Advanced Topics in Computer Graphics II
Geometry Processing

Surface Representations

October 15, 2024

Explicit and Implicit representations

- **Explicit representation**

$$\begin{aligned} \boldsymbol{p}\colon \mathbb{R} &\to \mathbb{R}^3 & \boldsymbol{q}\colon \mathbb{R}^2 &\to \mathbb{R}^3 \\ t &\mapsto (x(t)\,,\, y(t)\,,\, z(t))^T & (u,v) &\mapsto (x(u,v)\,,\, y(u,v)\,,\, z(u,v))^T \end{aligned}$$

Example:

$$\boldsymbol{p}(t) = r\,(\cos(t)\,,\, \sin(t)\,,\, 0)^T \quad \boldsymbol{q}(u,v) = r\,(\cos(u)\cos(v)\,,\, \sin(u)\cos(v)\,,\, \sin(v))^T$$

$$t \in [0, 2\pi] \qquad (u,v) \in [0,2\pi] \times \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$$

- **Implicit representation**

$$\begin{aligned} f\colon \mathbb{R}^2 &\to \mathbb{R} & g\colon \mathbb{R}^3 &\to \mathbb{R} \\ \mathcal{I} &= \{\boldsymbol{p} \in \mathbb{R}^2 \mid f(\boldsymbol{p}) = 0\} & \mathcal{I} &= \{\boldsymbol{p} \in \mathbb{R}^3 \mid g(\boldsymbol{p}) = 0\} \end{aligned}$$

Example:

$$f(x,y) = x^2 + y^2 - r^2 \qquad g(x,y,z) = x^2 + y^2 + z^2 - r^2$$

Explicit Representations

Advantages

- Explicit evaluation
- Simple to sample from
- Efficient approximation by picewise linear functioncs

Disadvantages:

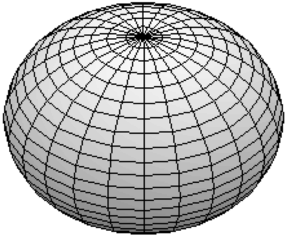
- Hard to determine inside/outside
- Hard to determine if a point is on the curve/surface

Parameterized Surfaces

A map

$$\boldsymbol{q} : \mathbb{R}^2 \to \mathbb{R}^d \qquad d = 1, 2, 3, \ldots$$
$$(u, v) \mapsto \boldsymbol{q}(u, v) = (x(u, v), y(u, v), z(u, v))^T$$

is called parameterized surface.



$$\boldsymbol{q}(u, v) = r(\cos(u)\cos(v), \sin(u)\cos(v), \sin(v))^T$$
$$(u, v) \in [-\pi, \pi] \times [-\pi/2, \pi/2]$$

As parameter domain, one often chooses the subset $D \subset \mathbb{R}^2$. Often

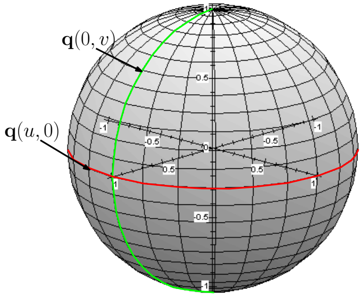
$$D = [a, b] \times [c, d] \text{ or } D = [0, 1]^2.$$

Parameterized Surfaces

The curves

$$\boldsymbol{p}(u) = q(u, v_0) \text{ for fixed } v = v_0 \text{ resp. } p(v) = q(u_0, v) \text{ for fixed } u = u_0$$

are called parameter curves of the surface.

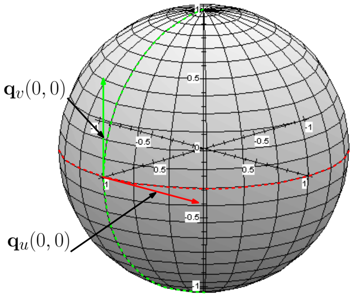


$$\boldsymbol{q}(u, v) = r(\cos(u)\cos(v), \sin(u)\cos(v), \sin(v))^T$$

Parameterized Surfaces

A surface is called n-times continuously differentiable, if the map $\boldsymbol{q}$ is n-times continuously differentiable, i.e. $\boldsymbol{q}$ has n-times continuously differentiable derivatives.

The vectors $\boldsymbol{q}_u(u,v) = \frac{\partial \boldsymbol{q}(u,v)}{\partial u}$ and $\boldsymbol{q}_v(u,v) = \frac{\partial \boldsymbol{q}(u,v)}{\partial v}$ are called **u-tangent** resp. **v-tangent** at (u,v).



Parameterized Surfaces

A surface is called regular, if $\boldsymbol{q}$ is at least one continuously differentiable and the vectors

$$\boldsymbol{q}_u(u,v) \text{ and } \boldsymbol{q}_v(u,v)$$

are linearly independent, for all $(u,v) \in D$.

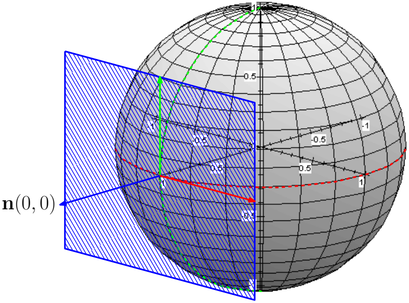
If $\boldsymbol{q}\colon D \to \mathbb{R}^d$ is a regular parameterized surface, we call the plane spanned by the vectors $\boldsymbol{q}_u(u_0,v_0)$ and $\boldsymbol{q}_v(u_0,v_0)$ tangential plane at the point $\boldsymbol{q}(u_0,v_0)$.

The vector

$$\boldsymbol{n}(u,v) = \frac{\boldsymbol{q}_u(u,v) \times \boldsymbol{q}_v(u,v)}{\|\boldsymbol{q}_u(u,v) \times \boldsymbol{q}_v(u,v)\|}$$

is called normal vector at the point $q(u_0,v_0)$.

Ii is perpendicular to the tangential plane and it is independent of the parameterization.



Tensor Product Surfaces

If $\{F_i^m \quad i = 0, \ldots, m\}$ and $\{G_i^n \quad i = 0, \ldots, n\}$ are bases of two function spaces R_1 and R_2 with real-valued univariate functions over the intervals $[a, b]$ resp. $[c, d]$, the bivariate real-valued functions

$$F_i^m G_j^n(u, v) = F_i^m(u) \cdot G_j^n(v)$$

with

$$i = 0, \ldots, m \qquad j = 0, \ldots, n \qquad (u, v) \in [a, b] \times [c, d]$$

form a basis of the tensor product space $R_1 \otimes R_2$ with dimension $(m + 1)(n + 1)$.

Example: Bernstein polynomials of degree m resp. n in $[0, 1]$. The bivariate polynomials

$$B_i^m B_j^n(u, v) = B_i^m(u) \cdot B_j^n(v)$$

with

$$i = 0, \ldots, m \qquad j = 0, \ldots, n \qquad (u, v) \in [0, 1] \times [0, 1]$$

form a basis of the tensor product space $P^m \otimes P^n$ of polynomials of degree m resp. n.

Tensor Product Surfaces

If a basis $F_i^m(u) \cdot G_j^n(v)$ of a tensor product space $R_1 \otimes R_2$ of functions in $[a,b] \times [c,d]$ and coefficients $\boldsymbol{c}_{ij} \in \mathbb{R}^d$ with $i = 0, \ldots, m$ and $j = 0, \ldots, n$ is given, we call the function respective to the tensor product bases

$$\boldsymbol{q}(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} \boldsymbol{c}_{ij} F_i^m(u) \cdot G_j^n(v), \qquad (u,v) \in [a,b] \times [c,d]$$

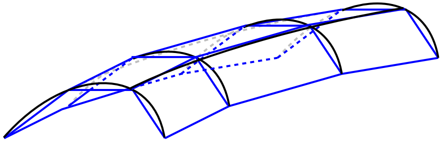
tensor product surface.

This equation can be written in matrix form as:

$$\boldsymbol{q}(u,v) = (F_0^m, \ldots, F_m^m) \begin{bmatrix} \boldsymbol{c}_{00} & \ldots & \boldsymbol{c}_{0n} \\ \vdots & \ddots & \vdots \\ \boldsymbol{c}_{m0} & \ldots & \boldsymbol{c}_{mn} \end{bmatrix} \begin{pmatrix} G_0^n \\ \vdots \\ G_n^n \end{pmatrix}$$

Tensor Product Surfaces

Tensor product surfaces are obtained easily from curve representations:



The movement of the polynomial curve

$$\boldsymbol{q}(u) = \sum_{i=0}^{m} \boldsymbol{a}_i F_i^m(u), \quad \boldsymbol{a}_i \in \mathbb{R}^d$$

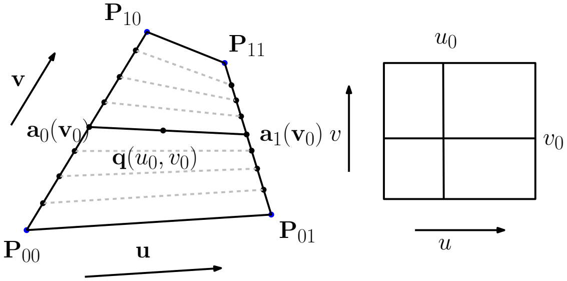
with its control points through space can be formulated as follows:
If the change of control points is defined by a polynomial curve, one obtains

$$\boldsymbol{a}_i(v) = \sum_{j=0}^{n} \boldsymbol{c}_{ij} G_j^n(v), \quad \boldsymbol{c}_{ij} \in \mathbb{R}^d$$

$$\boldsymbol{q}(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} \boldsymbol{c}_{ij} F_i^m(u) G_j^n(v), \quad \boldsymbol{c}_{ij} \in \mathbb{R}^d$$

Geometric Interpretation

Example: Four different points $\boldsymbol{P}_{00}, \boldsymbol{P}_{01}, \boldsymbol{P}_{10}, \boldsymbol{P}_{11}$ given in $\mathbb{R}^3$.

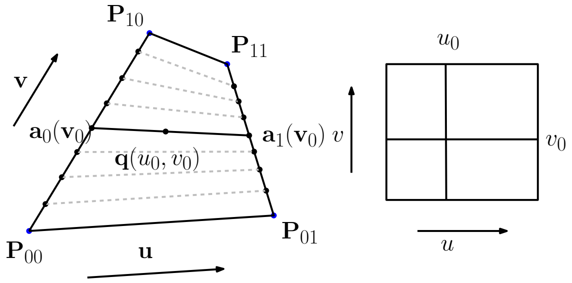


First, we interpolate between points $\boldsymbol{P}_{00}, \boldsymbol{P}_{10}$ and $\boldsymbol{P}_{01}, \boldsymbol{P}_{11}$ by two, in general skew, straight lines

$$\boldsymbol{a}_0(v) = (1-v)\boldsymbol{P}_{00} + v\boldsymbol{P}_{10} \text{ and } a_1(v) = (1-v)\boldsymbol{P}_{01} + v\boldsymbol{P}_{11}$$

parameterized in the interval $[0, 1]$

Geometric Interpretation



Every parameter value v_0 corresponds exactly to two points $\boldsymbol{a}_0(v_0), \boldsymbol{a}_1(v_1)$ on respectively on of the straight lines $\boldsymbol{a}_0$ and $\boldsymbol{a}_1$. Between these two points we interpolate again over a straight line

$$\boldsymbol{q}(u,v) = (1-u)\boldsymbol{a}_0(v) + u\boldsymbol{a}_1(v) = (1-u)(1-v)\boldsymbol{P}_{00} + (1-u)v\boldsymbol{P}_{10} + u(1-v)\boldsymbol{P}_{01} + uv\boldsymbol{P}_{11}$$

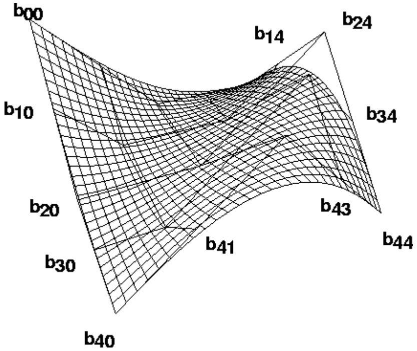
defined in the interval $[0,1]$. The so obtained surface is a hyperbolic paraboloid. By plugging the coordinates $\boldsymbol{P}_{00} = (0,0,0), \boldsymbol{P}_{01} = (1,0,0), \boldsymbol{P}_{10} = (0,1,0), \boldsymbol{P}_{11} = (1,1,1)$ into the points, we get $\boldsymbol{q}(u,v) = (u,v,uv)$.

Bézier-Tensor Product Surfaces

Use Bernstein-tensor product basis in $[0,1]$

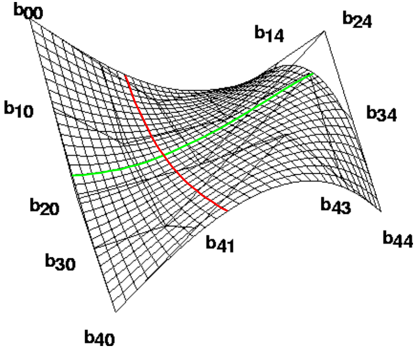
$$\boldsymbol{q}(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} b_{ij} B_i^m(u) B_j^n(v), \text{ where } b_{ij} \in \mathbb{R}^d$$

The coefficients b_{ij} are called Bézier points and form the control grid.



