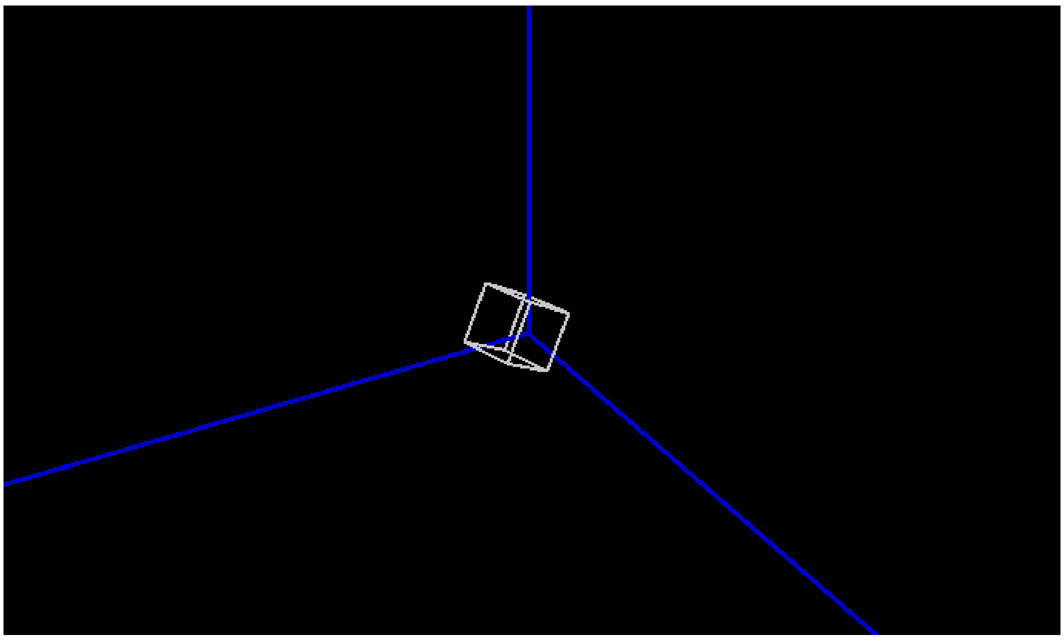
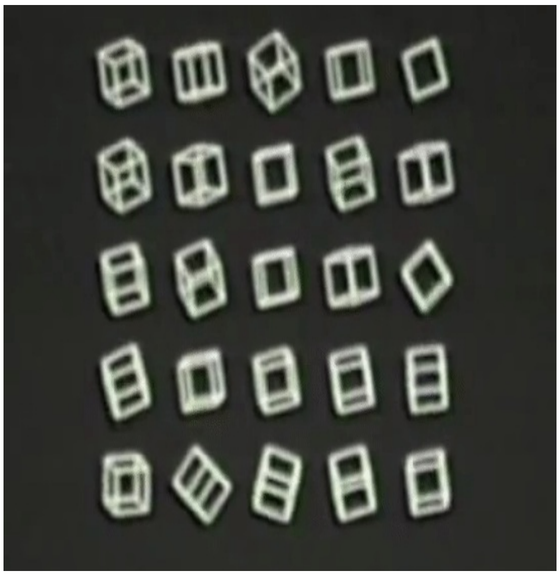
TD 1 - Homogeneous coordinates linear transformations (with an introduction to Processing)

(Luca Castelli Aleardi, 18th sept 2018)

How to create a simple screen saver? The goal of this TD is to provide a simple applet animating 3D cube screen saver, as in the video below on the right (by Manfred Mohr, 1973-74, Cubic Limit)





0. Getting started with Eclipse and Processing

Processing is a java based programming language providing an easy and nice environment for dealing with computer graphics, image processing, geometric modelling, ...
In order to correctly configure Processing and Eclipse, please follow these procedures.

