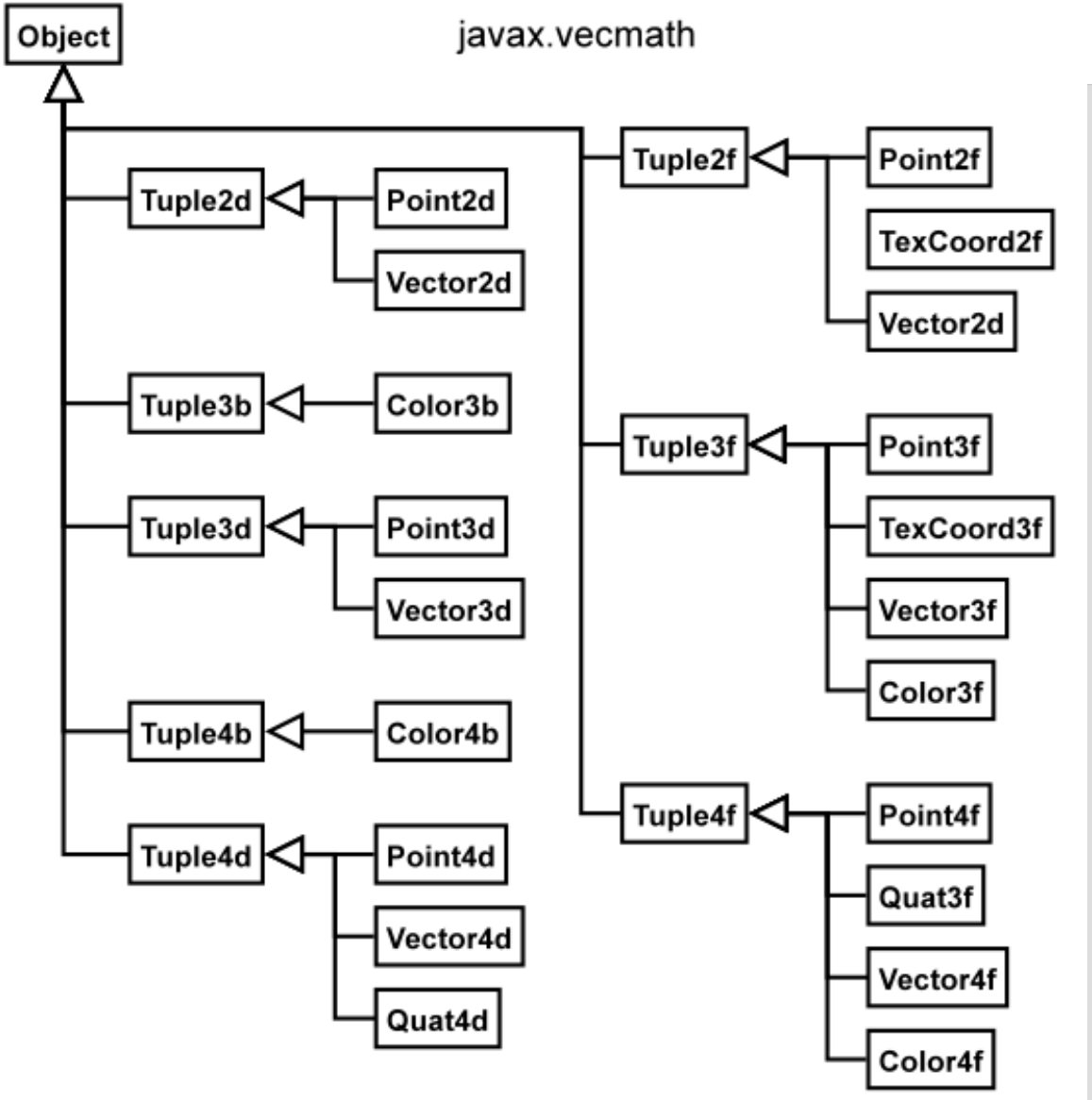
JAVA 3D

Corso di Immagini e Multimedialità
Lezione 3.3: JAVA3D – Proiezioni

cristian.virgili@uniud.it

Tuple


javax.vecmath
Object
Tuple2d
Point2d
Vector2d
Tuple3b
Color3b
Tuple3d
Point3d
Vector3d
Tuple4b
Color4b
Tuple4d
Point4d
Vector4d
Quat4d
Tuple2f
Point2f
TexCoord2f
Vector2d
Tuple3f
Point3f
TexCoord3f
Vector3f
Color3f
Tuple4f
Point4f
Quat3f
Vector4f
Color4f

Point

Rappresenta un punto nello spazio. Può rappresentare:

- un vertice

- la posizione di una sorgente luminosa

- etc.

Vector

Rappresenta una direzione:

- un asse di rotazione

- la normale di una superficie

- la direzione di un raggio luminoso

- una velocità

- etc.

