1 Introduction to Curves in Computer Graphics

1.1 Motivation

In computer graphics, curves are fundamental for representing a wide variety of shapes and paths. From the smooth outlines of characters in vector graphics to the trajectories of objects in animations, curves provide a flexible and intuitive way to define continuous forms. Unlike pixel-based raster images, vector graphics, which rely on mathematical descriptions of shapes, offer scalability without loss of quality, making curves indispensable in design and modeling tools.

1.2 Limitations of Polylines

A naive approach to representing curves might be to approximate them using a series of connected line segments, forming a polyline. While polylines are easy to store and render, they present significant drawbacks when it comes to design and manipulation. Consider editing a curve composed of many small line segments; adjusting the shape smoothly and intuitively becomes cumbersome. If we want a curve to pass through a specific point while maintaining smoothness, manipulating individual vertices of a polyline can lead to discontinuities and unnatural results. This lack of high-level control and smoothness makes polylines inadequate for many advanced graphics applications.

1.3 Parametric Representation of Curves and Surfaces

To overcome the limitations of polylines, computer graphics often employs parametric representations for curves and surfaces. In this approach, a curve is defined as a function of one or more parameters, typically denoted as t for curves and u, v for surfaces.

Definition 1.1 (Parametric Curve)**.** A parametric curve in 2D or 3D space is defined by a function $\gamma(t)$ that maps a parameter t from an interval (e.g., $[0, 1]$) to a point in 2D or 3D space.

$$\gamma(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} \quad \text{or} \quad \gamma(t) = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}$$

Similarly, surfaces can be represented parametrically using two parameters.

Definition 1.2 (Parametric Surface)**.** A parametric surface in 3D space is defined by a function $\beta(u, v)$ that maps two parameters u and v from a domain in 2D space to a point in 3D space.

$$\beta(u, v) = \begin{pmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{pmatrix}$$

While parametric representations are central to this chapter, it's worth noting that implicit representations offer an alternative. Implicit surfaces, for instance, are defined by an equation of the form $f(x, y, z) = 0$. However, we will primarily focus on parametric curves in this chapter.

1.4 Chapter Overview

This chapter will introduce Bezier curves and the fundamental concepts needed to understand and work with them. We will explore different basis functions used in curve representations, including monomial, Hermite, and Bernstein bases. We will also delve into the de Casteljau algorithm, a cornerstone for evaluating and subdividing Bezier curves. By the end of this chapter, you will gain a solid understanding of cubic curves and their mathematical foundations, equipping you with essential tools for curve design and manipulation in computer graphics.

2 Parametric Curves: Basic Concepts

2.1 What is a Parametrized Curve?

A parametrized curve is a way to describe a curve using a parameter, often denoted as t. As the parameter t varies over a certain range, the function $\gamma(t)$ traces out the points that form the curve. Think of it as a particle moving in space, where t represents time, and $\gamma(t)$ gives the particle's position at time t.

2.2 Primitives in Computer Graphics

In computer graphics, the term "primitives" refers to the basic building blocks used to construct more complex scenes and objects. For 2D graphics, common primitives include points, lines, triangles, and curves. These are the fundamental shapes that graphics systems are designed to handle efficiently. While lines and triangles are considered lower-level primitives, curves offer a higher-level primitive for design and modeling, which can then be converted into lower-level primitives (like polylines or triangles) for rendering.

2.3 Examples of Parametric Curves

2.3.1 Parametrized Line

Consider a line in 2D or 3D space. We can parametrize a line passing through a point $\vec{\gamma}_0$ and having a direction vector $\vec{d}$ as follows:

$$\gamma(t) = \vec{\gamma}_0 + t\vec{d} \tag{1}$$

Here, $\vec{\gamma}_0$ is the starting point of the line (when $t = 0$), and $\vec{d}$ is the direction vector that determines the line's orientation. As t varies, we move along the line.