1. Performing simple transformations on points in the plane

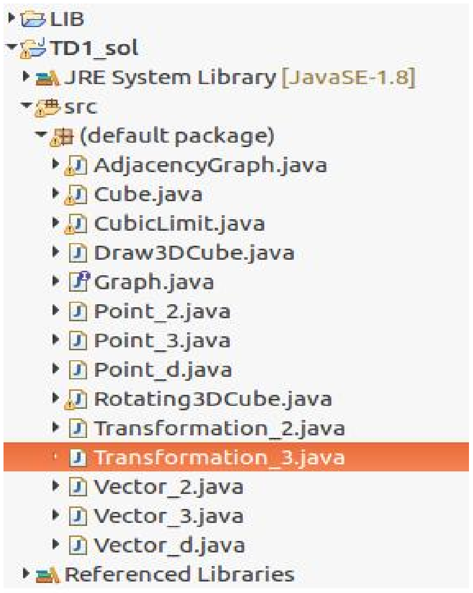
Files to download today

Here two Java libraries to install (in a "general Eclipse Project", use a LIB directory):

- Jama library (for matrix computations)
- Processing (for 2d rendering)

Here are some files to download (and extract to the /src folder of your Eclipse project):

- src.zip: contains a number of classes for representing and manipulating geometric objects (such as points, vectors, graphs, ...).
- Rotating3DCube.java: this class provided main methods for drawing graphs (to be completed today)
- Transformation_3: allows to describe projective transformations in 3D space, using matrix representation involving homogeneous coordinates.



Java documentation

Here is the javaDoc concerning the packages and classes used today:

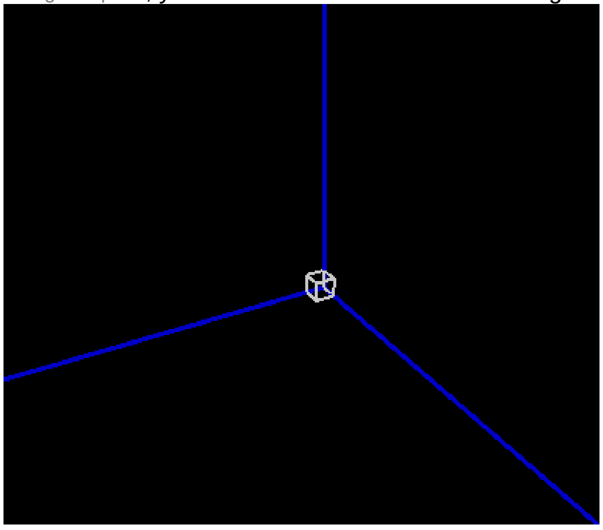
- Jama documentation;
- Processing documentation;
- TD1 documentation.

Testing the class Rotating3DCube

In order to provide a simple way of drawing and animating graphs (skeletons of 3D meshes) we will use classes Cube and Rotating3DCube.
Class Rotating3DCube provides a very basic and simple way of performing drawings in a 3D Processing frame.
Class Square provides the main methods for creating and animating 2D meshes. More precisely, it contains

- Cube cube, which stores the (geometric) graph to display: it is a 3D cube in our example (located in such a way that one of its vertices coincides with the origin)
- Draw3DCube renderer, providing methods for drawing 3D segments in 3D space

When executing the first time class Drawing2DSquare, you should obtain a frame visualizing a square, as below:

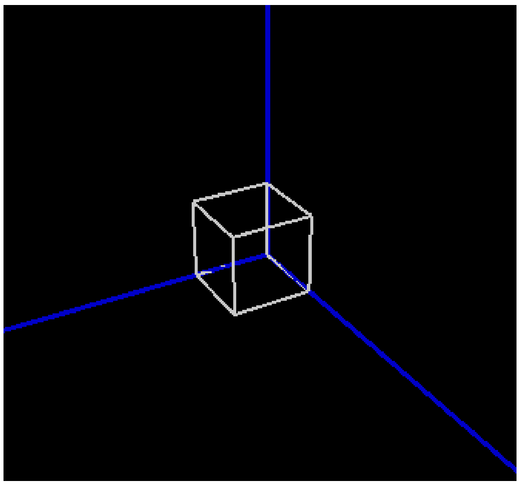


1.0 Scaling the cube

Recall that with the use of homogeneous coordinates, any scaling can be expressed in a matrix form.

