Introduction

This class notes are intended for classes in geometric modeling. I intentionnaly wrote these notes in English so that they can be used in many contexts, but also, because I believe that students should be used to work with material in English.
These notes are intended for being a concise, introductionary text for geometric modeling. There are many interesting, clear and well written books. The main advantage of these course notes compared to the books is to be more concise. These references will be given in these notes, for further reading.

Goals of geometric modeling

Geometric modeling is concerned about the mathematical kernel of CAD (Computer Aided Design) applications. These applications mainly require to manipulate objects with a computer. Althought volume representation may be used to model 3D objects like smoke, most solid objects are modeled with a surface representation. These surfaces need to be coded in some way both understandable by a computer, but also by a non mathematician user. Therefore, the chosen representation need to be numeric (for the computer) and intuitive (for the user).
There are two ways to interact with the world around us when modeling objects. The first way is to generate virtual objects. This part is concerned with synthesis. These virtual object may be intended to be created, for example, when creating a new car model using a CAD system ; but, these object may also be intended for staying virtual, like our favorite figures in animation movies, monsters in video games, or explanations added to a tool in the setting of augmented reality. The second way is to model real existing objects. This part is concerned with reconstruction and analysis. For example, in the setting of reverse engeneering, when you have a part of a motor that you would like to be able to reproduce, or in the setting of art, where models of existing statues are made avialable on the internet, like the Michael Angelo project of Stanford University ($http://graphics.stanford.edu/projects/mich/$).
In these notes, we shall study a particular model, the parametric, surface model of objects. A 3D (3-dimensional) object can be modeled by its boundary (its surface), or by its volume. Most applications use surface representation, but not all. For example, to model non rigid objects like smoke or liquid, a volume representation is prefered. The parametric setting is also, from far, the most commonly used model. Alternatively, implicit models are useful in certain applications, for example a video game, needing a lot of collision detection.
In this notes, we intend to study parametric surfaces in 3D, that is, 2-dimensional objects of a 3D space. But before getting to surfaces, the most natural thing to do is to study curves, that is, 1D objects in a 2D space, or more generally in a 3D space. Of course, most of the theory will hold for a m-dimensional object into a n-dimensional world (when $m < n$), but little interest is given by being so general when the useful settings are only the cases $n \leq 3$. Note though, that a fourth dimension may be used for time, commonly called the $3D + t$ or the $2D + t$ setting.

Chapitre 1

AFFINE GEOMETRY

Indeed, even before studying curves, we shall look in details at their building blocks : the points. A curve, or a surface, is defined as a set of points. These points will be parameterized by one, in the case of curves, or two parameters in the case of surfaces. But, let us first consider the spaces containing points.

1.0.1 Point : the building block

How would you define a curve or a surface ? indeed, most naturally, a curve or a surface is given as a set of points.
And, what are points ? 'Point' is actually a notion that is well studied in high school. There, your teachers probably told you a lot about points (mine actually did). In school, we studied points, different systems of coordinates to locate them in a referency, sets of points, and transformations that applies to them. Strangely, after graduating from high school, your teachers -or maybe they prefer to be called professors- did not seem to be really concerned with points (at least mine did not). We studied vectors a lot, vector spaces, and associated linear transformations and matrix representations, but little mention was given to points. Briefly, affine spaces where mentioned, but not extensively studied like their linear friends, the vector spaces.
Why is it important to distinguish points from vectors : If we take a referency, coordinates are defined in this referency. And, you should remember that indeed the point P and the vector $\vec{OP}$ have the same coordinates. So, we could just consider that working on P or on $\vec{OP}$ is equivalent. This assumption would make our life easy since OP is a vector, and we know well how to handle vector spaces. Unfortunately, $\vec{OP}$ and P are not the same. Here are two good reasons to distinguish them. First, consider a translation. The effect of the translation on a point P is to move the point by the translation vector. But, a translation on a vector $\vec{OP}$ has no effect, since a vector is the equivalence class of equipotent bi-points. Another example is that we can multiply vector by a scalar, in particular λOP makes sense : it is a vector with the same direction and orientation that OP and a norm $\lambda * ||OP||$. But multiplying a point by a scalar does not make sense. Suppose we take for the point λP the point P' such that $\vec{OP'} = \lambda \vec{OP}$. This definition is not correct because it depends of the origin O of the referency. For a different origin O' the result would be different. So, points have to be different than vectors.
In the following, we shall define point spaces, that is, affine spaces and their bases.

1.0.2 Affine Spaces

The points leave in an affine space. When studying linear algebra (vector spaces), affine spaces are often characterised by a point and a vector space. For a given point P in the affine space $\mathcal{A}$, all the other points of $\mathcal{A}$ are written as $P + v$ where v is a vector of the vector space V associated with $\mathcal{A}$. Note that everything we shall consider here is in finite dimension.

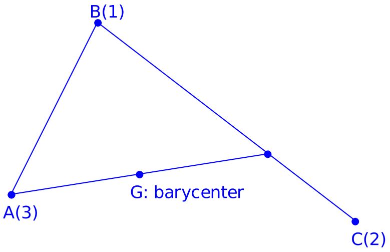
Here, we will rather define an affine space, independently of the associated vector space. As a vector space is a space of elements called vectors, stable by linear combination, an affine space is a space of elements called points stable by affine combinations. An affine combination is a particular linear combination such that the sum of the coefficients is one. That is, if A_i, $i = 0, ..., m$ is a set of points in an affine space $\mathcal{A}$, then

$$B = \sum_{i=0}^{m} \alpha_i A_i \text{ where } \sum_{i=0}^{n} \alpha_i = 1 \tag{1.1}$$

is also a point of the affine space $\mathcal{A}$. Affine combinations are closely related to barycenter. The point B defined by equation (1.1) is the barycenter of the point A_i respectively given the weight α_i. Therefore, an affine space can be defined such that, any barycenter of points in this space is also in this space. This interpretation gives an intuitive interpretation of affine combinations, at least for positive coefficients.

In this definition, the link with the associated vector space can still be made. As mentionned before, the associated vector space is the space of equivalence classes of equipotent points, that is, pair of points with the same difference.

A classical example of an affine space is the Euclidian plane, that is points in 2D : all the barycenters of two points in the plane, is a point in the plane.



Point arithmetic

We just defined affine combinations, and learnt that an affine combination of points is a point. We can do even more point arithmetic, using the link between points and vectors. In fact, the vector $\vec{AB}$ is given by the difference $B - A$. So, we can substract two points, and get a vector. More generally, any linear combination of points, such that the sum of the coefficients is 0 is a vector.