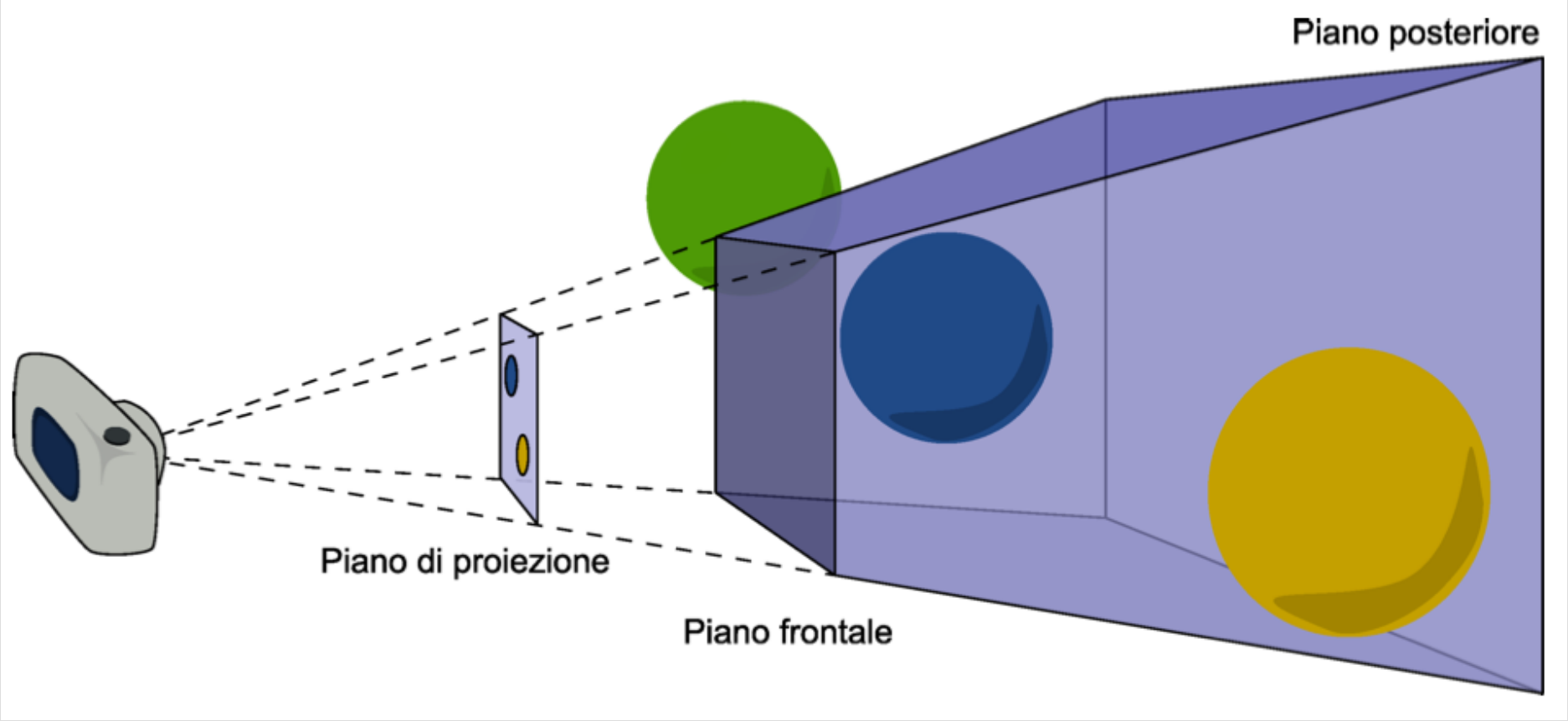
Color

Rappresenta un colore in uno di questi spazi:

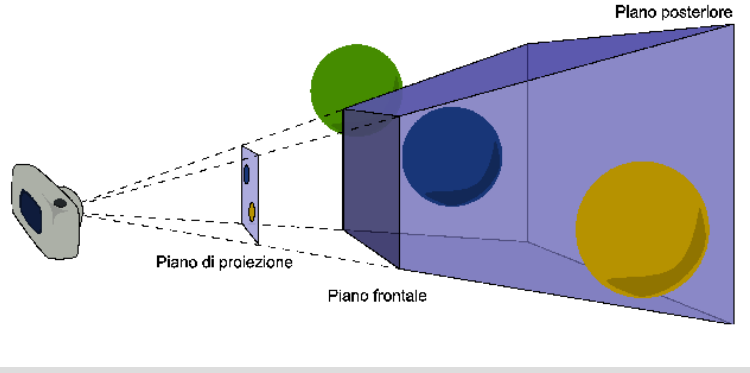
-RGB (red, green, blue): gamma dei colori visualizzabili

- RGBA (red, green, blue, alpha): colori con trasparenza.

Proiezioni



Proiezioni
Frustum



È un solido ottenuto tagliando un cono o una piramide con due piani paralleli.

- Rappresenta la porzione di universo visibile all'osservatore attraverso lo schermo.

- Tutto ciò che **NON** si trova all'interno del frustum (almeno parzialmente) **NON** viene preso in considerazione per la resa della scena (che viene chiamata: **frustum culling**).

