

Università degli studi di Udine

Corso di Immagini e Multimedialità – 2016-17

prof.Vito Roberto

Relazione finale

Galante Gregorio
mat.: 120337

Premessa:

Per motivi di carattere lavorativo non ho potuto seguire i corsi durante l'anno accademico, per questo motivo gli esercizi sono stati svolti dopo lo studio del materiale didattico presente sul sito di e-learning e potrebbero contenere aggiunte non richieste dalla consegna (uso di Background, utilizzo di DirectionalLight), implementate al solo scopo di migliorare la presentazione della scena 3D rappresentata. Tali aggiunte non sono descritte negli scenegraph nel caso in cui non siano esplicitamente richieste nella consegna dell'esercizio.

I codici sorgenti di tutti gli esercizi svolti sono disponibili e consultabili in una repository GitHub al seguente indirizzo:

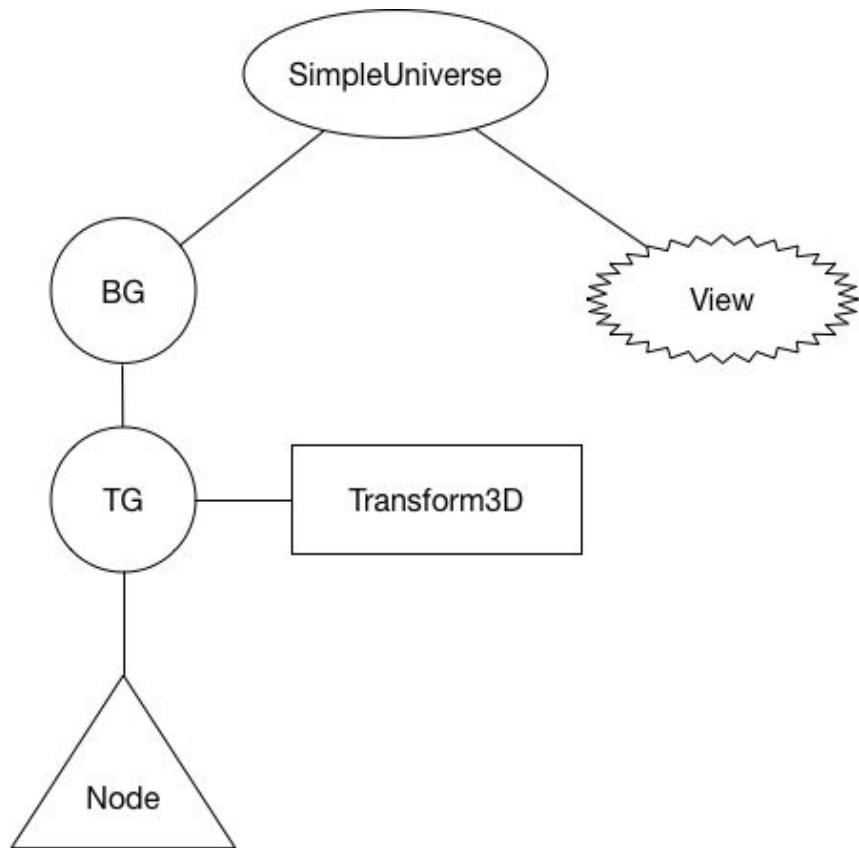
<https://github.com/gregogalante/java3d-exercises/tree/master/Relazione>

Esercizio 3.1

Creare una scena con un oggetto ColorCube e successivamente:

- (a) Effettuare le traslazioni lungo i singoli assi x, y, z
- (b) Effettuare le rotazioni attorno ai singoli assi x, y, z
- (c) Effettuare le scalature rispetto ai singoli assi x, y, z

Scenegraph



Codice

Applicazione delle traslazioni:

```
// This function creates and return a new branchgroup.  
private BranchGroup createBranchGroup() {  
    // initialize bg  
    BranchGroup bg = new BranchGroup();  
    // create main tg  
    TransformGroup tg = new TransformGroup();  
    tg.addChild(new ColorCube(0.3f));  
  
    // create a Transform3D object to apply the transformation  
    Transform3D translate = new Transform3D();  
    // use the function "setTranslation(Vector3d trans)" to create a transaltion  
    translate.setTranslation(new Vector3d(0.1f, 0.1f, 0.1f));  
    // apply the Transform3D object to the tg (with the funciton "setTransform(Transform3D t)")  
    tg.setTransform(translate);  
  
    // add tg to bg  
    bg.addChild(tg);  
    // return bg  
    return bg;  
}
```

Applicazione delle rotazioni:

```
// create a Transform3D object for every axis to apply the transformation  
Transform3D rotationX = new Transform3D();  
Transform3D rotationY = new Transform3D();  
Transform3D rotationZ = new Transform3D();  
// use the function "rotX", "rotY", "rotZ" to create a rotation  
rotationX.rotX(Math.PI * 0.1d);  
rotationY.rotY(Math.PI * 0.1d);  
rotationY.mul(rotationX);  
rotationZ.rotZ(Math.PI * 0.1d);  
rotationZ.mul(rotationY);  
// apply the Transform3D object to the tg (with the funciton "setTransform(Transform3D t)")  
tg.setTransform(rotationZ);
```

Applicazioni delle scalature:

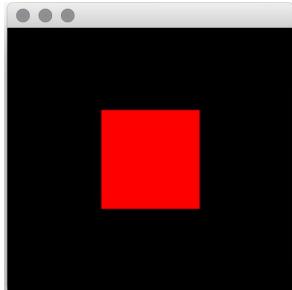
```
// create a Transform3D object to apply the transformation  
Transform3D scale = new Transform3D();  
// use the function "setScale(double value)" to scale the object  
scale.setScale(1.5d);  
// apply the Transform3D object to the tg (with the funciton "setTransform(Transform3D t)")  
tg.setTransform(scale);
```

Motivazioni

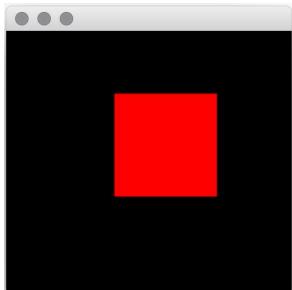
Ho impostato le trasformazioni gestite attraverso oggetti della classe Transform3D al TransformGroup legato al BranchGroup in modo da applicare tali variazioni a tutte le sue foglie (in questo caso l'oggetto ColorCube).

Risultato

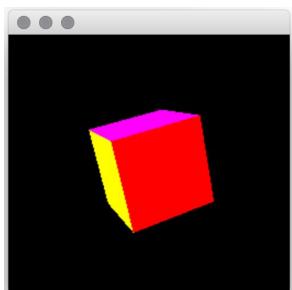
Situazione iniziale:



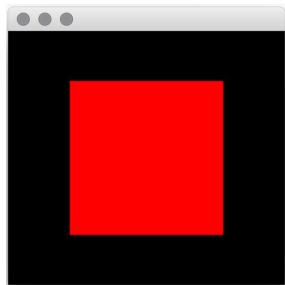
Applicazione delle traslazioni (valori 0.1f su tutti gli assi):



Applicazione delle rotazioni:



Applicazione delle scalature:



Conclusione

Attraverso oggetti della classe Transform3D è possibile applicare delle trasformazioni a tutti i figli di un TransformGroup.

In particolare la funzione setTranslate() di un oggetto Transform3D permette di eseguire una traslazione; le funzioni rotX(), rotY(), rotZ() permettono di eseguire delle rotazioni intorno ai singoli assi; la funzione setScale() permette di eseguire delle scalature.

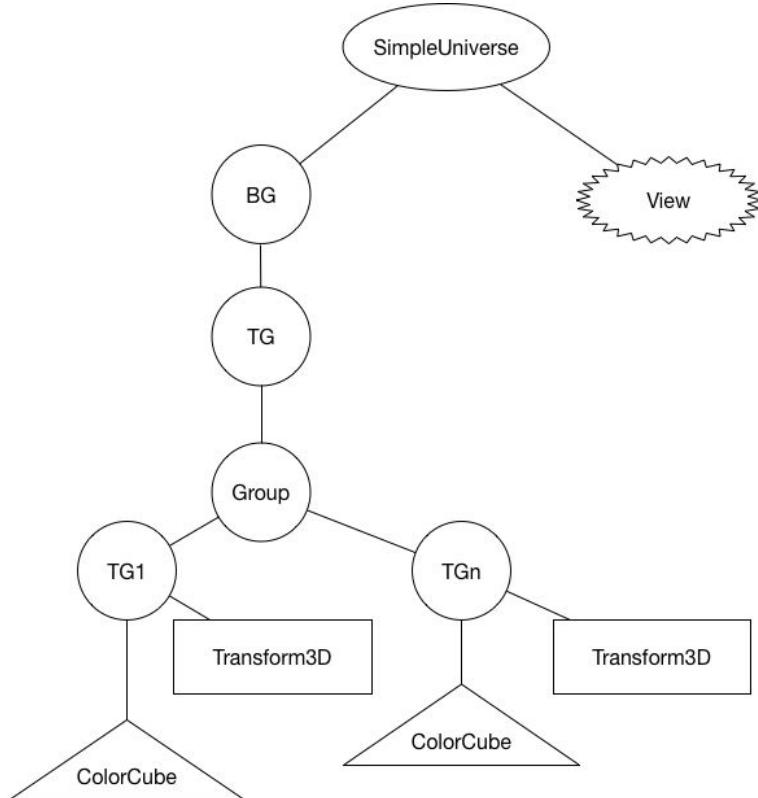
Per applicare un oggetto Transform3D ad un oggetto TransformGroup è possibile utilizzare la funzione setTransform() che richiede come parametro l'oggetto Transform3D.

In un oggetto TransformGroup è possibile applicare una sola trasformazione, per questo motivo è possibile unire due oggetti Transform3D attraverso la funzione mul() (si veda il codice delle rotazioni).

Esercizio 3.2

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

Scenegraph



NOTE: Con TG1 e TGn si indica che l'oggetto Group può contenere un numero variabile di figli uguali.

Codice

Creazione del BranchGroup:

```
// This function creates and return a new branchgroup.
private BranchGroup createBranchGroup() {
    // initialize bg
    BranchGroup bg = new BranchGroup();
    // create main tg
    TransformGroup tg = new TransformGroup();
    tg.addChild(new ColorCubeCircle(8));
    // add tg to bg
    bg.addChild(tg);
    // return bg
    return bg;
}
```

Costruttore classe ColorCubeCircle:

```
protected int number;
protected double theta;
protected double size;

public ColorCubeCircle(int number) {
    // check number is valid
    if (number < 1) {
        throw new RuntimeException("number should be one or more");
    }
    // set variables
    this.number = number;
    this.theta = calculateTheta();
    this.size = calculateSize();
    // create cubes
    for (int i = 0; i < this.number; i++) {
        createCube(i);
    }
}
```

Funzione createCube() della classe ColorCubeCircle:

```
protected void createCube(int position) {
    TransformGroup tg = new TransformGroup();
    Transform3D translate = new Transform3D();
    // define angle for the current counter
    double thetaCounter = position * this.theta;
    // define translation for transform3D
    translate.setTranslation(new Vector3d(
        Math.cos(thetaCounter + Math.PI / 2) / 2,
        Math.sin(thetaCounter + Math.PI / 2) / 2,
        0
    ));
    // add transform3D to the transform
    tg.setTransform(translate);
    // add new Colorcube to transform
    tg.addChild(new ColorCube(this.size));
    addChild(tg);
}
```

Motivazioni

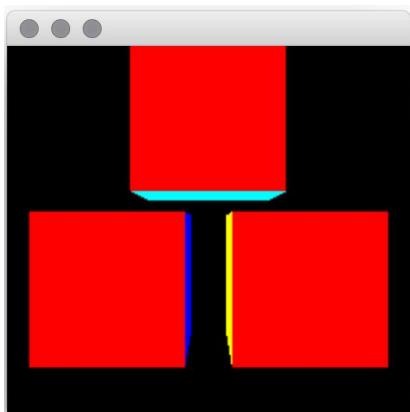
Ho utilizzato una classe esterna ColorCubeCircle (estensione di Group) per mantenere il codice più pulito e riutilizzabile.

Il numero di oggetti TransformGroup usati per visualizzare i cubi traslati è variabile rispetto al parametro “number” passato al costruttore di ColorCubeCircle.

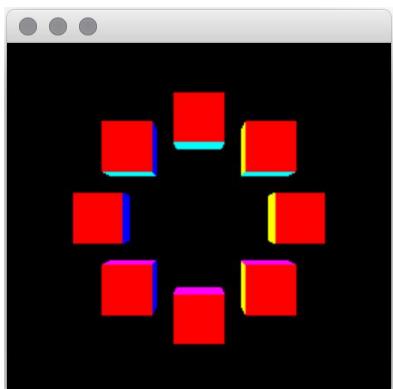
Le variabili theta e size permettono di variare il posizionamento dei cubi e la loro dimensione rispetto al numero che viene scelto.

Risultato

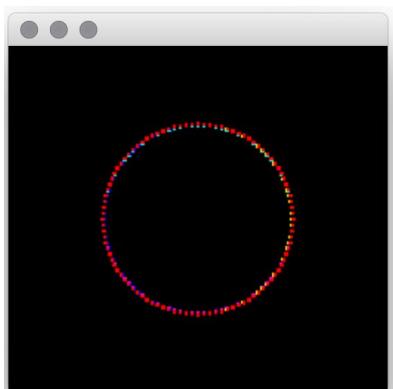
Risultato con 3 cubi:



Risultato con 8 cubi:



Risultato con 100 cubi:



Conclusione

Come per l'esercizio 3.1a è stato possibile applicare delle traslazioni ad un oggetto TransformGroup (e ai suoi figli) attraverso un oggetto Transform3D e la sua funzione setTranslate().