Bézier-Tensor Product Surfaces

- **Properties:** The coefficients b_{ij} are called Bézier points and form the control grid



- The parameter lines provide Bézier curves

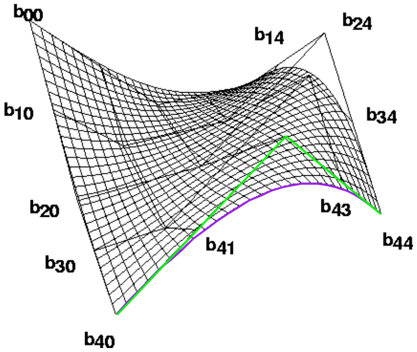
$$\boldsymbol{q}(u_0, v) = \sum_{j=0}^{n} \left(\sum_{i=0}^{m} b_{ij} B_i^m(u_0) \right) B_j^n(v) = \sum_{j=0}^{n} b_j(u_0) B_j^n(v)$$

Bézier-Tensor Product Surfaces

- The surface is located for $(u,v) \in [0,1]^2$ in the convex hull of the control polygon

$$\sum_{j=0}^{n}\sum_{i=0}^{m} B_i^m(u)B_j^n(v) = \sum_{j=0}^{n} 1 \cdot B_j^n(v) = 1 \geq 0, \text{ for } (u,v) \in [0,1]$$

- The Bézier points on the boundary curves are the boundary points of the Bézier grid



Bézier-Tensor Product Surfaces

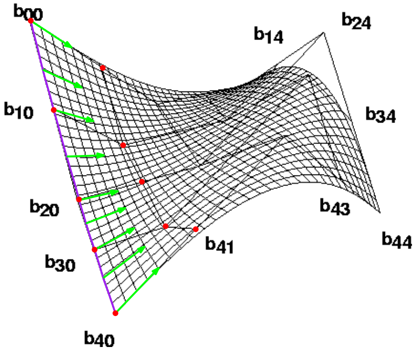
- The derivatives at the boundary points can be calculated like for the curves form the Bézier points of the boundary curves:

$$\frac{\partial \boldsymbol{q}}{\partial u}(u,v)\Big|_{u=0} = m\sum_{j=0}^{n} B_j^n(v)(b_{1j}-b_{0j})$$

$$\frac{\partial \boldsymbol{q}}{\partial v}(u,v)\Big|_{v=0} = n\sum_{i=0}^{m} B_i^m(u)(b_{i1}-b_{i0})$$

$$\frac{\partial \boldsymbol{q}}{\partial u}(u,v)\Big|_{u=1} = m\sum_{j=0}^{n} B_j^n(v)(b_{mj}-b_{(m-1)j})$$

$$\frac{\partial \boldsymbol{q}}{\partial v}(u,v)\Big|_{v=1} = n\sum_{i=0}^{m} B_i^m(u)(b_{in}-b_{i(n-1)})$$



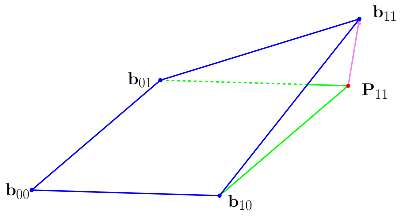
Twist Vectors

- The twist vectors at the corners are obtained via

$$\begin{aligned}
\boldsymbol{q}_{uv}(uv)|_{u=0,v=0} &= m \cdot n(b_{11} - b_{01} - b_{10} + b_{00})\\
\boldsymbol{q}_{uv}(uv)|_{u=0,v=1} &= n \cdot m(b_{1n} - b_{1(n-1)} - b_{0n} + b_{0(n-1)})\\
\boldsymbol{q}_{uv}(uv)|_{u=1,v=0} &= m \cdot n(b_{m1} - b_{(m-1)1} - b_{m0} + b_{(m-1)0})\\
\boldsymbol{q}_{uv}(uv)|_{u=1,v=1} &= n \cdot m(b_{mn} - b_{m(n-1)} - b_{(m-1)n} + b_{(m-1)(n-1)})
\end{aligned}$$

- Geometric interpretation of the twist vector:

$$\begin{aligned}
\boldsymbol{q}_{uv}(u,v)|_{u=0,v=0} &= m \cdot n(b_{11} - b_{01} - b_{10} + b_{00})\\
&= m \cdot n(b_{11} - \underbrace{(b_{00} + (b_{10} - b_{00}) + (b_{01} - b_{00}))}_{\boldsymbol{P}_{11}}
\end{aligned}$$



Evaluation of Points and Normals

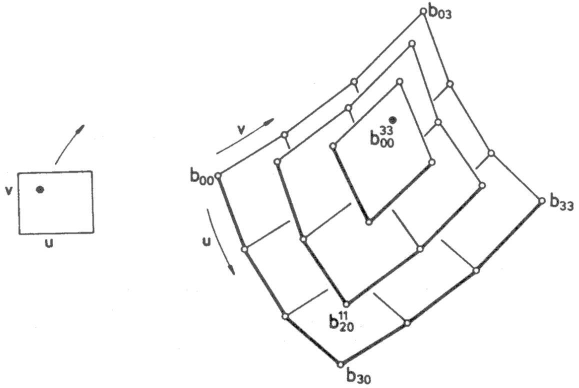
- **Evaluation:** apply de Casteljau-Algorithm twice

$$\boldsymbol{q}(u_0,v_0)=\sum_{j=0}^{n}\underbrace{\left(\sum_{i=0}^{m} b_{ij}B_i^m(u_0)\right)}_{\substack{\text{calculate for fixed } i,\, u_0 \text{ from}\\ b^0_{ij}(u_0)=b_{ij}(u_0)\\ \text{with de Casteljau}\\ b_j(u_0)=b^n_{ij}(u_0)}} B_j^n(v_0)=\underbrace{\sum_{j=0}^{n} b_j(u_0)B_j^n(v_0)}_{\text{evaluate with de Casteljau curve at } v=v_0}$$

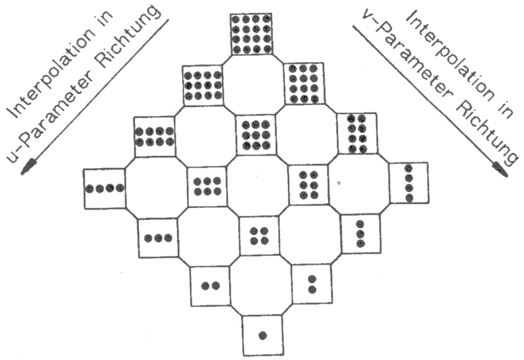
- Organize double loop well for $n<m$, i.e. smaller index in **inner** loop
- Analogous to Bézier curves the penultimate elements of the de Casteljau-scheme provide the directions of the partial derivatives q_u resp. q_v.

Evaluation of Points and Normals

- Besides the possibility of series connection of interpolation in $u-$ and $v-$direction, there is also the possibility for working alternately in $u-$ and $v-$ direction.
- **Example:** bicubic tensor product surface



- All in all, there are $\frac{(m+n)!}{m!n!}$ different evaluation possibilities of the surface point.
- **Example:** bicubic tensor product surface


Interpolation in u-Parameter Richtung
Interpolation in v-Parameter Richtung

Rational Bézier-surface

A rational Bézier-surface is defined by

$$\boldsymbol{R}(u,v) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} B_i^m(u)B_j^n(v)\cdot \tilde{b}_{ij}}{\sum_{i=0}^{m}\sum_{j=0}^{n} B_i^m(u)B_j^n(v)\cdot w_{ij}}$$

where

$$\tilde{b}_{ij} = \begin{cases} w_{ij}\cdot b_{ij}, & \text{if } w_{ij}\neq 0 \\ b_{ij}, & \text{if } w_{ij} = 0 \end{cases}$$

and $b_{ij}\in\mathbb{R}^3$, $0\le w_{ij}\in\mathbb{R}$, $w_{n0}=w_{0m}=1$, $i=0,\ldots,m$, $j=0,\ldots,n$

- The b_{ij} are called **Bézier-points**, w_{ij} are called **weights**.
- If all $w_{ij}=1$, the denominator is equal to one and $\boldsymbol{R}$ defines a nonrational Bézier-tensor product surface.

Rational Bézier-surface

It is usual to call the representation a tensor product surface, which is not completely correct. The basis functions

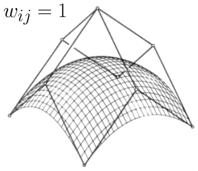
$$B(u,v) = \frac{w_{ij} B_i^m(u) B_j^n(v)}{\sum_{i=0}^m \sum_{j=0}^n B_i^m(u) B_j^n(v) \cdot w_{ij}}$$

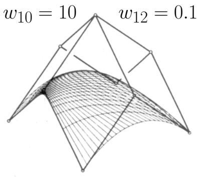
can not be decomposed in general into two factors due to the denominator!

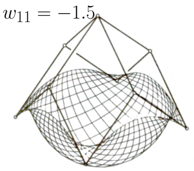
- The term has its reasoning in the interpretation as projection of a tensor product surface in $\mathbb{R}^4$
- Therefore rational Bézier-surfaces also possess a lot of properties of tensor product surfaces

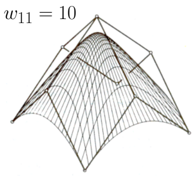
Rational Bézier-surface

- The properties discussed for rational Bézier-curves can be transferred analogously to the standard Bézier-tensor product surfaces also for rational Bézier-surfaces.
- **Example:** biquadric rational Bézier-surface









Rational Surfaces

(Semi-)circle as rational quadratic Bézier-curve

Choose homogeneous Bézier-points

$$\boldsymbol{b}_0 = (-r, 0, 1), \quad \boldsymbol{b}_1 = (0, r, 0), \quad \boldsymbol{b}_2 = (r, 0, 1)$$

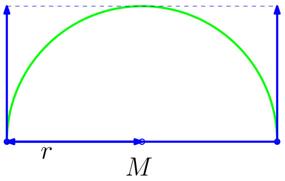
i.e.

$$w_0 = w_2 = 1, \quad w_1 = r$$

Then

$$\Pi(F(u)) = \left(\frac{F_x(u)}{F_w(u)}, \quad \frac{F_y(u)}{F_w(u)} \right) = R(u)$$

provides the equation of a (semi-)circle of radius r.

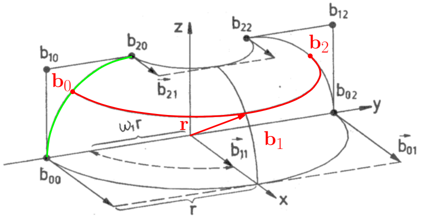


Rational Surfaces

Rational surfaces as rational Bézier-surfaces:

- Let the z-axis be the rotation axis.
- Let the meridian-curve of the surface be given in the (y, z) plane and a rational Bézier-curve of degree n.
- An arbitrary point $P = (0, r, z)$ on the meridian-curve moves during rotation on the curve given by the homogeneous Bézier-points

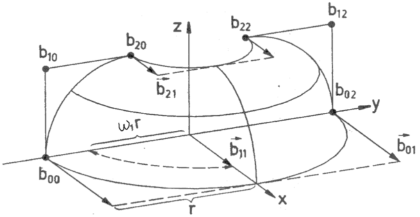
$$\boldsymbol{b}_0 = (0, -r, z, 1)$$
$$\boldsymbol{b}_1 = (r, 0, 0, 0)$$
$$\boldsymbol{b}_2 = (0, r, z, 1)$$



Rational Surfaces

The curve points b_0, b_n lie on the meridian-curve, i.e. the curve points $b_{00}, b_{01}, b_{02}, b_{n0}, b_{n1}, b_{n2}$ result accordingly to the formula above.

Due to the affine invariance ot the rational Bézier-curves, the rotation of every curve point of the meridian-curve can be attributed to the rotation of Bézier-points of the meridian, i.e. the formula above holds also for the inner Bézier-points $b_{i0}, b_{i1}, b_{i2}, \quad i = 1, \ldots, n-1$



TPPatches

To be able to connect two Bézier-patches

$$\boldsymbol{q}_1(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} b_{ij} B_i^m(u) B_j^n(v)$$

and

$$\boldsymbol{q}_2(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} c_{ij} B_i^m(u) B_j^n(v)$$

along the boundary curve in $v-$direction, first the two patches have to have the same boundary curve in $v-$direction, i.e. for the Bézier-points

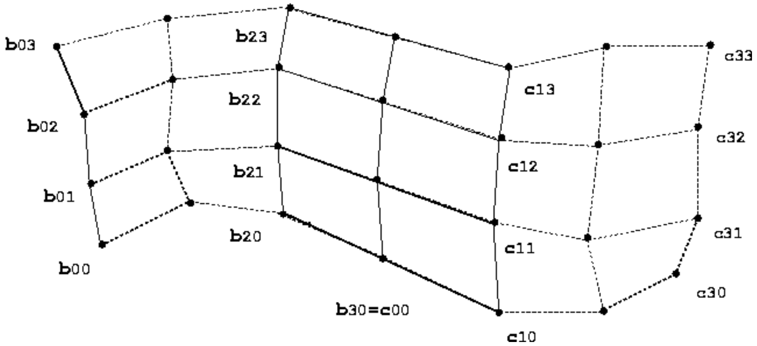
$$b_{mj} = c_{0j}, \quad j = 0, \ldots, n$$

holds, and second, along the boundary curve, i.e. for every fixed v, the derivatives in $u-$direction have to be identical.

