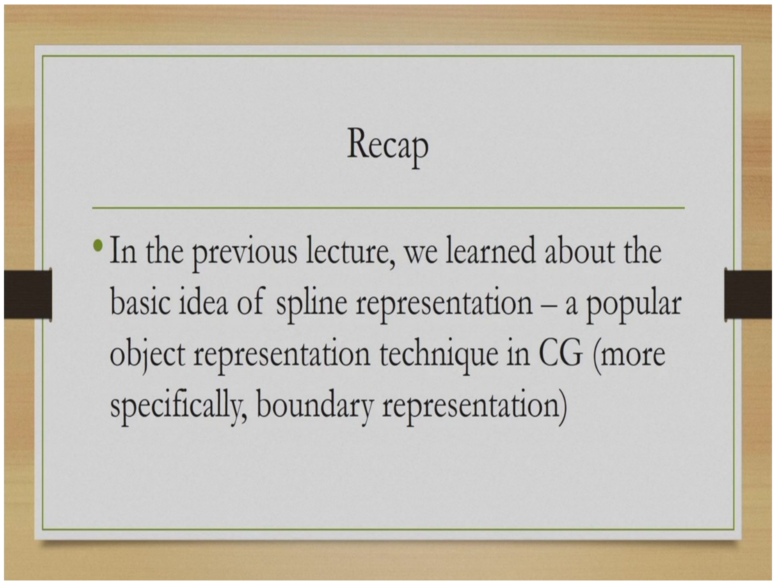
Computer Graphics
Professor. Doctor. Samit Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati
Lecture No. 08
Spline representation – II

Hello and welcome to lecture number 8 in the course Computer Graphics.

(Refer Slide Time: 0:37)



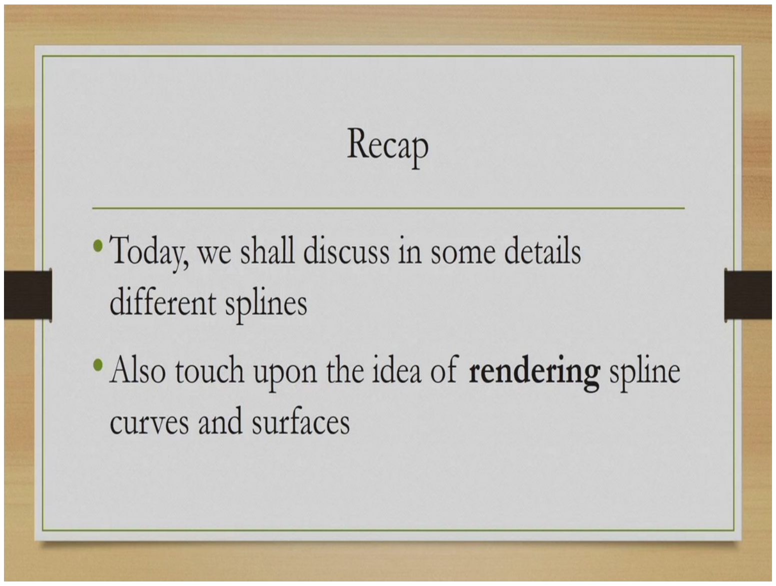
Recap

- In the previous lecture, we learned about the basic idea of spline representation – a popular object representation technique in CG (more specifically, boundary representation)

In the previous lecture we got introduced to the basic idea of splines. Now splines is one of the areas boundary representation techniques, as we have mentioned. And in spline in the introductory lecture, what we have learned, we learned about the basic idea, what is a spline then how to represent it and how those representations are created and various such introductory topics, including the types of splines.

Just to recap, let me briefly tell again about the basic idea of spline, so when we are talking of a spline, it is essentially a representation of a curve and the curve is represented in the form of collection of lower degree polynomials, so these polynomials joined together to give the overall curve and the polynomials are interpolated on the basis of a set of control points that are defined to represent the curves. So, Spline is essentially a representation where we join together multiple polynomial curves of lower order.

(Refer Slide Time: 2:11)



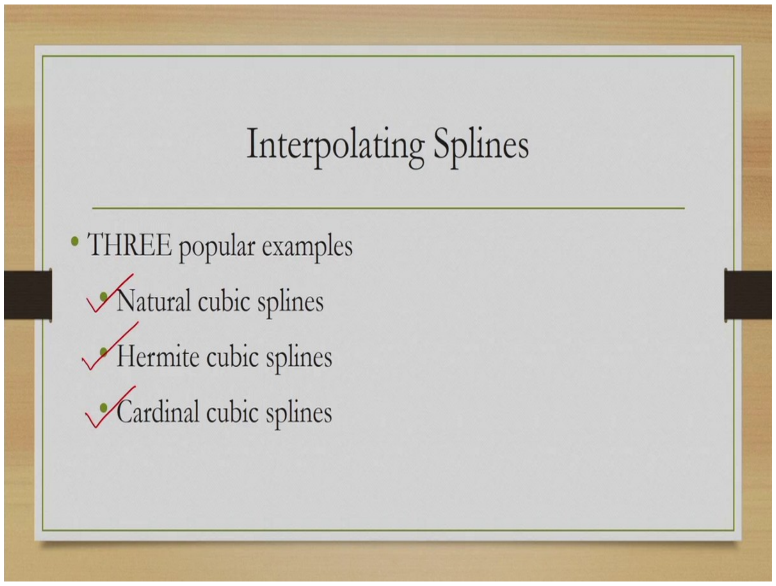
Recap

- Today, we shall discuss in some details different splines
- Also touch upon the idea of **rendering** spline curves and surfaces

Now, that was the basic idea, and during that discussion, we also talked about types of splines broadly, two types are there. One is interpolating splines. One is approximating splines. Today we are going to discuss in detail about these two types of splines. Along with that, we are going to talk about how these splines can be used to represent curves and surfaces in computer graphics. Let us start with the idea of interpolating splines.

If you recall, interpolating splines are those where we define a set of control points and the Spline curve passes through those points. And we also mention 3 popular interpolating splines.

(Refer Slide Time: 3:07)

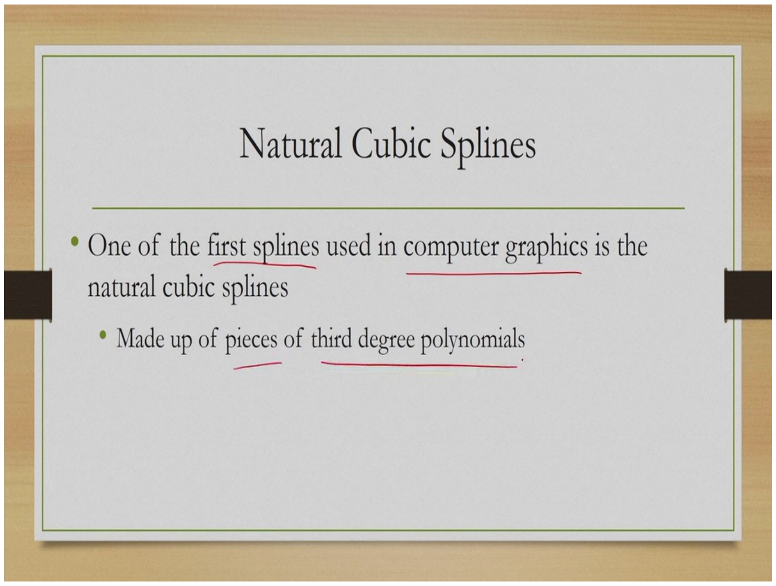


Interpolating Splines

- THREE popular examples
  - ✓ Natural cubic splines
  - ✓ Hermite cubic splines
  - ✓ Cardinal cubic splines

Namely the Natural cubic spline, the Hermite cubic splines and the Cardinal cubic splines, all 3 are interpolating splines, meaning that they are defined by control points which passes through the Spline curves. So, let us now, go into some detailed discussion on each of these popular types. We will start with natural cubic spline.

(Refer Slide Time: 3:44)

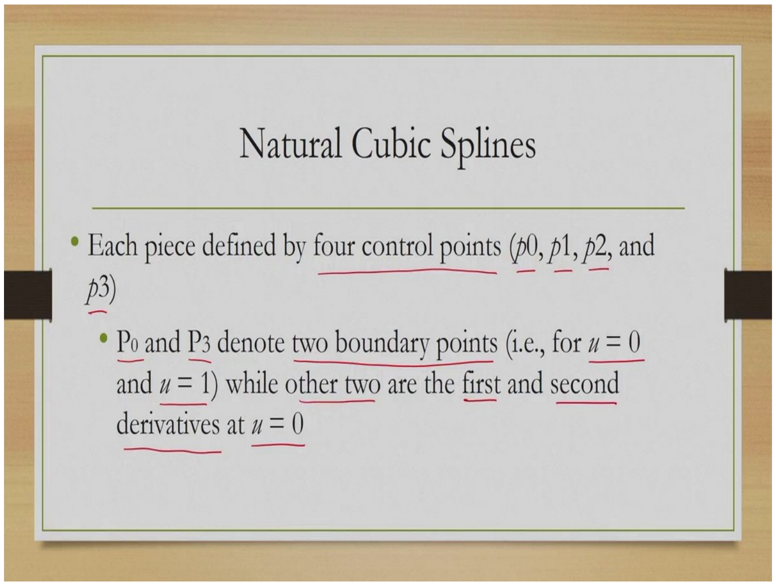


Natural Cubic Splines

- One of the first splines used in computer graphics is the natural cubic splines
  - Made up of pieces of third degree polynomials

Just a little bit of history. So this is one of the first splines. That was used in computer graphics applications. And as the name suggests, since it is a cubic spline so it is made up of pieces of third degree polynomials.

(Refer Slide Time: 4:09)



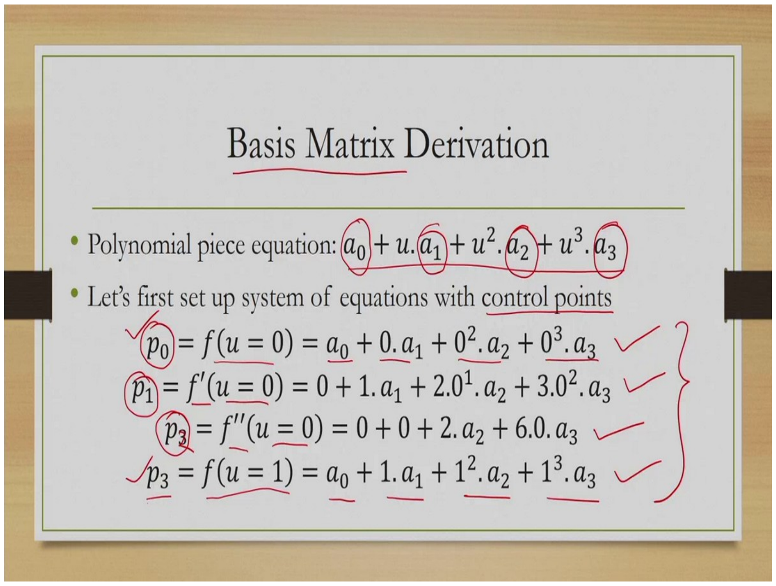
Natural Cubic Splines

- Each piece defined by four control points (*p*0, *p*1, *p*2, and *p*3)
  - P_0 and P_3 denote two boundary points (i.e., for $u = 0$ and $u = 1$) while other two are the first and second derivatives at $u = 0$

Now, each piece, each of the polynomial pieces is defined by 4 control points, we can denote them as p0, p1, p2, p3. Recollect that when we are talking of polynomial interpolation, what we mentioned earlier is that if we are using a polynomial of degree n, then we are going to have n plus 1 control points. So, similarly here since the degree is three, so we have 4 control points.

Now in the case of natural cubic splines two of the control points p0 and p3 denote two boundary points that means the boundary value of the parameter u so p0 refers to the point when u = 0 and p3 refers to the point where u = 1. Now, the other two control points are essentially denoting the first and second derivatives at the starting point that is equal to 0. So, two control points are essentially points on the curves p0 and p3, which represents the two boundary points and the other two points are not points on the Curves, rather the first and second order derivatives with respect to the parameter u at u equal to 0.

(Refer Slide Time: 5:47)



Basis Matrix Derivation

- Polynomial piece equation: $a_0 + u.a_1 + u^2.a_2 + u^3.a_3$
- Let's first set up system of equations with control points

$$p_0 = f(u=0) = a_0 + 0.a_1 + 0^2.a_2 + 0^3.a_3$$
$$p_1 = f'(u=0) = 0 + 1.a_1 + 2.0^1.a_2 + 3.0^2.a_3$$
$$p_3 = f''(u=0) = 0 + 0 + 2.a_2 + 6.0.a_3$$
$$p_3 = f(u=1) = a_0 + 1.a_1 + 1^2.a_2 + 1^3.a_3$$

Now let us try to represent this spline using one of the representation techniques. So, recollect again that spline can be represented in either of the two ways, namely a basis matrix representation and a blending function representations. We also mentioned that they are equivalent. So, any one representation is good enough. Now, in the context of interpolating representations, we will try to represent them using basis matrix form.

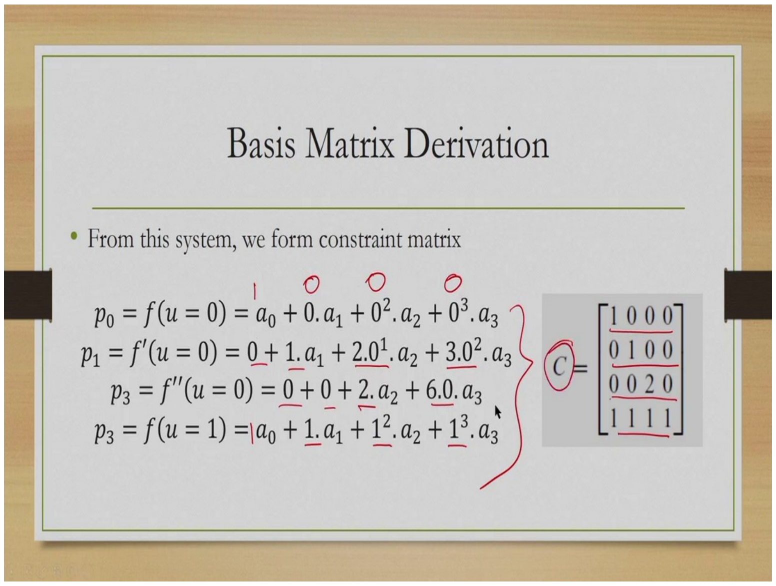
So, in case of natural cubic spline, since we are using cubic polynomial so the general polynomial piece equations should look something like this. Where a0, a1, a2 and a3 are coefficients, u is the parameter. Now we already know that p0 or the control point p0 and the control point p1 control point p2, it should be p2 and the control point p3 have specific meaning, so p0 and p3 represent the point at the boundaries. So, then we can set up the equations, something like this, p0 is the function value at u equal to 0 which we can obtain by replacing u with 0.

So, we will get a0, 0.a1, 0 square a2, 0 cube a3 just replace 0, replace u with 0. Similarly for p3, we have the function value at u equal to 1. So, in this case we replace u with 1 and we get something like this. These two are equations corresponding to the control points, p0 and p3, p1 and p2, as we said represent the first order and the second order derivative at u equal to 0. So, p1 is the first order derivative, p2 is the second order derivative with respect to u.

Now if we compute the derivatives and then replace u with value of 0 in case of p1, what we will get will get equation, something like this. You can try it yourself to first obtain the derivative and then replace the u values with 0. Now we compute the derivative again to get the second order derivative at u equal to 0 corresponding to p2 and then replace u with 0 again. And we get equation something like this.

So, these 4 equations represent the system of equations that we can obtain by utilizing the control point characteristics. And from these equations, from these set of equations, we can derive the basis matrix. How we can do that, we can first construct the constraint matrix by taking simply these values attached to the coefficients and then we take the inverse of that to get the basis matrix.

