KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT COMPUTERWETENSCHAPPEN
AFDELING NUMERIEKE ANALYSE EN
TOEGEPASTE WISKUNDE
Celestijnenlaan 200A – B-3001 Heverlee

POWELL–SABIN SPLINES FOR COMPUTER AIDED GEOMETRIC DESIGN

Promotor:
Prof. Dr. ir. P. Dierckx

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de toegepaste wetenschappen

door

Joris WINDMOLDERS

February 2003

KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT COMPUTERWETENSCHAPPEN
AFDELING NUMERIEKE ANALYSE EN
TOEGEPASTE WISKUNDE
Celestijnenlaan 200A – B-3001 Heverlee

POWELL–SABIN SPLINES FOR COMPUTER AIDED GEOMETRIC DESIGN

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de toegepaste wetenschappen

door

Joris WINDMOLDERS

U.D.C. 681.3*J6

February 2003

D/2002/7515/59

ISBN 90-5682-388-4

Rev. 1

Powell–Sabin splines for Computer Aided Geometric Design

Joris Windmolders
Departement Computerwetenschappen, K.U.Leuven
Celestijnenlaan 200A, B-3001 Heverlee, België

Abstract

We study Powell–Sabin (PS–)splines in their normalized B–spline representation for Computer Aided Geometric Design (CAGD). These piecewise quadratic polynomials with global C^1–continuity have certain advantages compared to the widely used tensor product B–splines and NURBS. We investigate the extension of PS–splines to Non Uniform Rational Powell–Sabin splines (NURPS). These have more flexibility for interactive design and allow to represent quadrics exactly. We also study the particular case of PS–splines on uniform triangulations, give a subdivision scheme for these so–called UPS–splines and use it to derive a new wavelet transform on a triangular grid, as well as an algorithm for the graphical display of UPS–spline surfaces. We use UPS–splines in a generally applicable algorithm that solves the polygonal hole problem, based on interpolation and subdivision techniques. We discuss a software prototype for working with UPS–splines, called PS–surf.

"First you have to learn all the rules,
then you have to forget them."

L. von Beethoven

Preface

This thesis studies Powell–Sabin (PS–)splines in their normalized B–spline representation in Computer Aided Geometric Design (CAGD). These have certain advantages compared to the widely used tensor product B–splines and NURBS. For example, they are defined over arbitrary triangulations of polygonal domains, while tensor product B–splines are restricted to rectangular geometries.

I would like to take the opportunity to thank my promotor, Prof. Paul Dierckx, who initiated me in the theory of splines, and who encouraged me to do this research. He had always faith in a good result, and this motivated me to continue with my work in periods where I seemed to be following a dead track. During the many conversations we had together, he gave me a lot of valuable scientific (and other) advice. He also helped me to greatly improve the presentation of this thesis. I also want to thank my assessors, who carefully read an early manuscript of this text, and made a lot of suggestions for its improvement too, as well as the other members of the jury, Adhemar Bultheel and Paul Sablonnière. Furthermore, I want to thank Anuschka Kerstens from Metris for providing me with the datasets of some CAD models, and the colleagues who worked in the same offices as I did, in particular Kurt Lust, for creating the right atmosphere for doing research, and for always being helpful. I also thank Evelyne Vanraes, who recently became interested in Powell–Sabin splines, and with whom I had a fruitful collaboration during the past months. I wish her good luck in her further research on PS–splines. Finally I would also like to thank my wife Veronique and my parents, for their support during the past years, as well as the IWT (Instituut voor de aanmoediging van Innovatie door Wetenschap en Technologie in Vlaanderen), the institute which provided me with a research grant.

Joris Windmolders
December 2002

Nederlandse samenvatting

Powell–Sabin Splines voor Computer Gesteund Geometrisch Ontwerpen

Inhoudsopgave

1 Inleiding

De mogelijkheid om complexe vormen voor te stellen op een manier die kan gebruikt worden door computers, kwam voor het eerst in de belangstelling in de late jaren 1950, bij de opkomst van machines voor de productie van vormen uit hout of staal. Essentieel hierbij is het gebruik van een gepaste wiskundige voorstelling voor krommen en oppervlakken. Gewoonlijk wordt een oppervlak voorgesteld als

$$\boldsymbol{s}(u) = \sum_i \boldsymbol{c}_i \phi_i(u), \quad u = (x, y) \in \Omega, \quad \boldsymbol{c}_i \in \mathbb{R}^3, \tag{1}$$

waar Ω het parameter domein is, en de basisfuncties $\phi_i(u)$ (stuksgewijze) veeltermen zijn van niet te hoge graad. Er worden dan een aantal voorwaarden opgelegd aan deze basis:

Convexe eenheidspartitie.

$$\begin{array}{rcl} \phi_i(u) & \geq & 0 \\ \sum_i \phi_i(u) & \equiv & 1. \end{array} \tag{2}$$

Lokaliteit.

$$\phi_i(u) = 0, \quad u \notin M_i \subset \Omega, \tag{3}$$

met M_i een kleine gesloten deelverzameling van Ω.

Parametrische continuïteit.

$$\phi_i(u) \in C^r[\Omega]. \tag{4}$$

Lineaire precisie. Er bestaan $X_i, Y_i \in \mathbb{R}$ waarvoor geldt dat

$$\begin{array}{rcl} x & = & \sum_i X_i\phi_i(u) \\ y & = & \sum_i Y_i\phi_i(u). \end{array} \tag{5}$$

Ze impliceren een aantal eigenschappen die dergelijke voorstellingen interessant maken in Computer Gesteund Geometrisch Ontwerpen (CAGD):

Controlepunten. Er kunnen controlepunten geassocieerd worden met de coëfficiënten $\boldsymbol{c}_i$. Door deze punten te verbinden bekomt men een controlenet, dat de vorm van het oppervlak weergeeft.

Convex omhullende eigenschap. Het oppervlak ligt binnen de convex omhullende van de controlepunten.

Lokale controle. Door een controlepunt te verplaatsen, wordt het oppervlak slechts lokaal gewijzigd in de buurt van dat punt.

Continuïteit. Ongeacht de keuze van de controlepunten heeft $\boldsymbol{s}(u)$ globale continuïteit van orde r.

Affien invariant. Om een affiene transformatie door te voeren op $\boldsymbol{s}(u)$, volstaat het deze toe te passen op de controlepunten.

De meest gebruikte voorstelling in commerciële paketten is de tensor product B–spline voorstelling voor oppervlakken en de uitbreiding ervan naar NURBS. Ze heeft echter als nadeel dat ze beperkt is tot rechthoekige domeinen Ω, terwijl in de praktijk vaak oppervlaktesegmenten met bijvoorbeeld drie zijden gewenst zijn. Verder wordt het domein door de knooppunten opgedeeld in een regelmatig rooster, hetgeen een adaptieve verfijning verhindert.

Als alternatief kan men daarom de Bernstein–Bézier voorstelling van oppervlakken over driehoeken overwegen. Ze is theoretisch zeer sterk onderbouwd [30], maar heeft een aantal nadelen vanuit praktisch oogpunt: ten eerste is de basis niet lokaal (men spreekt van pseudo–lokale controle), en ten tweede resulteren de continuïteitsvoorwaarden tussen aansluitende oppervlaktesegmenten in complexe relaties tussen de controlepunten. Dit bemoeilijkt het manipuleren van dergelijke oppervlakken op een intuïtieve, voorspelbare manier. Daarom onderzoeken we in dit proefschrift het gebruik van Powell–Sabin (PS–)splines over driehoeksverdelingen in hun genormalizeerde B–spline basis in CAGD. De basis voldoet aan alle gestelde voorwaarden met C^1–continuïteit [19], en laat toe om oppervlakken voor te stellen over een willekeurige veelhoek.

Het proefschrift is als volgt opgebouwd. In Hoofdstuk 2 geven we een overzicht van de fundamentele theorie van Bernstein–Bézier veeltermen over driehoeken en van PS–splines. Hoofdstuk 3 behandelt de uitbreiding van PS–splines naar Niet Uniforme Rationale Powell–Sabin Spline (NURPS) oppervlakken. We onderzoeken hoe de gewichten die daarbij aan elk controlepunt worden toegekend, extra mogelijkheden geven voor vrije vorm ontwerpen, en bestuderen de voorstelling van enkele kwadrieken als NURPS oppervlak. We letten daarbij op de numerieke stabiliteit, geïnspireerd door het rationaal de Casteljau algoritme van Farin. In Hoofdstuk 4 bespreken we een specifiek geval van PS–splines, met name deze over uniforme driehoeksverdelingen (UPS–splines). We geven efficiënte algoritmen voor het werken met UPS–splines, leiden subdivisieregels af, en ontwerpen als toepassing hiervan een UPS–spline wavelet transformatie, die gebruikt wordt voor ruisverwijdering uit een UPS–spline oppervlak in een drempel algoritme. Een tweede toepassing van subdivisie wordt gegeven in een algoritme voor de visualisatie van (rationale) UPS–spline oppervlakken. Hoofdstuk 5 behandelt een vaak voorkomend probleem in CAGD, het berekenen van een oppervlak dat een opening, begrensd door een aantal gegeven oppervlakken, opvult. We stellen een algemeen toepasbaar algoritme voor dat een UPS–spline oppervlak genereert dat de gegeven rand in een aantal punten interpoleert, en ze in tussenliggende punten zo goed mogelijk benadert. We steunen hierbij op subdivisie. Hoofdstuk 6 beschrijft een software prototype voor het werken met PS–splines, PS–surf, dat samen met het onderzoek ontwikkeld werd. We tonen aan hoe de visualisatie van UPS–spline oppervlakken efficiënt kan gebeuren met OpenGL, en beschrijven een aantal realistische CAD modellen met UPS–spline oppervlakken. Tenslotte geven we onze conclusies en suggesties voor toekomstig onderzoek in Hoofdstuk 7.

2 Powell–Sabin splines

2.1 Bernstein–Bézier veeltermen en oppervlakken over driehoeken

Zij $\mathcal{T}$ een niet–ontaarde driehoek met hoekpunten $V_i(x_i, y_i), i = 1, 2, 3$. Elk punt $u(x, y) \in \mathbb{R}^2$ kan dan uniek voorgesteld worden door zijn barycentrische coördinaten $\tau = (\tau_1, \tau_2, \tau_3)$ ten opzichte van $\mathcal{T}$. Deze zijn de oplossing van het stelsel

$$\begin{aligned} x_1\tau_1 + x_2\tau_2 + x_3\tau_3 &= x \\ y_1\tau_1 + y_2\tau_2 + y_3\tau_3 &= y \end{aligned} \tag{6}$$

$$\tau_1 + \tau_2 + \tau_3 \quad = \quad 1.$$

De Bernstein veeltermen van graad m over een driehoek zijn gedefinieerd als

$$B^m_\lambda = \frac{m!}{\lambda_1!\lambda_2!\lambda_3!}\tau_1^{\lambda_1}\tau_2^{\lambda_2}\tau_3^{\lambda_3}; \quad |\lambda| = m, \tag{7}$$

waarbij $\lambda = (\lambda_1, \lambda_2, \lambda_3)$ een multi–index is, en $|\lambda| := \lambda_1 + \lambda_2 + \lambda_3$. Deze veeltermen hebben een aantal interessante eigenschappen. Zo vormen ze binnen $\mathcal{T}$ een convexe eenheidspartitie: ze zijn er niet–negatief en sommeren tot één, hetgeen onmisbaar zal blijken te zijn voor CAGD toepassingen. Voorts kunnen ze eenvoudig geïntegreerd worden. De ruimte van veeltermen van graad $\leq m$ in twee veranderlijken wordt genoteerd als Π_m. Elke veelterm $p(u), u \in \mathbb{R}^2$, in Π_m, heeft een enige voorstelling als een Bernstein–Bézier veelterm

$$p(u) := b(\tau) = \sum_{|\lambda|=m} b_\lambda B^m_\lambda(\tau). \tag{8}$$

De coëfficiënten b_λ noemt men de Bézier ordinaten. Het is de gewoonte een Bernstein–Bézier veelterm schematisch voor te stellen door b_λ te associëren met het punt (λ/m). Bernstein–Bézier veeltermen kunnen op efficiënte wijze geëvalueerd en afgeleid worden via het de Casteljau algoritme. Dit algoritme heeft voorts nog andere toepassingen, zoals het afleiden van nodige en voldoende continuïteitsvoorwaarden voor Bernstein–Bézier veeltermen over naburige driehoeken, en subdivisie. Het blijkt echter dat dergelijke continuïteitsvoorwaarden niet eenvoudig te implementeren zijn. De grafe van een Bernstein–Bézier veelterm is een Bernstein–Bézier oppervlak

$$\boldsymbol{b}(\tau) := \{(x, y, z) \mid (x, y) = (\tau_1, \tau_2, \tau_3), \quad z = b(\tau), \quad \tau \in \mathcal{T}\}. \tag{9}$$

De punten $\boldsymbol{b}_\lambda(\frac{\lambda}{m}, b_\lambda)$ zijn de controlepunten van $\boldsymbol{b}(\tau)$, en hun lineaire interpolant is het controlenet. Dit is richtgevend voor de vorm van het oppervlak. Een gevolg van de eenheidspartitie–eigenschap van Bernstein veeltermen is dat het Bernstein–Bézier oppervlak binnen de convex omhullende van haar controlepunten ligt. Hetzelfde geldt voor de uitbreiding naar parametrische Bernstein–Bézier oppervlakken

$$\boldsymbol{b}(\tau) = \sum_{|\lambda|=m} \boldsymbol{b}_\lambda B_\lambda(\tau), \tag{10}$$

met controlepunten $\boldsymbol{b}_\lambda = (b^x_\lambda, b^y_\lambda, b^z_\lambda)$. Voor het vinden van een punt op het oppervlak, een raakvector of een normaalvector kan men beroep doen op het de Casteljau algoritme. Tot slot merken we op dat Bernstein–Bézier oppervlakken affien invariant zijn. Zo volstaat het om het controlenet te transleren als men eigenlijk het oppervlak wil verschuiven. Ook zijn Bernstein–Bézier oppervlakken invariant onder affiene parameter transformaties.

2.2 Powell–Sabin spline functies en oppervlakken

Met Δ beduiden we een conforme driehoeksverdeling van $\Omega \subset \mathbb{R}^2$ met hoekpunten V_i en driehoeken ρ_j, $j = 1, \ldots, t$, ook genoteerd $\rho_{i,j,k} := \rho(V_i, V_j, V_k)$. Een Powell–Sabin verfijning van Δ, genoteerd als Δ^*, wordt als volgt opgesteld:

1. Kies een punt Z_j in elke driehoek ρ_j, zodat als twee driehoeken ρ_j en ρ_l een gemeenschappelijke zijde hebben, de lijn die de punten Z_j en Z_l verbindt, deze zijde snijdt in een punt $R_{j,l}$ tussen de hoekpunten.
2. Verbind elk punt Z_j met de hoekpunten van ρ_j.
3. Voor elke zijde van ρ_j
   (a) die tot de rand $\partial\Omega$ behoort, verbind Z_j met een willekeurig punt tussen de hoekpunten.
   (b) die gemeenschappelijk is met een driehoek ρ_l, verbind Z_j met $R_{j,l}$.

De ruimte van stuksgewijs kwadratische veeltermen met C^1–continuïteit, ook de ruimte van Powell–Sabin (PS–) splines genoemd, wordt gedefinieerd als

$$S_2^1(\Delta^*) = \left\{ s(u), u \in \Omega \quad | \quad s \in C^1(\Omega), s|_{\rho_j^*} \in \Pi_2, j = 1, \ldots, t^* \right\}, \qquad (11)$$

waarbij ρ_j^* de driehoekjes zijn van de PS–verfijning. Powell en Sabin [57] hebben aangetoond dat er een unieke PS–spline $s(u) \in S_2^1(\Delta^*)$ bestaat zodat

$$s(V_i) = f_i, \quad \frac{\partial s}{\partial x}(V_i) = f_{x,i}, \quad \frac{\partial s}{\partial y} = f_{y,i}, \quad i = 1, \ldots, n. \qquad (12)$$

Dit kan gebruikt worden om een basis voor $S_2^1(\Delta^*)$ op te stellen. Het idee is om drie lineair onafhankelijke drie–koppels $(\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j}), j = 1, 2, 3$ te zoeken voor elk hoekpunt. Een B–spline basisfunctie is dan de enige oplossing van het interpolatieprobleem (12) met alle $(f_l, f_{x,l}, f_{y,l}) = (0, 0, 0)$ behalve voor $(f_i, f_{x,i}, f_{y,i}) = (\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j}) \neq (0, 0, 0)$, en wordt genoteerd als $B_i^j(u), u \in \mathbb{R}^2$. Ze is slechts lokaal verschillend van nul: $B_i^j(u) \equiv 0, u \notin M_i$ waarbij de molecule M_i enkel de driehoeken bevat die V_i als hoekpunten hebben. Elke $s(u) \in S_2^1(\Delta^*)$ heeft dan een enige voorstelling

$$s(u) = \sum_{i=1}^{n} \sum_{j=1}^{3} c_{i,j} B_i^j(u), \quad u \in \Omega. \qquad (13)$$

In deze thesis werken we met een lokale genormalizeerde B–spline basis voor PS–splines [19]. Dit will zeggen dat de basisfuncties een convexe eenheidspartitie vormen, hetgeen van belang is in CAGD toepassingen. Dierckx heeft aangetoond dat er een geometrische interpretatie is voor het opstellen van een dergelijke basis: zoek, voor elk hoekpunt, een driehoek $t_i(Q_{i,1}, Q_{i,2}, Q_{i,3})$ die een gegeven stel punten (de zogenaamde PS–punten) bevat. We noemen dit de PS–driehoek bij het hoekpunt V_i. Dit komt neer op het oplossen van een kwadratisch programmeringsprobleem voor elk hoekpunt. Er bestaat een één–één relatie tussen de drie–koppels $(\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j})$ en de coördinaten van de PS–driehoekspunten $Q_{i,j}(X_{i,j}, Y_{i,j})$. Wanneer men als PS–driehoek de kleinste driehoek kiest die de gegeven punten bevat, spreekt men van de optimale voorstelling; er zijn echter nog andere mogelijkheden denkbaar.

De Bernstein–Bézier voorstelling van een PS–spline kan op een efficiënte en numeriek stabiele manier berekend worden. Dit geeft aanleiding tot algoritmen voor het evalueren, afleiden en integreren van PS–splines. De grafe van een PS–spline noemen we een PS–spline oppervlak

$$\boldsymbol{s}(u) \left\{ \begin{array}{lcl} x & = & x \\ y & = & y \\ z & = & s(u) \end{array} \right. , \quad u = (x, y) \in \Omega, \tag{14}$$

met als controlepunten $\boldsymbol{c}_{i,j}(X_{i,j}, Y_{i,j}, c_{i,j})$. De controledriehoekjes worden gedefinieerd als $T_i(\boldsymbol{c}_{i,1}, \boldsymbol{c}_{i,2}, \boldsymbol{c}_{i,3})$ en zijn rakend aan het oppervlak in $\boldsymbol{s}(V_i)$. De uitbreiding naar parametrische PS–spline oppervlakken

$$\boldsymbol{s}(u) = \sum_{i=1}^{n} \sum_{j=1}^{3} \boldsymbol{c}_{i,j} B_i^j(u), \quad u \in \Omega, \tag{15}$$

met controlepunten $\boldsymbol{c}_{i,j} = \left(c_{i,j}^x, c_{i,j}^y, c_{i,j}^z\right)$ ligt nu voor de hand, net als de berekening van een punt op het oppervlak, een raakvector en een normaalvector door omzetting naar de voorstelling als Bernstein–Bézier oppervlak. Het oppervlak ligt binnen de convex omhullende van de controlepunten, en door de lokaliteit van de basisfuncties heeft een gebruiker lokale controle: het wijzigen van een controlepunt zal slechts lokaal invloed hebben op het oppervlak. In het proefschrift breiden we het concept van PS–splines eerst uit naar rationale PS–spline oppervlakken (NURPS), en onderzoeken later het specifieke geval van PS–splines over uniforme driehoeksverdelingen (UPS–splines), die een aantal specifieke voordelen hebben.

3 NURPS oppervlakken

3.1 Rationale Bernstein–Bézier oppervlakken

Net zoals B–spline curven en tensor product B–spline oppervlakken uitgebreid kunnen worden naar NURBS curven en oppervlakken, worden Bernstein–Bézier oppervlakken veralgemeend naar hun rationale tegenhangers [30],

$$\boldsymbol{b}(\tau) = \frac{\sum_{|\lambda|=m} w_\lambda \boldsymbol{b}_\lambda B_\lambda^m(\tau)}{\sum_{|\lambda|=m} w_\lambda B_\lambda^m(\tau)}, \quad \tau \in \mathcal{T}. \tag{16}$$

Met elk controlepunt $\boldsymbol{b}_\lambda$ wordt een gewicht w_λ geassocieerd. Als alle gewichten gelijk zijn aan één, bekomen we een niet–rational Bernstein–Bézier oppervlak. We veronderstellen dat $w_\lambda > 0$ zodat de noemer in (16) overal in $\mathcal{T}$ gedefinieerd is. Het oppervlak kan ook geschreven worden in termen van een basis die een convexe eenheidspartitie vormt. De evaluatie van een rationaal Bernstein–Bézier oppervlak kan gebeuren door het de Casteljau algoritme component per component op teller en noemer toe te passen, en nadien te delen. Dit kan echter leiden tot numerieke problemen als de gewichten erg variëren in grootte [30]. Het rationaal de Casteljau algoritme zal daarom de berekeningen herorganiseren, op een manier waarop er enkel convexe combinaties van de controlepunten genomen worden. Dit levert een duurdere, maar numeriek stabiele methode op. Als toepassing kunnen we de componentsgewijze partiële afgeleiden van $\boldsymbol{b}(\tau)$ berekenen.

De grafische weergave van B–spline curven en oppervlakken gebeurt veelal via subdivisie van het controlenet. In plaats van een groot aantal punten op het oppervlak te berekenen, wordt het controlenet afgebeeld, na enkele verfijningsstappen. Het subdivisieschema voor rationale Bernstein–Bézier oppervlakken, gebaseerd op het rationaal de Casteljau algoritme, is hier echter niet meteen voor geschikt. De reden is dat elke domeindriehoek opgesplitst wordt in drie deeldriehoeken door opsplitsing van de hoeken, terwijl de zijden onveranderd blijven. In deze thesis geven we daarom een alternatief rationaal subdivisieschema dat elke driehoek opsplitst in vier deeldriehoeken door opsplitsing van de zijden, terwijl de hoeken onveranderd blijven. Dit is niet alleen beter geschikt voor de visualisatie, maar is ook interessanter vanuit numeriek oogpunt.

De extra vrijheidsgraden die de gewichten opleveren, geven een gebruiker meer flexibiliteit bij het ontwerpen van vormen. Voor rationale Bézier curven heeft Farin vormparameters gedefinieerd die toelaten op een interactieve en voorspelbare manier, zonder getallen te moeten invoeren, de vorm van de curve te veranderen aan de hand van de gewichten. Voor rationale Bernstein–Bézier oppervlakken kan dit concept echter niet overgenomen worden [59]. Tot op vandaag is er geen geometrische methode gekend voor het manipuleren van de gewichten. Er is daarentegen heel wat literatuur verschenen rond het voorstellen van kwadrieken als rationale Bernstein–Bézier oppervlakken [4, 5, 22, 23, 31, 65]. De algemene voorwaarden waaronder dit mogelijk is, kunnen geformuleerd worden als volgt: *een driehoekig rationaal kwadratisch Bernstein–Bézier oppervlak stelt een kwadriek voor als en slechts als de drie randcurven elkaar snijden in een gemeenschappelijk punt met coplanaire raakvectoren.*

3.2 Niet uniforme rationale PS–spline oppervlakken

Op dezelfde manier als voorheen, worden PS–splines nu veralgemeend naar rationale PS–spline oppervlakken (NURPS),

$$\boldsymbol{s}(u) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{3} \boldsymbol{c}_{i,j} w_{i,j} B_i^j(u)}{\sum_{i=1}^{n}\sum_{j=1}^{3} w_{i,j} B_i^j(u)}, \qquad u = (u_1, u_2) \in \Omega, \tag{17}$$

waar $\boldsymbol{c}_{i,j} = (c_{i,j}^x, c_{i,j}^y, c_{i,j}^z)$ de controlepunten zijn en $w_{i,j} > 0$ de gewichten. Ook deze kunnen geschreven worden in termen van een basis die een convexe eenheidspartitie vormt op Ω. Bovendien zijn de basisfuncties dan lokaal verschillend van nul, hetgeen samen affiene invariantie, lokale controle en de convex omhullende eigenschap impliceert. Ook kunnen er controledriehoeken T_i gevonden worden die rakend zijn aan het oppervlak in $\boldsymbol{s}(V_i)$. Geïnspireerd door het rationaal de Casteljau algoritme, geven we een rationaal algoritme voor het afleiden van de rationale Bernstein–Bézier voorstelling van een NURPS oppervlak op een numeriek stabiele manier. Dit opent de deur naar het berekenen van een punt op het oppervlak, alsook een raakvlak en een normaalvector, en naar de grafische weergave steundend op ons alternatief rationaal subdivisieschema. Ook geven we rationale algoritmen voor het berekenen van de NURPS voorstelling van een gegeven rationaal Bernstein–Bézier oppervlak. Er zijn meerdere mogelijkheden om dit probleem op te lossen, afhankelijk van de keuze van de PS–driehoeken. We geven enerzijds een methode waarbij de NURPS controlepunten en gewichten samenvallen met die van de Bernstein–Bézier voorstelling, en anderzijds een methode waarbij de optimale NURPS voorstelling over een uniforme driehoek berekend wordt. De eerste methode is geschikt wanneer de domeindriehoek gegeven is, bijvoorbeeld voor functionele oppervlakken. De tweede methode is aangewezen bij parametrische oppervlakken. Terwijl de meeste commerciële paketten voor computer gesteund ontwerpen de NURBS voorstelling ondersteunen, is dit niet het geval voor het manipuleren van de gewichten. Vaak zijn ze ontoegankelijk voor de gebruiker. We onderzoeken in dit werk hoe de NURPS gewichten extra ontwerpmogelijkheden kunnen aanbieden:

1. Vooreerst definiëren we vormparameters door de gewichten een geometrische interpretatie te geven. Het veranderen van een gewicht komt neer op het verplaatsen van het raakpunt in de overeenkomstige controledriehoek. Een gebruiker kan dit raakpunt dus naar een willekeurige positie verslepen, terwijl de gewichten op een transparante manier aangepast worden.

