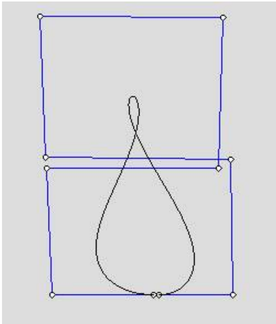
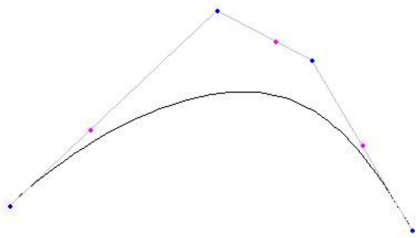
TD 3 -Bézier curves

(Luca Castelli Aleardi)

The goal of this TD session is to implement planar (rational) Bézier curves.





Examples of Bézier curves (non rational and rational case respectively)

1. Getting started (a Processing frame for testing interpolation schemes)

Files to download today

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

- geometry.jar: some classes for manipulating geometric objects (such as points, vectors and transformations in 2D).
- src.zip: the classes for drawing Bezier curves and surfaces.
- Curve.java: abstract class defining main methods for computing the plot of the curve
- DrawCurve.java: main drawing class, allowing to handle mouse events (adding, moving and deleting points).
- DrawSurface.java: main drawing class for drawing Bezier surfaces in 3D

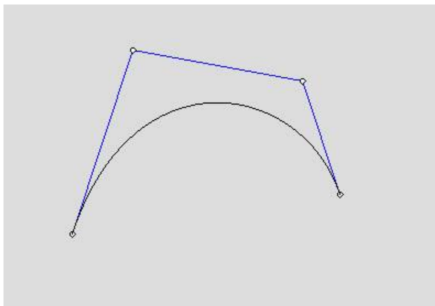
class drawCurve

Class DrawCurve provides simple methods for drawing (points and segments), and handling mouse events (for adding/moving/deleting points). Method setup() allows to select the preferred implementation of Bézier curves.

```
public void setup() {
 size(400,400); // size of the window
 // choose the scheme and curves to draw
 this.scheme=new Bezier(this, this.points);
}

public void draw() {
 background(220);

 if(points.isEmpty()) return; // no control points
 scheme.plotCurve(1./1000.);
}
```

2. Bézier curves

2.1 Implementing the de Casteljau algorithm



Example: cubic Bézier curve:
(x_0, y_0) ... (x_3, y_3).

Given a set of n points, we want to implement the de Casteljau algorithm. You are asked to

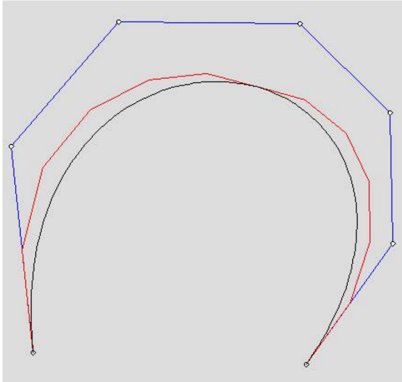
- write a class Bezier which extends class Curve.
- implement method Point_2 evaluate(double t) which computes the point b(t).
- implement method plotCurve(double dt) which draw the curve in the frame, by evaluating the curve with a given step.
-

a. implement method Point_2 recursiveDeCasteljau(int r, int i, double t) which computes the i-th Bézier point (at setp r), for value t, with the recursive definition.
b. implement method Point_2 iterativeDeCasteljau(double t) which computes the Bézier curve with the iterative scheme (in linear space).