Interfaces between Bézier-TPPatches

The derivatives in $u-$direction along the boundary curve $v = 1$, resp. $v = 0$ are identical if, and only if

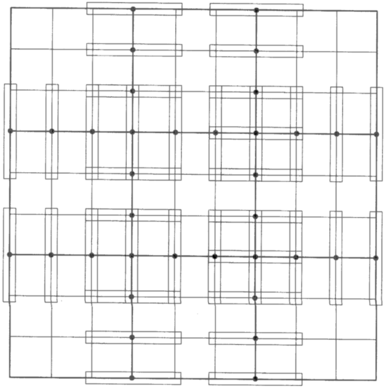
$$b_{mj} - b_{m-1} = c_{1j} - c_{0j}$$



If only G^1-continuity is expected, then the polygon segments have to be co-linear, but do not have to be in a certain aspect ratio.

Interfaces between Bézier-TPPatches

Composing multiple Bézier-patches



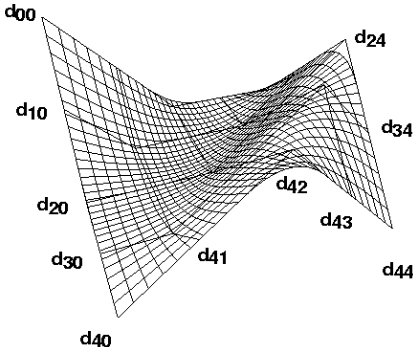
Please note: **three** Bézier-patches **can not** be connected C^1- or G^1-continuously in the same point.

BSpline-Tensor Product Patches

Use BSpline-tensor product basis in $[s_0, \ldots, s_{m+o+1}] \times [t_0, \ldots, t_{m+p+1}]$

$$\boldsymbol{q}(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} d_{ij} N_i^o(u) N_j^p(v), \text{ where } d_{ij} \in \mathbb{R}^d$$

- The coefficients d_{ij} are called **deBoor-points** and form the control grid.
- Like for Bézier-tensor product patches the properties of the curves are passed to the BSpline-tensor product patches.



Parametric Curves and Surfaces

Advantages

- Easy to generate points on the curve/surface
- Separate x, y, z components

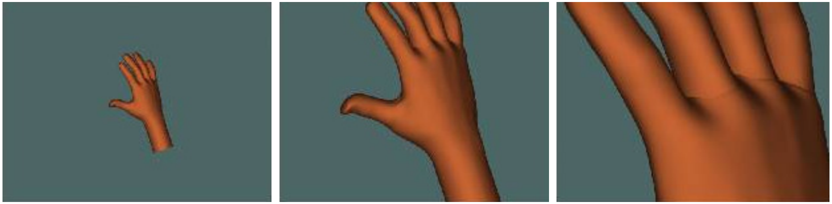
Disadvantages

- Hard to determine inside/outside
- Hard to determine if a point is **on** the curve/surface

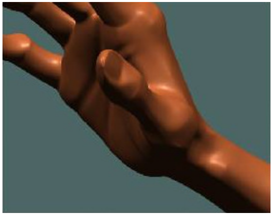
Subdivision Surfaces

Subdivision Surfaces

- Spline-Modelling (Toy Story): Assembling of Patches with smooth transitions is complicated

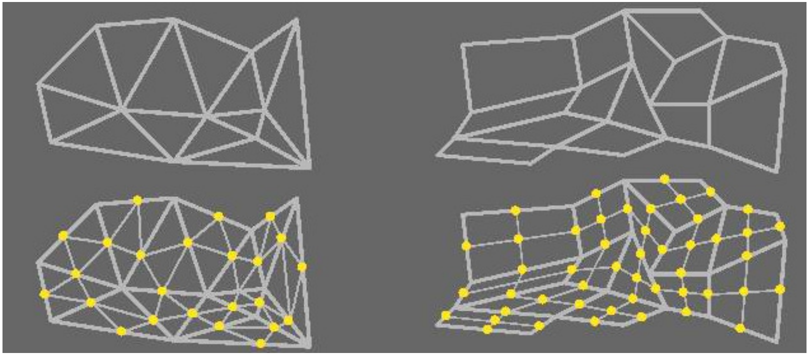


- Subdivision surfaces (Geri's Game): No problems during assembly



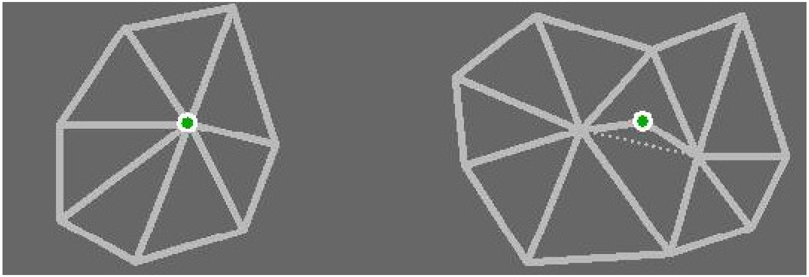
Subdivision Surfaces: Two rules

- **Topological rule:** Modifies the connectivity by insertion of new points



Subdivision Surfaces: Two rules

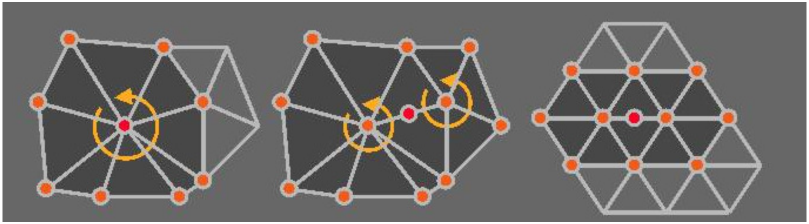
- **Geometric rule:** Calculating the position of new point by local linear combination of points



even on level i odd on level i

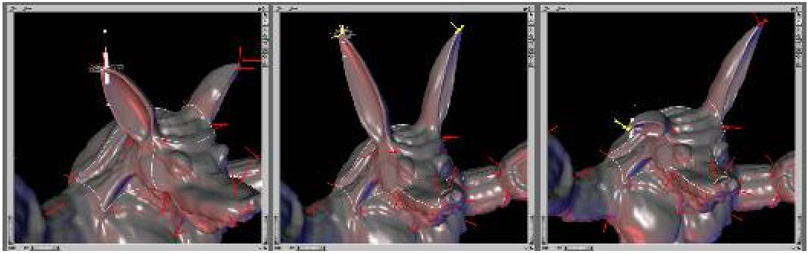
Desired Properties

- The calculation should be efficient
- The support of the subdivision rules should be compact
  - limited spatial influence of a control point
- Be affine invariant
- Achieve a certain smoothness (continuity)
  - C^1-continuity is generally easy to achieve
  - C^2-continuity is significantly more complicated



Subdivision Surfaces

- 1978 Doo-Sabin scheme[1], Catmull-Clark scheme[2]
- 1995 U. Reif:U. Reif. "A unified approach to subdivision algorithms near extraordinary vertices". In: *Computer Aided Geometric Design* 12.2 (1995), pp. 153–174
- Since 1995 a couple of other schemes was introduced

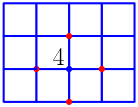


[1]Doo and Sabin, 1978.
[2]Catmull and Clark, 1978.

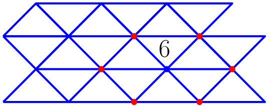
Subdivision of regular nets

Nets with facettes and nodes of the same order

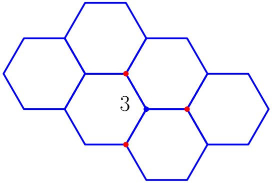
Rectangles



Triangles

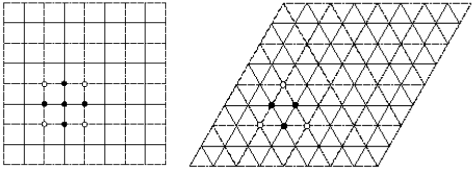


Hexagons

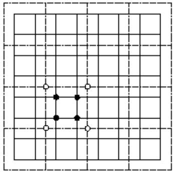


Subdivision of regular nets

- Vertex insertion (white=original points, black=new points)



- Corner cutting

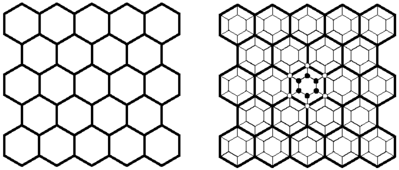


- Another distinction criterion: Approximated or Interpolated

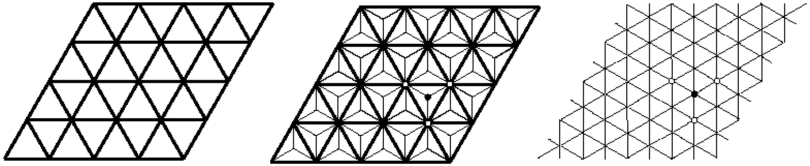
Subdivision of regular nets

Other nets and subdivision strategies

- Hexahedron: Old vertices are kept, for each face 6 new points are inserted



- $\sqrt{3}-$subdivision: for each face insert barycenter, then flip edges and scale resulting triangulation by $1/\sqrt{3}$ and rotate by 30°

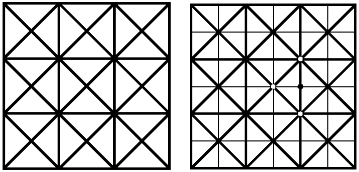


Subdivision of regular nets

Other nets and subdivision strategies

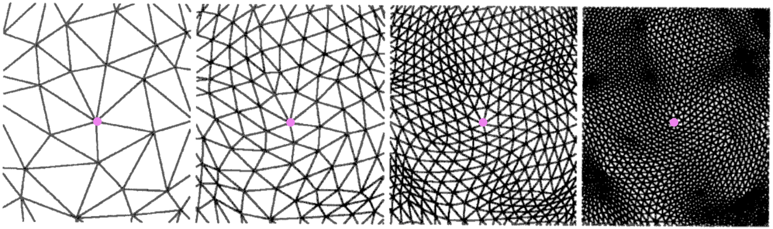
- Bisection: Used for so called $4 - 8$ subdivision. Subdivides the plane into identical triangles, but the order of the vertices is not constant.

For subdivision, the hypotenuse of the triangle is split and the resulting triangulation is scaled by $1/\sqrt{2}$ and rotated by 45°



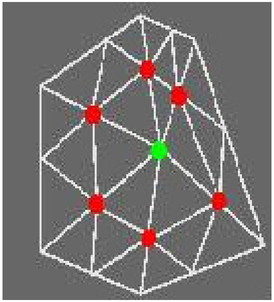
Subdivision Connectivity

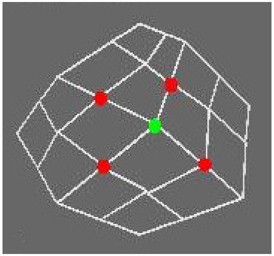
If regular subdivision techniques are applied to an arbitrary net, **all new generated points have a regular order**, i.e. an order 4 for quad nets, an order 6 for triangle nets and an order of 3 for a regular hexaeder subdivision.



The only irregular configurations can appear at the so called extraordinary points. The order of these points (surfaces) does not change during the regular subdivision.

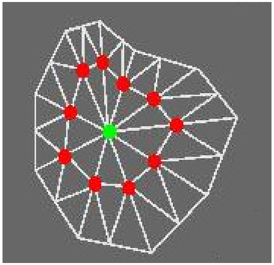
Extraordinary points

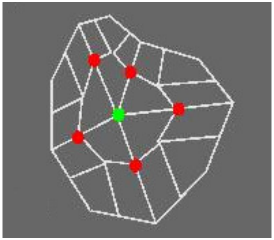




Regular points

valence$=6$ valence$=4$





Extraordinary points

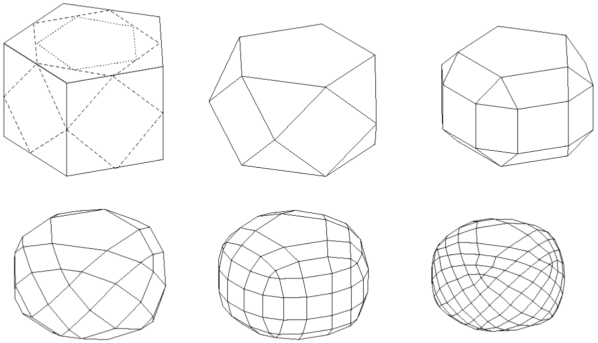
valence$\neq6$ valence$\neq4$

Subdivision Connectivity

- After a sufficient number of subdivision steps, the net mainly consists of large regular regions. These regular regions are only disturbed by a finite number of isolated singularities of the extraordinary points.
- An in such a way special connectivity of a net is called **Subdivision Connectivity**.
- After a couple of subdivisions, the convergence of the subdivision scheme is, due to its locality, not influenced by the extraordinary points
- Therefore, a subdivision scheme is initially designed and analyzed for regular nets. This solves the problem of surface generation almost everywhere.
- Only in the immediate surrounding of extraordinary points, special subdivision rules have to be found which guarantee the global convergence of the scheme.