2. Ten tweede kunnen speciale effecten gemodelleerd worden door te werken met de gewichten. We tonen aan dat als de gewichten bij de controlepunten van een hoekpunt V_i zeer groot worden, er lokaal vlakke segmenten ontstaan. Hiertoe steunen we op ons algoritme voor het berekenen van de rationale Bernstein–Bézier voorstelling, en de convex omhullende eigenschap.

3. Hierbij aansluitend geven we een aantal speciale effecten die niet rechtstreeks te maken hebben met de gewichten in de NURPS voorstelling: het modelleren van een piek, een rechte rand, een hoekpunt en een kromme rand met gedegenereerde controledriehoeken.

4. Tenslotte onderzoeken we het exact voorstellen van kwadrieken met behulp van NURPS. We geven een constructieve manier om gesloten uitdrukkingen te vinden voor oppervlaktesegmenten op een cilinder, kegel, en een bol. We doen dit door eerst een rationale Bernstein–Bézier voorstelling af te leiden en deze om te zetten naar een NURPS oppervlak.

   (a) Een cilinder kan opgesplitst worden in acht isometrische stukken, die elk als kwadratisch Bernstein–Bézier oppervlak kunnen voorgesteld worden. Dit leidt tot een samengestelde NURPS representatie.

   (b) Een kegel kan op analoge wijze gevonden worden als een samengesteld NURPS oppervlak met zes segmenten.

   (c) Een bol kan niet volledig voorgesteld worden door driehoekige rationale kwadratische Bernstein–Bézier segmenten. We geven daarom een onvolledige voorstelling van de bol als NURPS oppervlak, met openingen rond de meridiaan. De gebruiker heeft enigzins controle over de grootte van deze openingen door keuze van het aantal NURPS segmenten.

3.3 Besluit

We hebben de genormalizeerde B–spline voorstelling voor PS–spline oppervlakken veralgemeend naar een rationale vorm, NURPS oppervlakken, die geschikt is voor CAGD toepassingen. We hebben aangetoond hoe de onderliggende rationale Bernstein–Bézier voorstelling kan berekend worden via een rationaal, numeriek stabiel schema, geïnspireerd op het rationaal de Casteljau algoritme. We hebben een alternatief subdivisieschema gegeven voor rationale Bernstein–Bézier oppervlakken, dat beter geschikt is voor de grafische weergave dan de klassieke methode. De extra vrijheidsgraden die de NURPS gewichten met zich meebrengen werden onderzocht in het licht van vrije vorm ontwerpen. Dit gaf aanleiding tot de definitie van vormparameters, waardoor een gebruiker op een transparante manier de gewichten kan manipuleren, tot het modelleren van speciale effecten, en het exact voorstellen van een cilinder, kegel en een onvolledige voorstelling van de bol met openingen rond de meridiaan.

4 Uniforme PS–splines en wavelets

4.1 Uniforme PS–splines

Met $\mathcal{K} = \Gamma B \mathbb{Z}^2$,

$$\Gamma = \begin{bmatrix} 1 & 1/2 \\ 0 & \sqrt{3}/2 \end{bmatrix}, \tag{18}$$

beduiden we een uniform driehoeksrooster bestaande uit drie oneindige verzamelingen van parallelle lijnen op afstand B en onder een hoek $\pi/3$. We beschouwen $\mathcal{K}$ als een verzameling van punten V_i en driehoeken $\rho_{i,j,k}$. Dit rooster kan opgesplitst worden in een aantal disjuncte deelroosters:

$$\mathcal{K} = \bigcup_{i=0}^{3} (D\mathcal{K} + k_i), \tag{19}$$

met $k_i \in \mathcal{K}, k_0 = 0$, en

$$D = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}. \tag{20}$$

Zij Ω een veelhoek waarvan de zijden gelegen zijn op het rooster $\mathcal{K}$. De driehoeken $\rho_{i,j,k} \in \mathcal{K}$ die binnen de veelhoek liggen, vormen een uniforme driehoeksverdeling Δ van Ω. Een verfijning van Δ, bekomen door elke zijde in twee te splitsen wordt genoteerd als $D^{-1}\Delta$. In dit hoofdstuk bestuderen we PS–splines over uniforme driehoeksverdelingen, genaamd uniforme Powell–Sabin (UPS–) splines. Deze hebben een aantal voordelen ten opzichte van PS–splines over willekeurige driehoeksverdelingen:

1. Een PS–verfijning kan eenvoudig bepaald worden door de bissectrice van elke hoek te tekenen.

2. Er kan op voorhand een PS–driehoek gevonden worden die de PS–punten bevat. In dit werk leggen we de vorm van de PS–driehoeken vast voor alle hoekpunten: we gebruiken de PS–driehoek die optimaal is voor een molecule met zes driehoeken. Deze bevat zeker de PS–punten voor moleculen met minder driehoeken. Het oplossen van een kwadratisch programmeringsprobleem valt dus weg.

3. Hierdoor ligt de B–spline basis op voorhand vast, en kunnen een aantal algoritmen efficiënter geïmplementeerd worden. We geven het voorbeeld van

   (a) Een UPS–spline als oplossing van een interpolatieprobleem (12).

   (b) De berekening van de stuksgewijze Bernstein–Bézier voorstelling van een UPS–spline.

   (c) De integratie van een UPS–spline.

Ondanks de vormbeperking op het domein Ω, kunnen met UPS–splines oppervlakken met drie, vier, vijf of zes zijden voorgesteld worden; dit zijn de meest voorkomende gevallen.

4.2 Subdivisie van UPS–splines

Een gegeven UPS–spline $s(u) \in S_2^1(\Delta^*)$ kan voorgesteld worden over een verfijning $D^{-1}\Delta$ van de driehoeksverdeling, $s'(u) \in S_2^1(D^{-1}\Delta^*)$. We leiden formules af voor het berekenen van de controlepunten van $s'(u)$ als convexe combinaties van deze van $s(u)$.

Oude hoekpunten. De controlepunten van $s'(u)$ ter hoogte van een hoekpunt $V_i \in \Delta$ worden gegeven door

$$\boldsymbol{c}'_{i,k} = \frac{2}{3}\boldsymbol{c}_{i,k} + \frac{1}{6}\left(\boldsymbol{c}_{i,(k+1)_3+1} + \boldsymbol{c}_{i,(k+2)_3+1}\right), \quad k = 1,2,3, \tag{21}$$

met $i_3 = i \bmod 3$.

Nieuwe hoekpunten. De controlepunten van $s'(u)$ ter hoogte van een hoekpunt $V_i \in D^{-1}\Delta \setminus \Delta$ worden gegeven door

$$\begin{aligned} \boldsymbol{c}_{i,j,1} &= \frac{1}{2}\left(\boldsymbol{c}_{i,2} + \boldsymbol{c}_{i,3}\right) \\ \boldsymbol{c}_{i,j,2} &= \frac{1}{2}\boldsymbol{c}_{j,1} + \frac{1}{4}\left(\boldsymbol{c}_{i,2} + \boldsymbol{c}_{j,2}\right) \\ \boldsymbol{c}_{i,j,3} &= \frac{1}{2}\boldsymbol{c}_{j,1} + \frac{1}{4}\left(\boldsymbol{c}_{i,3} + \boldsymbol{c}_{j,3}\right) \end{aligned} \tag{22}$$

Meer algemeen kan een verfijning van de gegeven driehoeksverdeling bekomen worden door de zijden in k segmenten ($k \geq 2$) van lengte B/k te splitsen. We spreken dan van k–subdivisie. We geven een oplossing voor het k–subdivisieprobleem langsheen de randen van Δ: de controlepunten voor de hoekpunten van de verfijnde verdeling langsheen de rand $\partial\Delta$ kunnen geschreven worden als convexe combinaties van de oorspronkelijke controlepunten. Hiertoe steunen we op de oplossing van het interpolatieprobleem voor UPS–splines. Ook voor de andere nieuwe punten kunnen dergelijke uitdrukkingen bekomen worden, maar deze worden zeer complex en voor onze verdere toepassingen is k–subdivisie langsheen de rand voldoende.

4.3 UPS–spline wavelets

Het subdivisieschema voor UPS–spline wavelets kan ingepast worden in een lifting schema voor de constructie van wavelets over driehoeksroosters. Vooreerst tonen we aan dat de B–spline basisfuncties $B_k^j(u), k = 1, \ldots, n$ over Δ translaties en dilataties zijn van slechts drie functies

$$\Phi(u) = \begin{bmatrix} B_i^1(u) \\ B_i^2(u) \\ B_i^3(u) \end{bmatrix} \tag{23}$$

met $i \in \{1, \ldots, n\}$. We spreken van de multi–schaalfunctie. Deze kan uitgedrukt worden in termen van translaties en dilataties van zichzelf in de zogenaamde dilatatievergelijking. Dit laat ons toe om een multiresolutie voor UPS–splines te definiëren door $L_2(\mathbb{R}^2)$ te ontbinden in een rij van geneste deelruimten $\mathcal{V}_j = S_2^1(D^{-j}\Delta^*) \subset L_2(\mathbb{R}^2), j \in \mathbb{Z}$.

We gebruiken het lifting schema [13, 45, 68, 69, 70] om een wavelet transformatie te construeren. Dit schema heeft als voordeel dat de inverse transformatie eenvoudig terug te vinden is, door het achterwaarts te doorlopen. We spreken in deze context vaak over een signaal in plaats van een UPS–spline. Het idee is om een gegeven UPS–spline $s(u)$ te ontbinden in een laagfrequente en een aantal hoogfrequente componenten (die het detail weergeven), op een manier waarop het oorspronkelijke signaal can gereconstrueerd worden. Dit kan door te vertrekken van een triviale transformatie, en in een aantal lifting–stappen de eigenschappen van dit schema te verfijnen. We beschouwen hier drie stappen:

Splitsing. De triviale of luie wavelet transformatie splitst het gegeven signaal in vier deelsignalen overeenkomstig de vier disjuncte deelroosters $D\Delta + t_j$ van de oorspronkelijke driehoeksverdeling. De coëfficienten op het nulde deelrooster noemt men de schaalcoëfficiënten.

Predictie. Vertrekkende van het signaal op het nulde deelrooster $D\Delta$ worden de signalen op de andere deelroosters voorspeld. Dat kan door te beginnen met de controledriehoeken van het nulde deelrooster te scaleren met de inverse transformatie van (21). Dit brengt hen op een lager resolutieniveau. Van daaruit kunnen de controledriehoeken op de andere deelroosters voorspeld worden met de subdivisieregel (22). De predictie wordt afgetrokken van de werkelijke waarde, en dit levert de detail coëfficiënten of wavelet coëfficiënten. Voor kwadratische veeltermen zal de predictie exact zijn, en zijn de wavelet coëfficiënten gelijk aan nul. In het algemeen zeggen de wavelet coëfficiënten in welke mate de predictie niet juist is. Voor zacht verlopende signalen worden dan ook kleine wavelet coëfficiënten verwacht.

Bijsturen. Het nulde deelrooster bevat nu (door de scalering) een UPS–spline over een grover rooster $D\Delta$, dit is een zachtere versie van het oorspronkelijke signaal. Het is de laagfrequente componente. In de laatste stap wordt het signaal op het nulde deelrooster bijgestuurd vanop de andere deelroosters, zodanig dat het gemiddelde signaal over de laagfrequente componente gelijk blijft.

De basisfuncties voor de laagfrequente componente zijn vervat in de multi–schaalfunctie. De basisfuncties voor de detailsignalen kunnen gevonden worden door het schema achterwaarts te doorlopen, vetrekkende van één wavelet coëfficiënt gelijk aan één en de rest nul. Dit levert drie multi–wavelets $\Psi_i(u) := [\psi_{i,1}(u), \psi_{i,2}(u), \psi_{i,3}(u)]^T$, overeenkomstig $D\Delta + k_i, i = 1, 2, 3$. We tonen aan dat de integraal van een wavelet die volledig binnen Ω gelegen is, gelijk is aan nul. De ontbinding van een gegeven signaal via het lifting schema kan nu herhaald worden op de laagfrequente componente. Dit geeft aanleiding tot de wavelet decompositie:

$$s = \sum_{j=L}^{J-1} \sum_{k \in D^j \mathcal{K}} \sum_{i=1}^{3} W_{i;j,k} \Psi_{i;j,k} + \sum_{k \in D^L \mathcal{K}} C_{L,k} \Phi_{L,k}, \quad W_{i;j,k}, C_{L,k} \in \mathbb{R}^{3\times 1}. \tag{24}$$

In de klassieke wavelet theorie zijn de schaalfuncties en de wavelets translaties en dilataties van telkens één functie. Voor UPS–splines zijn de schaalfuncties en wavelets translaties en dilataties van telkens drie functies. We spreken dan ook van multi–wavelets. De uitbreiding van gewone naar multi–wavelets is bestudeerd in o.a. [12, 14, 35, 56]. Het voordeel van multi–wavelets is dat men, door complexere bijsturings–stappen te ontwerpen, ervoor kan zorgen dat de wavelets aan meerdere eigenschappen tegelijkertijd voldoen, zoals orthogonaliteit, symmetrie, lokaliteit, continuïteit en reproductie van veeltermen tot zekere graad. In deze thesis geven we enkele suggesties omtrent het vinden van een bijsturings–stap die ervoor zorgt dat de wavelets in de buurt van de rand $\partial\Omega$ een integraal gelijk aan nul hebben. Bovendien bewijzen we dat er in de predictie–stap wel degelijk van het multi–wavelet aspect gebruik gemaakt wordt.

Als praktische toepassing van de wavelet transformatie geven we een algoritme voor het verwijderen van ruis uit een UPS–spline oppervlak door middel van een drempel–algoritme. Dit is een klassieke aanpak in signaalverwerking [24, 25, 26, 27, 39], die steunt op de veronderstelling dat het signaal kan voorgesteld worden door een klein aantal grote coëfficiënten. De overige, kleine coëfficiënten worden als minder belangrijk beschouwd, en zijn relatief meer aangetast door ruis. Ze worden dan ook weggelaten. De idee is te starten met een wavelet transformatie van het gegeven signaal, van bijvoorbeeld f stappen. Dan volgen er evenveel drempel–stappen. In elke stap worden de wavelet coëfficiënten gewijzigd: degene die in absolute waarde kleiner zijn dan een waarde $\epsilon_k, k = 1, \ldots, f$, worden gelijk gesteld aan nul; voor de andere zijn er twee mogelijkheden:

Een harde drempel. De overige wavelet coëfficiënten blijven ongewijzigd;

Een zachte drempel. De absolute waarde van de wavelet coëfficiënten wordt verminderd met ϵ_k, terwijl hun teken onveranderd blijft.

Om de drempel–stap te beëindigen wordt de inverse wavelet transformatie doorgevoerd. We hebben dit schema getest in een aantal experimenten, waaruit bleek dat de zachte drempel resulteert in een betere ruisverwijdering, maar ook dat het uiterst moeilijk is om a priori een geschikte waarde voor ϵ_k op te stellen.

Een andere, meer praktische toepassing van het subdivisieschema voor UPS–splines is de grafische weergave van UPS–spline oppervlakken. Door herhaaldelijk subdivisie toe te passen, worden een steeds groter aantal controledriehoeken gegenereerd. Deze zijn telkens kleiner, sluiten nauwer aan bij het oppervlak en zijn rakend aan het oppervlak in hun zwaartepunt. Door de raakpunten te verbinden, bekomen we een draadmodel van het oppervlak. Tevens is de nodige informatie voorhanden om de normalen in de gegenereerde punten te vinden, wat leidt tot een visualisatie als realistisch oppervlak, met belichting. We stellen een blok matrix structuur voor als gegevensstructuur voor het UPS–spline oppervlak. Zodoende kunnen de punten op het oppervlak gevonden worden door het uitvoeren van lokale matrixbewerkingen. Het schema kan dan ook hardwarematig geïmplemeteerd worden. Verder zullen we zien dat deze structuur ook aanleiding geeft tot een zeer efficiënte visualisatie in OpenGL. Ons visualisatie algoritme gebruikt enkel convexe combinaties van de controlepunten. Het kan daardoor onmiddellijk uitgebreid worden naar een rationaal schema voor de weergave van rationale UPS–spline oppervlakken. Als toepassing geven we eerst een algemene methode om de PS–driehoek van een zeker hoekpunt te vervangen door een andere driehoek, die ook de PS–punten bevat. Concreet gebruiken we dit om de optimale voorstelling van een (rationaal) UPS–spline oppervlak om te zetten in een uniforme voorstelling. Dit laat toe de rationale UPS–spline oppervlakken uit het vorige hoofdstuk die een cylinder, kegel en bol weergeven, en gegeven waren in hun optimale voorstelling, te visualiseren.

4.4 Besluit

We hebben Powell–Sabin splines over uniforme driehoeksverdelingen bestudeerd. Ze hebben een aantal voordelen ten opzichte van PS–splines over willekeurige driehoeksverdelingen, terwijl ze toch toelaten om de meest voorkomende geometrieën (drie tot zes zijden) voor te stellen. Door de vorm van de PS–driehoeken op voorhand vast te leggen, wordt vermeden dat een kwadratisch programmeringsprobleem moet opgelost worden voor elk hoekpunt, bij het opstellen van de basis. Uniforme PS–driehoeken leiden tot efficiënte algoritmen voor het interpolatieprobleem, de integratie van UPS–splines en het terugvinden van de Bernstein–Bézier voorstelling. Bovendien hebben we een efficiënt en numeriek stabiel subdivisieschema voorgesteld, dat gebruikt kan worden voor de grafische weergave van UPS–spline oppervlakken, maar ook aanleiding geeft tot een nieuwe wavelet transformatie over regelmatige driehoeksroosters. Als toepassing van dit laatste hebben we een algoritme gegeven voor het verwijderen van ruis uit een UPS–spline oppervlak.

5 Het probleem van de veelhoekige opening

5.1 Inleiding

Een vaak voorkomend probleem in Computer Gesteund Geometrisch Ontwerpen is het opvullen van een opening, begrensd door een aantal gegeven oppervlakken. De meeste methoden gepubliceerd in de literatuur gaan uit van veronderstellingen over de wiskundige vorm van de omliggende oppervlakken [9, 10, 32, 50]. In dit werk geven we een methode die dat niet doet, en daardoor algemeen toepasbaar is. Anderzijds zullen de gegeven randcurven slechts benaderd worden door het opvullende oppervlaktesegment, maar dat is voor praktische toepassingen geen bezwaar. De gegevens die de gebruiker moet voorzien bestaan uit de randcurven $\boldsymbol{p}$, de eenheidsraakvectoren $\vec{\gamma}$ en de eenheidsnormaalvectoren $\vec{n}$ aan de omliggende oppervlakken, telkens geparametriseerd op het eenheidsinterval. Het algoritme berekent dan een vector $\vec{\delta} = \vec{n} \times \vec{\gamma}$. We zullen ernaar verwijzen alsof ze door de gebruiker voorzien werd. De gegevens in een punt V_i worden genoteerd als $(\boldsymbol{p}_i, \vec{\gamma}_i, \vec{\delta}_i)$. Ons algoritme berekent een opvullend oppervlak in de vorm van een UPS–spline oppervlak, en bestaat in het algemeen uit drie fazen:

eerst wordt een initiële oplossing opgesteld, deze wordt dan verfijnd op de rand; en tenslotte worden de inwendige controledriehoekjes gezocht. Vooreerst zullen we bekijken hoe interpolatie in de randpunten en benadering in tussenliggende punten kan bekomen worden.

5.2 De benaderingsvergelijkingen

We zullen de controledriehoekjes in $V_i \in \partial\Delta$ bepalen door de voorziene gegevens in die punten te interpoleren via de zogenaamde interpolatievergelijkingen. Dit levert een aantal vrijheidsgraden op, die kunnen benut worden voor het benaderen van de randcurven tussen de interpolatiepunten in. Voor het opstellen van de interpolatievergelijkingen maken we onderscheid tussen

1. **Interpolatie voor inwendige randpunten.** Voor de punten $V_i \in \partial\Delta$ die geen hoekpunten zijn, wordt een controledriehoekje gegeven door

$$\left\{ \begin{array}{lcl} \boldsymbol{c}_{i,1} & = & \boldsymbol{p}_i - \alpha_i \vec{\gamma}_i \\ \boldsymbol{c}_{i,2} & = & \boldsymbol{p}_i + \frac{\alpha_i}{2}\vec{\gamma}_i + \beta_i\vec{\delta}_i \\ \boldsymbol{c}_{i,3} & = & \boldsymbol{p}_i + \frac{\alpha_i}{2}\vec{\gamma}_i - \beta_i\vec{\delta}_i . \end{array} \right. \tag{25}$$

Het controledriehoekje ligt dus in het vlak $\boldsymbol{p}_i + \mu\vec{\gamma}_i + \nu\vec{\delta}_i, \mu, \nu \in \mathbb{R}$, voldoet aan $\boldsymbol{s}(V_i) = \boldsymbol{p}_i$, en er schieten twee vrijheidsgraden over: α_i en β_i (we zullen verder spreken over de α– en de β–factoren).

2. **Interpolatie voor hoekpunten.** Een controledriehoekje voor de overige hoekpunten $V_i \in \partial\Delta$, wordt gegeven door

$$\left\{ \begin{array}{lcl} \boldsymbol{c}_{i,1} & = & \boldsymbol{p}_i - \alpha_1 \vec{\gamma}_1 \\ \boldsymbol{c}_{i,2} & = & \boldsymbol{p}_i + \alpha_1\vec{\gamma}_1 + \alpha_2\vec{\gamma}_2 \\ \boldsymbol{c}_{i,3} & = & \boldsymbol{p}_i - \alpha_2\vec{\gamma}_2 , \end{array} \right. \tag{26}$$

waarbij $\vec{\gamma}_1$ de raakvector is aan de inkomende curve en $\vec{\gamma}_2$ deze aan de uitgaande curve. Het controledriehoekje ligt dus in het vlak $\boldsymbol{p}_i + \mu\vec{\gamma}_1 + \nu\vec{\gamma}_2, \mu, \nu \in \mathbb{R}$ met $\boldsymbol{s}(V_i) = \boldsymbol{p}_i$, en opnieuw zijn er twee vrijheidsgraden, α_1 en α_2.

Deze vrijheidsgraden kunnen benut worden door interpolatie van de gegevens op te leggen in het punt $V_{i,j}$ halfweg tussen twee naburige punten V_i en V_j. De voorwaarde $\boldsymbol{s}(V_{i,j}) = \boldsymbol{p}_{i,j}$ geschreven in termen van de α–vrijheidsgraden voor V_i en V_j is dan

$$\alpha_i\vec{\gamma}_i - \alpha_j\vec{\gamma}_j = 4\boldsymbol{p_{i,j}} - 2(\boldsymbol{p_i} + \boldsymbol{p_j}) = \boldsymbol{q_{i,j}} . \tag{27}$$

Een analoge voorwaarde voor interpolatie van de $\vec{\delta}$–vector in $V_{i,j}$ kan geschreven worden in termen van β_i en β_j:

$$\beta_{i,j}\vec{\delta}_{i,j} = \frac{1}{2}(\beta_i\vec{\delta}_i + \beta_j\vec{\delta}_j). \tag{28}$$

Dit zijn de benaderingsvergelijkingen.

5.3 Het algoritme

Het algoritme bestaat uit drie fazen:

1. **Een initiële oplossing** wordt bekomen door (26) op te leggen in elk hoekpunt van de driehoeksverdeling, en de benaderingsvergelijking (27) in kleinste kwadraten zin op te lossen voor elke zijde V_iV_j. Dit levert

$$\alpha_i = \frac{1}{D}\left((\vec{\gamma}_i \cdot \boldsymbol{q}_{i,j}) - (\vec{\gamma}_j \cdot \boldsymbol{q}_{i,j})(\vec{\gamma}_i \cdot \vec{\gamma}_j)\right) \quad (29)$$

$$\alpha_j = \frac{1}{D}\left(-(\vec{\gamma}_j \cdot \boldsymbol{q}_{i,j}) + (\vec{\gamma}_i \cdot \boldsymbol{q}_{i,j})(\vec{\gamma}_i \cdot \vec{\gamma}_j)\right), \quad (30)$$

met $D = 1 - (\vec{\gamma}_i \cdot \vec{\gamma}_j)^2$ voor het geval dat $\vec{\gamma}_i \neq \pm\, \vec{\gamma}_j$. Als daarentegen $\vec{\gamma}_i = \pm\vec{\gamma}_j$, stellen we

$$\alpha_i = \alpha_j = \frac{1}{2}||\boldsymbol{p}_j - \boldsymbol{p}_i||. \quad (31)$$

Deze oplossing is exact indien de rand een rechte lijn is. Eens de α–factoren gekend zijn voor de hoekpunten, kunnen de overeenkomstige β–factoren als schaalfactoren bij de δ–vectoren eenduidig berekend worden. De initiële oplossing is bepaald over het hele gebied Ω. Hoe dit bereikt kan worden voor een vijfhoek en een zeshoek zullen we later toelichten.