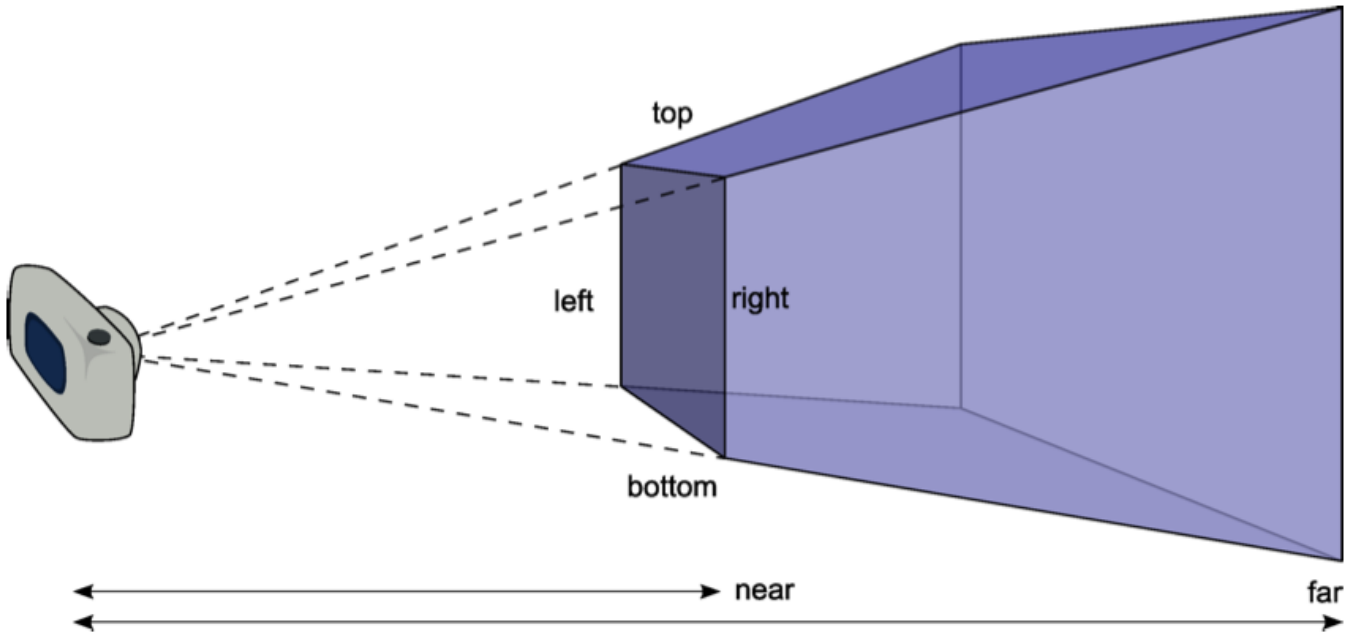
Proiezioni tramite Transform3D

- Transform3D permette di rappresentare sia le proiezioni prospettiche che ortografiche (parallele).

- Fornendo a Transform3D la descrizione del frustum possiamo ottenere la proiezione corrispondente.

frustum()

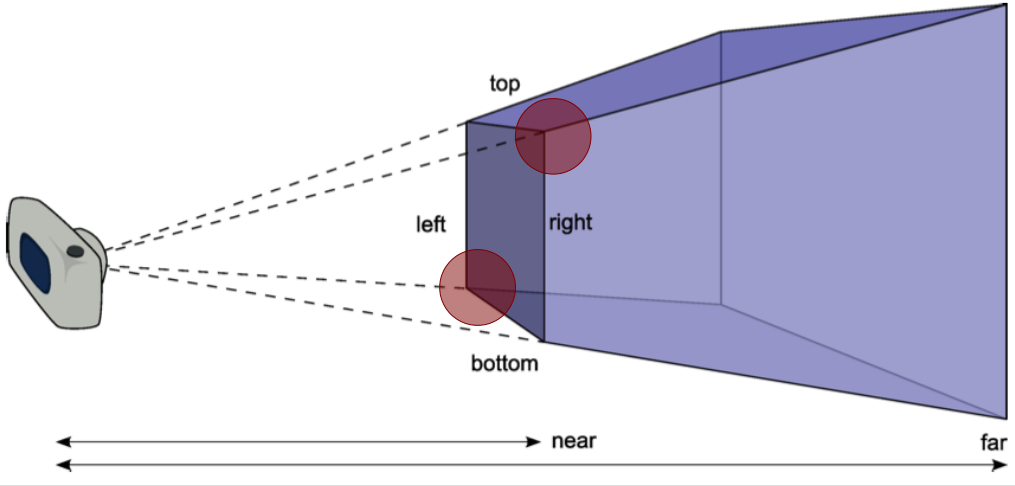
```
public void frustum(double left,
                    double right,
                    double bottom,
                    double top,
                    double near,
                    double far)
```



frustum()