Parameterization of Subdivision Surfaces

- In case of a surface the subdivision creates a sequence of polyhedrons with more and more points and faces. A simple for example is "cutting corners"[3].
- Intuitively the limit of this sequence of refined polyhedrons is the subdivision surface.



Reif, Peters, 1997

The simplest subdivision scheme for smoothing polyhedra

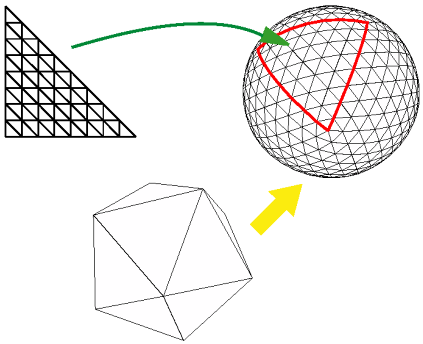
- For arbitrary polyhedrons the subdivision surface **cannot be parameterized over a planar parameter domain**

[3] J. Peters and U. Reif. "The simplest subdivision scheme for smoothing polyhedra". In: *ACM Transactions on Graphics (TOG)* 16.4 (1997), pp. 420–431.

Parameterization of Subdivision Surfaces

Therefore, subdivision surfaces are parameterized **over the initial polyhedron**:

- In case of midpoint subdivision of triangles, i.e. the subdivision rules are applied separately to the triangles of the initial polyhedron. Each point on its surface corresponds therefore to a point on the surface.



Parameterization of Subdivision Surfaces

This leads to the same situation as for curves: We have a sequence of piecewise linear functions over the same parameter domain.

If this sequence converges uniformly, we obtain as the limit a funcion f of the set of surface points $|K|$ of the initial polyhedron $K \in \mathbb{R}^3$, i.e.

$$f\colon |K| \to \mathbb{R}^3$$

Comments:

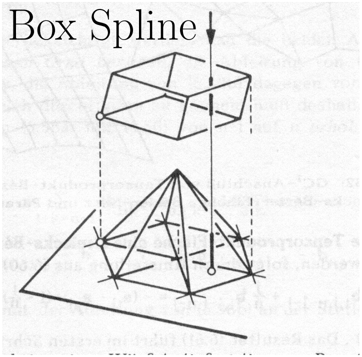
- If the subdivision scheme is reduced to a spline subdivision scheme in the regular case, the resulting parameterization is identical with the spline parameterization.
- For the parameterization of the subdivision surface it is important, that the initial polyhedrons do not intersect with each other (otherwise one point on the control polygon would be aassociated with several image points). This constraint can be avoided however, if the initial net is embedded into $\mathbb{R}^4$

Important Subdivision Schemes

Loop scheme:[4]

- Based on spline basis function (three-directional quadratic box-spline, C^2-continuous) over regular triangulation.
- The generating polynomial of this spline is

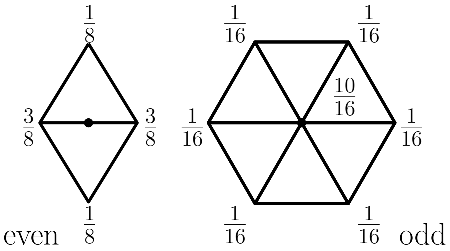
$$S(z_1, z_2) = \frac{1}{16}(1+z_1)^2(1+z_2)^2(1+z_1z_2)^2.$$



[4]Loop, 1987.

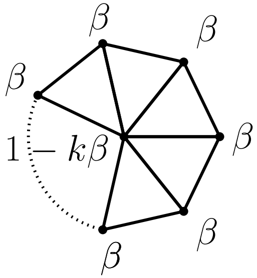
Subdivision Schemes: Loop

For the three-directional box spline the weights for the new points at each subdivision step shown in the picture are generated



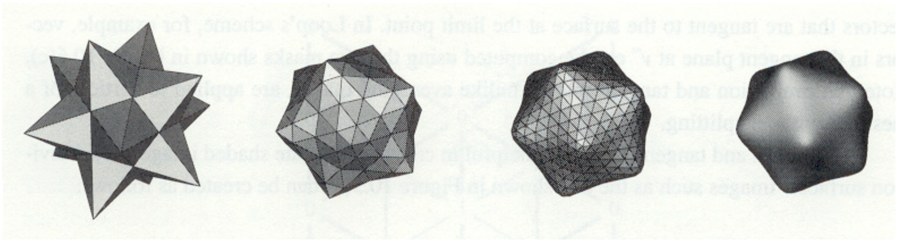
For the extraordinary points of order k Loop has proposed the following coefficients:

$$\beta = \frac{1}{k}\left(\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\left(\frac{2\pi}{k}\right)\right)^2\right)$$



Subdivision Schemes: Loop

Example:

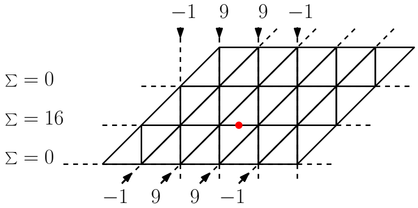


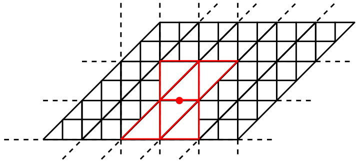
Notice: The polyhedrons generated during the subdivision process shrink more and more compared to the initial polyhedron.

Subdivision Schemes: Modified Butterfly

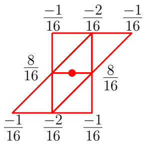
The Butterfly scheme[5] is a generalization of the four point interpolation scheme[6] on triangular meshes.

The bivariate subdivision rules should therefore deliver the four point rule when projected onto the grid lines:





Solution:



$\frac{-1}{16}$ $\frac{-2}{16}$ $\frac{-1}{16}$
$\frac{8}{16}$
$\frac{8}{16}$
$\frac{-1}{16}$ $\frac{-2}{16}$ $\frac{-1}{16}$

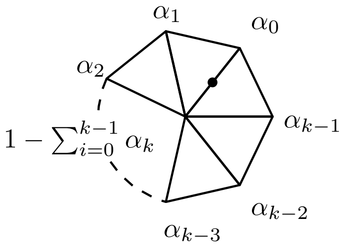
[5] Dyn, Levin, et al., 1990.

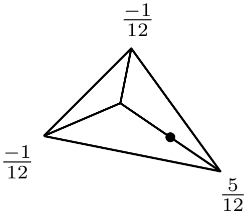
[6] Dyn, Gregory, et al., 1987.

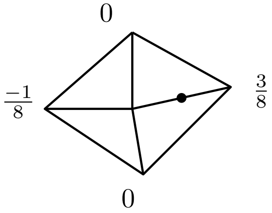
Subdivision Schemes: Modified Butterfly

Extraordinary points:

$$\alpha_i = \frac{1}{k}\left(\frac{1}{4} + \cos\left(\frac{2\pi i}{k}\right) + \frac{1}{2}\cos\left(\frac{4\pi i}{k}\right)\right), \quad i = 1, \ldots, k-1, \quad k > 5$$



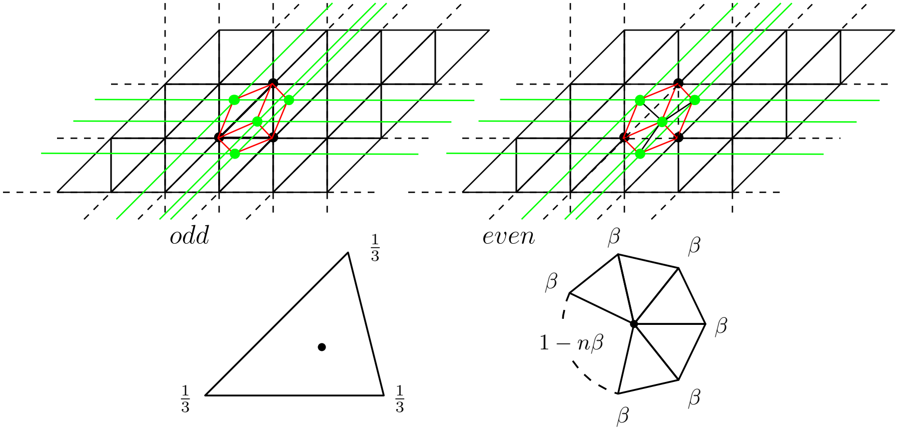




With this choice the modified Butterfly scheme delivers a C^1-continuous interpolating surface on arbitrary triangle nets. (Not C^2-continuous on regular nets)

Subdivision Schemes: $\sqrt{3}$

The $\sqrt{3}$-subdivision[7] scheme generates only one triangle per triangle at each subdivision step.

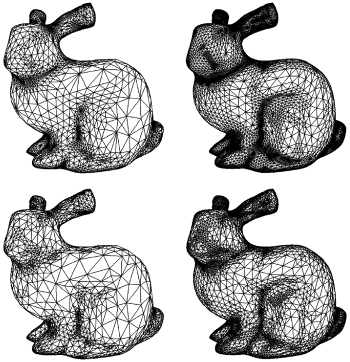


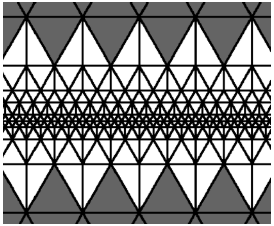
$$\beta(n) = \frac{4 - 2\cos(2\pi/n)}{9n}$$

[7]Kobbelt, 2000.

Subdivision Schemes: $\sqrt{3}$

Examples:





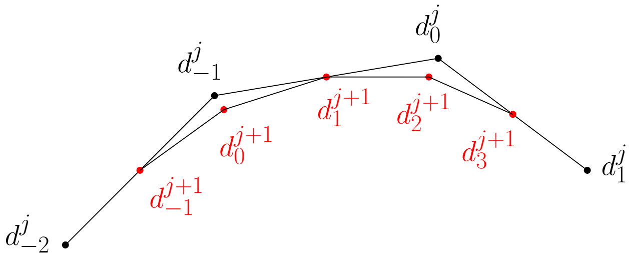
Local refinement

Subdivision Schemes: Catmull-Clark

The Catmull-Clark[8] subdivision is based on a bicubic tensorproduct surface. The subdivision scheme therefore delivers a surface which is C^2-continuous, except at the extraordinary points, where only C^1-continuity can be proven.

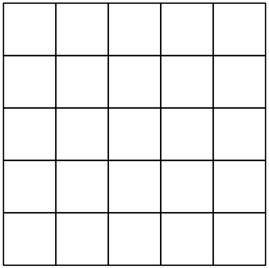
Reminder: cubic BSplines

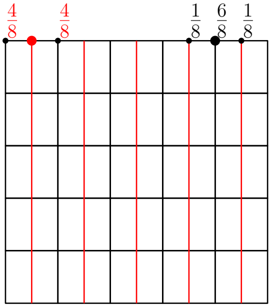
$$d_{2i+1}^{j+1} = \frac{1}{8}(4d_{j-1}^{j} + 4d_i^j)$$
$$d_{2i}^{j+1} = \frac{1}{8}(d_{i-2}^{j} + 6d_{j-1}^{j} + d_i^j)$$

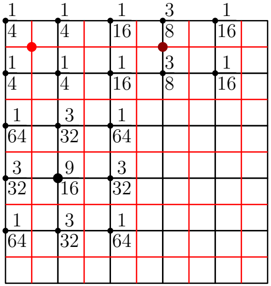


[8] Catmull and Clark, 1978.

Regular subdivision (Tensorproduct BSpline)



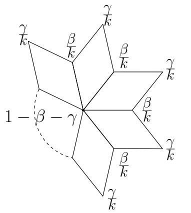




Subdivision Schemes: Catmull-Clark

Extraordinary points:

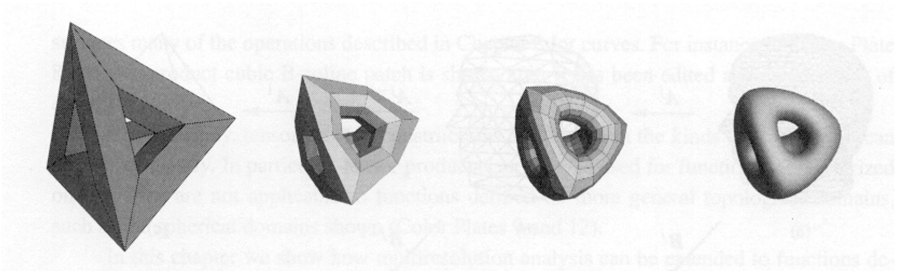
$$\beta = \frac{3}{2k}$$
$$\gamma = \frac{1}{4k}$$



This choice delivers a C^2-continuous limit surface almost everywhere. At the extraordinary points it is C^1.

