ETH zürich

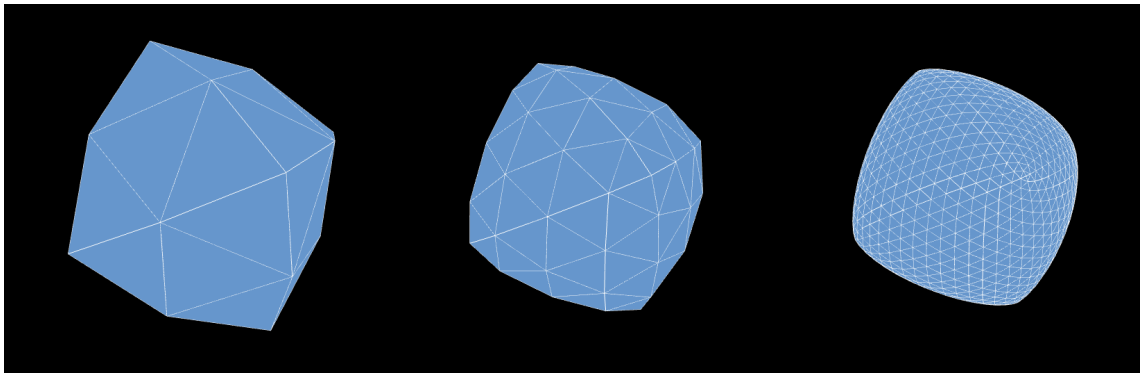
computer graphics laboratory

Prof. Dr. Markus Gross

Visual Computing

Exercise 12: Curves & Surfaces

Hand-out Date: 10 December 2024



Goals

- Understand B-Spline Curves.
- Understand the principles of Mesh Subdivision.

Resources

The lecture slides and exercise slides are accessible via Moodle and GitHub.

Tasks

1. B-spline Curve

In this task, we will construct a B-Spline step by step to understand its formulation. Recall that a B-spline curve is defined by its degree, a set of control points, and a knot vector. The control points influence the shape of the curve, whereas the knot vector is a sequence of values that determine the parametrization of the B-spline curve. Specifically, ti divides the curve into segments, which are associated to a set of control points. Concerning the degree, in this exercise the focus are quadratic splines.

Suppose we are given four points $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$, and 5 knots t_0, t_1, t_2, t_3, and t_4. The knots are ordered such that $t_0 \leq t_1 < t_2 < t_3 \leq t_4$.

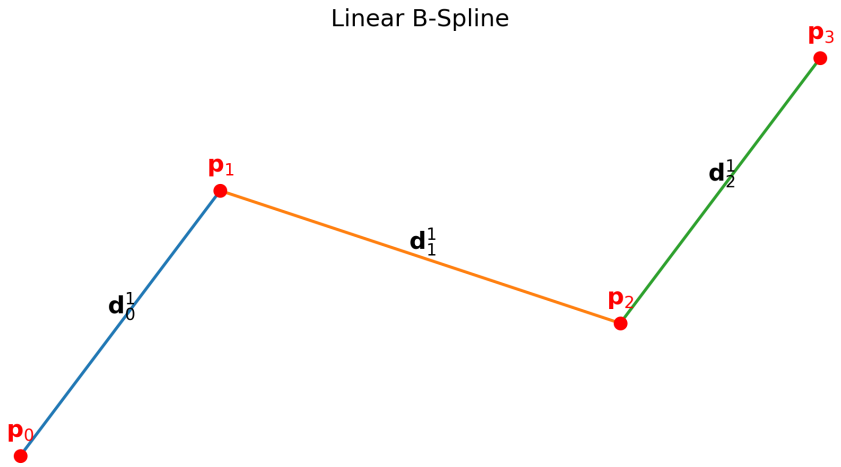


Figure 1: Linear B-spline.

Linear B-spline. Given two points $\mathbf{p}_0$ and $\mathbf{p}_1$, as well as the knots t_0 and t_1, the convex combination of these two points gives a line segment as

$$\mathbf{d}_0^1\left(t \mid \mathbf{p}_0, \mathbf{p}_1; t_0, t_1\right) = \frac{t_1 - t}{t_1 - t_0}\mathbf{p}_0 + \frac{t - t_0}{t_1 - t_0}\mathbf{p}_1, \quad t \in [t_0, t_1]\,, \tag{1}$$

where the superscript 1 means the degree is one. Please, write down the formulation for the other two consecutive line segments $\mathbf{d}_1^1$ and $\mathbf{d}_2^1$. Then, assemble the formula for the linear B-spline comprised of these line segments. This construction can be generalized to produce smooth, piece-wise polynomial curves of higher degrees.