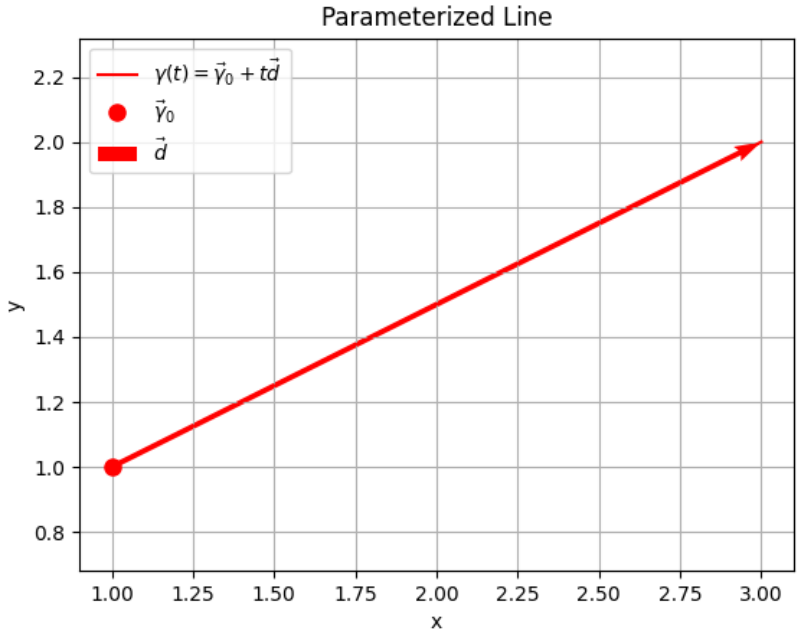

Figure 1: Parametrized Line

2.3.2 Parametrized Circle

A circle centered at the origin with radius 1 can be parametrized using trigonometric functions:

$$\gamma(t) = \begin{pmatrix} \cos(t) \\ \sin(t) \end{pmatrix} \tag{2}$$

Here, as t varies from 0 to 2π, the function $\gamma(t)$ traces out a circle. It is important to note that while this is a smooth curve, the functions $\cos(t)$ and $\sin(t)$ are not polynomials. Most of the curves we will discuss in detail in this chapter will be polynomial curves, but it's crucial to recognize that not all curves are polynomial.

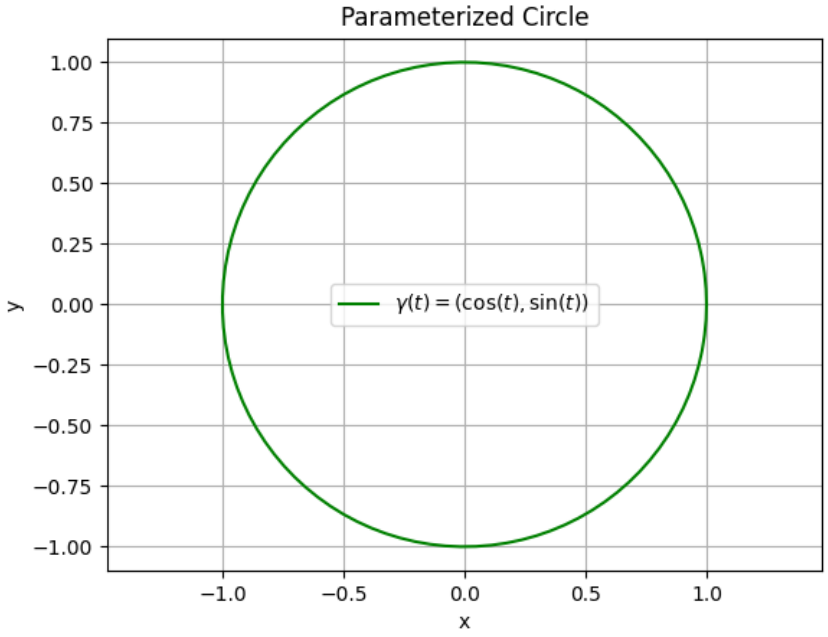

Figure 2: Parametrized Circle

2.4 Vector Graphics vs. Raster Graphics

The choice of curve representation is deeply tied to the type of graphics system being used: vector or raster.

- **Vector Graphics:** Represent images using mathematical descriptions of geometric shapes like points, lines, curves, and polygons. Vector graphics are resolution-independent, meaning they can be scaled up or down without losing quality. Formats like SVG, PDF, and formats used by drawing programs (e.g., Adobe Illustrator, Inkscape) are vector-based. Curves in vector graphics are typically represented parametrically using splines or Bezier curves.
- **Raster Graphics:** Represent images as a grid of pixels, where each pixel is assigned a color. Raster graphics are resolution-dependent; scaling them up can lead to pixelation. Formats like JPEG, PNG, and BMP are raster-based. Programs like MS Paint and image editing software like Adobe Photoshop often work with raster images.