(Refer Slide Time: 9:53)



Basis Matrix Derivation

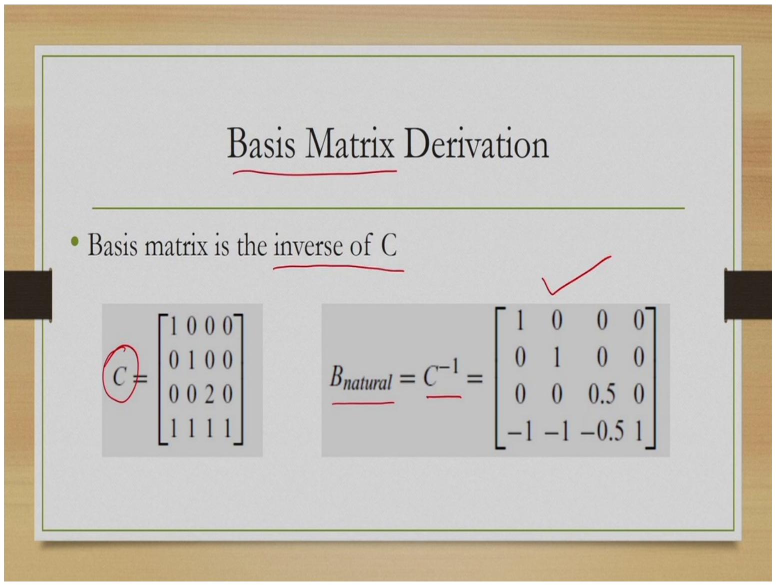
- From this system, we form constraint matrix

$$p_0 = f(u=0) = \overset{1}{a_0} + \overset{0}{0}.a_1 + \overset{0}{0^2}.a_2 + \overset{0}{0^3}.a_3$$
$$p_1 = f'(u=0) = 0 + 1.a_1 + 2.0^1.a_2 + 3.0^2.a_3$$
$$p_3 = f''(u=0) = 0 + 0 + 2.a_2 + 6.0.a_3$$
$$p_3 = f(u=1) = a_0 + 1.a_1 + 1^2.a_2 + 1^3.a_3$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

So, what is the constraint matrix? Let us recast the equation here. So, from there what we get, as we can see, 1, 0, 0, 0, this is the first row. 0 1 0 0. This is the second row, 0 0, 2 and 0 this is the third row and finally 1, 1, 1, 1. This is the fourth row. So, just by utilizing these values, we get the constant matrix C.

(Refer Slide Time: 10:41)



Basis Matrix Derivation

- Basis matrix is the inverse of C

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

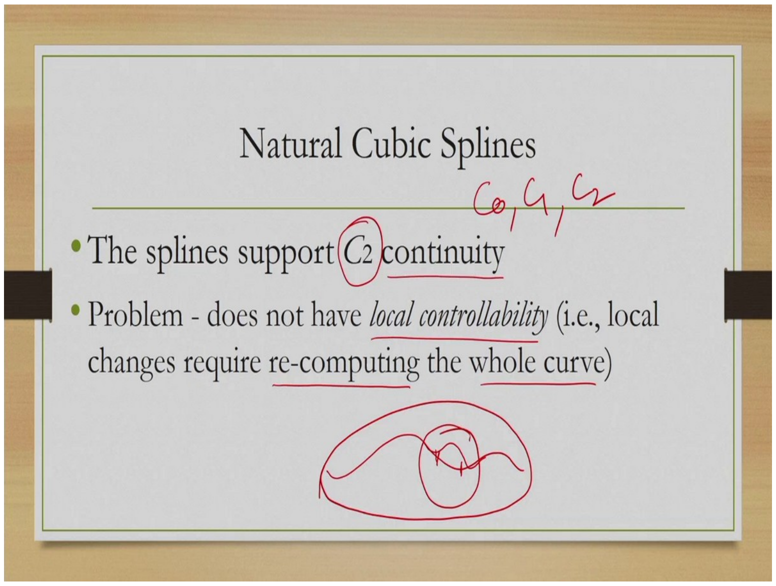
$$B_{natural} = C^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ -1 & -1 & -0.5 & 1 \end{bmatrix}$$

Now, to get basis matrix what we need to do, we need to take the inverse of C, how to compute the inverse that we will not discuss here, of course, because that is a very basic knowledge. And you can refer to any book on basic matrix. Also, you can refer to the appendix of the reference material that will be mentioned at the end of this lecture to learn how to get the inverse of a matrix, assuming we already know that. So, if this is our constraint matrix C and if we take the inverse, the matrix that we will get is this one. And this is the basis matrix for the natural cubic spline.

So, let us recollect the series of steps we followed to get the basis matrix first we defined the control points then using those control points. We set up the system of equations and from the equations we formulated the constraint matrix C. Then we took the inverse of C to get the basis matrix, since basis matrix is the characteristic matrix for the cubic polynomials and that natural Cubic spline, are made up of cubic polynomials.

So, we can say that this basis matrix, which is characteristics of cubic polynomial is good enough to represent the natural cubic spline. So, we will follow that line of argument as discussed in the previous lecture.

(Refer Slide Time: 12:34)



Natural Cubic Splines

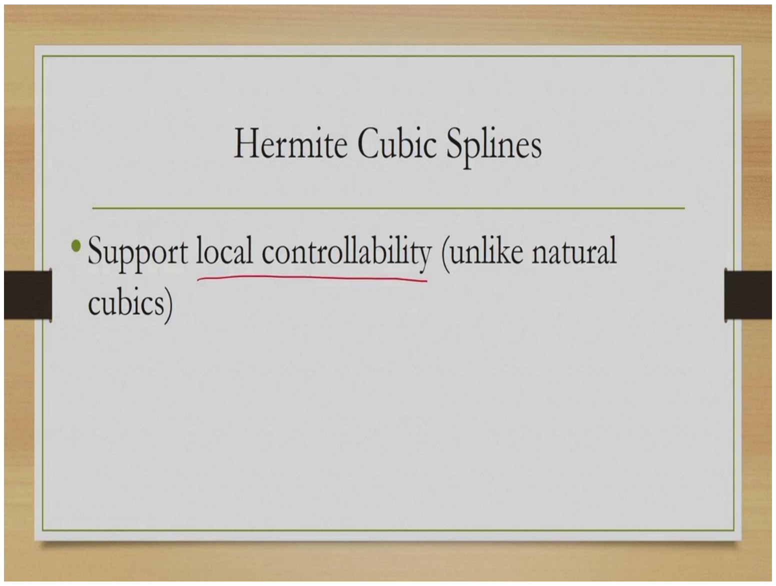
- The splines support C_2 continuity
- Problem - does not have *local controllability* (i.e., local changes require re-computing the whole curve)

So, if we simply use the matrix, then we can say that this is the natural cubic spline instead of specifying the equations or anything. Now, another important property here that these cubic splines have are that they support continuity conditions up to C2 continuity, parametric continuity up to C2 that means they support C0 continuity, C1 continuity and C2 continue.

But the problem is that cubic splines do not have local controllability. That means if we have a spline and we want to change its shape slightly by modifying a very few points on the curve, then we have to re-compute the whole curve rather than re-computing, only the localized area. So, that is essentially a very inefficient approach. For example, suppose I have curve like this and I want to change this part to something like this, but then instead of just restricting our computation within this part, we have to actually compute the whole curve again. So, it does not support local controllability.

Now, let us come to the second type of interpolating splines that is Hermite cubic splines like natural cubic, Hermite cubic splines are also represented by 4 control points.

(Refer Slide Time: 14:39)

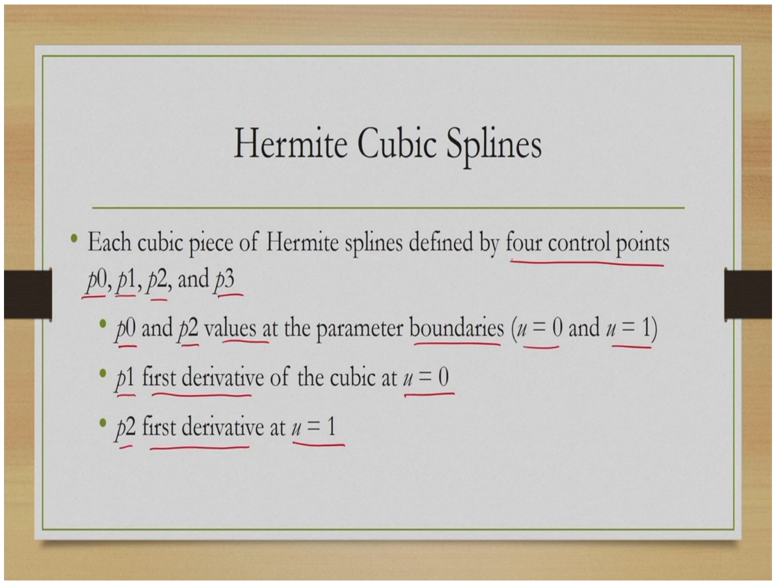


Hermite Cubic Splines

- Support local controllability (unlike natural cubics)

However, it has one important property it supports local controllability. So, the main problem with natural cubic splines is alleviated with the Hermite cubic splines.

(Refer Slide Time: 15:00)



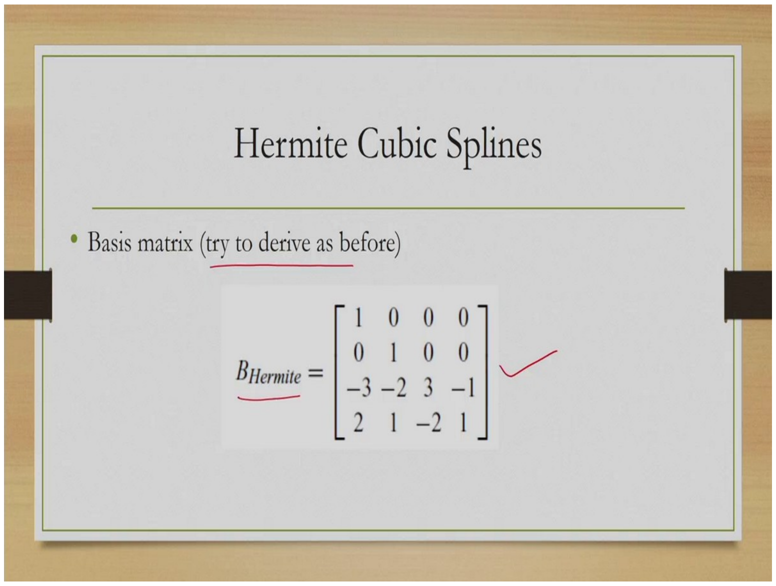
Hermite Cubic Splines

- Each cubic piece of Hermite splines defined by four control points *p*0, *p*1, *p*2, and *p*3
  - *p*0 and *p*2 values at the parameter boundaries (*u* = 0 and *u* = 1)
  - *p*1 first derivative of the cubic at *u* = 0
  - *p*2 first derivative at *u* = 1

So, the 4 control points that defines Hermite cubic splines can be denoted by p0, p1, p2 and p3. Now where p0 and p2 are the values at the parameter boundaries that means u equal to 0 and u equal to 1, p1 is the first derivative at u equal to 0 and p2 is the first derivative at u = 1. So, earlier we had p0 and p3 to be the points at the boundary p1 and p2 are the first and second derivatives at the same point that is u equal to 0.

Now what we have is p0 and p2 here using to represent the points at the parameter boundaries, namely u equal to 0 and u equal to 1. P1 is the first derivative at u equal to 0 and P2 is the first derivative at u equal to 1. So, at both the boundary points we have first derivative denoted by the control points P1 and P2.

(Refer Slide Time: 16:14)



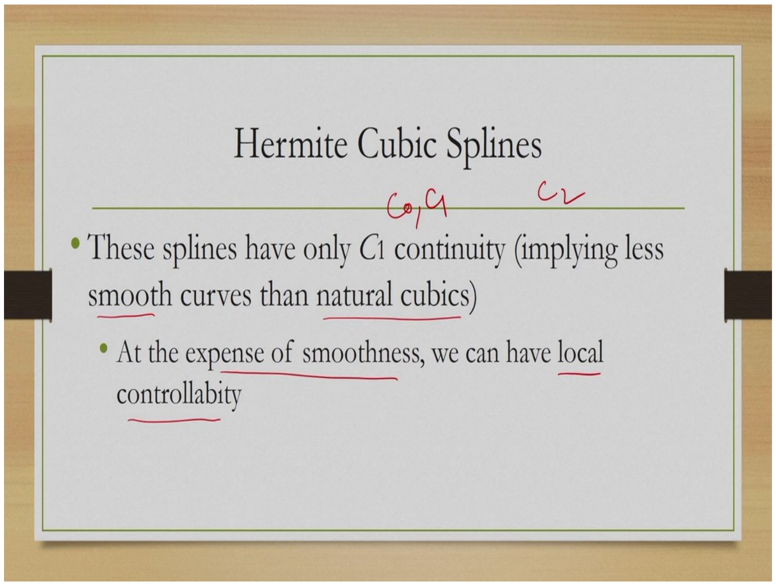
Hermite Cubic Splines

- Basis matrix (try to derive as before)

$$B_{Hermite} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & -2 & 3 & -1 \\ 2 & 1 & -2 & 1 \end{bmatrix}$$

Following the approach we used to derive the basis matrix for natural cubic splines, we can also derive the basis matrix for the Hermite cubic splines, will not do that here. And you can try it yourself. The approach is same. So, you set up a system of equations, identify the constraint matrix take its inverse and get the basic matrix. The Matrix looks something like this, which is the representation for the Hermite cubic splines.

(Refer Slide Time: 17:04)



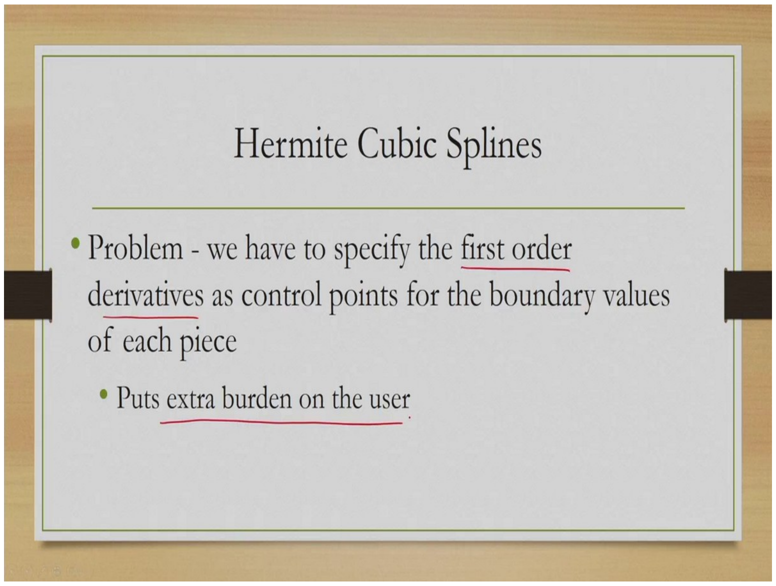
Hermite Cubic Splines

C_0, C_1 C_2

- These splines have only $C1$ continuity (implying less smooth curves than natural cubics)
  - At the expense of smoothness, we can have local controllabity

Now, although the Hermite cubic splines support local controllability, but they do not support all parametric continuity conditions, unlike the natural cubics, they support only C0 and C1 continuity conditions and do not support C2 condition. But natural cubics support all these threes C0, C1, C2. This implies that the curve that results from the Hermite cubics polynomials are less smooth compared to natural cubic polynomial base spline. So, at the expense of smoothness. We are getting the local controllability property, so we are getting local controllability, but we are losing to some extent, the degree of smoothness in the curve.

