Short CAGD

Ida Hönigmann

I. General

Definition 1 (Combinations).

Linear Combination	$\sum_{i=1}^{n} \lambda_i v_i$	$v_i \in \mathbb{R}^d, \lambda_i \in \mathbb{R}$
Affine Combination	$\sum_{i=1}^{n} \lambda_i v_i$	above and $\sum_{i=1}^{n} \lambda_i = 1$
Convex Combination	$\sum_{i=1}^{n} \lambda_i v_i$	above and $\forall i : \lambda_i \geq 0$

Definition 2 (Affine Transformation).

$$\alpha : \mathbb{R}^n \to \mathbb{R}^m \textit{ with}$$
$$\exists l : \mathbb{R}^n \to \mathbb{R}^m \textit{ ... linear } \exists v \in \mathbb{R}^m : \alpha(x) = l(x) + v$$

Definition 3 (Convex Set, Convex Hull).

$$M \textit{...convex} \iff \forall x, y \in M \forall t \in [0,1] : tx + (1-t)y \in M$$
$$conv(M) := \{\sum_{i=0}^{n} \lambda_i v_i : v_i \in M, \lambda_i \geq 0, \sum_{i=0}^{n} \lambda_i = 1, n \in \mathbb{N}\}$$

II. Parameterized Curves

Definition 4 (Tangential Vector, Velocity).

$$tangent := \dot{c}(t) \qquad vel := ||\dot{c}(t)||$$

Definition 5 (Regular, Singular).

$$regular \iff \dot{c}(t) \neq 0 \qquad singular \iff \dot{c}(t) = 0$$

Definition 6 (Curvature, Torsion).

$$\kappa(t) := \frac{\det(\dot{c}(t), \ddot{c}(t))}{||\dot{c}(t)||^3} \qquad \textit{for } c : I \to \mathbb{R}^2$$
$$\kappa(t) := \frac{||\dot{c}(t) \times \ddot{c}(t)||}{||\dot{c}(t)||^3} \qquad \textit{for } c : I \to \mathbb{R}^3$$
$$\tau(t) := \frac{\det(\dot{c}(t), \ddot{c}(t), \dddot{c}(t))}{||\dot{c}(t) \times \ddot{c}(t)||^2} \qquad \textit{for } c : I \to \mathbb{R}^3$$

Definition 7 (Vertex). $c(t)$ *with* $\dot{\kappa}(t) = 0$

III. Bezier Curves

Algorithm 1 (of de Casteljou).

$$b_i^0(t) := b_i \quad b_i^j(t) := (1-t)b_i^{j-1}(t) + tb_{i+1}^{j-1}(t) \quad b(t) := b_0^n(t)$$

Definition 8 (Bernstein Polynomials).

$$B_i^n(t) := \binom{n}{i} t^i (1-t)^{n-i} \in \mathbb{R}[t]$$

Theorem 1 (Bernstein Representation of Bezier Curve).

$$b_i^j(t) = \sum_{l=0}^{j} B_l^j(t) b_{i+l}$$

Proof. Induction over j using the fact that $B_l^j(t) = (1-t)B_l^{j-1}(t) + tB_{l-1}^{j-1}(t)$. □

Lemma 1.

$$\sum_{i=1}^{n} B_i^n(t) = 1$$
$$t \in [0,1] \implies B_i^n(t) \geq 0$$
$$B_i^n(\alpha t) = \sum_{j=0}^{n} B_i^j(\alpha) B_j^n(t)$$
$$\{B_0^n(t), ..., B_n^n(t)\} \textit{ forms a basis of } \Pi_n$$

Lemma 2 (Endpoint Interpolating).

$$b(0) = b_0 \qquad b(1) = b_n$$

Lemma 3.

$$\dot{b}(t) = n(b_1^{n-1}(t) - b_0^{n-1}(t))$$

Proof. Calculation using the fact that $\frac{d}{dt} B_i^n(t) = n(B_{i-1}^{n-1}(t) - B_i^{n-1}(t))$. □

Theorem 2 (Invariant under Affine Transformations).