Parametric curves are essential for vector graphics because they provide a compact and scalable way to represent smooth, complex shapes. When vector graphics are rendered for display, these curves are often tessellated into simpler primitives (like lines or triangles) which are then rasterized (converted into pixels).

3 Basis Functions and Curve Representation

3.1 The Need for Basis Functions

Representing curves directly using polynomial equations can become complex and unwieldy, especially when we need to manipulate or design curves interactively. Basis functions offer a more structured and manageable approach. Instead of defining a curve by its polynomial coefficients directly, we define it as a linear combination of simpler, pre-defined functions called basis functions. This method simplifies curve construction, manipulation, and evaluation.

3.2 General Formula

In the basis function approach, a curve $\gamma(t)$ is expressed as a sum of basis functions $\phi_i(t)$ multiplied by coefficients a_i:

$$\gamma(t) = \sum_{i=0}^{n} a_i \phi_i(t) \tag{3}$$

Here, $\{\phi_0(t), \phi_1(t), \ldots, \phi_n(t)\}$ is the set of basis functions, and $\{a_0, a_1, \ldots, a_n\}$ are the coefficients. The choice of basis functions $\phi_i(t)$ determines the properties of the curves we can represent and influences how easily we can control and manipulate them.

3.3 Monomial Basis

The monomial basis is perhaps the most straightforward set of basis functions, consisting of powers of t.

Definition 3.1 (Monomial Basis)**.** The monomial basis functions are given by:

$$\phi_i(t) = t^i, \quad i = 0, 1, 2, 3, \ldots$$

For cubic polynomials (degree 3), the monomial basis functions are $\{1, t, t^2, t^3\}$.

A cubic polynomial $f(t)$ in the monomial basis is expressed as:

$$f(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0 \tag{4}$$

where a_0, a_1, a_2, a_3 are the coefficients.

3.3.1 Example: Representing a Polynomial in Monomial Basis

Consider the polynomial $f(t) = 1 - t + t^2$. In the monomial basis, this polynomial is represented by the coefficients $a_0 = 1$, $a_1 = -1$, $a_2 = 1$, and $a_3 = 0$. We can represent these coefficients as a vector:

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 1 \\ 0 \end{pmatrix}$$

3.3.2 Matrix Representation

We can express a set of functions, say $x(t), y(t), z(t), w(t)$, in matrix form using the monomial basis. For example, if we have:

$$
\begin{aligned}
x(t) &= 1 + t^2 \\
y(t) &= 1 + t + t^2 \\
z(t) &= 0 \\
w(t) &= 0
\end{aligned}
$$

We can write this in matrix form as:

$$
\begin{pmatrix} x(t) \\ y(t) \\ z(t) \\ w(t) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}
$$

Here, the matrix contains the coefficients, and the vector contains the monomial basis functions $\left(1, t, t^2, t^3\right)^T$.

3.3.3 Drawbacks of Monomial Basis

Remark 3.2. While the monomial basis is simple to understand, it has several drawbacks in the context of curve design and computer graphics:

- **Lack of Intuitive Control:** Modifying the coefficients a_i in the monomial form does not provide an intuitive way to control the shape of the curve. It's not immediately clear how changing a coefficient will affect the curve's form.
- **Difficulty in Interpolation:** Solving interpolation problems, such as finding a curve that passes through specific points or has certain tangent properties, becomes algebraically complex in the monomial basis.
- **Numerical Instability:** For higher-degree polynomials, the monomial basis can lead to numerical instability due to the large range of values that powers of t can take, especially for t outside the interval $[0, 1]$.

These limitations motivate the need for alternative basis functions that offer better control, stability, and ease of computation for curve design and manipulation.

4 Hermite Curves

4.1 Motivation: Specifying Points and Tangents

In interactive curve design, it is often natural to specify a curve by defining its endpoints and the tangents at those endpoints. This approach provides intuitive control over the curve's shape. Hermite curves are designed precisely for this purpose. They allow us to construct a cubic curve segment given two endpoints and the tangent vectors at those points.

4.2 The Hermite Interpolation Problem

The Hermite interpolation problem for cubic curves can be stated as follows:

Problem: Given two points $\vec{p}_0$ and $\vec{p}_1$ and tangent vectors $\vec{m}_0$ and $\vec{m}_1$ at $\vec{p}_0$ and $\vec{p}_1$ respectively, find a cubic polynomial curve $\gamma(t)$ such that:

$$
\begin{aligned}
\gamma(0) &= \vec{p}_0 \\
\gamma(1) &= \vec{p}_1 \\
\gamma'(0) &= \vec{m}_0 \\
\gamma'(1) &= \vec{m}_1
\end{aligned}
$$

Here, $\gamma'(t)$ denotes the derivative of $\gamma(t)$ with respect to t, representing the tangent vector to the curve at parameter t.