(Refer Slide Time: 18:15)

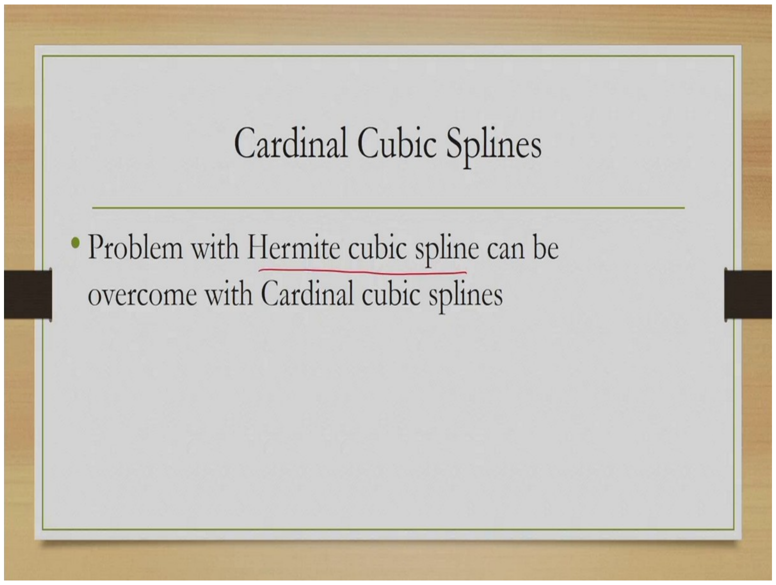


Hermite Cubic Splines

- Problem - we have to specify the first order derivatives as control points for the boundary values of each piece
  - Puts extra burden on the user

Now, one problem here is that we have to specify the first order derivatives. As control points and both the boundaries, clearly this puts extra burden on the user.

(Refer Slide Time: 18:47)

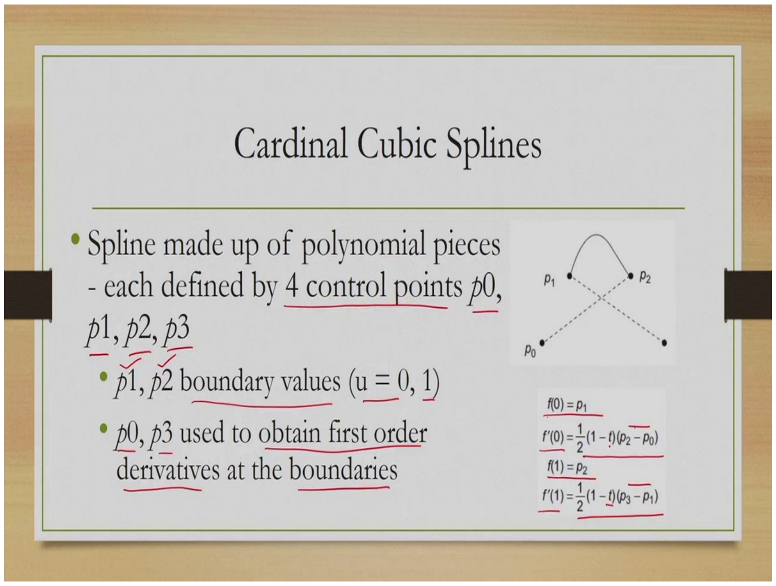


Cardinal Cubic Splines

- Problem with Hermite cubic spline can be overcome with Cardinal cubic splines

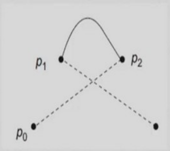
To avoid that we have another spline curve that is the cardinal cubic splines. Now, with this spline, we can resolve the problem that we faced with Hermite cubic splines that is having to define the first order derivative and both the boundary points.

(Refer Slide Time: 19:08)



Cardinal Cubic Splines

- Spline made up of polynomial pieces - each defined by 4 control points $p0$, $p1$, $p2$, $p3$
  - $p1$, $p2$ boundary values (u = 0, 1)
  - $p0$, $p3$ used to obtain first order derivatives at the boundaries

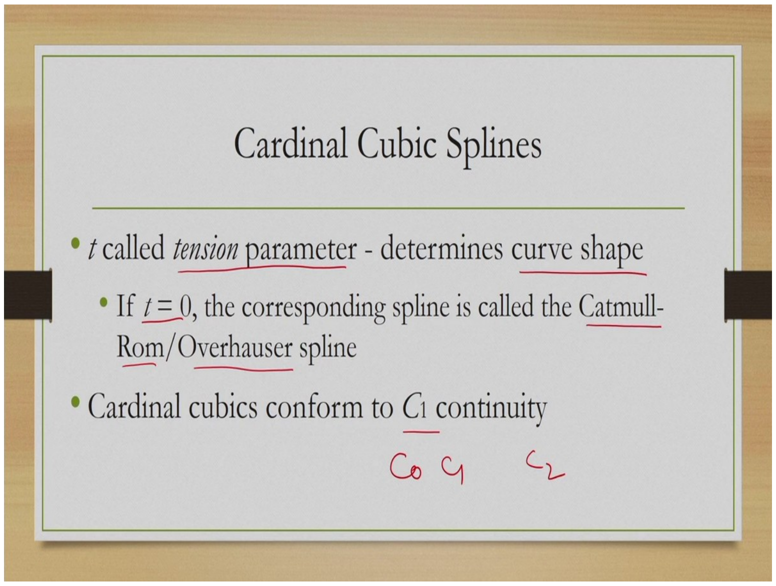


$$f(0) = p_1$$
$$f'(0) = \frac{1}{2}(1-t)(p_2 - p_0)$$
$$f(1) = p_2$$
$$f'(1) = \frac{1}{2}(1-t)(p_3 - p_1)$$

Like before since again, we are dealing with cubic splines, this splines is also made up of polynomial pieces, and each piece is defined by 4 control points. Again, we are denoting them using p0 to p3. Same notations we are using but here p1 and p2 represent the boundary values that means at u equal to 0 and u equal to 1. P0 and p3 are used to obtain first order derivatives at the boundaries. How, look at this system of equations here. This is the p1 control point which is the function value at 0. This is p2 control point which is the function value at 1, u equal to 1.

Now first order derivative at u = 0 can be obtained using this equations. Similarly first order derivative at u = 1 can be obtained using these equations. Where we have used that to control points in this fashion. And also used one additional parameter t in both cases.

(Refer Slide Time: 20:40)


Cardinal Cubic Splines

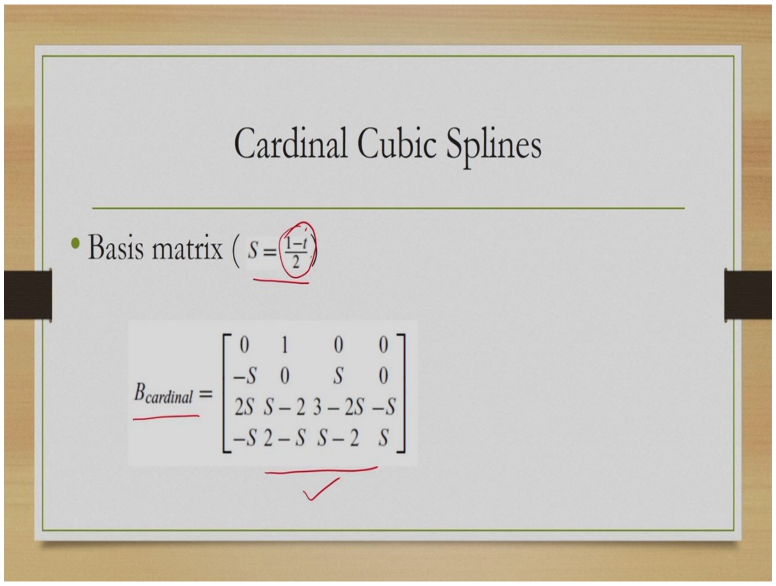
- *t* called *tension* parameter - determines curve shape
  - If $t = 0$, the corresponding spline is called the Catmull-Rom/Overhauser spline
- Cardinal cubics conform to C_1 continuity

$C_0 \quad C_1 \quad C_2$

Now t is called tension Parameter. So, essentially it determines the shape of the curve when t equal to 0 what we get is the catmull-rom or overhuset spline. So, that is a special type of spline when t equal to 0. So, using the value of t we can actually control the shape of the overall spline curve. So, here we do not have to actually compute the derivatives instead we can derive it using the control points and the value of t, so that actually makes the life of the user simpler.

But again, the Cardinal Cubic also suffered from the same problem. That is it supports up to C0 and see one parametric continuity conditions. It does not support C2. So, it has less smooth curve than natural Cubics.

(Refer Slide Time: 21:48)



Cardinal Cubic Splines

- Basis matrix ($S = \frac{1-t}{2}$)

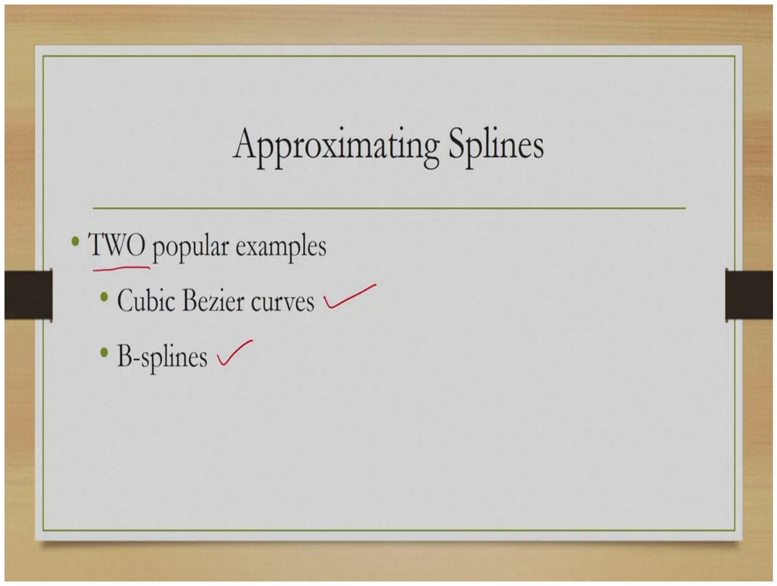
$$B_{cardinal} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -S & 0 & S & 0 \\ 2S & S-2 & 3-2S & -S \\ -S & 2-S & S-2 & S \end{bmatrix}$$

So, then what is left is the basis matrix formulation. This is slightly complicated, but again, you can try using the same approach by setting up the system of equations where some rearrangements may required. Then you create the constraint matrix, take the inverse to get the basis matrix for the cardinal cubic spline. Which is represented in the form of S where S is this expression.

So, to summarize, we have learned about 3 interpolating splines. Each of these interpolating splines are made up of polynomial pieces. And these pieces are Cubic. So, each piece is defined by 4 control points. In case of natural cubic, the control points are defined in a particular way, in case of Hermite cubic it is defined in a different way and in case of Cardinal Cubic, it is defined in yet another way. So, natural Cubic is very smooth, but it has some problems.

Those problems are particularly related to local controllability, which is taken care of in the Hermite Cubic, but at the expense of smoothness. But specifying Hermite Cubic's is slightly difficult, which is again taken care of by Cardinal Cubic's. But the smoothness remains less compared to natural Cubic's. Now, let us move to the other category of splines, namely the approximate splines, so what are these splines? Just to recap here, we have a set of control points, but this spline need not pass through all the points. Instead, the control points are used to define the convict's hull, which determines the shape of the overall Spline curve.

(Refer Slide Time: 24:18)

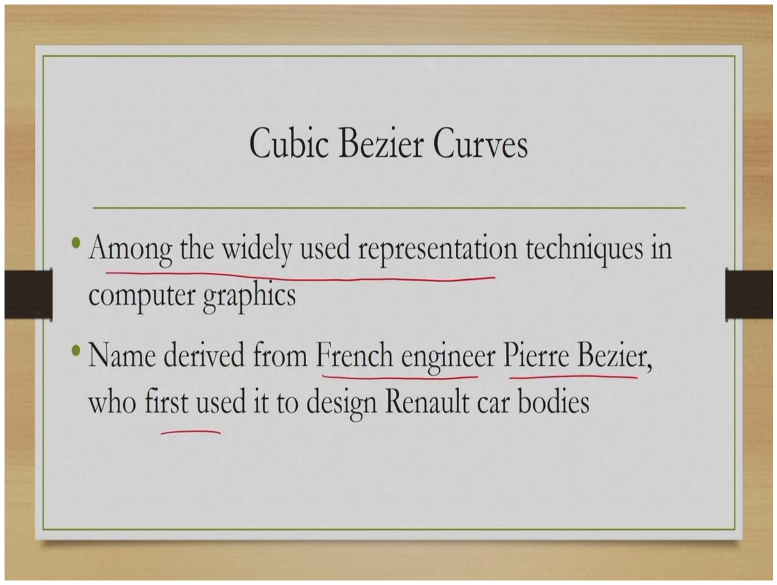


Approximating Splines

- TWO popular examples
  - Cubic Bezier curves ✓
  - B-splines ✓

Now, there are two popular approximating splines used in computer graphics. One is called Cubic Bezier curves, and the other one is B splines. Like in the case of interpolating splines, let us try to understand these types in a little more details. We will start with cubic Bezier curves.

(Refer Slide Time: 24:47)

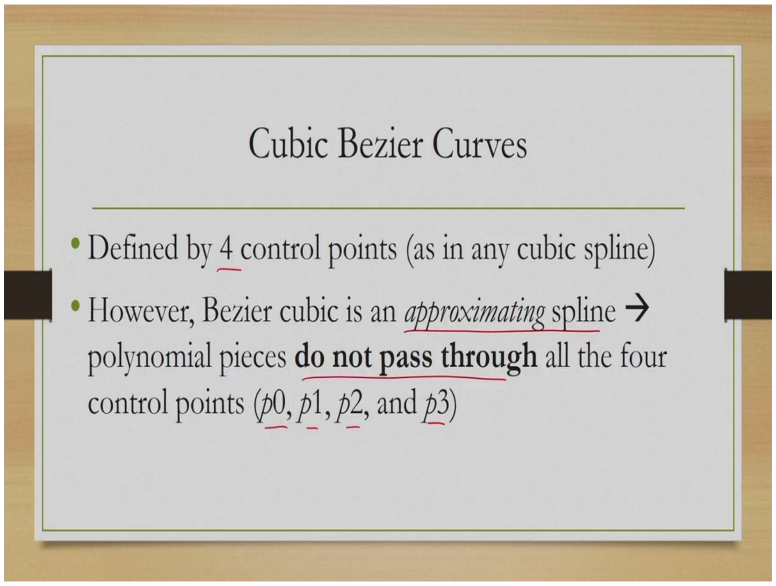


Cubic Bezier Curves

- Among the widely used representation techniques in computer graphics
- Name derived from French engineer Pierre Bezier, who first used it to design Renault car bodies

Now, these particular curves, the Cubic Bezier curves are among the widely used representation technique in graphics. The name has been derived from the French engineer Pierre Bezier, who first used it to design Renault Car bodies.

(Refer Slide Time: 25:17)

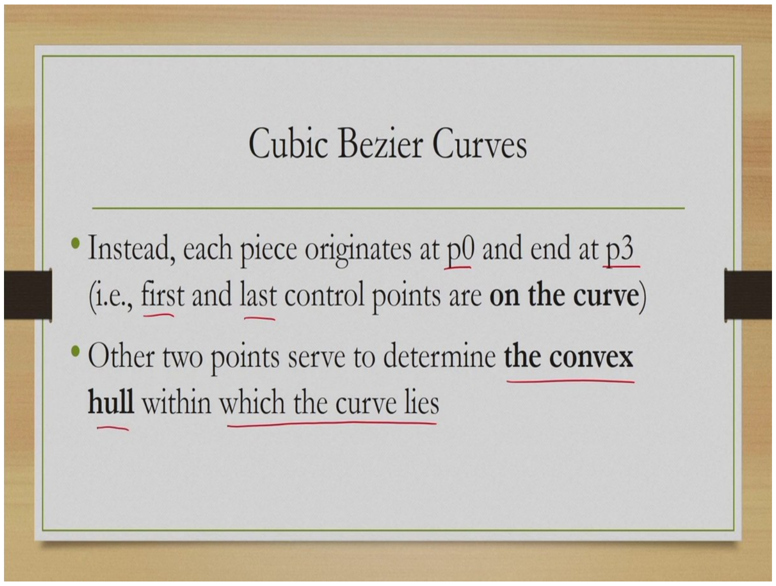


Cubic Bezier Curves

- Defined by 4 control points (as in any cubic spline)
- However, Bezier cubic is an *approximating* spline → polynomial pieces **do not pass through** all the four control points (*p*0, *p*1, *p*2, and *p*3)

Since it is Cubic curve so defined by 4 control points as before, but here the difference is that the curve is an approximating Spline that means the polynomial pieces do not pass through all the 4 control points. We can denote these points like before p0, p1, p2 and p3. So, there are 4 points, but it is not necessary that the curve passes through all the points.