Subdivision Schemes: Catmull-Clark

Example:

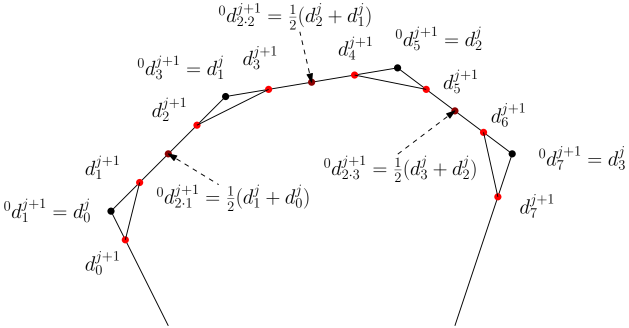


Subdivision Schemes: Doo-Sabin

The Doo-Sabin subdivision[9] is based on a quadratic tensorproduct surface. The subdivision scheme therefore delivers a surface, which is C^1-continuous.

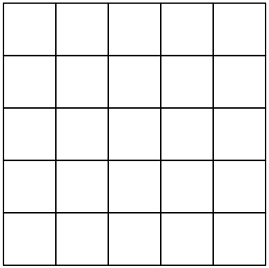
Reminder: Quadratic BSplines

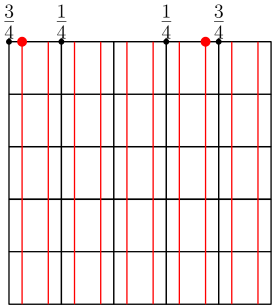
$$d_i^{j+1} = \frac{1}{4}(3d_{j-1}^j + d_i^j)$$
$$d_{i+1}^{j+1} = \frac{1}{4}(d_{i-1}^j + 3d_i^j)$$

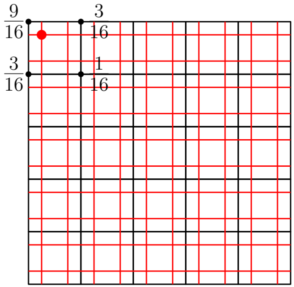


[9] Doo and Sabin, 1978.

Regular subdivision (Tensorproduct BSpline)



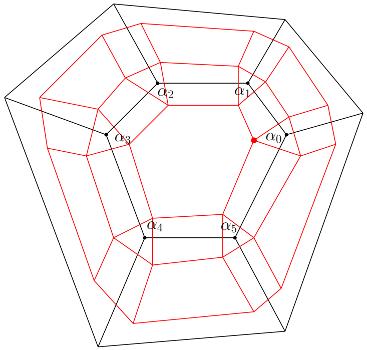




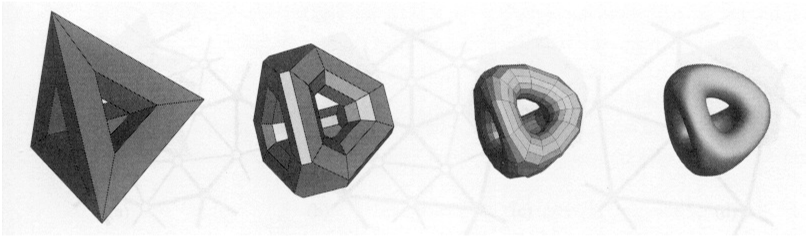
Subdivision Schemes: Doo-Sabin

Extraordinary points:

$$\alpha_0 = \frac{1}{4} + \frac{5}{4}k$$
$$\alpha_i = \frac{1}{4k}\left(3 + 2\cos\left(\frac{2\pi i}{k}\right)\right), \quad i = 1, \ldots, k-1$$



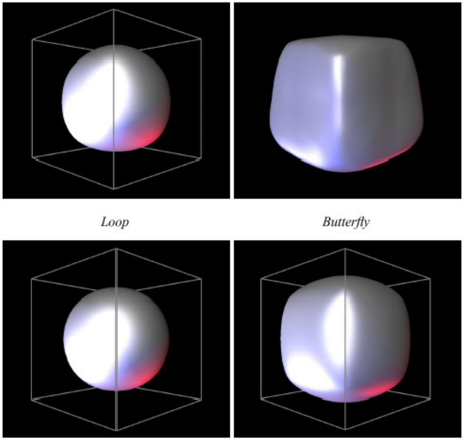
Example:



Comparison of Subdivision Schemes

Notice:

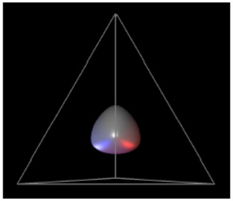
- Different smoothness
- Different symmetry (cube has to be triangulated in advance for a triangle-scheme)

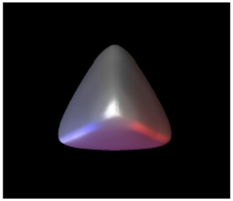


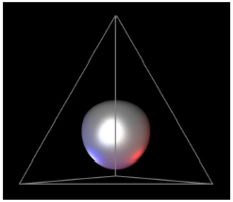
Comparison of Subdivision Schemes

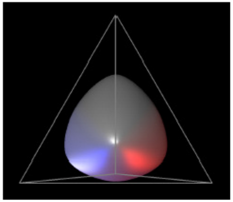
Notice:

- Different smoothness
- Different shrinking


Loop

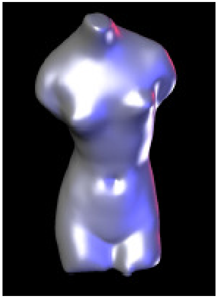

Butterfly

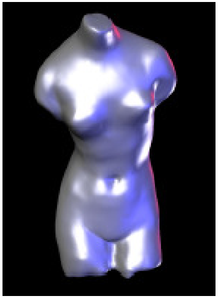

Catmull-Clark

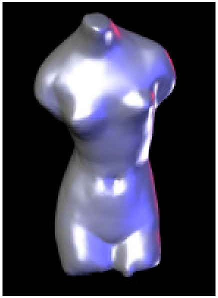

Doo-Sabin

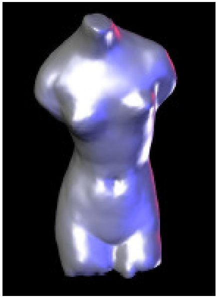
Comparison of Subdivision Schemes

Notice: For smooth triangular nets the results of different schemes are similar


Loop

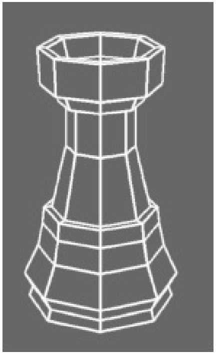

Butterfly


Catmull-Clark

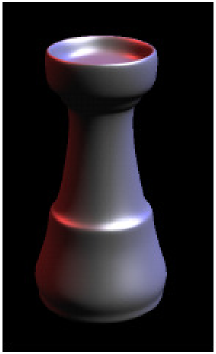

Doo-Sabin

Comparison of Subdivision Schemes

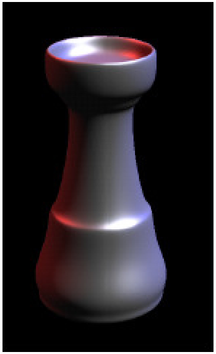
Notice: Triangulating the initial polyhedron before subdivision delivers very bad results for Catmull-Clark surfaces



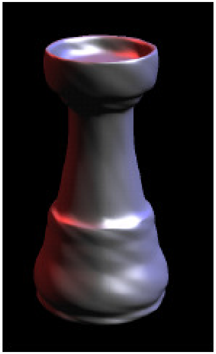
Initial mesh



Loop



Catmull-Clark



Catmull-Clark,after triangulation

Manifolds and Meshes

When working with 3D geometry, our meshes usually represent one or more discretized manifolds. But what *is* a manifold?

Manifolds and Meshes

When working with 3D geometry, our meshes usually represent one or more discretized manifolds. But what *is* a manifold?

Manifold

In mathematics, a manifold is a topological space that locally resembles Euclidean space near each point.

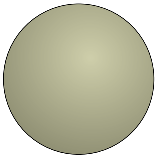
Manifolds and Meshes

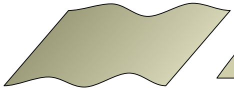
When working with 3D geometry, our meshes usually represent one or more discretized manifolds. But what *is* a manifold?

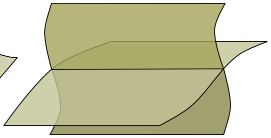
Manifold

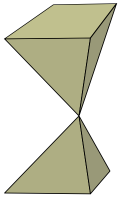
In mathematics, a manifold is a topological space that locally resembles Euclidean space near each point.

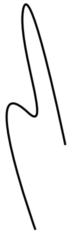
Let's see a few examples:


(a) 2-manifold


(b) 2-manifold


(c) Not a manifold


(d) Not a manifold


(e) 1-manifold

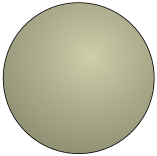
Manifolds and Meshes

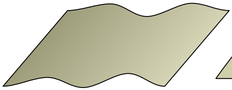
When working with 3D geometry, our meshes usually represent one or more discretized manifolds. But what *is* a manifold?

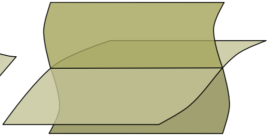
Manifold

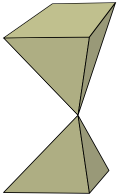
In mathematics, a manifold is a topological space that locally resembles Euclidean space near each point.

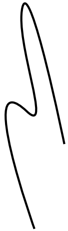
Let's see a few examples:


(a) 2-manifold


(b) 2-manifold

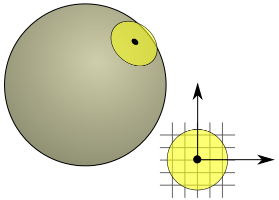

(c) Not a manifold

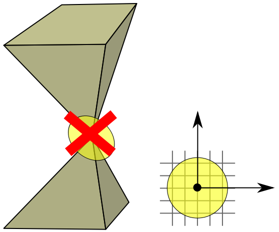

(d) Not a manifold


(e) 1-manifold

- ▶ The neighborhood of a point on a k-manifold "looks like" $\mathbb{R}^k$.
- ▶ Example: To each point of a 2-manifold, one can map an open neighborhood into an XY-plane.

Manifolds


(a) A local coordinate system around a point on a 2-manifold


(b) There is no unique coordinate system at the apex of the two pyramids

- Each point on a 2-manifold has an open neighborhood with a local XY-coordinate system.
- Points in the neighborhood can be parametrized using 2D coordinates.
- Analogy: Moving north/west/south/east on the surface of the earth.
- Computer Graphics: Many applications require manifold meshes.
- Most 3D models are composed of 2-manifold surface meshes.

Meshes

There are different types of meshes with different quality:

- "Triangle Soups": A set of triangles without any connectivity or other common properties.
    - Triangle soups still may contain some structure, e.g., triangles may overlap only at vertices and edges but use multiple vertices for the same point in space.
- General Meshes: A usual mesh consists of triangles which share vertices and thus have shared edges at their intersection, so they form a common surface. Such a mesh can still have parts of bad quality
    - Cracks
    - Self-Intersections
    - Overlaps of different mesh parts
- Surface Meshes: A triangulated surface without cracks, which may still contain non-manifold parts (T-pieces)
- 2-Manifold Surface Meshes: A triangulated 2-manifold.
- "Watertight" 2-Manifold Meshes: A 2-manifold mesh without boundary edges.

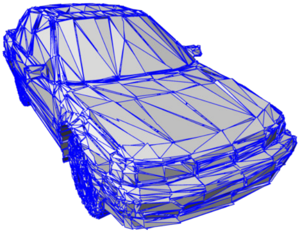
Meshes

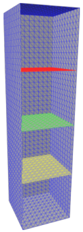
There are different types of meshes with different quality:

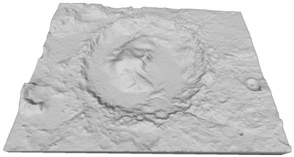
- "Triangle Soups": A set of triangles without any connectivity or other common properties.
  - Triangle soups still may contain some structure, e.g., triangles may overlap only at vertices and edges but use multiple vertices for the same point in space.
- General Meshes: A usual mesh consists of triangles which share vertices and thus have shared edges at their intersection, so they form a common surface. Such a mesh can still have parts of bad quality
  - Cracks
  - Self-Intersections
  - Overlaps of different mesh parts
- Surface Meshes: A triangulated surface without cracks, which may still contain non-manifold parts (T-pieces)
- 2-Manifold Surface Meshes: A triangulated 2-manifold.
- "Watertight" 2-Manifold Meshes: A 2-manifold mesh without boundary edges.

Quality is still application dependent. Texturing, for example, usually requires cutting watertight meshes into 2-manifold patches.

Mesh Examples


(a) A car geometry as triangle soup.


(b) A non-manifold model with T-junctions.


(c) A 2-manifold mesh of the gale crater on mars (by NASA)

(d) A watertight 2-manifold mesh. (From the Stanford 3D model repository)

Mesh Data Structures

There are different data structures for meshes:

- Vertex and face lists
- Adjacency matrices
- Half-edge structure
- Many more (e.g., the winged-edge structure)[10],[11].

[10]K. Weiler. "Edge-based data structures for solid modeling in curved-surface environments". In: *IEEE Computer graphics and applications* 5.1 (1985), pp. 21–40.

[11]P. Lienhardt. "Topological models for boundary representation: a comparison with n-dimensional generalized maps". In: *Computer-aided design* 23.1 (1991), pp. 59–82.