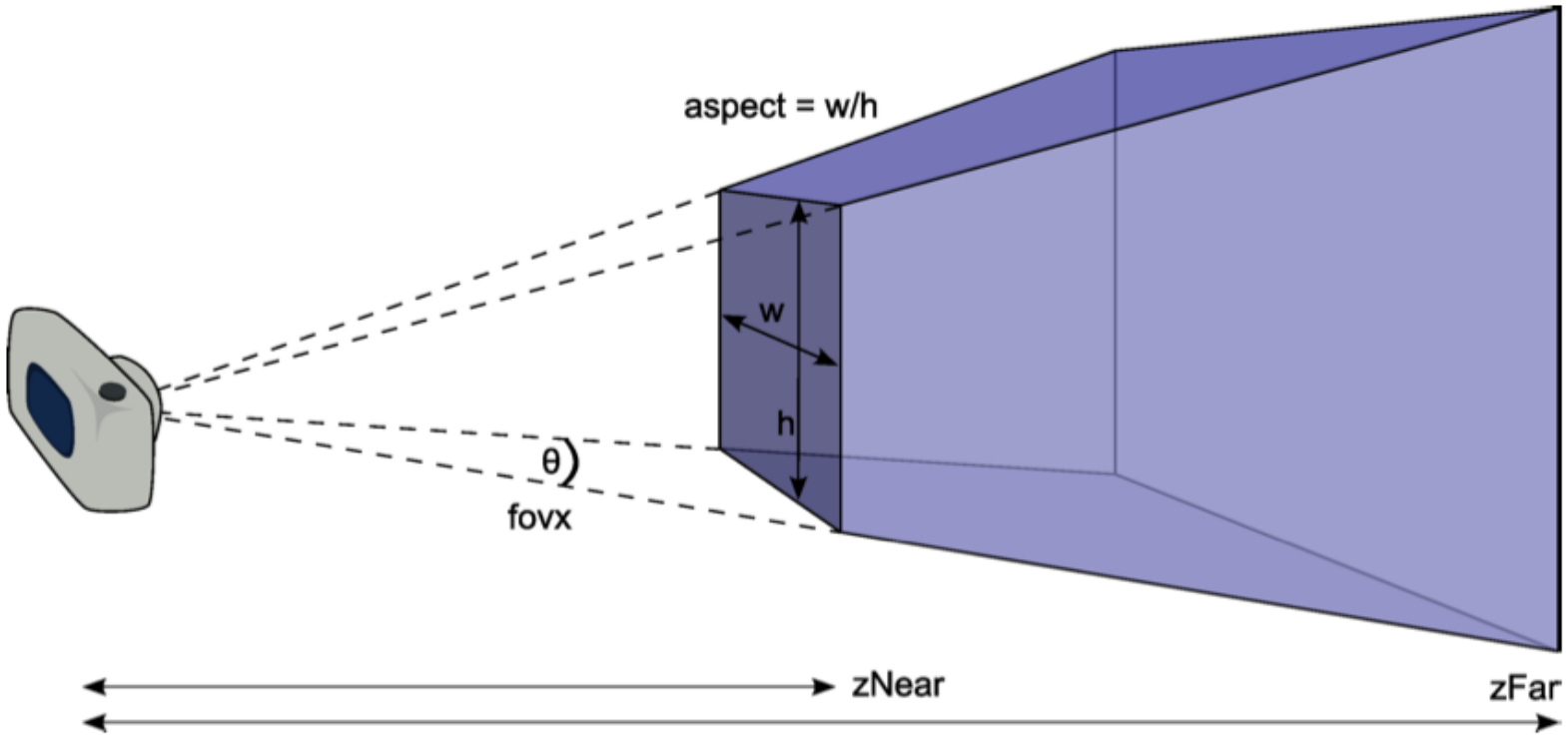
Le coordinate vengono così definite: il piano di proiezione è definito dagli angoli del quadrato che hanno queste coordinate:
Angolo in basso a sinistra (left,bottom,-near)
Angolo in alto a destra (right, top, -near)

```
public void frustum(double left,
                    double right,
                    double bottom,
                    double top,
                    double near,
                    double far)
```



