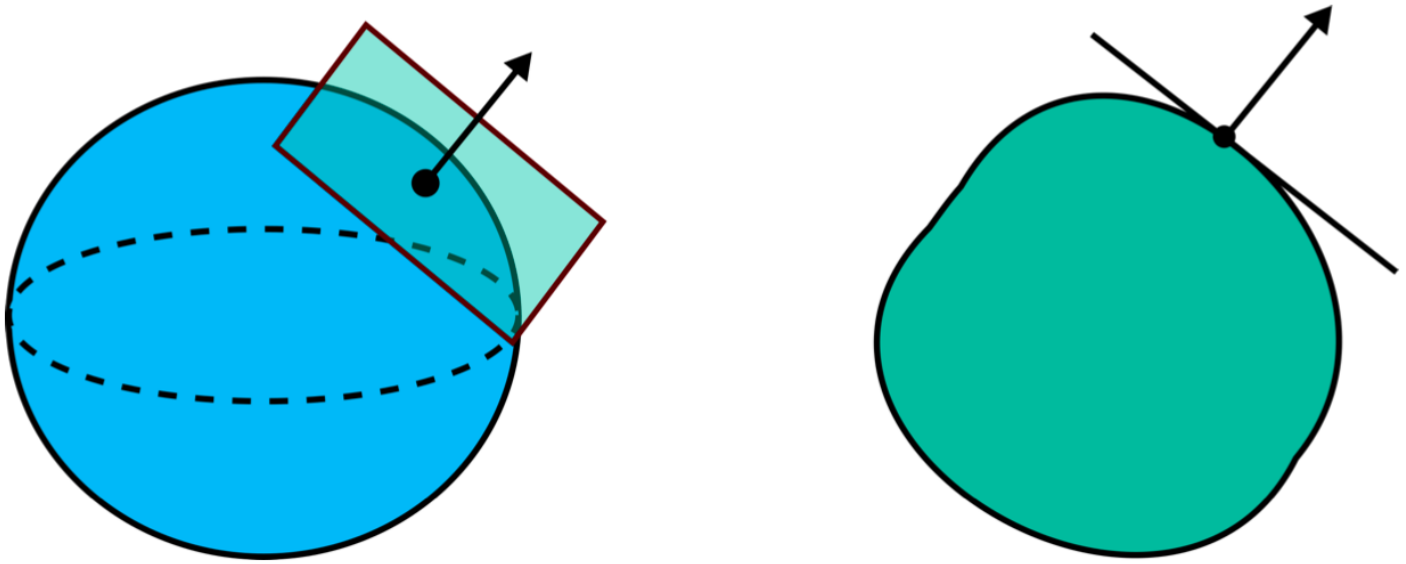
Methods in Computational Science – Function approximation for multidimensional domains (ch.10)

Johan Hoffman

Approximation for multidimensional domains

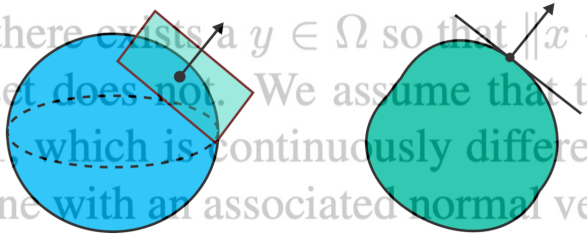


Approximation for multidimensional domains

Let the subset $\Omega \subset R^d$ be the domain of a function $f : \Omega \to R$. We define the *diameter* of Ω by

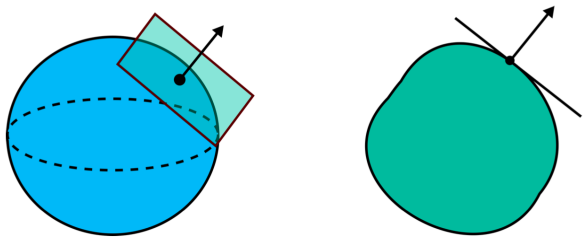
$$\text{diam}(\Omega) = \sup_{x,y \in \Omega} \|x - y\|,$$

and if $\text{diam}(\Omega) < \infty$ we say that the domain is *bounded*. The domain is an *open set* if for every $x \in \Omega$ there exists an *open ball* $B(x, r) = \{y \in R^d : \|x - y\| < r\}$ such that $B(x, r) \subset \Omega$, and the domain Ω is a *closed set* if its complement $\Omega^c = \{x \in R^d : x \notin \Omega\}$ is open. The *boundary* of Ω is defined as $\partial\Omega = \overline{\Omega} \cap \overline{\Omega^c}$, where the set $\overline{\Omega}$ is the *closure* of Ω in R^n, which consists of all $x \in R^d$ such that for any $\epsilon > 0$ there exists a $y \in \Omega$ so that $\|x - y\| < \epsilon$. A closed set contains its boundary, whereas an open set does not. We assume that the boundary is a parameterized hypersurface of dimension $d - 1$ which is continuously differentiable so that for each $x \in \partial\Omega$ we can define a tangent hyperplane with an associated normal vector. For an open domain Ω, we denote by $C^k(\Omega)$ the Banach space of k times continuously differentiable functions.



Approximation for multidimensional domains

Let the subset $\Omega \subset R^d$ be the d



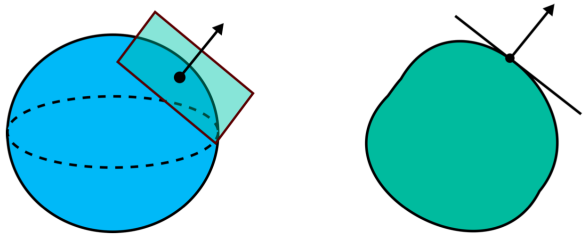
Ve define the *diameter* of Ω by

and if $\text{diam}(\Omega) < \infty$ we say tha main is an *open set* if for every $x \in \Omega$ there exists an *open ball* $B(x, r) = \{y \in R^d : \|x - y\| < r\}$ such that $B(x, r) \subset \Omega$, and the domain Ω is a *closed set* if its complement $\Omega^c = \{x \in R^d : x \notin \Omega\}$ is open. The *boundary* of Ω is defined as $\partial\Omega = \overline{\Omega} \cap \overline{\Omega^c}$, where the set $\overline{\Omega}$ is the *closure* of Ω in R^n, which consists of all $x \in R^d$ such that for any $\epsilon > 0$ there exists a $y \in \Omega$ so that $\|x - y\| < \epsilon$. A closed set contains its boundary, whereas an open set does not. We assume that the boundary is a parameterized hypersurface of dimension $d - 1$, which is continuously differentiable so that for each $x \in \partial\Omega$ we can define a tangent hyperplane with an associated normal vector. For an open domain Ω, we denote by $C^k(\Omega)$ the Banach space of k times continuously differentiable functions.