(Refer Slide Time: 25:51)

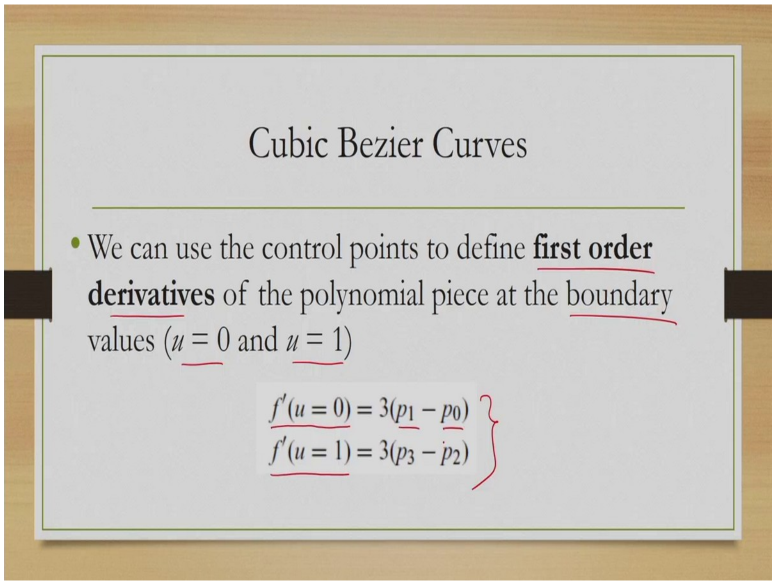


Cubic Bezier Curves

- Instead, each piece originates at p0 and end at p3 (i.e., first and last control points are **on the curve**)
- Other two points serve to determine **the convex hull** within which the curve lies

Instead, these points are used to define the convex hull, so each piece originates at p0 and ends at p3, that is the first and the last control points. The other two points are used to determine the convex hull within which the curve lies. So, 2 points are on the curve and other 2 points are controlling the shape of the curve.

(Refer Slide Time: 26:24)



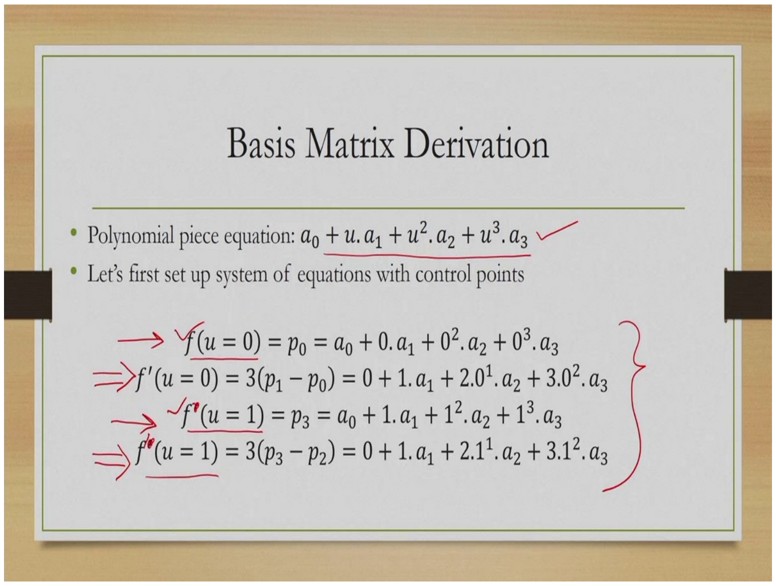
Cubic Bezier Curves

- We can use the control points to define **first order derivatives** of the polynomial piece at the boundary values ($u = 0$ and $u = 1$)

$$f'(u=0) = 3(p_1 - p_0)$$
$$f'(u=1) = 3(p_3 - p_2)$$

Now, the control points can be used to define first order derivatives at the boundary values that is u equal to 0 and u equal to one using these expressions. So, this is the first order derivative at u equal to 0 and this is the first order derivative u equal to 1. And these two derivatives are defined in terms of the control points as shown in these examples.

(Refer Slide Time: 27:05)



Basis Matrix Derivation

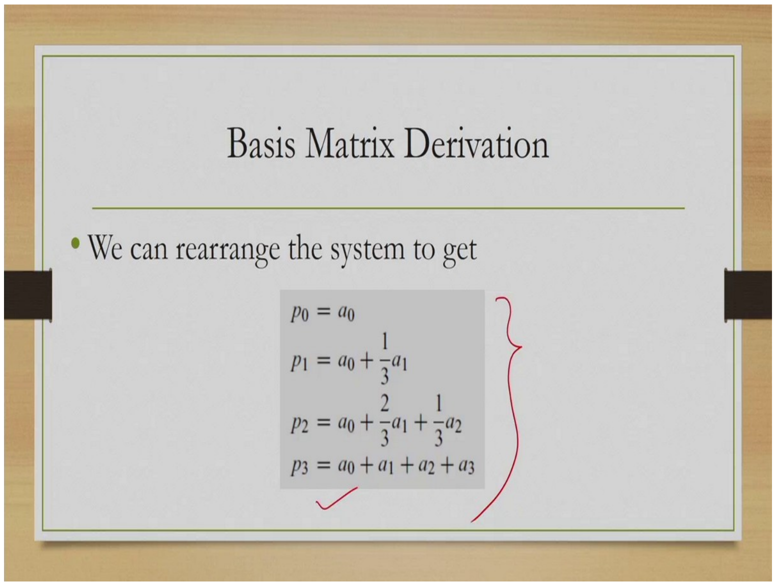
- Polynomial piece equation: $a_0 + u.a_1 + u^2.a_2 + u^3.a_3$
- Let's first set up system of equations with control points

$$f(u=0) = p_0 = a_0 + 0.a_1 + 0^2.a_2 + 0^3.a_3$$
$$f'(u=0) = 3(p_1 - p_0) = 0 + 1.a_1 + 2.0^1.a_2 + 3.0^2.a_3$$
$$f(u=1) = p_3 = a_0 + 1.a_1 + 1^2.a_2 + 1^3.a_3$$
$$f'(u=1) = 3(p_3 - p_2) = 0 + 1.a_1 + 2.1^1.a_2 + 3.1^2.a_3$$

Now we can set up the equation. This is the Cubic polynomial equation, as we have seen before, and if we replace the values, the boundary values. And the. This should be the boundary value. This is one value and this is another value, and this would be the first derivative equal to 1. So, then what we get is this system of equation. So, this is boundary value at u equal to 0 boundary value at u equal to 1 this is first order derivative u equal to one and first order derivative at u equal to 0.

So, for each we can actually replace u with the corresponding value and then get the set of equations, as we have seen earlier.

(Refer Slide Time: 28:19)



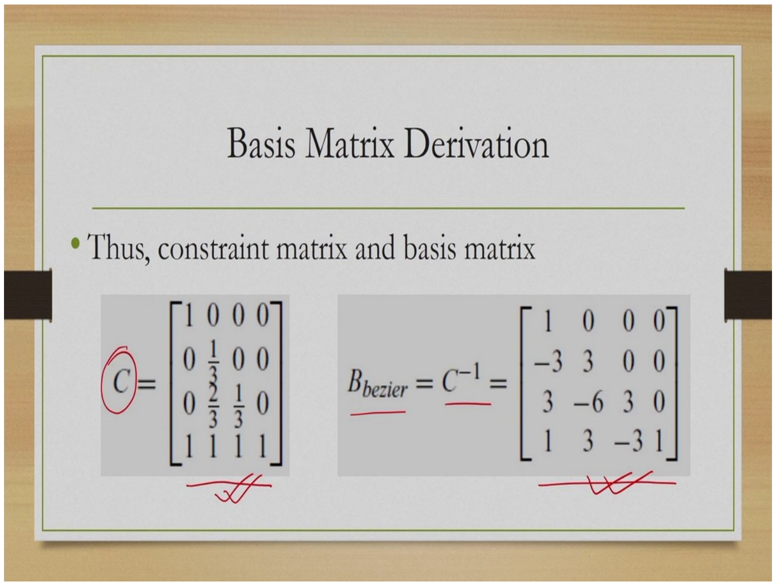
Basis Matrix Derivation

- We can rearrange the system to get

$$\left.\begin{aligned} p_0 &= a_0 \\ p_1 &= a_0 + \frac{1}{3}a_1 \\ p_2 &= a_0 + \frac{2}{3}a_1 + \frac{1}{3}a_2 \\ p_3 &= a_0 + a_1 + a_2 + a_3 \end{aligned}\right\}$$

From this set of equation we can rearrange and get this form shown here for the control points.

(Refer Slide Time: 28:39)



Basis Matrix Derivation

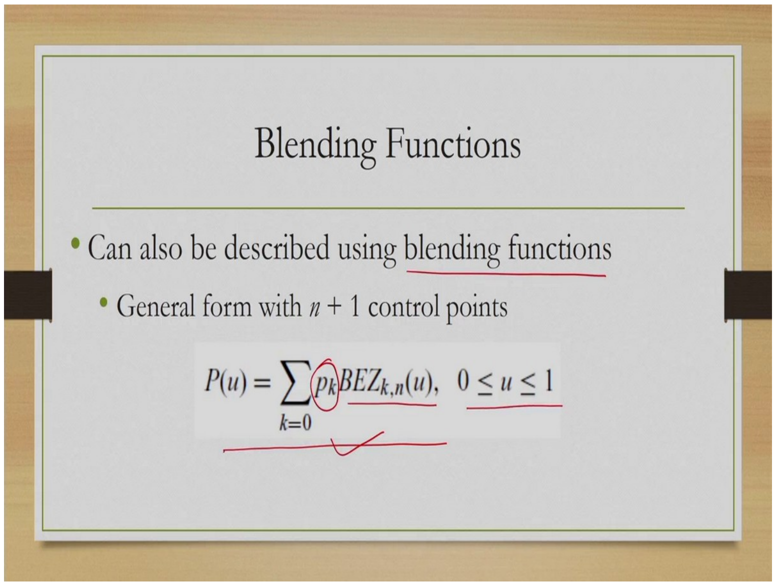
- Thus, constraint matrix and basis matrix

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 \\ 0 & \frac{2}{3} & \frac{1}{3} & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$B_{bezier} = C^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ 1 & 3 & -3 & 1 \end{bmatrix}$$

Now, from that rearranged form, we can get the constraint matrix as shown here. This is the constraint matrix. Then we take the inverse of the constraint matrix C^{-1} to get the basis matrix, representation of the Bezier curves, which is something like this. So, note here we followed the same approach that is first we created the set of equations using the control point characteristics, then we formulated the constraint matrix and finally we took the inverse of the constraint matrix to get the basis matrix.

(Refer Slide Time: 29:32)



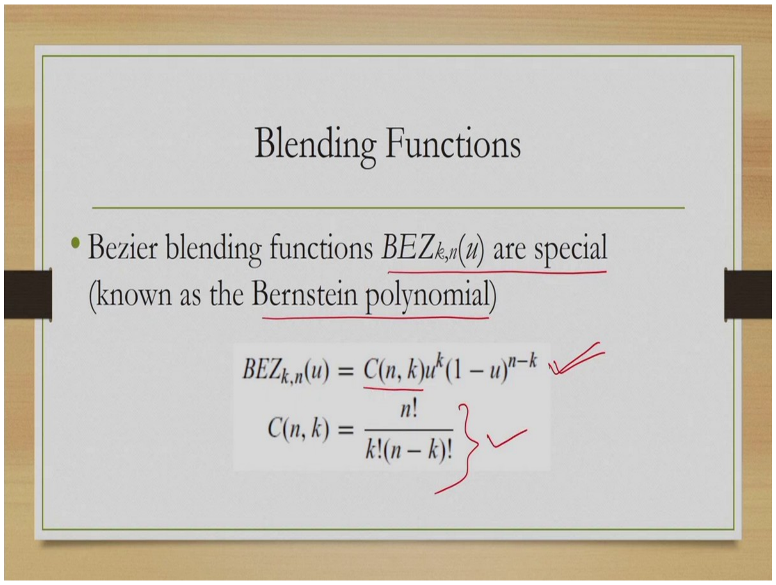
Blending Functions

- Can also be described using blending functions
  - General form with $n + 1$ control points

$$P(u) = \sum_{k=0} p_k BEZ_{k,n}(u), \quad 0 \leq u \leq 1$$

There is actually a blending function representation also for Bezier curves. We already said they are equivalent, but blending function representation for Bezier curves is also quite popularly used. Which as a general form something like this, where pk is the control points and Bez is the blending functions. And the function is defined within this range is u between 0 and 1.

(Refer Slide Time: 30:12)



Blending Functions

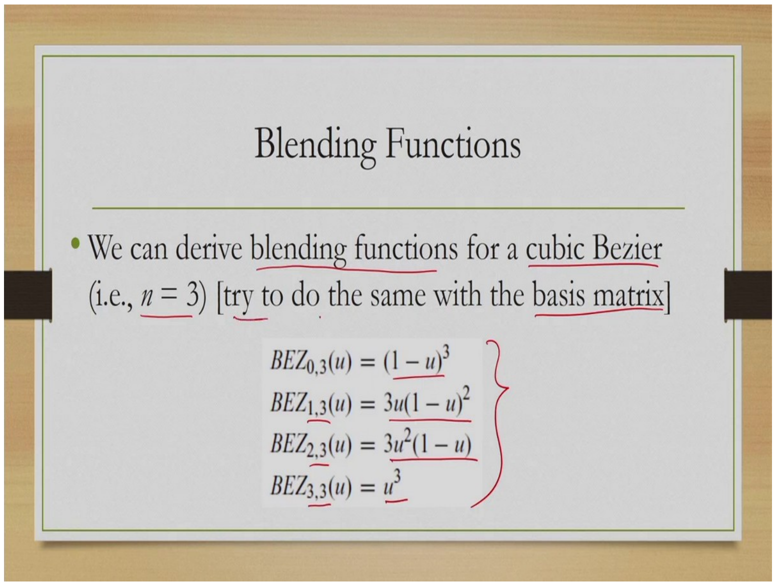
- Bezier blending functions $BEZ_{k,n}(u)$ are special (known as the Bernstein polynomial)

$$BEZ_{k,n}(u) = C(n,k)u^k(1-u)^{n-k}$$

$$C(n,k) = \frac{n!}{k!(n-k)!}$$

Now, these blending functions are actually special functions, sometimes they are known as Bernstein Polynomial which has this form where C is this one. So, Bezier curves we can represent using blending functions where the blending functions are called Bernstein polynomials, which takes the forms on here in this first line. Where the term C (n, k) is shown in the second line.

(Refer Slide Time: 31:00)



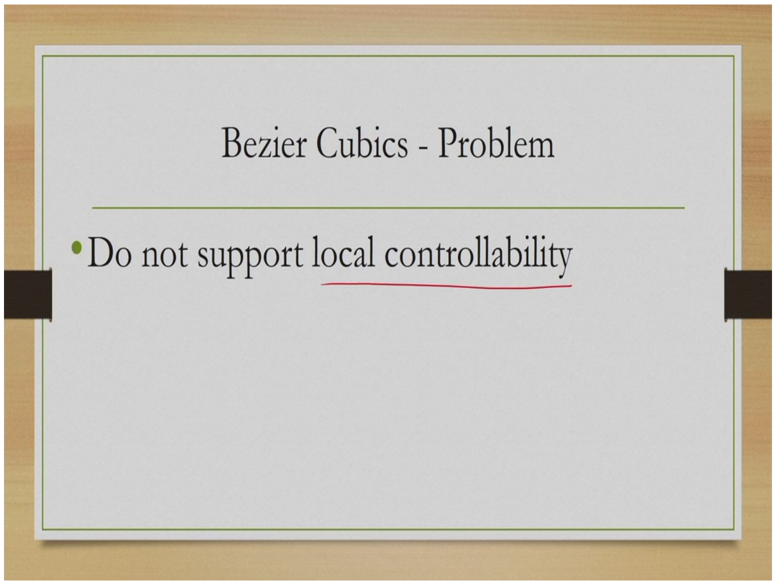
Blending Functions

- We can derive blending functions for a cubic Bezier (i.e., $n = 3$) [try to do the same with the basis matrix]

$$BEZ_{0,3}(u) = (1-u)^3$$
$$BEZ_{1,3}(u) = 3u(1-u)^2$$
$$BEZ_{2,3}(u) = 3u^2(1-u)$$
$$BEZ_{3,3}(u) = u^3$$

Now, if we expand the blending functions for say a cubic Bezier where n equal to 3, then how it looks? BEZ 0, 3 is $(1-u)^3$, 1, 3 is this one 2, 3 is this one and 3, 3 is this one. So, these are the blending functions for Bezier Cubic curve. Same thing we can derive from the basis matrix also. As I said in the previous lecture, that they are equivalent. One can be derived from another and you can try to do the same.