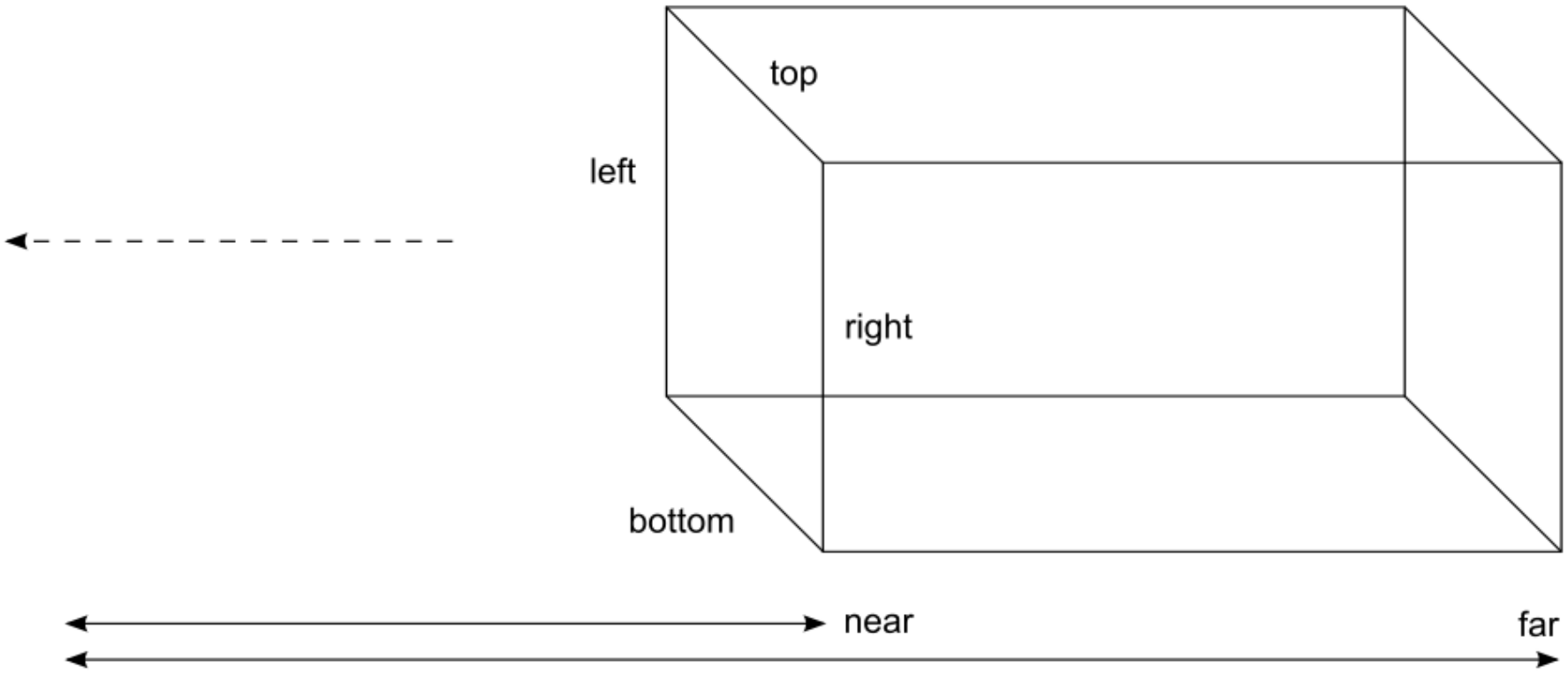
prospective()

```
public void perspective(double fovx,
                        double aspect,
                        double zNear,
                        double zFar)
```



ortho()

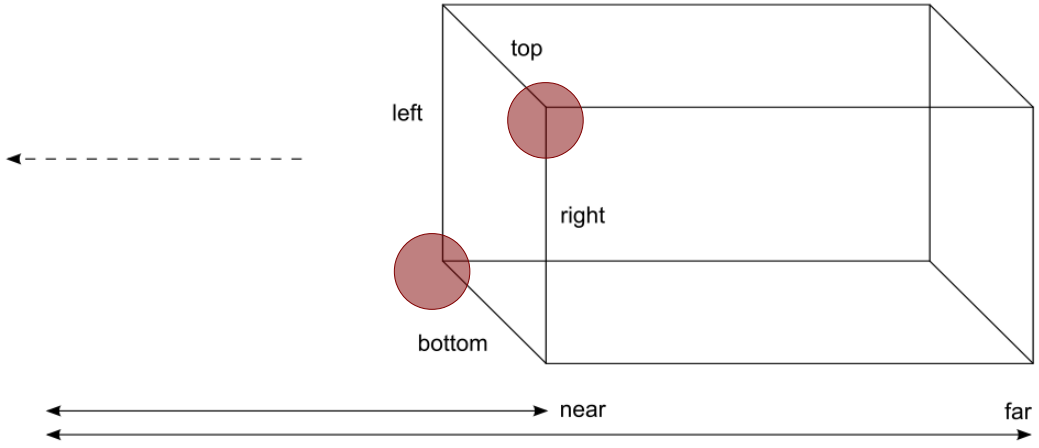
```
public void ortho(double left,
                  double right,
                  double bottom,
                  double top,
                  double near,
                  double far)
```



ortho()

Le coordinate vengono così definite: il piano di proiezione è definito dagli angoli del quadrato che hanno queste coordinate:
Angolo in basso a sinistra (left,bottom,-near)
Angolo in alto a destra (right, top, -near)

```
public void ortho(double left,
                  double right,
                  double bottom,
                  double top,
                  double near,
                  double far)
```