You can follow the code below:

```
public class Bezier extends Curve {

 Transformation_2 transformation;

 public Bezier(DrawCurve frame, LinkedList<Point_2> p) {
  super(frame, p);
  transformation=null;
 }

 public Bezier(DrawCurve frame, LinkedList<Point_2> p, Transformation_2 transformation) {
  super(frame, p);
  this.transformation=transformation;
 }

 public Bezier(DrawCurve frame, Point_2[] points, Transformation_2 transformation) {
  super(frame, points);
  this.transformation=transformation;
 }

 ...
 ...
 ...

 public Bezier[] subdivide(double t) {
  Point_2[] controlPolygon=this.points;
  int n=this.points.length-1; // degree and number of edges of the control polygon

  Point_2[] b0=new Point_2[n+1]; // first control polygon
  Point_2[] b1=new Point_2[n+1]; // second control polygon
  Bezier[] result=new Bezier[2]; // the pair of Bezier curves to return as result
```

```
//throw new Error("To be completed: TD3");

// store and return the pair of new control polygons
result[0]=new Bezier(this.frame, b0, this.transformation);
result[1]=new Bezier(this.frame, b1, this.transformation);
return result;
}

}
```



A Bézier curve subdivided with parameter t=0.5
The two sub-curves are drawn in red.

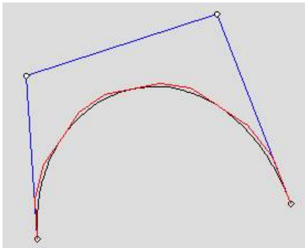
2.2 Compute Bézier curves with the efficient (nested) evaluation

a. implement method `Point_2 BernsteinBezier(double t)` which computes the Bézier curve with nested evaluation of the Bernstein form of a Bézier curve.

Remark:

What do you observe implementing and comparing these methods?

2.3 Render a Bézier curve with the subdivision method

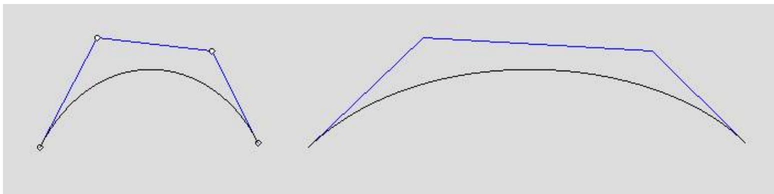


Example of subdivision rendering of a cubic Bézier curve

a. implement method `public Bezier[] subdivide(double t)` which subdivides the Bézier curve into a couple of Bézier curves, with parameter `t`. Modifiy the plot method of your class, in order to visualize the corresponding control polygons of the two curves (see the picture on the right).
b. implement method `public void subdivisionRendering(int n)` which renders the Bézier curve (subdivide `n` times with parameter `t=0.5`). Remove comments from file `DrawCurve.java` in order to plot the curve with the subdivision method.

2.4 Check affine and projective invariance of Bézier curves

Now we want to chech the affine invariance of Bézier curves.

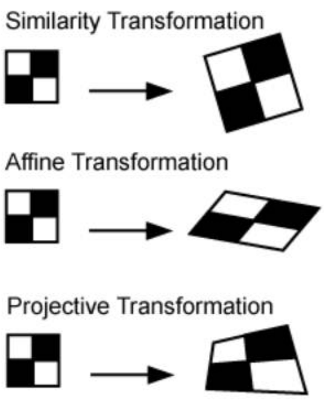


- modify your class `Bezier` in order to plot the original curve and the curve after affine transformation.

a. firstly, plot the apply the transformation to all points of b(t)
b. apply the transformation only to the control points, and then compute the curve

Suggestion:
add a transformation variable `Transformation_2 transformation` to your class `Bezier`.

- repeat your experience testing with projective transformations (bonus).

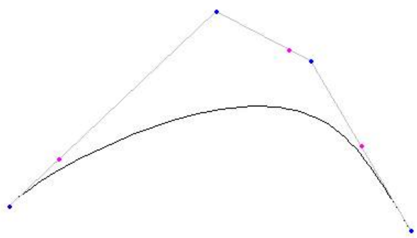


3. Rational Bézier curves (bonus)

3.1 Implementing rational de Casteljau algorithm

Here you are given a set of `n` points $\mathbf{b_i}$ and a set of `n` weights $\mathbf{w_i}$: weights are choosen at the beginning (as constants)

- complete class `RationalBezier` which extends class `Curve`.
- implement method `Point_2 evaluate(double t)` which computes the point b(t).
- implement method `plotCurve(double dt)` which draw the curve in the frame, by evaluating the curve with a given step.