Vertex and Face Lists

Idea (Computer Graphics): What are meshes made of?

Vertex and Face Lists

Idea (Computer Graphics): What are meshes made of? Vertices and polygons.

Vertex and Face Lists

Idea (Computer Graphics): What are meshes made of? Vertices and polygons.

- Store a list of vertices and a list of polygons.
- $V = [v_1, \ldots, v_n]$ is an *ordered* list of vertices, usually with $V \subset \mathbb{R}^3$, this means each vertex has a unique index.
- $F = \{[v_i, v_j, v_k], \ldots\}$ is a set of faces defined by ordered tuples of 3 or more distinct vertex indices.

Vertex and Face Lists

Idea (Computer Graphics): What are meshes made of? Vertices and polygons.

- Store a list of vertices and a list of polygons.
- $V = [v_1, \dots, v_n]$ is an *ordered* list of vertices, usually with $V \subset \mathbb{R}^3$, this means each vertex has a unique index.
- $F = \{[v_i, v_j, v_k], \dots\}$ is a set of faces defined by ordered tuples of 3 or more distinct vertex indices.

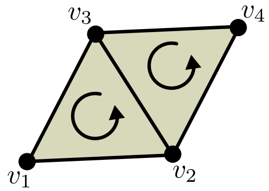


Figure: A simple mesh with two oriented triangles.

- Example: $V = \{v_1, v_2, v_3, v_4\}$ with $v_i \in \mathbb{R}^3$, $F = \{[v_1, v_2, v_3], [v_2, v_4, v_3]\}$.
- The counter-clockwise vertex order defines the orientations of the triangles.

Vertex and Face Lists

Idea (Computer Graphics): What are meshes made of? Vertices and polygons.

- Store a list of vertices and a list of polygons.
- $V = [v_1, \dots, v_n]$ is an *ordered* list of vertices, usually with $V \subset \mathbb{R}^3$, this means each vertex has a unique index.
- $F = \{[v_i, v_j, v_k], \dots\}$ is a set of faces defined by ordered tuples of 3 or more distinct vertex indices.

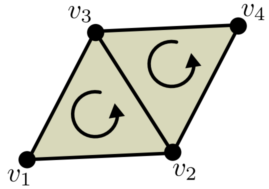


Figure: A simple mesh with two oriented triangles.

- Example: $V = \{v_1, v_2, v_3, v_4\}$ with $v_i \in \mathbb{R}^3$, $F = \{[v_1, v_2, v_3], [v_2, v_4, v_3]\}$.
- The counter-clockwise vertex order defines the orientations of the triangles.
- Real-world application: The Wavefront OBJ file format.
- Advantages: It is easy to insert new vertices and faces, efficient storage.
- Disadvantages: Neighborhood queries are costly and edges are only stored implicitly.

Neighborhood Queries

- There are nine different types of neighborhood queries[12],[13].
- Depending on the data structure, some queries are easy, and some are complicated.
- Which ones are actually needed depends on the application.

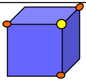
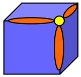
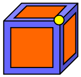
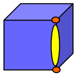
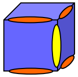
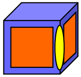
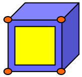
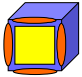
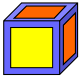
	Vertex	Edge	Face
Vertex			
Edge			
Face			

Table: Given a mesh element (yellow) we want to query adjacent mesh elements (orange).

[12] K. Weiler. "Edge-based data structures for solid modeling in curved-surface environments". In: *IEEE Computer graphics and applications* 5.1 (1985), pp. 21–40.

[13] T. Woo. "A combinatorial analysis of boundary data structure schemata". In: *IEEE Computer Graphics and Applications* 5.03 (1985).

Mesh Data Structures

Every approach has its advantages and disadvantages[14].

- Is fast access to arbitrary mesh elements required?
- Or is it okay to store some parts only implicitly?
- Does the structure have to be fast to build?
- Should it be optimized for performance or memory?
- Which queries (e.g., neighbor elements) are needed?

[14] X. Ni and M. S. Bloor. “Performance evaluation of boundary data structures”. In: *IEEE Computer Graphics and Applications* 14.6 (1994), pp. 66–77.

Mesh Data Structures

Every approach has its advantages and disadvantages[14].

- Is fast access to arbitrary mesh elements required?
- Or is it okay to store some parts only implicitly?
- Does the structure have to be fast to build?
- Should it be optimized for performance or memory?
- Which queries (e.g., neighbor elements) are needed?

- There is no one-size-fits-all structure.
- It is important to know when to use which structure.

[14] X. Ni and M. S. Bloor. "Performance evaluation of boundary data structures". In: *IEEE Computer Graphics and Applications* 14.6 (1994), pp. 66–77.

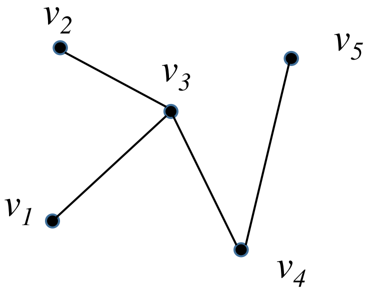
Adjacency matrices

Idea 1 (graph theory): Store the adjacency information in a lookup table.

Adjacency matrices

Idea 1 (graph theory): Store the adjacency information in a lookup table.

- The adjacency matrix of a simple labeled graph is a matrix with rows and columns labeled by graph vertices, with a 1 or 0 in position (v_i, v_j) according to whether v_i and v_j are adjacent or not.

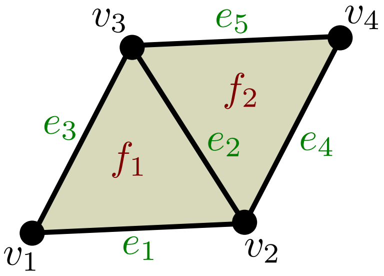


$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure: The adjacency matrix of a simple graph

Adjacency matrices

Idea 2: Extend this representation to inlcude edges, faces, etc.



	f_1	f_2
e_1	1	0
e_2	1	1
e_3	1	0
e_4	0	1
e_5	0	1

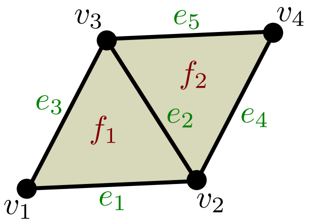
(a) The face-edge adjacency table

	e_1	e_2	e_3	e_4	e_5
v_1	1	0	1	0	0
v_2	1	1	0	1	0
v_3	0	1	1	0	1
v_4	0	0	0	1	1

(b) The edge-vertex adjacency table

Adjacency matrices

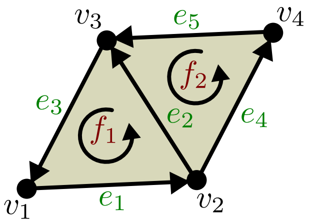
- Use a matrix as simple lookup table.



$$\partial_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \quad \partial_1 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Adjacency matrices

- Use a matrix as simple lookup table.

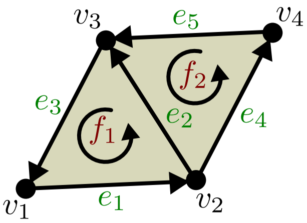


$$\partial_2 = \begin{pmatrix} 1 & 0 \\ 1 & -1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \quad \partial_1 = \begin{pmatrix} -1 & 0 & 1 & 0 & 0 \\ 1 & -1 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 \end{pmatrix}$$

- We assign each mesh element an arbitrary but fixed orientation.
- We store the relative orientation using a *signed* adjacency matrix.
- A combined lookup of several elements is possible using matrix-vector multiplication between ∂_k and an inidcator vector.
- Inner elements cancel out each other, therefore we call ∂_k the *boundary operator*.

Adjacency matrices

- Use a matrix as simple lookup table.

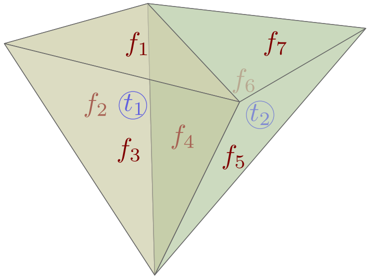


$$\partial_2 = \begin{pmatrix} 1 & 0 \\ 1 & -1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \quad \partial_1 = \begin{pmatrix} -1 & 0 & 1 & 0 & 0 \\ 1 & -1 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 \end{pmatrix}$$

- We assign each mesh element an arbitrary but fixed orientation.
- We store the relative orientation using a *signed* adjacency matrix.
- A combined lookup of several elements is possible using matrix-vector multiplication between ∂_k and an inidcator vector.
- Inner elements cancel out each other, therefore we call ∂_k the *boundary operator*.
- Advantages: Easy to implement structure, fast direct adjacency tests, each element has an unique index. Useful for schemes like Discrete Exterior Calculus (DEC).
- Disadvantages: More memory required, no cheap query for adjacency like "next edge of the triangle".

3D Adjacency matrices

- A tetrahedral mesh also has an matrix for the tetrahedron to face adjacency information.



(a) Two tetrahedra with a shared face

	t_1	t_2
f_1	1	0
f_2	1	0
f_3	1	0
f_4	1	1
f_5	0	1
f_6	0	1
f_7	0	1

(b) The neighborhood table

$$\partial_3 := \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & -1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}$$

(c) The adjacency matrix

Other adjacency matrices not shown here:

- The 7×9 face-to-edge matrix ∂_2.
- The 9×5 edge-to-vertex matrix ∂_1.

Halfedge Data Structure

- Idea (Geometric Modeling): Use pointers to connect adjacent elements[15].

[15] M. Mäntylä. "Topological analysis of polygon meshes". In: *Computer-aided design* 15.4 (1983), pp. 228–234.

Halfedge Data Structure

- Idea (Geometric Modeling): Use pointers to connect adjacent elements[15].
- Split each edge e_i into two *half-edges* h_i and h'_i with opposite orientation and assign each triangle the half-edge that matches its orientation.
- For triangles at the boundary, one of the edges belongs to the boundary.

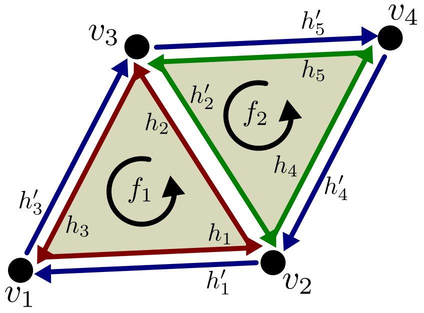


Figure: The half-edges of two adjacent triangles.

[15] M. Mäntylä. "Topological analysis of polygon meshes". In: *Computer-aided design* 15.4 (1983), pp. 228–234.

Halfedge Data Structure

- Idea: Use pointers to connect adjacent elements.
- Split each edge e_i into two *half-edges* h_i and h_i' with opposite orientation and assign each triangle the half-edge that matches its orientation.
- For triangles at the boundary, one of the edges belongs to the boundary.

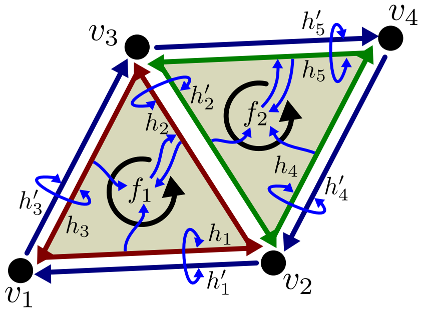


Figure: The half-edges of two adjacent triangles.

- Using pointers between half-edges, faces, and vertices, all important queries can be implemented efficiently.
- Cannot store meshes with non-manifold edges (T-pieces).

Halfedge Data Structure

- A half-edge has a pointer to . . .
  - The vertex it is pointing to.
  - The opposite half-edge.
  - The face it belongs to.
  - The next half-edge (and optionally to the previous half-edge).
- A face has a pointer to one of its half-edges.
- A vertex has a pointer to one of its outgoing half-edges.
- These pointers are sufficient to navigate the mesh.
  - An adjacent face can be found using the face-pointer of the opposite half-edge.
  - The edges (and vertices) of a face can be found by starting at an arbitrary half-edge of the face and then following the next-pointers.
  - A half-edge has a pointer to its end point. Its start point is the end point of the opposite half-edge.
  - Exercise: How can we find the triangles adjacent to a given vertex?

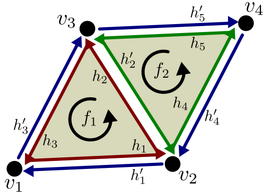


Figure: The half-edges of two adjacent triangles.

Simplicial Complex

Convex Set

A set $S \subseteq \mathbb{R}^n$ is convex, when for each $p, q \in S$ and $\alpha \in [0, 1]$ the point $r = \alpha p + (1 - \alpha)q$ is in S.

Simplicial Complex

Convex Set

A set $S \subseteq \mathbb{R}^n$ is convex, when for each $p, q \in S$ and $\alpha \in [0, 1]$ the point $r = \alpha p + (1 - \alpha)q$ is in S.