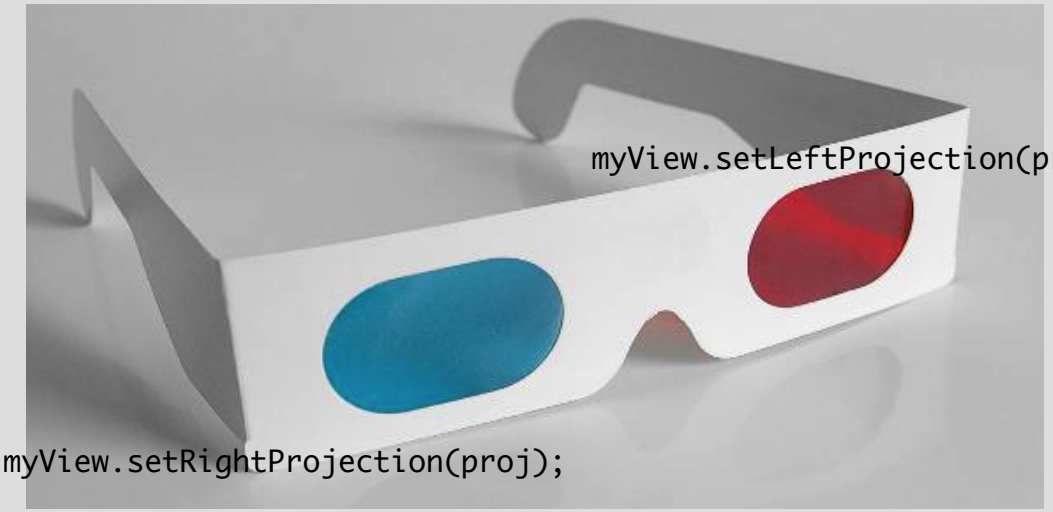


NB: Compatibility mode

Per dire al sistema che la matrice di proiezione può essere modificata bisogna Settare la proprietà della classe View “compatibility” a true

```
View myView = simpleU.getViewer().getView();
myView.setCompatibilityModeEnable(true);
```

leftProjection()
rightProjection()



```
myView.setLeftProjection(proj);
```

```
myView.setRightProjection(proj);
```

Esempio di proiezione (ortogonale)

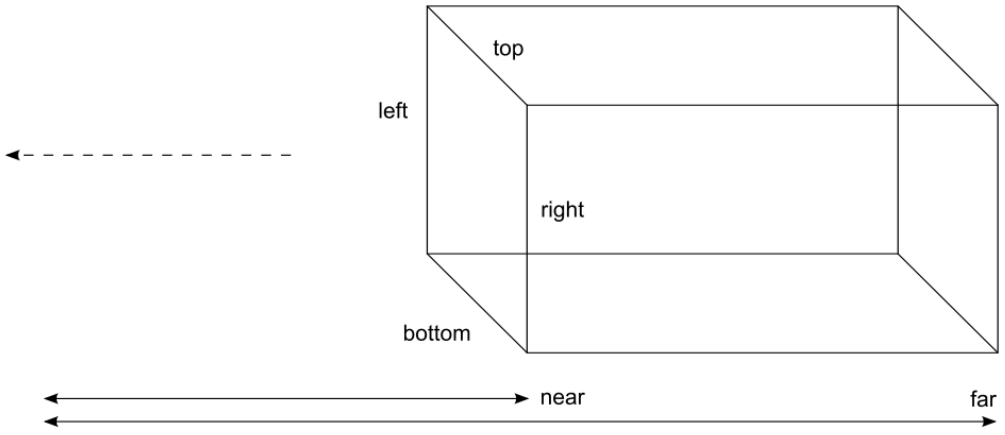
```
SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
simpleU.getViewingPlatform().setNominalViewingTransform();

// accedo all'oggetto view del SimpleUniverse
View myView = simpleU.getViewer().getView();

//Abilitazione del compatibility mode per
//modificare la matrice di proiezione .
myView.setCompatibilityModeEnable(true) ;

// Creazione di una trasformazione
Transform3D proj=new Transform3D() ;

 // Impostazione della matrice di proiezione ortografica
proj.ortho(-2,2,-2.0,2.0, 0.1, 10.0);

//Esempio della proiezione di default:
//  Proporzioni fa   adattare del Canvas3D. a  seconda      delle    dimensioni .
// double ratio =1024.0/768.0;
// proj.perspective(0.25*Math.PI, ratio, 0.1, 10.0);

myView.setLeftProjection(proj) ;

// Se disponessimo di un sistema stereoscopico sarebbe necessario anche:
// currView.setRightProjectio( proj ) ;
BranchGroup scene = createSceneGraph();    // creazione del sottografo principale
simpleU.addBranchGraph(scene);
```

RELAZIONE FINALE
Esercizio 3.4

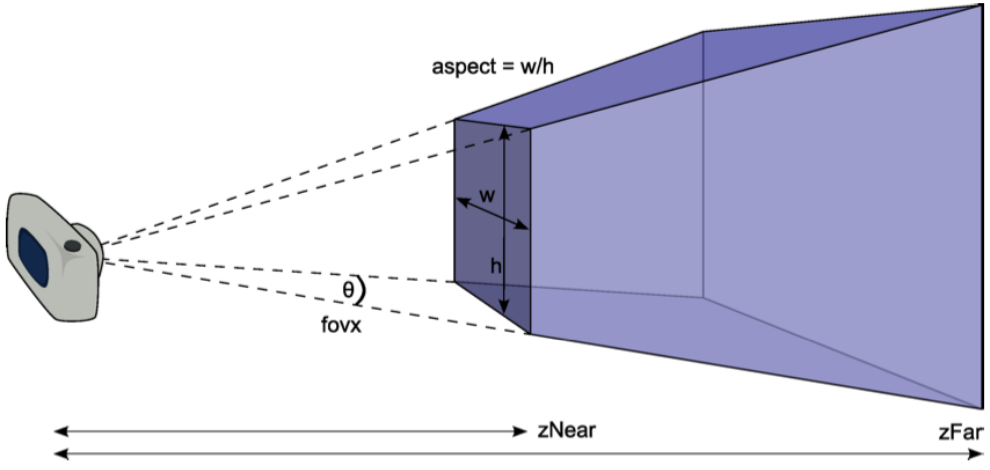
Riprendendo dall'esercizio precedente una scene con 2 o 3 punti di fuga, impostare la proiezione con **setLeftProjection**()