Let consider equation (1.1) and use some point arithmetic :

$$\begin{aligned} B &= \sum_{i=0}^{m} \alpha_i A_i \text{ where } \sum_{i=0}^{n} \alpha_i = 1 && (1.2)\\ &= (\sum_{i=0}^{n} \alpha_i) A_0 + \sum_{i=0}^{n} \alpha_i (A_i - A_0) && (1.3)\\ &= A_0 + \sum_{i=0}^{n} \alpha_i \vec{A_0 A_i}. && (1.4) \end{aligned}$$

So to find the point B, start from A_0 and add one by one the vectors $\alpha_i \vec{A_0 A_i}$.
So, to summarize on points arithmetic, linear combinations of points are valid only if the sum of the coefficients is 1 (that is, an affine combination) or if the sum of the coefficients is 0. If the sum is 1, the combination is a point, if the sum is 0, the combination is a vector.

1.0.3 Affine basis

Points basis

A basis is a set of elements, such that any other element of the space can be expressed relatively to this basis. This expression must be unique. In the context of finite vector space, a basis is a set of vectors, such that any element of the space can be defined by a linear combination of the vectors in the basis. The coefficients of the linear combination are called coordinates, and must be uniquely defined. In the case of vector spaces, we have the property that all the bases of a space have the same number of elements ; this number is called the dimension of the space.
Similarly, a basis of a finite affine space is a set of points, such that, any point in the affine space can be defined as an affine combination of the points in the basis. The coefficients in the affine combination are the barycentric coordinates of the point and are unique. Intuitively, the point is the barycenter of the basis points with weights equal to the corresponding coordinate. Note that this intuitive property is not a characterisation, since the weights could all be multiplied by a same constant without changing the barycenter. The uniqueness comes from the fact that an affine combination forces the sum of the weights to be equal to 1. These weights are called the barycentric coordinates of the points.

Point and Vectors basis

Conformly to the definition of an affine space $\mathcal{A}$ associated with the vector space V :

$$\mathcal{A} = \{P' = P + v | P \text{ is a given point, } v \text{ in } V\}.$$

In this setting, a basis of the affine space is more naturally defined by the point P called the origin, and an arbitrary basis of vectors of V. The link with the previous definition of a basis is that for an affine basis of points $(A_i)_{0 \leq i \leq m}$ of A, the corresponding basis with one point and n vectors is the point A_0 and the vectors $(\vec{A_0 A_i})_{1 \leq i \leq n}$. It is actually equivalent to state that $(A_i)_{0 \leq i \leq n}$ is an affine basis of A or to state that $(\vec{A_0 A_i})_{1 \leq i \leq n}$ is a vector basis of V. From this equivalence, we can deduce that the number of points in any given affine basis is $n + 1$ where n is the dimension of the associated vector space. For an arbitrary point B and equation (1.4), we have :

$$B = \sum_{i=0}^{n} \alpha_i A_i \text{ with } \sum \alpha_i = 1$$

$$= \quad A_0 + \sum_{i=1}^{n} \alpha_i \vec{A_0 A_i}$$

So, the affine coordinates of B relative to the affine basis $(A_i)_{0 \leq i \leq n}$ are $(\alpha_0, \ldots, \alpha_n)$ and the cartesian coordinates of B relative to the cartesian basis $(A_0, \vec{A_0 A_1}, \ldots, \vec{A_0 A_n})$ are $(\alpha_1, \ldots, \alpha_n)$. From now on, a vector $\vec{AB}$ will simply be denoted AB.

Finding Barycentric Coordinates

Let us consider two distinct points A, and B, and a point G of the line (AB). The two points (A, B) forma basis of the line. We want to know the barycentric coordinates (α, β) of G relatively to A and B, that is, so that :

$$G = \alpha A + \beta B \text{ where } \alpha + \beta = 1.$$

In fact, one can verify with point arithmetic that the coordinates are defined by :

$$\begin{aligned} GB &= \alpha AB \\ AG &= \beta AB \end{aligned}$$

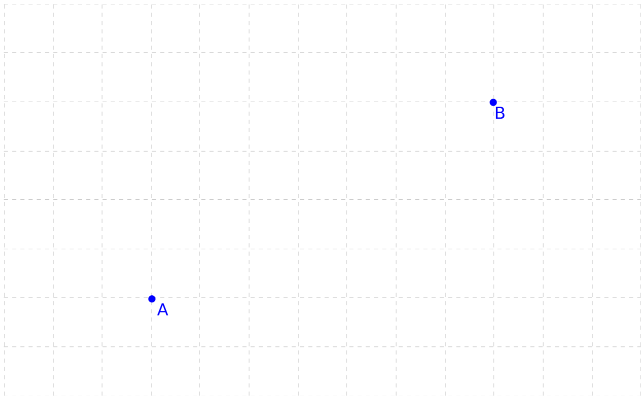
Be aware that, at first these relations are not so natural : if you consider the length of AB to be 1, α is the signed distance from G to B, and β is the signed distance from A to G. To get back the intuitiveness, it is important to think in terms of barycenter, as the weigth of a point increases, the barycenter gets closer to this point.
In the case of 3 points A, B abd C, being an affine basis of the plane, a point G has coordinates α, β and γ, where

— α is the signed area of the triangle ABM over the area of the triangle ABC,
— β is the signed area of the triangle BCM over the area of the triangle ABC,
— γ is the signed area of the triangle CAM over the area of the triangle ABC.

Examples and Exercises

Two arbitrary (distinct) points A and B define an affine basis. On the figure, draw the points

— $M1$ of affine coordinates $(1/2, 1/2)$,
— $M2$ of affine coordinates $(1/4, 3/4)$,
— $M1$ of affine coordinates $(-1, 1)$.



Now, draw two arbitrary points on the line (AB), one between A and B, one outside the segment $[AB]$. Find the barycentric coordinates of these points in the basis A, B.

1.0.4 Affine transformations

As linear transformations are the homomorphism associated with vector spaces, affine transformations are the homomorphism associated with affine spaces. A transformation is linear if the image of a linear combination of vectors is the linear combination of the images of these vectors. A transformation is affine if the image of an affine combination of points is the affine combination of the images of the points. Intuitively, that means that an affine transformation preserves barycenters :

$$f\left(\sum \alpha_i A_i\right) = \sum \alpha_i f(A_i) \text{ when } \sum \alpha_i = 1. \tag{1.5}$$

But in fact, you already know affine transformation. Since long, you have been told that an affine tranformation was of the form :

$$f(t) = at + b. \tag{1.6}$$

Indeed, the transformations given by equation (1.6) for arbitrary a and b are all the affine transformations from $\mathbb{R}$ to $\mathbb{R}$. You can easily check that the characteristic property (1.5) holds for such a function.