Approximation for multidimensional domains

Let the subset $\Omega \subset R^d$ be the domain of a function $f : \Omega \to R$. We define the *diameter* of Ω by

$$\text{diam}(\Omega) = \sup_{x,y} \|x - y\|.$$



and if $\text{diam}(\Omega) < \infty$ we say tha main is an *open set* if for every $x \in \Omega$ there exists an *open ball* $r\}$ such that $B(x, r) \subset \Omega$, and the domain Ω is a *closed set* if i $x \notin \Omega\}$ is open. The *boundary* of Ω is defined as $\partial\Omega = \overline{\Omega} \cap \overline{\Omega^c}$, where the set $\overline{\Omega}$ is the *closure* of Ω in R^n, which consists of all $x \in R^d$ such that for any $\epsilon > 0$ there exists a $y \in \Omega$ so that $\|x - y\| < \epsilon$. A closed set contains its boundary, whereas an open set does not. We assume that the boundary is a parameterized hypersurface of dimension $d - 1$, which is continuously differentiable so that for each $x \in \partial\Omega$ we can define a tangent hyperplane with an associated normal vector. For an open domain Ω, we denote by $C^k(\Omega)$ the Banach space of k times continuously differentiable functions.

Approximation for multidimensional domains

Now we seek to approximate a function $f : \Omega \to R$ by another function $f_N \approx f$, for which we can measure the approximation error by a residual $R(f_N)$, with the property that

$$R(f) = 0.$$

Therefore, we can interpret (10.1) as an equation with exact solution $f \in V$ and approximate solution $f_N \in V_N$, where V is a suitable Hilbert space and $V_N \subset V$ a finite dimensional approximation space. The residual can express the pointwise error $R(f_N) = f - f_N$, or be more general where (10.1) defines a differential or integral equation.

Minimization, projection and collocation

$$\min_{f_N \in V_N} \|R(f_N)\|,$$

which leads to a *least squares method* to solve the equation (10.1). In a *Galerkin method* we instead seek a function $f_N \in V_N$ for which the residual is orthogonal to the approximation space V_N with respect to an inner product,

$$(R(f_N), v) = 0, \quad \forall v \in V_N,$$

therefore also called a *Galerkin projection*. We say that the Galerkin projection is a *weak form* of equation (10.1). Analogous to interpolation, we may also seek to satisfy equation (10.1) in a set of N nodes x_i, referred to as a *collocation method*, such that

$$R(f_N)(x_i) = 0, \quad i = 1, ..., N.$$

Minimization, projection and collocation

Example 10.1. If equation (10.1) represents a differential equation, the finite difference method is a collocation method, whereas the finite element method is a Galerkin method. For the trivial equation $R(f_N) = f - f_N$, collocation is identical to interpolation, whereas the Galerkin method and the least squares method both correspond to orthogonal projection of f onto V_N.

Discretization of the domain

To construct the finite dimensional approximation space V_N we typically need to discretize the domain Ω of the function we seek to approximate. For $\Omega \subset R^d$ such a discretization may take the form of a *mesh*, defined as a collection of disjoint subdomains, or elements, over which a set of basis functions $\{\phi_j\}$ are defined with associated coordinates $\{\alpha_j\}$, where each basis function is nonzero only for a small subset of elements in the mesh. It follows that any $f_N \in V_N$ can be expressed as

$$f_N(x) = \sum_{j=1}^{N} \alpha_j \phi_j(x).$$

Discretization of the domain

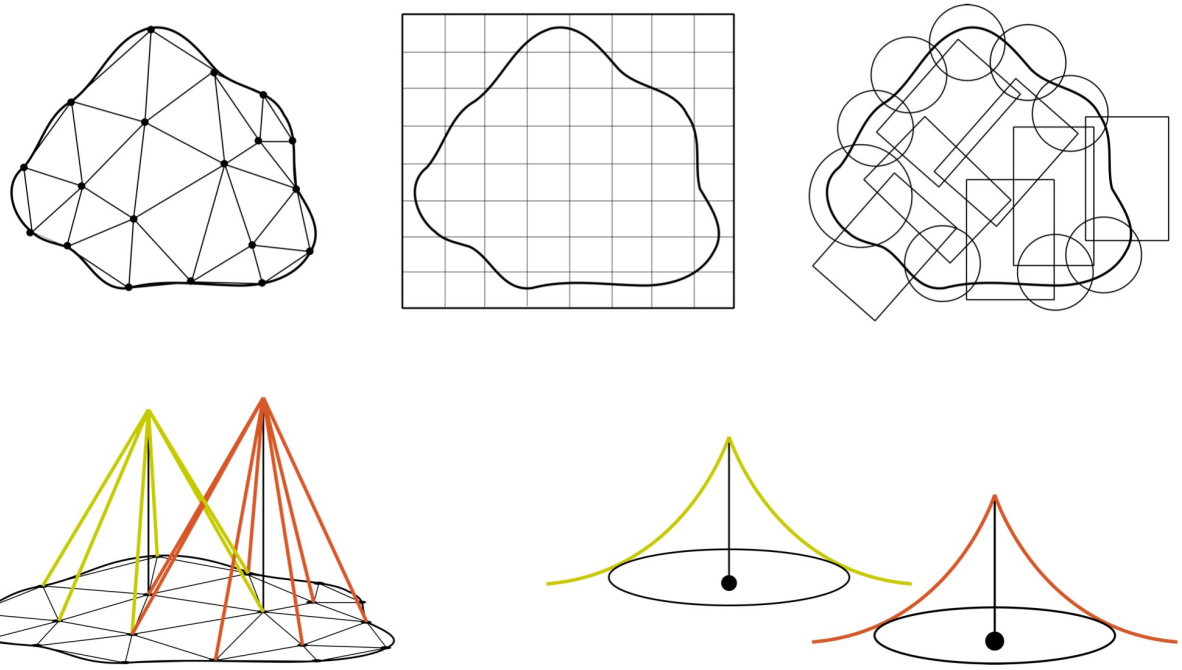
$$f_N(x) = \sum_{j=1}^{N} \alpha_j \phi_j(x).$$