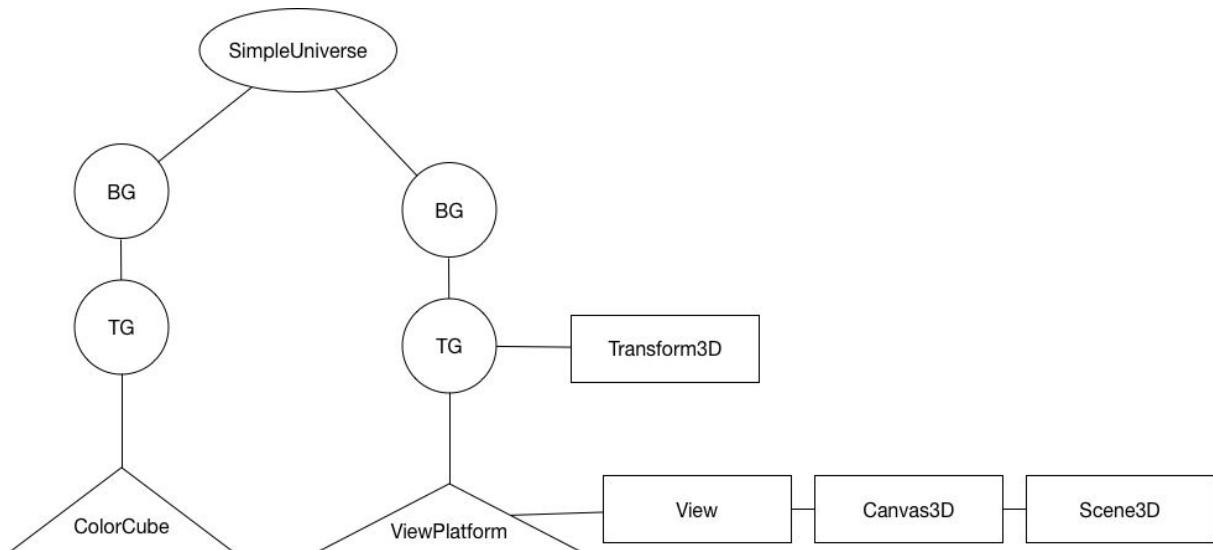
Esercizio 3.3

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

Impostazione del BranchGroup e chiamata delle funzione di lookAt:

```
// create branch group
BranchGroup branchGroup = createBranchGroup();

// translate user position
translateLookAt(universe, 1.0f, 1.0f, 4.0f);
```

Funzione translateLootAt():

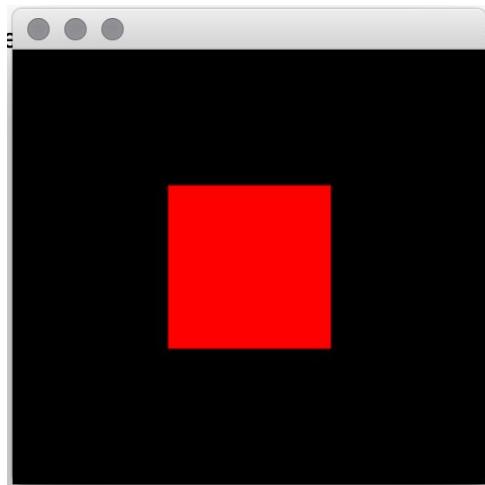
```
// This function translates the user position on the world.
private void translateLookAt(SimpleUniverse universe, float x, float y, float z) {
    // find viewing platform
    ViewingPlatform viewingPlatform = universe.getViewingPlatform();
    // find view
    View view = universe.getViewer().getView();
    // find transformgroup for view
    TransformGroup vptg = viewingPlatform.getViewPlatformTransform();
    // create lookat transformation
    Transform3D lookAtT3D = new Transform3D();
    lookAtT3D.lookAt(
        new Point3d(x, y, z),
        new Point3d(0.0, 0.0, 0.0),
        new Vector3d(0.0, 1.0, 0.0)
    );
    lookAtT3D.invert();
    // set lookat transformation to tg
    vptg.setTransform(lookAtT3D);
}
```

Motivazioni

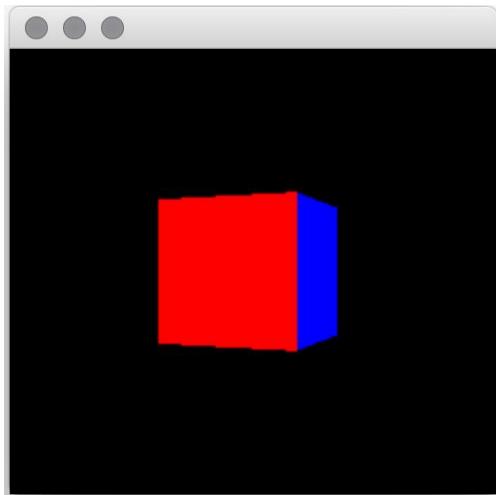
L'utilizzo di lookAt() è stato implementato in una funzione a parte per semplificare la lettura del codice.

Risultato

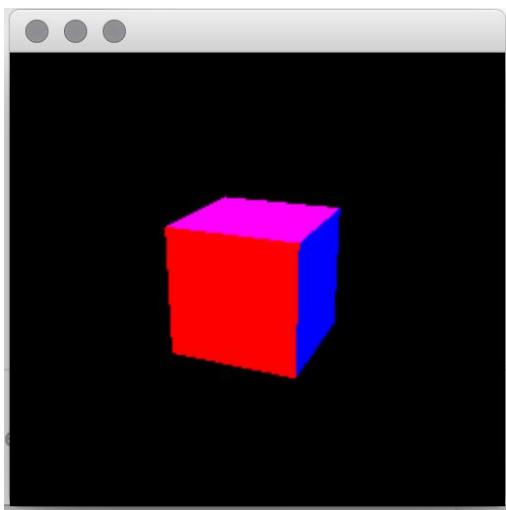
Situazione iniziale:



Visualizzazione con due punti di fuga:



Visualizzazione con tre punti di fuga:



Conclusion

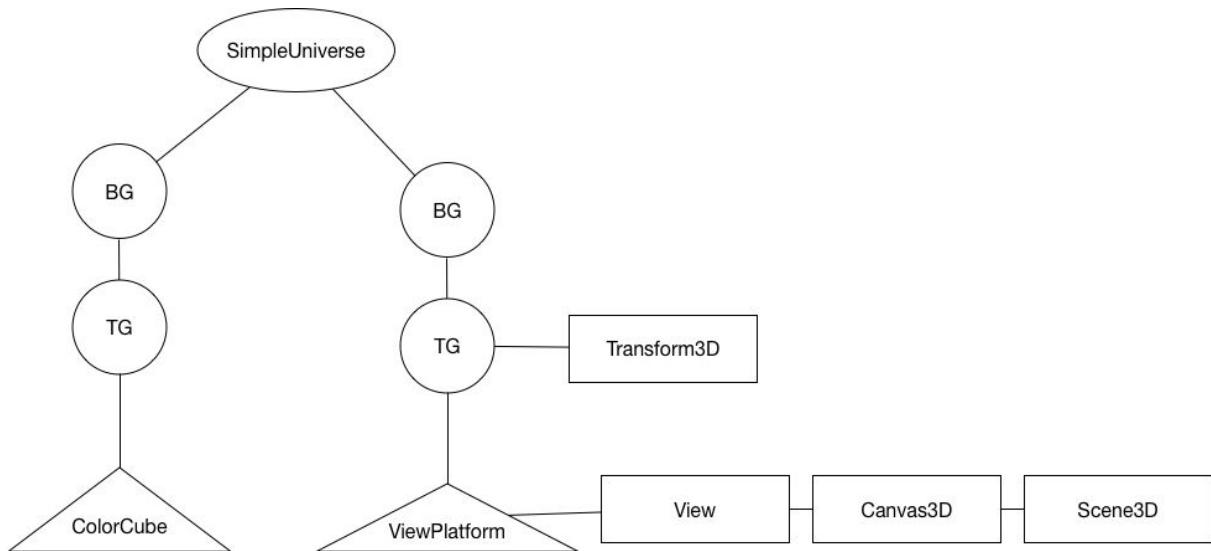
Attraverso la funzione `lookAt` è possibile modificare il punto di vista dell'osservatore. In particolare il primo parametro della funzione permette di traslare nei tre assi la posizione dell'osservatore.

Esercizio 3.4

Riprendendo dall'esercizio precedente una scena 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



Codice

Funzione per l'applicazione di una proiezione prospettica:

```
private void applyProjection(SimpleUniverse universe) {
    // find view of the universe
    View view = universe.getViewer().getView();
    view.setCompatibilityModeEnable(true);
    // create transformation for projection
    Transform3D otg = new Transform3D();
    double ratio = 1024.0/768.0;
    otg.perspective((Math.PI / 4), ratio, 0, 2.4);
    // use set left projection
    view.setLeftProjection(otg);
}
```

Funzione per l'applicazione di una proiezione ortografica:

```
private void applyProjection(SimpleUniverse universe) {  
    // find view of the universe  
    View view = universe.getViewer().getView();  
    view.setCompatibilityModeEnable(true);  
    // create transformation for projection  
    Transform3D otg = new Transform3D();  
    otg.ortho(-1, 1, -1, 1, 1, 4);  
    // use set left projection  
    view.setLeftProjection(otg);  
}
```

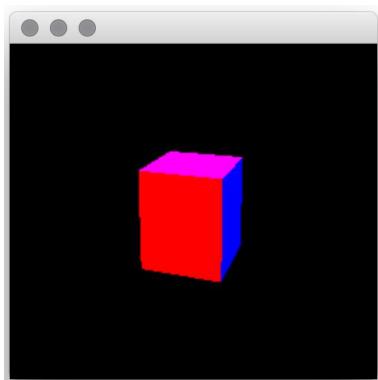
Motivazioni

Nel caso della proiezione prospettica è stata modificata la distanza focale e l'apertura angolare.

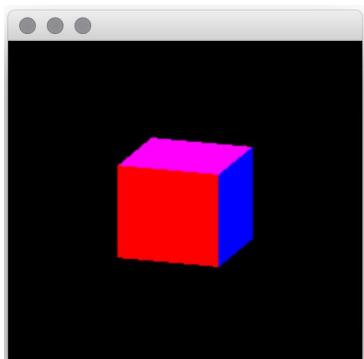
Nel caso della proiezione ortografica sono stati modificati i vari valori della proiezioni modificando l'aspetto percepito del cubo.

Risultato

Applicazione della proiezione prospettica:



Applicazione della proiezione ortografica:



Conclusione

Attraverso la funzione setLeftProjection() è possibile applicare una oggetto Transform3D all'oggetto View dell'osservatore.

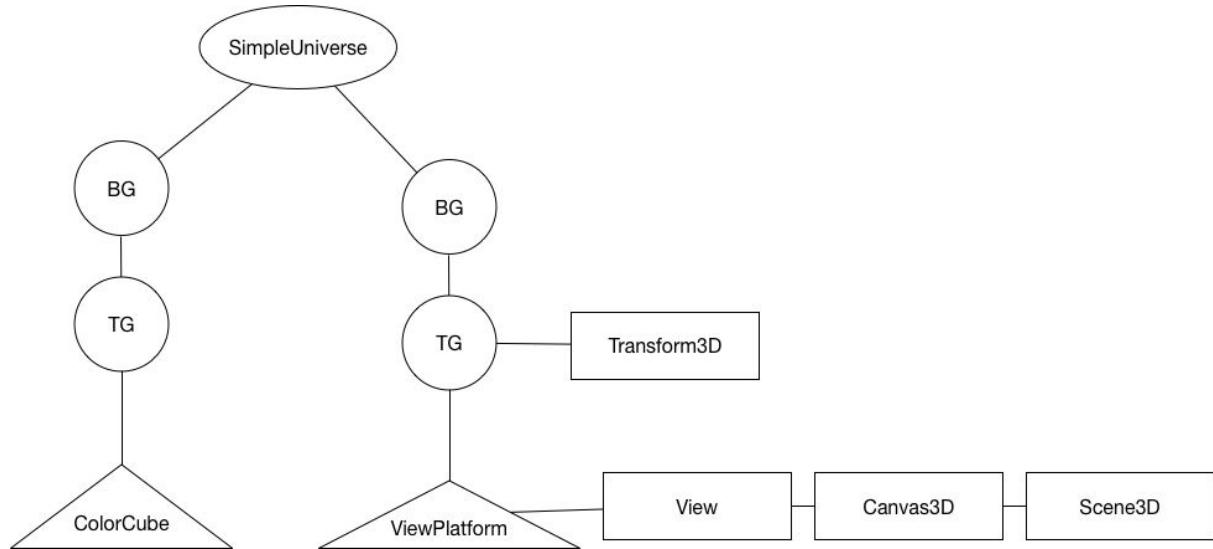
In particolare, utilizzando la funzione perspective() di Transform3D è possibile modificare l'angolo di visuale, il ratio e la distanza focale; attraverso la funzione ortho() è possibile applicare una trasformazione ortografica dell'immagine percepita.

Esercizio 3.5

Sperimentare le stesse proiezioni dell'esercizio 3.4 con gli opportuni metodi di View. Riprendendo dall'esercizio precedente una scena 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).

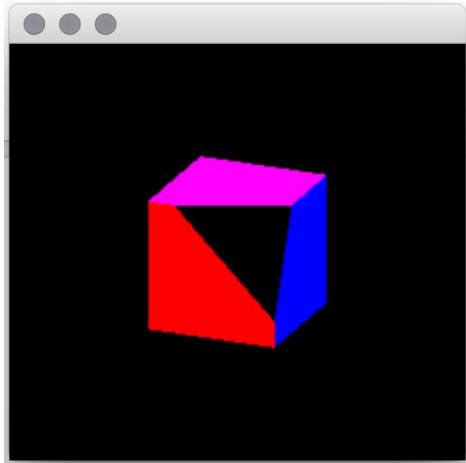
Scenegraph



Codice

```
private void applyProjection(SimpleUniverse universe) {
    // find view
    View view = universe.getViewer().getView();
    // change back distances
    view.setBackClipDistance (9.0);
    // change front distances
    view.setFrontClipDistance (0.45);
    // set field of view
    view.setFieldOfView(Math.PI/2);
    // set type of projection
    view.setProjectionPolicy(View.PARALLEL_PROJECTION);
}
```

Risultato



Conclusione

Attraverso le funzioni dell'oggetto View è stato possibile modificare la percezione delle immagini da parte dell'utente.

Attraverso la funzione setBackClipDistance() è stata modificata la distanza focale massima.

Attraverso la funzione setFrontClipDistance() è stata modificata la distanza focale minima.

Attraverso la funzione setFieldOfView() è stato modificato l'angolo di visuale.