2. **De iteratiestap** begint met subdivisie van de oplossing uit de vorige stap. Dit kan door de α– en β–factoren te delen door twee. Dan wordt in elk punt V_k halfweg tussen V_i en V_j een nieuwe controledriehoek volgens (25) gecreëerd, en de vrijheidsgraden worden afgeleid door de interpolatievoorwaarden op te leggen in $V_{i,k}$ en $V_{k,j}$. Dit levert:

$$\alpha_k = \frac{1}{2}(\vec{\gamma}_k \cdot (\alpha_i\vec{\gamma}_i - \alpha_j\vec{\gamma}_j + \boldsymbol{q}_{k,j} - \boldsymbol{q}_{i,k})), \quad (32)$$

en

$$\beta_k = \frac{\beta_i(\vec{\delta}_i \cdot \vec{\delta}_{i,k})(\vec{\delta}_k \cdot \vec{\delta}_{i,k}) + \beta_j(\vec{\delta}_j \cdot \vec{\delta}_{k,j})(\vec{\delta}_k \cdot \vec{\delta}_{k,j}) - \beta_i(\vec{\delta}_i \cdot \vec{\delta}_k) - \beta_j(\vec{\delta}_j \cdot \vec{\delta}_k)}{2 - (\vec{\delta}_k \cdot \vec{\delta}_{i,k})^2 - (\vec{\delta}_k \cdot \vec{\delta}_{k,j})^2}. \quad (33)$$

Als $\vec{\delta}_{i,k} = \vec{\delta}_k = \vec{\delta}_{k,j}$ (zoals voor een vlakke curve) dan bepalen we β_k uit subdivisie:

$$\beta_k = \frac{1}{2}\left(\vec{\delta}_k \cdot (\beta_i\vec{\delta}_i + \beta_j\vec{\delta}_j)\right). \quad (34)$$

Het iteratieschema wordt een vast aantal stappen uitgevoerd, of tot de residu's $r_{i,k} = \|\boldsymbol{q_{i,k}} - \alpha_i \vec{\gamma}_i + \alpha_k \vec{\gamma}_k\|$ (en $r_{k,j}$ analoog) kleiner zijn dan een waarde ϵ.

3. **De inwendige controlepunten.** Hierop komen we uitgebreid terug in de volgende paragrafe.

5.4 De inwendige controlepunten

De doelstelling is een vloeiend oppervlak te vinden dat de gegeven opening opvult. De initiële oplossing kent een vloeiend verloop, en is gedefinieerd over het hele gebied Ω, maar sluit over het algemeen niet nauwkeurig aan bij de rand. Na de laatste iteratiestap wordt de rand wel voldoende nauwkeurig benaderd, maar rest er het probleem van de inwendige controledriehoekjes te berekenen. We geven eerst een aantal intuïtieve methoden die proberen de zachtheid van de initiële oplossing en de kwaliteit van de uiteindelijke benadering op de rand te combineren. Nadien bespreken we een wiskundig meer gefundeerde oplossing, die probeert de zachtheid van het inwendig verloop te optimalizeren.

1. **Kopie van de initiële oplossing.** De inwendige controlepunten kunnen bekomen worden door subdivisie van de initiële oplossing. Zoals gesteld, zal deze vorm niet noodzakelijk goed aansluiten bij het verloop aan de rand van het oppervlak, en geeft ze vaak aanleiding tot ongewenste verschijnsels aan de rand.

2. **Uitmiddelen van de omliggende.** Een controledriehoekje in het inwendige kan berekend worden als het gemiddelde van de zes omliggende controledriehoekjes. Vertrekkende vanaf de rand, berekent men in een aantal stappen telkens een ring van driehoekjes die iets meer naar het midden van het oppervlak ligt dan de vorige ring. In elke ring wordt een controledriehoekje berekend door de zes omliggende uit te middelen. Deze zijn afkomstig van de gekende randoplossing, de initiële oplossing, of van de reeds eerder uitgemiddelde oplossing van een vorige ring. Op die manier worden kenmerken van de rand naar het midden toe uitgemiddeld. Deze methode heeft echter de neiging om ook de ongewenste artefacten uit de vorige oplossing naar het inwendige toe uit te middelen.

3. **Onmiddellijk bijsturen.** Een goed compromis zou zijn om kenmerken van de rand in rekening te brengen vooraleer de iteratie faze is afgelopen. Dat kan door het inwendige te berekenen uit subdivisie van de initiële oplossing, waarbij de rand telkens overschreven wordt met de meest recente benadering. Op die manier worden randkenmerken

naar het midden toe in rekening gebracht zo gauw ze beschikbaar zijn in de benadering.

4. **Bruggen bouwen.** Een laatste intuïtieve methode is gebaseerd op k–subdivisie. De idee is om met dit subdivisieschema bruggen van controledriehoeken te bouwen die op een continue wijze telkens twee controledriehoekjes op overstaande zijden verbinden. Men kan op die manier bruggen bouwen in drie richtingen, en de resultaten uitmiddelen in elk inwendig punt V_i.

5. **Stroomlijnen.** Een gekende techniek voor het genereren van esthetisch aantrekkelijke oppervlakken is het stroomlijnen [44, 61, 80]. Door wiskundige uitdrukkingen op te stellen die een maat geven van de welgevormdheid van een oppervlak, kan men een doelfunctie opstellen. Het maximalizeren van deze doelfunctie, onder een aantal technische beperkingen zoals interpolatie, levert een gestroomlijnd oppervlak. In dit werk gebruiken we een maat voor het vloeiend zijn van een PS–spline [20], en passen deze toe voor het stroomlijnen van UPS–spline oppervlakken. Het idee is om voorwaarden op te stellen opdat elke componente van een PS–spline oppervlak één enkele veelterm is op Ω. Deze voorwaarden vormen een overgedetermineerd stelsel. De oplossing hiervan in de kleinste kwadraten zin komt neer op het minimalizeren van de sprong in de tweede orde afgeleiden van de componenten, en levert een gestroomlijnd oppervlak. Als beperking stellen we voorop dat de controledriehoekjes langsheen de rand gekend zijn. Op die manier is de techniek van het stroomlijnen onmiddellijk toepasbaar voor het berekenen van de inwendige controledriehoekjes bij het probleem van de veelhoekige opening.

We hebben deze verschillende methoden getest door de openingen rond de meridiaan van de onvolledige NURPS voorstelling van de bol op te vullen. Om de resultaten te vergelijken, hebben we de Gaussiaanse kromming van de opvullende oppervlakken vergeleken. In het ideaal geval zou de opvulling op de bol gelegen zijn, en zou de kromming in alle punten gelijk zijn. We weten echter dat dit niet mogelijk is met een niet–rationale voorstelling. Daarom streven we naar een zo egaal mogelijk verloop van de Gaussiaanse kromming. Als besluit kunnen we stellen dat op het "onmiddellijk bijsturen" na, de intuïtieve methoden aanleiding geven tot grote fluctuaties in de kromming. Het beste resultaat werd bekomen door de methode van het stroomlijnen, maar het goedkopere alternatief van "onmiddellijk bijsturen" is zeker ook de moeite waard. We kunnen stellen dat het opvullen van een opening op een vloeiende manier, zonder bijkomende gegevens over het inwendige, geen eenvoudige opdracht is, en dat een algemeen toepasbare methode moeilijk te vinden is. Tot slot merken we op dat het drempel algoritme, mits een goede drempelwaarde, kan toegepast worden om de resultaten, bekomen via de intuïtieve metoden, verder te verbeteren.

5.5 Opmerking over het aantal zijden

De geschetste methoden kunnen onmiddellijk toegepast worden voor drie– en vierhoekige geometrieën. Voor een oppervlak met vijf of zes zijden, moeten er echter extra controledriehoekjes gevonden worden:

1. **Op de rand.** Het driehoekje V_k op de rand, gelegen tussen V_i en V_j, kan gevonden worden door de benaderingsvergelijkingen op te lossen in $V_{i,k}$ en $V_{k,j}$.

2. **In het inwendige.** Van zodra de controlepunten langsheen de rand gekend zijn, kan een controledriehoekje voor het inwendige gevonden worden door het stroomlijn–algoritme toe te passen. We hebben daartoe dit algoritme specifiek uitgewerkt voor het geval dat Δ een molecule is met zes driehoeken.

5.6 Verdere toepassingen

Dankzij de algemene vorm van de invoervereisten, is het algoritme voor het probleem van de veelhoekige opening ook toepasbaar in andere situaties waarbij een stel randcurven, raakvectoren en normaalvectoren gegeven zijn, en een aansluitend oppervlak dient gezocht te worden. Zo hebben we het schema toegepast voor het vloeiend verbinden van twee snijdende cilinders, en het plaatsen van een kap bovenop een prisma.

5.7 Besluit

We hebben een algemeen toepasbaar algoritme gegeven voor het opvullen van een veelhoekige opening. De gebruiker dient enkel de randcurven te voorzien, net als de raakvectoren en de normaalvectoren aan de omliggende oppervlakken. Er wordt dan een UPS–spline oppervlak berekend dat de gegeven randkrommen in een aantal punten interpoleert, en tussen deze punten de gegevens benadert. Onze oplossing maakt gebruik van subdivisie voor UPS–splines. Voor het berekenen van de inwendige controledriehoekjes hebben we een aantal intuïtieve methoden onderzocht, net als een klassieke techniek voor het stroomlijnen van oppervlakken. Uit onze tests blijkt dat een algemeen toepasbare methode moeilijk te geven is, en dat de keuze voor een stuk afhangt van wat de gebruiker verwacht. Zo levert de stroomlijn–methode goede resultaten voor het opvullen van een opening in een bol, terwijl we later in een ander voorbeeld voor “onmiddellijk bijsturen” zullen kiezen. Ons algoritme is toepasbaar voor problemen met drie tot zes zijden, en kan ook gebruikt worden voor o.a. het vloeiend aaneensluiten van snijdende oppervlakken, of het plaatsen van een kap bovenop een prisma.

6 PS-surf: een CAD software prototype

Samen met het onderzoek, hebben we een programma geschreven voor het werken met UPS–spline oppervlakken. Het werd geschreven in `Visual C++`, gebruik makend van de Microsoft Foundation Classes (MFC). Voor het grafische gedeelte doen we een beroep op OpenGL [83]. We lichten hier de keuzes toe die we gemaakt hebben in de gegevensvoorstelling, om een efficiënte visualisatie mogelijk te maken. Ook bespreken we de globale structuur van het project, de belangrijkste mogelijkheden van het programma, en we gebruiken het in een aantal toepassingen met realistische CAD modellen.

6.1 Gegevensvoorstellingen in een interactieve omgeving

We geven een objectgerichte (OO)–voorstelling voor PS–splines over willekeurige driehoeksverdelingen, en geven aan hoe ermee kan gewerkt worden om oppervlakken als een net van driehoeken te visualiseren met OpenGL. Het blijkt dat de efficiëntie sterk afhankelijk is van het aantal functie aanroepen, en van de redundante verwerking van de hoekpunten die gemeenschappelijk zijn tussen naburige driehoeken. Het concept van een "hoekpuntenrij" wordt toegepast om het aantal functie aanroepen en de redundantie te verminderen. De gegevensvoorstelling is echter niet de meest efficiënte wanneer we werken met UPS–spline oppervlakken. In dat geval verkiezen we de blok matrix structuur. We tonen aan dat het aantal functie aanroepen en de redundante verwerking van gemeenschappelijke hoekpunten daardoor drastisch kunnen verminderd worden zonder bijkomende informatie te moeten berekenen. Hiertoe combineren we de voordelen de "hoekpuntenrij" met die van een "strook van driehoeken" als basis element. Een bijkomend voordeel van de blok matrix structuur is dat de topologsiche informatie van de driehoeksverdeling er inherent in vervat zit, zodat hier geen aparte klassen voor moeten voorzien worden.

6.2 PS–surf

De architectuur van het pakket is modulair. PS–surf is het uitvoerbare programma, dat met behulp van vensters en dialoogschermen de gebruiker toelaat met UPS–splines te werken. Het steunt op de Microsoft Foundation

Classes (MFC). Het eigenlijke werk gebeurt echter in een aantal onderliggende draagbare bibliotheken:

MyOpenGL biedt een aantal klassen voor het werken met geometrische objecten (kubus, bol, UPS–spline oppervlak, NURBS oppervlak,...), en voor de visualisatie ervan (standpunt van de waarnemer, belichting, materiaal, beelden van een scene,...). Alle voorwerpen kunnen op een hiërarchische manier georganiseerd worden in scènes.

MyGeometry implementeert een aantal geometrische hulproutines, zoals de berekening van een scalair produkt. Verder bevat ze een aantal `FITPACK` [17] routines voor het evalueren en afleiden van B–spline functies.

MyLinLag bevat een klasse voor het oplossen van overgedetermineerde stelsels via Givens orthogonalisatie.

Alle modules zijn volgens OO–principes opgebouwd. We illustreren met enkele voorbeelden hoe we gebruik maken van de voordelen van deze aanpak.

1. Alle klassen die geometrische voorwerpen voorstellen, zijn afgeleid van een klasse `CGLObject`. Deze bevat functionaliteit voor het automatisch berekenen van een geschikt standpunt van de waarnemer, voor het instellen van de zichtbaarheid, voor het werken met materialen en het toepassen van geometrische transformaties. Alle klassen die afgeleid zijn van `CGLObject` kunnen automatisch in een boomstructuur georganiseerd worden in het programma PS–surf, en bewerkt worden via dialoogschermen. Het enige dat zo een klasse moet implementeren is de visualisatie met OpenGL, en de berekening van een omsluitende balk. Bijvoorbeeld, de klasse `CGLUPSSpline` erft onrechtstreeks van `CGLObject`, en beschikt automatisch over twee materialen: eentje voor het oppervlak en eentje voor het controlenet.

2. PS–surf kan in de toekomst uitgebreid worden naar PS–spline oppervlakken over willekeurige driehoeksverdelingen. Daartoe hebben we een gemeenschappelijke voorouder klasse voor PS–spline oppervlakken, `CGLPSSpline`, gecreeërd. Wanneer een afgeleide klasse de zuivere virtuele routines van deze abstracte klasse implementeert, wordt ze automatisch geïntegreerd in het programma PS–surf, dat specifieke mogelijkheden biedt voor PS–spline oppervlakken, zoals het bekijken van de driehoeksverdeling of het wegschrijven ervan in picTEX [1] of TEXdraw [41] formaat, het aanklikken van PS–driehoeken, het manipuleren van controledriehoeken, het uitvoeren van een wavelet transformatie en het bekijken van het spectrum van de wavelet coëfficiënten.

3. PS–surf kan uitgebreid worden met nieuwe CAD toepassingen, gebruik makend van bestaande krachtige klassen. Zo hebben we het algoritme voor het probleem van de veelhoekige opening geïntegreerd, net als het benaderen van NURBS oppervlakken door UPS–spline oppervlakken.

6.3 De functionaliteit van PS–surf

Algemene mogelijkheden

1. Het hoofdscherm bevat drie componenten: een boomstructuur voor de organisatie van voorwerpen in scènes, een venster met een ruimtelijk beeld van geselecteerde items, en een venster waarin het domein van een PS–spline oppervlak kan bekeken worden. In het ruimtelijk beeld kan het voorwerp interactief geroteerd worden met muisbewegingen.

2. Via de menubalk kunnen dialoogschermen geopend worden voor het bewerken van de materialen van de voorwerpen en het toepassen van geometrische transformaties. Hierbij werden alle mogelijkheden van OpenGL gebruikt, zoals drie kleur componenten voor materialen, transparantie en emissie van licht.

3. Lichtbronnen kunnen toegevoegd worden, met keuze van de positie, richting en kleur. Ze kunnen werken als spotlicht of als puntbron. De intensiteit kan verminderd worden in functie ven de afstand tot de lichtbron.

4. Elk voorwerp, ook een scène, kan bekeken worden vanuit een eigen standpunt van de waarnemer. De waarnemer kan vrij bewegen via knoppen onderaan het scherm.

5. Een voorwerp kan afgebeeld worden met vlakke inkleuring of Gourad inkleuring, als belicht model of als draadmodel, of met weergave van één van de verschillende types kromming volgens een eigen kleurcodering.

6. Invoer en uitvoer naar binair bestandsformaat zijn voorzien, alsook het inlezen van NURBS modellen via IGES [75] bestanden, en het opslaan van het ruimtelijk beeld in Bitmap formaat of naar het klembord.

Werken UPS–spline oppervlakken

1. Een beeld van de domeinveelhoek laat toe om PS–spline driehoeken aan te klikken. De overeenkomstige controlepunten en gewichten kunnen dan gewijzigd worden.

2. Volgende algoritmen kunnen toegepast worden op UPS–spline oppervlakken:
    (a) subdivisie,
    (b) stroomlijnen met randbeperkingen,
    (c) de wavelet transformatie,
    (d) verwijderen van ruis met het drempel-algoritme.
3. Een UPS–spline oppervlak kan interactief opgebouwd worden, of kan bekomen worden als resultaat van het probleem van de veelhoekige opening (eventueel op basis van een gegeven NURBS oppervlak), of als benadering van een NURBS oppervlak (zie verder).

6.4 Toepassingen in modellering

Reverse Engineering. Het gebruik van fysische modellen is een onderdeel van de ontwerpcyclus van heel wat werkstukken. Vooraleer de productiefaze kan beginnen, is er echter een CAD model van dergelijke voorwerpen nodig. Het omzetten van een fysisch model in een CAD model noemt men Reverse Engineering (RE). In [42] wordt een geïntegreerde aanpak voorgesteld voor het modelleren van vrije vorm oppervlakken in RE. Ze is gebaseerd op de NURBS voorstelling voor oppervlakken, omwille van haar beschikbaarheid in CAD paketten. NURBS voorstellingen zijn gebaseerd op een vierhoekige geometrie, terwijl voor heel wat praktische modellen het gebruik van driehoekige geometrieën noodzakelijk is. De auteur geeft daarom algoritmen voor het werken met gedegenereerde (of ontaarde) NURBS oppervlakken: door twee hoekpunten van een vierhoek te laten samenvallen, bekomt men een driehoek. Er zijn hier echter een aantal nadelen aan verbonden: de normale in het gedegenereerde hoekpunt is niet gedefinieerd, hetgeen een probleem kan zijn voor snijmachines in CAM systemen. Voorts is de dichtheid aan controlepunten relatief hoger in de buurt van het ontaarde punt, dan in de rest van het oppervlak. Dit veroorzaakt plaatselijk rimpels in het oppervlak. Als alternatief stellen we daarom voor een driehoekig UPS–spline oppervlak te gebruiken in plaats van een ontaard NURBS oppervlak. Concreet berekenen we een UPS–spline oppervlak via het algoritme van de veelhoekige opening, met de NURBS randcurven, hun afgeleiden en normalen als invoer. Het feit dat de normale in het ontaarde punt niet bestaat is geen probleem aangezien we daar enkel gebruik maken van de afgeleiden. Als voorbeeld tonen we hoe een achteruitkijkspiegel van een auto kan gemodelleerd worden als een combinatie van NURBS en UPS–spline oppervlakken. De keuze voor het berekenen van het inwendige viel op het "omiddellijk bijsturen", in dit geval gaf het "stroomlijnen" immers te vlakke resultaten.

Benadering van een NURBS oppervlak. Een tweede toepassing behelst het benaderen van een NURBS oppervlak

$$\boldsymbol{S}(\bar{u},\bar{v}), (\bar{u},\bar{v}) \in R = [\bar{u}_0, \bar{u}_0 + W] \times [\bar{v}_0, \bar{v}_0 + H], \tag{35}$$

met een vierzijdig UPS–spline oppervlak $\boldsymbol{s}(u,v)$. Dat kan door componentsgewijze interpolatie van de functiewaarden en partiële afgeleiden van het NURBS oppervlak in de hoekpunten V_i van de driehoeksverdeling. Zij (n_v+1) en (n_u+1) het aantal rijen, resp. kolommen van de driehoeksverdeling Δ. Noem $\theta : \Omega \to R$ de lineaire afbeelding die het domein Ω van $\boldsymbol{s}(u,v)$ afbeeldt op R, dan worden de controlepunten van het benaderend UPS–spline oppervlak gegeven door

$$\begin{aligned}
\boldsymbol{c}_{i,1} &= \boldsymbol{S}\left(\theta(V_i)\right) - \frac{W}{2n_u}\frac{\partial \boldsymbol{S}\left(\theta(V_i)\right)}{\partial \bar{u}} && (36)\\
\boldsymbol{c}_{i,2} &= \boldsymbol{S}\left(\theta(V_i)\right) - \frac{H}{2n_v}\frac{\partial \boldsymbol{S}\left(\theta(V_i)\right)}{\partial \bar{v}} && (37)\\
\boldsymbol{c}_{i,3} &= \boldsymbol{S}\left(\theta(V_i)\right) + \frac{W}{2n_u}\frac{\partial \boldsymbol{S}\left(\theta(V_i)\right)}{\partial \bar{u}} + \frac{H}{2n_v}\frac{\partial \boldsymbol{S}\left(\theta(V_i)\right)}{\partial \bar{v}}, && (38)
\end{aligned}$$

voor $i = 1, \ldots, n$. Men kan ervoor zorgen dat het UPS–spline model ongeveer evenveel controlepunten gebruikt als het NURBS model, door n_u en n_v te kiezen als

$$n_u \approx \frac{\sqrt{3}}{3} k_{\bar{u}} - 1, \quad n_v \approx \frac{\sqrt{3}}{3} k_{\bar{v}} - 1, \tag{39}$$

waarbij $k_{\bar{u}}$ en $k_{\bar{v}}$ het aantal controlepunten zijn van het NURBS model in de $\bar{u}-$, resp. $\bar{v}-$richting. Als voorbeeld hebben we een UPS–spline model opgesteld van de rugleuning van een autostoel.

6.5 Besluit

PS–surf is een computerprogramma dat het werken met UPS–spline oppervlakken demonstreert. Voor de grafische weergave doet het een beroep op OpenGL. We hebben aangetoond dat de blok matrix structuur voor UPS–splines aanleiding geeft tot een efficiënte visualisatie in OpenGL. Het hoofdprogramma steunt op een aantal andere modules waarin het eigenlijke rekenwerk gebeurt. Deze zijn volgens de principes van object georiënteerd ontwerp opgebouwd. De voordelen van overerving werden geïllustreerd met enkele voorbeelden, met name het gebruik van een krachtige basis klasse, de uitbreiding naar PS–splines over willekeurige driehoeksverdelingen, en het integreren van nieuwe CAD toepassingen. We hebben een overzicht gegeven van de mogelijkheden die het programma aanbiedt. Tot slot hebben we

PS–surf gebruikt voor het modelleren van realistische voorwerpen, enerzijds om het gebruik van ontaarde NURBS oppervlakken te vermijden, en anderzijds om een volledig NURBS model te benaderen door een UPS–spline oppervlak.

7 Besluit en toekomstig onderzoek

7.1 Het belang van PS–splines in CAGD

In dit proefschrift werd het gebruik van Powell–Sabin (PS)–splines in CAGD bestudeerd. PS–splines hebben een aantal voordelen vergeleken met de alom gebruikte tensor product B–splines en NURBS, maar ook ten opzichte van de theoretisch goed onderbouwde Bernstein–Bézier veeltermen over driehoeken:

1. Ze laten toe oppervlakken te definiëren over een domein met een willekeurige geometrie. Tensor product B–splines zijn beperkt tot rechthoekige Domeinen.

2. Lokale adaptiviteit is mogelijk in de driehoeksverdeling, terwijl voor tensor product B–splines het domein in een regelmatig rooster opgedeeld wordt door de knopen.

3. In hun genormalizeerde B–spline voorstelling hebben ze gelijkaardige eigenschappen als degene die tensor product B–splines zo succesvol hebben gemaakt, en op dat vlak scoren ze beter dan Bernstein–Bézier veeltermen over driehoeken:

   (a) associatie met controledriehoeken die rakend zijn aan het oppervlak,

   (b) affiene invariantie,

   (c) lokale controle,

   (d) de convex omhullende eigenschap,

   (e) globale C^1–continuïteit ongeacht de keuze van de coëfficiënten.

4. De B–spline basis kan gevonden worden door het oplossen van een aantal kleinschalige kwadratische programmeringsproblemen.

5. PS–splines kunnen gevonden worden als oplossing van een interpolatieprobleem waar functie– en afgeleide waarden gegeven zijn. Dit laat toe om gekende vormen te benaderen door PS–spline oppervlakken.

6. De Bernstein–Bézier voorstelling kan efficiënt en op numeriek stabiele wijze berekend worden. Daardoor kunnen we steunen op de rijke theorie rond Bernstein–Bézier veeltermen, zoals het de Casteljau algoritme.

7.2 De resultaten van het onderzoek

NURPS: een uitbreiding van PS–spline oppervlakken. We hebben PS–spline oppervlakken veralgemeend naar hun rationale vorm, de zogenaamde NURPS oppervlakken. Deze voorstelling is geschikt voor CAGD. We hebben aangetoond dat alle algoritmen voor PS–splines die uitsluitend convexe combinaties gebruiken, veralgemeend kunnen worden naar een numeriek stabiel algoritme voor NURPS oppervlakken. Dit werd geïllustreerd voor de berekening van de rationale Bernstein–Bézier voorstelling, en steunt op het idee achter het rationaal de Casteljau algoritme. Voor subdivisie van rationale Bernstein–Bézier oppervlakken, hebben we een alternatief schema gegeven dat beter geschikt is voor de grafische weergave. We hebben onderzocht hoe de gewichten die met elk controlepunt geassocieerd worden in de NURPS voorstelling, bijkomende mogelijkheden bieden voor vrije vorm ontwerpen. Een gebruiker kan de gewichten op een transparante manier manipuleren door het concept van vormparameters, iets dat niet mogelijk is voor rationale Bernstein–Bézier oppervlakken. Verder hebben we het modelleren van speciale effecten behandeld: pieken, rechte randen en gekromde randen. Ook de voorstelling van kwadrieken als NURPS kwam aan bod, en we hebben expliciete formules gegeven voor het geval van een cilinder, kegel en bol. Deze laatste kan niet volledig voorgesteld worden als rationaal kwadratisch oppervlak. Daarom geven we een onvolledige voorstelling, waarbij de gebruiker controle heeft over de grootte van de openingen die zich voordoen rond het meridiaanvlak.

UPS–splines: een speciaal geval van PS–splines. In het geval dat de driehoeksverdeling uniform is opgebouwd uit gelijkzijdige driehoeken, zijn er een aantal voordelen ten opzichte van algemene PS–splines:

1. Een PS–verfijning kan onmiddellijk gevonden worden.
2. De vorm van de PS–driehoeken kan op voorhand vastgelegd worden. Het oplossen van een kwadratisch programmeringsprobleem wordt zodoende vermeden.
3. De B–spline basis is op voorhand gekend.