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

```
public void ortho(double left,
                  double right,
                  double bottom,
                  double top,
                  double near,
                  double far)
```



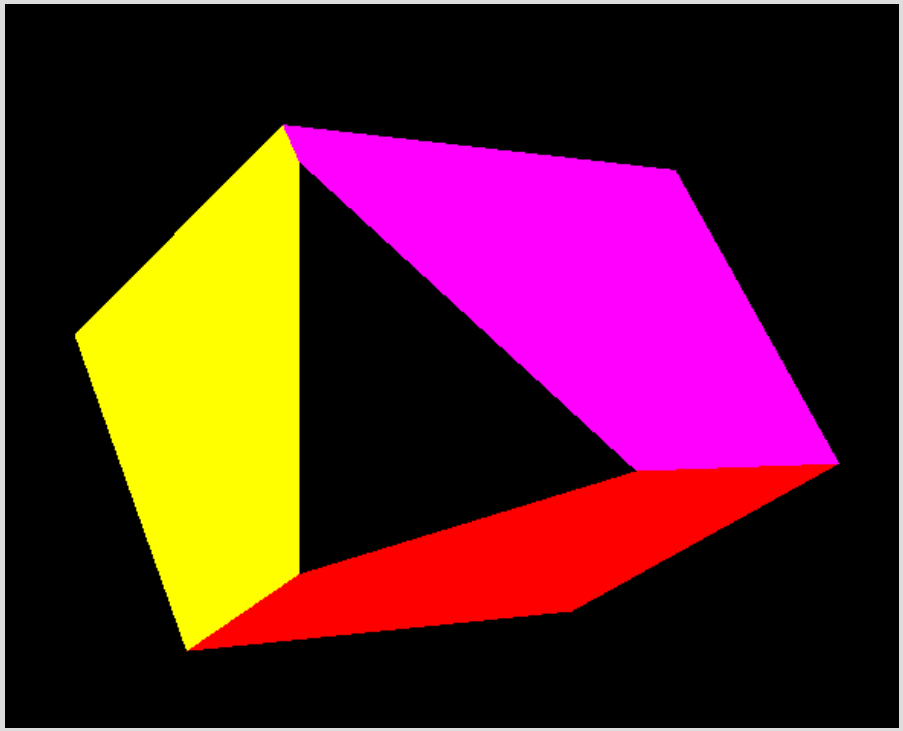

```
public void perspective(double fovx,
                        double aspect,
                        double zNear,
                        double zFar)
```

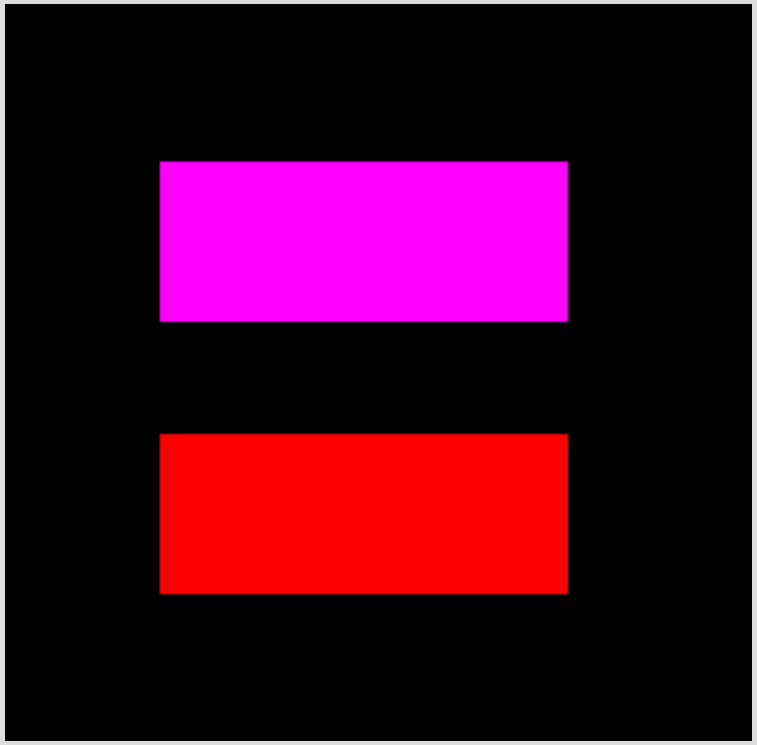


Caso particolare

Caso particolare: traslare il cubo su x di 0.4 e ruotarlo sull'asse delle x di PI/4 dopodiché provare a "tagliare" il cubo spostando il piano di proiezione.

```
proj.ortho(-2,2,-2,2, 2.1, 10.0);
```





Compatibility mode

setLeftProjection() è utile per manipolare direttamente la proiezione,
ma l'uso del compatibility mode è sconsigliato poiché porta svantaggi in termini di limiti nella portabilità e l'impossibilità di utilizzare alcune funzionalità del modello della view.

Per cui è meglio utilizzare altri metodi (policy) della *View* per ottenere lo stesso risultato.