(Refer Slide Time: 31:46)

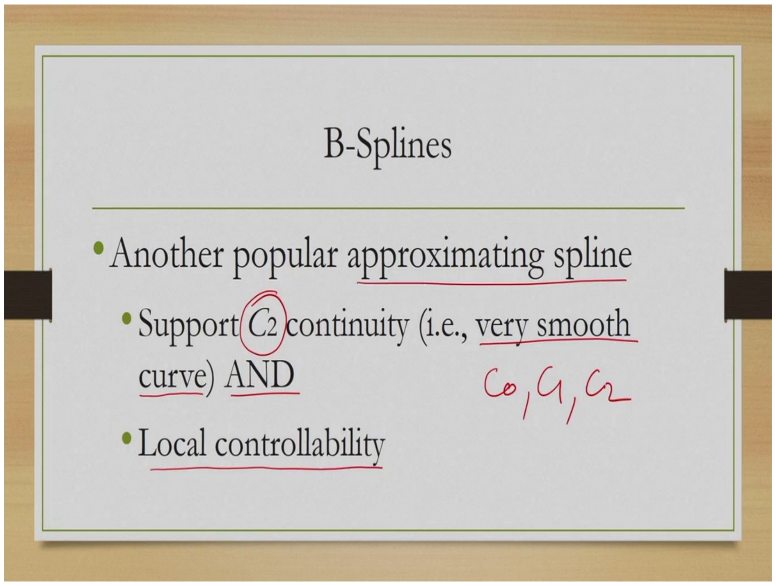


Bezier Cubics - Problem

- Do not support local controllability

So, one major problem with the Bezier Cubic is that they do not support local controllability. That is, of course a major concern. Now, let us move to the other category of splines that is B splines.

(Refer Slide Time: 32:14)



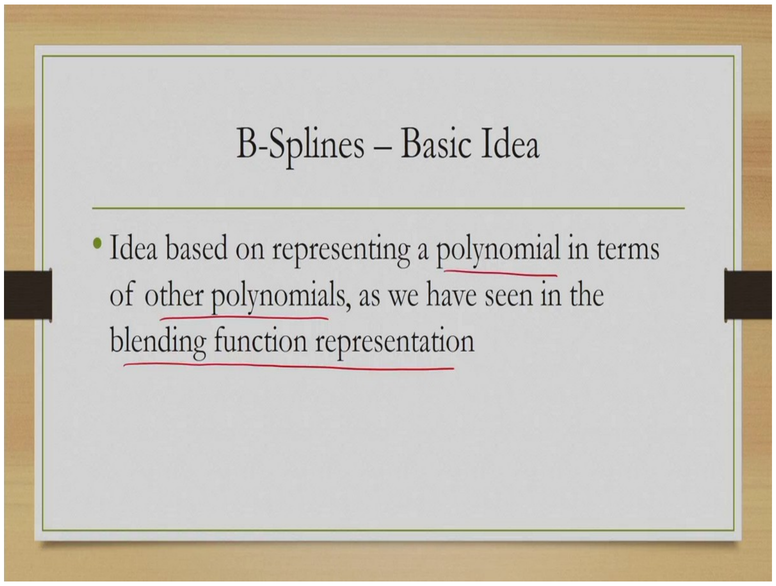
B-Splines

- Another popular approximating spline
  - Support C_2 continuity (i.e., very smooth curve) AND
  - Local controllability

C_0, C_1, C_2

Now here, the idea is slightly more complicated. Now, B splines are approximating splines, as we have already mentioned, they support up to C2 continuity conditions, parametric continuity conditions. That means C0, C1 and C2 all three they support. In other words, these splines give us a very smooth curve and they support local controllability. So, Bezier curves do not have local controllability, whereas these splines is very smooth. They are very smooth as well as they support local controllability. So, all our problems are solved with these splines.

(Refer Slide Time: 33:02)

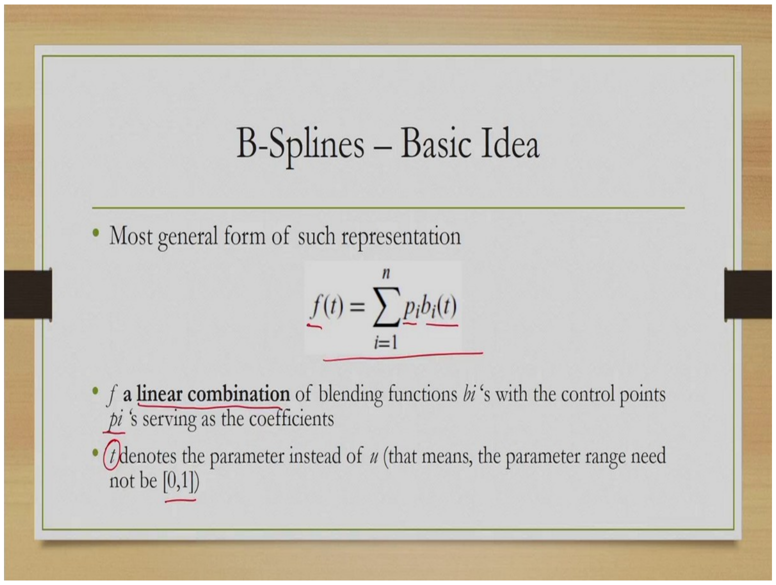


B-Splines – Basic Idea

- Idea based on representing a polynomial in terms of other polynomials, as we have seen in the blending function representation

Now, let us try to understand the basic idea, the mathematics behind it is slightly complicated, so we will try to keep it at the minimum for simplicity. What is the idea? The idea is based on representing a polynomial in terms of other polynomials, as we have seen in the blending function representation. So, that is where this idea starts, that a polynomial can be represented in terms of other polynomial.

(Refer Slide Time: 33:40)



B-Splines – Basic Idea

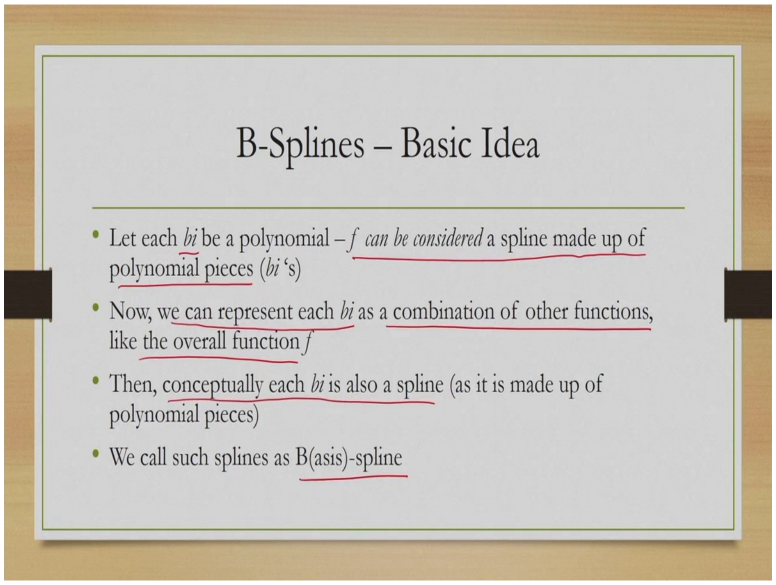
- Most general form of such representation

$$f(t) = \sum_{i=1}^{n} p_i b_i(t)$$

- f a **linear combination** of blending functions bi 's with the control points pi 's serving as the coefficients
- t denotes the parameter instead of u (that means, the parameter range need not be [0,1])

So, what is the most general form of such representation, we have encountered this expression in the previous lecture pi control points and bi is the blending function. So, we are assuming in this representation that the function f is a linear combination of the blending functions where the control points p serve as the coefficients. Now, in this definition, one thing to be noted here is that we are using a parameter t instead of u. Now this parameter is more general. That means that we need not restrict ourselves to the range defined for you that is between 0 and 1. So, the t can take any value, not necessarily within the range 0 to 1.

(Refer Slide Time: 34:40)



B-Splines – Basic Idea

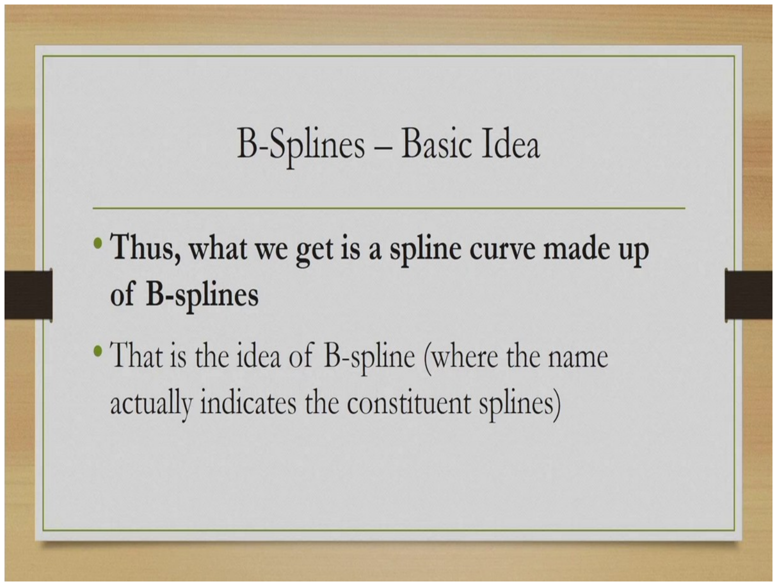
- Let each *bi* be a polynomial – *f can be considered* a spline made up of polynomial pieces (*bi* 's)
- Now, we can represent each *bi* as a combination of other functions, like the overall function *f*
- Then, conceptually each *bi* is also a spline (as it is made up of polynomial pieces)
- We call such splines as B(asis)-spline

Now, each bi is a polynomial. Let us assume that so then f can be considered as a spline made up of polynomial pieces, which is basically the idea now we can represent each bi as a combination of other functions, like the overall function f. So, f is represented as a linear combination of bi’s, each bi in turn can be represented as a combination of other functions. Then conceptually, each bi is also spline, because it is made up of other polynomials.

So, when we talked about Spline, we said that Spline is a curve which is made up of lower degree polynomials now each of these polynomials. Again, we are assuming to be made up of even other polynomials. So, this polynomial pieces are themselves splines rather than simple polynomials. And that is the overall idea of B splines. That is we are having a spline, which is made up of basis spline rather than polynomial pieces.

Now, each basis spline is made up of polynomial pieces. So, the definition is one level higher than basic definition of Spline. In spline we have a curve made up of polynomial pieces in this spline. We have a curve made up of basis splines, which in turn is made up of polynomial pieces.

(Refer Slide Time: 36:19)

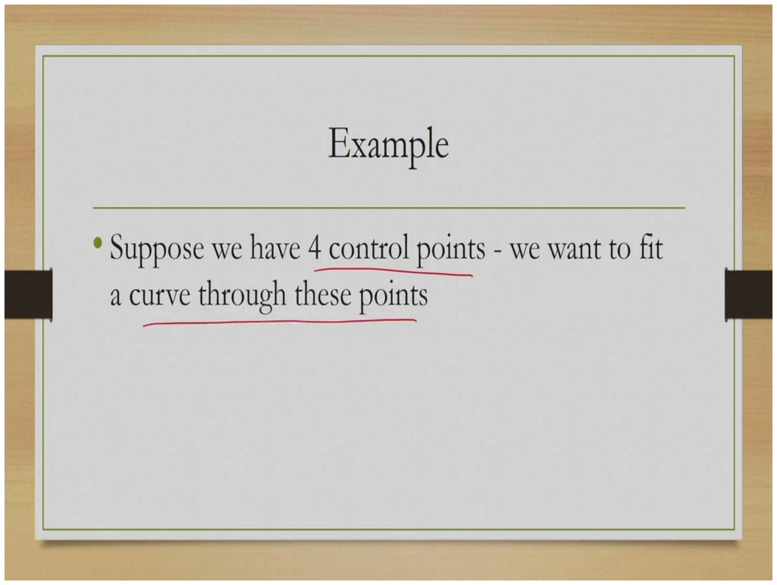


B-Splines – Basic Idea

- **Thus, what we get is a spline curve made up of B-splines**
- That is the idea of B-spline (where the name actually indicates the constituent splines)

So, what we get is a spline made up of B-splines and that is the overall idea. So, when we are talking of B- splines, we are actually referring to the constituent splines.

(Refer Slide Time: 36:36)

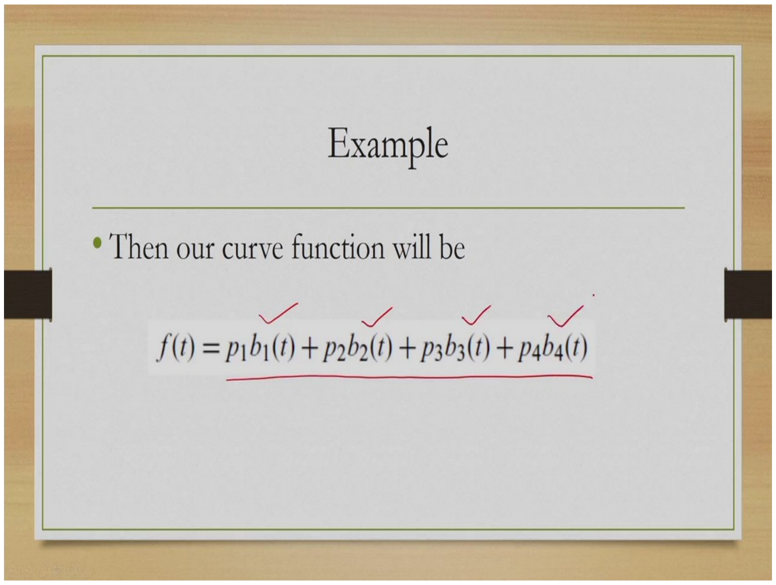


Example

- Suppose we have 4 control points - we want to fit a curve through these points

Let us try to understand in terms of an example. Suppose we have 4 control points and we want to fit a curve through those points.

(Refer Slide Time: 36:51)



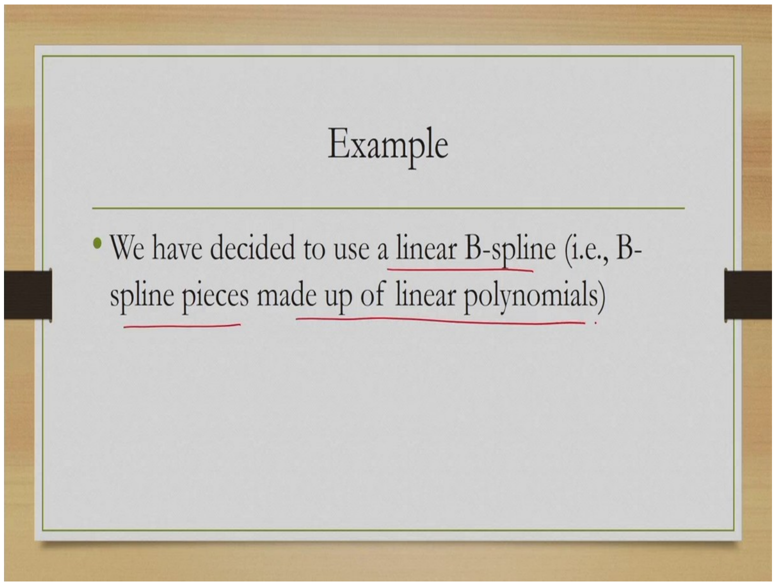
Example

- Then our curve function will be

$$f(t) = \underline{\overset{\checkmark}{p_1b_1(t)} + \overset{\checkmark}{p_2b_2(t)} + \overset{\checkmark}{p_3b_3(t)} + \overset{\checkmark}{p_4b_4(t)}}$$

So, since we have 4 points. So, our curve function will look something like this, we expand the general form on shown earlier to get the 4 blending functions in the general form.