Dit leidt tot efficiënte berekeningen met UPS–splines voor o.a. het interpolatieprobleem en het berekenen van de Bernstein–Bézier spline voorstelling. We hebben ook een subdivisieschema afgeleid dat toelaat een gegeven UPS–spline voor te stellen over een verfijnde driehoeksverdeling, door convexe combinaties te nemen van de oorspronkelijke coëfficiënten. Dit werd veralgemeend naar k–subdivisie. We hebben ook twee toepassingen van subdivisie gegeven: de UPS–spline wavelet transformatie en de visualisatie van UPS–spline oppervlakken.

De UPS–spline wavelet transformatie is een nieuwe wavelet transformatie op een regelmatig driehoekrooster. De observatie dat de basisfuncties translaties en dilataties zijn van slechts drie functies, gaf aanleiding tot de definitie van een multiresolutie–analyse. Dan hebben we een lifting schema opgesteld dat een gegeven UPS–spline opsplitst in een laagfrequente en een aantal hoogfrequente componenten (de details). Als predictie–stap namen we subdivisie, als bijsturing gaven we een eenvoudige oplossing die het gemiddelde over de laagfrequente componente bewaart. De basisfuncties bij de detailsignalen zijn de UPS–spline wavelets. Ze combineren de voordelen van B–spline wavelets, multi–wavelets en niet–separabele wavelets. We hebben een voorstel gedaan van hoe de multi–wavelet aspecten verder kunnen gebruikt worden in een complexere bijsturings–stap. Als toepassing hebben we ook een drempel algoritme gegeven voor het verwijderen van ruis uit een UPS–spline oppervlak.

De grafische weergave van NURPS oppervlakken gebeurt door het berekenen van de rationale Bernstein–Bézier voorstelling, gevolgd door ons alternatief subdivisieschema. Zodoende wordt een groot aantal punten op het oppervlak berekend. We hebben ook de grafische weergave van UPS–splines bestudeerd. Steunende op subdivisie, kan een aantal punten op het oppervlak gevonden worden, net als de normalen in die punten. Dit laat toe het oppervlak weer te geven als draadmodel of als belicht model. Het algoritme kan uitgebreid worden naar rationale UPS–spline oppervlakken. We hebben een blok matrix structuur gebruikt als gegevensvoorstelling, en aangetoond dat deze aanleiding geeft tot een zeer efficiënte visualisatie in OpenGL, door het terugdringen van het aantal functie oproepen, en het verminderen van de redundante verwerking van hoekpunten.

Benadering en conversie. We hebben een oplossing gegeven voor het benaderen van een gegeven NURBS oppervlak met een UPS–spline oppervlak. Daarnaast zijn er ook een aantal conversieschema's aan bod gekomen:

1. De omzetting van een NURPS oppervlak naar een rationaal Bernstein–Bézier oppervlak, en omgekeerd.

2. Het vervangen van een PS–driehoek door een andere driehoek die de PS–punten bevat. Dit laat bijvoorbeeld toe om de uniforme voorstelling te berekenen van een optimaal NURPS oppervlak.

Het probleem van de veelhoekige opening. We hebben een algemeen toepasbaar algoritme opgesteld voor de berekening van een oppervlak dat een opening opvult, begrensd door een aantal gegeven oppervlakken. De randcurven worden benaderd, hetgeen zinvol is voor praktische toepassingen. De gebruiker voorziet routines voor het berekenen van punten op de randcurven, alsook raakvectoren en normalen, telkens geparametrizeerd op het eenheidsinterval. Een initiële oplossing wordt in een aantal stappen verfijnd langsheen de rand, zodanig dat de gegevens in een aantal punten geïnterpoleerd worden, en zo goed mogelijk benaderd in tussenliggende punten. Een aantal mogelijkheden voor het berekenen van het inwendige van het oppervlak werden onderzocht, steunende op o.a. subdivisie, k–subdivisie en stroomlijnen. Ze werden vergeleken aan de hand van de Gaussiaanse kromming. Uit onze experimenten bleek dat een algemeen toepasbare methode, die in elke situatie bevredigende resultaten geeft, moeilijk te vinden is. De methodes van "onmiddellijk bijsturen" en "stroomlijnen" dragen onze voorkeur weg, en de keuze is afhankelijk van wat de gebruiker verwacht. Als testvoorbeeld hebben we de opening in de NURPS voorstelling van de bol opgevuld, en hebben we ontaarde NURBS oppervlakken in een CAD model van een autospiegel vervangen door driehoekige UPS–spline oppervlakken.

Stroomlijnen. We bespraken een maat voor het vloeiend zijn van UPS–spline oppervlakken, en gebruikten deze om een probleem van stroomlijning onder beperkingen op te lossen: gegeven een UPS–spline oppervlak, zoek de inwendige controledriehoeken zodat het oppervlak vloeiend is, terwijl de controledriehoeken aan de rand ongewijzigd blijven.

PS–surf is een CAD programma dat de resultaten uit het proefschrift illustreert, gebruik makend van ons visualisatie algoritme voor UPS–splines, en technisch gesproken gebruik maakt van `Visual C++`, MFC en OpenGL. NURBS oppervlakken kunnen ingelezen worden uit een IGES bestand, en worden gevisualiseerd. PS–surf bestaat uit een hoofdprogramma (een menu–gebaseerde Windows toepassing) en een aantal object–georiënteerde modules. De voordelen van overerving werden geïllustreerd, bijvoorbeeld door het gebruik van een krachtige basis klasse, en het integreren van nieuwe CAD toepassingen. PS–surf is klaar voor de uitbreiding naar algemene PS–spline oppervlakken, en we hebben daartoe al enige suggesties gemaakt omtrent datastructuren en visualisatie. We hebben PS–surf gebruikt voor het modelleren van enkele realistische voorbeelden, een autospiegel en een autozetel. Zodoende konden we ontaarde NURBS oppervlakken vermijden, en illustreren dat het mogelijk is om complexe vormen voor te stellen als UPS–spline oppervlakken.

7.3 Toekomstig onderzoek

1. Het is gekend dat parametrische continuïteit vaak te streng is om voorwaarden voor het vloeiend verloop van oppervlakken op te leggen. Voor B–spline curven bijvoorbeeld, worden deze voorwaarden afgezwakt tot geometrische continuïteit. Dit levert zelfs enkele extra vrijheidsgraden. Men spreekt van de β–spline voorstelling. Het is een open vraag of dit concept toepasbaar is op PS–spline oppervlakken.

2. Verwijzend naar het drempel algoritme, zou het nuttig zijn om te beschikken over een manier om een geschikte drempelwaarde te vinden.

3. De bijsturings–stap van ons lifting schema is eenvoudig gehouden. Nochthans kunnen omwille van het multi–wavelet aspect, complexere ontwerpen gevonden worden die toelaten meerdere eigenschappen op te leggen aan de wavelets. We hebben al enkele suggesties gegeven, maar het zou nuttig zijn te weten wat de mogelijkheden in dit verband zijn, en met welk nut naar praktische toepassingen toe.

4. Ons algoritme voor het probleem van de veelhoekige opening berekent een oppervlak dat de gegeven curven in een aantal punten interpoleert. De controledriehoeken in deze punten hebben twee vrijheidsgraden: de α–factoren en de β–factoren. Het zijn de eerste die bepalen hoe goed de randcurven benaderd worden, terwijl de tweede eerder een invloed hebben naar het inwendige van het oppervlak toe. Het is interessant om te onderzoeken of de β–factoren een rol kunnen spelen in het bereiken van een vloeiend inwendige van het oppervlak.

5. Een andere uitbreiding van dit algoritme is het gebruik van rationale UPS–spline oppervlakken. Het lijkt een interessante, maar zeker niet–triviale uitdaging om de gewichten in dit verband te gebruiken.

6. Ook het gebruik van convexiteitsvoorwaarden [82] voor het berekenen van het inwendige is een interessante mogelijkheid.

7. De meest voor de hand liggende, maar zeker niet de minst uitdagende mogelijkheden voor toekomstig onderzoek liggen in het uitbreiden van onze resultaten voor UPS–splines naar algemene PS–splines. In verband hiermee vermelden we dat, na een vruchtbare samenwerking,

Vanraes et al. een subdivisieschema voor PS–splines over algemene driehoeksverdelingen hebben afgeleid [77]. In het licht van de resultaten van dit proefschrift, zijn er daarbij heel wat mogelijkheden voor verder onderzoek:

(a) De grafische weergave van PS–spline oppervlakken kan efficiënter geïmplementeerd worden dan via ons subdivisieschema voor rationale Bernstein–Bézier oppervlakken. De regelmatigheid die in de driehoeksverdeling ontstaat na enkele subdivisiestappen moet daarbij weerspiegeld worden in de gegevensstructuur, om de performantie in OpenGL zo hoog mogelijk te houden door het vermijden van redundante verwerking van hoekpunten, en het verminderen van het aantal functie aanroepen.

(b) Subdivisie voor PS–splines kan gebruikt worden als predictie–stap in een algemene PS–spline wavelet transformatie. Deze zou aanleiding geven tot niet–separabele B–spline multi–wavelets van de tweede generatie. Ook de toepasbaarheid hiervan in klassieke domeinen als compressie en ruisverwijdering kan onderzocht worden.

(c) Willemans [82] bestudeerde benaderingsproblemen met Powell–Sabin splines onder beperkingen. Het zou interessant zijn om deze resultaten te combineren met subdivisietechnieken. Bijvoorbeeld, een iteratief algoritme waar een PS–spline oppervlak dat een gegeven stel meetpunten benadert, steunend op subdivisie, zou erg nuttig zijn. Als er ergens een molecule M_i ontstaat zonder meetpunten, kan de controledriehoek T_i gevonden worden uit subdivisie. Bovendien heeft het oppervlak dan een structuur die wavelet–transformeerbaar is, hetgeen de deur opent naar verdere toepassingen zoals ruisverwijdering via een drempel algoritme.

We geloven dat PS–splines een toekomst hebben in CAGD, en hopen dat dit proefschrift een eerste stap is in de fundamentele en praktische uitwerking hiervan.

Notations

Lower Case Greek Symbols

$\tilde{\alpha}_{\boldsymbol{b}_\lambda}$	real value, also interpreted as barycentric coordinate
α_i	scaling factor at V_i
$\tilde{\alpha}_{i,j}$	real value, also interpreted as barycentric coordinate
$(\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j})$	function and partial derivative values of $B_i^j(u)$ at V_i
$(\tilde{\alpha}_{i,1}, \tilde{\alpha}_{i,2}, \tilde{\alpha}_{i,3})$	shape parameters for NURPS
$\tilde{\beta}_{\boldsymbol{b}_\lambda}$	real value, also interpreted as barycentric coordinate
β_i	scaling factor at V_i
$\beta_{i,j}$	scaling factor at $V_{i,j}$
$\vec{\gamma}$	unit tangent vector
$\vec{\gamma_i}$	unit tangent vector at V_i
$\vec{\delta}$	unit cross–boundary tangent vector
$\vec{\delta_i}$	unit cross–boundary tangent vector at V_i
ϵ_l^k	real value, a threshold
θ_k	Bézier ordinate
$\boldsymbol{\theta}_k$	Bézier control point
$\tilde{\boldsymbol{\theta}}_k$	homogeneous Bézier control point
κ_1, κ_2	principle curvatures
κ_{abs}	absolute curvature
κ_{norm}	normalized curvature
$\lambda = (\lambda_1, \lambda_2, \lambda_3)$	multi–index
$\frac{\lambda}{m}$	Bézier triangle points
$\lambda_{i,j}$	real value, also interpreted as barycentric coordinate
$\tilde{\lambda}_{i,j}$	real value, also interpreted as barycentric coordinate
$\xi_i^j(\tau)$	rational basis function
$\xi_\lambda(u)$	rational basis function
$\rho_{i,j,k}$	shorthand for $\rho(V_i, V_j, V_k)$
ρ_j	triangle of a triangulation
ρ_j^*	triangle of the PS–refinement of a triangulation
$\rho_{i,j,k}$	triangle of Δ having vertices V_i, V_j and V_k

$\rho(A, B, C)$	triangle with vertices A, B and C
$\rho(V_i, V_j, V_k)$	triangle with given vertices
$\sigma = (\sigma_1, \sigma_2, \sigma_3)$	barycentric coordinates
$\tilde{\tau}_i$	real value, also interpreted as barycentric coordinate
$\tau = (\tau_1, \tau_2, \tau_3)$	barycentric coordinates
$\psi_{i;j}(u)$	wavelet
ω	Bézier ordinate
$\boldsymbol{\omega}$	Bézier control point
$\tilde{\boldsymbol{\omega}}$	homogeneous Bézier control point

Upper Case Greek Symbols

Γ	matrix
Δ	conforming triangulation
Δ^*	Powell–Sabin refinement of a triangulation
$\tilde{\Delta}$	extended triangulation
Θ	affine transformation
Π_m	space of bivariate polynomials of total degree $\leq m$
$\Phi(u)$	multi–scaling function
$\Phi^j(u)$	dilates of $\Phi(u)$
$\Phi^j_k(u)$	translates and dilates of $\Phi(u)$
$\Psi_i(u)$	multi–wavelet function
$\Psi_{i;j,k}(u)$	translates and dilates of $\Psi_i(u)$
Ω	polygon

Lower Case Arabic Symbols

a_i	Bézier ordinate
$\tilde{a}_i$	real value, also interpreted as barycentric coordinate
$\tilde{\boldsymbol{a}}_i$	homogeneous Bézier control point
$\boldsymbol{b}_i$	Bézier control point
$\tilde{\boldsymbol{b}}_i$	homogeneous Bézier control point
b_λ	Bézier ordinate
$\boldsymbol{b}_\lambda(\frac{\lambda}{m}, b_\lambda)$	Bézier control point
$\boldsymbol{b}_\lambda(b^x_\lambda, b^y_\lambda, b^z_\lambda)$	Bézier control point
b^r_λ	intermediate de Casteljau ordinate
$\boldsymbol{b}^r_\lambda$	intermediate de Casteljau point
$\tilde{\boldsymbol{b}}(w_\lambda \boldsymbol{b}_\lambda, w_\lambda)$	homogeneous Bézier control point
$\tilde{\boldsymbol{b}}_\lambda$	homogeneous Bézier control point
$b(\tau)$	Bernstein–Bézier polynomial
$\boldsymbol{b}(\tau)$	Bernstein–Bézier surface
$\tilde{\boldsymbol{b}}(\tau)$	homogeneous rational Bernstein–Bézier surface

$\tilde{\boldsymbol{b}}_{\mathcal{T}}(\tau)$	homogeneous rational Bernstein–Bézier surface on $\mathcal{T}$
$b_{\mathcal{T}}(\tau)$	Bernstein–Bézier polynomial on $\mathcal{T}$
$\boldsymbol{b}_{\mathcal{T}}(\tau)$	Bernstein–Bézier surface on $\mathcal{T}$
c	real value
$\boldsymbol{c}_i$	B–spline control point
$c_{i,j}$	B–spline coefficient
$\boldsymbol{c}_{i,j}$	B–spline control point
$\boldsymbol{c}_{i,j,k}$	B–spline control point
$\boldsymbol{c}_{i,j}(X_{i,j}, Y_{i,j}, c_{i,j})$	B–spline control point
$\boldsymbol{c}_{i,j}(c^x_{i,j}, c^y_{i,j}, c^z_{i,j})$	B–spline control point
$\tilde{\boldsymbol{c}}_{i,j}$	homogeneous B–spline control point
$d = (d_1, d_2, d_3)$	barycentric direction
$\tilde{d}_{i,j}$	real value
$\tilde{\boldsymbol{d}}_\lambda$	homogeneous Bézier control point
$\boldsymbol{d}^r_\lambda$	intermediate de Casteljau point
d_x	unit barycentric direction in x–direction
d_y	unit barycentric direction in y–direction
$\boldsymbol{e}_{i,j}(u)$	boundary Bézier curve
$\tilde{e}_{i,j}$	real value
$f(\tau), f(\tau_1, \tau_2, \tau_3)$	bivariate function over a triangle
f_i	function value at V_i
$f_{x,i}$	value of partial derivate wrt x at V_i
$f_{y,i}$	value of partial derivate wrt y at V_i
j_3	j mod 3
k_i	vector in $\mathcal{K}$
$k_{j,l}$	real value
m	degree of a polynomial
m_i	molecule number of vertex V_i
n	number of vertices of a triangulation
$\vec{n}$	unit normal vector
$\boldsymbol{n}(\tau)$	normal vector to $\boldsymbol{b}(\tau)$ at τ
$\boldsymbol{n}(u)$	normal vector to $\boldsymbol{s}(u)$ at u
$\boldsymbol{p}$	user supplied boundary curve
$p(u)$	a polynomial
$\boldsymbol{p}(u)$	a polynomial surface
$\boldsymbol{p}_i$	Bézier control point
$(\boldsymbol{p}_i, \vec{\gamma}_i, \vec{\delta}_i)$	user supplied values for $(\boldsymbol{p}, \vec{\gamma}, \vec{\delta})$ at V_i
$\boldsymbol{p}_{i,j}$	$\boldsymbol{p}$ evaluated at $V_{i,j}$
$\boldsymbol{q}_{i,j}$	shorthand notation
r	radius
r_k	Bézier ordinate

$\boldsymbol{r}_k$	Bézier control point
$\tilde{\boldsymbol{r}}_k$	homogeneous Bézier control point
$r_{i,k}$	residual
$\boldsymbol{r}_{i,j}$	Bézier control point
s_i	Bézier ordinate
$\boldsymbol{s}_i$	Bézier control point
$\tilde{\boldsymbol{s}}_i$	homogeneous Bézier control point
$s(u)$	a Powell–Sabin spline
$s'(u)$	a Powell–Sabin spline
$s(u)\mid_A$	restriction of of $s(u)$ to $A \subset \Omega$
$\boldsymbol{s}(u)$	PS–spline surface
$\tilde{\boldsymbol{s}}(u)$	rational PS–spline surface in homogeneous space
$\boldsymbol{s}^f(u)$	PS–spline surface at resolution level f
$s_x(u)$	x–component of parametric PS–spline surface
$s_y(u)$	y–component of parametric PS–spline surface
$s_z(u)$	z–component of parametric PS–spline surface
t	number of triangles of a triangulation
t^*	number of triangles of the PS–refinement of a triangulation
$t_i(Q_{i,1}, Q_{i,2}, Q_{i,3})$	PS–triangle corresponding to vertex V_i
$\boldsymbol{t}_i$	Bézier control point
$\tilde{\boldsymbol{t}}_i$	homogeneous Bézier control point
u_i	Bézier ordinate
$\boldsymbol{u}_i$	Bézier control point
$\tilde{\boldsymbol{u}}_i$	homogeneous Bézier control point
$u(u_1, u_2)$	a point in the plane
$u(x, y)$	a point in the plane
v_i	Bézier ordinate
$\boldsymbol{v}_i$	Bézier control point
$\tilde{\boldsymbol{v}}_i$	homogeneous Bézier control point
$v(x, y)$	a point in the plane
w_i	Bézier ordinate
w_λ	weight of a Bézier control point $\boldsymbol{b}_\lambda$
$w_{\boldsymbol{b}_\lambda}$	weight of a Bézier control point $\boldsymbol{b}_\lambda$
w_λ^r	intermediate de Casteljau weight
$\boldsymbol{w}_i$	Bézier control point
$w_{i,j}$	weight of a control point $\boldsymbol{c}_{i,j}$
$\tilde{\boldsymbol{w}}_i$	homogeneous Bézier control point
$w(\tau)$	homogeneous component of a rational Bernstein–Bézier surface
(x_i, y_i)	Cartesian coordinates of a vertex V_i

Upper Case Arabic Symbols

B	base of a uniform triangulation
$B_{i,j}$	submatrix of a block matrix M
$B_i^j(u)$	the jth B–spline basis function corresponding to V_i
$B_\lambda^m(\tau)$	Bernstein–Bézier polynomial of total degree m
$\hat{B}_\lambda^m(\tau)$	Bernstein–Bézier polynomial of total degree m
C^r	the space of r times continuously differentiable functions
$C_{L,k}$	scaling coefficient matrix
D_d^r	rth directional derivative with respect to d
I	unit matrix
$L_{i,j}$	real value, also interpreted as barycentric coordinate
$L'_{i,j}$	real value, also interpreted as barycentric coordinate
$L''_{i,j}$	real value, also interpreted as barycentric coordinate
$\tilde{L}_{i,j}$	real value, also interpreted as barycentric coordinate
$\tilde{L}'_{i,j}$	real value, also interpreted as barycentric coordinate
$\tilde{L}''_{i,j}$	real value, also interpreted as barycentric coordinate
$L_2(\mathbb{R}^2)$	space of square integrable functions
M_i	molecule of vertex V_i
$P_i(s)$	prediction on branch $D\Delta + k_i$
Q_i	Powell–Sabin triangle point
$Q_{i,j,k}$	Powell–Sabin triangle point
$Q_{i,j}(X_{i,j}, Y_{i,j})$	Powell–Sabin triangle point
$R_{j,l}$	Powell–Sabin refinement point on edge common to ρ_j and ρ_l
$S_2^1(\Delta^*)$	space of Powell–Sabin splines
$\boldsymbol{S}(\bar{u}, \bar{v})$	NURBS surface
$T_i(\boldsymbol{c}_{i,1}, \boldsymbol{c}_{i,2}, \boldsymbol{c}_{i,3})$	control triangle corresponding to vertex V_i
$V_i(x_i, y_i)$	vertex
$V_{i,j}$	the midpoint of the line segment between V_i and V_j
$X_{i,j}$	B–spline coefficient of $s(u) = x$
$(X_{i,j}, Y_{i,j})$	Cartesian coordinates of $Q_{i,j}$
$(X_{i,j}, Y_{i,j}, c_{i,j})$	Cartesian coordinates of B–spline control point
$Y_{i,j}$	B–spline coefficient of $s(u) = y$
$W_{i;j,k}$	wavelet coefficient matrix
$\boldsymbol{W}^k(W_x^k, W_y^k, W_z^k)$	series of wavelet coefficients
Z	interior point of a triangle ρ
Z_j	interior point of a triangle ρ_j

Calligrafic Symbols

$\mathcal{B}$	Bézier control net
$\mathcal{K} = \Gamma B \mathbb{R}^2$	uniform triangular grid
$\mathcal{T}$	a triangle
$\hat{\mathcal{T}}$	a triangle
$\mathcal{T}(a, b, c)$	Bézier subtriangle of a PS–refinement
$\mathcal{V}_j$	space spanned by $\Phi_k^j(u)$
$\mathcal{W}_{i,j}$	space spanned by $\Psi_{i;j,k}(u)$

Other Symbols

$[\ldots]$	convex hull
$\|\|.\|\|$	Euclidean norm
$(\cdot)$	dot product
$\|\lambda\|$	sum of elements of λ
$\|D\|$	determinant of a matrix D
$A(\mathcal{T})$	area of the triangle $\mathcal{T}$
$\delta c_{i,j}$	noise
$\delta\Delta$	vertices on the edge of a triangulation Δ
$\delta\Omega$	boundary of a polygon Ω
$D\mathcal{K} + k_i$	ith subgrid of $\mathcal{K}$
$LP(s)$	low frequency component of s
$HP(s)$	high frequency component of s
$\mathbb{R}^n$	space of real numbers

Abbreviations

CAD	Computer Aided Design
CAGD	Computer Aided Geometric Design
DC	Direct Current
GCV	Generalized Cross Validation
PS	Powell–Sabin
RE	Reverse Engineering
UPS	Uniform Powell–Sabin

Contents

Chapter 1

Introduction

The ability of representing complex shapes in a way that can be used by computers, first came into attention in the late 1950s, when hardware became available for the machining of 3D shapes out of wood or steel. The pioneers in this area, P. Bézier (at Citroën), P. de Casteljau (at Renault), and S. Coons (who consulted for Ford), achieved the major breakthroughs of Computer Aided Geometric Design (CAGD) by the development of Bézier surfaces (independently discovered by Bézier and de Casteljau) and Coons patches, in the early 1960s. Ten years later, CAGD had become a discipline in its own right, and today it is an integral part of the development cycle of many products in a CAD/CAM environment.

An essential aspect of CAGD is the use of an appropriate mathematical representation for curves and surfaces. It has to allow to represent complex shapes accurately, but it must also enable a designer, most often a non–mathematician, to modify the design in a predictable, intuitive way. Usually, a surface is represented as

$$\boldsymbol{s}(u) = \sum_i \boldsymbol{c}_i \phi_i(u), \quad u = (x, y) \in \Omega, \quad \boldsymbol{c}_i \in \mathbb{R}^3, \tag{1.1}$$

where Ω is the parameter domain, and the basis functions $\phi_i(u)$ are (piecewise) polynomials whose degree is chosen not too high. A number of conditions are imposed on the basis functions. The most common ones are:

convex partition of unity

$$\begin{array}{rcl} \phi_i(u) & \geq & 0 \\ \sum_i \phi_i(u) & \equiv & 1, \end{array} \tag{1.2}$$

local support

$$\phi_i(u) = 0, \quad u \notin M_i \subset \Omega, \tag{1.3}$$

where M_i is a small, simply connected subset of Ω,

parametric continuity

$$\phi_i(u) \in C^r[\Omega], \tag{1.4}$$

linear precision

$$\begin{aligned} x &= \textstyle\sum_i X_i \phi_i(u) \\ y &= \textstyle\sum_i Y_i \phi_i(u), \end{aligned} \tag{1.5}$$

with $X_i, Y_i \in \mathbb{R}$.

These conditions imply a number of properties that make representations of the form (1.1) such powerful tools in CAGD.

Control points. The coefficients $\boldsymbol{c}_i$ have a geometric interpretation: they are control points that determine the shape of the surface. If one connects the control points by straight lines, the control net appears, which mimics the shape of the underlying surface. So the control net gives a rough idea of how the surface will look, and the control points can be placed with a specific shape in mind.