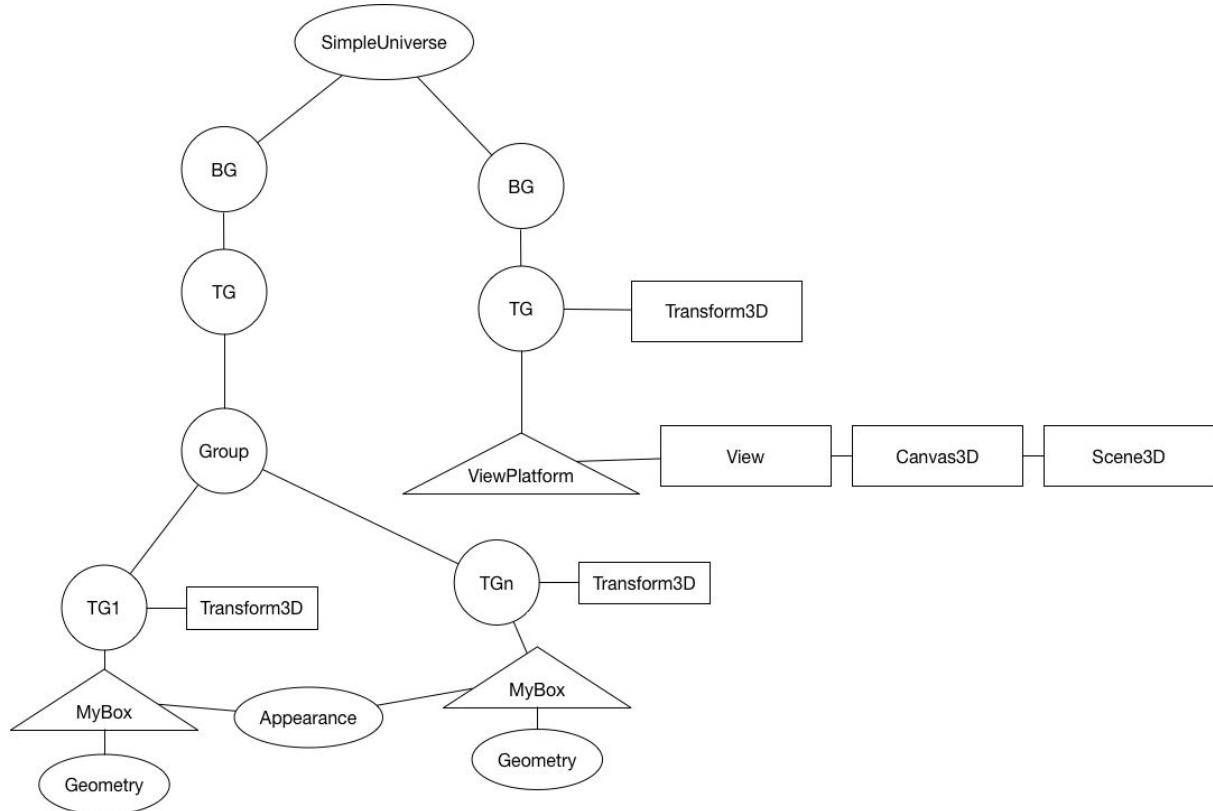
Attraverso la funzione setProjectionPolicy() è stata variata la policy di visualizzazione della proiezione.

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



NOTE: Con TG1 e TGn si indica che l'oggetto Group può contenere un numero variabile di figli uguali.

Codice

Funzione per la creazione del BranchGroup:

```
// This function creates and return a new branchgroup.
private BranchGroup createBranchGroup() {
    // initialize bg
    BranchGroup bg = new BranchGroup();
    // create main tg
    TransformGroup tg = new TransformGroup();
    tg.addChild(new MayaPyramid(4.0f));
    // translate pyramid
    Transform3D translate = new Transform3D();
    translate.setTranslation(new Vector3d(0.0f, -1.0f, 0.0f));
    tg.setTransform(translate);
    // add tg to bg
    bg.addChild(tg);
    // return bg
    return bg;
}
```

Costruttore classe MayaPyramid:

```
public MayaPyramid(float size) {
    this.unit = (size / 15);
    this.appearance = createAppearance();
    createLevels();
}
```

Funzioni di MayaPyramid per la creazione degli oggetti MyBox che compongono la piramide:

```
protected void createLevels() {
    for (int i = 0; i < 10; i++) {
        addChild(createLevel(i));
    }
}

protected TransformGroup createLevel(int counter) {
    TransformGroup tg = new TransformGroup();
    // create box (change size for the last box)
    float bottomWidth = (counter == 9) ? ((15 - counter - 2) * this.unit) : ((15 - counter) * this.unit);
    float topWidth = (counter == 9) ? bottomWidth : (bottomWidth - (this.unit / 2));
    float height = (counter == 9) ? (this.unit * 2) : this.unit;
    MyBox box = new MyBox(topWidth, topWidth, bottomWidth, bottomWidth, height, this.appearance);
    tg.addChild(box);
    // translate box
    Transform3D translate = new Transform3D();
    translate.setTranslation(new Vector3d(0.0f, (counter * this.unit), 0.0f));
    tg.setTransform(translate);
    // return tg
    return tg;
}
```

Costruttore classe MyBox:

```
protected Point3d [] points = new Point3d[8];
protected Point3d [] vectors = new Point3d[14];
protected TriangleStripArray triangleStrip = null;

public MyBox(float topWidth, float topLength, float bottomWidth, float bottomLength,
            float height, Appearance appearance) {
    if (appearance == null) {
        appearance = createAppearance();
    }
    // define points
    points[0] = new Point3d(-(topWidth / 2), +(height / 2), +(topLength / 2));
    points[1] = new Point3d(+((topWidth / 2), +(height / 2), +(topLength / 2));
    points[2] = new Point3d(-(bottomWidth / 2), -(height / 2), +(bottomLength / 2));
    points[3] = new Point3d(+((bottomWidth / 2), -(height / 2), +(bottomLength / 2));
    points[4] = new Point3d(+((topWidth / 2), +(height / 2), -(topLength / 2));
    points[5] = new Point3d(-((topWidth / 2), +(height / 2), -(topLength / 2));
    points[6] = new Point3d(+((bottomWidth / 2), -(height / 2), -(bottomLength / 2));
    points[7] = new Point3d(-((bottomWidth / 2), -(height / 2), -(bottomLength / 2));
    // define vectors
    vectors[0] = points[1]; vectors[1] = points[0]; vectors[2] = points[2];
    vectors[3] = points[5]; vectors[4] = points[7]; vectors[5] = points[6];
    vectors[6] = points[2]; vectors[7] = points[3]; vectors[8] = points[1];
    vectors[9] = points[6]; vectors[10] = points[4]; vectors[11] = points[5];
    vectors[12] = points[1]; vectors[13] = points[0];
    // set geometry
    int [] stripCounts = {{vectors.length}};
    triangleStrip = new TriangleStripArray(vectors.length, GeometryArray.COORDINATES, stripCounts);
    triangleStrip.setCoordinates(0, vectors);
    setGeometry(triangleStrip);
    // set geometry info
    GeometryInfo gInfo = new GeometryInfo(triangleStrip);
    NormalGenerator normgen = new NormalGenerator();
    normgen.generateNormals(gInfo);
    setGeometry(gInfo.getGeometryArray());
    // manage appearance
    setAppearance(appearance);
}
```

Motivazioni

La classe MayaPyramid estende Group in modo da poter essere utilizzata per raggruppare diversi oggetti TransformGroup, ognuno contenente una foglia MyBox usata per rappresentare un singolo piano della piramide.

Il parametro “size” di MayaPyramid permette di indicare il lato della piramide (la quale ha base quadrata) e viene anche utilizzato per calcolare il resto delle dimensioni usate in modo tale da mantenere le giuste proporzioni della figura.

La funzione createLevel(), richiamata per ogni livello della piramide, istanzia un oggetto MyBox, esegue una traslazione e ritorna l’oggetto TransformGroup relativo.

La classe MyBox permette di creare un tronco di piramide specificando nel costruttore le dimensioni delle singole basi, l’altezza e l’appearance che deve essere applicata (se impostata a null viene utilizzata una appearance di default).

Risultato

Rappresentazione di un singolo oggetto MyBox con base maggiore 1, base minore 2 e altezza 2:



Rappresentazione della piramide risultante da diverse posizioni:





Conclusione

Utilizzando una classe che estende Shape3D è stato possibile costruire una figura definendo i punti dei suoi vertici.

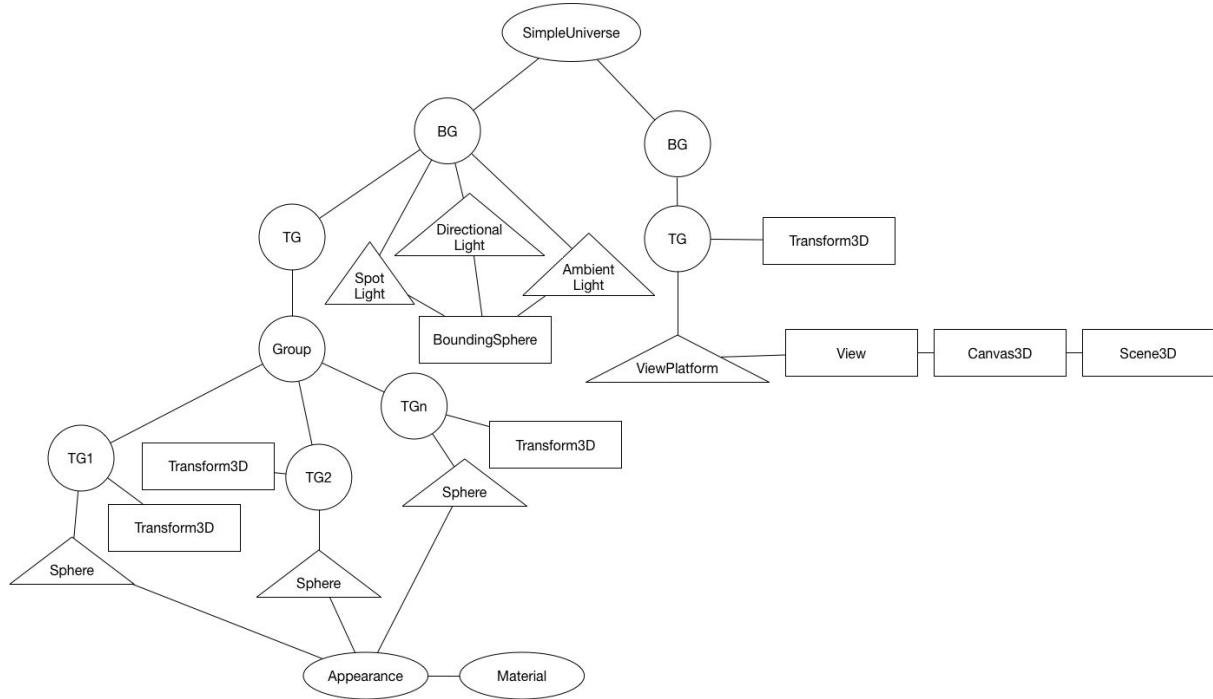
Per definire i piani della figura è stato utilizzato un oggetto di tipo TriangleStripArray.

Utilizzando una classe che estende Group è stato possibile definire una figura composta da più sottofigure.

Esercizio 3.7

Testare le luci su una matrice 5x5 di sfere.

Scenegraph



NOTE: Con TG1, TG2 e TGn si indica che l'oggetto Group può contenere un numero variabile di figli uguali.

Codice

Funzione per la creazione del BranchGroup:

```
// This function creates and return a new branchgroup.
private BranchGroup createBranchGroup() {
    // initialize bg
    BranchGroup bg = new BranchGroup();
    // create main tg
    TransformGroup tg = new TransformGroup();
    tg.addChild(new SphereMatrix(10)); // <-- NOTE: edit
    // add tg to bg
    bg.addChild(tg);
    // return bg
    return bg;
}
```

Costruttore classe SphereMatrix:

```
public SphereMatrix(int sphereNumber) {
    if (sphereNumber < 1) {
        throw new RuntimeException("sphereNumber should be one or more");
    }
    this.number = sphereNumber;
    // initialize appearance
    this.appearance = createAppearance();
    // initialize spheres radius
    this.radius = createRadius();
    // create spheres
    createSpheres();
}
```

Funzione di SphereMatrix per la creazione dell'Appearance:

```
protected Appearance createAppearance() {
    Appearance app = new Appearance();
    Material material = new Material();
    app.setMaterial(material);
    return app;
}
```

Funzioni di SphereMatrix per la creazione delle sfere che compongono la matrice:

```
protected void createSpheres() {
    for (int y = 0; y < this.number; y++) {
        for (int x = 0; x < this.number; x++) {
            addChild(createSphereTG(x, y));
        }
    }
}

protected TransformGroup createSphereTG(int x, int y) {
    TransformGroup tg = new TransformGroup();
    Sphere s = new Sphere(
        this.radius,
        Primitive.GEOMETRY_NOT_SHARED|Primitive.GENERATE_NORMALS,
        this.appearance
    );
    Transform3D t3d = new Transform3D();
    t3d.setTranslation(calculateSphereTranslation(x, y));
    tg.setTransform(t3d);
    tg.addChild(s);
    return tg;
}
```

Funzione utilizzata nel Main per testare la luce ambientale:

```
// This function add an ambient light to the world.  
private void addAmbientLight(BranchGroup branchGroup) {  
    // create light  
    AmbientLight light = new AmbientLight();  
    // add bounds to light  
    light.setInfluencingBounds(this.defaultBound);  
    // add light to tg  
    branchGroup.addChild(light);  
}
```

Funzioni utilizzate nel Main per testare la luce direzionale:

```
private void addDirectionalLight(BranchGroup branchGroup) {  
    addDirectionalLight(branchGroup, -1.0f, -0.5f, 1.0f);  
}  
  
// This function add a directional light to the world with a custom direction.  
private void addDirectionalLight(BranchGroup branchGroup, float dirX, float dirY, float dirZ) {  
    // create light  
    DirectionalLight light = new DirectionalLight();  
    light.setDirection(dirX, dirY, dirZ);  
    // add bounds to light  
    light.setInfluencingBounds(this.defaultBound);  
    // add light to tg  
    branchGroup.addChild(light);  
}
```

Funzione utilizzata nel Main per l'applicazione di una luce di tipo SpotLight:

```
// This function add a spot light with default position and angle.  
private void addSpotLight(BranchGroup branchGroup) {  
    addSpotLight(branchGroup, 0.0f, 0.0f, 1.0f, (float)Math.PI/6.0f);  
}  
  
// This function add a spot light to the world with custom position and angle.  
private void addSpotLight(BranchGroup branchGroup, float posX, float posY, float posZ, float angle) {  
    // create light  
    SpotLight light = new SpotLight();  
    light.setPosition(new Point3f(posX, posY, posZ));  
    light.setSpreadAngle(angle);  
    // add bounds to light  
    light.setInfluencingBounds(this.defaultBound);  
    // add light to tg  
    branchGroup.addChild(light);  
}
```

Motivazioni

La classe SphereMatrix estende Group in modo da poter essere utilizzata per raggruppare un insieme di oggetti Sphere.