The mesh may be constructed to approximate the domain, or a fixed background mesh can be used which is cut by the domain. Alternatively, the approximation is *meshfree*, where instead the discretization is based on a set of nodes $\{x_j\}$ in Ω, with associated weight functions, or *radial basis functions*,

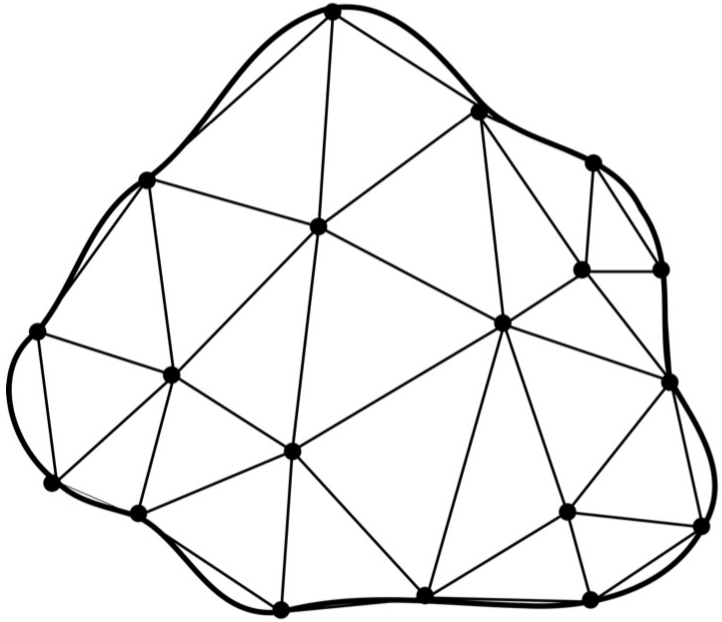
$$\phi_j(x) = \phi(\|x - x_j\|),$$

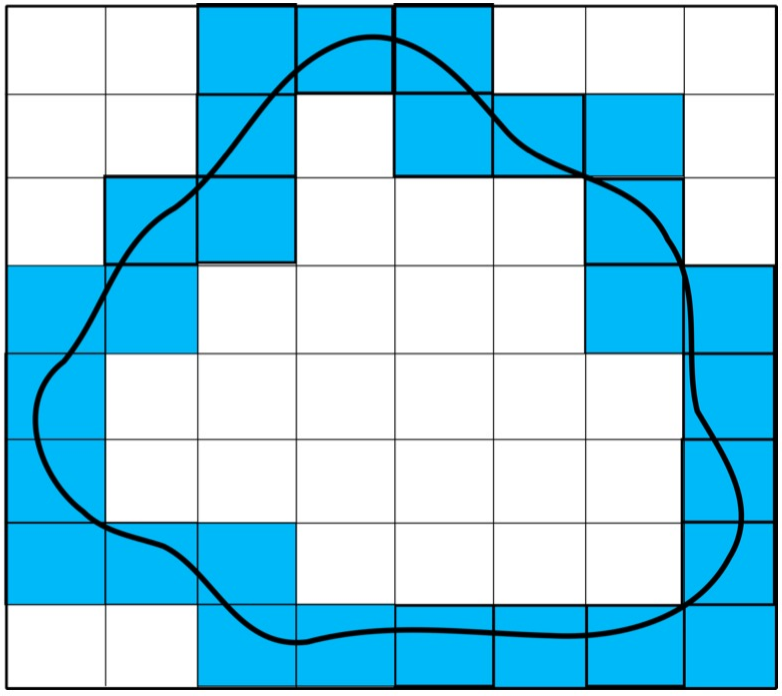
defined in terms of the Euclidian distance in Ω.

Discretization of the domain



Structured grid vs unstructured mesh





Unstructured mesh

$$\Omega \approx \Omega^h = \bigcup_{i=1}^{N} \Omega_i^h, \quad \Omega_i^h \cap \Omega_j^h = \emptyset, \quad \forall i \neq j.$$

The accuracy of the approximation $\Omega \approx \Omega^h$ depends on with what precision $\partial\Omega^h$ can represent the geometric shape of the boundary $\partial\Omega$, including the local normal vector $n(x)$.

A common choice of element type in an unstructured mesh is a *simplex*, which corresponds to a triangle in R^2 (2-simplex) and a tetrahedron in R^3 (3-simplex). A general d-simplex can be constructed recursively from $d+1$ points, or *vertices*, by connecting each new vertex to d other vertices by line segments, or *edges*, according to a given rule. A vertex is also denoted a 0-face, an edge a 1-face, and the $(d-1)$-face is referred to as a *facet*. In R^2, a simplex mesh

$$\mathcal{T}^h = \{K\}$$

is a collection of triangular elements K, formed by vertices connected by edges, whereas in R^3 a simplex mesh is a collection of tetrahedral elements K, formed by vertices, edges and facets.