View policy

View permette di impostare diverse policy:

- View policy;
- Projection policy;
- Screen scale policy;
- Window resize policy;
- Window movement policy.

Vi sono altre policy descritte nella documentazione

View
Projection Policy

Si possono indicare le seguenti politiche di proiezione tramite **setProjectionPolicy**():

- PARALLEL_PROJECTION;
- PERSPECTIVE_PROJECTION (default).

Questa policy viene usata in combinazione con i metodi di impostazione della proiezione:
- setFieldOfView()
- setBackClipDistance()
- setFrontClipDistance()

Visibility Policy

Si possono indicare le seguenti politiche di visibilità
tramite **setVisibilityPolicy**():

- VISIBILITY_DRAW_VISIBLE (default): disegna solo gli oggetti visibili;

- VISIBILITY_DRAW_INVISIBLE: disegna solo gli oggetti invisibili;

- VISIBILITY_DRAW_ALL: disegna tutti gli oggetti.

View
Window resize policy

Si possono indicare le seguenti politiche di ridimensionamento tramite **setWindowResizePolicy**():

- VIRTUAL_WORLD: la visibilità dipende dalla dimensione della finestra (più grande è la finestra più cose vedo);

- PHYSICAL_WORLD (default): la zona inquadrata rimane invariata al variare della finestra (se ingrandisco la finestra si ingrandisce l'immagine).

N.B.: funziona direttamente solo con la PARALLEL_PROJECTION. Altrimenti deve essere usato in combinazione con altre policy.

Esempio di variazioni delle policy

```
BranchGroup scene = createSceneGraph();     // creazione del sottografo principale

//Creazione del SimpleUniverse
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 (10.0);

//Impostazione del clip dal piano frontale
myView.setFrontClipDistance (0.1);

//Impostazione del campo visivo
myView.setFieldOfView(Math.PI/4);

//Impostazione del tipo di proiezione
myView.setProjectionPolicy(View.PERSPECTIVE_PROJECTION);
//View.setProjectionPolicy(View.PARALLEL_PROJECTION);

simpleU.addBranchGraph(scene);
```

RELAZIONE FINALE
Esercizio 3.5

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

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

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

Termini

A questo punto possiamo approfondire due termini di Java3D: *live* e *compiled*

inserendo un branch graph in un Locale lo si rende *vivo*, insieme a tutti i suoi figli. Gli oggetti vivi possono essere renderizzati ed i loro parametri non possono essere più modificati.

Per rendere un parametro modificabile anche quando l'oggetto è vivo, la corrispondente **capability** deve essere impostata "*modificabile*" prima che l'oggetto diventi vivo.

Capability (1)

Invocando il metodo *compile()* del BranchGroup, Java3D converte l'intero branch graph in una rappresentazione interna per motivi di performance.

La conversione in una rappresentazione interna ha molti “effetti collaterali”

- un effetto è quello di bloccare i valori delle trasformazioni affini ed altri oggetti dello scene graph

Se non specificato prima, il programma non avrà più modo (**capability**) di cambiare i valori di tali oggetti

Capability (2)

Ci sono casi, però, in cui un programma ha la necessità di cambiare tali valori anche dopo che gli oggetti sono diventati vivi

per esempio, è possibile creare animazioni cambiando i valori di un *TransformGroup*

Per far accadere ciò, il *TransformGroup* deve essere capace di cambiare anche dopo essere diventato ***vivo***

Capability(3)

Una caratteristica di un oggetto che può essere modificata è detta **capability**

- ogni oggetto ha un insieme diverso di capability

- Impostando o azzerando una capability si decide se una particolare caratteristica può essere modificata dopo che l'oggetto è diventato vivo

Esempio di capability

Aggiungiamo al nostro mondo virtuale la possibilità di interagire con esso.

```
public Rotazione() {
                setLayout(new BorderLayout()); //layout manager del container
                //trova la miglior configurazione grafica per il sistema
                GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
                // Canvas3: si occupa del rendering 3D on-screen e off-screen
                Canvas3D canvas3D = new Canvas3D(config);
                add("Center", canvas3D);
                SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
                simpleU.getViewingPlatform().setNominalViewingTransform();
                TransformGroup viewTransformGroup =
                        simpleU.getViewingPlatform().getViewPlatformTransform();
                 //Comportamento predefinito di rotazione legata agli eventi del mouse
                 MouseRotate rotateBehavior = new MouseRotate ( ) ;
                 //      Legame fa il comportamento e il TranformGroup.
                 rotateBehavior.setTransformGroup(viewTransformGroup) ;

                 //Zona in cui tenere conto degli eventi. Sfera di raggio 1 con centro nello 0
                 rotateBehavior.setSchedulingBounds(new BoundingSphere(new Point3d(0.0,0.0,0.0), 1.0));

                //TranformGroup da legare alla rotazione interattiva.
                 TransformGroup mainTG = new TransformGroup() ;
                 ColorCube cube = new ColorCube(0.4);
                 mainTG.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE) ;
                 mainTG.addChild(cube);
                 BranchGroup objRoot = new BranchGroup( ) ;
                 objRoot.addChild(mainTG);
                 objRoot.addChild(rotateBehavior);

                simpleU.addBranchGraph(objRoot);
        }
```