4.3 Derivation of Hermite Basis Functions

Let's consider a cubic polynomial in monomial form:

$$p(t) = at^3 + bt^2 + ct + d \tag{5}$$

Its derivative is:

$$p'(t) = 3at^2 + 2bt + c \tag{6}$$

We want to determine the coefficients a, b, c, d such that the curve satisfies the Hermite interpolation conditions. Let h_0, h_1, h_2, h_3 represent the values $p(0), p(1), p'(0), p'(1)$ respectively. Applying the conditions, we get a system of linear equations:

$$\begin{aligned} p(0) &= d = h_0 \\ p(1) &= a + b + c + d = h_1 \\ p'(0) &= c = h_2 \\ p'(1) &= 3a + 2b + c = h_3 \end{aligned}$$

In matrix form, this system is:

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{pmatrix}$$

To solve for a, b, c, d in terms of h_0, h_1, h_2, h_3, we can invert the matrix and multiply by the vector $\begin{pmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{pmatrix}$.

Alternatively, we can solve the system directly. From the first and third equations, we immediately have $d = h_0$ and $c = h_2$. Substituting these into the second and fourth equations, we get:

$$\begin{aligned} a + b + h_2 + h_0 &= h_1 \\ 3a + 2b + h_2 &= h_3 \end{aligned}$$

Rearranging:

$$\begin{aligned} a + b &= h_1 - h_0 - h_2 \\ 3a + 2b &= h_3 - h_2 \end{aligned}$$

Multiply the first equation by 2 and subtract it from the second:

$$(3a + 2b) - 2(a + b) = (h_3 - h_2) - 2(h_1 - h_0 - h_2)$$

$$a = h_3 - h_2 - 2h_1 + 2h_0 + 2h_2 = 2h_0 - 2h_1 + h_2 + h_3$$

Substitute a back into $a + b = h_1 - h_0 - h_2$:

$$b = (h_1 - h_0 - h_2) - a = (h_1 - h_0 - h_2) - (2h_0 - 2h_1 + h_2 + h_3) = -3h_0 + 3h_1 - 2h_2 - h_3$$

Thus we have:

$$\begin{aligned} a &= 2h_0 - 2h_1 + h_2 + h_3 \\ b &= -3h_0 + 3h_1 - 2h_2 - h_3 \\ c &= h_2 \\ d &= h_0 \end{aligned}$$

Substituting these coefficients back into $p(t) = at^3 + bt^2 + ct + d$ and regrouping terms with respect to h_0, h_1, h_2, h_3, we obtain:

$$\begin{aligned} p(t) &= (2h_0 - 2h_1 + h_2 + h_3)t^3 + (-3h_0 + 3h_1 - 2h_2 - h_3)t^2 + h_2 t + h_0 \\ &= h_0(2t^3 - 3t^2 + 1) + h_1(-2t^3 + 3t^2) + h_2(t^3 - 2t^2 + t) + h_3(t^3 - t^2) \end{aligned}$$

So, we can express $p(t)$ as a linear combination of Hermite basis functions $h_i(t)$ and the values h_i:

$$p(t) = h_0 h_0(t) + h_1 h_1(t) + h_2 h_2(t) + h_3 h_3(t) \tag{7}$$

where $h_0(t) = 2t^3 - 3t^2 + 1$, $h_1(t) = -2t^3 + 3t^2$, $h_2(t) = t^3 - 2t^2 + t$, and $h_3(t) = t^3 - t^2$.

4.4 The Hermite Basis Functions

The Hermite basis functions for cubic curves are:

$$\begin{aligned} h_0(t) &= 2t^3 - 3t^2 + 1 \\ h_1(t) &= -2t^3 + 3t^2 \\ h_2(t) &= t^3 - 2t^2 + t \\ h_3(t) &= t^3 - t^2 \end{aligned}$$

These basis functions have the following properties:

- $h_0(0) = 1, h_0(1) = 0, h_0'(0) = 0, h_0'(1) = 0$
- $h_1(0) = 0, h_1(1) = 1, h_1'(0) = 0, h_1'(1) = 0$
- $h_2(0) = 0, h_2(1) = 0, h_2'(0) = 1, h_2'(1) = 0$
- $h_3(0) = 0, h_3(1) = 0, h_3'(0) = 0, h_3'(1) = 1$

These properties ensure that when we form the curve $p(t) = h_0 h_0(t) + h_1 h_1(t) + h_2 h_2(t) + h_3 h_3(t)$, we automatically satisfy the Hermite interpolation conditions: $p(0) = h_0$, $p(1) = h_1$, $p'(0) = h_2$, and $p'(1) = h_3$.