Conforming mesh

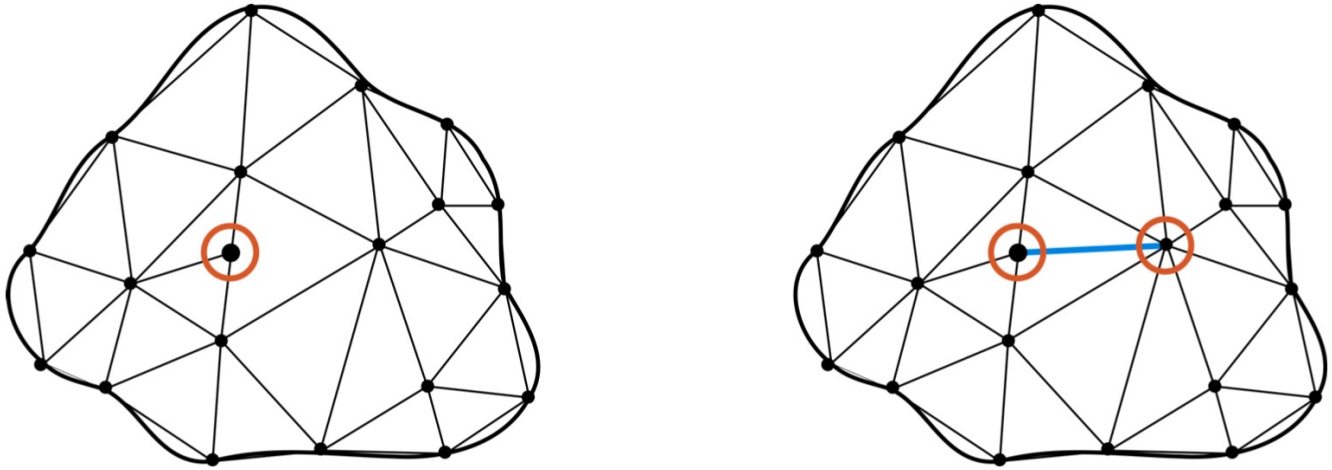


Figure 10.5. *Illustration of a non-conforming triangular mesh with a hanging node (left), which can be made into a conforming mesh by adding a new edge that eliminates the hanging node (right).*

Mesh generation

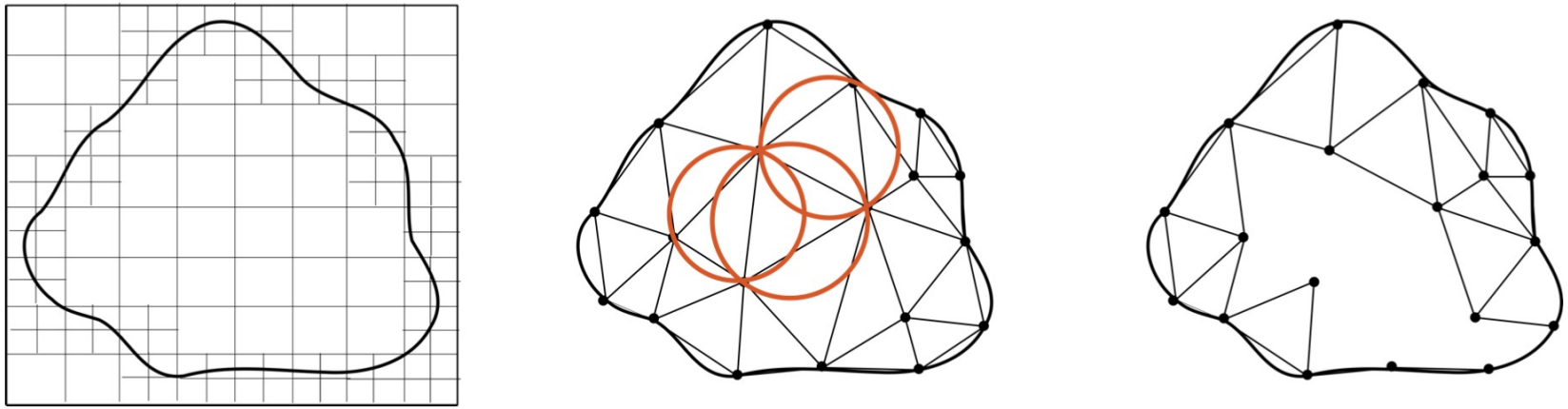


Figure 10.6. *Mesh generation by a quadtree algorithm (left), by the Delaunay condition (center), and by advancing front mesh generation (right).*

Mesh topology

The topology of an unstructured mesh is described by the *connectivity* of its *mesh entities*,

$$
\begin{aligned}
\mathcal{V} &= \{v_1, ..., v_n\}, \\
\mathcal{E} &= \{e_1, ..., e_k\}, \qquad e_i \in \mathcal{V} \times \mathcal{V}, \\
\mathcal{F} &= \{f_1, ..., f_m\}, \qquad f_i \in \mathcal{V} \times \mathcal{V} \times \mathcal{V}, \\
\mathcal{K} &= \{K_1, ..., K_M\}, \quad K_i \in \mathcal{V} \times \mathcal{V} \times \mathcal{V} \times \mathcal{V},
\end{aligned}
$$

here for the example of a tetrahedral mesh, where each edge is defined by two vertices, each triangular facet by three vertices, and each tetrahedral element by four vertices.

Distributed parallel mesh

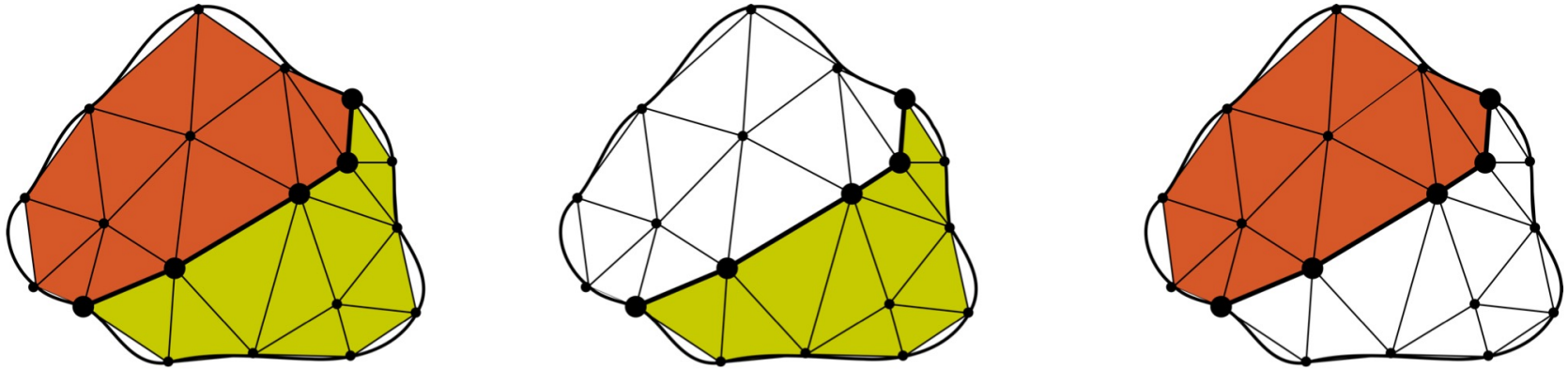
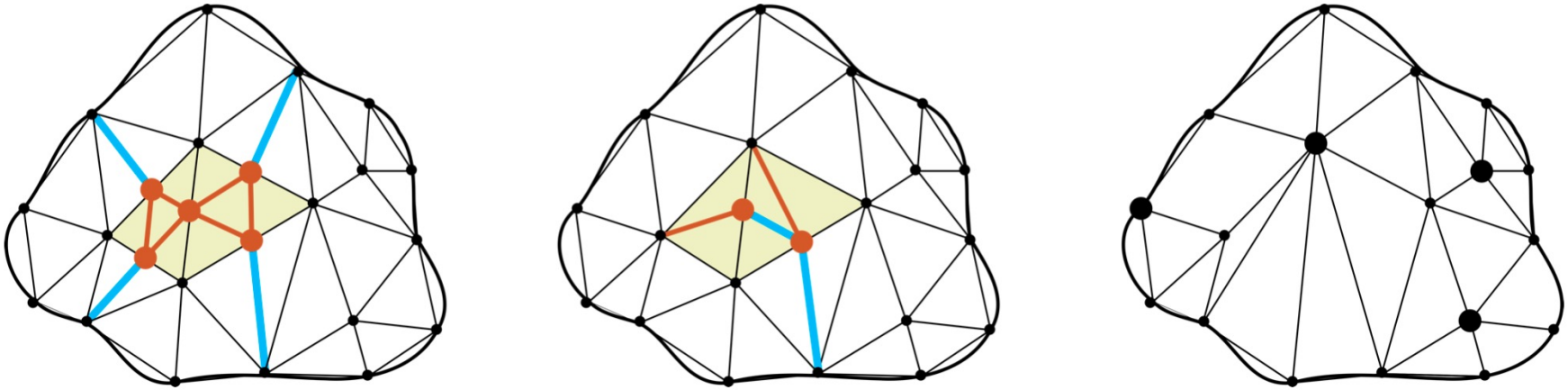


