMaterial(material);

//Caricamento texture

String localPathTerra = new File("").getAbsolutePath() + "\\src/Texture\\";

TextureLoader loaderTerra = new TextureLoader(localPathTerra + "earth.jpg", "RGB", new Container());

Texture texture = loaderTerra.getTexture();

//Definizione attributi della texture ed impostazione texture

TextureAttributes attributes = new TextureAttributes();

attributes.setTextureMode(TextureAttributes.*COMBINE*);

appearance.setTexture(texture);

appearance.setTextureAttributes(attributes);

return appearance;

}

/\*\*

\* Funzione che imposta la rotazione della terra

\*

\* @return

\*/

public void rotazioneTerra(BranchGroup branchGroup, Sphere terra) {

TransformGroup rotazioneTerra = new TransformGroup();

Transform3D ruotaTerra = new Transform3D(); //Creazione rotazione della terra

ruotaTerra.rotY(-Math.*PI*/4.0f);

//(loopCount, increasingAlphaDuration) => (ciclo infinito, tempo in cui alpha passa da 0 a 1)

Alpha alpha = new Alpha(-1, 5000);

//Interpolator serve per creare animazioni basate sul tempo

//RotationInterpolator cambia l'orientamento di un oggetto (alpha, target, trasformazione, angolo partenza, angolo arrivo)

RotationInterpolator rotazioneTerraInterpolator = new RotationInterpolator(

alpha,

rotazioneTerra,

ruotaTerra,

0.0f,

(float)Math.*PI* \* 2.0f

);

//Bound relativo all'interpolator

BoundingSphere boundsTerra = new BoundingSphere();

rotazioneTerraInterpolator.setSchedulingBounds(boundsTerra);

//Permette il cambiamento della trasformazione associata al TransfomrGroup anche dopo la compilazione

rotazioneTerra.setCapability(TransformGroup.*ALLOW\_TRANSFORM\_WRITE*);

//Aggiunta della terra

rotazioneTerra.addChild(terra);

//Aggiunta dell'interpolator e della rotazione della terra

branchGroup.addChild(rotazioneTerraInterpolator);

branchGroup.addChild(rotazioneTerra);

}

/\*\*

\* Funzione che imposta l'orbita della luna

\*

\* @return

\*/

public void rotazioneLuna(BranchGroup branchGroup, Sphere luna) {

//TG generale

TransformGroup orbita = new TransformGroup();

//TG relativo alla posizione della luna

TransformGroup posizioneLuna = new TransformGroup();

//TG relativo all'orbita della luna

TransformGroup orbitaLuna = new TransformGroup();

//posizione della luna

Transform3D posLuna = new Transform3D();

posLuna.setTranslation(new Vector3d(2.5, 0.5, 0.0));

posizioneLuna.setTransform(posLuna);

//orbita della luna intorno alla terra

Transform3D orbitazioneLuna = new Transform3D();

orbitazioneLuna.rotY(-Math.*PI* / 4.0f);

//(loopCount, increasingAlphaDuration) => (ciclo infinito, tempo in cui alpha passa da 0 a 1)

Alpha alpha = new Alpha(-1, 5000);

//Interpolator serve per creare animazioni basate sul tempo

//RotationInterpolator cambia l'orientamento di un oggetto (alpha, target, trasformazione, angolo partenza, angolo arrivo)

RotationInterpolator orbitaLunainterpolator = new RotationInterpolator(

alpha,

orbitaLuna,

orbitazioneLuna,

0.0f,

(float)Math.*PI* \* 2.0f

);

//Bound relativo all'interpolator

BoundingSphere boundsLuna = new BoundingSphere();

orbitaLunainterpolator.setSchedulingBounds(boundsLuna);

//Permette il cambiamento della trasformazione associata al TransfomrGroup anche dopo la compilazione

orbitaLuna.setCapability(TransformGroup.*ALLOW\_TRANSFORM\_WRITE*);

//Aggiunta della luna

orbita.addChild(luna);

//Aggiunta dell'orbita

posizioneLuna.addChild(orbita);

//Aggiunta dell'interpolator e della posizione della luna rispetto l'origine

orbitaLuna.addChild(orbitaLunainterpolator);

orbitaLuna.addChild(posizioneLuna);

branchGroup.addChild(orbitaLuna);

}

/\*\*

\* Funzione che imposta textura alla sfera usata come luna

\*

\* @return appearance da applicare alla luna

\*/

public Appearance lunaAppearance() {

Appearance appearance = new Appearance();