(Refer Slide Time: 37:17)

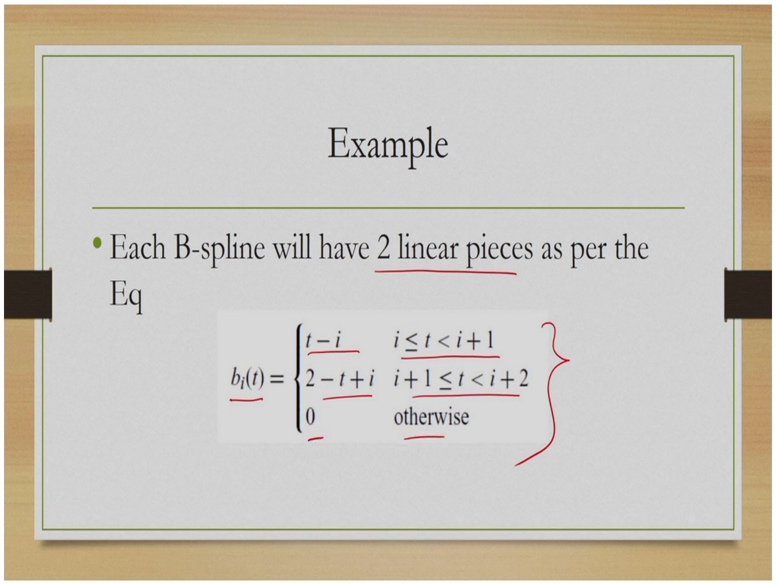


Example

- We have decided to use a linear B-spline (i.e., B-spline pieces made up of linear polynomials)

Now, assume that we are using a linear B-spline that means B-splines pieces made up of linear polynomials.

(Refer Slide Time: 37:30)



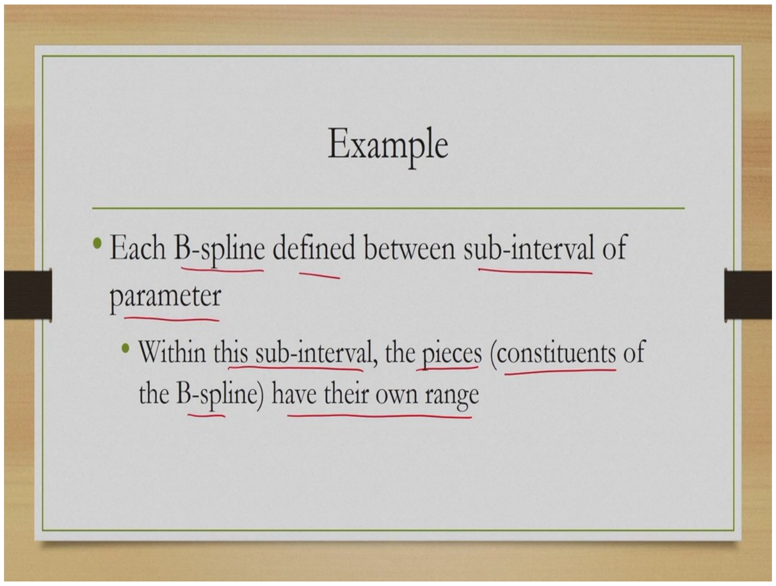
Example

- Each B-spline will have 2 linear pieces as per the Eq

$$b_i(t) = \begin{cases} t-i & i \le t < i+1 \\ 2-t+i & i+1 \le t < i+2 \\ 0 & \text{otherwise} \end{cases}$$

Then, each B-spline will have two linear pieces as per the following equation. This is the B-spline and these are the linear pieces defined within range of t.

(Refer Slide Time: 37:58)

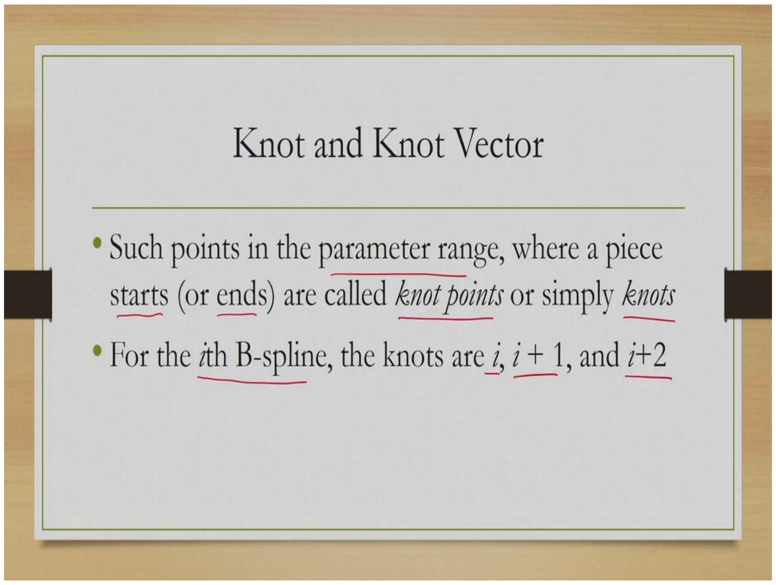


Example

- Each B-spline defined between sub-interval of parameter
  - Within this sub-interval, the pieces (constituents of the B-spline) have their own range

So, as you can see in the previous formulation, that each B-spline is defined between sub intervals of the parameter T now within this sub interval, the pieces that made that are the constituents of the B-splines have their own range. So, we have a range for the B- spline and we have sub ranges for the constituent polynomial pieces of the spline.

(Refer Slide Time: 38:27)

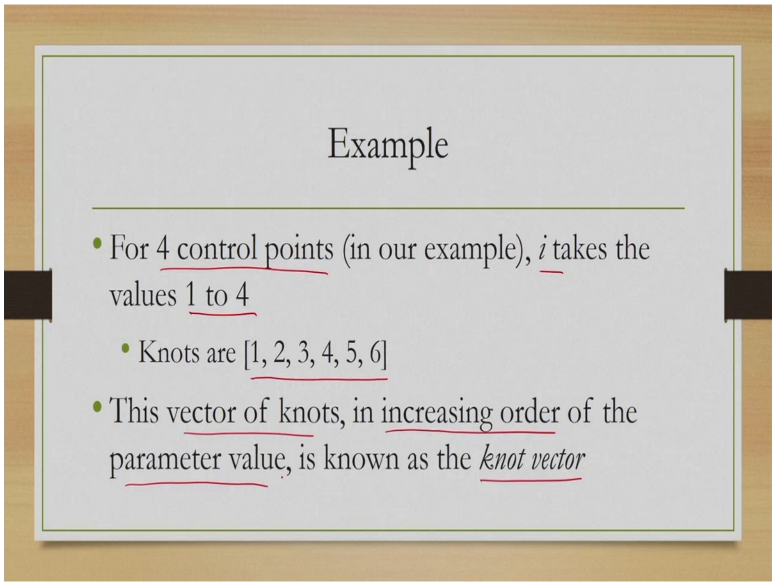


Knot and Knot Vector

- Such points in the parameter range, where a piece starts (or ends) are called *knot points* or simply *knots*
- For the *i*th B-spline, the knots are *i*, *i* + 1, and *i*+2

Now, those points in the parameter range where the piece starts or ends are called knot points or simply knot for the ith B-spline, the knots are i+1 and i+2. Just to recollect, so we have a spline made up of B- spline. Now each B-spline is defined within a range that range is subdivided for each constituent piece of a B-spline. Now, where a piece starts or ends are called knot points or simply knots. For the ith B-spline. We can define the knots as i, i+1 and i+2.

(Refer Slide Time: 39:21)

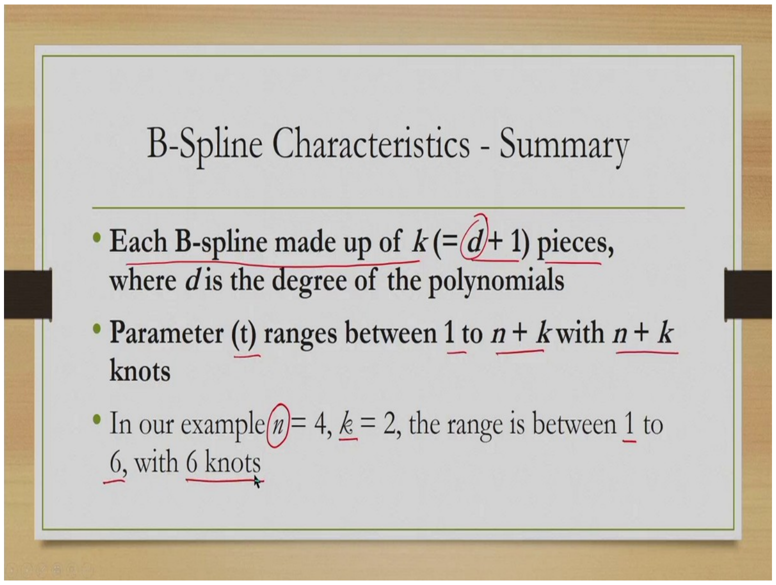


Example

- For 4 control points (in our example), *i* takes the values 1 to 4
    - Knots are [1, 2, 3, 4, 5, 6]
- This vector of knots, in increasing order of the parameter value, is known as the *knot vector*

For the 4 control points in our example, i takes the value 1 to 4. So, then we can have the knots. As 1, 2, 3, 4, 5, 6, these are the 6 knot points, and this set of points is the vector of knots, also known as the knot vector. Which is, of course, an increasing order of parameter values.

(Refer Slide Time: 39:57)

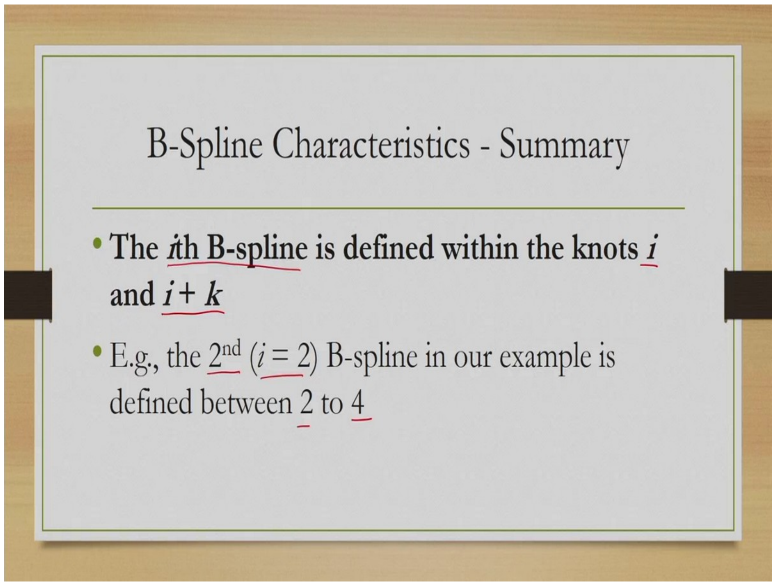


B-Spline Characteristics - Summary

- **Each B-spline made up of k (= d + 1) pieces, where d is the degree of the polynomials**
- **Parameter (t) ranges between 1 to $n + k$ with $n + k$ knots**
- In our example n = 4, k = 2, the range is between 1 to 6, with 6 knots

So, then to summarize each B- spline is made up of k where k equal to d+1, d is the degree of the polynomial. Now parameter t ranges between 1 to n+k having n+k knots. In our example n is 4, n is the number of control points k is 2 since d is 1. So, the range is between 1 to 6 with 6 knots.

(Refer Slide Time: 40:38)

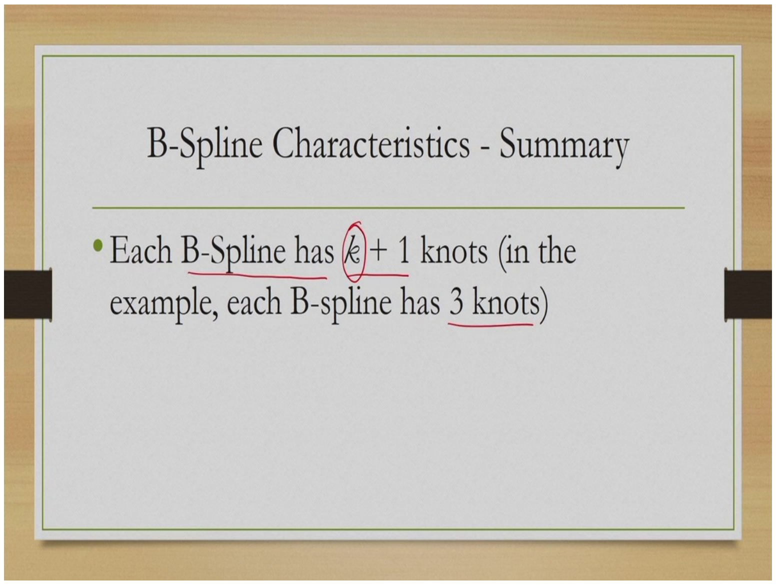


B-Spline Characteristics - Summary

- **The *i*th B-spline is defined within the knots *i* and *i* + *k***
- E.g., the 2nd ($i = 2$) B-spline in our example is defined between 2 to 4

The ith B- spline is defined within the knots i and i+k, for example, the second B-spline that means i=2 is defined between 2 and 2+2 or 4, because k is 2. So, between 2 to 4, that means the knot values are 2, 3 and 4.

(Refer Slide Time: 41:10)

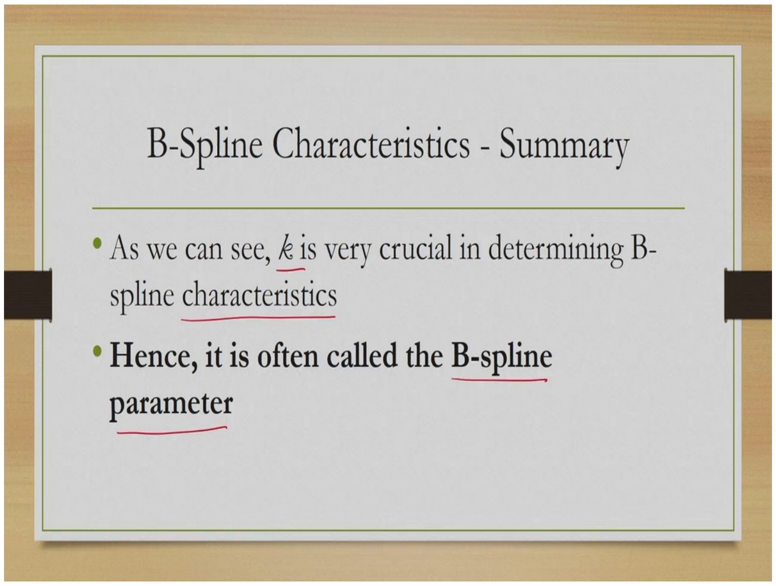


B-Spline Characteristics - Summary

- Each B-Spline has $k + 1$ knots (in the example, each B-spline has 3 knots)

So, each B-spline has k+1 knot's. Now, in our example, since k equal to 2. So, each B-spline will have 3 knots for any other value of k the B-spline will have k+1 knots. So, these are the characteristics of the B-spline.

