ETH zürich

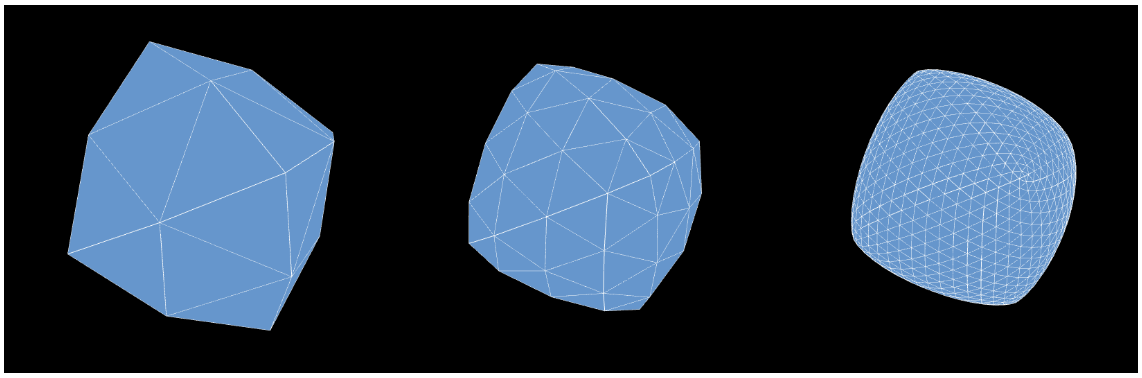
computer graphics laboratory

Prof. Dr. Markus Gross

Visual Computing

Exercise 11: Curves & Surfaces

Hand-out Date: 12 December 2023



Goals

- Understand the concept of B-Spline curves.
- Understand the principles of mesh subdivision.

Resources

The lecture slides and exercise slides are accessible via the Visual Computing Course Web Page.

Tasks

1. B-spline Curve

In this task, we will construct the quadratic B-Spline step by step to understand its concept. Recall that B-spline curves are defined by its degree, a set of control points, and a knot vector. The control points influence the shape of the curve. The knot vector is a sequence of values that determine the parameterization of the B-spline curve. It divides the curve into segments, and each segment is associated with a specific set of control points. 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, t_4. Further, we have $t_0 \leq t_1 < t_2 < t_3 \leq t_4$.

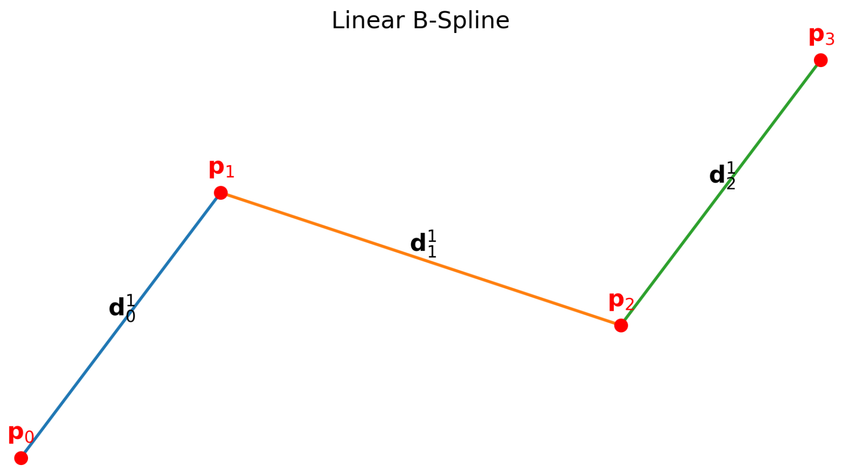


Figure 1: Linear B-spline.

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 formulas for other 2 consecutive line segments $\mathbf{d}_1^1$ and $\mathbf{d}_2^1$, and create 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.

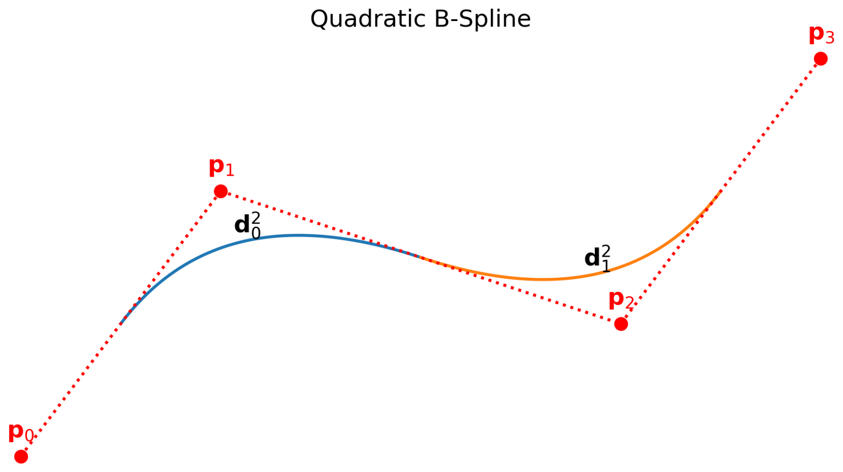


Figure 2: Quadratic B-spline.

Quadratic B-spline. Going to one degree higher, the construction of quadratic spline curves is based on the repeated averaging of two consecutive line segments.

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, the reason for picking every other knot in the representation of the line segments is that then the interval $[t_1, t_2]$ is within the domain of both segments. This ensures that the two line segments can be combined in a *convex* combination to form a quadratic curve segment. Please write down the formula for the next quadratic curve segment, and create the formula for the quadratic B-spline comprised of these two curve 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.

Implement de Boor's algorithm The construction process can be summarized by the de Boor's algorithm. You can try implementing the algorithm using any software you are familiar with. This helps to understand how changes in these elements impact the curve. A python script `deBoor.py` which includes a minimal implementation and visualization is given. The control points and the knot vector are global parameters in the code which you can modify as you like. You can run the script with the following command

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

When the first and last knot 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 concentrate on implementing the Loop subdivision algorithm. 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. Key aspects of this task include:

(a) **Wireframe Rendering:** Wireframe is often used for visualization when it comes to geometry processing. Find relevant code of `Task 2a` in the code and implement the missing lines. A custom mesh class is implemented for you already and you can retrieve wireframe vertices using `getWireframeVertices`. In `render` function, bind the corresponding buffers and render the line geometry with `gl.drawArrays(gl.LINES, ...)`.

(b) **Helper Functions to Compute Adjacency:** Adjacency information is necessary for all subdivision algorithms. Read the code of `Mesh`, `Face`, `Edge` and `Vertex` class and understand how the mesh structure is stored in such data structures. Fill in the function `faceAdjacency` and `vertexAdjacency`.

(c) **Implement Loop Algorithm:** Implement Loop algorithm in `LoopSubdivision`. You can refer to the C++ implement from `libigl`, which is a header-only C++ geometry processing library. Note that the weights to computed new vertex positions are slightly different from the lecture slides. You can play with different number of subdivision iterations to see how the mesh evolves.