- Given two points from the set S, the line connecting the points is completely inside the set.
- Examples:

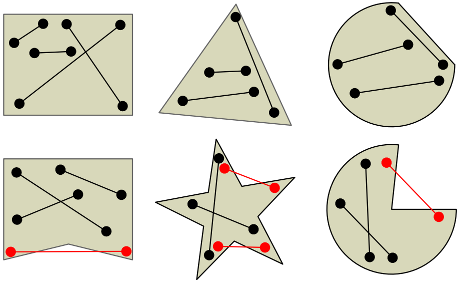


Figure: In a convex set, each pair of points can "see" each other. In a non-convex set there are points, which cannot be connected without intersecting a boundary.

Simplicial Complex

Convex Hull

The convex hull of a set of points S in $\mathbb{R}^n$ is the smallest subset of $\mathbb{R}^n$, which contains all points in S and is convex.

Simplicial Complex

Convex Hull

The convex hull of a set of points S in $\mathbb{R}^n$ is the smallest subset of $\mathbb{R}^n$, which contains all points in S and is convex.

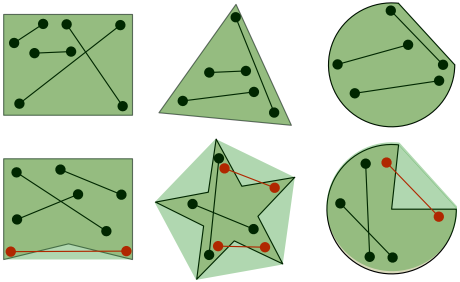


Figure: The convex hull of S contains all points, which lie between two points in S

- Simple idea: Add the missing points to the set.

$$\text{conv}(S) := \bigcup_{p,q \in S} \{\alpha p + (1-\alpha)q | \alpha \in [0,1]\}$$

Simplicial Complex

Simplex

A k-Simplex is the convex hull of $k+1$ affinely independent points.

Simplicial Complex

Simplex

A k-Simplex is the convex hull of $k+1$ affinely independent points.

- Generalization of known mesh elements like points, edges, triangles, and tetrahedra.
- Affinely independent: Not all points lie on a common line/plane/hyperplane.
- Very simple connectivity: Each point is connected to every other point.
- A k-simplex is the simplest k-dimensional polyhedron.
- Basic building blocks for tesselating k-dimensional polyhedra.
  - Generalization of triangulating polygons.

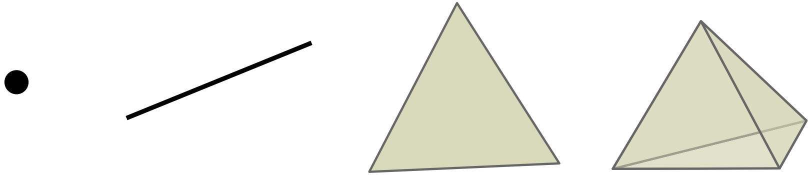


Figure: A 0-simplex (point), 1-simplex (edge), 2-simplex (triangle) and 3-simplex (tetrahedron).

- The generalization of a triangle mesh is a *simplicial complex*.
- Many mesh algorithms can be extended for arbitrary simplicial complexes.

Simplicial Complex

- **Barycentric coordinates** allow to define a coordinates system inside a simplex, which is independent of the position in space.
- How near is the point to the vertices of the simplex?
- The simplex vertices have the barycentric coordinates $(1, 0, \ldots, 0), \ldots, (0, \ldots, 0, 1)$ and $(0, \ldots, 0)$.
- Using linear interpolation one can describe each point p inside the simplex with a vector $(\alpha_1, \ldots, \alpha_n)$ as

$$\boldsymbol{p} = \sum_{k=1}^{n} \alpha_k v_k \quad \text{with } \alpha_k \in [0, 1] \text{ and } \sum_{k=1}^{n} \alpha_k = 1.$$

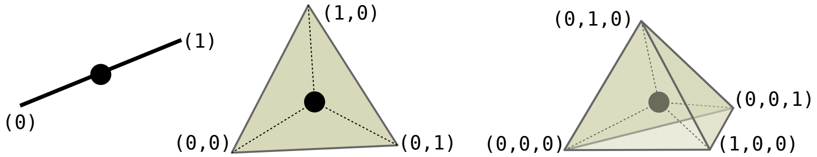


Figure: Barycentric coordinates describe the relative position inside a simplex

Simplicial Complex

Reference Simplex

The k-dimensional unit simplex is the simplex with the origin and the euclidean base vectors $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n$ as vertices.

- The vertices are $\boldsymbol{o} = (0, \ldots, 0), \boldsymbol{e}_1 = (1, 0, \ldots, 0), \ldots, \boldsymbol{e}_n = (0, \ldots, 0, 1)$.
- The barycentric coordinates of the vertices are the same as the euclidean ones.
- Many operations become easier by transforming a general simplex to the reference simplex first.
  - Example: The finite element method test functions are defined for the reference simplex to allow for easier computations.

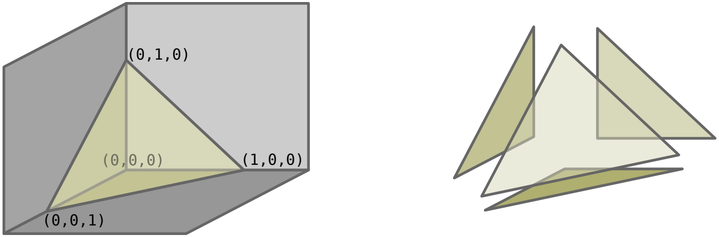


Figure: The reference simplex in 3D

Simplicial Complex

- A k-simplex is the convex hull of $k+1$ vertices, so it is uniquely defined by the set of its vertices.
- Any subset of the vertices of a simplex is a face of the simplex.
- Example: $\{v_1, v_2, v_3\}$ is a triangle and each subset $\{v_1, v_2\}, \{v_2, v_3\}$ and $\{v_3, v_1\}$ is an edge of the triangle. The subsets $\{v_1\}$, $\{v_2\}$ and $\{v_3\}$ describe the vertices of the triangle.
- We write $\sigma^{k-1} \prec \sigma^k$, when σ^{k-1} is a face of σ^k, and denote the set of all $(k-1)$-simplices on the boundary of σ^k as $\partial\sigma^k$.

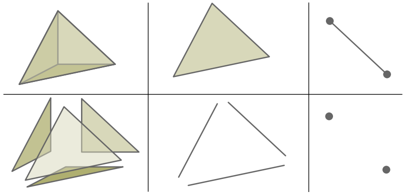


Figure: The $(k-1)$-boundaries of simplices with dimension $k = 3, 2, 1$.

Simplicial Complex

Simplicial Complex

A n-dimensional simplicial complex is a set of simplices, such that for each k-simplex all its ℓ-faces with $0 \leq \ell < k$ are also in the set and two simplices either do not intersect or intersect at a common face.

Simplicial Complex

Simplicial Complex

A n-dimensional simplicial complex is a set of simplices, such that for each k-simplex all its ℓ-faces with $0 \leq \ell < k$ are also in the set and two simplices either do not intersect or intersect at a common face.

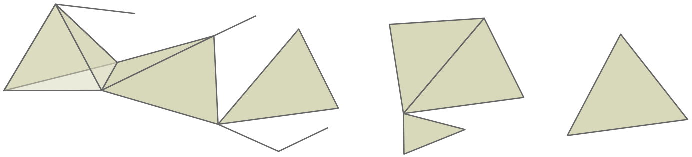


Figure: A valid simplicial complex.

Simplicial Complex

Simplicial Complex

A n-dimensional simplicial complex is a set of simplices, such that for each k-simplex all its ℓ-faces with $0 \leq \ell < k$ are also in the set and two simplices either do not intersect or intersect at a common face.

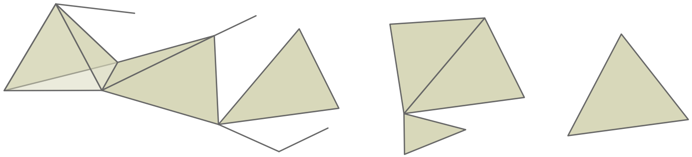


Figure: A valid simplicial complex.

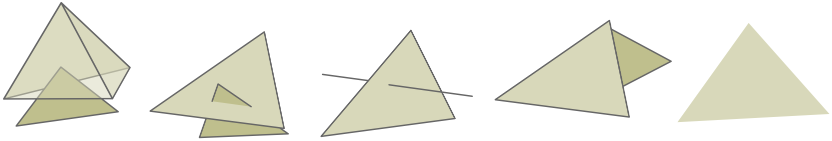


Figure: Different intersections, which are not in a common face. Note that the partial edge intersection in the second to last picture is no common face. In the last picture, the edges of the triangle are missing in the simplicial complex.

Abstract Simplices

- We treated simplices up to now as convex subsets of $\mathbb{R}^n$.
- We can define a simplex more abstract as a set of labeled vertices.
- Vertices are objects, which do not have to be the same as geometric points.
- An **arbitrary** set of vertex labels can be used as vertices.
- Example: $V = \{$😊, ☹, 😛, 🧙$\}$

Abstract Simplices

- We treated simplices up to now as convex subsets of $\mathbb{R}^n$.
- We can define a simplex more abstract as a set of labeled vertices.
- Vertices are objects, which do not have to be the same as geometric points.
- An **arbitrary** set of vertex labels can be used as vertices.
- Example: $V =$ {🙂, 🙁, 😛, 🧙}
  - The sets {🙂, 🙁, 🧙} and {🙁, 🙂, 😛} are valid 2-simplices.

Abstract Simplices

- We treated simplices up to now as convex subsets of $\mathbb{R}^n$.
- We can define a simplex more abstract as a set of labeled vertices.
- Vertices are objects, which do not have to be the same as geometric points.
- An **arbitrary** set of vertex labels can be used as vertices.
- Example: $V = \{$🙂, ☹️, 😛, 🧙$\}$
  - The sets $\{$🙂, ☹️, 🧙$\}$ and $\{$☹️, 🙂, 😛$\}$ are valid 2-simplices.
  - The set $\{$🙂, ☹️$\}$ is the common 1-face of the 2-simplices.

Abstract Simplices

- We treated simplices up to now as convex subsets of $\mathbb{R}^n$.
- We can define a simplex more abstract as a set of labeled vertices.
- Vertices are objects, which do not have to be the same as geometric points.
- An **arbitrary** set of vertex labels can be used as vertices.
- Example: $V = \{$😊, ☹, 😛, 🧙$\}$
  - The sets {😊, ☹, 🧙} and {☹, 😊, 😛} are valid 2-simplices.
  - The set {😊, ☹} is the common 1-face of the 2-simplices.
  - The full set of faces of {😊, ☹, 🧙} are
    the edges {😊, ☹}, {😊, 🧙}, {☹, 🧙}, and the vertices {😊}, {☹}, {🧙}

Abstract Simplices

- We treated simplices up to now as convex subsets of $\mathbb{R}^n$.
- We can define a simplex more abstract as a set of labeled vertices.
- Vertices are objects, which do not have to be the same as geometric points.
- An **arbitrary** set of vertex labels can be used as vertices.
- Example: $V = \{$🙂, ☹, 😀, 😴$\}$
  - The sets {🙂, ☹, 😴} and {☹, 🙂, 😀} are valid 2-simplices.
  - The set {🙂, ☹} is the common 1-face of the 2-simplices.
  - The full set of faces of {🙂, ☹, 😴} are the edges {🙂, ☹}, {🙂, 😴}, {☹, 😴}, and the vertices {🙂}, {☹}, {😴}
- We can draw simplicial complexes as graphs. In 2D there is a planar embedding.

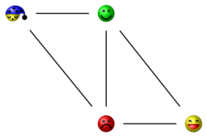


Figure: One possible representation of the simplicial complex in the example above

Abstract Simplices

- We treated simplices up to now as convex subsets of $\mathbb{R}^n$.
- We can define a simplex more abstract as a set of labeled vertices.
- Vertices are objects, which do not have to be the same as geometric points.
- An **arbitrary** set of vertex labels can be used as vertices.
- Example: $V = \{$🙂, ☹, 😛, 😴$\}$
  - The sets $\{$🙂, ☹, 😴$\}$ and $\{$☹, 🙂, 😛$\}$ are valid 2-simplices.
  - The set $\{$🙂, ☹$\}$ is the common 1-face of the 2-simplices.
  - The full set of faces of $\{$🙂, ☹, 😴$\}$ are
    the edges $\{$🙂, ☹$\}$, $\{$🙂, 😴$\}$, $\{$☹, 😴$\}$, and the vertices $\{$🙂$\}$, $\{$☹$\}$, $\{$😴$\}$
- We can draw simplicial complexes as graphs. In 2D there is a planar embedding.

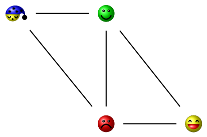


Figure: One possible representation of the simplicial complex in the example above

- Treating simplices as sets makes it easier to think about high-dimensional simplices.
- It is hard to embed a 5-simplex into 2D or 3D space, but it is easy to see that $\sigma^4 = \{v_2, v_3, v_4, v_5, v_6\}$ is a 4-face of the 5-simplex $\sigma^5 = \{v_1, v_2, v_3, v_4, v_5, v_6\}$.