Now, let's consider the affine space of points in the Euclidian plane. You already know about a lot of transformations in this space :

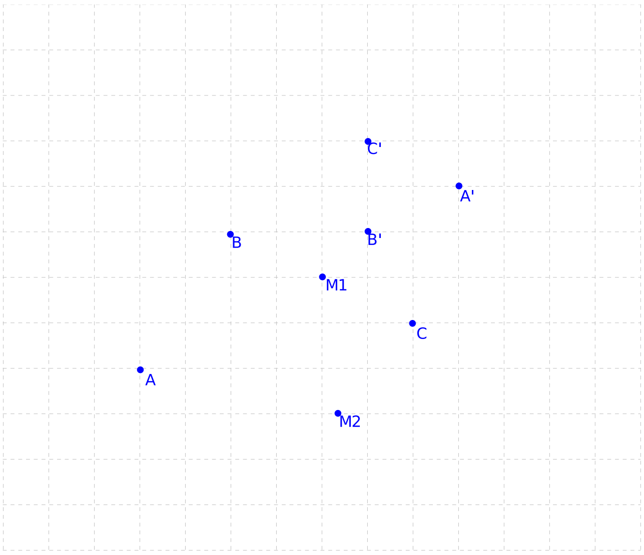
— translations,
— rotations,
— projection : orthogonal or parallel to an arbitrary direction,
— projection with a center of projection,
— homothety...

Are all these affine ? To check, we need to be sure that they preserve barycenters. Indeed, all of them are except for the projection with a center of projection. You can draw an example to show that barycenters are not preserved.

You know that, for a linear transformation to be defined, it is sufficient to know the image of a basis. That is, the images of n vectors completly define the linear transformation. This is also the case for affine transformation, it is enough to know the images of the $n+1$ points of a affine basis.

Exercise

The three points A, B and C define an affine basis of the plane, since they are in general position, that is, not aligned. A planar affine transformation T is characterized by three points A', B' and C' being the respective images of A, B and C. Draw the images M_1' and M_2' of the points M_1 and M_2 in the plane by the transformation T.


C'
A'
B
B'
M1
C
A
M2

Chapitre 2

PARAMETRIC CURVES and SURFACES : GENERALITIES

In this section we will define a parametric curve, and define particular parametric curves : interpolating curves, and approximating curves.

2.0.5 Definition

In the following, we shall call a **parametric curve** the geometric representation of a function, defined on a real interval I, whose values belong to an affine space of dimension 2 or 3.
A **parametric surface** will be defined similarly but on an interval of $\mathbb{R}^2$.

Curves and surfaces may also be defined implicitly, that is, given a so called potential function from $\mathbb{R}^2$ or $\mathbb{R}^3$ to $\mathbb{R}$, curves or surfaces may be defined as level sets of the potential function.

2.0.6 Different representations

There are different ways to define a curve. The most common consists in defining the curve either implicitly or explicitly. Example: An introductory example, the line Let us take the example of the line, and think about the different ways you know to define a line. A line is defined by two points A and B. One way to define the line (AB) is :

$$(AB) = \{M|\vec{AM}//\vec{AB}\} = \{M|\vec{AM}.\vec{n} = 0\}$$

where $\vec{n} = (-(y_B - y_A), x_B - x_A)$ is a vector normal to the line. This is called an implicit equation of the line : a point belongs to the line if its coordinates verify the equation.
Another way to define the line is to consider a parametric equation. The line (AB) is defined by :

$$(AB) = \{M|M = A + t\vec{AB}\}$$

At each parameter t is $\mathbb{R}$ corresponds a point on the line ; this is a parametric representation of the line (AB).
Implicit and parametric representation are different and may be convenient for different operations : for examples parametric representation are well adapted for defining points on a surfaces. It is not the case for implicit representation : finding point on the curve/surface is not easy, but given a point in space, checking if it is on the curve/surface is easy. Thus, implicit models are good for collision detection. Of course, in an ideal world, one would

have both representation of the same object, but note that it is not easy to go from implicit to parametric representation or conversely form parametric to implicit.

In the following, we will be mainly interested in parametric representation. Let us start with curves.

2.0.7 Parameterization

We shall actually use the term 'curve' for a function parameterizing a curve, which is indeed not correct since a curve can be parameterized by different functions. For example

$$(AB) = \{M | M = A + t\vec{AB}\} = \{M | M = A + t^2\vec{AB}\} = \{M | M = A + sin(t)\vec{AB}\}$$

is the same line. Think about the way the parameterization changes the line. In the following we will actually call a parametric curve the parametric function corresponding to the curve, even if this is abusive.
Parametric curves are defined by a real function with values in an affine space. That is :

$$\begin{array}{rcl} f : \mathbb{R} & \rightarrow & \mathcal{A} \\ t & \mapsto & f(t) \end{array}$$

such that $f(t)$ is a point. If we choose a referency, with point (origin) and vectors, then we can express the points by its so called Cartesian coordinates. We will always choose orthonormal vectors in the referency. Talking about orthonormal referency assumes that the underlying vector space has a scalar product. This is the case, in particular, for the euclidean plane. Then, each coordinate, is a function of the parameter t : A 2D point $f(t) = (x(t), y(t))$.
We saw that a same curve may have different parameterization. If you imagine that a parameterization is a trajectory, like the path of a flying insect. Representing the same curve means that the trajectory is the same, but the speed (first derivative) at which the insect flies maybe different, and in this case the parameterization is different. Does parameterization matter ? Indeed , yes. One property we want to insure for a parameterization is that it is regular.

Definition
A parameterization F of a curve is **regular** if, for all t, $F(t) \neq 0$.

Non regular parameterization are bad !
Exercice
Let us look at the following example : Let :

$$\begin{array}{rcl} f : \mathbb{R} & \rightarrow & \text{affine plane} \\ t & \mapsto & (t^2, -t^4) \text{ if } t \leq 0 \\ & & (t^2, t^2) \text{ otherwise} \end{array}$$

Draw the corresponding curve, and show that this function is C^1 everywhere, but the parameterization is non regular.

What we note, is that the parameterization may be C^1 everywhere, but the curve does not have a continuous tangent. That is, the smoothness of the function does not insure the smoothness of the curve. The parametric representation does not insure geometric property. This is because the parameterization is non regular. With a non regular parameterization you can have the inverse problem : a smooth curve, and a non C^1 parameterization.

Exercice
Find an example of a curve which is tangent continuous but does not have a C^1 parameterisation.
Having a regular parameterization insures that the smoothness of the function corresponds to the smoothness of the curve : thus, it is possible to study the smoothness of the curve, looking at the smoothness of the parameterization.

More generally, the properties of a curve independent of the parametrization are called **geometric properties**. The geometric continuity of the tangent of a curve is called G^1. Being G^1 is actually equivalent of having a uniform parametrization that is C^1.