Material material = new Material();

//Dimunuzione lucentezza della luna (riflette meno la luce)

material.setShininess(80.0f);

material.setSpecularColor(new Color3f(0.0f, 0.0f, 0.0f));

appearance.setMaterial(material);

//Caricamento texture

String localPathTerra = new File("").getAbsolutePath() + "\\src/Texture\\";

TextureLoader loaderTerra = new TextureLoader(localPathTerra + "moon.jpg", "RGB", new Container());

Texture texture = loaderTerra.getTexture();

//Definizione attributi della texture ed impostazione texture

TextureAttributes attributes = new TextureAttributes();

attributes.setTextureMode(TextureAttributes.*COMBINE*);

appearance.setTexture(texture);

appearance.setTextureAttributes(attributes);

return appearance;

}

/\*\*

\* Funzione che crea la luce del sole (settando anche una direzione)

\*

\* @return directLight da applicare alla terra

\*/

public DirectionalLight luceSolare() {

BoundingSphere lightBoundingSphere = new BoundingSphere(

new Point3d(0.0, 0.0, 0.0),

40.0

);

//Simulare la luce solare

DirectionalLight light = new DirectionalLight();

light.setDirection(new Vector3f(-4.5f, -1.0f, -1.5f));

light.setInfluencingBounds(lightBoundingSphere);

return light;

}

public static void main(String[] args) {

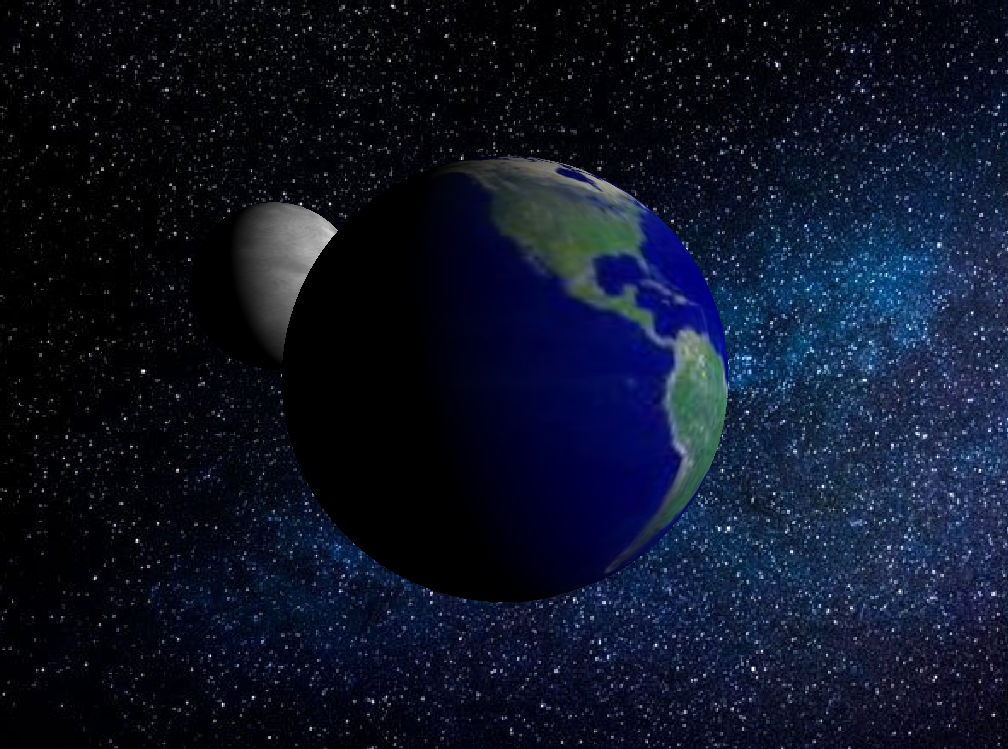
new MainFrame(new TerraLunaRotazione(), 1024, 768);

}

}

Si è scelto di scomporre il programma in più funzioni ognuna dedicata ad un compito preciso: gestire l’appearance della terra, della luna, gestire la rotazione della terra, della luna e relativa alla creazione della luce che simula quella solare. Per la rotazione della luna è stato necessario creare un TransformGroup generale, uno per la rotazione (con relativo Transform3D associato) ed uno per il posizionamento (con relativo Transform3D associato). La rotazione della Terra e della Luna avviene tramite la definizione di un RotationInterpolator (serve per creare animazioni basate sul tempo) al quale viene passato come parametro alpha con loopCount = -1, in questo modo l’animazione è basata su un ciclo infinito. Inoltre per permettere il cambiamento dei Transform3D associati ai rispettivi TransformGroup, sia per la Luna che per la Terra è stato necessario impostare la relativa capability ad “ALLOW\_TRANSFORM\_WRITE”.