Convex hull property. Each point on the surface belongs to the convex hull of the control points. This is important for determining quickly whether surfaces intersect or not.

Local control. By moving a control point to a new position, the surface will only be modified in the neighbourhood of that point, and elsewhere remains untouched.

Continuity. Regardless of the choice of the control points, $\boldsymbol{s}(u)$ has global continuity of degree r.

Affine invariance. If an affine transformation is to be performed on $\boldsymbol{s}(u)$, it can be applied to the control points only.

The aforementioned Bézier representation of surfaces on rectangular domains satisfies (1.2) and (1.5), but doesn't fulfill (1.3). For representing complex shapes with low degree polynomials, a large number of Bézier surfaces have to be joined, and this makes it hard to implement global continuity conditions. This is not the case for the more recently developed B–spline representation. It satisfies (1.2)–(1.5), and has all the attractive properties that we have mentioned. Therefore, tensor product B–spline surfaces and their rational extension to NURBS surfaces are today the most commonly used surface types in commercial CAD and computer graphics applications.

However, a drawback is that they are restricted to four–sided domains Ω, while in practice, many 3D models require the use of surface patches with, e.g., three edges. Moreover, the knots split up Ω into a rectangular grid, which makes an adequate adaptive refinement not obvious.

As an alternative, Farin [30] studied the Bernstein–Bézier representation of piecewise polynomials on triangulations of polygonal domains. His theory resolves the problem of the domain shape restriction, but has the same drawbacks as mentioned before with respect to rectangular Bézier surfaces. A number of authors (Davydov [15], Lai [46, 47], Schumaker [62, 63], Wang [78]) studied the construction of local bases for polynomial spline spaces on arbitrary triangulations. One of the difficulties is to determine the dimension of these spline spaces. For example, this dimension can not in general be expressed in terms of the number of vertices and triangles of the triangulation because it also depends on its topology and geometry. The use of so–called "split" triangulations can remedy this problem. In this work, we shall use the Powell–Sabin split. It allows to define the space of piecewise quadratic polynomials with global C^1 continuity on a refined triangulation. The dimension of this space is easy to determine [57]. Moreover, Dierckx [19] has shown how a normalized B–spline basis satisfying (1.2)–(1.5) can be constructed. We call this space the space of Powell–Sabin (PS–)splines. In this thesis, we study the use of PS–splines in their B–spline representation in CAGD. The Bernstein–Bézier representation of polynomials on triangles will thereby prove to be a useful tool, because it underlies the B–spline representation, and it is theoretically well elaborated.

In Chapter 2 we recall the fundamental theory of Bernstein–Bézier polynomials on triangles and of PS–splines. We review basic concepts such as barycentric coordinates, polynomials on triangles, and we discuss the de Casteljau algorithm and its applications. We also consider Bernstein–Bézier surfaces and show how to calculate with them. Then we define the space of PS–splines, review the construction of a normalized local B–spline basis, and show how control triangles can be associated with the B–spline coefficients. We investigate the differentiation and integration of PS–splines and show how to derive their Bernstein–Bézier representation. We also discuss calculating with PS–spline surfaces.

Chapter 3 extends PS–spline surfaces to the rational form, called NURPS surfaces. Again, we first review the rational Bernstein–Bézier form, with the rational de Casteljau algorithm as the main calculation tool, and then we generalize PS–spline surfaces to NURPS surfaces by assigning an extra weight to each control point. We show that the NURPS representation satisfies (1.2)–(1.5) as well, and extend the conversion scheme to the Bernstein–Bézier representation to its rational form. We demonstrate that the weights give the user extra flexibility for modeling special effects, and that NURPS allow to represent quadric surfaces exactly. This is investigated for the cylinder, cone and sphere.

In Chapter 4 we study a particular case of PS–splines, those on uniform triangulations. We call them UPS–splines. Though this puts a restriction on the domain polygon, there are a number of advantages: the basis is known beforehand, calculation schemes for e.g. integration of UPS–splines can be implemented more efficiently, and a subdivision scheme is presented. The latter is used to the derive non–separable B–spline multi–wavelets on triangular grids, using lifting, but also, more practically, for the efficient graphical display of UPS–spline surfaces.

In Chapter 5 we give a solution to a common, but difficult problem in CAGD, the polygonal hole problem. A UPS–spline surface is calculated iteratively, that fills in a hole, given by a set of surrounding surfaces. We use subdivision techniques to determine a solution that interpolates the given data in a number of points, and fits them in between the interpolation points.

Chapter 6 describes a software prototype, called PS–surf, that has been developed to illustrate the results of the preceding chapters. We show how UPS–spline surfaces can be rendered efficiently using OpenGL, and present a number of realistic, complex CAD models using UPS–spline surfaces.

Finally we give our conclusions and suggestions for further research in Chapter 7.

Chapter 2

Powell–Sabin splines

2.1 Introduction

In this chapter we give the basic theory of piecewise quadratic polynomials on triangulations. Our aim is not to be complete, nor to prove everything rigorously. We will merely settle notation and cover the necessary background needed in further chapters. This chapter consists of two parts.

In Section 2.2 we review the Bernstein–Bézier representation of polynomials on the triangle [29]. We will show how the properties of the underlying Bézier basis allow us to associate a control net with a Bernstein–Bézier polynomial, giving us insight in the shape of the corresponding Bernstein–Bézier surface. A central position is taken by the de Casteljau algorithm (Section 2.2.5): not only yields it the solution for the evaluation of Bernstein–Bézier polynomials; also the calculation of derivatives, the subdivision problem, and expressing continuity conditions between Bernstein–Bézier polynomials on adjacent triangles are applications of this powerful algorithm. Moreover, it can be interpreted geometrically using the concept of intermediate de Casteljau points. We will also have a look at parametric Bernstein–Bézier surfaces (Section 2.2.6) and consider the calculation of surface points, tangent planes and normal vectors in Section 2.2.7.

Although Bernstein–Bézier polynomials provide a nice theoretical framework for dealing with piecewise polynomials on triangulations, there are two main disadvantages that hinder their practical application in CAGD: first, there is no apparent local control. Instead, similar to Bézier curves, one can speak of pseudo–local control. Second, continuity conditions between Bernstein–Bézier polynomials on adjacent triangles result in nontrivial relations between their coefficients, that are not only hard to implement but also make it difficult for a designer to make predictable local changes. Therefore, we will look for piecewise polynomials on triangulations with global continuity properties that are suitable for CAGD.

The next part, Section 2.3, is devoted to Powell–Sabin (PS–)splines and surfaces [19, 20]. In Section 2.3.1, we show how an interpolation problem, where function value and derivative information is given at a number of vertices of an arbitrary triangulation, can be solved. If the given triangulation is refined according to the Powell–Sabin split [57], then the solution of the interpolation is unique within the space of piecewise quadratic polynomials with global C^1 continuity; it is a Powell–Sabin spline. We will look for a locally supported normalized B–spline basis for PS–splines and show why it is useful in CAGD applications (Section 2.3.2). The concept of control triangles allows us to modify a PS–spline surface locally, and gives us insight in its shape. The algorithm for constructing a basis also has a geometric interpretation: find a triangle for each vertex, that contains a particular number of points. In Section 2.3.3, the evaluation, differentiation and integration of PS–splines is considered, and an algorithm for calculating the Bernstein–Bézier representation is given. The extension to parametric PS–spline surfaces is made in Section 2.3.4, and finally Section 2.3.5 shows how to calculate a surface point, a tangent plane and a normal vector to a PS–spline surface.

2.2 Bernstein–Bézier polynomials and Bernstein–Bézier surfaces

This section brings together the properties of Bernstein–Bézier polynomials and –surfaces on triangles. Since we are mainly interested in quadratic polynomials, the examples and figures in this section always refer to this particular case. Unless stated otherwise, the proofs and further details can be found in the comprehensive work of Farin [29].

2.2.1 Barycentric coordinates and affine transformations

A point u given in its Cartesian coordinates $(x, y) \in \mathbb{R}^2$, is denoted $u(x, y)$.

Definition 2.1 *Let $\mathcal{T}$ be a non–degenerate triangle with vertices V_i having Cartesian coordinates $(x_i, y_i), i = 1, 2, 3$. Any point $u(x, y)$ can then be uniquely represented by its barycentric coordinates $\tau = (\tau_1, \tau_2, \tau_3)$ with* *respect to $\mathcal{T}$, where $\tau_i, i = 1,2,3$ are the solution of the system*

$$\begin{aligned} x_1\tau_1 + x_2\tau_2 + x_3\tau_3 &= x \\ y_1\tau_1 + y_2\tau_2 + y_3\tau_3 &= y \\ \tau_1 + \tau_2 + \tau_3 &= 1. \end{aligned} \tag{2.1}$$

We will sometimes write $(x, y) = (\tau_1, \tau_2, \tau_3)$ when (τ_1, τ_2, τ_3) are the barycentric coordinates of the point $u(x, y)$ with respect to $\mathcal{T}$. If the point $u(x, y)$ lies within $\mathcal{T}$, then the barycentric coordinates are nonnegative. The barycentric coordinates of the vertices V_1, V_2, V_3 are $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$ respectively.

Definition 2.2 *Any transformation preserving collinearity and ratios of distances is an affine transformation. An affine transformation has the form*

$$\Theta : \mathbb{R}^3 \to \mathbb{R}^3 : \Theta u = Mu + N, \tag{2.2}$$

where $M \in \mathbb{R}^{3\times 3}$ and $N \in \mathbb{R}^3$.

Reflection, rotation, translation, scaling, shear,... are all affine transformations. If $M^T M = I$, the transformation is called a Euclidean map and leaves lengths and angles unchanged. If $\operatorname{rank}(M) = 2$, then Θ is a parallel projection onto a plane. If $\operatorname{rank}(M) = 1$, then it is a projection onto a straight line. Barycentric coordinates have the property of affine invariance:

Property 2.1 *Let Θ be an affine transformation, and let $u = \sum_{i=1}^3 \tau_i V_i$, then the transformed point Θu has barycentric coordinates (τ_1, τ_2, τ_3) with respect to the transformed triangle $\Theta\mathcal{T}$, i.e.*

$$\Theta u = \sum_{i=1}^{3} \tau_i \Theta V_i. \tag{2.3}$$

In the following we shall deal with bivariate functions over triangles. We shall write these in terms of barycentric coordinates, e.g., $f(\tau_1, \tau_2, \tau_3)$, which is a function of three dependent variables. This raises the problem of how to handle differentiation: the terms $\frac{\partial f}{\partial \tau_i}$ have no geometric interpretation. Therefore, one resorts to directional derivatives.

Definition 2.3 *Let τ and σ be barycentric coordinates of two arbitrary points u and v, then their difference d defines a barycentric direction. Besides, d is called a unit barycentric direction if the Euclidean norm $||u - v|| = 1$.*

A barycentric direction d always satisfies $|d| := \sum_{i=1}^3 d_i = 0$.

Definition 2.4 *The directional derivative of a function $f(\tau)$ with respect to a barycentric direction d is given by*

$$D_d f(\tau) = \sum_{i=1}^{3} \frac{\partial f}{\partial \tau_i} d_i. \tag{2.4}$$

2.2.2 Bernstein polynomials

Let $\lambda = (\lambda_1, \lambda_2, \lambda_3)$ be an ordered set of nonnegative integers and define $|\lambda| := \lambda_1 + \lambda_2 + \lambda_3$.

Definition 2.5 *The Bernstein polynomials of degree m over a triangle are defined by*

$$B_\lambda^m(\tau) = \frac{m!}{\lambda_1!\lambda_2!\lambda_3!}\tau_1^{\lambda_1}\tau_2^{\lambda_2}\tau_3^{\lambda_3}; \quad |\lambda| = m. \tag{2.5}$$

These polynomials form a convex partition of unity on $\mathcal{T}$:

$$\left.\begin{aligned} B_\lambda^m(\tau) &\geq 0 & (2.6)\\ \sum_{|\lambda|=m} B_\lambda^m(\tau) &\equiv 1 & (2.7)\end{aligned}\right\} ,\tau \in \mathcal{T}.$$

Furthermore, their integral can be easily calculated as is stated in the following theorem.

Theorem 2.1

$$\int_{\mathcal{T}} B_\lambda^m(\tau)dxdy = \frac{2A(\mathcal{T})}{(m+1)(m+2)} \tag{2.8}$$

where $A(\mathcal{T})$ is the area of $\mathcal{T}$.

For a proof of this result, we refer to [82].

2.2.3 Bernstein–Bézier polynomials

Definition 2.6 *Let Π_m denote the space of bivariate polynomials of total degree $\leq m$. Any polynomial $p(u), u \in \mathbb{R}^2$, in Π_m, can be uniquely represented as a Bernstein–Bézier polynomial*

$$p(u) := b(\tau) = \sum_{|\lambda|=m} b_\lambda B_\lambda^m(\tau), \tag{2.9}$$

where b_λ are called the Bézier ordinates. Sometimes we will write $b_{\mathcal{T}}(\tau)$ to denote a Bernstein–Bézier polynomial over a particular domain triangle $\mathcal{T}$.

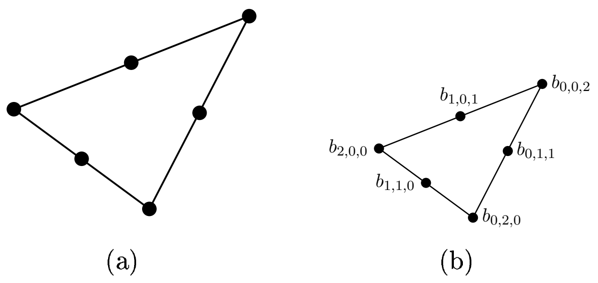


Figure 2.1: (a) The Bézier triangle points of a triangle for $m = 2$. They form a 2–partition of the triangle. (b) Schematic representation of the Bézier ordinates of a quadratic Bernstein–Bézier polynomial.

Definition 2.7 *An m–partition of a triangle $\mathcal{T}$ is obtained by splitting it up in a number of uniform subtriangles, such that each edge of $\mathcal{T}$ is divided into m equal parts.*

Definition 2.8 *The Bézier triangle points of $\mathcal{T}$ are the points with barycentric coordinates $\frac{\lambda}{m}$ with respect to $\mathcal{T}$ for $|\lambda| = m$. They form an m–partition of $\mathcal{T}$.*

Figure 2.1(a) shows the Bézier triangle points of a triangle for $m = 2$. These points play an important role in representing linear functions:

Property 2.2 *(Linear precision) Linear functions can be represented exactly as Bernstein–Bézier polynomials, e.g.,*

$$\begin{aligned} \tau_1 &= \sum_{|\lambda|=m} \frac{\lambda_1}{m} B_\lambda^m(\tau) \\ \tau_2 &= \sum_{|\lambda|=m} \frac{\lambda_2}{m} B_\lambda^m(\tau) \\ \tau_3 &= \sum_{|\lambda|=m} \frac{\lambda_3}{m} B_\lambda^m(\tau). \end{aligned} \tag{2.10}$$

2.2.4 Bernstein–Bézier surfaces

Definition 2.9 *The graph of a Bernstein–Bézier polynomial (2.9) on a triangle $\mathcal{T}$ with vertices V_1, V_2, V_3, is the set of points*

$$\boldsymbol{b}(\tau) := \{(x, y, z) \mid (x, y) = (\tau_1, \tau_2, \tau_3),\ z = b(\tau),\ \tau \in \mathcal{T}\} \tag{2.11}$$

and is called a Bernstein–Bézier surface.

A Bernstein–Bézier surface over a particular triangle $\mathcal{T}$ is denoted $\boldsymbol{b}_{\mathcal{T}}(\tau)$. In view of (2.10), we have

$$\boldsymbol{b}(\tau) = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \\ z \end{bmatrix} = \sum_{|\lambda|=m} \begin{bmatrix} \lambda_1/m \\ \lambda_2/m \\ \lambda_3/m \\ b_\lambda \end{bmatrix} B^m_\lambda(\tau) = \sum_{|\lambda|=m} \begin{bmatrix} \lambda/m \\ b_\lambda \end{bmatrix} B^m_\lambda(\tau). \quad (2.12)$$

This leads to the following definition:

Definition 2.10 *The points $\boldsymbol{b}_\lambda(\frac{\lambda}{m}, b_\lambda)$ are the Bézier control points of the surface $\boldsymbol{b}(\tau)$. Their linear interpolant $\mathcal{B}$ is called the Bézier control net.*

It should be clear now why it is common to represent a Bernstein–Bézier surface schematically by associating each Bézier ordinate b_λ with the corresponding Bézier triangle point $\frac{\lambda}{m}$, as on Figure 2.1(b) for $m = 2$.

Definition 2.11 *(The convex hull) By $[\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n], n \geq 1$, we denote the convex hull of the 3D points $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n$.*

A surface $\boldsymbol{b}(\tau)$ lies within the convex hull of its control points, $[\boldsymbol{b}_\lambda]$, as follows from (2.6)–(2.7). Figure 2.2 shows a Bernstein–Bézier surface, together with its control points and the control net.

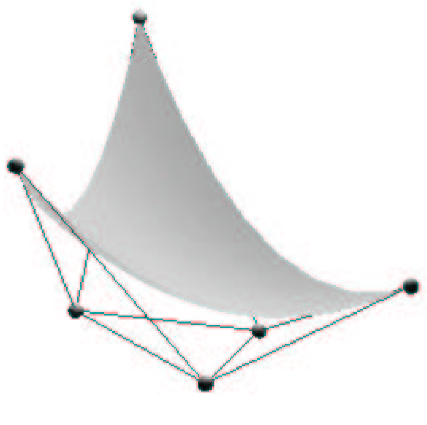


Figure 2.2: A quadratic Bernstein–Bézier surface with its control net.

2.2.5 Calculating with Bernstein–Bézier polynomials

Evaluation of Bernstein–Bézier polynomials

Bernstein–Bézier polynomials can be evaluated by the de Casteljau algorithm:

Theorem 2.2 *(The de Casteljau algorithm)*

$$b(\tau) = b^m_{0,0,0}(\tau) \tag{2.13}$$

where

$$b^0_{\lambda_1,\lambda_2,\lambda_3}(\tau) = b_{\lambda_1,\lambda_2,\lambda_3}, \quad |\lambda| = m, \tag{2.14}$$

and

$$b^r_{\lambda_1,\lambda_2,\lambda_3}(\tau) = \tau_1 b^{r-1}_{\lambda_1+1,\lambda_2,\lambda_3}(\tau) + \tau_2 b^{r-1}_{\lambda_1,\lambda_2+1,\lambda_3}(\tau) + \tau_3 b^{r-1}_{\lambda_1,\lambda_2,\lambda_3+1}(\tau),$$
$$|\lambda| = m - r, \quad r = 1, \ldots, m. \tag{2.15}$$

The b^r_λ are called intermediate de Casteljau ordinates. It is a numerically stable algorithm for $\tau \in \mathcal{T}$ because only convex combinations have to be made. Schumaker and Volk [64] developed an alternative which is more efficient computationally. However, since we need the de Casteljau algorithm in a theoretical context only, we will not consider the alternative here.

Differentiation of Bernstein–Bézier polynomials

Theorem 2.3 *The r-th directional derivative of $b(\tau)$ with respect to d is given by*

$$\begin{aligned} D^r_d b(\tau) &= \frac{m!}{(m-r)!} \sum_{|\lambda|=r} b^{m-r}_\lambda(d) B^r_\lambda(\tau) && (2.16) \\ &= \frac{m!}{(m-r)!} \sum_{|\lambda|=m-r} b^r_\lambda(d) B^{m-r}_\lambda(\tau), \quad r = 0, \ldots, m. && (2.17) \end{aligned}$$

Theorem 2.3 has the following interpretation: in order to compute $D^r_d b(\tau)$, one has to perform $m - r$ de Casteljau steps with respect to τ and r de Casteljau steps with respect to d. These may be carried out in any order. The partial derivatives of a Bernstein–Bézier polynomial can also be found:

Algorithm 2.1 *Given a Bernstein–Bézier polynomial $b(\tau)$, compute its partial derivatives $\frac{\partial}{\partial x} b(\tau)$ and $\frac{\partial}{\partial y} b(\tau)$ at $u = (x, y)$.*

1. Calculate the barycentric coordinates τ of u with respect to $\mathcal{T}$.

2. Compute the unit barycentric direction along the x–direction, say d_x, and along the y–direction, say d_y, i.e.

$$d_x = \frac{(y_2 - y_3, y_3 - y_1, y_1 - y_2)}{e} \tag{2.18}$$

$$d_y = \frac{(x_3 - x_2, x_1 - x_3, x_2 - x_1)}{e} \tag{2.19}$$

with

$$e = \begin{vmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{vmatrix}. \tag{2.20}$$

3. Use the de Casteljau algorithm, performing $m-1$ steps with respect to τ, and one step with respect to d_x or d_y, in order to compute $\frac{\partial}{\partial x}b(\tau)$, resp. $\frac{\partial}{\partial y}b(\tau)$.

Integration of Bernstein–Bézier polynomials

By (2.8) and (2.9) we immediately have

$$\int_{\mathcal{T}} b(\tau)dxdy = \frac{2A(\mathcal{T})}{(m+1)(m+2)} \sum_{|\lambda|=m} b_\lambda. \tag{2.21}$$

The area $A(\mathcal{T})$ is given by

$$A(\mathcal{T}) = \frac{|e|}{2}. \tag{2.22}$$

Bernstein–Bézier polynomials on neighbour domain triangles

Theorem 2.4 *Let $\mathcal{T}$ be a triangle with vertices V_1, V_2 and V_3 and let $\hat{\mathcal{T}}$ be a triangle with vertices $\hat{V}_1, V_2$ and V_3. Let $\hat{V}_1$ have barycentric coordinates σ with respect to $\mathcal{T}$. Denote with $b(\tau), \hat{b}(\hat{\tau})$ a Bernstein–Bézier polynomial of degree m on $\mathcal{T}$, resp. $\hat{\mathcal{T}}$. The two polynomials $b(\tau)$ and $\hat{b}(\hat{\tau})$ are identical if*

$$\hat{b}_{r,\lambda_2,\lambda_3} = b^r_{0,\lambda_2,\lambda_3}(\sigma); \quad 0 \le r \le m, \quad r + \lambda_2 + \lambda_3 = m. \tag{2.23}$$

Proof: see [28].

This theorem has two important consequences, stated in the next two corollaries:

Corollary 2.1 *(Subdivision) Suppose $\hat{V}_1$ is inside $\mathcal{T}$, as in Figure 2.3. The intermediate Bézier ordinates $b^r_\lambda, |\lambda| = m - r$ are the Bézier ordinates of the sub–patches defined over the three subtriangles $\{\hat{V}_1, V_1, V_2\}$, $\{\hat{V}_1, V_2, V_3\}$, $\{\hat{V}_1, V_3, V_1\}$.*

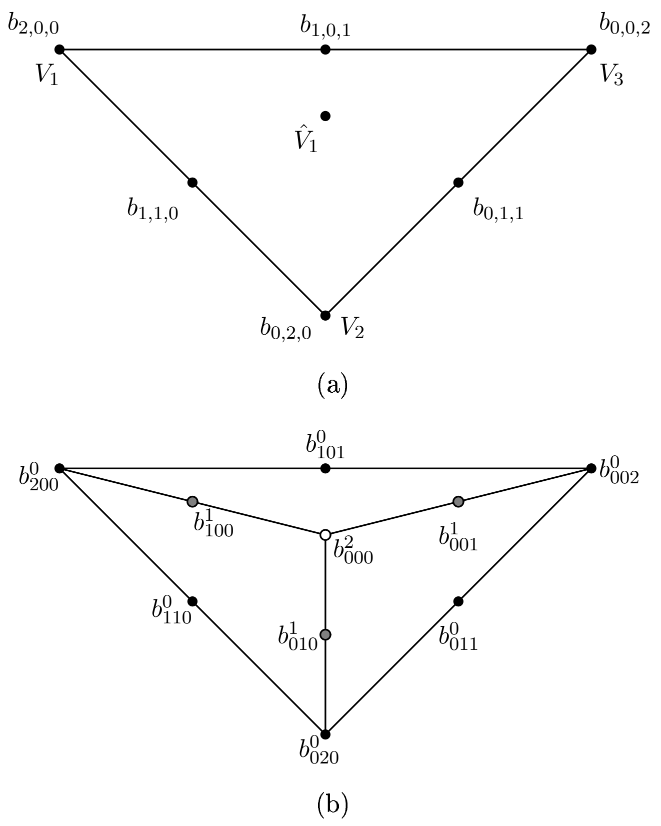


Figure 2.3: Subdivision of quadratic Bernstein–Bézier polynomials on the triangle. (a) The original Bernstein–Bézier triangle and the interior point $\hat{V}_1$. (b) The Bernstein–Bézier triangles after subdivision.

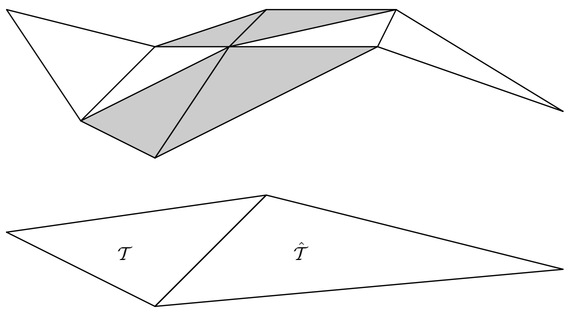


Figure 2.4: C^1 continuity of two adjacent quadratic Bézier triangles requires the shaded pairs of polygons to be an affine map of the domain triangles.

This provides a numerically stable subdivision algorithm: a given Bernstein–Bézier polynomial on $\mathcal{T}$ can now be represented on a subtriangle of $\mathcal{T}$.

Corollary 2.2 *(C^r-continuity) A necessary and sufficient condition for $b(\tau)$ and $\hat{b}(\hat{\tau})$ to be C^r-continuous across the boundary V_2V_3 is*

$$\hat{b}_{\lambda_1,\lambda_2,\lambda_3} = b^{\lambda_1}_{0,\lambda_2,\lambda_3}, \quad \lambda_1 = 0,\dots,r, \quad \lambda_1+\lambda_2+\lambda_3 = m. \tag{2.24}$$

For the case $m = 2, r = 1$, the shaded pairs of triangles in Figure 2.4 have to be coplanar, and moreover each pair has to be an affine map of the pair of domain triangles.