Arc length parametrization and tangent

There is an ideal parametrization : the uniform parametrization. It corresponds to a constant speed (or constant first derivative), more precisely, a vector speed equal to 1. This parametrization is called in french "parametrisation en abscisse curviligne".
This parametrization is a good theoretical tool since it permits to express easily some quantities. In practice, the arc length parametrization is hard to get. Nevertheless, a good parametrization will be as uniform as possible.

The tangent of a curve is given by the derivative $f'(t)$. The derivative $f'(t)$ is the limit of $\frac{f(t+\epsilon)-f(t)}{\epsilon}$ when ϵ goes to 0. A difference of point is a vector, so the tangent is a vector.
The length of the curve between two points $f(t_0)$ and $f(t)$ on the curve is given by :

$$s(t) = \int_{t_0}^{t} \|f'(u)\| \, \mathrm{d}u.$$

If the parametrization is regular, $\|f'(u)\| = ds/dt$ is non zero, so we can do a change of variable between t and s. If we parameterize the curve by s instead of t (change of variable), we get the arc length parametrization, that we note here $f(s)$. The vector df/ds is the *unit* tangent vector (because $d(s^{-1})/du (= dt/ds) = 1/\|f'(u)\|$).

Osculating plane and curvature

We can now consider the second derivative of the curve, that we denote $f''(t)$. It measures the variation of the curve relative to the tangent direction.
If we consider $f(s)$, the arc length parametrization. Because $\|df/ds\| = (df/ds).(df/ds) = 1$ then $(df/ds).(d^2f/ds^2) = 0$ (by differentiation). We call *curvature* $\kappa(t) = \|d^2f/ds^2\|$. In terms of an arbitrary parametrization :

$$\kappa(t) = \frac{\|f'(t) \times f''(t)\|}{\|f'(t)\|^3}.$$

The radius of curvature is given by $R = \frac{1}{\kappa}$.
The first and second derivative span a plane, called the *osculating plane*, which is the plane approximating the curve the best.

Torsion

The variation of the curve to the osculating plane is measured by the torsion :

$$\tau(t) = \frac{det(f(t), f'(t), f''(t)))}{\|f'(t) \times f''(t)\|}.$$

In terms of s, the torsion can be expressed more easily by :

$$\tau(s) = \frac{det(f(s), df/ds(s), d^2f/ds^2)}{\kappa(s)^2}.$$

κ and τ are independent of the parametrization.

We want to be able to describe smooth curves intuitively. The first way is to give some points and to look for curve passing through, that is, to interpolate the points. The second way is to give a rough idea on how the curve should look like, that will be done by giving a control polygon, that is, a piecewise linear curve describing the shape we want to generate. The section on interpolation will address the first way of defining curves, the section on approximation the second way.

Chapitre 3

INTERPOLATION

3.0.8 Interpolating two points

Given are two points A and B, the simplest interpolating curves going through these two points is the line (AB). For example, the line l_{AB} cab be parameterised by :

$$l_{AB}(t) = (1-t)A + tB.$$

In this parameterisation, $l_{AB}(0) = A$ and $l_{AB}(1) = B$.
More generally, can we parameterise the line (AB) such that for two arbitrary parameters t_A and t_B, $l_{AB}(t_A) = A$ and $l_{AB}(t_B) = B$? Yes indeed. And the general expression is :

$$l_{AB}(t) = \frac{t_B - t}{t_B - t_A}A + \frac{t - t_A}{t_B - t_A}B,$$

or more intuitively written :

$$\begin{tikzcd}
A \arrow[dr, "\frac{t_B-t}{t_B-t_A}"'] & & B \arrow[dl, "\frac{t-t_A}{t_B-t_A}"] \\
& l_{AB}(t) &
\end{tikzcd}$$

Note that $l_{AB}(t)$ well defined : $l_{AB}(t)$ is an affine combination of the points A and B since the sum of the coefficients is 1. The point $l_{AB}(t)$ is actualy the barycenter of A and B with wiegth $\frac{t-t_A}{t_B-t_A}$ and $\frac{t_B-t}{t_B-t_A}$.

3.0.9 Interpolating three points

Let us now interpolate 3 points P_0, P_1 and P_2 at parameters t_0, t_1, and t_2, with a simple curve. What does 'simple' mean ? We can not interpolate 3 points in arbitrary position with a line. So, we have to take a curve with more degrees of liberty. In fact, this curve is going to be a polynomial of degree 2 (a line is a polynomial of degree 1).
We shall use the same idea that we used before. We can define the line l_{01} going through P_0 at t_0 and through P_1 at t_1 :

$$l_{01}(t) = \frac{t_1 - t}{t_1 - t_0}P_0 + \frac{t - t_0}{t_1 - t_0}P_1.$$

Similarly, the line l_{12} is passing through P_1 at t_1 and through P_2 at t_2 :

$$l_{12}(t) = \frac{t_2 - t}{t_2 - t_1}P_1 + \frac{t - t_0}{t_2 - t_1}P_2.$$

We now call the function $l_{012}(t)$:

$$l_{012}(t) = \frac{t_2 - t}{t_2 - t_0} l_{01}(t) + \frac{t - t_0}{t_2 - t_0} l_{12}(t).$$

$$\begin{tikzcd}
l_{01}(t) \arrow[dr, "\frac{t_2-t}{t_2-t_1}"] & & l_{12}(t) \arrow[dl, "\frac{t-t_0}{t_2-t_0}"'] \\
& l_{012}(t) &
\end{tikzcd}$$

We do have (you can check as an exercise)

— $l_{012}(t_0) = P_0$
— $l_{012}(t_1) = P_1$
— $l_{012}(t_2) = P_2$

Also, the function $l_{012}(t)$ has degree 2 since $l_{01}(t)$ and $l_{12}(t)$ have degree 1 and their coefficients have degree 1 also.

3.0.10 General setting : Interpolating $n + 1$ points

We now want to interpolate P_0, P_1... P_n at the parameter values t_0, t_1... t_n respectively. We can use the same idea as for 3 points recursively. Suppose you already have the functions $l_{01\dots n-1}$ and $l_{12\dots n}$. Then :

$$l_{01\dots n}(t) = \frac{t_n - t}{t_n - t_0} l_{01\dots n-1}(t) + \frac{t - t_0}{t_n - t_1} l_{12\dots n}(t).$$

This can be written as a dynamic programming algorithm, called Neville's algorithm :

$$\begin{tikzcd}
P_0 \arrow[dr] & & P_1 \arrow[dl] \arrow[dr] & & P_2 \arrow[dl] & \dots & P_n \arrow[dl] \\
& l_{01}(t) \arrow[dr] & & l_{12}(t) \arrow[dl] & & {} & \\
& & l_{012}(t) & & & & \\
& & {} \arrow[dr] & \vdots & {} \arrow[dl] & & \\
& & & l_{01\dots n}(t) & & &
\end{tikzcd}$$

Remark : the basis functions associated with interpolation are the Lagrange polynomials. The more the degree increases, the more the Lagrange basis function oscillate. For this reasons, a interpolating curve for high degree does not look nice (it also oscillate), and it most of the cases, approximations will be preferred.