Simplex Orientation

- So far, we have seen simplices without orientation.
- This makes sense when treating simplices like convex subsets of $\mathbb{R}^n$.
- Geometric objects often have an orientation, like edge direction, front and back sides, or inside and outside.
- By defining the simplex as an *ordered* set of vertices, we can define orientation for simplices of arbitrary dimension.
  - An edge $\sigma^1 = [v_1, v_2]$ points from v_1 to v_2
  - We look at the front side of triangle $\sigma^2 = [v_1, v_2, v_3]$, when the vertex order is counter-clockwise (right hand rule).
- Special case: A 0-simplex has no orientation.

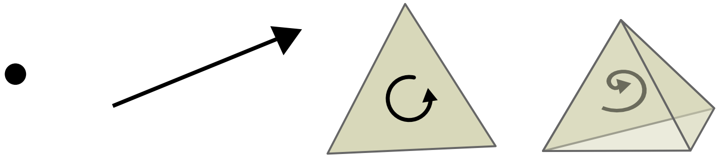


Figure: Oriented simplices of dimension $0, 1, 2$ and 3.

Permutations

- An ordered set can have an even and odd permutation, i.e., an even or odd number of swapped adjacent vertices. Even permutations describe the same simplex and odd permutations describe the simplex with flipped orientation.
- Example (edge): By swapping the vertices of the simplex $\sigma^1 = [v_1, v_2]$ we get the edge $-\sigma^1 = [v_2, v_1]$ with opposite orientation.
- Example (triangle): The simplex $[v_1, v_3, v_2]$ has the opposite orientation of σ^2.
- The simplex $[v_3, v_1, v_2]$ (two swaps) has the same orientation as σ^2.

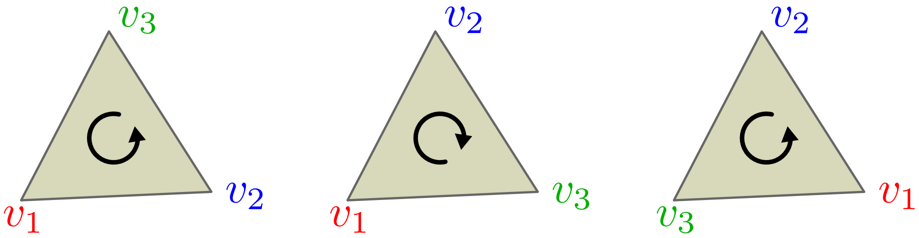
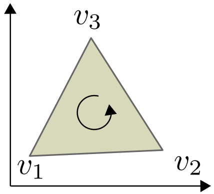


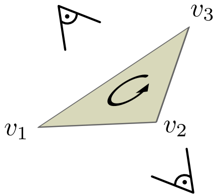
Figure: A 2-simplex (triangle) $[v_1, v_2, v_3]$, its odd permutation $[v_1, v_3, v_2]$ and its even permutation $[v_3, v_1, v_2]$

Orientation in $\mathbb{R}^n$

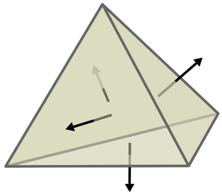
- For $k = n$, we can define an absolute orientation.
  - Example for $n = k = 2$: A triangle in $\mathbb{R}^2$ either has clockwise $(-)$ or counter-clockwise $(+)$ orientation.
  - For $n = 3$ and $k = 2$, it is not possible to define an absolute orientation, because one can "look" from different directions at a triangle in $\mathbb{R}^3$ and see different sides.
  - Example in $\mathbb{R}^3$: The normals of the faces of a tetrahedron either point into the tetrahedron $(-)$ or outward the tetrahedron $(+)$.



(a) A triangle in $\mathbb{R}^2$ can be oriented by its vertex order in the plane.



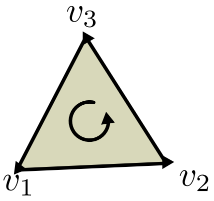
(b) In 3D a triangle does not have an absolute orientation.



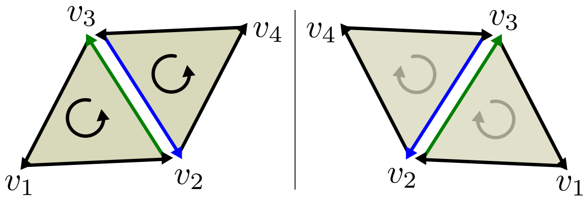
(c) A tetrahedron in $\mathbb{R}^3$ can be oriented by its surface normal.

Relative Orientation

- The orientation of the simplex induces an orientation of its faces, e.g., in 2D, the directed edges of a triangle are a cycle with the same orientation as the triangle.
- When a $(k-1)$ simplex belongs to the boundary of a k-simplex, it has a positive relative orientation when its orientation matches the orientation induced by the k-simplex.
- When two k-simplices have a shared $(k-1)$-face, we can define a relative orientation using the shared face.
- When the two adjacent k-simplices orient the shared face differently, the k-simplices have the same orientation. A 2D example is the half-edge data structure.
- The edges of adjacent triangles have opposite directions, independent from the point of view.



(a) The vertex order of a triangle orients its boundary edges.



(b) Two triangles have positive relative orientation, when they orient the shared edge differently, like in the half-edge data structure.

Dual Mesh

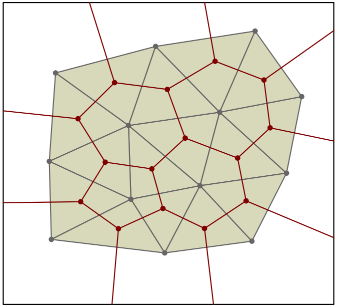


Figure: A mesh and its dual mesh (red).

- Each triangle contains a dual vertex, each edge is crossed by a dual edge, and the polygons formed by dual edges are cycles around vertices.

Dual Mesh

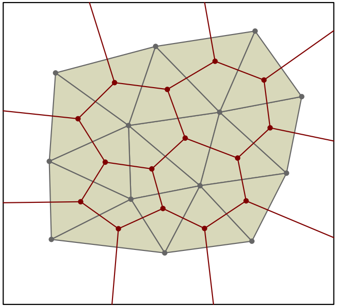


Figure: A mesh and its dual mesh (red).

- Each triangle contains a dual vertex, each edge is crossed by a dual edge, and the polygons formed by dual edges are cycles around vertices.
- When using *dual* meshes, we call the original mesh the *primal* mesh.
- Questions: Where to place the dual vertices? What happens at the boundary?

Dual Mesh

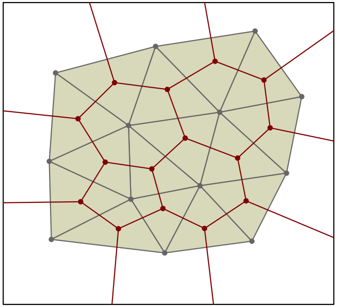


Figure: A mesh and its dual mesh (red).

- Each triangle contains a dual vertex, each edge is crossed by a dual edge, and the polygons formed by dual edges are cycles around vertices.
- When using *dual* meshes, we call the original mesh the *primal* mesh.
- Questions: Where to place the dual vertices? What happens at the boundary? Answer: It depends on the use-case.

Dual Meshes

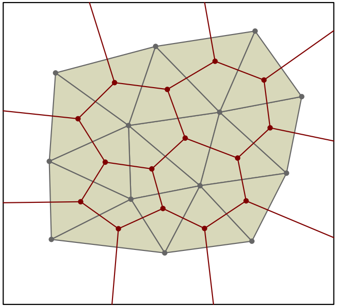


Figure: A mesh and its dual mesh (red).

- Dual vertices are often placed at the centers of triangles.
- There are different centers, e.g., the barycenter and the circumcenter. The barycenter guarantees that the vertex is inside the triangle, the circumcenter guarantees that dual edges are orthogonal to primal edges.
- Dual boundary edges are often treated by placing an additional dual vertex at the boundary edge.

Dual Meshes in higher dimensions

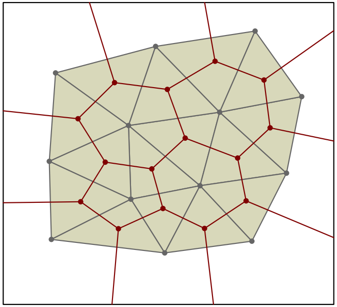


Figure: A mesh and its dual mesh (red).

- We can define dual meshes in higher dimensions, e.g., for tetrahedral meshes, as well.
- A n-dimensional simplicial complex induces a n-dimensional dual cell complex.
- Each primal k-simplex has an associated dual $(n-k)$-cell.
- The adjacency relation between k and $(k+1)$-simplices is the same as the relation between dual $(n-k)$ and $(n-k-1)$ cells.

Dual Cells

- Each n-dimensional primal simplex has a dual vertex.
- Dual edges connect two dual vertices and are dual to $(n-1)$ simplices.
- Dual faces (loops of dual edges) are dual to $(n-2)$ simplices.
- ...
- Dual cells in general are *no* simplices.

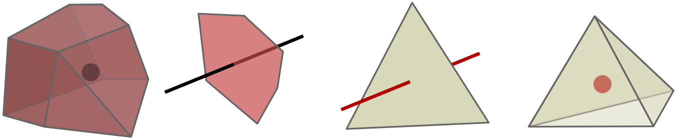


Figure: The dual cells (red) for $0, 1, 2$ and 3-simplices in $\mathbb{R}^3$.

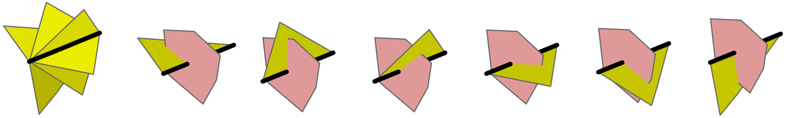


Figure: The dual 2-cell corresponding to a primal 1-simplex (edge) is a polygon with one side per triangle which is adjacent to the primal edge.

Orienting Dual Cells

- Dual Cells can be oriented using the orientation of the primal simplices
- Example in 3D:
- A dual edge points in the direction of the normal of the corresponding primal face.
- This automatically orients the edge loops of the dual faces.
- Examples:
  - Dual 3-cells (polyhedra) have an outward pointing normal because primal 0-simplices have a positive orientation (by definition).
  - Dual 2-cells (polygons) are oriented such that the normal points in the same direction as the primal 1-simplex (edge).
  - Dual 1-simplices (edges) point into the direction of the normal of the primal 2-simplex (triangle).
  - Dual vertices have the orientation of the primal n-simplices.

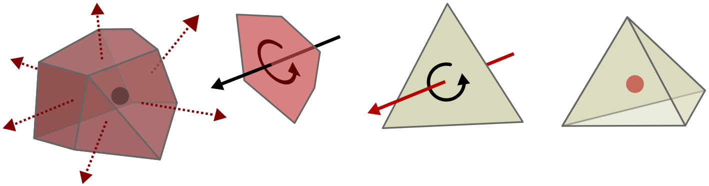


Figure: The dual cells are oriented using the primal simplices

References I

[1] E. Catmull and J. Clark. “Recursively Generated B-Spline Surfaces on Arbitrary TopologicalMeshes”. In: *Computer Aided Design* 10.6 (1978), pp. 350–355.

[2] D. Doo and M. Sabin. “Analysis of the Behaviour of Recursive Division Surfaces near Extraordinary Points”. In: *Computer Aided Design* 10.6 (1978), pp. 356–360.

[3] N. Dyn, J. Gregory, and D. Levin. “Four-Point Interpolatory Subdivision Scheme for Curve Design”. In: *Computer Aided Geometric Design* 10.6 (1987), pp. 257–268.

[4] N. Dyn, D. Levin, and J. Gregory. “Butterfly Subdivision Scheme for Surface Interpo-lation with Tension Control”. In: *ACM Transactions on Graphics*. Vol. 9. 2. ACM. 1990, pp. 160–169.

[5] L. Kobbelt. “$\sqrt{3}$ Subdivision”. In: *Computer Graphics Proceedings*. 2000.

[6] P. Lienhardt. “Topological models for boundary representation: a comparison with n-dimensional generalized maps”. In: *Computer-aided design* 23.1 (1991), pp. 59–82.

References II

[7] C. Loop. “Smooth Subdivision Surfaces Based on Triangles”. PhD thesis. Jan. 1987. URL: https://www.microsoft.com/en-us/research/publication/smooth-subdivision-surfaces-based-on-triangles/.

[8] M. Mäntylä. “Topological analysis of polygon meshes”. In: *Computer-aided design* 15.4 (1983), pp. 228–234.

[9] X. Ni and M. S. Bloor. “Performance evaluation of boundary data structures”. In: *IEEE Computer Graphics and Applications* 14.6 (1994), pp. 66–77.

[10] J. Peters and U. Reif. “The simplest subdivision scheme for smoothing polyhedra”. In: *ACM Transactions on Graphics (TOG)* 16.4 (1997), pp. 420–431.

[11] U. Reif. “A unified approach to subdivision algorithms near extraordinary vertices”. In: *Computer Aided Geometric Design* 12.2 (1995), pp. 153–174.

[12] K. Weiler. “Edge-based data structures for solid modeling in curved-surface environments”. In: *IEEE Computer graphics and applications* 5.1 (1985), pp. 21–40.

[13] T. Woo. “A combinatorial analysis of boundary data structure schemata”. In: *IEEE Computer Graphics and Applications* 5.03 (1985).