Splines (Vorlesung Splines)

Hermite Splines

„We want to interpolate with the tangents of the points."

- using pre-defined tangents
- length or direction of the tangents changes the spline segments
- automatically generate a continuous curve between two points
- we have two points P_0, P_1 and their derivatives P_0', P_1'

$$\rightarrow G_H = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} P_0 \\ P_1 \\ P_0' \\ P_1' \end{pmatrix}$$ (inserting 0/1 for t into $[t^3\ t^2\ t^1\ t^0]$)
(inserting 0/1 for t into $[3t^2\ 2t^1\ 1t^0\ 0]$)

$$\rightarrow M_H = G_H^{-1} = \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \quad (H_0^3\ H_3^3\ H_1^3\ H_2^3)$$

$$\rightarrow \begin{aligned} H_0^3(t) &= 2t^3 - 3t^2 + 1 \\ H_3^3(t) &= -2t^3 + 3t^2 \\ H_1^3(t) &= t^3 - 2t^2 + t \\ H_2^3(t) &= t^3 - t^2 \end{aligned}$$

Bezier Splines

„We want to interpolate by using control points."

- using extra control points to define the curve between the two points

$$\rightarrow M_B = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \rightarrow \begin{aligned} B_0^3(t) &= (1-t)^3 \\ B_1^3(t) &= 3t(1-t)^2 \\ B_2^3(t) &= 3t^2(1-t) \\ B_3^3(t) &= t^3 \end{aligned}$$ (Bernsteinpolynome)

Hull-Property

- cubic Bezier curve (4 control points)

$$P(t) = \sum_{i=0}^{3} B_i^3(t)\, P_i = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t) P_2 + t^3 P_3$$

$$P'(t) = 3(1-t)^2 (P_1 - P_0) + 6t(1-t)(P_2 - P_1) + 3t^2 (P_3 - P_2)$$

- transformation from Hermite to Bezier

$$M_{Bezier} = M_{Hermite}\, M_{HB} \quad \text{with } M_{HB} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{pmatrix}$$

- transformation from Bezier to Hermite

$$M_{Hermite} = M_{Bezier}\, M_{BH} \quad \text{with } M_{BH} = M_{HB}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1/3 & 0 \\ 0 & 1 & 0 & 1/3 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

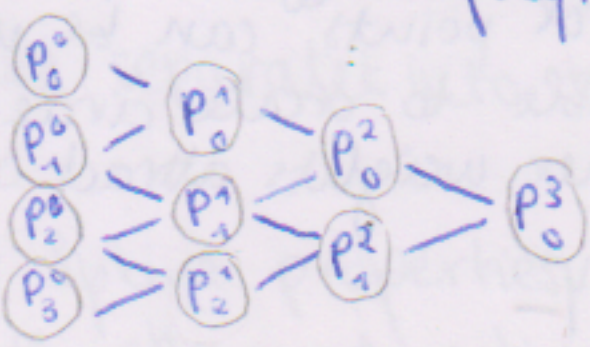
- curve calculation

$$P(t) = T\, M_H (M_{HB}\, G_B) \quad \text{and} \quad G_B = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

to draw: insert parameter t

De Casteljau

- recursive algorithm used to draw the Bezier curve by using the recursion formula for the Bernstein polynomials
- $P_i^k(t) = (1-t)\,P_i^{k-1}(t) + t \cdot P_{i+1}^{k-1}(t)$
  with k being the ~~number of control points minus 1~~ recursion depth



- when drawing connect the first points, then connect new points placed at t·length-between-points

Catmull-Rom Splines

- algorithm to smooth the joints between the spline elements
- while interpolating between point P_0 and P_2, the line connecting the two points controls the tangent at P_1
- used to achieve smooth interpolation for example for camera movement (attention: the control points need to be equally spread → different speeds)
- advantages: interpolation without overshooting, local control
- example-excercise: draw and calculate the tangent vectors for the catmull-rom spline with points $P_0, \dots P_5$

  solution: tangent for P_1: $P_2 - P_0$
  tangent for P_2: $P_3 - P_1$
  tangent for P_3: $P_4 - P_2$ etc.