- complete the function public static Transformation_3 scaling(double s) of class Transformation_3, which constructs and return a (matrix) transformation corresponding to a scaling.
- Modify method public void scale(double zoomFactor) in order to perform a scale of the cube: this method must return a Transformation_3

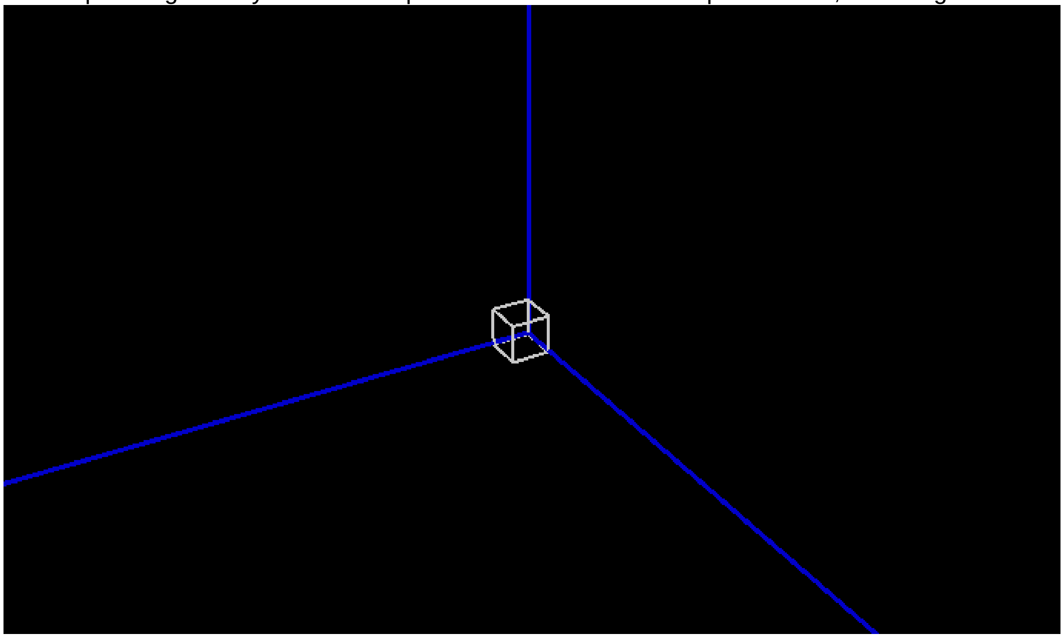
You can test your code pressing the key 'i' and 'o' (on your keyboard) to perform the scaling, obtaining the following result:



1.1 Moving (vertically) the cube

- Complete method moveZ(float speed) which moves the square along vertical direction.

You can test your code pressing the key 'u' and 'd' to perform to move the cube up and down, obtaining the result below:



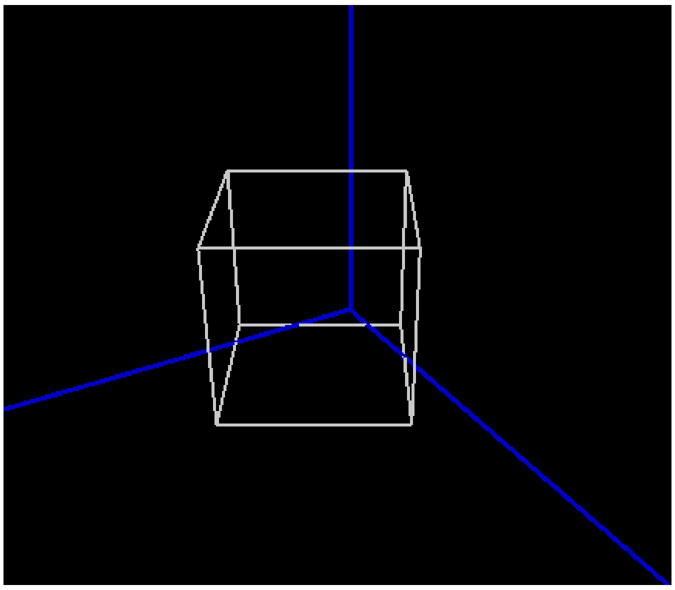
1.2 Rotating the cube (around its barycenter)