3.0.11 Further reading and related topics

You will get more if you follow Philippe Berger's class, or read his class notes. You may want to know more on interpolation. For that, try to answer the following questions or look for further references using the following keywords.

You may remember Lagrange polynomials. They form indeed a alternative basis (to the monomial basis) of the space of polynomials of degree less than or equal to n. What is their link to Neville algorithm ? quick answer : they are the corresponding basis functions.

You may want to interpolate more than just value. For example, derivatives -that is, tangent- for parametric curves. Look into Hermite interpolation.

Chapitre 4

BÉZIER CURVES

4.0.12 Motivation

Bézier curves are 'just' polynomial curves. The canonical basis for polynomials is the monomial basis. The Bézier curve is prefered against the monomial basis because the geometrical interpretattion makes more sense.
In the monomial basis, the successive coefficients represent tthe value of the polynomial at 0 and the successive derivatives at zero, so the information is very localized, and moreover difficult to interpret after the third coeffcient...
Bézier curve are an expression of a polynomial curve, where the coefficient correspond to so-called control points that characterize the shape of the curve.

Bézier curves

Now, we do not want to interpolate the points, but to follow the shape of the control polygons. Our starting point(s), or input, is a family of so called control points $(P_0, P_1, \dots, P_n)$ defining a control polygon. We want to find a curve/function, defined by these points.
There are many ways Bézier curves could be introduced. We shall present two here.

Analytic definition of a Bézier curve

First, we are looking for in a set of points defined relatively to our input, the control points. That is, the function P can be written :

$$P(t) = \sum_{k=0}^{n} c_k(t) P_k \text{ where } \sum_{i=0}^{n} c_k(t) = 1.$$

Note, that the sum of the parameters has to be 1 for all t, so that, for all t, $P(t)$ is a well defined point. If we want the function $P(t)$ to follow the shape of the control polygon (created by joining the points P_k), we need to have 'bell-shaped' function. Such functions are called unimodal. Here, we need to have unimodal function so that the weight given to a control point, that is, its influence, will increase, and then decrease. A good candidate are the Bernstein polynomials B_k^n on the interval $[0, 1]$. The Bernstein polynomial

$$B_k^n(t) = \tbinom{n}{k}(1-t)^{n-i}t^i.$$

Then, the Bézier curve defined by the points $(P_0, P_1, \dots P_n)$ is

$$P(t) = \sum_{i=0}^{n} B_k^n(t) P_k. \tag{4.1}$$

You can check that $\sum B_k^n(t) \equiv 1$, so the wiegthed sum of affine points in 4.1 is well defined.

Properties of a Bézier curve

From the expression of the Bernstein polynomials, you can derive many properties of Bézier curves :

1. A Bézier curve interpolates the first and the last control points
2. A Bézier curve is tangent to the first and last segments of the control polygon
3. Symmetry : The Bézier curve defined by the control points $(P_0, P_1, \dots, P_n)$ is similar to the Bézier curve defined by the points $(P_n, P_{n-1}, \dots, P_0)$.
4. The Bézier curve is in the convex hull of the control points.
5. Affine invariance : the image of a Bézier curve is the Bézier curve of the image of its control points.

To prove the first property, you should evaluate Bernstein polynomials at 0 and 1. At 0, for example, all the Bernstein polynomial vanish, except the first one, that means $P(0) = P_0$. For the second one, similarly, you can compute the derivative of a Bézier curve at 0 and 1. Considering only two terms is sufficient. You will find $P'(0) = n(P_1 - P_0)$. More generally, the derivetive of a Bézier curve is given by :

$$P'(t) = n \sum_{k=0}^{n-1} (P_{k+1} - P_k) B_k^{n-1}(t).$$

Recursively, the control points of the ith derivative $P^{(i)}$ are the ith discrete difference of the control points of P, up the a factor $n \dots (n-k)$.
To prove the third property, note that

$$B_k^n(t) = B_{n-k}^n(1-t),$$

and on the interval $[0, 1]$, $P(t)$ and $P(1-t)$ describe the same curve (with a different parameterisation though).
Bézier curves also have the variation diminishing property, which says that a curve does not oscillate more than its control polygon, or any line crosses a Bézier curve at most the number of time it crosses the control polygon. The variatio diminishing property can be proved using the subdivision algorithm.

Approximation of a Bézier curve

1 We have seen the convex hull property, saying that a Bézier curve lies in the **convex hull** of its control polygon. The convex hull takes $0(n \log n)$ time to compute, when n (or here $n+1$) is the number of points.

2 As a coarser approximation of the Bézier curve, one can take the **axis-aligned boundnig box**, which is much easier to compute (linear time) : the Bézier curve lies in the box $[x_min, x_max]x[y_min, y_max]$ where x_min and x_max are respectively the minimum and the maximum abscissae of the control points, and same for the ordinates.

3 Let us consider the line L joining the first and the last control points, such that, $L(0) = P_0$ and $L(1) = P_n$. If we want to approximate the Bézier curve by this line, the maximum error is bounded by the distance of the control points to this line. Wang has proved that the distance is bounded by :

$$|P(t) - L(t)| \leq \frac{n(n-1)}{8} ||\Delta_2 P||_\infty$$

where $||\Delta_2 P||_\infty = \max_{k=0\ldots n-2} |P_{k+2} - 2P_{k+1} + P_k|$. Note that these are the second difference of the control points, that is the control points of the second derivative of P.

4 Nairn, Peters and Lutterkort in 1999, and Reif in 2001, derived tight bounds for the distance between a Bézier curve and its control polygon. First, we parameterize the control polygon on the interval $[0, 1]$: we consider the piecewise linear function l such that the control point P_k is the image of $\frac{k}{n}$. In the fonctionnal case :

$$\forall t \in [0, 1], \;\; |P(t) - l(t)| \leq N_\infty(d) ||\Delta_2 P||_\infty,$$

where $N_\infty(d) = \frac{d}{8} - \frac{\omega}{8d}$ with $\omega = 0$ if d is even, 1 if d is odd. Thus, for the parametric case, we have :

$$\forall t \in [0, 1], \;\; ||P(t) - l(t)||_p \leq N_\infty(d) ||\Delta_2 P||_{p,\infty}.$$

Algorithmic definition of a Bézier curve : the de Casteljau algorithm

A second way to introduce Bézier curves, is to start with Neville's algorithm. But, this time, instead of interpolating with as many parameters as points, we always use the parameters 0 and 1.