4.5 Advantage of Hermite Basis

Remark 4.1. The primary advantage of the Hermite basis is that it directly solves the Hermite interpolation problem. It provides an intuitive way to construct curves by specifying endpoint positions and tangents, making it suitable for design interfaces where artists and designers want to control curve shapes in this manner.

4.6 Example: Different Parametrizations of the Same Curve

Consider the line segment from $(0, 0)$ to $(1, 2)$. We can parametrize this line in multiple ways using cubic Hermite curves. Two possible parametrizations are:

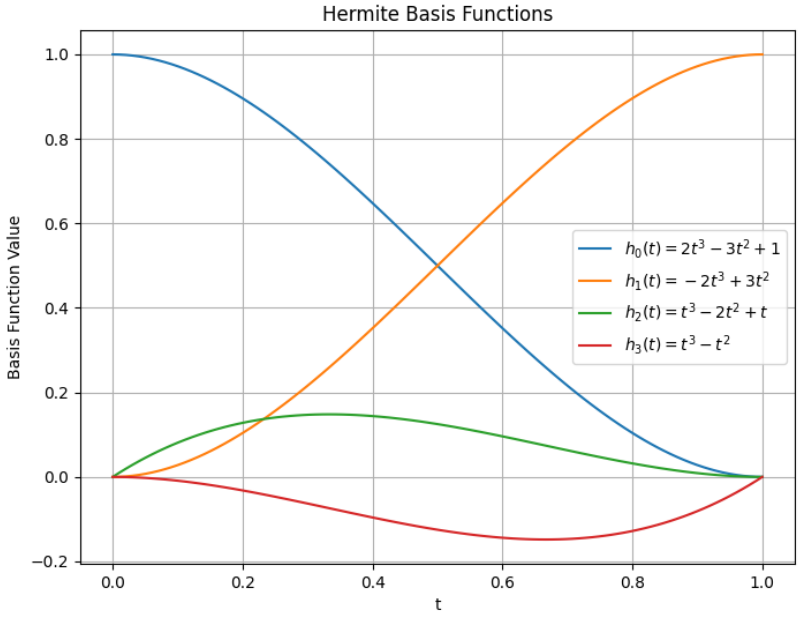

Figure 3: Hermite Basis Functions

1. **Constant Speed Parametrization:** Let $\vec{p}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $\vec{p}_1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$, $\vec{m}_0 = \vec{m}_1 = \vec{p}_1 - \vec{p}_0 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$. Using the Hermite basis, we get a parametrization $\gamma_1(t)$.

2. **Non-Constant Speed Parametrization:** Let $\vec{p}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $\vec{p}_1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$, $\vec{m}_0 = \vec{m}_1 = 0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$. Using the Hermite basis with these tangents, we get another parametrization $\gamma_2(t)$. In this case, the "car" slows down at the origin.

Both $\gamma_1(t)$ and $\gamma_2(t)$ trace out the same geometric line segment, but they traverse it at different "speeds". This illustrates that parametric representations of curves are not unique, and different parametrizations can lead to different dynamic properties even for the same geometric shape.

5 Bezier Curves and the de Casteljau Algorithm

5.1 Cubic Blossoms

The concept of blossoms provides a deeper understanding of polynomial curves and leads to efficient algorithms like the de Casteljau algorithm for evaluating Bezier curves.

Definition 5.1 (Cubic Blossom). For a cubic polynomial $f(t)$, its cubic blossom is a function $F(t_1, t_2, t_3)$ of three variables that satisfies the following properties:

1. **Symmetry:** $F(t_1, t_2, t_3)$ is symmetric in its arguments, i.e., $F(t_1, t_2, t_3) = F(t_{\sigma(1)}, t_{\sigma(2)}, t_{\sigma(3)})$ for any permutation σ of $\{1, 2, 3\}$.