Example: cubic rational Bézier curve:
the shape depends both on control points and weight points

a. implement method `Point_2 rationalDeCasteljau(int r, int i, double t)` which computes the Bézier point, at value t, with the recursive definition.

3.2 Test projective invariance of rational Bézier curves

- modify your class RationalBezier in order to plot the original curve and the curve after projective transformation.

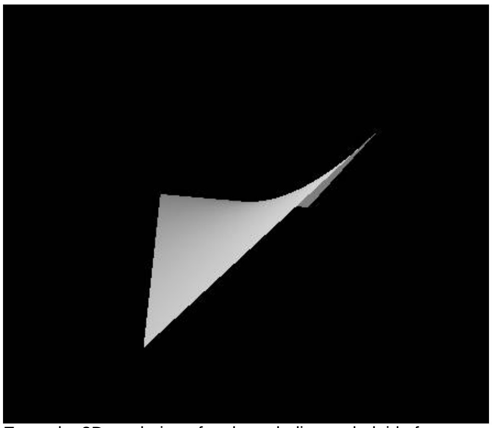
4. 3D with Processing

4.0 drawing surfaces with Processing: getting started

Class DrawSurface allows to render a a surface in a 3D frame. You can choose your favourite implementation (of a parametric surface), by removing/adding comments in the method setup().

- Test with class Paraboloid: this class extends class Surface.

Remark: please observe that we have performed a ***scaling*** of the surface points, in order to obtain a better rendering of the 3D scene.
(take a look to method evaluate(u, v) of class Paraboloid)



Example: 3D rendering of an hyperbolic paraboloid of equations y=zu
The rendering is performed by Processing, using quad strips

Implement the computation of a ***Bézier patch***.

4.1 Bézier patches (tensor product surfaces)

Re-use Bezier classes in order to design Bezier curves (coordinates must be computed in 3D), and then use the tensor product to compute the resulting surface (the Bézier patch)

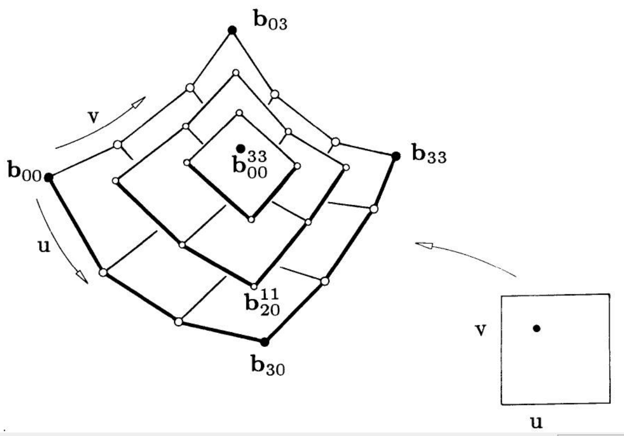
- complete class BezierPatchTensorProduct which extends class Surface.
- implement method Point_3 evaluate(double u, double v), which is required to plot the surface.

You can also use the tensor product (compute separetely two curves)

$$\mathbf{b}^{m,n}(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} \mathbf{b}_{i,j} B_i^m(u) B_j^n(v).$$

4.2 Bézier patches (De Casteljau algorithm) [Bonus]

- complete class BezierPatchDeCasteljau which extends class Surface.
- implement method Point_3 evaluate(double u, double v), which is required to plot the surface.



You can apply directly the De Casteljau algorithm:

Given $\{\mathbf{b}_{i,j}\}_{i,j=0}^{n}$ and $(u,v) \in \mathbb{R}^2$,

$$\mathbf{b}_{i,j}^{r,r} = \begin{bmatrix} 1-u & u \end{bmatrix} \begin{bmatrix} \mathbf{b}_{i,j}^{r-1,r-1} & \mathbf{b}_{i,j+1}^{r-1,r-1} \\ \mathbf{b}_{i+1,j}^{r-1,r-1} & \mathbf{b}_{i+1,j+1}^{r-1,r-1} \end{bmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix}.$$

$$r = 1, \ldots, n$$
$$i, j = 0, \ldots, n-r$$

where initial points are:

$$\mathbf{b}_{i,j}^{0,0} = \mathbf{b}_{i,j}.$$