##### Schermate



# ESERCIZIO 4.4

##### Traccia dell’esercizio

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

BoundingSphere

DirectLight

##### Scenegraph

Timpano [0]

Timpano [4]

Transform3D

TIMPANO

Scale [0]

Scale [4]

Transform3D

Col. [0]

Col. [26]

Transform3D

Abaco [0]

Abaco [4]

Transform3D

SCALE

COLONNE

ABACO

##### Codice relativo al Tempio

public class Tempio extends Group {

private float lunghezza;

private float altezza;

private TransformGroup[] TgScala;

private Transform3D[] t3dScala;

private Scale[] scala;

private TransformGroup[] TgColonna;

private Transform3D[] t3dColonna;

private Colonna[] colonna;

private Abaco[] abaco;

private TransformGroup[] TgAbaco;

private Transform3D[] t3dAbaco;

private Timpano[] timpano;

private TransformGroup[] TgTimpano;

private Transform3D[] t3dTimpano;

public Tempio() {

lunghezza = 0.6f;

altezza = 0.4f;

//Definizione elementi e array contenitivi

scala = new Scale[4];

TgScala = new TransformGroup[4];

t3dScala = new Transform3D[4];

colonna = new Colonna[26];

TgColonna = new TransformGroup[26];

t3dColonna = new Transform3D[26];

abaco = new Abaco[4];

TgAbaco = new TransformGroup[4];

t3dAbaco = new Transform3D[4];

timpano = new Timpano[2];

TgTimpano = new TransformGroup[2];

t3dTimpano = new Transform3D[2];

float scalaY = -0.3f;

float scalatura = 0.0f;

//Creazione e posizionamento Scalinata (Pavimento)

for (int i = 0; i < scala.length; i++) {

scala[i] = new Scale(lunghezza-scalatura, lunghezza/30, lunghezza\*1.5f-scalatura);

//Imposto TG e transform3D per ogni parte del scala

TgScala[i] = new TransformGroup();

t3dScala[i] = new Transform3D();

t3dScala[i].setTranslation(new Vector3f(0.0f, scalaY, 0.0f));

TgScala[i].setTransform(t3dScala[i]); //imposto trasformazione

TgScala[i].addChild(scala[i]); //Aggiungo scala al TG

//Cambio valori per avere le trasformazioni corrette

scalaY = scalaY+lunghezza/15f;

scalatura = scalatura+lunghezza/30f;

}

scalaY = -0.3f;

//Creazione e posizionamento Colonne

float xColonna = -lunghezza/1.3f;

float yColonna = scalaY+2\*lunghezza/15f+altezza/2f;

float zColonna = lunghezza\*1.4f-scalatura;

for (int i = 0; i < colonna.length; i++) {

colonna[i] = new Colonna(altezza);

TgColonna[i] = new TransformGroup();

t3dColonna[i] = new Transform3D();

if (i >= 6 && i <= 13) { //controlli sul n° colonne per la posizione

xColonna = -lunghezza/1.3f;

zColonna = zColonna-(lunghezza\*1.4f-scalatura)/4;

} else if (i >= 14 && i <= 18) {

xColonna = xColonna + lunghezza/3.2f;

zColonna = -lunghezza\*1.4f+scalatura;

} else if (i >= 19 && i <= 25) {

zColonna = zColonna+(lunghezza\*1.4f-scalatura)/4;

}

t3dColonna[i].setTranslation(new Vector3f(xColonna, yColonna, zColonna));

TgColonna[i].setTransform(t3dColonna[i]);

TgColonna[i].addChild(colonna[i]);

if (i < 6) {

xColonna = xColonna + lunghezza/3.2f;

}

}

//Creazione e posizionamento Abaco

float traslaXabaco = lunghezza/60f;

float traslaZabaco = lunghezza\*1.4f-scalatura-lunghezza/11.5f;

float xAbaco = lunghezza-scalatura-2\*lunghezza/20.0f;

float zAbaco = altezza/5;

for (int i = 0; i < abaco.length; i++) {

if (i == 1) { //controllo sul n° di abachi per la posizione giusta

traslaZabaco = -traslaZabaco;