2. **Affine Property:** F is affine in each argument. For example, for the first argument:

$$F(\alpha t_1 + \beta t_1', t_2, t_3) = \alpha F(t_1, t_2, t_3) + \beta F(t_1', t_2, t_3)$$

where $\alpha + \beta = 1$. The same holds for the second and third arguments.

3. **Diagonal Property:** $F(t, t, t) = f(t)$ for all t.

For every cubic polynomial, there exists a unique cubic blossom. Let's derive the cubic blossoms for monomial basis functions:

5.1.1 Blossom for $f(t) = 1$

The blossom is simply:

$$F(t_1, t_2, t_3) = 1$$

This satisfies all three properties: symmetry, affine property, and diagonal property.

5.1.2 Blossom for $f(t) = t$

The blossom is:

$$F(t_1, t_2, t_3) = \frac{t_1 + t_2 + t_3}{3}$$

This is symmetric and affine in each argument. For the diagonal property: $F(t, t, t) = \frac{t+t+t}{3} = t = f(t)$.

5.1.3 Blossom for $f(t) = t^2$

The blossom is:

$$F(t_1, t_2, t_3) = \frac{t_1 t_2 + t_1 t_3 + t_2 t_3}{3}$$

Again, symmetry and affine properties are satisfied. For the diagonal property: $F(t, t, t) = \frac{t^2+t^2+t^2}{3} = t^2 = f(t)$.

5.1.4 Blossom for $f(t) = t^3$

The blossom is:

$$F(t_1, t_2, t_3) = t_1 t_2 t_3$$

This is symmetric and affine in each argument. For the diagonal property: $F(t, t, t) = t \cdot t \cdot t = t^3 = f(t)$.

5.1.5 Generalization

For any cubic polynomial $f(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0$, its blossom $F(t_1, t_2, t_3)$ can be found by linearly combining the blossoms of $1, t, t^2, t^3$ with coefficients a_0, a_1, a_2, a_3:

$$F(t_1, t_2, t_3) = a_3(t_1 t_2 t_3) + a_2 \left(\frac{t_1 t_2 + t_1 t_3 + t_2 t_3}{3} \right) + a_1 \left(\frac{t_1 + t_2 + t_3}{3} \right) + a_0(1)$$

5.2 Geometric Interpretation of Bezier Curves

A cubic Bezier curve is defined by four control points, say $\vec{b}_0, \vec{b}_1, \vec{b}_2, \vec{b}_3$. These control points are geometrically related to the blossom of the Bezier curve. In fact, these control points are blossom values at specific parameter values:

$$\vec{b}_0 = F(0, 0, 0)$$
$$\vec{b}_1 = F(0, 0, 1)$$
$$\vec{b}_2 = F(0, 1, 1)$$
$$\vec{b}_3 = F(1, 1, 1)$$

The cubic Bezier curve $\gamma(t)$ is then given by the diagonal values of the blossom: $\gamma(t) = F(t,t,t)$. The tangent vectors at the endpoints of a Bezier curve can also be expressed in terms of its control points. For a cubic Bezier curve defined by control points $\vec{b}_0, \vec{b}_1, \vec{b}_2, \vec{b}_3$:

$$\gamma'(0) = 3(\vec{b}_1 - \vec{b}_0)$$
$$\gamma'(1) = 3(\vec{b}_3 - \vec{b}_2)$$

This relationship highlights how the first and last segments of the Bezier control polygon influence the tangents at the curve's endpoints.

5.3 The de Casteljau Algorithm

The de Casteljau algorithm is an efficient and geometrically intuitive method for evaluating a Bezier curve at a specific parameter value t. It is based on repeated linear interpolation of the control points.

5.3.1 Algorithm Steps

Given control points $\vec{b}_0, \vec{b}_1, \vec{b}_2, \vec{b}_3$ and a parameter value $t \in [0,1]$, the de Casteljau algorithm proceeds as follows:

Level 1 Interpolation:

$$\vec{b}_0^{(1)} = (1-t)\vec{b}_0 + t\vec{b}_1$$
$$\vec{b}_1^{(1)} = (1-t)\vec{b}_1 + t\vec{b}_2$$
$$\vec{b}_2^{(1)} = (1-t)\vec{b}_2 + t\vec{b}_3$$

These steps compute points that are t fraction of the way from $\vec{b}_0$ to $\vec{b}_1$, $\vec{b}_1$ to $\vec{b}_2$, and $\vec{b}_2$ to $\vec{b}_3$, respectively.