Figure 10.7. *A distributed mesh partitioned over two processors P_0 and P_1, where the two partitions $\mathcal{T}_0^h$ and $\mathcal{T}_1^h$ are marked in different colors (left). Each processor holds the mesh data assigned to it, together with data for the ghost entities marked in bold, which is synchronized between the processors at a certain frequency (center and right).*

Mesh refinement and coarsening



Polynomial interpolation in multiple dimensions

Polynomial interpolation in one dimension is based on an interval domain, a polynomial vector space, and a set of nodes. Analogously, for a simplicial mesh $\mathcal{T}^h = \{K\}$ in R^d, we first define an *element domain* K together with a finite dimensional polynomial space

$$P_K = \mathcal{P}^q(K),$$

which are the local *shape functions*. Then we define the *nodes*

$$\Sigma_K = \{\sigma_i\}_{i=0}^{n_q - 1},$$

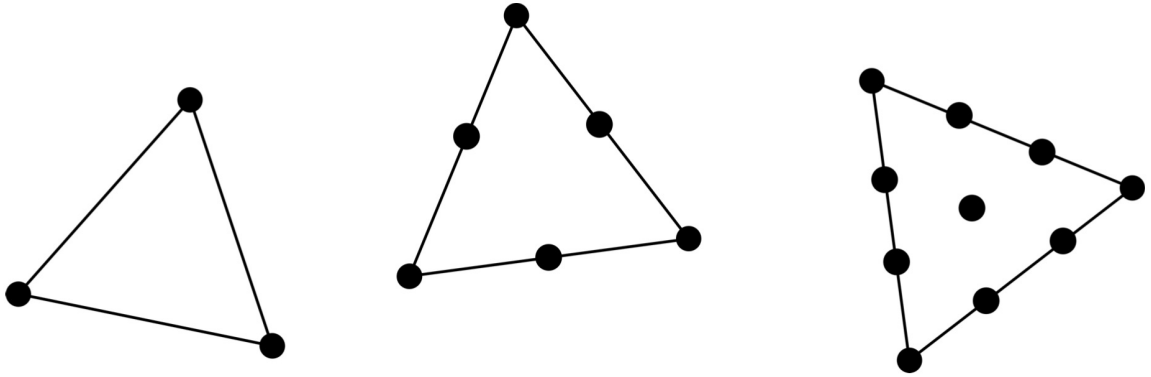
which is a set of local degrees of freedom which form a basis for the dual space P'_K of bounded linear functionals on P_K, a generalization of the pointwise function evaluations of the Lagrange nodal basis. The triplet (K, P_K, Σ_K) is referred to as a *finite element*, by which we can build global approximations on discretized domains.

Polynomial interpolation in multiple dimensions

The Lagrange polynomials $\{\lambda_i\}_{i=0}^{n_q-1}$ are defined to be a nodal basis for P_K with respect to a set of points in K with coordinates $\{x_i\}_{i=0}^{n_q-1}$. These points cannot be chosen arbitrarily, they need to be chosen such that the polynomials $\{\lambda_i\}_{i=0}^{n_q-1}$ form a linearly independent set in P_K. For Σ_K the set of evaluation functionals at the coordinates $\{x_i\}_{i=0}^{n_q-1}$,

$$\sigma_i(\lambda_j) = \lambda_j(x_i) = \delta_{ij},$$

since the Lagrange polynomials is a nodal basis.



Piecewise polynomial approximation

Similar to one dimensional domains, we use the local Lagrange shape functions to construct global approximation spaces, in the form of discontinuous and continuous piecewise polynomial approximation spaces over the mesh $\mathcal{T}^h$. The discontinuous approximation space is defined by

$$W_h^{(q)} = \{v : v|_K \in P_K\},$$

and the continuous approximation space by

$$V_h^{(q)} = \{v \in C(\Omega) : v|_K \in P_K\},$$

where $v|_K$ indicates the restriction of the function v to the element domain K, and $C(\Omega)$ is the vector space of continuous functions over the domain which is discretized by the mesh $\mathcal{T}^h$, assuming here for simplicity that $\Omega = \Omega^h$.