La funzione createAppearance() di SphereMatrix è utilizzata per creare un oggetto Appearance applicato alle varie sfere che vanno a comporre la matrice.

La funzione createSphereTG() crea e ritorna un oggetto TransformGroup. A tale oggetto viene assegnato come figlio un oggetto Sphere e viene applicata una trasformazione che trasla la sfera nella posizione corretta della matrice.

Attraverso i valori usati nel costruttore di Sphere

"Primitive.GEOMETRY_NOT_SHARED|Primitive.GENERATE_NORMALS" si imposta la normale della sfera in modo tale da permettergli di riflettere la luce e quindi subire il suo effetto.

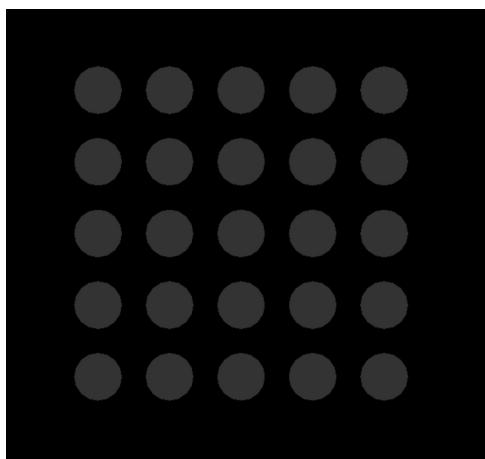
Per l'applicazione di una luce ambientale è stato istanziato un oggetto AmbientLight al quale viene impostato un BoundingSphere dove operare (definito precedentemente).

Per l'applicazione della luce direzionale è stato istanziato un oggetto DirectionalLight al quale viene impostato un BoundingSphere dove operare (definito precedentemente). Inoltre è stata utilizzata la funzione setDirection() per modificare la direzione di default della luce.

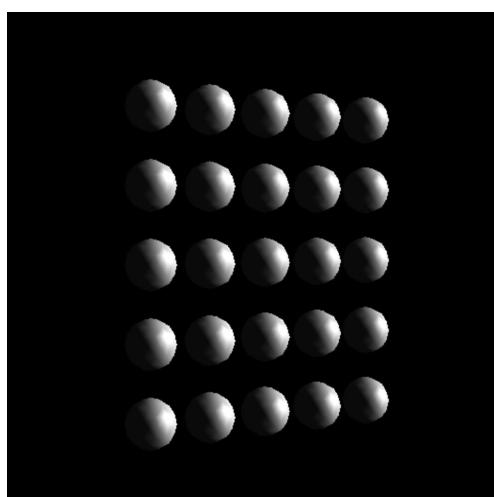
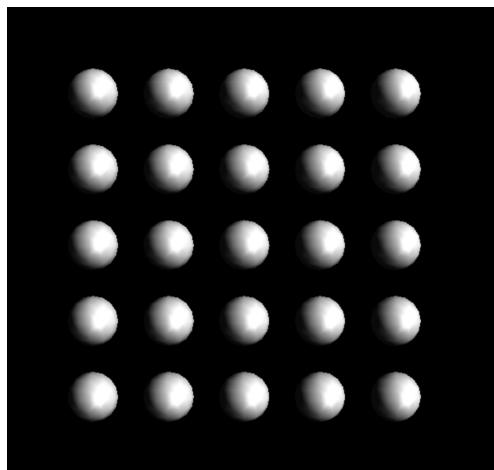
Per l'applicazione della luce di tipo spotlight è stato istanziato un oggetto di tipo SpotLight al quale viene impostato un BoundingSphere dove operare (definito precedentemente). Inoltre sono state utilizzate le funzioni setPosition() e setAngle() per impostare la posizione dell'origine della luce e l'angolo della sua propagazione.

Risultato

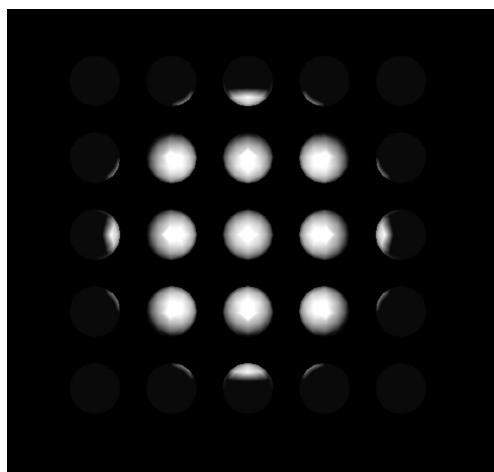
Applicazione di una luce ambientale:



Applicazione di una luce direzionale:



Applicazione di una luce di tipo SpotLight:



Conclusione

Le classi messe a disposizione da Java3D hanno permesso di testare diverse luci sulla matrice di sfere costruita da SphereMatrix.

Per permettere il corretto comportamento delle sfere rispetto alla luce è stato applicato un oggetto Material all'appearance.

Esercizio 3.8

Realizzare la raffigurazione 3D di una colonna composta da:

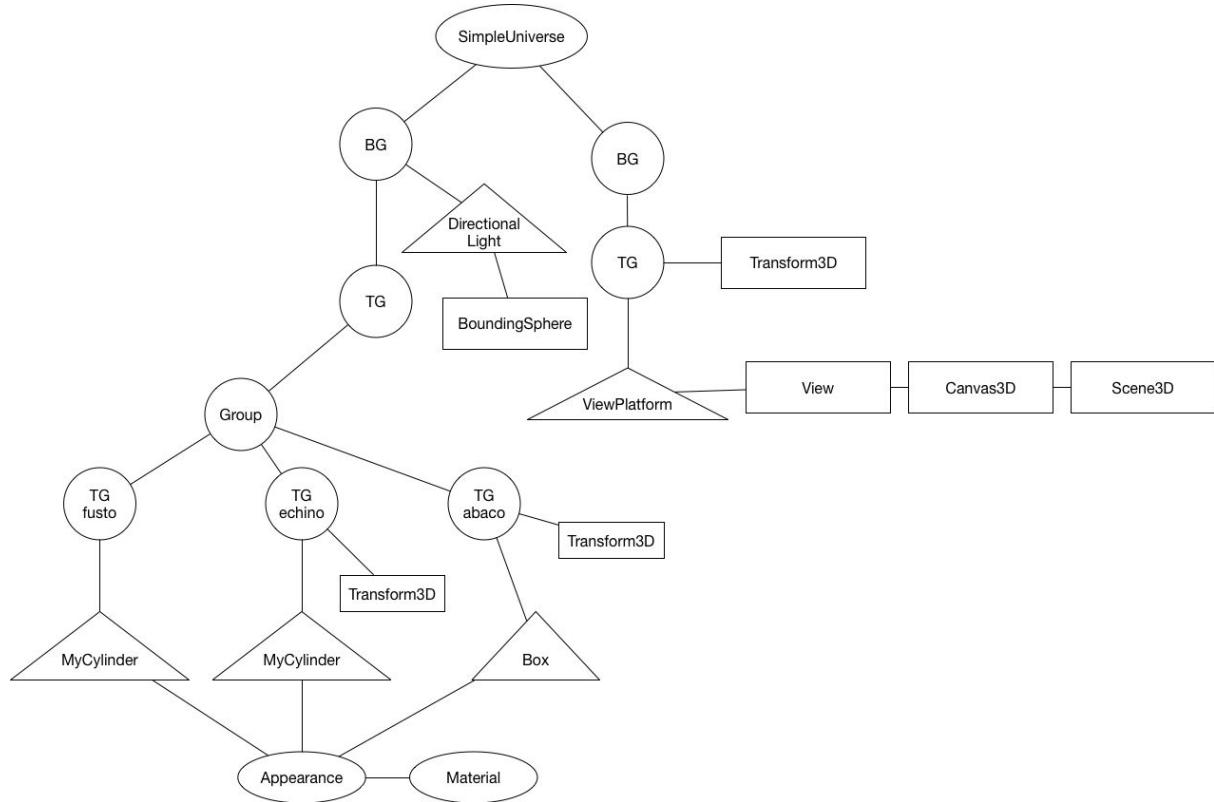
- Fusto: basato su un cilindro (trascurando l'entasi, si può usare la Primitive apposita).
- Echino: un tronco di cono (si può ricavare dall'esempio di cilindro già fornito variando i raggi superiore ed inferiore).
- Abaco: un parallelepipedo.

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).

Scenegraph



Codice

Funzione per la creazione del BranchGroup:

```
// This function creates and return a new branchgroup.  
private BranchGroup createBranchGroup() {  
    // initialize bg  
    BranchGroup bg = new BranchGroup();  
    // create main tg  
    TransformGroup tg = new TransformGroup();  
    tg.addChild(new Colonna(2.0f)); // <-- NOTE: edit here  
    // add tg to bg  
    bg.addChild(tg);  
    // return bg  
    return bg;  
}
```

Costruttore della classe Colonna:

```
// Colonna constructor 1.  
public Colonna(float height) {  
    this(height, null);  
}  
  
// Colonna constructor 2.  
public Colonna(float height, Appearance appearance) {  
    // define object values  
    this.unit = height / 10;  
    this.colonnaAppearance = (appearance == null) ? createAppearance() : appearance;  
    // add children to group  
    addChild(createFusto());  
    addChild(createEchino());  
    addChild(createAbaco());  
}
```

Funzione della classe Colonna per la creazione dell'appearance:

```
// This function creates and returns an appearance object for the object.  
protected Appearance createAppearance() {  
    Appearance appearance = new Appearance();  
    // add material  
    Material material = new Material();  
    material.setAmbientColor(239/255f,224/255f,203/255f);  
    material.setDiffuseColor(239/255f,224/255f,203/255f);  
    appearance.setMaterial(material);  
    // set polygon attributes  
    appearance.setPolygonAttributes(new PolygonAttributes(  
        PolygonAttributes.POLYGON_FILL,  
        PolygonAttributes.CULL_NONE,  
        0  
    ));  
    // return the appearance  
    return appearance;  
}
```

Funzione della classe Colonna per la creazione del fusto:

```
// This function generates and returns a Fusto object based on the given height.
protected TransformGroup createFusto() {
    TransformGroup tg = new TransformGroup();
    MyCylinder fusto = new MyCylinder(
        this.unit,
        (this.unit * 8.5f),
        this.colonnaAppearance
    );
    tg.addChild(fusto);
    return tg;
}
```

Funzione della classe colonna per la creazione dell'echino:

```
// This function generates and returns a Echino object based on the given height.
protected TransformGroup createEchino() {
    TransformGroup tg = new TransformGroup();
    MyCylinder echino = new MyCylinder(
        (this.unit * 1.5f),
        this.unit,
        this.unit,
        this.colonnaAppearance
    );
    // translate position
    Transform3D translate = new Transform3D();
    translate.setTranslation(new Vector3d(0.0f, (this.unit * 4.75f), 0.0f));
    tg.setTransform(translate);
    // add echino to tg and return it
    tg.addChild(echino);
    return tg;
}
```

Funzione della classe Colonna per la creazione dell'abaco:

```
protected TransformGroup createAbaco() {
    TransformGroup tg = new TransformGroup();
    Box abaco = new Box(
        (this.unit * 1.55f),
        (this.unit * 0.5f),
        (this.unit * 1.55f),
        -Primitive.GENERATE_NORMALS,
        this.colonnaAppearance
    );
    // translate position
    Transform3D translate = new Transform3D();
    translate.setTranslation(new Vector3d(0.0f, (this.unit * 5.25f), 0.0f));
    tg.setTransform(translate);
    // add abaco to tg and return it
    tg.addChild(abaco);
    return tg;
}
```

Costruttori della classe MyCylinder:

```
// MyCylinder constructor 1.
public MyCylinder(float radius, float height, Appearance appearance) {
    this(20, radius, radius, height, appearance);
}

// MyCylinder constructor 2.
public MyCylinder(float topRadius, float bottomRadius, float height, Appearance appearance) {
    this(20, topRadius, bottomRadius, height, appearance);
}

// MyCylinder constructor 3.
public MyCylinder(int steps, float topRadius, float bottomRadius, float height, Appearance appearance) {
    float top = height / 2;
    float bottom = -height / 2;
    // create vectors
    v = new Point3f[(steps + 1) * 2];
    for(int i = 0; i < steps; i++) {
        double angle = 2.0 * Math.PI * (double) i / (double) steps;
        float xInf = (float) Math.sin(angle) * bottomRadius;
        float yInf = (float) Math.cos(angle) * bottomRadius;
        float xSup = (float) Math.sin(angle) * topRadius;
        float ySup = (float) Math.cos(angle) * topRadius;
        v[i*2+0] = new Point3f(xInf, bottom, yInf);
        v[i*2+1] = new Point3f(xSup, top, ySup);
    }
    v[steps*2+0] = new Point3f(0.0f, bottom, bottomRadius);
    v[steps*2+1] = new Point3f(0.0f, top, topRadius);
    // create triangle strip
    int [] stripCounts ={(steps + 1) * 2};
    triangleStrip = new TriangleStripArray(
        (steps + 1) * 2,
        GeometryArray.COORDINATES,
        stripCounts
    );
    triangleStrip.setCoordinates(0, v);
    // create geometry
    GeometryInfo gInfo = new GeometryInfo(triangleStrip);
    NormalGenerator normgen = new NormalGenerator();
    normgen.generateNormals(gInfo);
    // set geometry
    setGeometry(triangleStrip);
    setGeometry(gInfo.getGeometryArray());
    // set appearance
    setAppearance(appearance);
}
```

Motivazioni

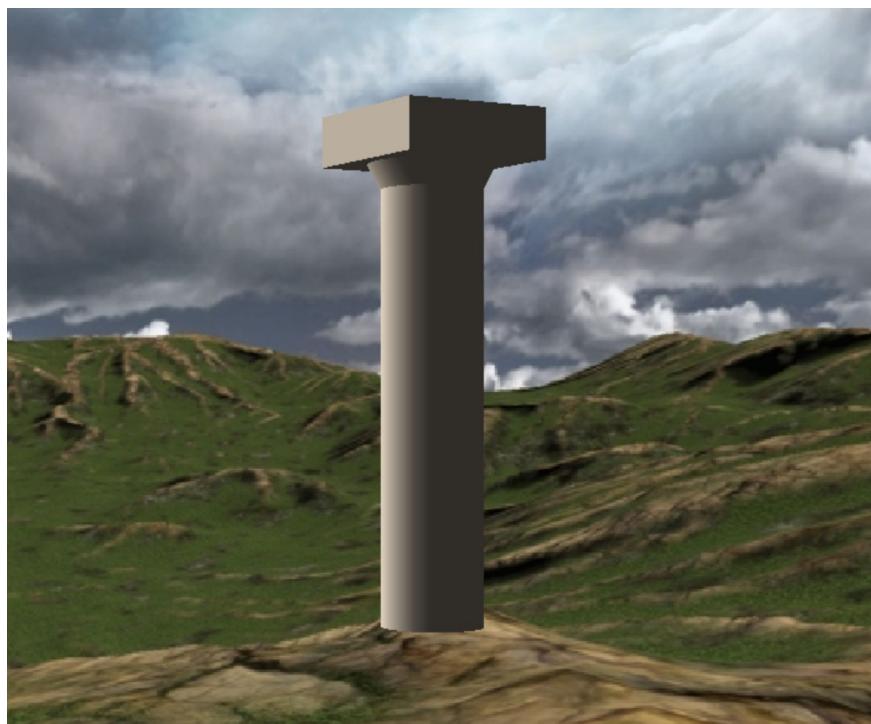
La classe Colonna estende Group in modo tale da raggruppare tra loro le varie parti che vanno a comporre la colonna.