2.2.6 Parametric Bernstein–Bézier surfaces

Definition 2.12 *A parametric Bernstein–Bézier surface of degree m is the set of points*

$$\boldsymbol{p}(u) := \boldsymbol{b}(\tau) \left\{ \begin{array}{ccl} x & = & \sum_{|\lambda|=m} b^x_\lambda B^m_\lambda(\tau) \\ y & = & \sum_{|\lambda|=m} b^y_\lambda B^m_\lambda(\tau) \\ z & = & \sum_{|\lambda|=m} b^z_\lambda B^m_\lambda(\tau) \end{array} \right. , \tau \in \mathcal{T}, \tag{2.25}$$

where τ are the barycentric coordinates of $u = (u_1, u_2) \in \mathbb{R}^2$; $\boldsymbol{b}(\tau)$ is written in shorthand as

$$\boldsymbol{b}(\tau) = \sum_{|\lambda|=m} \boldsymbol{b}_\lambda B^m_\lambda(\tau), \tag{2.26}$$

where $\boldsymbol{b}_\lambda = (b^x_\lambda, b^y_\lambda, b^z_\lambda)$ are the Bézier control points.

A parametric Bernstein–Bézier surface lies within the convex hull $[\boldsymbol{b}_\lambda]$ of its control points. A Bernstein–Bézier surface (2.11) is a particular case of (2.26), having $(x, y) = (u_1, u_2)$.

2.2.7 On calculating with parametric Bernstein–Bézier surfaces

In Section 2.2.5, we gave an overview of the applications of the de Casteljau algorithm for Bernstein–Bézier polynomials. This algorithm can also be applied component–wise to parametric Bernstein–Bézier surfaces, or, stated in a more "geometric" way, one can replace the input of the de Casteljau algorithm, the Bézier ordinates, with the Bézier control points of a parametric Bernstein–Bézier surface. A great number of geometric properties of the surface can now be extracted from the de Casteljau algorithm. Note that these also apply to the graph of a Bernstein–Bézier polynomial.

Definition 2.13 *The points $\boldsymbol{b}^r_\lambda$ calculated by the de Casteljau algorithm are called intermediate de Casteljau points.*

A point on the surface. Invariance properties.

Execution of the de Casteljau algorithm yields the point $\boldsymbol{b}(\tau) = \boldsymbol{b}^m_{0,0,0}$. Moreover, the surface is affine invariant under domain and range transformations.

1. **Affine invariance**. If the control net $\mathcal{B}$ is transformed by an affine map into $\Theta\mathcal{B}$, the corresponding surface $\boldsymbol{b}(\tau)$ is transformed into $\Theta\boldsymbol{b}(\tau)$. This follows from the fact that the de Casteljau algorithm uses linear interpolation only, which is an affine map.

2. **Invariance under affine parameter transformations**. The de Casteljau algorithm is in fact "blind" to the actual domain triangle that the Bernstein–Bézier polynomial is defined over, because it uses barycentric coordinates only. Indeed, by (2.3) a point u will have the same barycentric coordinates after an affine transformation of the domain triangle. Therefore, one may conclude that $\boldsymbol{b}_{\mathcal{T}}(u) = \boldsymbol{b}_{\Theta\mathcal{T}}(\Theta u)$, where Θ is an affine map.

A tangent plane

Computing the component–wise directional derivatives of a Bernstein–Bézier surface has the following geometric interpretation:

Corollary 2.3 *The triangle with vertices $\boldsymbol{b}^{m-1}_{1,0,0}$, $\boldsymbol{b}^{m-1}_{0,1,0}$ and $\boldsymbol{b}^{m-1}_{0,0,1}$ determines the tangent plane to the surface at $\boldsymbol{b}(\tau)$. Moreover, the tangent point has barycentric coordinates τ with respect to this triangle.*

For example, if $m = 2$, the tangent plane of $\boldsymbol{b}(\tau)$ at $(0,0,1)$ is spanned by $\boldsymbol{b}_{0,0,2}$, $\boldsymbol{b}_{1,0,1}$ and $\boldsymbol{b}_{0,1,1}$, and the tangent point is $\boldsymbol{b}_{0,0,2}$.

A normal vector

The component–wise computation of the partial derivatives of $\boldsymbol{b}(\tau)$ yields two vectors which are tangent to the surface. More particularly, they are tangent to the parameter curves on the surface in the u_1– and u_2–direction, through the point $\boldsymbol{b}(\tau)$. A normal vector to the surface at $\boldsymbol{b}(\tau)$ can now be calculated:

$$\boldsymbol{n}(\tau) = \frac{\partial}{\partial u_1}\boldsymbol{b}(\tau) \times \frac{\partial}{\partial u_2}\boldsymbol{b}(\tau) \tag{2.27}$$

More efficiently, we can use the above corollary and calculate a normal vector as the vector product of two arbitrary vectors in the tangent plane, e.g.,

$$\boldsymbol{n}(\tau) = (\boldsymbol{b}^{m-1}_{0,1,0} - \boldsymbol{b}^{m-1}_{1,0,0}) \times (\boldsymbol{b}^{m-1}_{0,0,1} - \boldsymbol{b}^{m-1}_{1,0,0}). \tag{2.28}$$

2.3 PS–spline functions and surfaces

Continuity conditions between Bernstein–Bézier polynomials over adjacent domain triangles require certain relations between their Bézier ordinates to be satisfied. Suppose an interactive tool for surface design represents its surfaces in their Bernstein–Bézier form. A user may or may not be aware of this. But when he or she changes the coordinates of one control point, in order to modify the shape of the surface locally, this may result in the (possibly unexpected) automatic modification of one or more other control points as the program wants to maintain a certain level of continuity. In order to avoid this kind of behaviour, we will look for spline functions on triangulations in a space with global C^1–continuity. Determining the dimension of such a space, and finding a local stable basis, can be simplified by using so–called split triangulations. In particular, we will use the Powell–Sabin split here. This section gives the relevant definitions and properties in this context; for details we refer to Dierckx [19, 20].

2.3.1 The linear space $S^1_2(\Delta^*)$

Let Ω be a simply connected subset of $\mathbb{R}^2$ with a polygonal boundary $\partial\Omega$. Let Δ be a conforming triangulation of Ω, with triangles ρ_j, $j = 1,\dots,t$ and vertices V_i, $i = 1,\dots,n$, having Cartesian coordinates (x_i, y_i). A particular triangle having vertices V_i, V_j and V_k will be denoted $\rho(V_i, V_j, V_k)$ or $\rho_{i,j,k}$. A triangulation is conforming if each triangle ρ_j contains no other vertices of any other triangle ρ_l but its own three vertices. Figure 2.5(a)

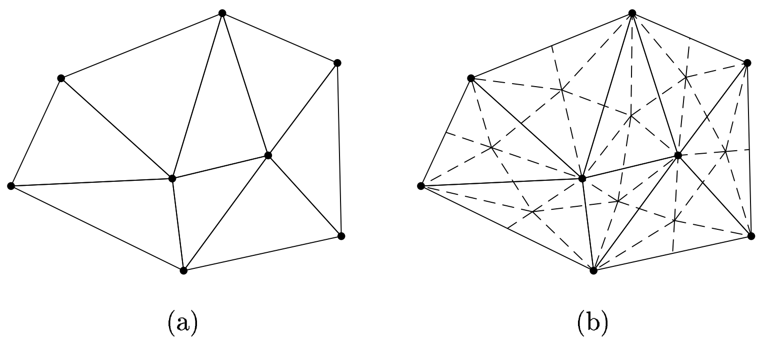


Figure 2.5: (a) A triangulation Δ of a polygonal domain Ω, having 8 vertices and 8 triangles. (b) A Powell–Sabin refinement of this triangulation. The centroid of each triangle has been chosen as interior point during the PS–refinement.

shows a conforming triangulation of a polygonal domain. The set of vertices on the boundary of Δ is denoted $\partial\Delta$. Each vertex V_i has the following characteristics:

Definition 2.14 *The molecule M_i of a vertex V_i consists of all triangles $\rho_j \in \Delta$ having V_i as a vertex.*

Definition 2.15 *The molecule number m_i of a vertex V_i is the number of triangles in the molecule M_i.*

Definition 2.16 *A Powell–Sabin (PS–)refinement Δ^* of Δ is a triangulation that divides each triangle $\rho_j \in \Delta$ into six smaller subtriangles in the following way, as shown in Figure 2.5(b).*

1. *Choose an interior point Z_j in each triangle ρ_j, so that if two triangles ρ_j and ρ_l have a common edge, then the line joining these interior points Z_j and Z_l intersects the common edge at a point $R_{j,l}$ between its vertices. Choosing Z_j as the incentre of each triangle ρ_j ensures the existence of the points $R_{j,l}$, but for practical reasons, other choices such as the centroid may be more appropriate.*
2. *Join each point Z_j to the vertices of ρ_j.*
3. *For each edge of the triangle ρ_j*
   (a) *which belongs to the boundary $\partial\Omega$: join Z_j to an arbitrary point of the edge between its vertices (e.g., the midpoint),*
   (b) *which is common to a triangle ρ_l: join Z_j to $R_{j,l}$.*

The subtriangles thus obtained are denoted $\rho_j^* \in \Delta^*, j = 1, \ldots, t^*$, and $t^* = 6t$.

Definition 2.17 *Given a conforming triangulation Δ of Ω and a PS–refinement Δ^*, the linear space of piecewise quadratic polynomials with C^1 continuity, also called the space of Powell–Sabin splines, is defined as:*

$$S_2^1(\Delta^*) = \left\{ s \in C^1(\Omega) : s|_{\rho_j^*} \in \Pi_2, j = 1, \ldots, t^* \right\}. \tag{2.29}$$

Powell and Sabin [57] studied refinements of the triangulation Δ in the context of contouring a function of two variables, by approximating that function with a piecewise quadratic polynomial. They found the PS–refinement to be the convenient one for solving the following interpolation problem: given any set of triples $(f_i, f_{x,i}, f_{y,i})$, $i = 1, \ldots, n$, find $s(u) \in S_2^1(\Delta^*)$ such that

$$s(V_i) = f_i, \quad \frac{\partial s}{\partial x}(V_i) = f_{x,i}, \quad \frac{\partial s}{\partial y} = f_{y,i}, \quad i = 1, \ldots, n. \tag{2.30}$$

They proved existence and uniqueness of the solution. It follows that the dimension of the space $S_2^1(\Delta^*)$ equals $3n$. This interpolation problem turns out to be particularly useful for constructing a local basis for $S_2^1(\Delta^*)$.

2.3.2 A normalized B–spline representation

A number of authors [20, 66] considered the construction of a locally supported basis for $S_2^1(\Delta^*)$. The general idea is to consider three linearly independent triplets $(\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j}), j = 1, 2, 3$ for each vertex V_i. A B–spline basis function then is the unique solution of the interpolation problem (2.30) with all $(f_l, f_{x,l}, f_{y,l}) = (0, 0, 0)$ except for $(f_i, f_{x,i}, f_{y,i}) = (\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j}) \neq (0, 0, 0)$. It is denoted $B_i^j(u), u \in \mathbb{R}^2$ and is locally supported: $B_i^j(u) \equiv 0, u \notin M_i$. Every $s(u) \in S_2^1(\Delta^*)$ then has a unique representation

$$s(u) = \sum_{i=1}^{n} \sum_{j=1}^{3} c_{i,j} B_i^j(u), \quad u \in \Omega. \tag{2.31}$$

Definition 2.18 *The basis $\left\{ B_i^j(u) \right\}_{i=1,\ldots,n}^{j=1,2,3}$ is said to form a convex partition of unity if*

$$B_i^j(u) \geq 0, \tag{2.32}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{3} B_i^j(u) \equiv 1. \tag{2.33}$$

This property, together with local support, is indispensable for the applicability of representations such as (2.31) in the CAGD field.

Shi et al. [66] showed that for a choice

$$(\alpha_{i,1},\beta_{i,1},\gamma_{i,1}) = (\frac{1}{4},0,\epsilon) \tag{2.34}$$

$$(\alpha_{i,2},\beta_{i,2},\gamma_{i,2}) = (\frac{1}{4},\epsilon,0) \tag{2.35}$$

$$(\alpha_{i,3},\beta_{i,3},\gamma_{i,3}) = (\frac{1}{2},-\epsilon,-\epsilon) \tag{2.36}$$

where $\epsilon \in [1/(4h), 1/(2h)]$ and h is the length of the longest edge of Δ, the corresponding B–splines form a convex partition of unity. The considered basis however may turn out to be very poor from a numerical point of view. For example, if h is large, then ϵ will be small. Hence, if there are large and small molecules in the triangulation, the three B–splines corresponding to a vertex that has a small molecule, will tend to be numerically almost linearly dependent. It seems therefore more appropriate to look for triplets $(\alpha_{i,j},\beta_{i,j},\gamma_{i,j})$ that depend on the molecule M_i.

Dierckx et al. [20] proposed a partially orthonormalized basis with respect to data points in a least–squares computation problem. Also, the Bézier ordinates on the subtriangles resulting from the PS–refinement have been derived explicitly. The B–splines however do not longer form a convex partition of unity, but for least–squares computations this is not immediately a shortcoming (see, e.g., the work of Willemans [82]).

More recently however, Dierckx [19] presented a different way of computing triplets $(\alpha_{i,j},\beta_{i,j},\gamma_{i,j})$ locally depending on M_i, such that the basis functions do form a convex partition of unity. We will use the latter basis for representing PS–splines in CAGD applications. We now briefly recall the construction of this basis, and its main properties.

Construction of the basis functions

Definition 2.19 *(PS–points of a vertex V_i) Consider all edges in the PS–refinement Δ^* that have V_i as a vertex. On each such edge, the Bézier triangle point which is located most nearby V_i, is a PS–point of V_i. V_i itself is also a PS–point of V_i.*

Figure 2.6(a) shows the PS–points for each vertex of the triangulation from the previous examples.

Definition 2.20 *(PS–triangle of V_i) A PS–triangle of a vertex V_i is a triangle that contains the PS–points of V_i. It is denoted as $t_i(Q_{i,1},Q_{i,2},Q_{i,3})$ with $Q_{i,j}, j = 1,2,3,$ being the vertices of the triangle t_i.*

Figure 2.6(b) shows a PS–triangle for each vertex V_i. We will discuss some strategies for finding PS–triangles at the end of this section.

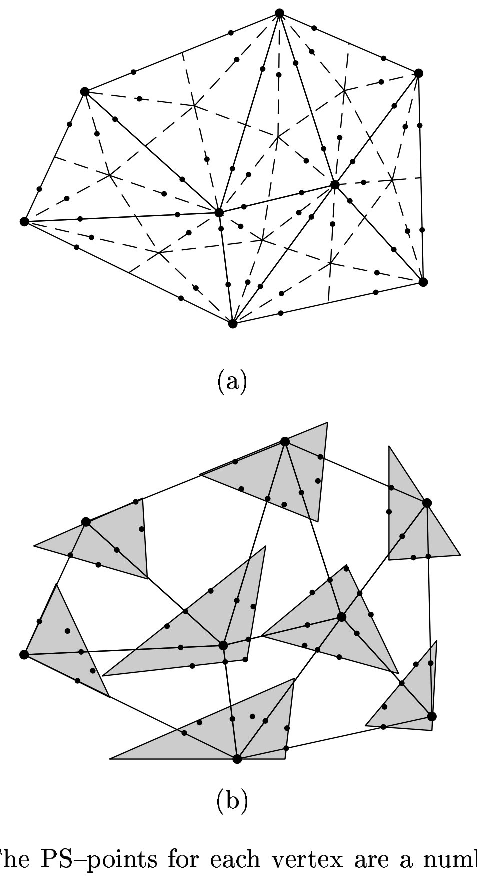


Figure 2.6: (a) The PS–points for each vertex are a number of particular Bézier triangle points in the neighbourhood of the vertex. (b) The PS–triangle corresponding to a vertex, is a triangle that contains the PS–points of that vertex.

Definition 2.21 *(PS–triangle points) The vertices of t_i are the PS–triangle points and are denoted $Q_{i,j}(X_{i,j}, Y_{i,j})$.*

Dierckx [19] proved that a necessary and sufficient condition for (2.32)–(2.33) is that the PS–points are inside the PS–triangles. Furthermore, he shows that there is a one–to–one correspondence between the values of $(\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j})$ and the coordinates of the PS–triangle points $Q_{i,j}, j = 1, 2, 3$.

Given the PS–triangle of a vertex V_i, three linearly independent triplets of real numbers can be found as follows:

1. $\alpha_i = (\alpha_{i,1}, \alpha_{i,2}, \alpha_{i,3})$ are the barycentric coordinates of V_i with respect to t_i,
$$V_i = \alpha_{i,1}Q_{i,1} + \alpha_{i,2}Q_{i,2} + \alpha_{i,3}Q_{i,3}, \tag{2.37}$$

2.
$$\beta_i = (\beta_{i,1}, \beta_{i,2}, \beta_{i,3}) = \left(\frac{Y_{i,2} - Y_{i,3}}{e_i}, \frac{Y_{i,3} - Y_{i,1}}{e_i}, \frac{Y_{i,1} - Y_{i,2}}{e_i}\right), \tag{2.38}$$

3.
$$\gamma_i = (\gamma_{i,1}, \gamma_{i,2}, \gamma_{i,3}) = \left(\frac{X_{i,3} - X_{i,2}}{e_i}, \frac{X_{i,1} - X_{i,3}}{e_i}, \frac{X_{i,2} - X_{i,1}}{e_i}\right), \tag{2.39}$$

where
$$e_i = \begin{vmatrix} X_{i,1} & Y_{i,1} & 1 \\ X_{i,2} & Y_{i,2} & 1 \\ X_{i,3} & Y_{i,3} & 1 \end{vmatrix}. \tag{2.40}$$

Remark the analogy between (2.38)–(2.40) and (2.18)–(2.20).

Corollary 2.4 *The α_i–, β_i– and γ_i–triplets thus calculated satisfy*
$$|\alpha_i| = 1, \quad |\beta_i| = |\gamma_i| = 0. \tag{2.41}$$

Consider the general interpolation problem (2.30). From the local support of the B–spline basis functions, it follows that the B–spline coefficients $(c_{i,1}, c_{i,2}, c_{i,3})$ must satisfy the equations
$$\begin{aligned} c_{i,1}\alpha_{i,1} + c_{i,2}\alpha_{i,2} + c_{i,3}\alpha_{i,3} = s(V_i) &= f_i, & (2.42)\\ c_{i,1}\beta_{i,1} + c_{i,2}\beta_{i,2} + c_{i,3}\beta_{i,3} = \frac{\partial s}{\partial x}(V_i) &= f_{x,i}, & (2.43)\\ c_{i,1}\gamma_{i,1} + c_{i,2}\gamma_{i,2} + c_{i,3}\gamma_{i,3} = \frac{\partial s}{\partial y}(V_i) &= f_{y,i}, & (2.44)\end{aligned}$$

or, taking into account (2.41):
$$\begin{aligned} c_{i,1} &= f_i + f_{x,i}\frac{\alpha_{i,2}\gamma_{i,1} + \gamma_{i,2}(1 - \alpha_{i,1})}{d_i} + f_{y,i}\frac{-\alpha_{i,2}\beta_{i,1} - \beta_{i,2}(1 - \alpha_{i,1})}{d_i}\\ c_{i,2} &= f_i + f_{x,i}\frac{-\alpha_{i,1}\gamma_{i,2} - \gamma_{i,1}(1 - \alpha_{i,2})}{d_i} + f_{y,i}\frac{\alpha_{i,1}\beta_{i,2} + \beta_{i,1}(1 - \alpha_{i,2})}{d_i}\\ c_{i,3} &= f_i + f_{x,i}\frac{-\alpha_{i,1}\gamma_{i,2} - \gamma_{i,1}\alpha_{i,2}}{d_i} + f_{y,i}\frac{\alpha_{i,1}\beta_{i,2} - \beta_{i,1}\alpha_{i,2}}{d_i} \end{aligned} \tag{2.45}$$

with

$$d_i = \begin{vmatrix} \alpha_{i,1} & \alpha_{i,2} & \alpha_{i,3} \\ \beta_{i,1} & \beta_{i,2} & \beta_{i,3} \\ \gamma_{i,1} & \gamma_{i,2} & \gamma_{i,3} \end{vmatrix} = \beta_{i,1}\gamma_{i,2} - \gamma_{i,1}\beta_{i,2} = \frac{1}{e_i}. \tag{2.46}$$

Properties of the B–spline representation

Linear and quadratic functions can be represented exactly as PS–splines. In particular, we will use the representations $s(u) = x$ and $s(u) = y$. The coefficients turn out to be precisely the corresponding components of the coordinates of $Q_{i,j}$:

$$x = \sum_{i=1}^{n}\sum_{j=1}^{3} X_{i,j}B_i^j(u) \tag{2.47}$$

$$y = \sum_{i=1}^{n}\sum_{j=1}^{3} Y_{i,j}B_i^j(u) \tag{2.48}$$

Definition 2.22 *The graph of a PS–spline is the set of points*

$$\boldsymbol{s}(u)\left\{\begin{array}{lcl} x & = & x \\ y & = & y \\ z & = & s(u) \end{array}\right. , \quad u = (x, y) \in \Omega, \tag{2.49}$$

and is called a PS–spline surface.

At this point we can define control points in a similar way as we did for Bernstein–Bézier surfaces:

Definition 2.23 *(Control points) The B–spline control points of a PS–spline surface are defined as $\boldsymbol{c}_{i,j}(X_{i,j}, Y_{i,j}, c_{i,j})$.*

A PS–spline surface $\boldsymbol{s}(u)$ lies within the convex hull if its control points, $[\boldsymbol{c}_{i,j}]$.

Definition 2.24 *(Control triangles) The control triangle T_i corresponding to vertex V_i is defined as $T_i(\boldsymbol{c}_{i,1}, \boldsymbol{c}_{i,2}, \boldsymbol{c}_{i,3})$.*

Figure 2.7 shows a PS–spline surface with its control triangles.

Property 2.3 *The control triangle T_i is tangent to the surface at V_i. The tangent point is $\boldsymbol{s}(V_i)$.*

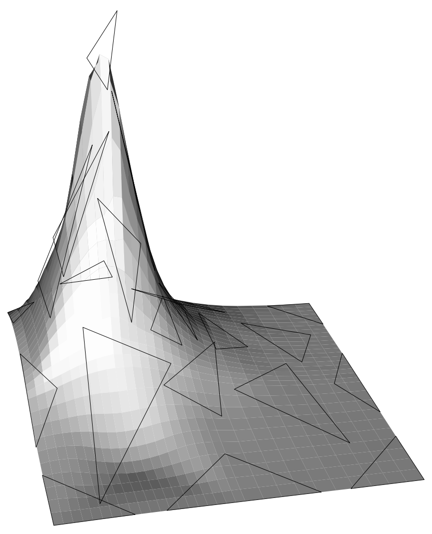


Figure 2.7: PS–spline surface with its control triangles.

The choice of the PS–triangles

There are many possible choices for the PS–triangles. One possibility [19] is to calculate a triangle of minimal area containing the PS–points, for the control points (and the control triangles) to be close to the surface. Computationally, this problem can be reduced to solving a quadratic programming problem in $(\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j})$ for each vertex V_i. The corresponding t_i will be referred to as the optimal triangle. However, it may be useful to have other procedures, for instance interactive ones, to find a suitable triangle t_i. In Chapter 4 we will look at a particular solution for uniform triangulations.

2.3.3 On calculating with PS–splines

Computing the Bernstein–Bézier representation

The Bézier representation of a PS–spline surface can be calculated from the B–spline representation [19]. For future reference, we give explicit formulae for this conversion. Consider a domain triangle $\rho_{i,j,k} \in \Delta$ with its PS–refinement on Figure 2.8(a), where

$$\begin{aligned} R_{i,j} &= \lambda_{i,j} V_i + (1 - \lambda_{i,j}) V_j && (2.50) \\ R_{j,k} &= \lambda_{j,k} V_j + (1 - \lambda_{j,k}) V_k && (2.51) \\ R_{k,i} &= \lambda_{k,i} V_k + (1 - \lambda_{k,i}) V_i && (2.52) \end{aligned}$$

$$Z = a_i V_i + a_j V_j + a_k V_k \tag{2.53}$$

Denote the Bézier ordinates as on Figure 2.8(b). They can be written as the following unique convex combinations of the B–spline coefficients:

$$s_i = \alpha_{i,1}c_{i,1} + \alpha_{i,2}c_{i,2} + \alpha_{i,3}c_{i,3} \tag{2.54}$$
$$u_i = L_{i,1}c_{i,1} + L_{i,2}c_{i,2} + L_{i,3}c_{i,3} \tag{2.55}$$
$$v_i = L'_{i,1}c_{i,1} + L'_{i,2}c_{i,2} + L'_{i,3}c_{i,3} \tag{2.56}$$
$$w_i = L''_{i,1}c_{i,1} + L''_{i,2}c_{i,2} + L''_{i,3}c_{i,3} \tag{2.57}$$

The coefficients in these formulae depend on the geometry of the PS–refinement at hand, and on the choice of the PS–triangles. For example, in (2.57), $(L''_{i,1}, L''_{i,2}, L''_{i,3})$ are nothing but the barycentric coordinates of the Bézier triangle point corresponding to w_i with respect to t_i. Since that point is inside t_i, these numbers are positive indeed. All the explicit formulae can be found in Dierckx [19]. Similar expressions hold for (s_j, u_j, v_j, w_j) and (s_k, u_k, v_k, w_k). The other Bézier ordinates can be found from the C^1–continuity conditions [60], e.g.,

$$r_k = \lambda_{i,j}u_i + \lambda_{j,i}v_j \tag{2.58}$$
$$\theta_k = \lambda_{i,j}w_i + \lambda_{j,i}w_j \tag{2.59}$$
$$\omega = a_i w_i + a_j w_j + a_k w_k, \tag{2.60}$$

where $\lambda_{j,i} := 1 - \lambda_{i,j}$. As an example, the Bézier ordinates of a basis function $B_i^l(x,y), l \in \{1,2,3\}$, are displayed on Figure 2.8(c). Figure 2.9 shows the Bézier control net of a PS–spline.