Level 2 Interpolation:

$$\vec{b}_0^{(2)} = (1-t)\vec{b}_0^{(1)} + t\vec{b}_1^{(1)}$$
$$\vec{b}_1^{(2)} = (1-t)\vec{b}_1^{(1)} + t\vec{b}_2^{(1)}$$

These steps interpolate between the points computed in Level 1.

Level 3 Interpolation:

$$\vec{b}_0^{(3)} = (1-t)\vec{b}_0^{(2)} + t\vec{b}_1^{(2)} \tag{8}$$

The final point $\vec{b}_0^{(3)}$ is the point on the Bezier curve at parameter t, i.e., $\gamma(t) = \vec{b}_0^{(3)}$.

5.3.2 Relationship to Cubic Blossoms

Remark 5.2. Each linear interpolation step in the de Casteljau algorithm corresponds to evaluating an affine combination of blossom values. For example, $\vec{b}_0^{(1)} = (1-t)\vec{b}_0 + t\vec{b}_1$ is equivalent to $F(0,0,t) = (1-t)F(0,0,0) + tF(0,0,1)$ due to the affine property of blossoms. The algorithm recursively computes blossom values with mixed parameters, eventually arriving at $F(t,t,t) = \gamma(t)$.

5.3.3 Advantage: Subdivision and Sampling

Remark 5.3. The de Casteljau algorithm is not only useful for evaluating points on a Bezier curve, but also for subdivision. At each level of interpolation, the points $\vec{b}_0^{(r)}, \vec{b}_1^{(r)}, \ldots$ form the control points of new Bezier curves. Specifically, at $t = 0.5$, $\{\vec{b}_0, \vec{b}_0^{(1)}, \vec{b}_0^{(2)}, \vec{b}_0^{(3)}\}$ and $\{\vec{b}_0^{(3)}, \vec{b}_1^{(2)}, \vec{b}_2^{(1)}, \vec{b}_3\}$ are control points for two Bezier curves that together form the original curve, subdivided at $t = 0.5$.

This subdivision property is crucial for adaptive tessellation and curve rendering. The de Casteljau