$$\alpha(b(t)) = \sum_{i=0}^{n} B_i^n(t) \alpha(b_i)$$

Theorem 3 (Bezier Curve is in Convex Hull).

$$t \in [0,1] \implies b(t) \in conv(\{b_0, ..., b_n\})$$

Theorem 4 (Bezier Curve is Symmetric).

$$b(t) \textit{ from points } b_0, ..., b_n \qquad \tilde{b}(t) \textit{ from points } b_n, ..., b_0$$
$$\implies b(t) = \tilde{b}(1-t)$$

Theorem 5 (Sub-division property).

$$\tilde{b}(t) := b(\alpha t) \qquad \hat{b}(t) := b((1-\alpha)t + \alpha)$$

Then

$$\tilde{b} \textit{ from points } b_0^0(\alpha), ..., b_0^n(\alpha)$$
$$\hat{b} \textit{ from points } b_0^n(\alpha), b_1^{n-1}(\alpha), ..., b_n^0(\alpha)$$

and "gluing" them together results in $b(t)$.

Theorem 6. *Every polynomial curve is a Bezier curve.*

Theorem 7 (Corner-cutting).

$$P_0 := (b_0, ..., b_n) \quad P_n := \textit{ sub-divide } n \textit{ times with } n = 1/2$$
$$\implies P_n \to b(t)$$

IV. B-SPLINE AND NURBS

Definition 9 (B-Spline). *Composition of Bezier Curves.*

Definition 10 (B-Spline Base Functions).

$$\alpha_i^r(t) := \begin{cases} \frac{t-t_i}{t_{i+r}-t_i} & t_{i+r}-t_i \neq 0, \\ 0, & \textit{else} \end{cases}$$

$$N_i^0(t) := \begin{cases} 1, & t \in [t_i, t_{i+1}) \\ 0, & \textit{else} \end{cases}$$

$$N_i^r(t) := \alpha_i^r(t) N_i^{r-1}(t) + (1-\alpha_i^r(t)) N_{i+1}^{r-1}(t)$$

Lemma 4.

$$\sum_{i=0}^{m} N_i^n(t) = 1 \qquad N_i^n(t) \geq 0$$

Theorem 8 (B-Spline Representation). *m ... number control points, n ... degree, $T = (t_0, ..., t_{m+n+1})$ ... knot vector with $t_i \leq t_{i+1}$, $t_i < t_{i+n+1}$*

$$s(t) = \sum_{i=0}^{m} N_i^n(t) c_i$$

Definition 11 (NURBS). *Construction by:*

1) *homogenize control points*
2) *scale by weight*
3) *construct B-Spline*
4) *project back onto $1 \times \mathbb{R}^d$*
5) *dehomogenize*

Lemma 5. *B-Spline curve with $m = n$ is a Bezier curve. NURBS with $w_i = w$ is a B-Spline.*

Remark 1. *w_i bigger $\implies$ curve is attracted towards c_i.*

Remark 2. *NURBS can construct conic sections (such as circles).*

Lemma 6. *B-Spline and NURBS are both affine invariant.*

V. FREE FORM SURFACES

Definition 12 (Free Form Surface). $b_{00}, b_{01}, ..., b_{mn} \in \mathbb{R}^d$, $g(s) = \sum_{i=0}^{m} D_i(s) p_i$, $h(t) = \sum_{j=0}^{n} E_j(t) q_j$

$$f(s,t) := \sum_{i=0}^{m} \sum_{j=0}^{n} D_i(s) E_j(t) b_{ij}$$

Remark 3. *Many same theorems as in Bezier, B-Spline and NURBS hold.*

VI. SUBDIVISION OF CURVES

Algorithm 2 (Chaikin). *copy once, average twice*

- *$\rightarrow$ quadratic B-Spline*
- *approximating*
- *affine invariant*
- *linear precision*