Ogni parte della colonna è composta da un oggetto TransformGroup al quale è in alcuni casi applicata una trasformazione.

Per il fusto e l'abaco sono stati istanziati due oggetti della classe MyCylinder, la quale permette di definire un cilindro specificando valori diversi per la base maggiore e la base minore. Per quanto riguarda l'echino è bastato istanziare un oggetto della primitiva Box.

Le leaf che vanno a comporre il fusto, l'abaco e l'echino condividono la stessa appearance, la quale viene generata dal costruttore nella classe Colonna.

Risultato:



Conclusione

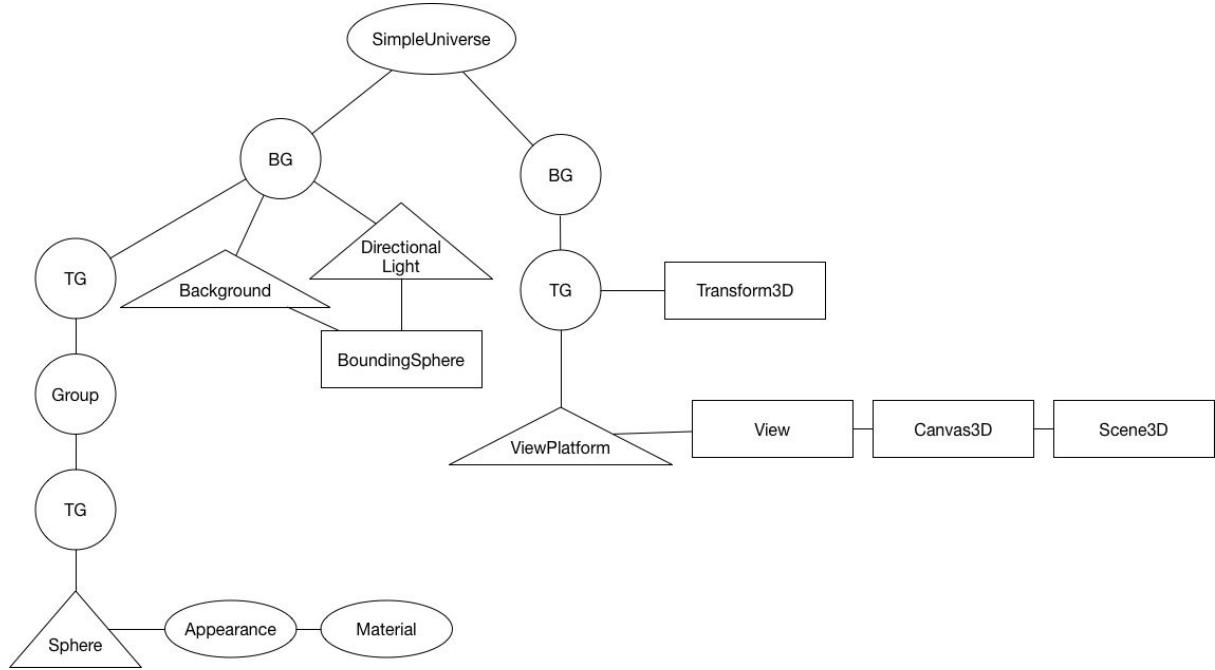
Attraverso l'estensione della classe Group è stato possibile raggruppare diversi oggetti TransformGroup all'interno di un unico oggetto.

Attraverso l'oggetto della classe Material assegnato all'appearance è stato possibile specificare i colori che deve assumere la colonna in modo da risultare coerente con la pietra.

Esercizio 4.1

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

Scenegraph



Codice

Funzione per la creazione del BranchGroup:

```
// This function creates and return a new branchgroup.
private BranchGroup createBranchGroup() {
    // initialize bg
    BranchGroup bg = new BranchGroup();
    // create main tg
    TransformGroup tg = new TransformGroup();
    tg.addChild(new Planet(1.0f, 0.0f, "../../../../images/earth.jpg", this.defaultBound));
    // add tg to bg
    bg.addChild(tg);
    // return bg
    return bg;
}
```

Funzioni per l'applicazione della luce:

```
// This function add a directional light to the world with the default direction.  
private void addDirectionalLight(BranchGroup branchGroup) {  
    addDirectionalLight(branchGroup, 1.0f, 1.0f, 1.0f);  
}  
  
// This function add a directional light to the world with a custom direction.  
private void addDirectionalLight(BranchGroup branchGroup, float dirX, float dirY, float dirZ) {  
    // create light  
    DirectionalLight light = new DirectionalLight();  
    light.setDirection(dirX, dirY, dirZ);  
    // add bounds to light  
    light.setInfluencingBounds(this.defaultBound);  
    // add light to tg  
    branchGroup.addChild(light);  
}
```

Funzione per l'applicazione del background:

```
private void addBackground(BranchGroup branchGroup, String imagepath) {  
    // load the texture  
    TextureLoader loader = new TextureLoader(imagepath, null);  
    // create image object  
    ImageComponent2D image = loader.getImage();  
    // create background  
    Background background = new Background();  
    // set image to background  
    background.setImage(image);  
    background.setImageScaleMode(Background.SCALE_FIT_MAX);  
    // set bound to background  
    background.setApplicationBounds(this.defaultBound);  
    // add background to branchgroup  
    branchGroup.addChild(background);  
}
```

Costruttore della classe Planet:

```
public Planet(float radius, float rotation, String texture, BoundingSphere bound) {  
    this.radius = radius;  
    this.rotation = rotation;  
    this.texture = texture;  
    this.bound = bound;  
    // create appearance  
    this.appearance = createAppearance();  
    // create sphere  
    addChild(createSphere());  
}
```

Funzione della classe Planet usata per la creazione dell'appearance:

```
// This function creates a new appearance for the planet.
protected Appearance createAppearance() {
    // initialize the appearance
    Appearance app = new Appearance();
    // create material
    Material material = new Material();
    material.setShininess(80.0f);
    material.setSpecularColor(new Color3f(0.0f, 0.0f, 0.0f));
    app.setMaterial(material);
    // add texture
    if (this.texture != "") {
        // load texture file
        TextureLoader textureLoader = new TextureLoader(this.texture, null);
        // initialize texture object
        Texture texture = textureLoader.getTexture();
        // add texture to the appearance
        app.setTexture(texture);
        // initialize texture attributes
        TextureAttributes textureAttributes = new TextureAttributes();
        textureAttributes.setTextureMode(TextureAttributes.COMBINE);
        app.setTextureAttributes(textureAttributes);
    }
    // return the appearance
    return app;
}
```

Funzione della classe Planet usata per la creazione della sfera:

```
// This function creates a new sphere used as planet.
protected TransformGroup createSphere() {
    TransformGroup tg = new TransformGroup();
    Sphere sphere = new Sphere(
        this.radius,
        Primitive.GENERATE_NORMALS|Primitive.GENERATE_TEXTURE_COORDS,
        100,
        this.appearance
    );
    // add sphere to tg
    tg.addChild(sphere);
    // add rotation to tg
    if (this.rotation > 0) {
        addRotation(tg);
    }
    // return tg
    return tg;
}
```

Motivazioni

Per impostare il background del branchgroup è stata implementata nel Main la funzione addBackground(). Essa utilizza un oggetto TextureLoader per caricare l'immagine da usare come sfondo e assegnarla poi ad un nuovo oggetto ImageComponent2D. L'oggetto risultante viene poi assegnato come immagine ad un oggetto Background che poi viene

aggiunto ai figli del branchgroup. L'oggetto BoundingSphere assegnato al background è stato definito come variabile d'istanza per poter essere poi condiviso anche con la luce che crea l'effetto "sole".

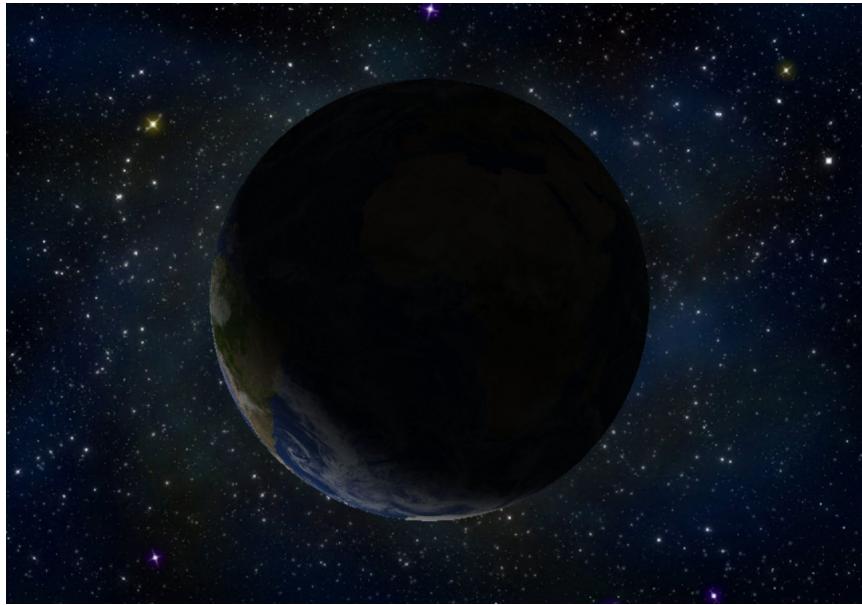
Per la creazione della luce è stata utilizzata la funzione addDirectionalLight() la quale aggiunge al branchgroup passato come parametro un oggetto di tipo DirectionalLight.

Per la creazione del pianeta è stata definita la classe Planet (estensione di Group), la quale contiene un oggetto TransformGroup con un oggetto Sphere come leaf.

La funzione createAppearance() di Planet ritorna un nuovo oggetto Appearance al quale viene impostato un materiale con un colore di default e, se passata nel costruttore la path di una texture, imposta anche la texture richiesta.

Risultato





Conclusione

Attraverso l'utilizzo della classe TextureLoader è stato possibile utilizzare immagini prelevate dal file system come texture.

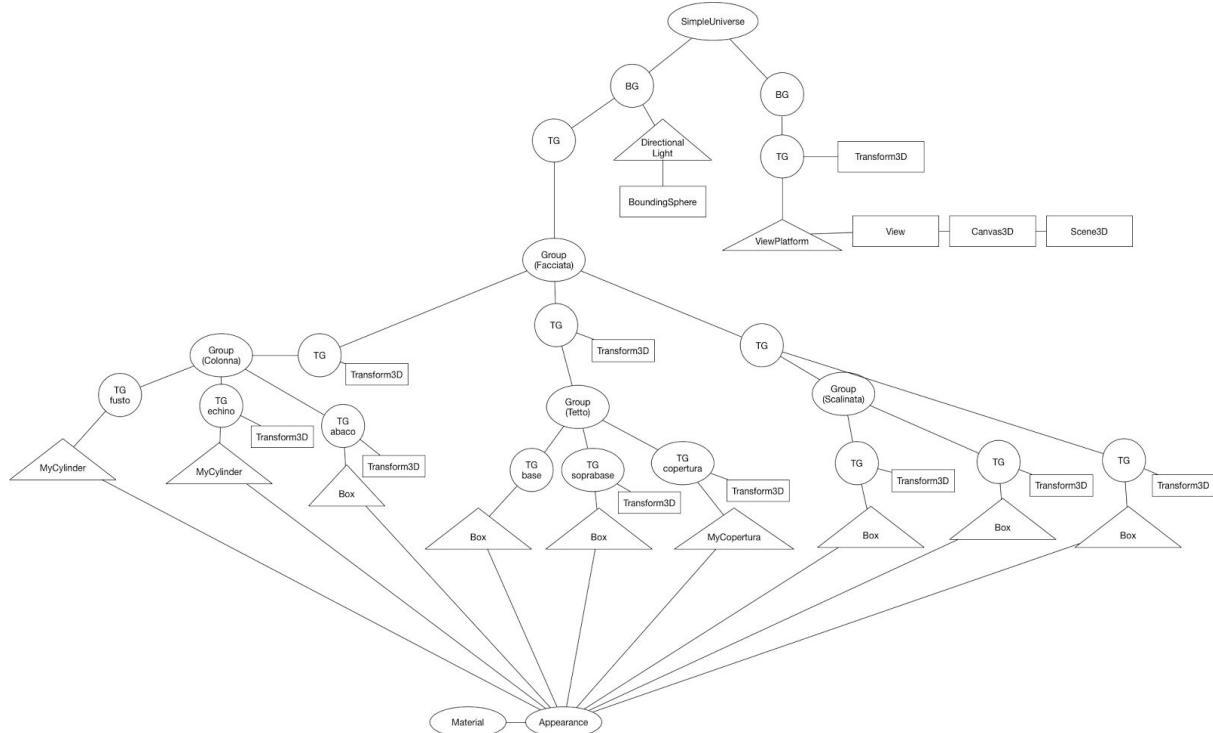
L'oggetto Texture ha permesso di impostare una texture nell'appereance di un oggetto.