} else if (i == 2) {

traslaXabaco = -lunghezza/1.3f+lunghezza/11.5f;

traslaZabaco = 0.0f;

xAbaco = altezza/5;

zAbaco = lunghezza\*1.5f-scalatura-2\*lunghezza/30f-altezza/2.7f;

} else if (i == 3) {

traslaXabaco = lunghezza/1.3f-lunghezza/19f;

}

abaco[i] = new Abaco(xAbaco, lunghezza/15, zAbaco);

TgAbaco[i] = new TransformGroup();

t3dAbaco[i] = new Transform3D();

t3dAbaco[i].setTranslation(new Vector3f(traslaXabaco, yColonna+altezza/2+altezza/15+altezza/6, traslaZabaco));

TgAbaco[i].setTransform(t3dAbaco[i]);

TgAbaco[i].addChild(abaco[i]);

}

//Creazione e posizionamento Timpano

float traslaYtimpano = yColonna+altezza/2+altezza/15+altezza/6;

float traslaZtimpano = lunghezza\*1.4f-scalatura-lunghezza/11.5f;

for (int i = 0; i < timpano.length; i++) {

timpano[i] = new Timpano(2\*(lunghezza-scalatura-2\*lunghezza/30f)/4f,2\*(lunghezza-scalatura-2\*lunghezza/30f),altezza/3);

TgTimpano[i] = new TransformGroup();

t3dTimpano[i] = new Transform3D();

t3dTimpano[i].setTranslation(new Vector3f(lunghezza/60f,traslaYtimpano+lunghezza/3.6f, traslaZtimpano));

TgTimpano[i].setTransform(t3dTimpano[i]);

TgTimpano[i].addChild(timpano[i]);

traslaZtimpano = -traslaZtimpano;

}

//Aggiungo singoli elementi

for(int i=0;i<scala.length;i++){ //aggiungo gli scalini uno a uno

addChild(TgScala[i]);

}

for(int i = 0; i < colonna.length; i++){ //aggiungo le colonne una a una

addChild(TgColonna[i]);

}

for(int i = 0; i < abaco.length; i++){ //aggiungo abachi uno ad uno

addChild(TgAbaco[i]);

}

for (int i = 0; i < timpano.length; i++) {

addChild(TgTimpano[i]); //Aggiungo timpani uno ad uno

}

}

}

##### Codice relativo alla creazione del tempio

public class CreaTempio extends Applet {

public CreaTempio() {

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();

//Transfomazione per visualizzazione della scena corretta

Transform3D transform = new Transform3D();

transform.lookAt(new Point3d(0.0, 0.0, 2.8),

new Point3d(0.0, 0.0, 0.0),

new Vector3d(0.0, 0.1, 0.0));

transform.invert();

TransformGroup vtg = simpleU.getViewingPlatform().getViewPlatformTransform();

vtg.setTransform(transform);

simpleU.addBranchGraph(scene);

//Possibilità ruotare visuale della scena tramite il mouse

OrbitBehavior orbit = new OrbitBehavior(canvas3D, OrbitBehavior.*REVERSE\_ROTATE*);

orbit.setSchedulingBounds(new BoundingSphere());

simpleU.getViewingPlatform().setViewPlatformBehavior(orbit);

}

public BranchGroup createSceneGraph() {

BranchGroup branchGroup = new BranchGroup();

//imposto la luce

luceDirezionale(branchGroup);

TransformGroup TG = new TransformGroup();

TG.addChild(new Tempio());

branchGroup.addChild(TG);

return branchGroup;

}

public void luceDirezionale(BranchGroup node){

BoundingSphere bounds = new BoundingSphere(new Point3d(0d,0d,0d),1d);

DirectionalLight directLight = new DirectionalLight();

directLight.setColor(new Color3f(1f,1f,1f));

directLight.setInfluencingBounds(bounds);

directLight.setDirection(new Vector3f(4f,0f,-5f));

node.addChild(directLight);

}

public static void main(String[] args) {

new MainFrame(new CreaTempio(), 1024, 768);

}

}

Le classi per la composizione del tempio non sono state riportate in quanto sono le stesse usate per la creazione della singola facciata senza alcuna modifica. Per creare il tempio sono stati usati quattro array TransformGroup per creare i singoli elementi (colonne, echini, abachi e scalinate).

Tramite l’OrbitBehavior è possibile usare il mouse per fare uno zoom in-out per esplorare liberamente la scena, sono stati proposti tre screenshot per mostrare questo comportamento.

##### Schermate