This new algorithm is called the de Casteljau algorithm, and is also a dynamic programming algorithm. This algorithm also corresponds to the geometric construction of a point on the curve.

$$\begin{tikzcd}
P_0 \arrow[dr, "1-t"] & & P_1 \arrow[dl, "t"'] \arrow[dr, "1-t"] & & P_2 \arrow[dl, "t"'] & \dots & P_n \arrow[dl, "t"'] \\
& * \arrow[dr, "1-t"] & & * \arrow[dl, "t"'] & & {} & \\
& & * & & & & \\
& & {} \arrow[dr, "1-t"] & \vdots & {} \arrow[dl, "t"'] & & \\
& & & P(t) & & &
\end{tikzcd}$$

As the Neville algorithm, the de Casteljau is an evaluation algorithm, that is, these algorithms find the value of the function for a given parameter t from the control points $(P_i)_{i=0}^n$.

Note that, from a complexity point of view, de Casteljau algorithm is a dynamic programming algorithm that runs in $O(n^2)$ time. – Warning : do not implement it recursively otherwise you'll get a $O(2^n)$ algoritm ! ! ! ! ! – Moreover, this algorithm can be implemented in place, that is, all the intermediate computations can be stored in the initial array of control points.

Justification : note that a path going from the control point P_k to the bottom of the algorithm, that is, $P(t)$, is going $n-k$ times to the right, and k times to the left. Moreover there are $\binom{n}{k}$ different paths to go from P_k to $P(t)$ (among going n steps, choose the k times when you go left). So the multiplying factor of P_k is $\binom{n}{k}(1-t)^{n-k}t^k = B_k^n(t)$.
Remark If you remplace P_k by 1, and all the other points by 0, you obtain $B_k^n(t)$ at the botom of the agorithm. Just above, you indeed have B_k^{n-1} and B_{k-1}^{n-1}, so it prooves the following recurrence relation :

$$B_k^n(t) = (1-t) * B_k^{n-1}(t) + t * B_{k-1}^{n-1}(t).$$

The de Casteljau algorithm will also be used for subdivision.

Blossoms
A well adapted tool for studying Bézier curves are blossoms. Blossoms were introduced by Lyle Ramshaw in 1985 -since Ramshaw has presented another tool, equivalent to blossoms, but more general : multiplying points.

Let us call $\mathcal{P}_n$ the set of polynomials of degree less than or equal to n, and $\mathcal{F}_n$ the set of functions with n variables multi-affine (affine on each variable) and symmetric. Both these spaces are vector space of dimension $n+1$ and are isomorphic.

Let $f \in \mathcal{F}_\backslash$, then for $x_1, \ldots, x_n$ in $\mathbb{R}$

- $f\left((1-\lambda)u + \lambda v, x_2, \ldots, x_n)\right) = (1-\lambda)f(u, x_2, \ldots, x_n) + \lambda f(v, x_2, \ldots, x_n)$, that is, f is affine (of degree one) on the first variable,
- $f(x_{\sigma(1)=,\ldots,x_\sigma(n)}) = f(x_1, \ldots, x_n)$ for an arbitrary permutation σ.

The relation that identifies a blossom p with the corresponding polynomial P is the diagonal property, that is :

$$p(t, \ldots, t) = P(t).$$

As an exercise, try to find the blossom of X, X^2 and X^4.

Property
Expression of the Bézier control points in terms of the blossom. If P is a polynomial of degree less than or equal to n, $(P_0, \ldots, P_n)$ are its Bézier control points and p denotes its blossom, then :

$$P_i = p(\underbrace{0, \ldots, 0}_{n-i \text{ times}}, \underbrace{1, \ldots, 1}_{i \text{ times}}).$$

We can re-interpret the de Casteljau algorithm in terms of blossoming, for example for curves of degree 3 :

$$\begin{tikzcd}
p(000) \arrow[dr, "1-t"] & & p(001) \arrow[dl, "t"'] \arrow[dr, "1-t"] & & p(011) \arrow[dl, "t"'] \arrow[dr, "1-t"] & & p(111) \arrow[dl, "t"'] \\
& p(00t) \arrow[dr, "1-t"] & & p(01t) \arrow[dl, "t"'] \arrow[dr, "1-t"] & & p(11t) \arrow[dl, "t"'] & \\
& & p(0tt) \arrow[dr, "1-t"] & & p(1tt) \arrow[dl, "t"'] & & \\
& & & P(t) = p(ttt) & & &
\end{tikzcd}$$

The values of the blossoms appearing inside the de Casteljau algorithm are defined by simply applying the multiaffinity and the symmetry property of the blossom. The fact that the Bézier points are the (uniquely defined) coefficients of the polynomial P in the Bernstein basis justifies the expression of the control points on term of the blossom.

Remark A Bézier curve can indded be defined on an arbitrary interval $[a, b]$ by an affine change of variable, that is :

$$P(u) = \sum_{i=0}^{n} P_i B_i^n \left(\frac{u-a}{b-a} \right)$$

in which case $p_i = p(\underbrace{a, \dots, a}_{n-i \text{ times}}, \underbrace{b, \dots, b}_{i \text{ times}})$.

The notion of blossom will be particularly usefull for expressing B-splines and their control points. But let's first start with subdivision.

Subdivision

The idea in subdivision is to aplit a Bézier curve into two parts. A Bézier curve $P(t)$ has control points $(P_0, P_1, \dots, P_n)$ on the interval $[0, 1]$. Now let us consider another curve Q such that $Q(t) = P(t/2)$. Well, the function Q and P actually parameterize the same curve, except that the speed at which the curve is drawn is twice less for Q than for P. In particular $Q(1) = P(1/2)$. But Q is a polynomial curve, and must have control points $(Q_0, Q_1, \dots, Q_n)$ for the interval $[0, 1]$, that is, there must be control points for P for the interval $[0, 1/2]$. Indeed, we even computed these points in the de Casteljau algorithm, they are the points on the left of the diagram. Similarly, the control points for the curve P on the interval $[1/2, 1]$ are the points $(R_0, R_1, \dots, R_n)$ appearing on the right of the diagram.

$$\begin{tikzcd}
Q_0 = P_0 \arrow[dr, "\frac{1}{2}"] & & P_1 \arrow[dl, "\frac{1}{2}"'] \arrow[dr, "\frac{1}{2}"] & & P_2 \arrow[dl, "\frac{1}{2}"'] & \dots & P_n = R_n \arrow[dl] \\
& Q_1 \arrow[dr, "\frac{1}{2}"] & & * \arrow[dl, "\frac{1}{2}"'] & & R_{n-1} & \\
& & Q_2 & & & & \\
& & & \vdots & & & \\
& & {} \arrow[dr, "\frac{1}{2}"] & & {} \arrow[dl, "\frac{1}{2}"'] & & \\
& & & Q_n = R_0 & & &
\end{tikzcd}$$

So, from the de Casteljau algorithm, one can find two families of control points, giving a better approximation of the Bézier curve. We can iterate this process. The control points for the left part of the curve can be subdivided again, and the ones from the right part of the curve too, so we shall have now 4 families of control points approximating the Bézier curve even better.