Algorithm 3 (Lane-Riesenfeld). *copy once, average n times*

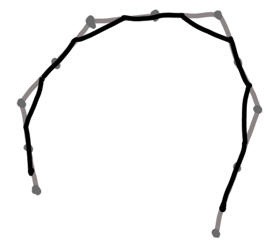

Fig. 1. Chaikin

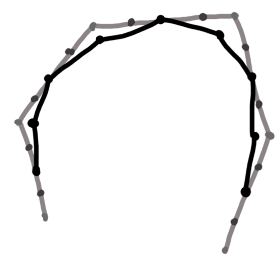

Fig. 2. Lane-Riesenfeld

- *$\rightarrow$ B-Spline of degree n*
- *approximating*
- *affine invariant*
- *linear precision*

Algorithm 4 (Four Point Scheme). $p_i^0 := p_i$
$p_i^l := -\frac{1}{16} p_{i-1}^{l-1} + \frac{9}{16} p_i^{l-1} + \frac{9}{16} p_{i+1}^{l-1} - \frac{1}{16} p_{i+2}^{l-1}$

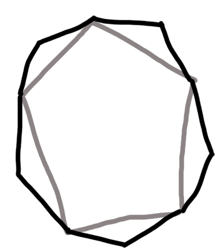

Fig. 3. Four Point Scheme

- *$\rightarrow C^1$ curve*
- *interpolating*
- *affine invariant*
- *linear precision*

VII. SUBDIVISION OF MESHES

Algorithm 5 (Loop). *triangle mesh*

- *$\rightarrow C^2$ in generic ($n = 6$) case*

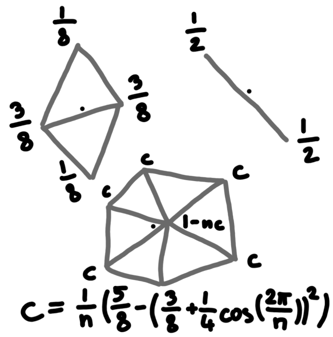

Fig. 4. Loop

- $\rightarrow C^1$ *otherwise*
- *approximating*
- *face splitting*

Algorithm 6 (Modified Butterfly)**.** *triangle mesh*

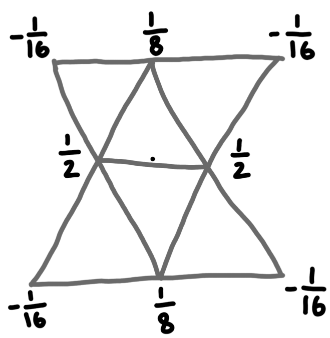

Fig. 5. Modified Butterfly

- $\rightarrow C^1$ *surface*
- *interpolating*
- *face splitting*

Algorithm 7 (Catmull Clark)**.** *square mesh*

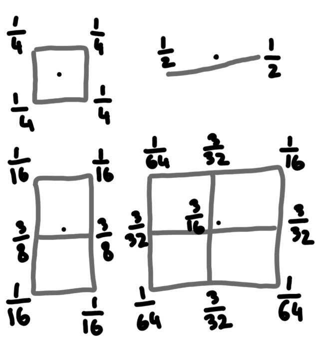

Fig. 6. Catmull Clark

- $\rightarrow C^2$ *in generic* ($n = 4$) *case*
- *approximating*
- *face splitting*

Algorithm 8 (Doo Sabin)**.** *any mesh*

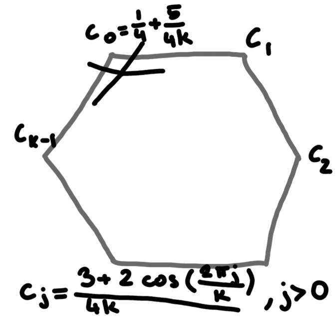

Fig. 7. Doo Sabin

- $\rightarrow C^1$ *surface*
- *approximating*
- *vertex splitting*

Algorithm 9 (Kobbelt)**.** *square mesh*

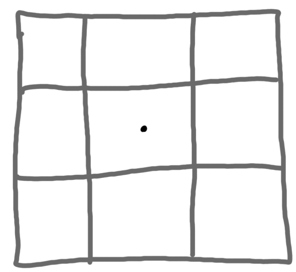

Fig. 8. Kobbelt

four point scheme horizontal or vertical, then four point scheme on results

- $\rightarrow C^1$ *surface*
- *interpolating*
- *face splitting*