Evaluation and differentiation of PS–splines

The evaluation and differentiation of PS–splines can be done simultaneously, since both rely on the de Casteljau algorithm.

Algorithm 2.2 *Given a PS–spline (2.31) and a point $u \in \Omega$, compute $s(u)$, $\frac{\partial}{\partial x}s(u)$ and $\frac{\partial}{\partial y}s(u)$.*

1. Find the Bézier subtriangle $\rho_l^* \in \Delta^*$ that contains u.

2. Compute the Bernstein–Bézier representation of the given PS–spline on ρ_l^*.

3. Calculate the barycentric coordinates of u with respect to ρ_l^*, say τ. For differentiation, also compute two unit barycentric directions in the x– and y–direction with respect to ρ_l^*, say d_x and d_y.

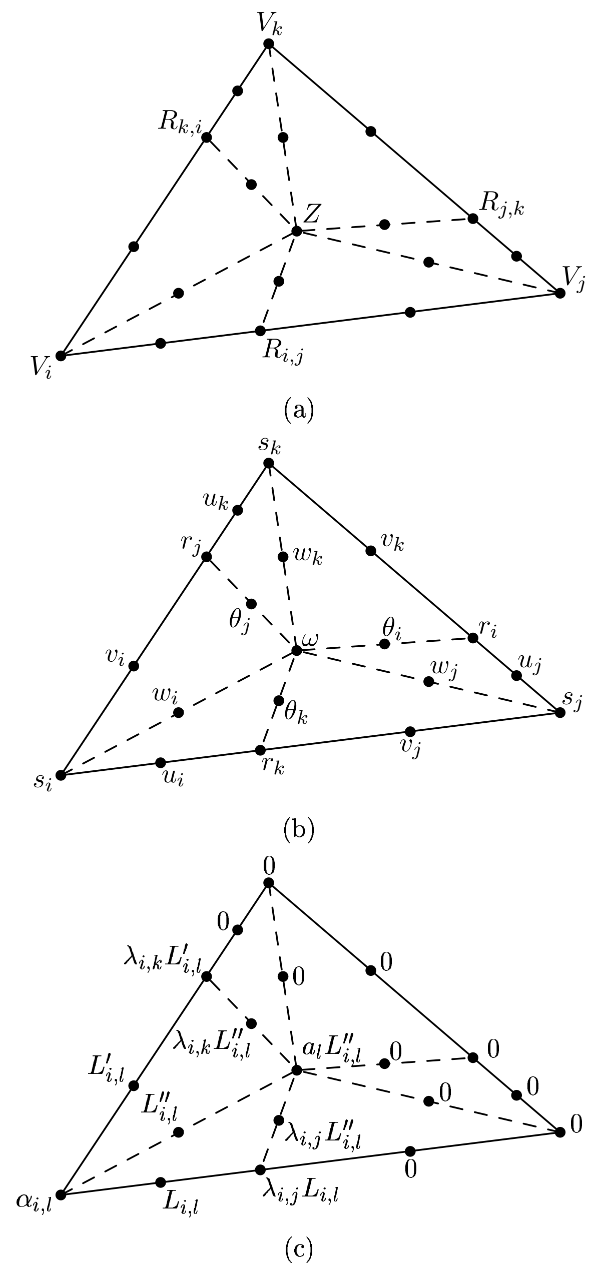


Figure 2.8: (a) PS–refinement of a triangle $\rho(V_i, V_j, V_k)$. (b) Schematic representation of the Bézier ordinates. (c) The Bézier ordinates of a basis function, $B_i^l(x, y), l \in \{1, 2, 3\}$.

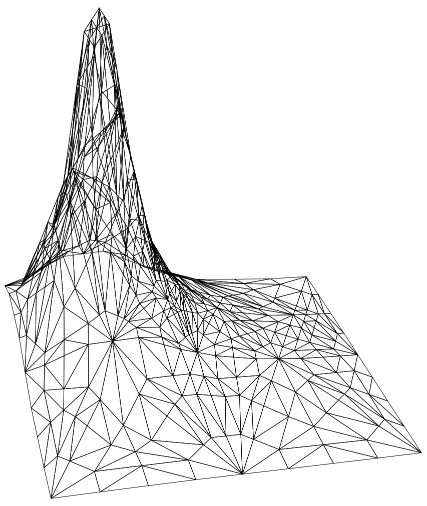


Figure 2.9: The Bézier control net of a PS–spline.

4. **a. Evaluation.** Use the de Casteljau algorithm with respect to τ, see (2.13)–(2.15).

   b. Differentiation. Use the de Casteljau algorithm, performing one step with respect to τ, and the other step with respect to d_x or d_y, as to obtain $\frac{\partial}{\partial x}s(u)$ or $\frac{\partial}{\partial y}s(u)$ as in Algorithm 2.1.

Integration of PS–splines

Given a triangulation Δ of Ω, its PS–refinement Δ^* having triangles ρ_j^*, and a PS–spline $s(u) \in S_2^1(\Delta^*)$,

$$\int_{\Omega} s(u)dxdy = \sum_{\rho_j^* \in \Delta^*} \int_{\rho_j^*} s(u)dxdy. \tag{2.61}$$

Consider the Bernstein–Bézier representation (2.9) of $s(u)$ restricted to one of the Bézier–subtriangles $\rho_j^* \in \Delta^*$, then from (2.21) it follows that

$$\int_{\rho_j^*} s(u)dxdy = \frac{A(\rho_j^*)}{6} \sum_{|\lambda|=2} b_\lambda, \tag{2.62}$$

where $A(\rho^*)$ is the area of the triangle ρ^*, as obtained by (2.20).

Example 2.1 *The integral of the basis function $B_i^l(x,y), l \in \{1,2,3\}$ restricted to $\rho_{i,j,k}$ can now be calculated. Referring to Figure 2.8(a) and (c), this yields*

$$\begin{aligned}
&\int_{\rho_{i,j,k}} B_i^l(x,y)dxdy = \\
&\quad \frac{1}{6}\Big(A\big(\rho^*(V_i,R_{i,j},Z)\big)\big(\alpha_{i,l}+(1+\lambda_{i,j})L_{i,l}+(1+\lambda_{i,j}+a_l)L''_{i,l}\big)\\
&\quad + A\big(\rho^*(V_j,R_{i,j},Z)\big)\big(\lambda_{i,j}L_{i,l}+(a_l+\lambda_{i,j})L''_{i,l}\big)\\
&\quad + A\big(\rho^*(V_j,R_{j,k},Z)\big)\big(a_l L''_{i,l}\big)\\
&\quad + A\big(\rho^*(V_k,R_{j,k},Z)\big)\big(a_l L''_{i,l}\big)\\
&\quad + A\big(\rho^*(V_k,R_{k,i},Z)\big)\big(\lambda_{i,k}L'_{i,l}+(a_l+\lambda_{i,k})L''_{i,l}\big)\\
&\quad + A\big(\rho^*(V_i,R_{k,i},Z)\big)\big(\alpha_{i,l}+(1+\lambda_{i,k})L'_{i,l}+(1+\lambda_{i,k}+a_l)L''_{i,l}\big)\Big)
\end{aligned} \tag{2.63}$$

2.3.4 Parametric Powell–Sabin spline surfaces

The concept of PS–splines can be extended easily to parametric surfaces.

Definition 2.25 *A parametric Powell–Sabin spline surface is defined as the set of points*

$$\boldsymbol{s}(u)\begin{cases} x &= \sum_{i=1}^n \sum_{j=1}^3 c_{i,j}^x B_i^j(u) \\ y &= \sum_{i=1}^n \sum_{j=1}^3 c_{i,j}^y B_i^j(u) \quad ,u=(u_1,u_2)\in\Omega, \\ z &= \sum_{i=1}^n \sum_{j=1}^3 c_{i,j}^z B_i^j(u) \end{cases} \tag{2.64}$$

and is written in shorthand as

$$\boldsymbol{s}(u) = \sum_{i=1}^{n}\sum_{j=1}^{3} \boldsymbol{c}_{i,j} B_i^j(u), \quad u \in \Omega. \tag{2.65}$$

The $\boldsymbol{c}_{i,j} = \big(c_{i,j}^x, c_{i,j}^y, c_{i,j}^z\big)$ are called B–spline control points.

A parametric PS–spline surface $\boldsymbol{s}(u)$ lies within the convex hull of its control points. Referring to (2.49), a PS–spline surface is a particular case of a parametric PS–spline surface with $x = u_1$ and $y = u_2$, i.e., $c_{i,j}^x = X_{i,j}$ and $c_{i,j}^y = Y_{i,j}$.

As for functional PS–splines, we can associate control triangles with $\boldsymbol{s}(u)$ by Definition 2.24. These are tangent to the surface:

Theorem 2.5 *The control triangle T_i of a parametric PS–spline surface is tangent to the surface at V_i. The tangent point is $\boldsymbol{s}(V_i)$.*

This will be proven in a more general setting in Chapter 3. Note that for parametric PS–spline surfaces, the choice of the control points $\boldsymbol{c}_{i,j}$ is free, whereas for functional PS–spline surfaces only the z–component of the control points can be chosen.

2.3.5 Calculating with parametric PS–spline surfaces

In a similar way as the de Casteljau algorithm has been applied to parametric Bernstein–Bézier surfaces in Section 2.2.7, Algorithm 2.2 can be used with the B–spline coefficients replaced by the control points of a parametric PS–spline surface. Referring to Section 2.2.7, the following geometric characteristics can then be calculated immediately (note that these also apply to the graph of a PS–spline):

- a point on the surface, $\boldsymbol{s}(u)$,
- a tangent plane to the surface at $\boldsymbol{s}(u)$ spanned by three intermediate de Casteljau points (see Corollary 2.3), or alternatively defined as the set of points

$$\boldsymbol{p} = \boldsymbol{s}(u) + a\frac{\partial}{\partial u_1}\boldsymbol{s}(u) + b\frac{\partial}{\partial u_2}\boldsymbol{s}(u), \quad a, b \in \mathbb{R}. \tag{2.66}$$

- a normal vector to the surface at $\boldsymbol{s}(u)$:

$$\boldsymbol{n}(u) = \frac{\partial}{\partial u_1}\boldsymbol{s}(u) \times \frac{\partial}{\partial u_2}\boldsymbol{s}(u). \tag{2.67}$$

  Note that for the graph of a PS–spline (2.49), this reduces to

$$\boldsymbol{n}(u) = \left(-\frac{\partial}{\partial x}s(u), -\frac{\partial}{\partial y}s(u), 1\right). \tag{2.68}$$

2.4 Concluding Remarks

We have given the basic theory of piecewise polynomials on triangles, in two parts.

The first part, Section 2.2, covered the topic of Bernstein–Bézier polynomials on triangulations. We showed that they enjoy nearly all the properties that are necessary to make them useful in CAGD: the Bézier basis functions form a convex partition of unity, Bernstein–Bézier polynomials are affine invariant, and linear functions can be represented exactly. Therefore, control points can be associated with the mathematical representation, giving us insight in the shape of the corresponding Bernstein–Bézier surface. Moreover, this surface has the convex hull property. The de Casteljau algorithm was shown to play a central role in the theory of Bernstein–Bézier polynomials in Section 2.2.5. It is the main tool for the evaluation of Bernstein–Bézier polynomials and calculating derivatives, for expressing continuity conditions between Bernstein–Bézier polynomials on adjacent triangles in terms of the Bézier ordinates, and for subdivision. We found that global continuity conditions for Bernstein–Bézier polynomials are hard to implement. The extension to parametric Bernstein–Bézier surfaces was made in Section 2.2.6, and in Section 2.2.7 we showed how to calculate a point on a Bernstein–Bézier surface, as well as a tangent plane and a surface normal.

In Section 2.3, we discussed the normalized B–spline representation for piecewise quadratic polynomials on arbitrary triangulations with global C^1 continuity, the Powell–Sabin splines. We showed that if the triangulation is refined by the Powell–Sabin split, PS–triangles come out as the unique solution of an interpolation problem where function and derivative values are given at the vertices. This leads to an algorithm for constructing a locally supported basis, given in Section 2.3.2. The basis forms a convex partition of unity, and together with affine invariance and linear precision, this property makes it an interesting representation in CAGD. In Section 2.3.2, control triangles are associated with the mathematical representation. These are tangent to the corresponding PS–spline surface, and allow to make predictable local changes that preserve C^1 continuity. We noted that there is a geometric interpretation of constructing a normalized B–spline basis: for each vertex, find a triangle in the domain plane that contains a number of particular points. It is clear that this problem always has a solution. We also gave an explicit solution for the interpolation problem, and for the calculation of the Bézier ordinates, the evaluation, differentiation and integration of PS–splines in Section 2.3.3. The extension to parametric PS–spline surfaces was made in Section 2.3.4, and we concluded with the calculation of points on a PS–spline surface, as well as tangent planes and normal vectors to the surface.

In the next chapter, we will extend these concepts to rational PS–spline polynomials (NURPS). We will show that NURPS are suitable for CAGD applications by proving similar properties for the basis functions as those that hold for Bernstein polynomials and normalized B–splines. We will also show that the algorithms in this chapter (for example, the calculation of the Bézier representation of a PS–spline) can be extended in a numerically stable way to NURPS.

In Chapter 4, we will investigate PS–splines on uniform triangulations, UPS–splines, and show that they have a number of advantages compared with general PS–splines. For example, the form of the PS–triangles can be fixed beforehand, and the calculation schemes of Chapter 2 can be simplified. Not only do they become more efficient in this way, they also give rise to a wide range of interesting applications.

Chapter 3

NURPS

3.1 Introduction

Rational B–spline curves and –tensor product surfaces are commonly used mathematical representations in commercially available computer aided design and computer graphics packages. In a similar way as classical B–spline curves and surfaces are generalized to NURBS, we will investigate the rational extension of Powell–Sabin B–splines to NURPS in this chapter.

First we review Farin's triangular rational Bernstein–Bézier surfaces [30] in Section 3.2. They can be seen as the projection of a 4D Bernstein–Bézier surface in the homogeneous space onto the Euclidean space. We show that the application of the de Casteljau algorithm directly to the homogeneous representation, followed by division through the homogeneous component, can cause numerical problems. The rational de Casteljau algorithm from Farin, given in in Section 3.2.1 is more expensive, but avoids numerical problems by rearranging the calculations. We will also mention the most important applications of the rational de Casteljau algorithm. In Section 3.2.2 we present an alternative subdivision scheme for rational Bernstein–Bézier surfaces, that is better suited for their graphical display than classical subdivision. In a similar way as rational Bézier curves can be used to represent conic sections exactly, rational Bernstein–Bézier surfaces allow to represent patches on quadric surfaces (also called "quadrics"). This has been thoroughly investigated in [4, 5, 22, 23, 31, 65]. We are particularly interested in representing patches on quadrics using triangular rational quadratic Bernstein–Bézier surfaces. The conditions under which this is possible, are given in Section 3.2.3. Another nice feature of rational Bézier curves, namely the existence of shape parameters, which allow to modify the weights in a predictable way, does not yet exist for rational Bernstein–Bézier surfaces.

In Section 3.3 we give the rational extension of PS–spline surfaces to NURPS surfaces. We show that this representation has the necessary properties to make it useful in CAGD. In Section 3.3.2 we show that algorithms for PS–splines that use convex combinations only, can be adapted to NURPS such that numerical stability is maintained, based on the idea behind the rational de Casteljau algorithm. We work out an imported example which is useful in further sections: the calculation of the rational Bernstein–Bézier representation of a given NURPS surface. As an application we show how to calculate a point on a NURPS surface, as well as the tangent plane and a normal vector.

Rational surface representations such as those treated here give a designer extra degrees of freedom compared with their non–rational counterparts through the weights that are associated with each control point. Remarkably, those weights are rarely used in commercial applications. A number of authors [34, 36, 58] studied NURBS weights as a shape control tool in particular applications, showing the usefulness of this extra degree of freedom. Ma [79] used the weights for obtaining a better fit to measured points in Reverse Engineering applications, compared to non–rational forms. In Section 3.4 we will study how the NURPS weights allow to control the shape of the surface. First we define the concept of control triangles for NURPS and prove in Section 3.4.1 that they are tangent to the surface. Next we show that shape parameters can be defined for NURPS surfaces. The weights can be interpreted geometrically, such that an intuitive, predictable way of surface manipulation is possible. The use of the weights becomes transparent to the user (Section 3.4.2). Relying on the conversion algorithm from NURPS surfaces to their rational quadratic Bernstein–Bézier representation, we show in Section 3.4.3 that local planar effects can be modeled using large weights for all the control points of a particular vertex. Other special effects such as cusps and corners will be studied in Section 3.4.4. They do not involve the weights directly.

In Section 3.5 we show how a given triangular rational quadratic Bernstein–Bézier surface can be converted to its NURPS representation in a number of ways. First we give what we call an identical representation (Section 3.5.1), in which the NURPS control points and weights coincide with those of the Bernstein–Bézier surface. This is useful for the functional case. For parametric surfaces, we give an optimal representation in Section 3.5.2 with PS–triangles of minimal area on an equilateral domain triangle. The latter method is used in Section 3.6, where we deal with the representation of quadrics as NURPS surfaces. Quadric surfaces are a basic, commonly used surface type, and are widely spread in industrial and architectonic applications. Nevertheless, they remain unsupported by most free–form design systems with their predominantly parametric representations based on polynomial or rational surface models. There is a lot of scientific literature available dealing with parametric representations of quadric patches. We already mentioned the work related to rational Bernstein–Bézier surfaces. Bangert et al. [3] used triangular segments of quadrics to build tangent plane continuous surfaces interpolating a given point set with prescribed normals. In Sections 3.6.1 and 3.6.2 we give a constructive approach for representing a cylinder and a cone as NURPS surfaces. In Section 3.6.3 an incomplete representation of the sphere as a NURPS surface is obtained. The user has some control on the size of the gaps in the surface.

3.2 Rational Bernstein–Bézier surfaces

In this section we give the rational extension of Bernstein–Bézier surfaces. Section 3.2.1 describes the rational de Casteljau algorithm and its applications. In Section 3.2.2 we give an alternative subdivision scheme for rational Bernstein–Bézier surfaces, that is better suited for the graphical display than classical subdivision of triangular Bernstein–Bézier surfaces. Section 3.2.3 finally discusses the extra possibilities due to the weights, but also the limitations of rational Bernstein–Bézier surfaces with respect to surface design.

3.2.1 The rational de Casteljau algorithm and its applications

In a similar way as B–spline curves and tensor product B–spline surfaces are extended to NURBS curves and surfaces, Bernstein–Bézier surfaces on the triangle can be generalized to their rational counterparts [30].

Definition 3.1 *A rational Bernstein–Bézier surface is the projection of a non–rational 4D Bernstein–Bézier surface in the homogeneous space onto the Euclidean space,*

$$\boldsymbol{b}(\tau) = \frac{\sum_{|\lambda|=m} w_\lambda \boldsymbol{b}_\lambda B^m_\lambda(\tau)}{\sum_{|\lambda|=m} w_\lambda B^m_\lambda(\tau)}, \quad \tau \in \mathcal{T}. \tag{3.1}$$

A weight w_λ is associated with each control point $\boldsymbol{b}_\lambda$. The non–rational Bernstein–Bézier surface in the homogeneous space is denoted $\tilde{\boldsymbol{b}}(\tau)$. It has a 4D control net, being the linear interpolant of the points $\tilde{\boldsymbol{b}}(w_\lambda \boldsymbol{b}_\lambda, w_\lambda)$.

If all weights are equal to one, we obtain the standard non–rational Bernstein–Bézier surface. We shall impose that $w_\lambda > 0$ in order for (3.1) to be defined everywhere on $\mathcal{T}$. If we define the functions

$$\xi_\lambda^m(\tau) = \frac{w_\lambda B_\lambda^m(\tau)}{\sum_{|\sigma|=m} w_\sigma B_\sigma^m(\tau)}, \tag{3.2}$$

then

$$\boldsymbol{b}(\tau) = \sum_{|\lambda|=m} \boldsymbol{b}_\lambda \xi_\lambda^m(\tau), \tag{3.3}$$

where ξ_λ^m form a convex partition of unity,

$$\begin{cases} \xi_\lambda^m(\tau) \geq 0 \\ \sum_{|\lambda|=m} \xi_\lambda^m(\tau) \equiv 1. \end{cases} \tag{3.4}$$

This implies the convex hull property and affine invariance. Furthermore, rational Bernstein–Bézier surfaces enjoy projective invariance: if they are transformed by a projective transformation, we could just as well apply that transformation to the 4D control net, and we would end up with the same surface after division through the homogeneous component.

The rational de Casteljau algorithm. A rational Bernstein–Bézier surface may be evaluated by applying the de Casteljau algorithm to both numerator and denominator and finally dividing through. However, while simple, this method is not numerically stable for weights that vary significantly in magnitude. Indeed, if some of the w_λ are large, the 3D intermediate points $(w_\lambda^r \boldsymbol{b}_\lambda^r)$ are no longer in the convex hull $[\boldsymbol{b}_\lambda]$. A loss of accuracy may then result. A more expensive, but numerically stable way is to project each intermediate de Casteljau point $(w_\lambda^r \boldsymbol{b}_\lambda^r, w_\lambda^r)$ onto the Euclidean space. This is the idea behind Farin's rational de Casteljau algorithm:

Algorithm 3.1 *(The rational de Casteljau algorithm)*

$$\boldsymbol{b}(\tau) = \boldsymbol{b}_{0,0,0}^m \tag{3.5}$$

where

$$w_{\lambda_1,\lambda_2,\lambda_3}^0 = w_{\lambda_1,\lambda_2,\lambda_3}, \tag{3.6}$$

$$\boldsymbol{b}_{\lambda_1,\lambda_2,\lambda_3}^0 = \boldsymbol{b}_{\lambda_1,\lambda_2,\lambda_3}, \tag{3.7}$$

the intermediate de Casteljau weights are calculated as

$$w_{\lambda_1,\lambda_2,\lambda_3}^r = \tau_1 w_{\lambda_1+1,\lambda_2,\lambda_3}^{r-1} + \tau_2 w_{\lambda_1,\lambda_2+1,\lambda_3}^{r-1} + \tau_3 w_{\lambda_1,\lambda_2,\lambda_3+1}^{r-1}, \tag{3.8}$$

and the projections of the intermediate de Casteljau points are given by

$$\boldsymbol{b}^r_{\lambda_1,\lambda_2,\lambda_3} = \tilde{\tau}_1 \boldsymbol{b}^{r-1}_{\lambda_1+1,\lambda_2,\lambda_3} + \tilde{\tau}_2 \boldsymbol{b}^{r-1}_{\lambda_1,\lambda_2+1,\lambda_3} + \tilde{\tau}_3 \boldsymbol{b}^{r-1}_{\lambda_1,\lambda_2,\lambda_3+1}, \tag{3.9}$$

for $r = 1, \dots, m$ *and* $|\lambda| = m - r$, *with*

$$\tilde{\tau}_1 = \frac{\tau_1 w^{r-1}_{\lambda_1+1,\lambda_2,\lambda_3}}{w^r_{\lambda_1,\lambda_2,\lambda_3}} \tag{3.10}$$

$$\tilde{\tau}_2 = \frac{\tau_2 w^{r-1}_{\lambda_1,\lambda_2+1,\lambda_3}}{w^r_{\lambda_1,\lambda_2,\lambda_3}} \tag{3.11}$$

$$\tilde{\tau}_3 = \frac{\tau_3 w^{r-1}_{\lambda_1,\lambda_2,\lambda_3+1}}{w^r_{\lambda_1,\lambda_2,\lambda_3}}. \tag{3.12}$$

Since $\tilde{\tau}_1 + \tilde{\tau}_2 + \tilde{\tau}_3 = 1$, the projections of the intermediate de Casteljau points are now in the convex hull $[\boldsymbol{b}_\lambda]$.

Applications. If we rewrite (3.1) as

$$\boldsymbol{b}(\tau) = \frac{\boldsymbol{p}(\tau)}{w(\tau)} \tag{3.13}$$

with

$$\boldsymbol{p}(\tau) = \sum_{|\lambda|=m} w_\lambda \boldsymbol{b}_\lambda B^m_\lambda(\tau) \tag{3.14}$$

and

$$w(\tau) = \sum_{|\lambda|=m} w_\lambda B^m_\lambda(\tau), \tag{3.15}$$

then we have

$$D_d \boldsymbol{p}(\tau) = w(\tau)\ D_d \boldsymbol{b}(\tau) + D_d w(\tau)\ \boldsymbol{b}(\tau). \tag{3.16}$$

The component–wise directional derivative of a rational Bernstein–Bézier surface can then be found as

$$D_d \boldsymbol{b}(\tau) = \frac{1}{w(\tau)} \left(D_d \boldsymbol{p}(\tau) - D_d w(\tau)\ \boldsymbol{b}(\tau) \right). \tag{3.17}$$

For higher order derivatives, application of Leibniz' rule yields:

$$D^r_d \boldsymbol{b}(\tau) = \frac{1}{w(\tau)} \left(D^r_d \boldsymbol{p}(\tau) - \sum_{j=1}^{r} \binom{r}{j} D^j_d w(\tau) D^{r-j}_d \boldsymbol{b}(\tau) \right). \tag{3.18}$$

Referring to Section 2.2.5, and more particularly to Algorithm 2.1, the component–wise partial derivatives $\frac{\partial}{\partial u_1}\boldsymbol{b}(\tau)$ and $\frac{\partial}{\partial u_2}\boldsymbol{b}(\tau)$ can now be computed. These may be used to calculate the tangent plane and a normal vector to the surface, as in Section 2.2.7, or, alternatively, we can use the fact that $\boldsymbol{b}^{m-1}_{1,0,0}$, $\boldsymbol{b}^{m-1}_{0,1,0}$ and $\boldsymbol{b}^{m-1}_{0,0,1}$ determine a tangent plane to the surface at $\boldsymbol{b}(\tau)$.