Proposition The sequence of control polygons generated by iterative subdivision converges to the Bézier curve defined by the initial control polygon.

(idea of) proof You can show that the distance between any two consecutive control points is divided by 2 at each subdivision step. Then, we know that some of these control points are on the curve -namely the first and last point of each family-. This is sufficient to prove the proposition.

Application : Variation diminishing property A Bézier curve never oscillates more than its control polygon, that is, The number of intersection of an arbitrary line with a Bézier curve is at most the number of intersections of this line with the control polygon of the curve.

(idea of) proof Subdivision is a sequence of corner cutting operations. One can show that corner cutting is variation diminishing.

Approximation and subdivision Let P be a Bézier curve, and Q and R the left and right polynomial after a subdivision at 1/2. Then, the second differences of Q, $||\Delta_2 Q||_\infty = \max_{k=0\ldots n-2} |Q_{k+2} - 2Q_{k+1} + Q_k|$ are $\frac{||\Delta_2 P||_\infty}{4}$, and $||\Delta_2 R||_\infty$ are $\frac{||\Delta_2 P||_\infty}{4}$. Thus, if we consider the 2^i control polygons generated by i subdivision steps. It can be proved that after each subdivision, the error of the approximation by the piecewise linear function through the first and last control point (Wang), and the approximation through the control polygons are both divided by 4.
Remark : The number of subdivision steps to get an ϵ approximation of a Bézier curve can be computed in advance. **Application : an intersection algorithm for Bézier curves** The diverse approxiation techniques can be used to propose a recursive intersection algorithm for Bézier curves. The base case consists in filtering using the approximation techniques. Since we have bounds, filtering can be done with garantees. The recursive case, consists in subdividing and looking at intersection on the subdivided parts.

Tensor product Bézier surfaces
The easiest way to generate a surface from a curve is to use tensor product (or, as signal processing people call it : 'separated vairables'). A parametric surface is givan by a two variable function, so :

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

I general, we choose the same degree in each variable $m = n$.
Interpretation : rewrite $S(u, v) = \sum_{i=0}^{n} P_i(v) B_i^n(v)$. where $P_i(v) = \sum_{j=0}^{m} P_{i,j} B_j^m(v)$. There

are $n+1$ Bézier curves in v, and for a given v, $P_i(v)$ is a control point on this curve. These $n+1$ points are control points of a Bézier curve in u.
Note that we could have instead consider $n+1$ curves in u.

There are specific algorithms for surfaces (de Casteljau, subdivision), but in terms of programming, the easiest is to consider applying the algorithm on one direction, and then on the other.

Chapitre 5

B-SPLINE CURVES

B-splines

One major drawback of Bézier curves is that the Bernstein polynomials are globally supported. That means, they are never zero. In fact, just because they are polynomials, they can never vanish on an non empty open interval. Only at a finite number of points, namely n, can a polynomial of degree n vanish. For our Bézier curve, it means that the weight corresponding to a point is almost never zero. That is, these points influence the whole curve. So, when the user moves only one control point, the whole curve needs to be recomputed. In order to find weight functions that have local support (will vanish outside a given interval), we use piecewise polynomials, called splines. The associated basis is the B-spline basis. Each spline (piecewise polynomial) can be expressed in a B-spline basis. In the case of the B-spline basis functions, we won't give their explicit expression since it is complicated, and therefore useless in practice. For general splines, we shall use the de Boor algorithm for evaluating a point on the spline. For uniform B-splines have a nice, simple and easy to implement subdivision algorithm (at least for the points not at the boundary of the curve).

Introduction B-splines have a particular property : minimizing the support of a B-splines function is almost enough to completly characterize the spline (in fact, up to a constant factor).
Exercise : Find the function of minimal support considering (a) piecewise linear splines, and then (b) degree 2 spline with connections parameters (so called *knots*) at the integers.

General B-splines

To define a B-spline space, one should first choose a degree n and a knot vector. The degree corresponds to the degree of the polynomial parts of the curve. The degree is always the same for the different part of the curve (that is, we do not stick together polynomials of different degrees). The knot vector correponds to the parameters where we allow some discontinuties.
If you stick together two polynomial curves of degree n : for the connection to be C^0 the curves simply have to be connected. To be C^1, they have to have the same tangent, to be C^2 the same curvature, and so on. If you force the connection to be C^n, you actually force the polynomial to be exactly the same. To convince yourself, think about it in terms of degree of freedom.

A **polynomial spline** is a piecewise polynomial function. At the parameter values where the polynomial pieces meet, a chosen continuity, up to C^{n-1} is fixed.

Given a chosen degree n, an increasing sequence of parameter values called the *knot vector* $V = (\dots, t_i, t_{i+1}, \dots)$ defines the continuity conditions. If a knot t appears μ times within the knot vector V, then the continuity of a spline defined by n and V at t is $C^{n-\mu}$. Note that a knot may not appear more than n times in V.

Indeed, the family of splines defined by n and V is spanned by a B-spline basis $(N_k^n(t))_k$ is characterized being of minimal support and form a partition of unity, that is :

— the function $N_k^n(t)$ has support $[t_k, t_{k+n+1}]$,

—

$$\sum_k N_k^n(t) \equiv 1.$$

Being of minimal support means that no spline of the same family, that is, defined by n and V has a smaller support than the basis functions.

The basis functions follow the recurrence relation on n :

$$N_k^0(t) = 1_{[t_k, t_{k+1}]}$$

$$N_{k,n}(t) = \frac{t - t_k}{t_{k+n} - t_k} N_{k,n-1}(t) + \frac{t_{k+n+1} - t}{t_{k+n+1} - t_{k+1}} N_{k+1,n-1}(t)$$

Computing the basis functions can be done using the following scheme (which leads to a dynamic programming algorithm) [**?**] p.384.
Note that, if $t \in [t_i, t_{i+1}]$ there are $n+1$ non vanishing basis functions of degree n :

$$N_{i-n}^n, \dots, N_i^n.$$

The values $N_{i-n}^n(t), \dots, N_i^n(t)$ depend on the knots $\{t_{i-n+1}, \dots, t_i, t_{i+1}, \dots, t_{i+n}\}$.
In terms of the blossom, the control points are given by evaluating the blossom at n consecutive values of the knot vector. That is :

$$P_k = s(t_{k+1}, \dots, t_{k+n+1})$$

and $S(t) = \sum_k P_k N_k^n(t)$.
Then, an evaluation algorithm similar to de Casteljau, computes the value of the spline function from the control points. As in de Casteljau, the number of levels is equal to the degree which is also the number of arguments in the blossom. This is the de Boor algorithm.