L'oggetto Background ha permesso di definire lo sfondo dell'universo, il quale è stato facilmente applicato al branchgroup come una qualsiasi altra leaf tramite la funzione addChild().

Esercizio 4.2

- Applicare una texture agli elementi della colonna dorica creata Precedentemente.
- Riprodurre la facciata del tempio di Poseidone di Pæstum.

Scenegraph



NOTE: Per semplicità è stato rappresentato un solo Group Colonna anche se nell'universo ne sono state utilizzati sei.

Codice

Funzione utilizzata per la generazione del branchgroup:

```
// This function creates and return a new branchgroup.
private BranchGroup createBranchGroup() {
    // initialize bg
    BranchGroup bg = new BranchGroup();
    // create main tg
    TransformGroup tg = new TransformGroup();
    tg.addChild(new Facciata()); // <-- NOTE: edit here w
    // add tg to bg
    bg.addChild(tg);
    // return bg
    return bg;
}
```

Costruttore della classe Facciata:

```
public Facciata() {
    // create shared appearance
    this.facciataAppearance = createAppearance();
    // create scalinata
    addChild(createScalinata());
    // create tetto
    addChild(createTetto());
    // create colonne
    for (int i = 0; i < 6; i++) {
        addChild(createColonna(i));
    }
}
```

Funzione della classe Facciata usata per la generazione dell'appearance:

```
protected Appearance createAppearance() {
    Appearance appearance = new Appearance();
    // add material
    Material material = new Material();
    material.setAmbientColor(239/255f,224/255f,203/255f);
    material.setDiffuseColor(239/255f,224/255f,203/255f);
    appearance.setMaterial(material);
    // load texture file
    TextureLoader textureLoader = new TextureLoader("../images/pietra.jpg", null);
    // initialize texture object
    Texture texture = textureLoader.getTexture();
    texture.setBoundaryModeS(Texture.WRAP);
    texture.setBoundaryModeT(Texture.WRAP);
    // add texture to the appearance
    appearance.setTexture(texture);
    // initialize texture attributes
    TextureAttributes textureAttributes = new TextureAttributes();
    textureAttributes.setTextureMode(TextureAttributes.COMBINE);
    textureAttributes.setPerspectiveCorrectionMode(TextureAttributes.NICEST);
    appearance.setTextureAttributes(textureAttributes);
    // initialize and add text coordinates generator
    TexCoordGeneration tcg = new TexCoordGeneration(
        TexCoordGeneration.OBJECT_LINEAR,
        TexCoordGeneration.TEXTURE_COORDINATE_3
    );
    appearance.setTexCoordGeneration(tcg);
    // add style
    appearance.setPolygonAttributes(new PolygonAttributes(
        PolygonAttributes.POLYGON_FILL,
        PolygonAttributes.CULL_NONE,
        0
    ));
    // return the appearance
    return appearance;
}
```

Funzioni della classe Facciata per la creazione delle singole parti della facciata:

```
protected TransformGroup createScalinata() {
    TransformGroup tg = new TransformGroup();
    Scalinata scalinata = new Scalinata(2.75f, 1.0f, 3, this.facciataAppearance);
    this.scalinataHeight = scalinata.getHeight();
    tg.addChild(scalinata);
    return tg;
}
```

```
protected TransformGroup createTetto() {
    TransformGroup tg = new TransformGroup();
    Tetto tetto = new Tetto(3.0f, 1.0f, this.facciataAppearance);
    this.tettoHeight = tetto.getHeight();
    tg.addChild(tetto);
    // add correct transformation to tg
    Transform3D translate = new Transform3D();
    translate.setTranslation(new Vector3d(
        0.0f,
        (this.scalinataHeight + this.tettoHeight + 2),
        0.0f
    ));
    tg.setTransform(translate);
    return tg;
}
```

```
protected TransformGroup createColonna(int numColonna) {
    TransformGroup tg = new TransformGroup();
    Colonna colonna = new Colonna(2.0f, this.facciataAppearance);
    tg.addChild(colonna);
    // add correct transformation to tg
    Transform3D translate = new Transform3D();
    translate.setTranslation(new Vector3d(
        ((numColonna > 2) ? -(2.5 - numColonna) : (numColonna - 2.5)),
        (this.scalinataHeight + 1),
        0.0f
    ));
    tg.setTransform(translate);
    return tg;
}
```

Costruttore della classe Scalinata:

```
public Scalinata(float x, float z, int numScale, Appearance appearance) {  
    // set initial values  
    if (numScale < 1) {  
        throw new RuntimeException("numScale should be one or more");  
    }  
    this.numScale = numScale;  
    this.baseWidth = x;  
    this.baseLength = z;  
    this.size = calculateSize();  
    this.scalaAppearance = (appearance == null) ? createAppearance() : appearance;  
    // create scale  
    for (int i = 0; i < this.numScale; i++) {  
        addChild(createScala(i));  
    }  
}
```

Funzione della classe Scalinata utilizzata per la creazione di un singolo scalino:

```
protected TransformGroup createScala(int numScala) {  
    TransformGroup tg = new TransformGroup();  
    Box scala = new Box(  
        calculateWidth(numScala),  
        this.size,  
        calculateLength(numScala),  
        -Primitive.GENERATE_NORMALS|Primitive.GENERATE_TEXTURE_COORDS,  
        this.scalaAppearance  
    );  
    tg.addChild(scala);  
    // add correct transformation to tg  
    Transform3D translate = new Transform3D();  
    translate.setTranslation(new Vector3d(  
        0.0f,  
        ((numScala) * this.size * 2),  
        0.0f  
    ));  
    tg.setTransform(translate);  
    return tg;  
}
```

Costruttore della classe Tetto:

```
public Tetto(float x, float z, Appearance appearance) {  
    // set initial values  
    this.baseWidth = x;  
    this.baseLength = z;  
    this.size = calculateSize();  
    this.tettoAppearance = (appearance == null) ? createAppearance() : appearance;  
    // create components  
    addChild(createBase());  
    addChild(createSopraBase());  
    addChild(createCopertura());  
}
```

Funzioni della classe Tetto per la creazione delle sue componenti:

```
protected TransformGroup createBase() {
    TransformGroup tg = new TransformGroup();
    Box base = new Box(
        (this.baseWidth),
        this.size,
        (this.baseLength),
        -Primitive.GENERATE_NORMALS|Primitive.GENERATE_TEXTURE_COORDS,
        this.tettoAppearance
    );
    tg.addChild(base);
    return tg;
}
```

```
protected TransformGroup createSopraBase() {
    TransformGroup tg = new TransformGroup();
    Box base = new Box(
        (this.baseWidth - (this.size / 2)),
        (this.size / 2),
        (this.baseLength - (this.size / 2)),
        -Primitive.GENERATE_NORMALS|Primitive.GENERATE_TEXTURE_COORDS,
        this.tettoAppearance
    );
    tg.addChild(base);
    // add correct transformation to tg
    Transform3D translate = new Transform3D();
    translate.setTranslation(new Vector3d(
        0.0f,
        this.size + (this.size / 2),
        0.0f
    ));
    tg.setTransform(translate);
    return tg;
}
```

```
protected TransformGroup createCopertura() {
    TransformGroup tg = new TransformGroup();
    MyCopertura copertura = new MyCopertura(
        this.baseWidth,
        (this.size * 2),
        this.baseLength,
        this.tettoAppearance
    );
    // add correct transformation to tg
    Transform3D translate = new Transform3D();
    translate.setTranslation(new Vector3d(
        0.0f,
        (this.size * 1.5) + (this.size / 2),
        0.0f
    ));
    tg.setTransform(translate);
    tg.addChild(copertura);
    return tg;
}
```

Costruttore della classe MyCopertura:

```
// MyCopertura.
public MyCopertura(float width, float height, float length, Appearance appearance) {

    p1 = new Point3f(
        0.0f,
        (height * 2),
        (length)
    );
    p2 = new Point3f(
        0.0f,
        (height * 2),
        -(length)
    );
    p3 = new Point3f(
        (width),
        0.0f,
        (length)
    );
    p4 = new Point3f(
        (width),
        0.0f,
        -(length)
    );
    p5 = new Point3f(
        -(width),
        0.0f,
        (length)
    );
    p6 = new Point3f(
        -(width),
        0.0f,
        -(length)
    );

    v[0] = p6; v[1] = p2; v[2] = p5;
    v[3] = p1; v[4] = p3; v[5] = p2;
    v[6] = p4; v[7] = p6; v[8] = p3;
    v[9] = p5;

    int [] stripCounts = [22];
    triangleStrip = new TriangleStripArray(
        22,
        GeometryArray.COORDINATES,
        stripCounts
    );
    triangleStrip.setCoordinates(0, v);

    GeometryInfo gInfo = new GeometryInfo(triangleStrip);
    NormalGenerator normgen = new NormalGenerator();
    normgen.generateNormals(gInfo);

    setGeometry(triangleStrip);
    setGeometry(gInfo.getGeometryArray());

    setAppearance(appearance);
}
```

Codice modificato rispetto all'esercizio 3.8 della classe Colonna (Aggiunta di GENERATE_TEXTURE_COORDS nel parametro infflags del costruttore Box):

```
protected TransformGroup createAbaco() {
    TransformGroup tg = new TransformGroup();
    Box abaco = new Box(
        (this.unit * 1.55f),
        (this.unit * 0.5f),
        (this.unit * 1.55f),
        -Primitive.GENERATE_NORMALS|Primitive.GENERATE_TEXTURE_COORDS,
        this.colonnaAppearance
    );
    // translate position
    Transform3D translate = new Transform3D();
    translate.setTranslation(new Vector3d(0.0f, (this.unit * 5.25f), 0.0f));
    tg.setTransform(translate);
    // add abaco to tg and return it
    tg.addChild(abaco);
    return tg;
}
```

Motivazioni

La creazione dell'appearance è stata gestita dalla classe Facciata per essere poi condivisa da tutti gli altri oggetti che vanno a comporla (evitando così di replicarla inutilmente).

Per la composizione della Facciata sono state usate tre classi:

- Scalinata: compone una scalinata posizionando uno sopra l'altro degli oggetti Box.
- Colonna: compone una singola colonna composta da fusto, abaco e echino (si veda esercizio 3.8).
- Tetto: compone il tetto costruendo una base, una soprabase di dimensione inferiore ed una copertura basata sulla classe MyCopertura (estensione di Shape3D).

Risultato





Conclusione

Attraverso la classe Facciata è possibile replicare istanze della facciata del tempio.

La classe Colonna dell'esercizio 3.8 è stata facilmente riutilizzata come parte che compone la facciata. Le altre classi sviluppate per rappresentare la facciata sono facilmente riutilizzabili anche per sviluppare altri oggetti complessi.

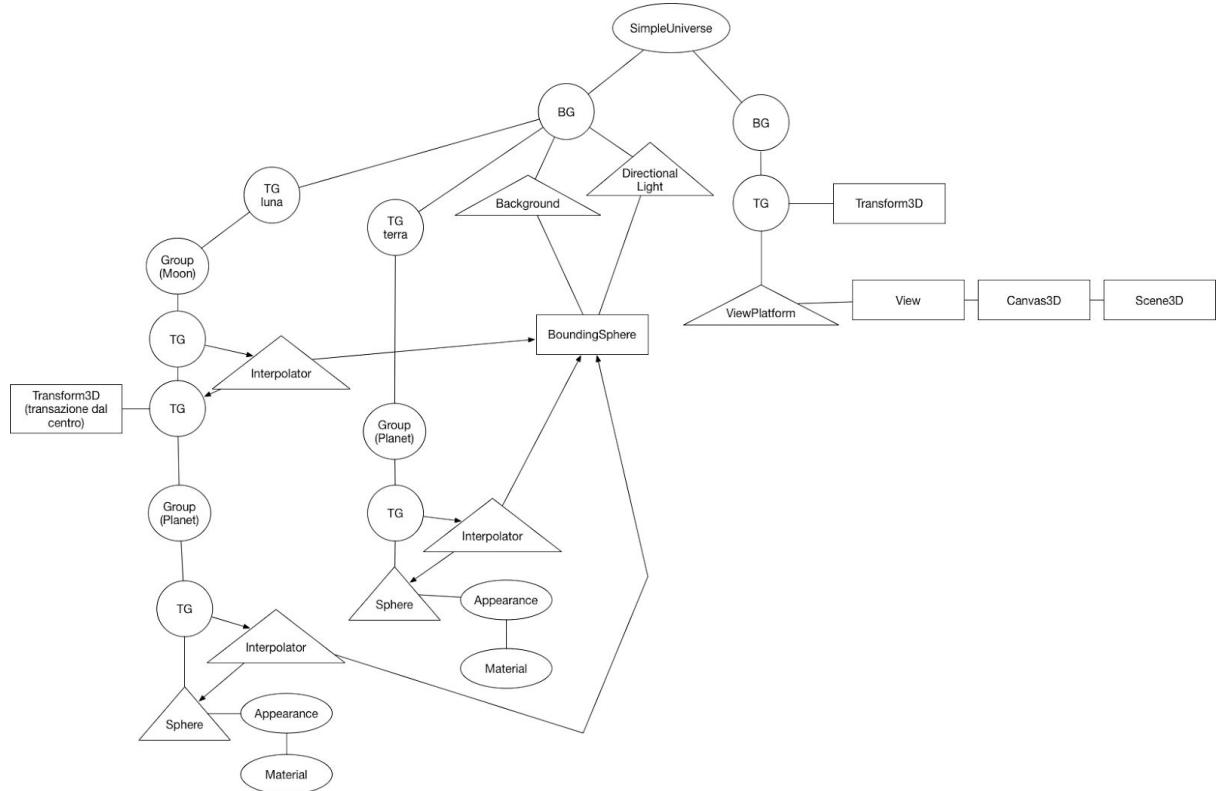
Esercizio 4.3

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

Funzione per la creazione del branchgroup:

```
// This function creates and return a new branchgroup.
private BranchGroup createBranchGroup() {
    // initialize bg
    BranchGroup bg = new BranchGroup();
    // create main tg
    TransformGroup tg = new TransformGroup();
    tg.addChild(new Planet(0.5f, 4.0f, "../../images/earth.jpg", this.defaultBound));
    tg.addChild(new Moon(0.1f, 1.0f, 4.0f, 4.0f, this.defaultBound));
    // add tg to bg
    bg.addChild(tg);
    // return bg
    return bg;
}
```