B-Splines

- to control how quickly you can progress through a specific spline
- the control points are wheigted by a recursive function:

$$N_i^0(t) = \begin{cases} 1 & \text{if } t_i < t < t_{i+1} \\ 0 & \text{otherwise} \end{cases} \qquad N_i^n(t) = \frac{t - t_i}{t_{i+n} - t_i} N_i^{n-1}(t) - \frac{t + t_{i+n+1}}{t_{i+n+1} - t_{i+1}} N_{i+1}^{n-1}(t)$$

(Similiar to the Bernstein function for the Bezier curve)

- changes are very local → cue B-Splines can control parameterization
- uniform B-Splines cannot be used to draw a circle
- deBoor-Algorithm is equivalent to de Casteljau for Bezier

NURBS:

- non-uniform rational B-Splines (compared to UBS)
- the main difference to uniform B-Splines is that the control points can be wheighted → therefore also possible to create circle, cosine, sine (also need to normalize)
- higher weights attract curve to point, lower repel the curve

Continuity:

- describes how smooth a surface transits from one curve segment to another segment
- Parametric Continuity: C^0: when both segments of the curve intersect at one endpoint (tangent vectors are identical)
  C^1: when C^0-continuous and the first derivative is the same from both segments at the point
  C^2: when C^0 and C^1 continuous and the second derivative is the same from both segments at the point
- Geometric Continuity: G^0: both segments are joined in one point (same as C^0)
  (tangent vectors only have same direction, not nec same length)
  G^1: when G^0-continuous and the tangent vectors at the point are parallel (proportional)
  G^2: when G^0 and G^1 continuous and the second derivative is proportional at the joint point
- Examples: · Bezier curves in general are C^∞ continuous
  · Catmull-Rom-Splines have C^1 continuity at joints
  · attention: polynoms in general are infinitely continuous, therfore Bezier, Catmull-Rom etc. as well problems are the connection points!
  B-Splines have very high continuousity (C_{n-1} at simple knots, C_{n-k} at knot with multipl. k)

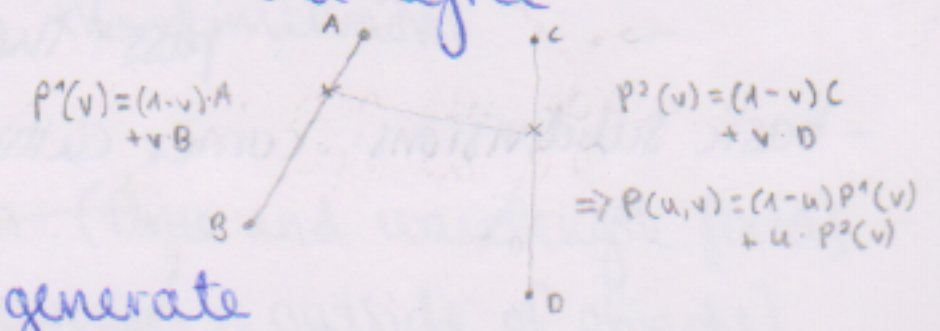
Spline Surfaces (vorlesung subdivisionsurfaces) 13.02.

Tensor Product Surfaces

"How do we generate a spline surface with a tensor product?"

- the weighting base functions are separated into two dimensions
- you basically apply e.g. Bezier Spline properties into one direction and then onto the other direction, afterwards multiply them together (tensor product)
- it's commonly the same basis function and degree

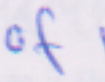
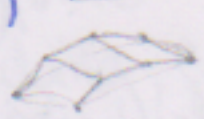
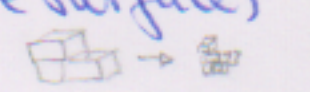
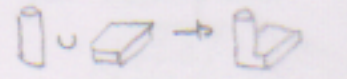
$$P(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} B_i^m(u)\, B_j^n(v)\, P_{ij}$$



- simplest case: Bilinear Patch
- often used with Bezier curves to generate continuous Bezier patches (e.g. utah teapot)
- also possible to use on volumes

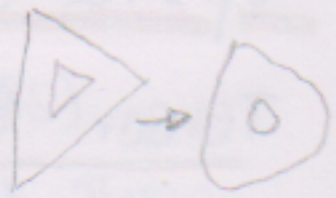
Surface Representations

"How do we represent 3D objects in a computer?"