Piecewise polynomial approximation

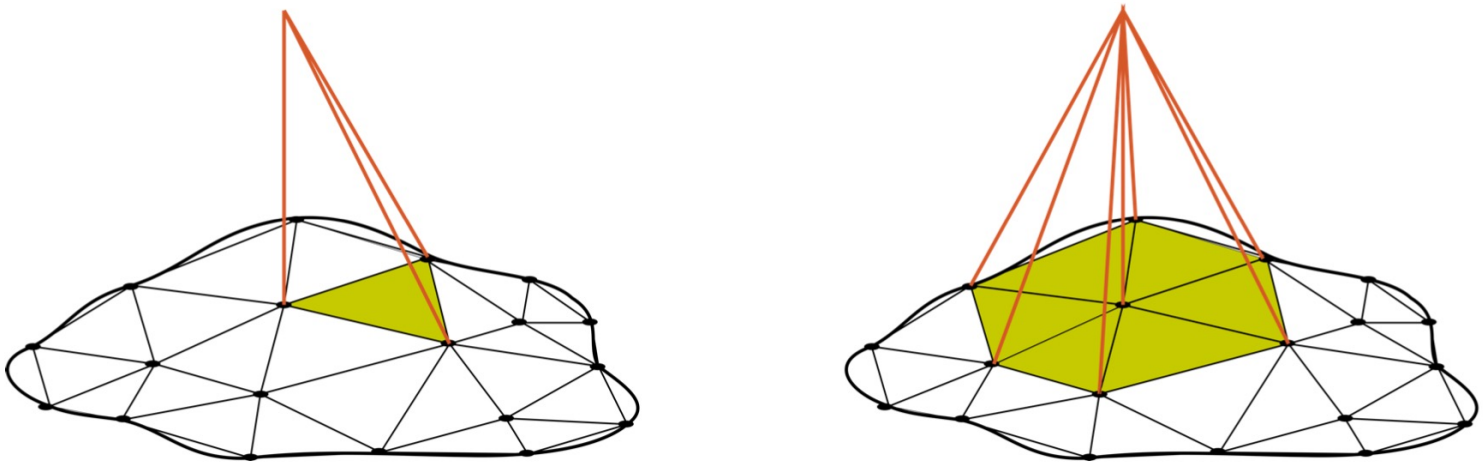


Figure 10.10. *Global basis functions for $W_h^{(1)}$ (left) and $V_h^{(1)}$ (right) over a triangular mesh, with the support of each basis function indicated in the figure. The support of the basis function of the discontinuous approximation space is localized to one element whereas the support of the basis function for the continuous space is equal to the star of the vertex associated with the basis function.*

The reference element

For each simplex $K \in \mathcal{T}^h$ we define a finite element (K, P_K, Σ_K) which can be mapped back to a *reference element* $(\hat{K}, \hat{P}_K, \hat{\Sigma}_K)$ by an affine map $F_K : \hat{K} \to K$, given by

$$x = F_K(\hat{x}) = A_K \hat{x} + b_K,$$

for $x \in K$ and $\hat{x} \in \hat{K}$, with A_K a nonsingular matrix. We can also express the map F_K in terms of the linear Lagrange shape functions λ_i and the vertex coordinates $x_i \in R^d$ of the simplex K,

$$x = F_K(\hat{x}) = \sum_{i=0}^{n_1 - 1} x_i \lambda_i(\hat{x}).$$

The reference element

Computations on the finite element (K, P_K, Σ_K) can, therefore, instead be performed on the reference element $(\hat{K}, \hat{P}_K, \hat{\Sigma}_K)$, by using F_K and the (constant) Jacobian of the affine map,

$$J_K = F_K' = A_K = \sum_{i=0}^{n_1-1} x_i \hat{\nabla} \lambda_i(\hat{x}), \quad (J_K)_{ij} = \frac{\partial x_i}{\partial \hat{x}_j}.$$

This allows for efficient algorithms over the mesh $\mathcal{T}^h$, such as Algorithm 9.2, which can be traversed simply by updating the affine map F_K for each K. For a function $u : K \to R$, by the map F_K we can express the integral and the gradient of the function in the coordinates $\hat{x}$ and gradient $\hat{\nabla}$ of the reference element $\hat{K}$,

$$\int_K u(x)\, dx = \int_{\hat{K}} u(F_K(\hat{x}))|\det(J_K)|\, d\hat{x}, \quad \nabla u(x) = (J_K^T)^{-1} \hat{\nabla} u(\hat{x}).$$

The reference element

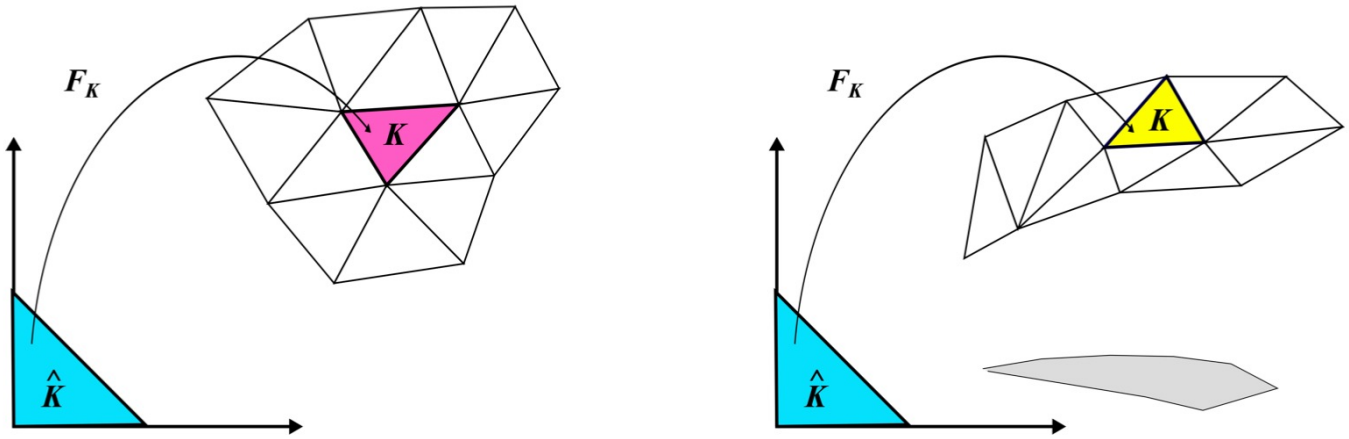
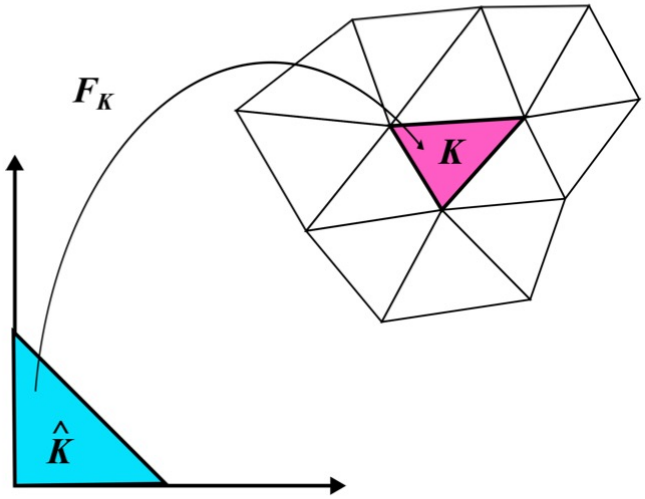