(Refer Slide Time: 41:32)

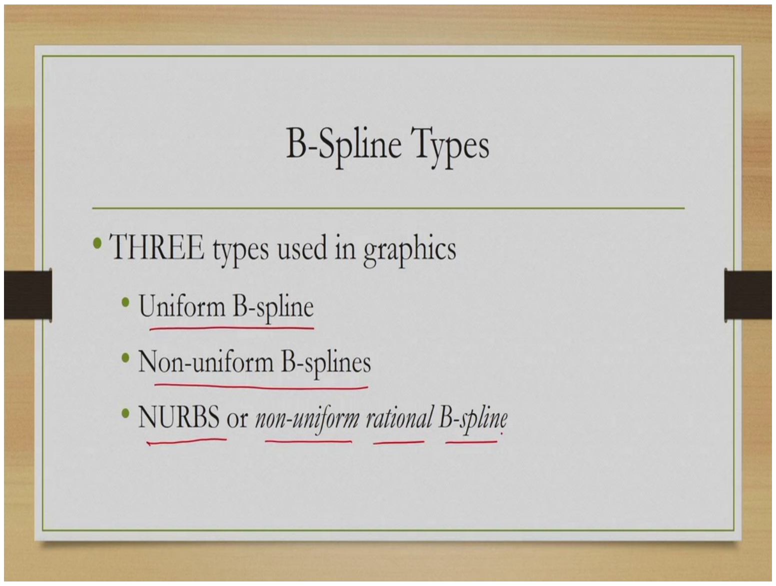


B-Spline Characteristics - Summary

- As we can see, *k* is very crucial in determining B-spline characteristics
- **Hence, it is often called the B-spline parameter**

Another characteristics is k is actually very crucial in determining all these characteristics. So, this k value is often called the B-spline parameter. So, you should keep this in mind that k is very important, it plays a very crucial role in determining the B- spline characteristics that we have enumerated earlier. So, often k is called the B- spline parameter. Now, let us see various types of B-spline. So far, we have given a basic idea, basic introductory idea of B- spline. Now let us see the types that are there for B spline.

(Refer Slide Time: 42:29)

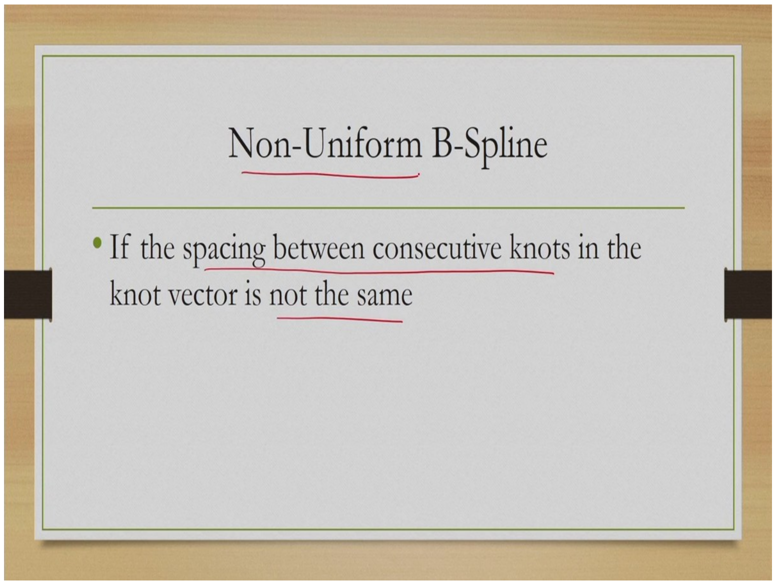


B-Spline Types

- THREE types used in graphics
  - Uniform B-spline
  - Non-uniform B-splines
  - NURBS or *non-uniform rational B-spline*

There are 3 types, uniform B-spline, non-uniform B-spline and nurbs or non-uniform rational B-spline we will briefly introduce each of these types.

(Refer Slide Time: 42:44)

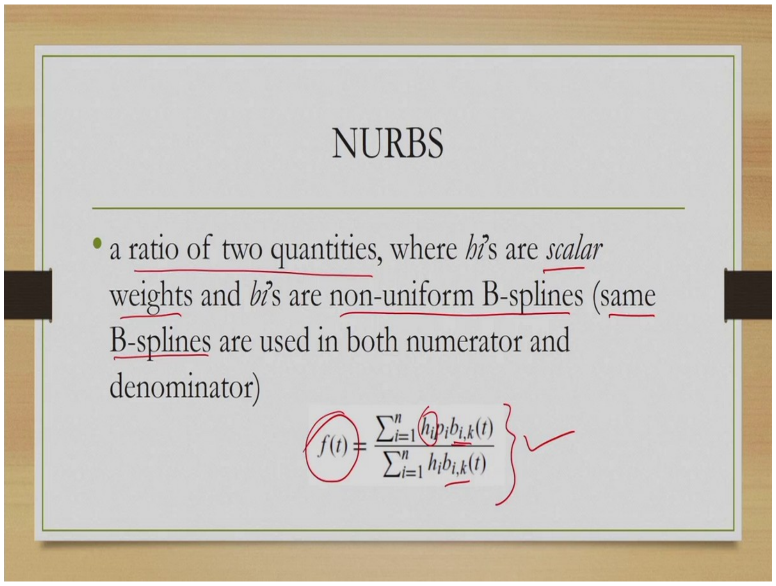


Non-Uniform B-Spline

- If the spacing between consecutive knots in the knot vector is not the same

So, when we have the knots that are uniformly spaced in the knot vector, like in the case of the example we have seen before, those B-spline are called uniform B-splines. If the knot vectors are not uniformly spaced, spacing between consecutive knots is not the same. Then we get non-uniform B spline.

(Refer Slide Time: 43:20)



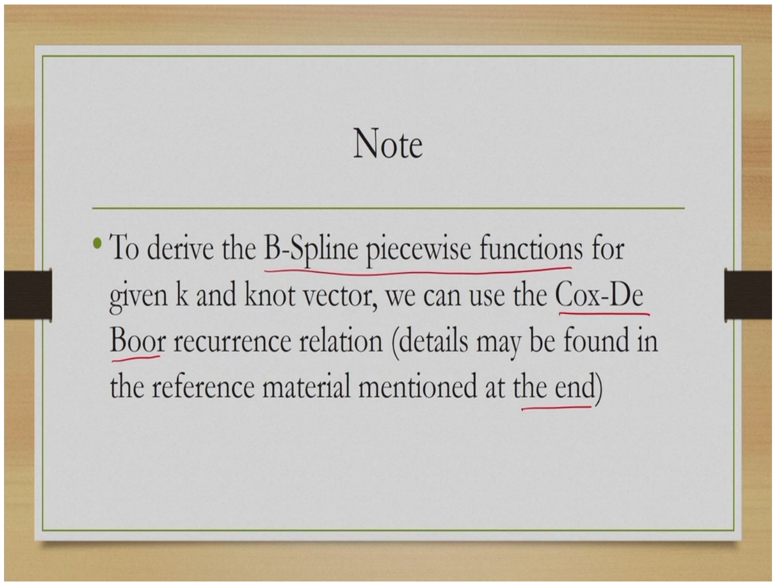
NURBS

- a ratio of two quantities, where *hi*'s are *scalar* weights and *bi*'s are non-uniform B-splines (same B-splines are used in both numerator and denominator)

$$f(t) = \frac{\sum_{i=1}^{n} h_i p_i b_{i,k}(t)}{\sum_{i=1}^{n} h_i b_{i,k}(t)}$$

And finally, nurbs referred to the B-splines which actually refers to ratio of two quantities shown here in this expression. Each i's are the scalar weights and bi's are non-uniform B-splines and also it should be noted that the same B-spline are used in both numerator and denominator. So, nurbs essentially refer to this ratio, so we have a uniform B-spline where knot points are uniformly spaced, we have non-uniform B-spline, where knot points are not uniformly spaced. And we have nurbs where we talk of ratio of two quantities, each of which is made up of same non-uniform B-spline.

(Refer Slide Time: 44:33)

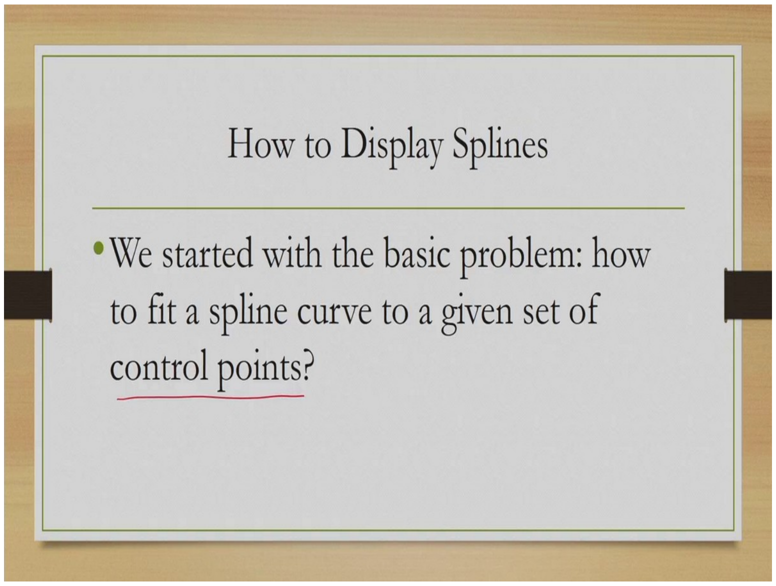


Note

- To derive the B-Spline piecewise functions for given k and knot vector, we can use the Cox-De Boor recurrence relation (details may be found in the reference material mentioned at the end)

Just for your information, although will not go into the details of these. That we can derive the piecewise functions for B-spline, but that is complicated and there is one recurrence relation we can use to do that called Cox de Boor recurrence relation. If you are interested, you may refer to the book and the material mentioned at the end of this lecture. So, far, we have discussed about different types of splines. Now, the basic idea of splines is that using this, we should be able to represent some curves and render it on the screen. Now how can we display spline curves?

(Refer Slide Time: 45:27)

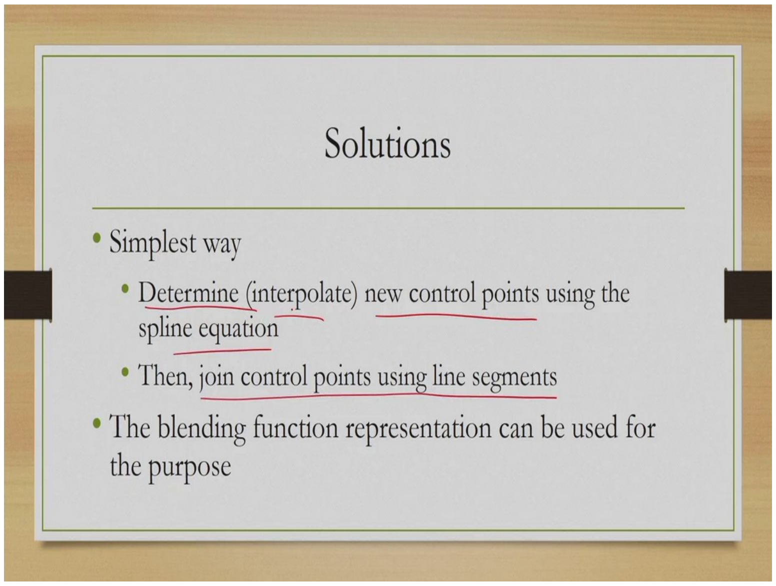


How to Display Splines

- We started with the basic problem: how to fit a spline curve to a given set of control points?

So, where we started how to fit a curve to a given set of control points, so we are given a set of control points and we want to fit a curve, we discussed how to use the idea of spline to get the curve.

(Refer Slide Time: 45:49)

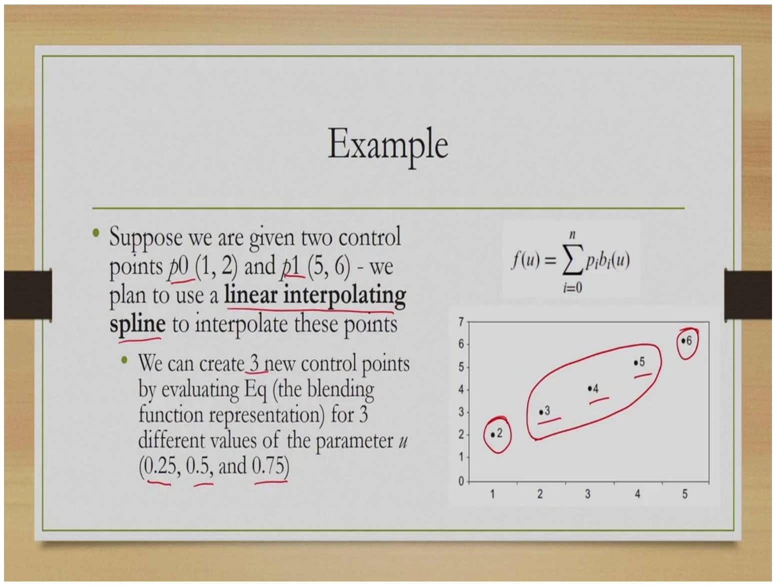


Solutions

- Simplest way
  - Determine (interpolate) new control points using the spline equation
  - Then, join control points using line segments
- The blending function representation can be used for the purpose

Now what we can do. So, we got the spline. Then what we can do the simplest way is to determine or interpolate new control points using the spline equation and then joined the control points using line segments. We can also use the blending function representation for interpolating these points.

(Refer Slide Time: 46:14)



Example

- Suppose we are given two control points $p0$ (1, 2) and $p1$ (5, 6) - we plan to use a **linear interpolating spline** to interpolate these points
  - We can create 3 new control points by evaluating Eq (the blending function representation) for 3 different values of the parameter u (0.25, 0.5, and 0.75)

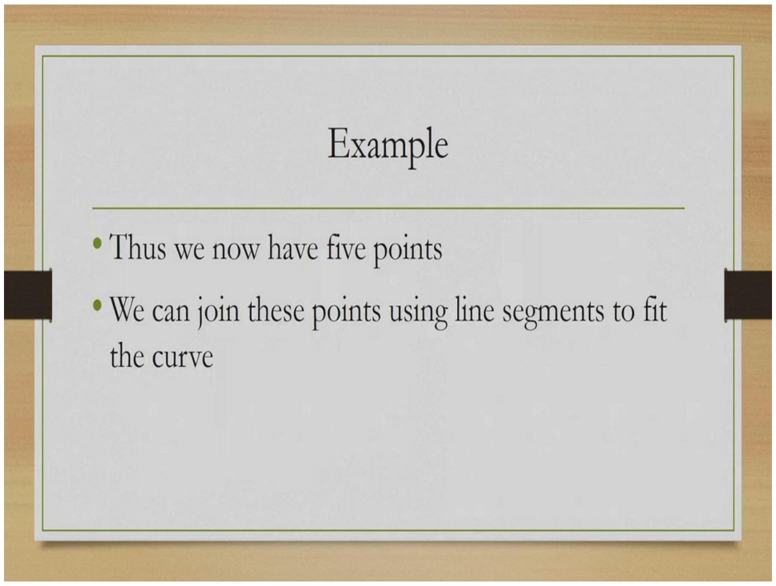
$$f(u) = \sum_{i=0}^{n} p_i b_i(u)$$

•2
•3
•4
•5
•6
0
1
2
3
4
5
6
7
1
2
3
4
5

As an example, suppose we are given two control points p0 and p1 this p0 and this is p1 since two control points out there we can use a linear interpolating spline and once we identify the spline representation using the approaches mentioned already, we can evaluate the spline function to create 3 new control points shown here. And we will get the parameter value for these control points defined here, and then we will get the actual function value.

So, we are given two control points, we are using a linear interpolating spline and using that spline, we are creating 3 new control points and once these points are created. We can join them using a line segment to get the curve.