Exercise : Find the expression of the minimally supported degree 2 spline with knot vector $\mathbb{Z}$ using the de Boor algorithm.

One way to find more control points is to simply insert a knot, this is a knot insertion algorithm. We will see knot insertion/subdivision in the particular setting of uniform B-splines.

Uniform B-splines and subdivision Splines are called uniform when the knot vector is $\mathbb{Z}$.

In that case (a more general case has been developped recently [Cashman 2009]), subdividing consists in going from the knot vector $\mathbb{Z}$ to the knot vector $\mathbb{Z}/2$.
We can use the blossom to find the coefficients of subdivision : we see how to find the new control points, that is, the ones associated with $\mathbb{Z}/2$ as a function of the old control points (the ones on $\mathbb{Z}$).

Let's do it for degree 2 (for degree 1 it is possible, but a little strange since the blossom only has 1 parameter). We have a set of original control points $P_i^2 = s(i+1, i+2)$ and we look for the set of new control points $\tilde{P}_i^2 = s(i+\frac{1}{2}, i+1)$ or $\tilde{P}^2_{i+\frac{1}{2}} = s(i+1, i+1+\frac{1}{2})$.
We find (1 step with blossoming) :

$$\tilde{P}_i^2 = \frac{3}{4}P_{i-1}^2 + \frac{1}{4}P_i^2,$$

$$\tilde{P}^2_{i+\frac{1}{2}} = \frac{1}{4}P_{i-1}^2 + \frac{3}{4}P_i^2.$$

We can do the same for degree 3 :
We have a set of original control points $P_i^3 = s(i+1, i+2, i+3)$ and we look for the set of new control points $\tilde{P}_i^3 = s(i+\frac{1}{2}, i+1, i+\frac{3}{2})$ or $\tilde{P}^3_{i+\frac{1}{2}} = s(i+1, i+1+\frac{1}{2}, i+2)$.
We find :

$$\tilde{P}_i^3 = \frac{1}{8}P_{i-2}^3 + \frac{6}{8}P_{i-1}^3 + \frac{1}{8}P_i^3,$$

$$\tilde{P}^3_{i+\frac{1}{2}} = \frac{1}{2}P_{i-1}^3 + \frac{1}{2}P_i^3.$$

You can also compute the points P_i^1 for $i \in \mathbb{Z}$ and the points $\tilde{P}_i^1$ for $i \in \mathbb{Z}/2$:

$$\tilde{P}_i^1 = P_{i-1}^1 + P_i^1$$

$$\tilde{P}^1_{i+\frac{1}{2}} = P_i^1$$

In fact, there is a more general way to derive the new points from the old ones. You can note that, the points for degree 3 are indeed the midpoint of the points of degree 2, and same between degree 2 and 1, that is, for $i \in \mathbb{Z}/2$:

$$\tilde{P}_i^3 = \frac{1}{2}\tilde{P}^2_{i-\frac{1}{2}} + \frac{1}{2}\tilde{P}_i^2,$$

$$\tilde{P}_i^2 = \frac{1}{2}\tilde{P}^1_{i-\frac{1}{2}} + \frac{1}{2}\tilde{P}_i^1.$$

We can even push this relation by considering degree 0 splines. For degree 0 (piecewise constant functions), on the interval $[t_k, t_k + 1]$ the function is equal to P_k. So :

$$\tilde{P}_i^0 = P_i^0,$$

$$\tilde{P}^0_{i+\frac{1}{2}} = P_i^0.$$

So we have also :

$$\tilde{P}_i^1 = \frac{1}{2}\tilde{P}^0_{i-\frac{1}{2}} + \frac{1}{2}\tilde{P}_i^0.$$

The general case also holds :

$$\tilde{P}_i^{n+1} = \frac{1}{2}\tilde{P}^n_{i-\frac{1}{2}} + \frac{1}{2}\tilde{P}_i^n.$$

So, we can derive a very simple subdivision algorithm for uniform splines in two steps.

(a) duplicate the control points (subdivision of degree 0),
Do n x (b) Take the midpoint of two consecutive points in the control polygon.

$$(a)\ \ (\ldots, P_{-1}, P_0, P_1 \ldots) \longrightarrow (\ldots, P_{-1}, P_{-1}, P_0, P_0, P_1, P_1 \ldots)$$

$$\begin{tikzcd}
(b)\ldots & P_0 \arrow[dl, "\frac{1}{2}"'] \arrow[dr, "\frac{1}{2}"] & & P_0 \arrow[dl, "\frac{1}{2}"'] \arrow[dr, "\frac{1}{2}"] & & P_1 \arrow[dl, "\frac{1}{2}"'] \arrow[dr, "\frac{1}{2}"] & \ldots \\
\tilde{P}^1_0 \arrow[dr, "\frac{1}{2}"] & & \tilde{P}^1_{\frac{1}{2}} \arrow[dl, "\frac{1}{2}"'] \arrow[dr, "\frac{1}{2}"] & & \tilde{P}^1_1 \arrow[dl, "\frac{1}{2}"'] \arrow[dr, "\frac{1}{2}"] & & {} \arrow[dl, "\frac{1}{2}"'] \\
& \tilde{P}^2_0 & & \tilde{P}^2_{\frac{1}{2}} & & \tilde{P}^2_1 &
\end{tikzcd}$$

The second step, that will be repeated n times, where n is the degree of the spline, consists on taking for each two consecutive points, the mid-point of these two points.
One step of subdivision is then done by doing first linear subdivision, and then n times averaging. As in the Bézier case, subdivision maybe be iterated to get better and better approximation of the curve.
The splines keep properties very similar to Bézier curves : they follow the shape of the control polygon, stay inside the convex hull of the control points, and have the variation diminishing property.

Bibliography

To find some material about blossoming (floraison en français)
Pyramid Algorithms : A Dynamic Programming Approach to Curves and Surfaces for Geometric Modeling by Ron Goldman

Subdivision Methods for Geometric Design : A Constructive Approach by Joe Warren, Henrik Weimer

Same topic, but the presentation is not so close to the class, see books of CAGD by G. Farin, or Curves and Surfaces in Geometric Modeling... Jean Gallier.
all these boooks are at the library.