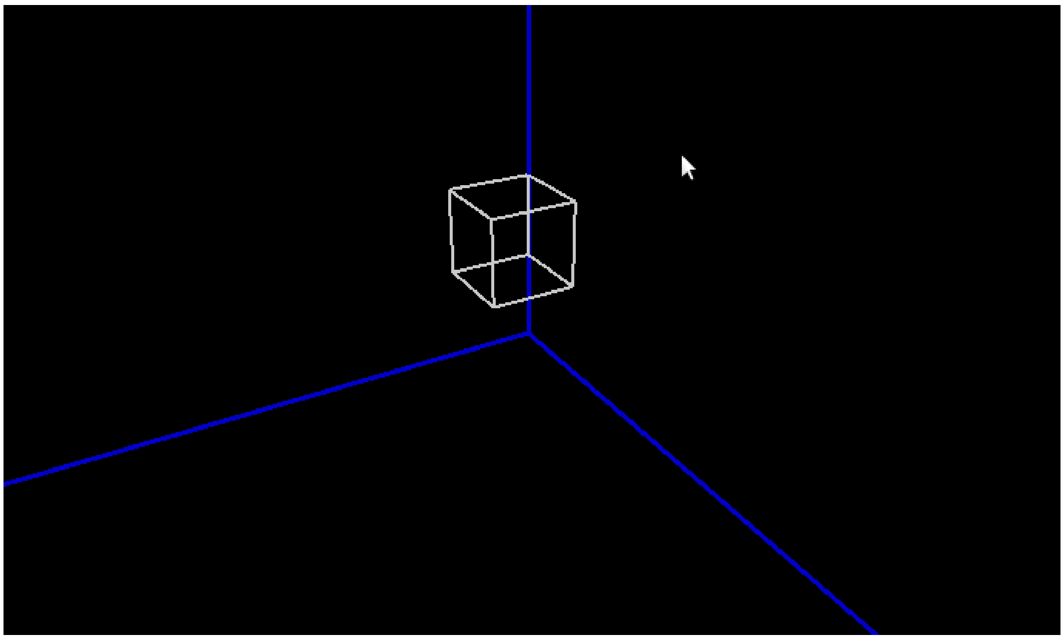
Recall that rotations can also be expressed in terms matrix multiplications, as in the example below (rotation around z axe). In your case, you should use a matrix of size 4x4, as we make usedof homogeneous coordinates.

$$\mathcal{R}_{z,\theta}\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}x\cos\theta - y\sin\theta\\ x\sin\theta + y\cos\theta\\ z\end{pmatrix}$$

- complete the functions rotationAxisX(double theta) of class Transformation_3, which construct an return the (matrices) transformations corresponding to rotations around the three axis.
- Complete the functions rotateX(double angle), rotateY(double angle) and rotateZ(double angle) of class Rotating3DCube in order to rotate the cube arounf its barycenter.

Suggestion: you can first implement and test the rotation around the x-direction, using the key 'x' on your keyboard.
Once you have implemented all 3 rotations, you sould be able to obtain (using the 3 keys 'x', 'y' and 'z':

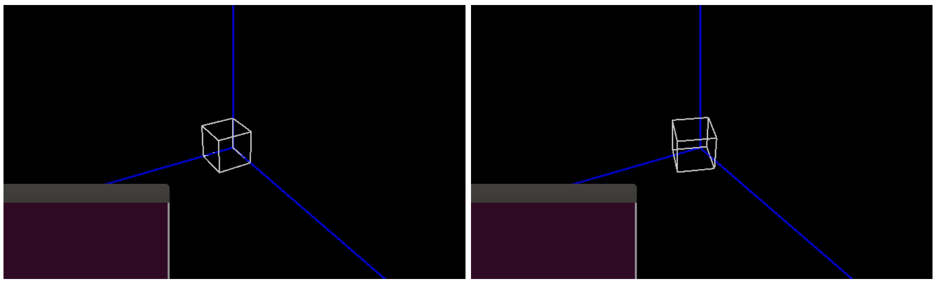




1.3 Rotating the cube around a diagonal (bonus)

- Complete method `public void rotateAroundDiagonal(double angle)` in order to perform a rotation of the cube around one of its diagonals.