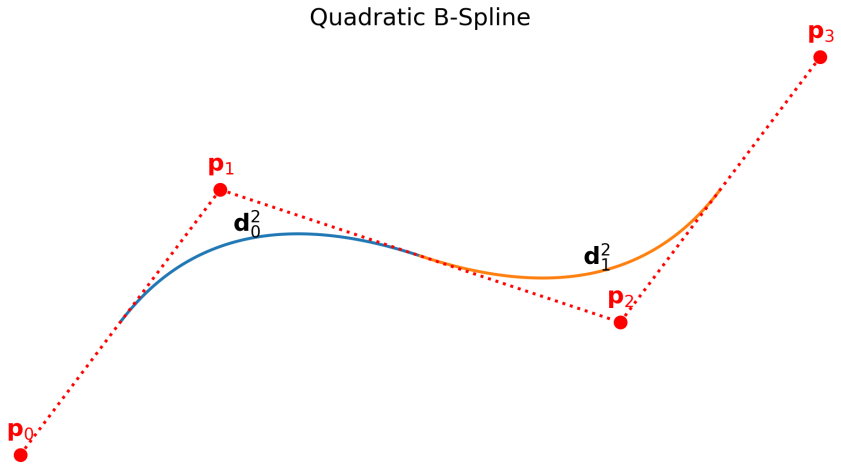


Figure 2: Quadratic B-spline.

Quadratic B-spline. Now, we consider a curve of curves of one degree higher (quadratic). The construction of quadratic spline curves is based on the repeated

averaging of two consecutive line segments. Formally, for two consecutive line segments $\mathbf{d}_0^1$ and $\mathbf{d}_1^1$, we can define the quadratic B-Spine curve as

$$\mathbf{d}_0^2\left(t \mid \mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2; t_0, t_1, t_2, t_3\right) = \frac{t_2 - t}{t_2 - t_1}\mathbf{d}_0^1\left(t \mid \mathbf{p}_0, \mathbf{p}_1; t_0, t_2\right) + \tag{2}$$

$$\frac{t - t_1}{t_2 - t_1}\mathbf{d}_1^1\left(t \mid \mathbf{p}_1, \mathbf{p}_2; t_1, t_3\right) \quad t \in [t_1, t_2]\,. \tag{3}$$

Here, we pick every second knot in the representation of the line segments as the interval $[t_1, t_2]$ is within the domain of both segments. This way, a *convex* combination is created and a quadratic curve segment is obtained. Please write down the formula for the next quadratic curve segment, and create the formula for the quadratic B-spline comprised of these two quadratic segments.

Properties. Given the construction process shown above, you may now have a deeper understanding of the B-spline curve. Please, write down four properties of the B-spline curve.

Construction using the de Boor's algorithm The construction process described above can be summarized using the de Boor algorithm. If you want, you can create your own implementation of the algorithm using any programming language that you are familiar with.

The objective of the exercise is to understand how changes in knots and points impact the shape of the curve. A Python script `deBoor.py`, which includes a minimal implementation and visualization, is already provided. In the code, the control points and the knot vector are global parameters. You can modify them as you wish. We suggest running the script with the following command:

```
python3 deBoor.py --degree 2
```

When the first and last knots have multiplicity of *degree* + 1, the B-Spline curve interpolates the endpoints. You can see the visualization by running:

```
python3 deBoor.py --degree 2 --interpolate_endpoints
```

2. Mesh Subdivision (Loop algorithm)

In this task, you will implement the Loop subdivision scheme. Loop subdivision is a method for refining triangular meshes to produce smoother surfaces. Your task is to apply the Loop subdivision algorithm to a given triangular mesh structure. You will start with the existing code framework, and complete the missing part. Some key aspects of this exercise include:

(a) **Wireframe Rendering:** Wireframe is often used to visualize triangular meshes. You can find the relevant code of `Task 2a` in the file *ex12.js*. Please, implement the missing lines. To simplify the exercise, a custom mesh class is already provided to you, where it is possible to retrieve the wireframe vertices using `getWireframeVertices`. In the `render` function, bind the buffers and render the line geometry using `gl.drawArrays(gl.LINES, ...)`.

(b) **Helper Functions for the computation of Adjacency:** Adjacency information is necessary for subdivision algorithms. Open the file *mesh.js* and read the code of `Mesh`, `Face`, `Edge` and `Vertex` class. Then, understand how the mesh structure is stored in such data structures. Finally, complete the functions `faceAdjacency` and `vertexAdjacency`.

(c) **Implementation of the Loop Algorithm:** Open the file *mesh.js* and implement the Loop algorithm in the function `LoopSubdivision`. You can refer to the C++ implementation from `libigl`. Note that the weights used to compute new vertex positions are slightly different from the lecture slides. You can play with the number of subdivision iterations to see how the mesh evolves.