Costruttore della classe Moon:

```
public Moon(float radius, float translation, float selfRotation, float orbitRotation, BoundingSphere bound) {
    this.radius = radius;
    this.translation = translation;
    this.selfRotation = selfRotation;
    this.orbitRotation = orbitRotation;
    this.bound = bound;
    // create moon rotation
    addChild(createMoonRotation());
}
```

Funzione della classe Moon per la creazione di un TransformGroup il quale il quale applica la rotazione intorno alla terra alla luna:

```
protected TransformGroup createMoonRotation() {
    TransformGroup tg;
    if (this.orbitRotation > 0) {
        tg = new TransformGroup();
        // create transformation for the sphere rotation
        Transform3D sphereRotation = new Transform3D();
        sphereRotation.rotY(- Math.PI / 4.0f);
        // create alpha
        Alpha alpha = new Alpha(-1, 50000);
        // create rotation interpolator
        RotationInterpolator sphereRotationInterpolator = new RotationInterpolator(
            alpha,
            tg,
            sphereRotation,
            0.0f,
            (float) Math.PI * this.orbitRotation
        );
        // create bounding sphere and set to interpolator
        sphereRotationInterpolator.setSchedulingBounds(this.bound);
        // permit rotation after the creation
        tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        // add sphere as tg child
        tg.addChild(sphereRotationInterpolator);
        tg.addChild(createMoon());
    } else {
        tg = createMoon();
    }
    return tg;
}
```

Funzione della classe Moon per la creazione della luna attraverso una istanza di Planet:

```
protected TransformGroup createMoon() {
    TransformGroup tg = new TransformGroup();
    tg.addChild(new Planet(this.radius, this.selfRotation, "../../images/moon.jpg", this.bound));
    if (this.translation > 0) {
        Transform3D translation = new Transform3D();
        translation.setTranslation(new Vector3d(this.translation, 0.0, 0.0));
        tg.setTransform(translation);
    }
    return tg;
}
```

Costruttore della classe Planet:

```
public Planet(float radius, float rotation, String texture, BoundingSphere bound) {
    this.radius = radius;
    this.rotation = rotation;
    this.texture = texture;
    this.bound = bound;
    // create appearance
    this.appearance = createAppearance();
    // create sphere
    addChild(createSphere());
}
```

Funzione della classe Planet per la creazione della sfera:

```
// This function creates a new sphere used as planet.
protected TransformGroup createSphere() {
    TransformGroup tg = new TransformGroup();
    Sphere sphere = new Sphere(
        this.radius,
        Primitive.GENERATE_NORMALS|Primitive.GENERATE_TEXTURE_COORDS,
        100,
        this.appearance
    );
    // add sphere to tg
    tg.addChild(sphere);
    // add rotation to tg
    if (this.rotation > 0) {
        addRotation(tg);
    }
    // return tg
    return tg;
}
```

Funzione della classe Planet per applicare una rotazione al pianeta:

```
protected void addRotation(TransformGroup tg) {
    // create transformation for the sphere rotation
    Transform3D sphereRotation = new Transform3D();
    sphereRotation.rotY(- Math.PI / 4.0f);
    // create alpha
    Alpha alpha = new Alpha(-1, 50000);
    // create rotation interpolator
    RotationInterpolator sphereRotationInterpolator = new RotationInterpolator(
        alpha,
        tg,
        sphereRotation,
        0.0f,
        (float) Math.PI * this.rotation
    );
    // create bounding sphere and set to interpolator
    sphereRotationInterpolator.setSchedulingBounds(this.bound);
    // permit rotation after the creation
    tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    // add sphere as tg child
    tg.addChild(sphereRotationInterpolator);
}
```

Funzione della classe Planet per la creazione dell'appearance del pianeta:

```
// This function creates a new appearance for the planet.
protected Appearance createAppearance() {
    // initialize the appearance
    Appearance app = new Appearance();
    // create material
    Material material = new Material();
    material.setShininess(80.0f);
    material.setSpecularColor(new Color3f(0.0f, 0.0f, 0.0f));
    app.setMaterial(material);
    // add texture
    if (this.texture != "") {
        // load texture file
        TextureLoader textureLoader = new TextureLoader(this.texture, null);
        // initialize texture object
        Texture texture = textureLoader.getTexture();
        // add texture to the appearance
        app.setTexture(texture);
        // initialize texture attributes
        TextureAttributes textureAttributes = new TextureAttributes();
        textureAttributes.setTextureMode(TextureAttributes.COMBINE);
        app.setTextureAttributes(textureAttributes);
    }
    // return the appearance
    return app;
}
```

Funzione utilizzata per applicare la luce solare al branchgroup:

```
// This function add a directional light to the world with the default direction.  
private void addDirectionalLight(BranchGroup branchGroup) {  
    addDirectionalLight(branchGroup, 1.0f, 0.0f, 1.0f);  
}  
  
// This function add a directional light to the world with a custom direction.  
private void addDirectionalLight(BranchGroup branchGroup, float dirX, float dirY, float dirZ) {  
    // create light  
    DirectionalLight light = new DirectionalLight();  
    light.setDirection(dirX, dirY, dirZ);  
    // add bounds to light  
    light.setInfluencingBounds(this.defaultBound);  
    // add light to tg  
    branchGroup.addChild(light);  
}
```

Funzione utilizzata per applicare l'immagine di background al branchgroup:

```
private void addBackground(BranchGroup branchGroup, String imagepath) {  
    // load the texture  
    TextureLoader loader = new TextureLoader(imagepath, null);  
    // create image object  
    ImageComponent2D image = loader.getImage();  
    // create background  
    Background background = new Background();  
    // set image to background  
    background.setImage(image);  
    background.setImageScaleMode(Background.SCALE_FIT_MAX);  
    // set bound to background  
    background.setApplicationBounds(this.defaultBound);  
    // add background to branchgroup  
    branchGroup.addChild(background);  
}
```

Motivazioni

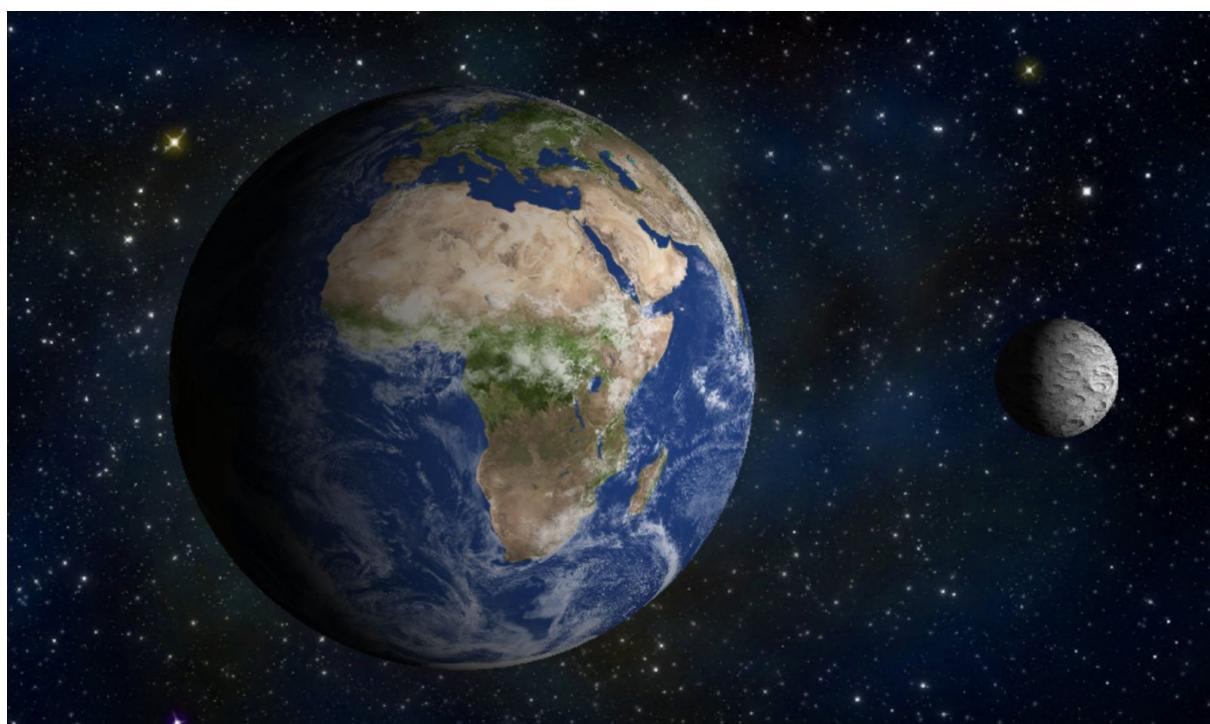
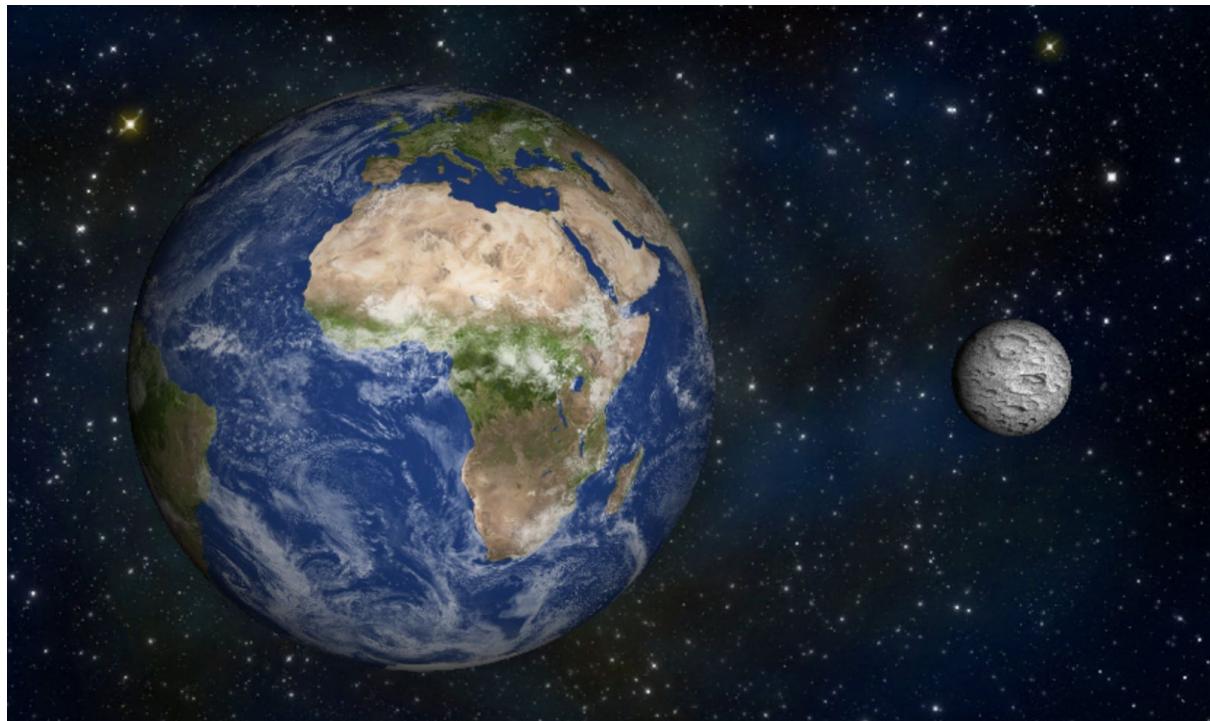
Per la realizzazione di entrambi i pianeti è stata utilizzata la classe Planet.

La classe Planet è stata sviluppata in modo tale da permettere di impostare in fase di inizializzazione quale texture usare (e se usarla) e che tipo di rotazione intorno a al proprio asse eseguire (o se non eseguirla); questo ha permesso di utilizzare tale classe sia per la rappresentazione della terra sia per quella della luna.

Dovendo applicare una seconda rotazione alla luna (per seguire l'orbita intorno alla terra), è stata sviluppata una classe Moon la quale implementa un oggetto Planet specifico per la luna, lo assegna ad un oggetto TransformGroup traslato dal centro ed applica una rotazione ad esso intorno al centro.

Per applicare le rotazioni è stata utilizzata la classe RotationInterpolator.

Risultato



Conclusione

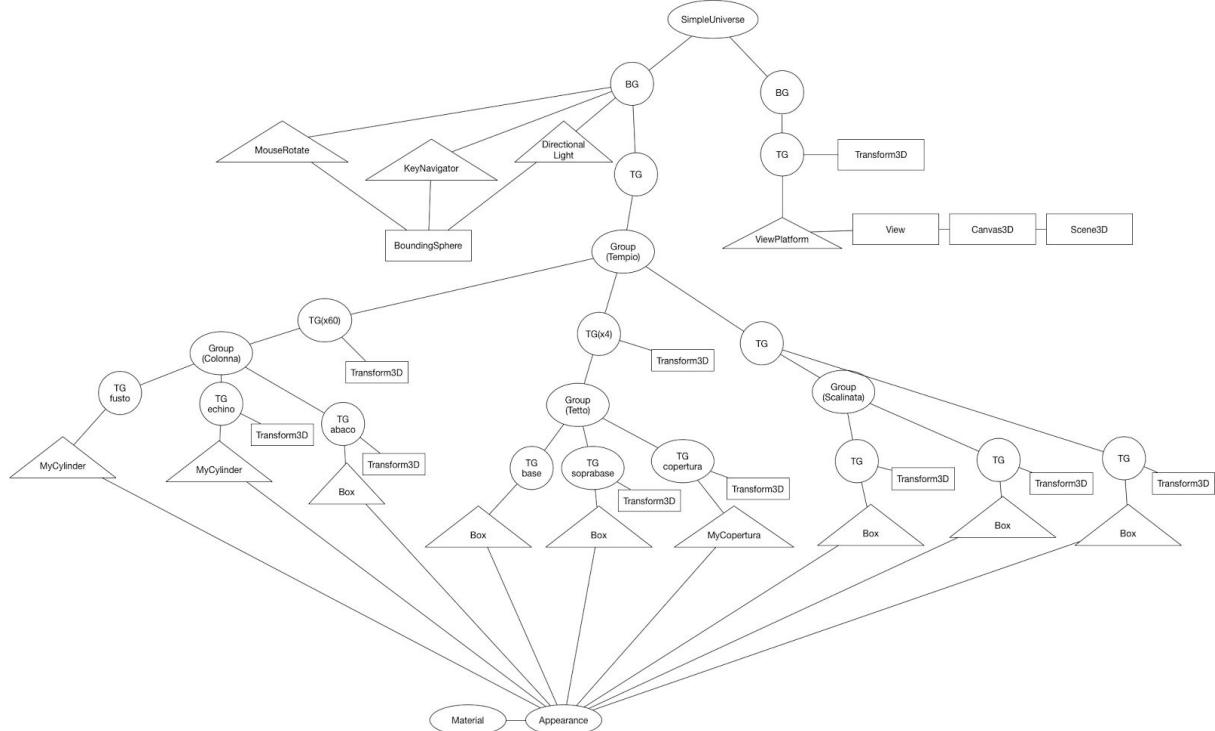
Grazie all'utilizzo della classe Planet è stato facile costruire entrambi i pianeti ed applicare ad essi una o più rotazioni.