2. Animating the cube



The Processing library allows us also to animate objects in 3D space: this is done by the `draw()` function, whose are instructions are run periodically, in order to have an animation effect.

2.1 A decreasing/increasing cube

As first example, you can just run the `CubicLimit` class: you should obtain an animating cube, whose size decreases and increases

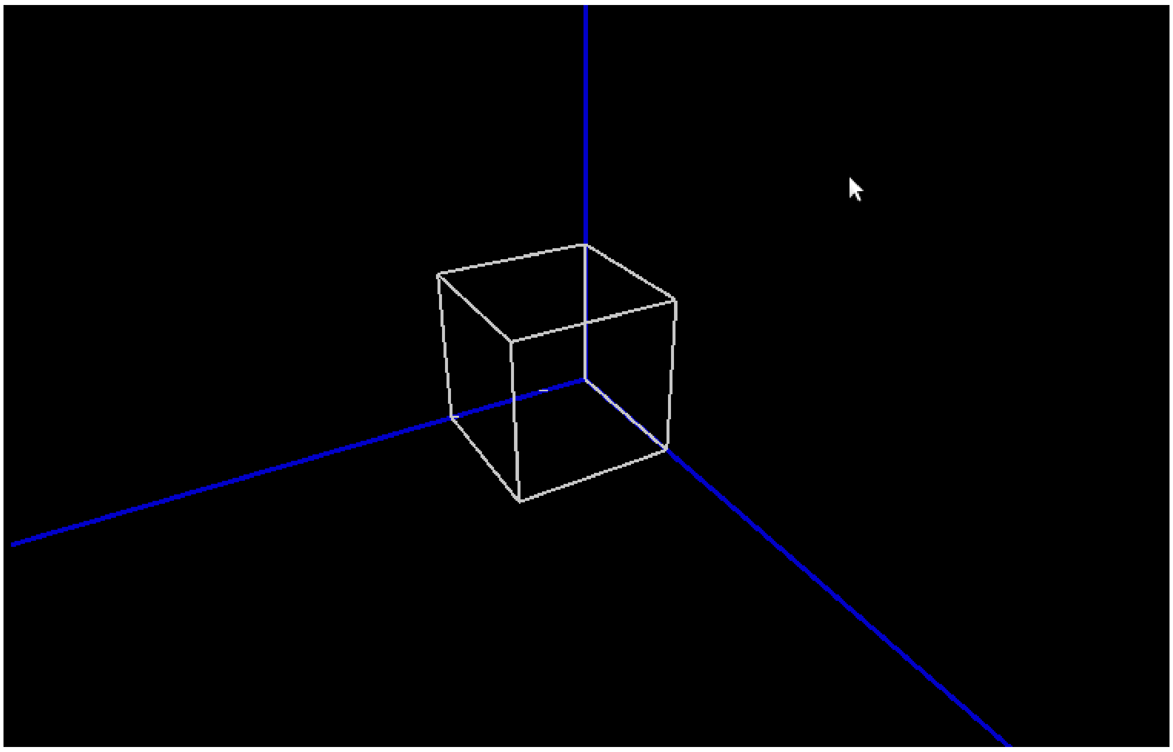
over time asin the picture above.
Take a look to the function public Cube animatingCube(double size), which creates a small cube and perform the update of the scaling transformation. This function is called by the draw() function, in order to create an animation effect.

2.2 A rotating cube

Complete the function public Cube rotatingCube(double size), which creates a small cube and computes the animation of a rotating cube.
To test your code, you have just to remove the comment in thedraw() function:

```
//Cube cube=this.rotatingCube(80); // remove comment to test this function
```

3. An arcball using quaternions: dealing with mouse events



The goal of this section is to provide a basic implementation of an arcball (as in the example above): this allows to deal with mouse events to manipulate the rotation of your 3D scene.
You will be asked to use quaternions to perform rotations in 3D space
 You have to complete the class Quaternion, in order to:

-) complete function multiply(), which perform the multiplication of two quaternions,
-) complete function getValue(), returning the angle and rotation vector encoded by the quaternion
Then you can test your code by running the class CubicLimit: you have just to remove the comment in the function setup():
//ArcBall arcball = new ArcBall(this); // for interaction with mouse events and 3D rendering