(Refer Slide Time: 47:35)

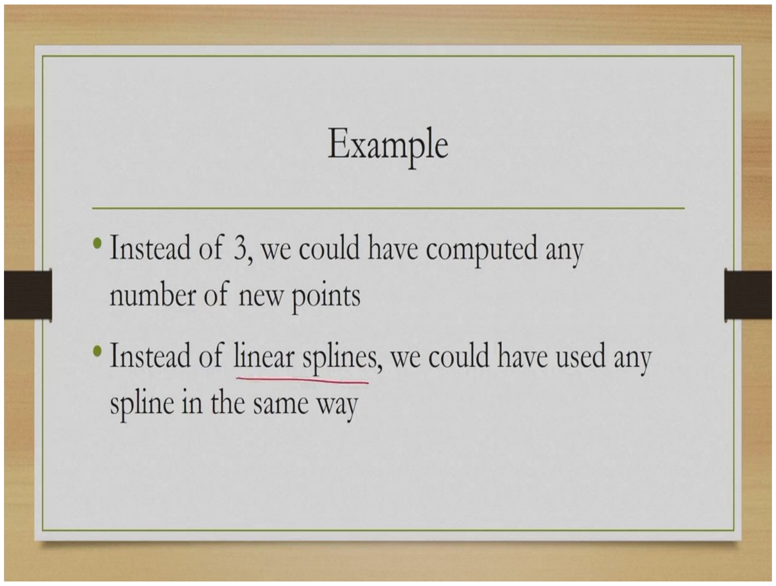


Example

- Thus we now have five points
- We can join these points using line segments to fit the curve

And we can keep on adding more and more points to get a final curve.

(Refer Slide Time: 47:42)

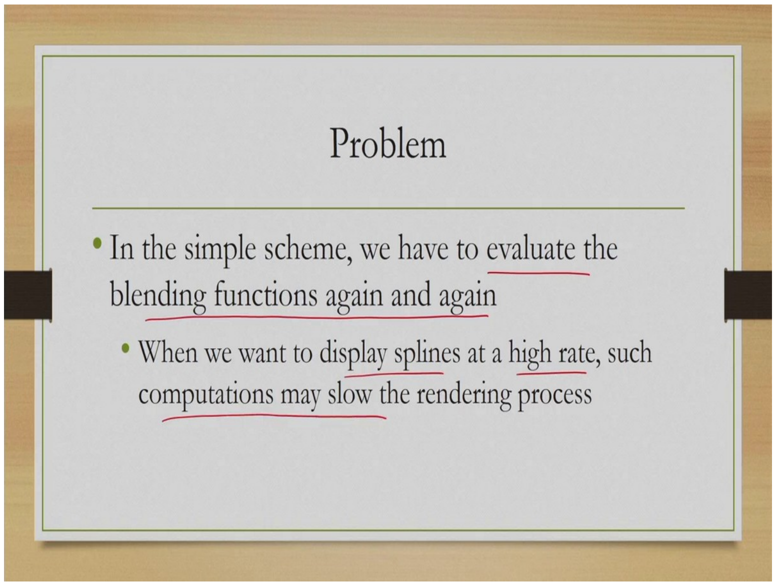


Example

- Instead of 3, we could have computed any number of new points
- Instead of linear splines, we could have used any spline in the same way

In fact, if more control points are there we can use more control points to get better splines, higher order splines instead of linear splines and do the same approach, follow the same approach. So, anything is possible.

(Refer Slide Time: 48:04)

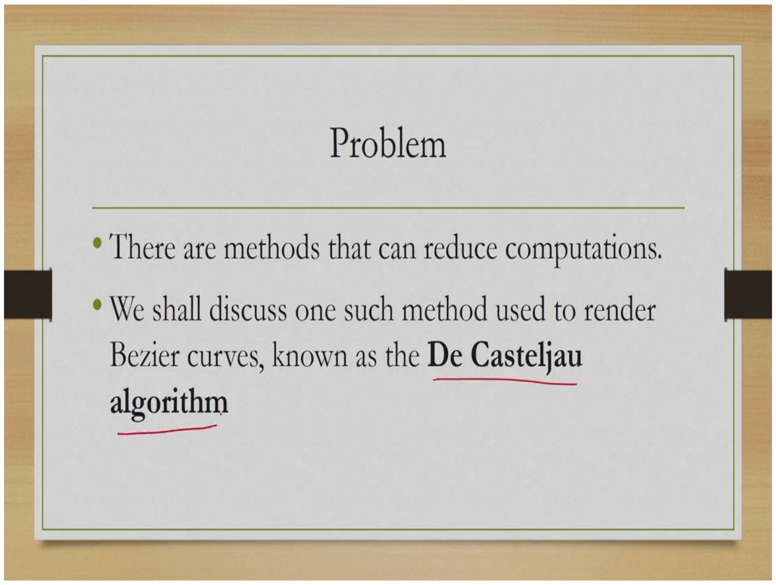


Problem

- In the simple scheme, we have to evaluate the blending functions again and again
  - When we want to display splines at a high rate, such computations may slow the rendering process

But the main problem is that we have to evaluate the blending function again and again. Now, when we want to display the splines at a very high rate, these computations may be costly and may slow the rendering process. So, we require some alternative approach.

(Refer Slide Time: 48:22)

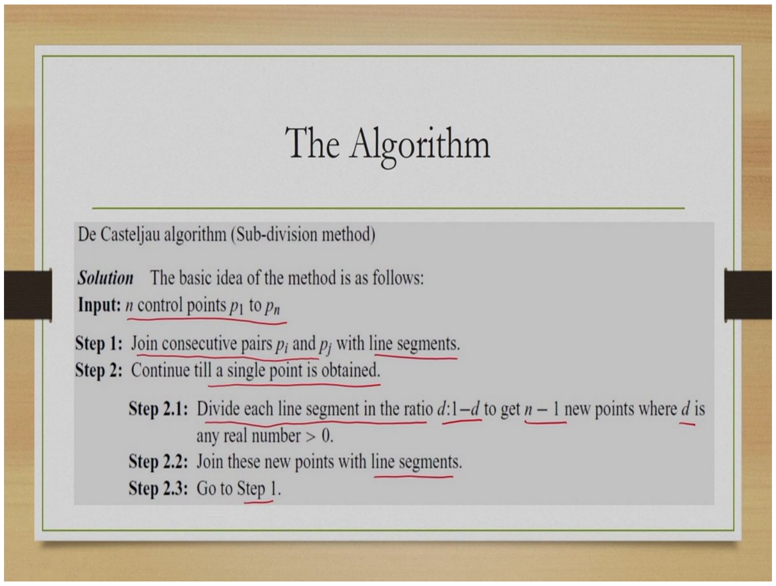


Problem

- There are methods that can reduce computations.
- We shall discuss one such method used to render Bezier curves, known as the **De Casteljau algorithm**

One of those approaches is known as De Casteljau algorithm. And we will have a quick look at the algorithm.

(Refer Slide Time: 48:36)



The Algorithm

De Casteljau algorithm (Sub-division method)

Solution The basic idea of the method is as follows:

Input: n control points p_1 to p_n

Step 1: Join consecutive pairs p_i and p_j with line segments.

Step 2: Continue till a single point is obtained.

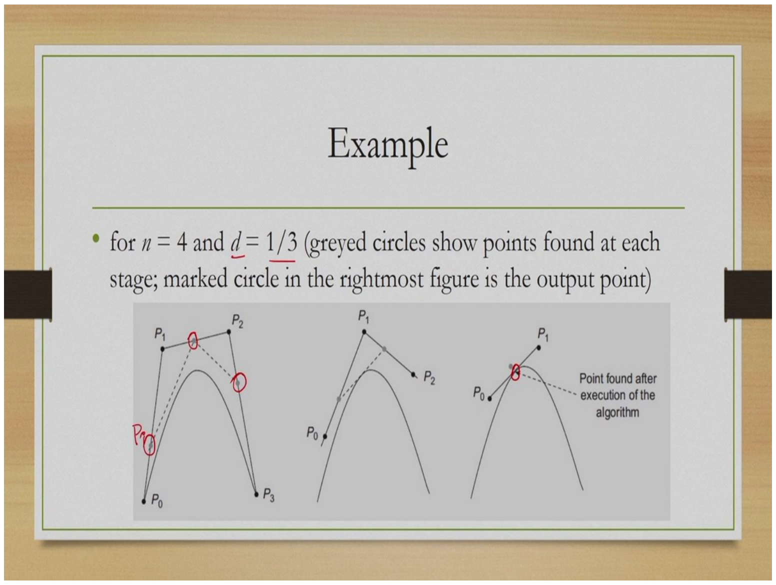
Step 2.1: Divide each line segment in the ratio $d{:}1-d$ to get $n-1$ new points where d is any real number > 0.

Step 2.2: Join these new points with line segments.

Step 2.3: Go to Step 1.

So, the algorithm consists of a few steps. They are n control points given. What we do, we join the consecutive pairs of control points with line segments and then we continue till a single point is obtained. What we do, we first divide each line segment in the ratio, d:1-d to get n-1 new points where d is any real number greater than 0. Now join these new points with line segments and then we go back to step 1.So, this is kind of look how it works.

(Refer Slide Time: 49:31)



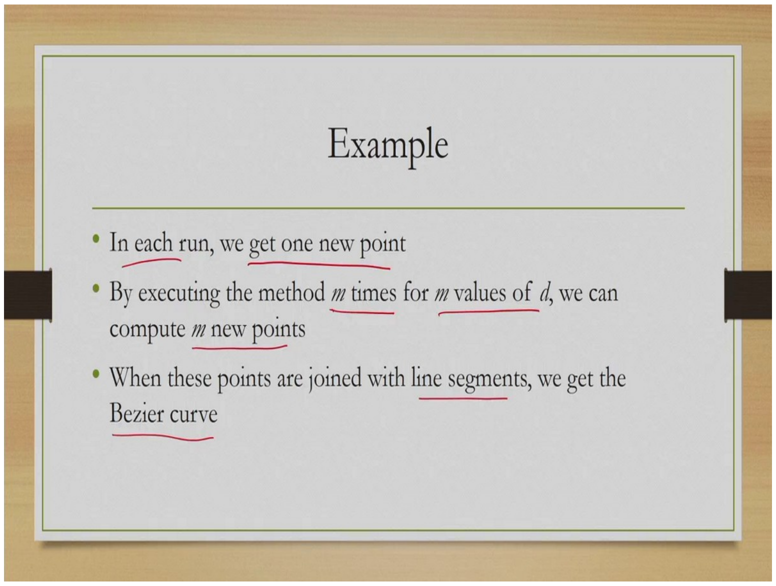
Example

- for $n = 4$ and $d = 1/3$ (greyed circles show points found at each stage; marked circle in the rightmost figure is the output point)

Let us see one example, suppose here n equal to 4 and we have decided on d to be one third. So, p0, p1, p2, p3 are the initial points, then at one third of the p are say p0 and p1 we took one point. Similarly, one third of the p are p1, p2 you took another point and then another point? And then these p0 has been modified to refer to this. Then we continue in the loop to finally get point here after the algorithm is executed

So, this division continues till we get one point. And at that point, we stop. So, that is the output point. This is, in short how we can use an iterative procedure to a simpler procedure to get points on the curve using the algorithm.

(Refer Slide Time: 51:01)



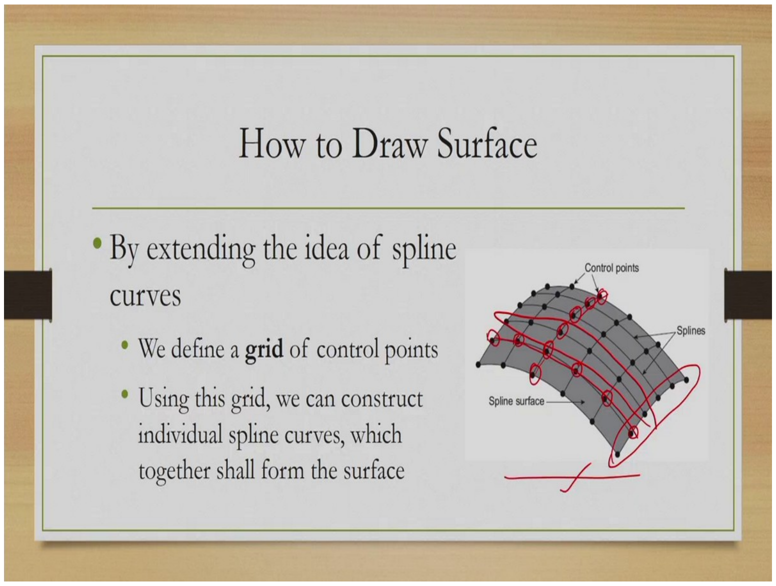
Example

- In each run, we get one new point
- By executing the method *m* times for *m* values of *d*, we can compute *m* new points
- When these points are joined with line segments, we get the Bezier curve

So, what we are doing in each run, we are getting a new point and we can execute it any time we want to get m new points, when these points are joined with line segments, we get the Bezier curves. So, here it has to be noted that we get this particular curve, not all Spline types. So, this algorithm actually refers to creation of a Bezier curves, which is an approximating spline. We can execute the algorithm m times to get the m points on the Bezier curve. So, that is how we can get the points and create the curve.

Now there is another thing in computer graphics, so we are not necessarily satisfied with only curves. What we need are surfaces also curved surfaces. So, using spline idea how can we get curved surfaces is very simple.

(Refer Slide Time: 52:13)



How to Draw Surface

- By extending the idea of spline curves
  - We define a **grid** of control points
  - Using this grid, we can construct individual spline curves, which together shall form the surface

Control points
Splines
Spline surface

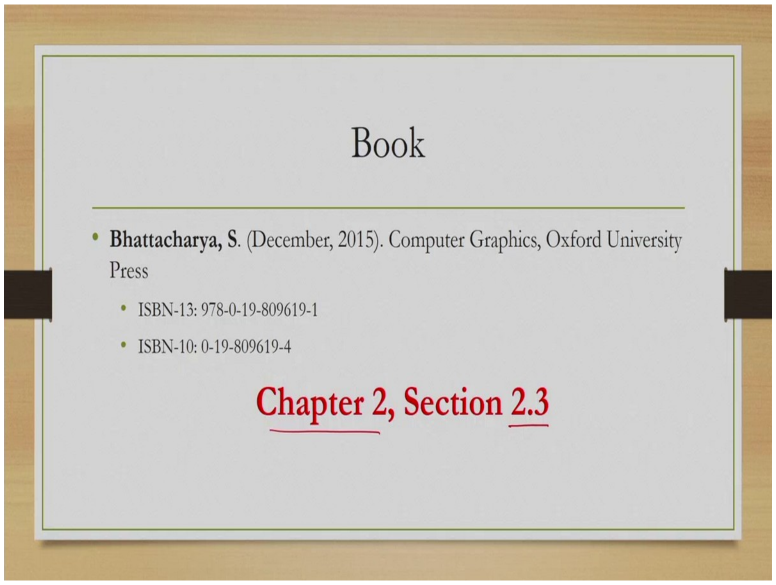
Look at this figure here. We can actually create a grid of control points like shown in this figure. So, each subset of the grid can be used to get a curve in different ways and when we put together these curves, we get a surface, so to get the surface, we define a grid shown in the field of circles here using one subset, we get one spline curve. Using another subset, we get another Spline curve which are intersecting, and when we get this grid of curves, we get the surface, the surface is defined by that grid. So, this is the simple way of getting a surface.

So, to recap today, we have learned about many new things. First of all, we learned different types of splines. We talked about 3 interpolating splines, natural cubic, Hermite cubic and Cardinal cubic. We also talked about two approximating splines, namely Bezier curves and B-splines.

The idea of B-splines is slightly complicated. And we explained it in terms of example. Then we touched upon the idea of how to display a spline curve simple approaches first get the curve spline equations or spline representation using that try to interpolate new points on the curve and join those points using line segments to get the final curve. But here evaluating the splines, equations at each new point may be time consuming, may not be efficient solution.

So, to avoid that some efficient algorithms are proposed. One of those is the De Casteljau method. Now, in this approach, we can get Bezier cubic curves using simple iterative procedure, which does not involve lots of computations. Also, it touched upon the idea of spline surfaces that is essentially creating set of curves and then using those curves to define the overall surface.

(Refer Slide Time: 55:10)



Book

- **Bhattacharya, S.** (December, 2015). Computer Graphics, Oxford University Press
  - ISBN-13: 978-0-19-809619-1
  - ISBN-10: 0-19-809619-4

Chapter 2, Section 2.3

Whatever we have discussed today can be found in this book, Chapter 2. So, you can refer to Section 2.3 of the book to get more details, including some topics that we have excluded from our discussion today. Namely, Cox de boor equations to get the B-spline blending functions. With this, we complete our discussion on spline. One more topic is left in the overall discussion on object representation which we will cover in the next lecture. Thank you and goodbye.