Figure 10.11. *Simplicial mesh with the reference element $\hat{K}$ and the affine map $F_K : \hat{K} \to K$, for the cases of a two dimensional domain (left) and a parameterized surface in three dimensions (right).*

Assembly algorithm

ALGORITHM 9.2. (A, b) = assemble_system(f).
Input: function **f**
Output: assembled matrix **A** and vector **b**.

```
1:  for k=0:no_elements-1 do
2:      q = get_no_local_shape_functions(k)
3:      loc2glob = get_local_to_global_map(k)
4:      for i=0:q do
5:          b[i] = integrate_vector(f, k, i)
6:          for j=0:q do
7:              a[i,j] = integrate_matrix(k, i, j)
8:          end for
9:      end for
10:     add_to_global_vector(b, loc2glob)
11:     add_to_global_matrix(a, loc2glob)
12: end for
13: return A, b
```



Barycentric coordinates

Given a d-simplex K, by equation (10.6) we can express any $x \in R^d$ as a linear combination of $\{x_i\}_{i=0}^d \subset R^d$, the coordinates of the vertices of K,

$$x = x_0\lambda_0(x) + x_1\lambda_1(x) + ... + x_d\lambda_d(x),$$

where the linear Lagrange basis functions λ_i here are interpreted as the *barycentric coordinates* of x with respect to K, with $\lambda_i(x_j) = \delta_{ij}$. Alternatively, we can view λ_i as an *influence function* for the *control point* x_i. For a point $x \in K$, all barycentric coordinates satisfy the condition $0 \leq \lambda_i \leq 1$, and since the barycentric coordinates form a partition of unity,

$$\lambda_d(x) = 1 - \sum_{i=0}^{d-1} \lambda_i(x).$$

Barycentric coordinates

The barycentric coordinates also allow for exact integration formulas, here listed for $d = 1, 2, 3$.

$$\frac{1}{|K|}\int_K \lambda_1^{m_1}\lambda_2^{m_2}\,dx = \frac{1!m_1!m_2!}{(1+m_1+m_2)!} \qquad (d=1)$$

$$\frac{1}{|K|}\int_K \lambda_1^{m_1}\lambda_2^{m_2}\lambda_3^{m_3}\,dx = \frac{2!m_1!m_2!m_3!}{(2+m_1+m_2+m_3)!} \qquad (d=2)$$

$$\frac{1}{|K|}\int_K \lambda_1^{m_1}\lambda_2^{m_2}\lambda_3^{m_3}\lambda_4^{m_4}\,dx = \frac{3!m_1!m_2!m_3!m_4!}{(3+m_1+m_2+m_3+m_4)!} \qquad (d=3)$$

Therefore, in the case of piecewise linear approximation on simplex elements we have exact formulas for integrals of products of basis functions, which we can use to implement the functions **integrate_vector(f, k, i)** and **integrate_matrix(k, i, j)** in Algorithm 9.2.

Bernstein polynomials

The *Bernstein polynomial* of degree q is defined for $t \in [0,1]$ by

$$B_i^{(q)}(t) = \binom{q}{i} t^i (1-t)^{q-i}, \quad i = 0, 1, ..., q,$$

for which we have the following recurrence relation,

$$B_i^{(q)}(t) = (1-t)B_i^{(q-1)}(t) + tB_{i-1}^{(q-1)}(t), \quad i = 0, 1, ..., q, \quad B_0^{(0)} = 1.$$

Example 10.4. The cubic Bernstein polynomials take the form,

$$B_0^{(3)}(t) = (1-t)^3, \quad B_1^{(3)}(t) = 3t(1-t)^2,$$
$$B_2^{(3)}(t) = 3t^2(1-t), \quad B_3^{(3)}(t) = t^3.$$