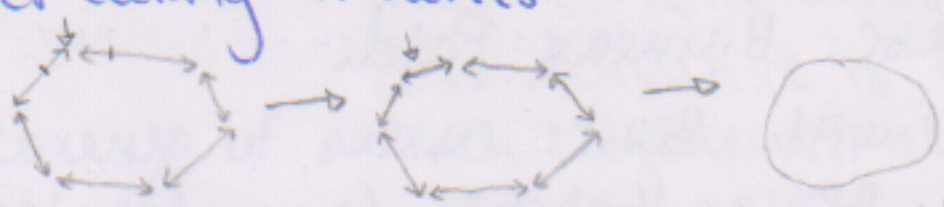
- different types:
  - raw data (eg. range image, point cloud, polysoup)
  - surfaces (e.g. mesh, subdivision, implicit)
  - solids (e.g. voxels, BSP tree)
  - neural representation (e.g. neural graphics prim.)
- range image: aquired with range scanner (depth values)
- point cloud: unstructured 3D sample points
- polygon-soup: unstructured set of polygons 
- mesh: connected set of polygons (usually triangles) 
- parametric surface: e.g. tensor product surfaces 
- subdivision surface: you got coarse mesh then subdivide (later)
- implicit surface: describe surface implicitly (if points fullfill characteristic $x^2+y^2+y^2=1$ then on sphere surface) 
- voxels: fine grid describes body (e.g. aquired from mri)
- BSP-tree:
  - CSG: Constructive Solid Geometry (boolean operations on simple shapes e.g. addition, substraction) 
- Neural Graphics: using many meshes, very new concept (too complex for exam)

Subdivision Surfaces

"How do we generate surfaces that have smooth seams?"

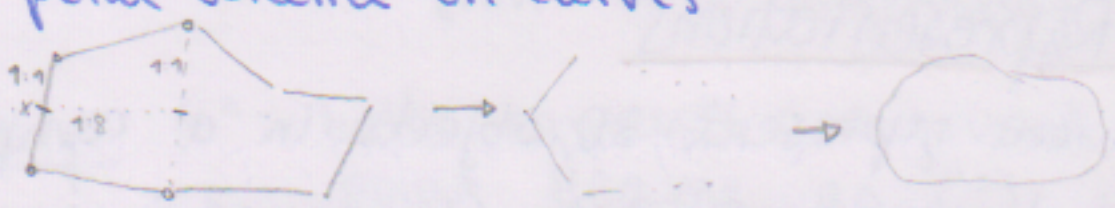


- motivation: splines like Bezier, NURBS have problems when sticking together different surfaces → hard edges or holes
- solution: subdivision surfaces which use simple meshes, where the meshes are recursively subdivided and refined
- two types: · interpolating schemes: limit surface will pass through original set of data points
  · approximating schemes: limit surface will not necessarily pass through the original set of data points
- basic subdivisions: · corner cutting on curves:



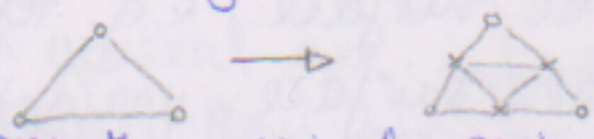
→ recursively replacing corners with 2 new points
→ approximating scheme (is rather a bit smaller)

· 4-point scheme on curves

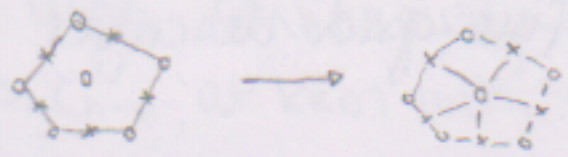


→ recursively looking at a vertex and it's two neighboring points, then dividing the vertex and the neighbor-conecting vertex in half and creating a new point at 1:8 of the lines between the middle
→ interpolating scheme (goes through control points)

- in general: · a subdivision curve is generated by repeatedly applying a subdivision operator to a given control polygon
  · "Does the subdivision process converge?"
  · "Does the subdivision process converge to a smooth curve?"
  → central questions
- examples: · triangular subdivision



→ creating more vertices from the original control net
→ every face is replaced by 4 new triangular faces
→ approximating
· loop's scheme
· quadrilateral subdivision



→ creating more vertices from the original control net
→ every face is replaced by quadrilateral faces
· Catmull Clark's Scheme

- adaptive subdivision: adaptively subdivide mesh where needed
- edges and creases: use weighting to preserve edge or crease