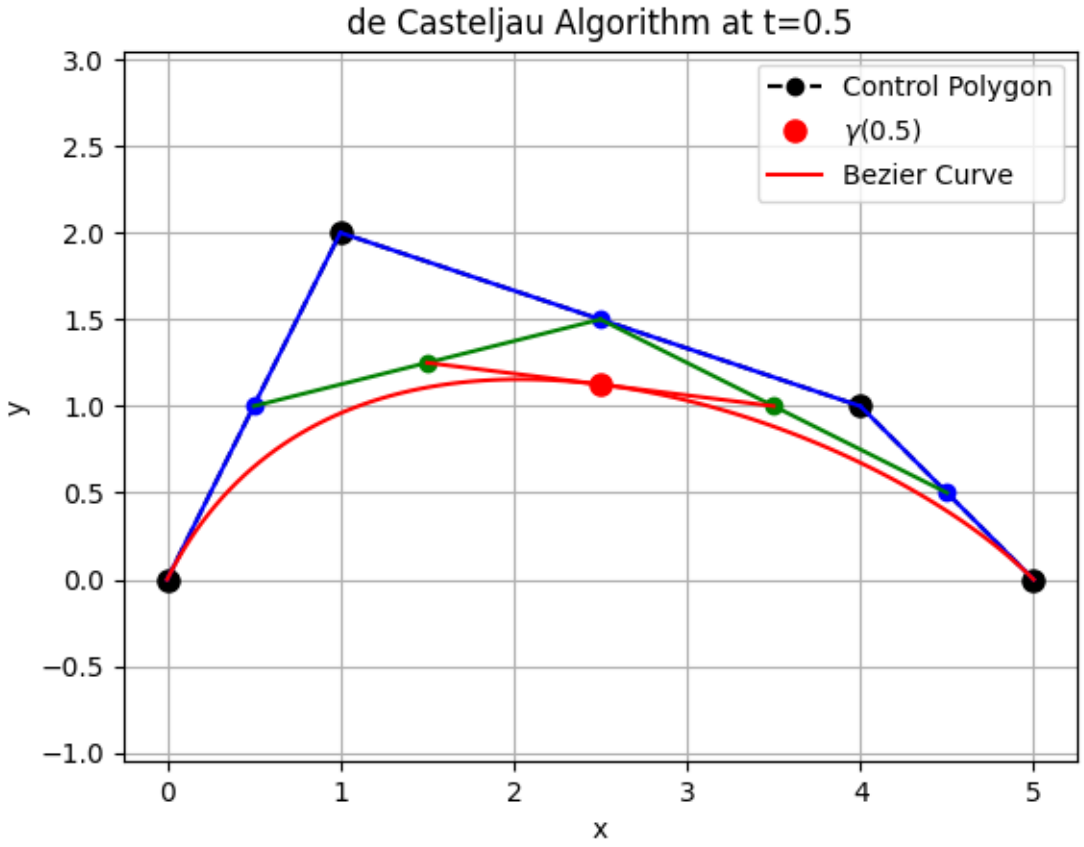

Figure 4: de Casteljau Algorithm at t=0.5

algorithm also provides a stable and efficient way to sample points along the Bezier curve, which is valuable for converting vector curves into raster approximations.

5.4 Bernstein Basis (Optional)

The Bernstein basis functions for cubic Bezier curves are defined as:

$$
\begin{aligned}
B_0^3(t) &= (1-t)^3 \\
B_1^3(t) &= 3t(1-t)^2 \\
B_2^3(t) &= 3t^2(1-t) \\
B_3^3(t) &= t^3
\end{aligned}
$$

A cubic Bezier curve with control points $\vec{b}_0, \vec{b}_1, \vec{b}_2, \vec{b}_3$ can be expressed in the Bernstein basis as:

$$\gamma(t) = \sum_{i=0}^{3} \vec{b}_i B_i^3(t) = \vec{b}_0 B_0^3(t) + \vec{b}_1 B_1^3(t) + \vec{b}_2 B_2^3(t) + \vec{b}_3 B_3^3(t) \tag{9}$$

5.4.1 Conversion Matrix

The conversion matrix from the monomial basis $\begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}$ to the Bernstein basis $\begin{pmatrix} B_0^3(t) \\ B_1^3(t) \\ B_2^3(t) \\ B_3^3(t) \end{pmatrix}$ is:

$$M_{B \leftarrow M} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix}$$

And the inverse matrix, from Bernstein to Monomial basis is:

$$M_{M \leftarrow B} = M_{B \leftarrow M}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1/3 & 0 & 0 \\ 1 & 2/3 & 1/3 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

5.5 Comparison of Bases

	Monomial Basis	Hermite Basis	Bernstein Basis
Basis Functions	$\{1, t, t^2, t^3\}$	$\{h_0(t), h_1(t), h_2(t), h_3(t)\}$	$\{B_0^3(t), B_1^3(t), B_2^3(t), B_3^3(t)\}$
Control	Coefficients a_i	Values and Tangents at Endpoints	Bezier Control Points
Intuitive Control	Low	Medium	High (Geometric)
Interpolation	Algebraically Complex	Direct (by definition)	Geometrically via de Casteljau
Evaluation	Direct Polynomial Evaluation	Direct Basis Summation	de Casteljau Algorithm
Subdivision	Complex	Not naturally suited	de Casteljau Algorithm
Use Cases	Mathematical Analysis	Curve Interpolation, CAD	Vector Graphics, Curve Design

Table 1: Comparison of Cubic Curve Bases

Remark 5.4. Each basis serves different purposes and provides unique advantages. The Monomial basis is simple mathematically but lacks intuitive geometric control. The Hermite basis directly addresses endpoint interpolation. The Bernstein basis, together with the de Casteljau algorithm, offers geometric intuition, stability, and efficient evaluation and subdivision, making it particularly suitable for computer graphics applications.

6 Conclusion

6.1 Review

In this chapter, we explored the representation of cubic curves in computer graphics, focusing on three different basis functions: monomial, Hermite, and Bernstein. Each basis provides a unique way to define and manipulate cubic curves, suitable for various applications and computational tasks. We highlighted the de Casteljau algorithm as a central tool for evaluating and subdividing Bezier curves, emphasizing its geometric elegance and efficiency. Understanding these foundations is crucial for further exploration into splines and advanced curve and surface modeling techniques in computer graphics.

6.2 Preview

In the next lecture, we will build upon the concepts of cubic curves to explore splines. Splines are constructed by joining multiple curve segments together smoothly, allowing for the creation of more complex and versatile curves. We will investigate techniques for ensuring continuity and smoothness at the joints between curve segments, paving the way for modeling intricate shapes and surfaces.