Bernstein polynomials

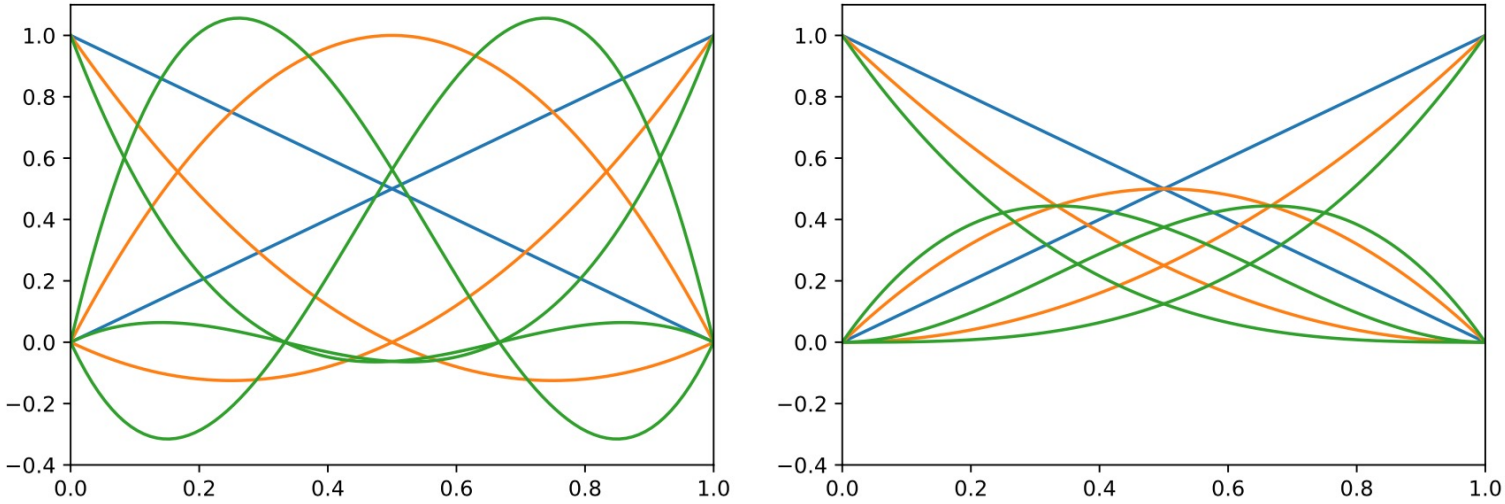


Figure 10.12. *Lagrange polynomials (left) and Bernstein polynomials (right) for* $q = 1, 2, 3$*, and uniformly spaced nodes on* $[0, 1]$*. We observe that whereas the higher order Lagrange polynomials oscillate around zero, the Bernstein polynomials are all nonnegative.*

Bézier curves and surfaces

$$B_q(t) = \sum_{i=0}^{q} P_i B_i^{(q)}(t), \quad t \in [0,1].$$

We refer to $B_q(t)$ as a *Bézier curve* in R^d, where we can view the control points as coordinates in the Bernstein basis, or alternatively, the Bernstein polynomials as influence functions for the control points P_i.

The Bézier curve of degree q generalizes to a *Bézier surface* of degree (m, n), which is parameterized over the unit square, with $(n+1)(m+1)$ control points $P_{i,j}$,

$$B_q(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{i,j} B_i^{(n)}(u) B_j^{(m)}(v), \quad (u,v) \in [0,1]^2.$$

Bézier curves and surfaces

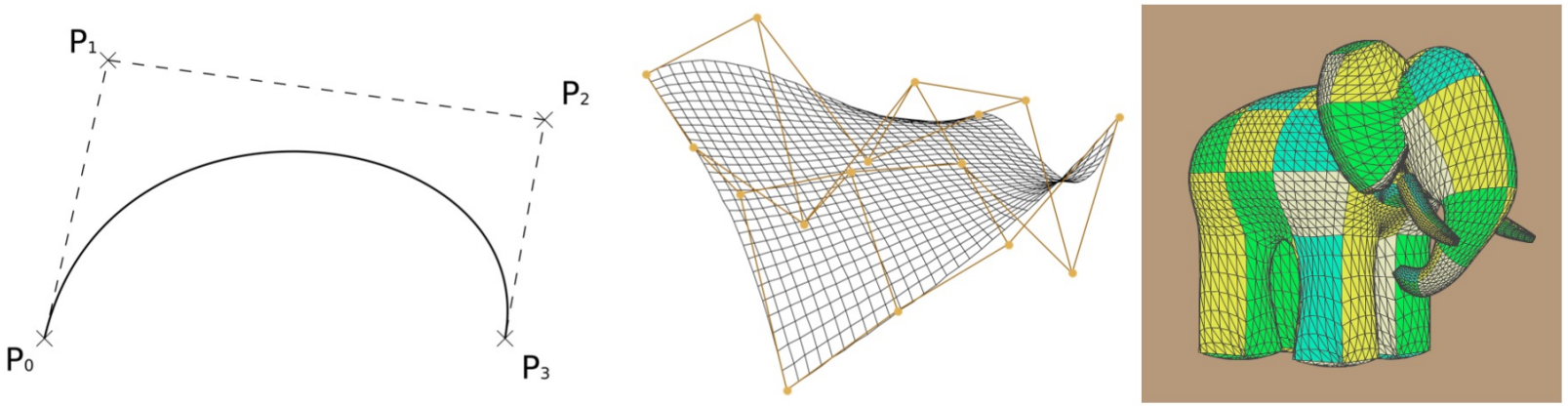


Figure 10.14. *A cubic Bézier curve in R^2 (left), a bicubic Bézier surface in R^3 (center), and a composite surface of bicubic Bézier surfaces (right).*

B-splines

$$C(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) P_i, \quad \xi \in [0,1].$$

Then we subdivide the parameter domain $[0,1]$ by a *knot vector*

$$\Xi = \{\xi_1 \leq \xi_2 \leq \ldots \leq \xi_{n+p+1}\},$$

with $\xi_1 = 0$ and $\xi_{n+p+1} = 1$. B-spline basis functions of order (degree) p with n control points, are then generated by the following recursion formula

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+1+p} - \xi}{\xi_{i+1+p} - \xi_{i+1}} N_{i+1,p-1}(\xi),$$

where

$$N_{i,0}(\xi) = \begin{cases} 1, & \xi_i \leq \xi < \xi_{i+1}, \\ 0, & else. \end{cases}$$

NURBS (Non-Uniform Rational B-splines)

$$R_{i,p}(\xi) = \frac{w_i N_{i,p}(\xi)}{\sum_{j=1}^{n} w_j N_{j,p}(\xi)}, \quad \xi \in [0,1],$$

by which we can express NURBS curves,

$$C(\xi) = \sum_{i=1}^{n} R_{i,p}(\xi) P_i, \quad \xi \in [0,1],$$

and multivariate NURBS in d dimensions,

$$S(\hat{\xi}) = \sum_{i=1}^{\hat{n}} \hat{R}_{i,p}(\xi) \hat{P}_i, \quad \hat{\xi} \in [0,1]^d,$$

with $\hat{R}_{i,p}$ the weighted tensor product B-spline basis $\hat{N}(i,p)$.

NURBS (Non-Uniform Rational B-splines)

We can then define the NURBS function spaces to be used for approximation,

$$S_{p,\Xi} = \text{span}\{R_{i,p}\}_{i=1}^{n}, \quad \hat{S}_{p,\Xi} = \text{span}\{\hat{R}_{i,p}\}_{i=1}^{\hat{n}},$$

for which we want to define interpolation operators,

$$\mathcal{I}_{p,\Xi} : L^2([0,1]) \to S_{p,\Xi}, \quad \hat{\mathcal{I}}_{p,\Xi} : L^2([0,1]^d) \to \hat{S}_{p,\Xi}.$$

Similar to approximation with a simplicial mesh, we can approximate the domain Ω by a NURBS discretization Ω_h,

$$\Omega^h = \bigcup_{i=1}^{N} \Omega_i^h.$$