Queste rotazioni hanno permesso di far ruotare i due pianeti intorno al proprio asse e di far eseguire alla luna un'orbita intorno alla terra.

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



NOTE: I nodi TG con tra parentesi (xN) indicano il numero di volte che tale sottografo viene ripetuto.

Codice

Funzione per la creazione del branchgroup:

```
// This function creates and return a new branchgroup.
private BranchGroup createBranchGroup() {
    // initialize bg
    BranchGroup bg = new BranchGroup();
    // create main tg
    TransformGroup tg = new TransformGroup();
    tg.addChild(new Tempio()); // <-- NOTE: edit here with your code
    // rotate tg with mouse
    addMouseMovementsToTransformGroup(tg, bg);
    // add tg to bg
    bg.addChild(tg);
    // return bg
    return bg;
}
```

Costruttore della classe Tempio:

```
public Tempio() {
    // create shared appearance
    this.tempioAppearance = createAppearance();
    // create scalinata
    addChild(createScalinata());
    // create colonne
    addColonneChild();
    // create tetto
    addTettoChild();
    // create colonne internal
    addColonneInternalChild();
}
```

Funzioni della classe Tempio per la creazione delle colonne esterne:

```
protected void addColonneChild() {
    // front
    for (int i = 0; i < 6; i++) {
        addChild(createColonna(i, "front"));
    }
    // back
    for (int i = 0; i < 6; i++) {
        addChild(createColonna(i, "back"));
    }
    // left
    for (int i = 0; i < 12; i++) {
        addChild(createColonna(i, "left"));
    }
    // right
    for (int i = 0; i < 12; i++) {
        addChild(createColonna(i, "right"));
    }
}

protected TransformGroup createColonna(int numColonna, String type) {
    TransformGroup tg = new TransformGroup();
    Colonna colonna = new Colonna(2.0f, this.tempioAppearance);
    tg.addChild(colonna);
    // add correct transformation to tg
    Transform3D translate = new Transform3D();
    if (type == "front" || type == "back") {
        translate.setTranslation(new Vector3d(
            ((numColonna > 2) ? -(2.5 - numColonna) : (numColonna - 2.5)),
            (this.scalinataHeight + 1),
            (type == "front" ? 6.5f : -6.5f)
        ));
    } else {
        translate.setTranslation(new Vector3d(
            (type == "left" ? 2.5f : -2.5f),
            (this.scalinataHeight + 1),
            ((numColonna > 5) ? -(5.5 - numColonna) : (numColonna - 5.5))
        ));
    }
    tg.setTransform(translate);
    return tg;
}
```

Funzioni della classe Tempio per la creazione dei tetti (uno per lato):

```
protected void addTettoChild() {
    // front
    addChild(createTetto("front"));
    // back
    addChild(createTetto("back"));
    // left
    addChild(createTetto("left"));
    // right
    addChild(createTetto("right"));
}

protected TransformGroup createTetto(String type) {
    boolean showCopertura = (type == "front" || type == "back");
    float width = (showCopertura ? 3.0f : 0.5f);
    float length = (showCopertura ? 0.5f : 6.0f);
    TransformGroup tg = new TransformGroup();
    Tetto tetto = new Tetto(width, length, showCopertura, this.temPIOAppearance);
    this.tettoHeight = tetto.getHeight();
    tg.addChild(tetto);
    // add correct transformation to tg
    Transform3D translate = new Transform3D();
    if (type == "front" || type == "back") {
        translate.setTranslation(new Vector3d(
            0.0f,
            (this.scalinataHeight + this.tettoHeight + 2),
            (type == "front" ? 6.5f : -6.5f)
        ));
    } else {
        translate.setTranslation(new Vector3d(
            (type == "left" ? 2.5f : -2.5f),
            (this.scalinataHeight + this.tettoHeight + 2),
            0.0f
        ));
    }
    tg.setTransform(translate);
    return tg;
}
```

Funzioni della classe Tempio per la creazione delle colonne interne:

```
protected void addColonneInternalChild() {
    // front
    for (int i = 0; i < 4; i++) {
        addChild(createColonnaInternal(i, "front"));
    }
    // back
    for (int i = 0; i < 4; i++) {
        addChild(createColonnaInternal(i, "back"));
    }
    // left
    for (int i = 0; i < 8; i++) {
        addChild(createColonnaInternal(i, "left"));
    }
    // right
    for (int i = 0; i < 8; i++) {
        addChild(createColonnaInternal(i, "right"));
    }
}

protected TransformGroup createColonnaInternal(int numColonna, String type) {
    TransformGroup tg = new TransformGroup();
    float height = ((type == "front" || type == "back") ? 2.0f : 1.7f);
    Colonna colonna = new Colonna(height, this.tempioAppearance);
    tg.addChild(colonna);
    // add correct transformation to tg
    Transform3D translate = new Transform3D();
    if (type == "front" || type == "back") {
        translate.setTranslation(new Vector3d(
            ((numColonna > 2) ? -(1.5 - numColonna) : (numColonna - 1.5)),
            (this.scalinataHeight + 1),
            (type == "front" ? 4f : -4f)
        ));
    } else {
        translate.setTranslation(new Vector3d(
            (type == "left" ? 1.5f : -1.5f),
            (this.scalinataHeight + 0.85f),
            ((numColonna > 3) ? -(1.75 - (numColonna / 1.95)) : ((numColonna / 1.95) - 1.75))
        ));
    }
    tg.setTransform(translate);
    return tg;
}
```

Funzione della classe Tempio per la creazione della scalinata:

```
protected TransformGroup createScalinata() {
    TransformGroup tg = new TransformGroup();
    Scalinata scalinata = new Scalinata(3.0f, 7.0f, 3, this.tempioAppearance);
    this.scalinataHeight = scalinata.getHeight();
    tg.addChild(scalinata);
    return tg;
}
```

Funzione della classe Tempio per la creazione dell'appearance condivisa:

```
protected Appearance createAppearance() {
    Appearance appearance = new Appearance();
    // add material
    Material material = new Material();
    material.setAmbientColor(239/255f,224/255f,203/255f);
    material.setDiffuseColor(239/255f,224/255f,203/255f);
    appearance.setMaterial(material);
    // load texture file
    TextureLoader textureLoader = new TextureLoader("../images/pietra.jpg", null);
    // initialize texture object
    Texture texture = textureLoader.getTexture();
    texture.setBoundaryModeS(Texture.WRAP);
    texture.setBoundaryModeT(Texture.WRAP);
    // add texture to the appearance
    appearance.setTexture(texture);
    // initialize texture attributes
    TextureAttributes textureAttributes = new TextureAttributes();
    textureAttributes.setTextureMode(TextureAttributes.COMBINE);
    textureAttributes.setPerspectiveCorrectionMode(TextureAttributes.NICEST);
    appearance.setTextureAttributes(textureAttributes);
    // initialize and add text coordinates generator
    TexCoordGeneration tcg = new TexCoordGeneration(
        TexCoordGeneration.OBJECT_LINEAR,
        TexCoordGeneration.TEXTURE_COORDINATE_3
    );
    appearance.setTexCoordGeneration(tcg);
    // add style
    appearance.setPolygonAttributes(new PolygonAttributes(
        PolygonAttributes.POLYGON_FILL,
        PolygonAttributes.CULL_NONE,
        0
    ));
    // return the appearance
    return appearance;
}
```

Costruttore della classe Tetto (modificato dall'esercizio 4.2 per rendere dinamica la visualizzazione della copertura):

```
public Tetto(float x, float z, boolean showCopertura, Appearance appearance) {
    // set initial values
    this.baseWidth = x;
    this.baseLength = z;
    this.size = calculateSize(showCopertura);
    this.tettoAppearance = (appearance == null) ? createAppearance() : appearance;
    // create components
    addChild(createBase());
    addChild(createSopraBase(showCopertura));
    if (showCopertura) {
        addChild(createCopertura());
    }
}
```

Funzione utilizzata nel Main per permettere di navigare all'interno dell'universo attraverso la tastiera:

```
// This function adds the possibility to user to move the viewer with keys inside the world.
private void addKeyMovementsToBranchGroup(SimpleUniverse universe, BranchGroup branchGroup) {
    // find view platform transformgroup
    TransformGroup viewTg = universe.getViewingPlatform().getViewPlatformTransform();
    // create viewTg transformation
    Transform3D viewTransform = new Transform3D();
    // create behaviour to navigate with keys
    KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(viewTg);
    // create bound for key navigator
    keyNavBeh.setSchedulingBounds(this.defaultBound);

    // add behaviour to branchgroup
    branchGroup.addChild(keyNavBeh);
}
```

Funzione utilizzata nel Main per permettere di ruotare l'oggetto TransformGroup principale attraverso il mouse:

```
private void addMouseMovementsToTransformGroup(TransformGroup transformGroup, BranchGroup branchGroup)
// permit movements
transformGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
transformGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
// create behaviour
MouseRotate mouseRotate = new MouseRotate(transformGroup);
// set boundsphere
mouseRotate.setSchedulingBounds(this.defaultBound);
// add all
branchGroup.addChild(mouseRotate);
}
```

Funzioni utilizzate nel Main per applicare una luce direzionale:

```
// This function add a directional light to the world with the default direction.
private void addDirectionalLight(BranchGroup branchGroup) {
    addDirectionalLight(branchGroup, 1.0f, 1.0f, 1.0f);
}

// This function add a directional light to the world with a custom direction.
private void addDirectionalLight(BranchGroup branchGroup, float dirX, float dirY, float dirZ) {
    // create light
    DirectionalLight light = new DirectionalLight();
    light.setDirection(dirX, dirY, dirZ);
    // add bounds to light
    light.setInfluencingBounds(this.defaultBound);
    // add light to tg
    branchGroup.addChild(light);
}
```

Motivazioni

Per svolgere questo esercizio sono state principalmente applicate alcune modifiche al codice dell'esercizio 4.2.

La classe Facciata è stata trasformata nella classe Tempio.

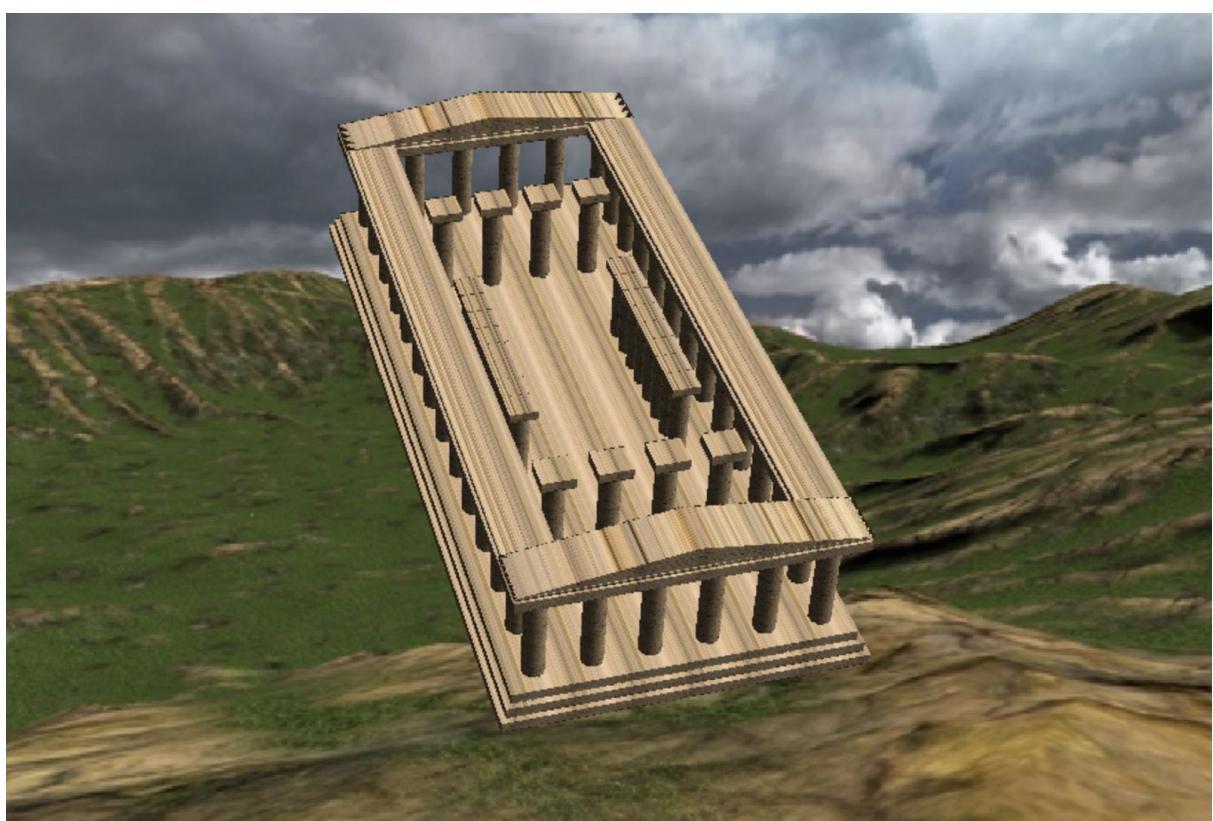
La classe Scalinata è stata utilizzata per costruire la base del tempio. Ad essa, rispetto all'esercizio 4.2, sono state fatte solo delle piccole modifiche relative alla dimensione del singolo scalino.

La classe Tetto è stata modificata permettendo di specificare nel costruttore se visualizzare o meno la copertura del tetto.

La classe Colonna non ha subito variazioni.

Risultato





Conclusion

Recuperando il codice dell'esercizio 4.2 nel quale veniva mostrata la facciata del tempio, è stato costruito l'intero tempio.

Inoltre, l'utilizzo di KeyNavigatorBehavior ha permesso di navigare all'interno dell'universo utilizzando la tastiera e MouseRotate ha permesso di ruotare l'oggetto rappresentato attraverso il mouse.