3.2.2 Mid–edge subdivision

Subdivision is a commonly used tool for the graphical display of B–spline curves and surfaces. Instead of calculating points on the surface directly, the control net is displayed, after a few subdivision steps. Indeed, the control net converges to the surface after subsequent subdivision. The subdivision scheme for rational Bernstein–Bézier surfaces, based on the de Casteljau algorithm from Section 3.2.1, has a few disadvantages when applied in this manner. Figure 3.1 illustrates schematically how a control net is refined by two consecutive subdivision steps at the barycenter. It is clear that the angles are split up each step, while the edges remain unchanged. Therefore the triangles become more and more steep. This is a disadvantage both numerically and for a smooth graphical display. A solution for this problem is to create a new vertex in the middle of each edge. This is called mid–edge (or uniform) subdivision. Figure 3.2 shows the result for two subdivision steps schematically. In the domain plane, the subtriangles are congruent with the original one. Hence, the angles remain unchanged and, each step, the edges are split up in the middle. We shall now derive a mid–edge subdivision scheme for rational Bernstein–Bézier surfaces.

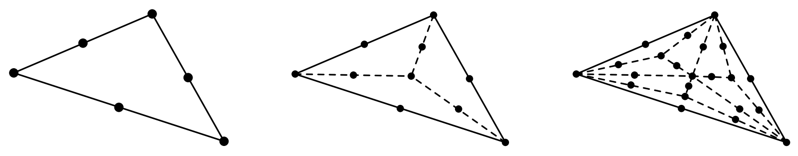


Figure 3.1: Two consecutive steps of the classical rational subdivision scheme displayed schematically. The triangles become more and more steep each step.

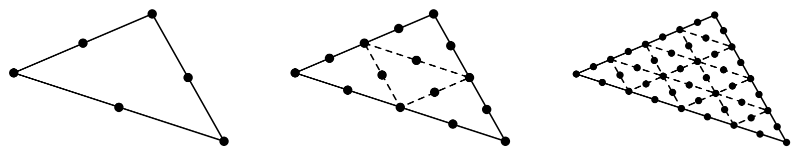


Figure 3.2: Two consecutive steps of the modified rational subdivision scheme displayed schematically. Each sub–triangle is congruent with the initial triangle.

Consider a rational Bernstein–Bézier surface in its homogeneous form on the domain triangle $\mathcal{T}$,

$$\begin{aligned}\tilde{\boldsymbol{b}}_{\mathcal{T}}(\tau) &= \tilde{\boldsymbol{a}}_1 B^2_{2,0,0}(\tau) + \tilde{\boldsymbol{b}}_3 B^2_{1,1,0}(\tau) + \tilde{\boldsymbol{a}}_2 B^2_{0,2,0}(\tau) \\ &+ \tilde{\boldsymbol{b}}_1 B^2_{0,1,1}(\tau) + \tilde{\boldsymbol{a}}_3 B^2_{0,0,2}(\tau) + \tilde{\boldsymbol{b}}_2 B^2_{1,0,1}(\tau). \qquad (3.19)\end{aligned}$$

Figure 3.3(a) shows the control points in the homogeneous space schematically. We shall show that the rational Bernstein–Bézier representation after mid–edge subdivision, on Figure 3.3(b), is given by

$$\tilde{\boldsymbol{v}}_i = \tilde{\boldsymbol{a}}_i \qquad (3.20)$$

$$\tilde{\boldsymbol{u}}_i = \frac{1}{2}\left(\tilde{\boldsymbol{a}}_i + \tilde{\boldsymbol{b}}_{(i+1)_3+1}\right) \qquad (3.21)$$

$$\tilde{\boldsymbol{s}}_i = \frac{1}{2}\left(\tilde{\boldsymbol{a}}_i + \tilde{\boldsymbol{b}}_{i_3+1}\right) \qquad (3.22)$$

$$\tilde{\boldsymbol{r}}_i = \frac{1}{2}\left(\tilde{\boldsymbol{u}}_{i_3+1} + \tilde{\boldsymbol{s}}_{(i+1)_3+1}\right) \qquad (3.23)$$

$$\tilde{\boldsymbol{t}}_i = \frac{1}{4}\left(\tilde{\boldsymbol{a}}_i + \tilde{\boldsymbol{b}}_i + \tilde{\boldsymbol{b}}_{i_3+1} + \tilde{\boldsymbol{b}}_{(i+1)_3+1}\right), i = 1, 2, 3, \qquad (3.24)$$

where $k_3 = k \bmod 3$. It consists of four rational quadratic Bernstein–Bézier surfaces. Let $\hat{\mathcal{T}}$ denote the sub–triangle corresponding to one of these, say

$$\begin{aligned}\tilde{\boldsymbol{b}}_{\hat{\mathcal{T}}}(\tau) &= \tilde{\boldsymbol{v}}_1 \hat{B}^2_{2,0,0}(\tau) + \tilde{\boldsymbol{u}}_1 \hat{B}^2_{1,1,0}(\tau) + \tilde{\boldsymbol{r}}_3 \hat{B}^2_{0,2,0}(\tau) \\ &+ \tilde{\boldsymbol{t}}_1 \hat{B}^2_{0,1,1}(\tau) + \tilde{\boldsymbol{r}}_2 \hat{B}^2_{0,0,2}(\tau) + \tilde{\boldsymbol{s}}_1 \hat{B}^2_{1,0,1}(\tau), \qquad (3.25)\end{aligned}$$

where $\hat{B}^2_\lambda(\tau)$ are the quadratic Bernstein polynomials on $\hat{\mathcal{T}}$. We shall now prove that $\tilde{\boldsymbol{b}}_{\mathcal{T}}(V) \equiv \tilde{\boldsymbol{b}}_{\hat{\mathcal{T}}}(V)$ for any $V \in \hat{\mathcal{T}}$. The other subtriangles can be treated similarly. If the barycentric coordinates of V with respect to $\mathcal{T}$, resp. $\hat{\mathcal{T}}$ are denoted τ, resp. $\hat{\tau}$, then

$$\begin{aligned}\tau_1 &= \hat{\tau}_1 + \frac{\hat{\tau}_2}{2} + \frac{\hat{\tau}_3}{2} \\ \tau_2 &= \frac{\hat{\tau}_2}{2} \qquad (3.26) \\ \tau_3 &= \frac{\hat{\tau}_3}{2}.\end{aligned}$$

Now the definition of the Bernstein polynomials (2.5) applied to (3.19), together with (3.26), yields, after rearranging,

$$\tilde{\boldsymbol{b}}_{\mathcal{T}}(V) = \hat{\tau}_1^2\left(\tilde{\boldsymbol{a}}_1\right) + 2\hat{\tau}_1\hat{\tau}_2\left(\frac{\tilde{\boldsymbol{a}}_1}{2} + \frac{\tilde{\boldsymbol{b}}_3}{2}\right) + \hat{\tau}_2^2\left(\frac{\tilde{\boldsymbol{a}}_1}{4} + \frac{\tilde{\boldsymbol{a}}_2}{4} + \frac{\tilde{\boldsymbol{b}}_3}{2}\right)$$

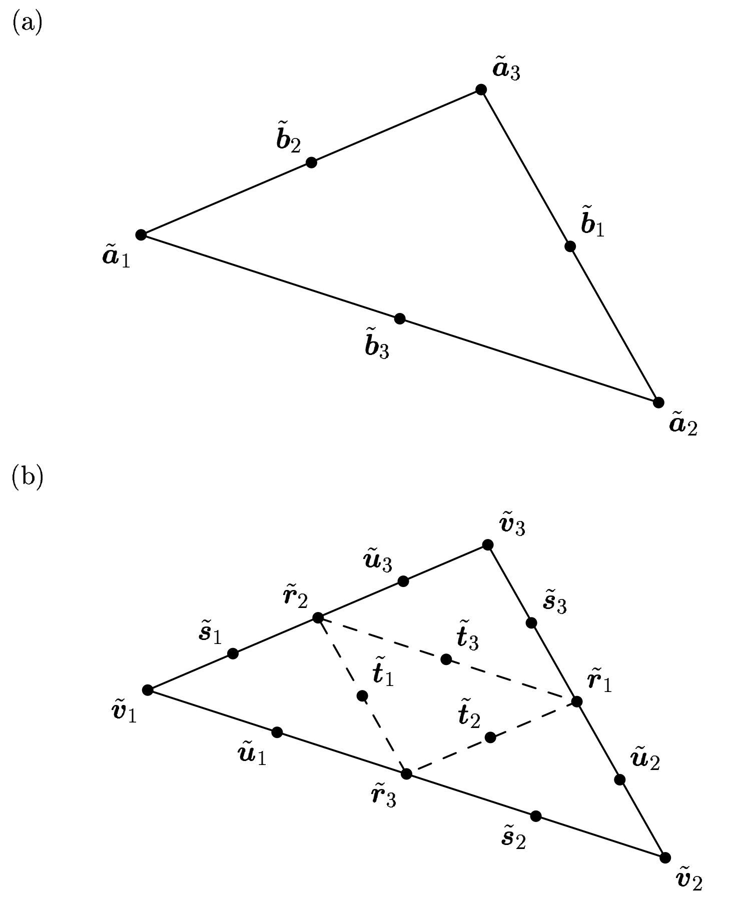

(a)
$\tilde{\boldsymbol{a}}_3$
$\tilde{\boldsymbol{b}}_2$
$\tilde{\boldsymbol{b}}_1$
$\tilde{\boldsymbol{a}}_1$
$\tilde{\boldsymbol{b}}_3$
$\tilde{\boldsymbol{a}}_2$
(b)
$\tilde{\boldsymbol{v}}_3$
$\tilde{\boldsymbol{u}}_3$
$\tilde{\boldsymbol{r}}_2$
$\tilde{\boldsymbol{s}}_3$
$\tilde{\boldsymbol{s}}_1$
$\tilde{\boldsymbol{t}}_3$
$\tilde{\boldsymbol{t}}_1$
$\tilde{\boldsymbol{r}}_1$
$\tilde{\boldsymbol{v}}_1$
$\tilde{\boldsymbol{t}}_2$
$\tilde{\boldsymbol{u}}_1$
$\tilde{\boldsymbol{u}}_2$
$\tilde{\boldsymbol{r}}_3$
$\tilde{\boldsymbol{s}}_2$
$\tilde{\boldsymbol{v}}_2$

Figure 3.3: The homogeneous form of a rational Bernstein–Bézier surface, (a) in the initial state, and (b) after mid–edge subdivision.

$$\begin{aligned}
&+ \quad 2\hat{\tau}_2\hat{\tau}_3\left(\frac{\tilde{\boldsymbol{a}}_1}{4}+\frac{\tilde{\boldsymbol{b}}_1}{4}+\frac{\tilde{\boldsymbol{b}}_2}{4}+\frac{\tilde{\boldsymbol{b}}_3}{2}\right)+\hat{\tau}_3^2\left(\frac{\tilde{\boldsymbol{a}}_1}{4}+\frac{\tilde{\boldsymbol{a}}_3}{4}+\frac{\tilde{\boldsymbol{b}}_2}{2}\right)\\
&+ \quad 2\hat{\tau}_3\hat{\tau}_1\left(\frac{\tilde{\boldsymbol{a}}_1}{2}+\frac{\tilde{\boldsymbol{b}}_2}{2}\right)\\
&= \quad \tilde{\boldsymbol{b}}_{\hat{T}}(\tau). \qquad (3.27)
\end{aligned}$$

With the rational de Casteljau algorithm in mind, it should be noticed that working in the homogeneous space is not numerically stable. In order to improve numerical stability, the calculations can be rearranged in a similar way as in Section 3.2.1. For example, equation (3.21) can be written in its rational form as follows,

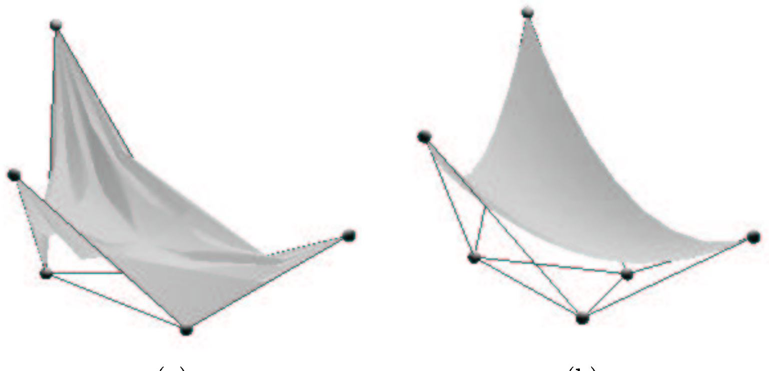
$$w_{\boldsymbol{u}_i} = \frac{1}{2}\left(w_{\boldsymbol{a}_i}+w_{\boldsymbol{b}_j}\right) \qquad (3.28)$$

$$\tilde{\alpha}_{\boldsymbol{a}_i} = \frac{\frac{1}{2}w_{\boldsymbol{a}_i}}{w_{\boldsymbol{u}_i}} \qquad (3.29)$$

$$\tilde{\beta}_{\boldsymbol{b}_j} = \frac{\frac{1}{2}w_{\boldsymbol{b}_j}}{w_{\boldsymbol{u}_i}} \qquad (3.30)$$

$$\boldsymbol{u}_i = \tilde{\alpha}_{\boldsymbol{a}_i}\boldsymbol{a}_i+\tilde{\beta}_{\boldsymbol{b}_j}\boldsymbol{b}_j, \qquad (3.31)$$

where $w_{\boldsymbol{b}_\lambda}$ denotes the weight of $\boldsymbol{b}_\lambda$ and $j = (i+1)_3+1$. Figure 3.4 illustrates the difference between classical and modified subdivision applied recursively.



(a) (b)

Figure 3.4: (a) A surface rendered using classical subdivision recursively. (b) A surface rendered using mid–edge subdivision recursively.

3.2.3 Rational Bernstein–Bézier surface design

Shape parameters. Farin [30] introduced the concept of "shape parameters" in the context of rational Bézier curves. These are parameters with an intuitive geometric interpretation. They are related to the weights in a way that allows to calculate the weights from the shape parameters and vice versa. For rational Bézier curves, a geometric handle along each leg of the control polygon accompanying such a curve, allows to influence the shape of the curve in a predictable way. Such an approach may be preferable compared with entering numerical values for the weights. Surprisingly, this nice property does not carry over to triangular rational Bernstein–Bézier surfaces (first noted by Ramshaw [59]). So far, no geometric means is known that could define weights similar to the weight point approach for Bernstein–Bézier curves.

Representing quadrics exactly. The representation of quadrics using rational Bernstein–Bézier patches has been investigated in the literature. Rational quartic triangular patches always seem sufficient for this purpose. For example, the octant of a sphere represented as a triangular quadric Bernstein–Bézier patch appeared in [31]. On the contrary, not every triangular quadric patch can be represented as rational quadratic Bernstein–Bézier patch. Boehm et al. [4, 5] and Dietz et al. [22, 23] derived conditions for rational quadratic triangular patches and rational biquadratic rectangular patches geometrically and algebraically. They confirm the results of Sederberg et al. [65] in the more general context of Steiner surfaces, and are stated in the next theorem:

Theorem 3.1 *A triangular rational quadratic Bernstein–Bézier surface represents a non–degenerate quadric if and only if all three boundary curves meet in a common point and have coplanar tangents there.*

If the quadric is a paraboloid, then the common intersection point may be at infinity.

3.3 Non uniform rational PS–spline surfaces

In this section we give the rational extension of PS–spline surfaces (Section 3.3.1) and we show how numerically stable algorithms for PS–spline surfaces can be adapted to non uniform rational PS–spline surfaces (Section 3.3.2).

3.3.1 From PS-splines to NURPS

Following the same pattern as in Section 3.2, PS–spline surfaces can be extended to piecewise rational surfaces:

Definition 3.2 *A Non Uniform Rational Powell–Sabin spline (NURPS) surface has the form*

$$\boldsymbol{s}(u) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{3} \boldsymbol{c}_{i,j} w_{i,j} B_i^j(u)}{\sum_{i=1}^{n} \sum_{j=1}^{3} w_{i,j} B_i^j(u)}, \qquad u = (u_1, u_2) \in \Omega, \tag{3.32}$$

where $\boldsymbol{c}_{i,j} = (c_{i,j}^x, c_{i,j}^y, c_{i,j}^z)$ are the B–spline control points. We impose that $w_{i,j} > 0$ in order for $\boldsymbol{s}(u)$ to be defined anywhere on Ω.

A weight $w_{i,j}$ is associated with every control point $\boldsymbol{c}_{i,j}$. A NURPS surface can also be seen as the 3D–projection into the Euclidean space of a 4D PS–spline surface in the homogeneous space:

$$\tilde{\boldsymbol{s}}(u) = \sum_{i=1}^{n} \sum_{j=1}^{3} \tilde{\boldsymbol{c}}_{i,j} B_i^j(u), \tag{3.33}$$

where the 4D control points are defined as

$$\tilde{\boldsymbol{c}}_{i,j} = (w_{i,j} c_{i,j}^x, w_{i,j} c_{i,j}^y, w_{i,j} c_{i,j}^z, w_{i,j}). \tag{3.34}$$

If all $w_{i,j} = 1$, then (3.32) reduces to (2.65). The following can also be readily verified:

$$\boldsymbol{s}(u) = \sum_{i=1}^{n} \sum_{j=1}^{3} \boldsymbol{c}_{i,j} \xi_i^j(u), \tag{3.35}$$

where the functions

$$\xi_i^j(u) = \frac{w_{i,j} B_i^j(u)}{\sum_{k=1}^{n} \sum_{l=1}^{3} w_{k,l} B_k^l(u)} \tag{3.36}$$

form a convex partition of unity:

$$\left\{ \begin{array}{l} \xi_i^j(u) \geq 0 \\ \sum_{i=1}^{n} \sum_{j=1}^{3} \xi_i^j(u) \equiv 1 \end{array} \right., \qquad u \in \Omega, \tag{3.37}$$

and are locally supported:

$$\xi_i^j(u) \equiv 0, \qquad u \notin M_i. \tag{3.38}$$

This implies not only the affine invariance and convex hull properties, but also local control.

3.3.2 On calculating with NURPS

Many computation schemes for Powell–Sabin B–splines can essentially be reduced to the repeated computation of a number of similar convex combinations of the control points. This is an advantage from the numerical point of view. For example, the calculation of the Bernstein–Bézier representation (Section 2.3.3), the evaluation through the de Casteljau algorithm (Section 2.2.5), conversion of a Bernstein–Bézier surface to a B–spline representation (Section 3.5), subdivision (Section 4.4), ... are algorithms that involve convex combinations only. These can be adapted to NURPS surfaces, such that numerical stability is maintained. As an example, we consider the calculation of the rational Bernstein–Bézier representation of a NURPS surface. Then we show how it can be used to determine a point on a NURPS surface, a tangent plane, and a normal vector to the surface.

Calculation of the rational Bernstein–Bézier representation

Consider a NURPS surface $\boldsymbol{s}(u)$ and a domain triangle $\rho_{i,j,k} \in \Delta$ on Figure 3.5(a). The PS–refinement points are given by

$$
\begin{aligned}
R_{i,j} &= \lambda_{i,j} V_i + \lambda_{j,i} V_j && (3.39)\\
R_{j,k} &= \lambda_{j,k} V_j + \lambda_{k,j} V_k && (3.40)\\
R_{k,i} &= \lambda_{k,i} V_k + \lambda_{i,k} V_i && (3.41)\\
Z &= a_i V_i + a_j V_j + a_k V_k. && (3.42)
\end{aligned}
$$

We want to calculate the rational Bernstein–Bézier representation of $\boldsymbol{s}(u)$. The 4D Bézier control points are displayed schematically on Figure 3.5(b). Let $w_{\boldsymbol{b}_\lambda}$ denote the weight of a Bézier control point $\boldsymbol{b}_\lambda$. So we have, for example, for control point $\boldsymbol{\theta}_k$,

$$\tilde{\boldsymbol{\theta}}_k = (w_{\boldsymbol{\theta}_k} \boldsymbol{\theta}_k, w_{\boldsymbol{\theta}_k}). \tag{3.43}$$

Clearly, applying (2.54)–(2.60) component–wise to the numerator, and to the denominator of (3.32), and finally dividing through can lead to numerical problems as was the case for the de Casteljau algorithm applied to rational Bernstein–Bézier surfaces in the homogeneous space, since the 3D points $(w_{i,j}\boldsymbol{c}_{i,j})$ do not necessarily lie in the convex hull $[\boldsymbol{c}_{i,j}]$.

Therefore, we borrow the idea behind the rational de Casteljau algorithm from Farin (Section 3.2.1) to improve numerical stability. By rearranging the calculations, it is possible to avoid working in the homogeneous space.

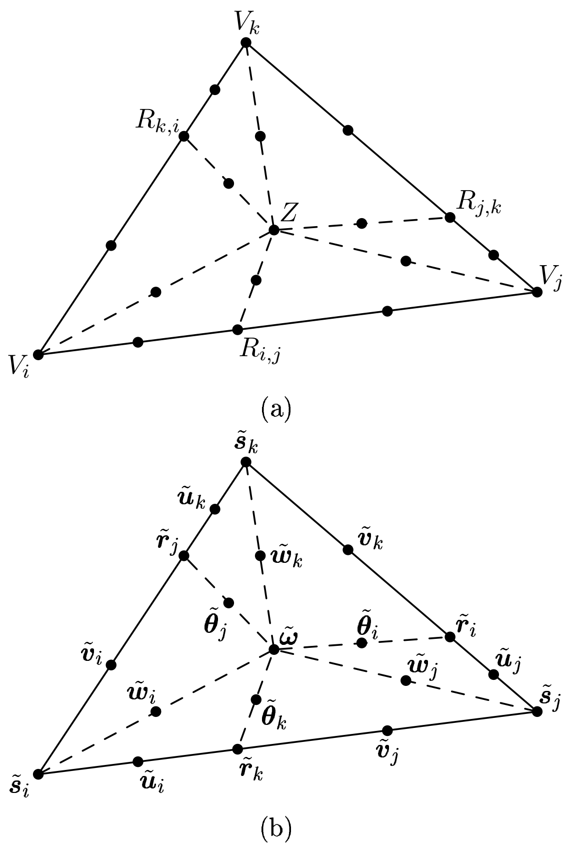

V_k
$R_{k,i}$
Z
$R_{j,k}$
V_j
V_i
$R_{i,j}$
(a)
$\tilde{\boldsymbol{s}}_k$
$\tilde{\boldsymbol{u}}_k$
$\tilde{\boldsymbol{r}}_j$
$\tilde{\boldsymbol{w}}_k$
$\tilde{\boldsymbol{v}}_k$
$\tilde{\boldsymbol{\theta}}_j$
$\tilde{\boldsymbol{\omega}}$
$\tilde{\boldsymbol{\theta}}_i$
$\tilde{\boldsymbol{r}}_i$
$\tilde{\boldsymbol{v}}_i$
$\tilde{\boldsymbol{w}}_j$
$\tilde{\boldsymbol{u}}_j$
$\tilde{\boldsymbol{s}}_j$
$\tilde{\boldsymbol{w}}_i$
$\tilde{\boldsymbol{\theta}}_k$
$\tilde{\boldsymbol{v}}_j$
$\tilde{\boldsymbol{s}}_i$
$\tilde{\boldsymbol{u}}_i$
$\tilde{\boldsymbol{r}}_k$
(b)

Figure 3.5: (a) PS–refinement of a triangle $\rho(V_i, V_j, V_k)$. (b) Schematic representation of the 4D Bézier control points.

First, the weights of the Bézier control points $\boldsymbol{s}_i, \boldsymbol{u}_i, \boldsymbol{v}_i$ and $\boldsymbol{w}_i$ are calculated:

$$w_{\boldsymbol{s}_i} = \alpha_{i,1} w_{i,1} + \alpha_{i,2} w_{i,2} + \alpha_{i,3} w_{i,3} \tag{3.44}$$
$$w_{\boldsymbol{u}_i} = L_{i,1} w_{i,1} + L_{i,2} w_{i,2} + L_{i,3} w_{i,3} \tag{3.45}$$
$$w_{\boldsymbol{v}_i} = L'_{i,1} w_{i,1} + L'_{i,2} w_{i,2} + L'_{i,3} w_{i,3} \tag{3.46}$$
$$w_{\boldsymbol{w}_i} = L''_{i,1} w_{i,1} + L''_{i,2} w_{i,2} + L''_{i,3} w_{i,3}. \tag{3.47}$$

These combinations are convex, so we have, e.g.,

$$L_{i,1} + L_{i,2} + L_{i,3} = 1. \tag{3.48}$$

Then, the following scalars are defined:

$$\tilde{\alpha}_{i,j} = \frac{\alpha_{i,j} w_{i,j}}{w_{\boldsymbol{s}_i}} \quad \tilde{L}_{i,j} = \frac{L_{i,j} w_{i,j}}{w_{\boldsymbol{u}_i}} \tag{3.49}$$
$$\tilde{L}'_{i,j} = \frac{L'_{i,j} w_{i,j}}{w_{\boldsymbol{v}_i}} \quad \tilde{L}''_{i,j} = \frac{L''_{i,j} w_{i,j}}{w_{\boldsymbol{w}_i}}, \tag{3.50}$$

for $j = 1, 2, 3$. The Bézier control points can now be computed as

$$\boldsymbol{s}_i = \tilde{\alpha}_{i,1} \boldsymbol{c}_{i,1} + \tilde{\alpha}_{i,2} \boldsymbol{c}_{i,2} + \tilde{\alpha}_{i,3} \boldsymbol{c}_{i,3} \tag{3.51}$$
$$\boldsymbol{u}_i = \tilde{L}_{i,1} \boldsymbol{c}_{i,1} + \tilde{L}_{i,2} \boldsymbol{c}_{i,2} + \tilde{L}_{i,3} \boldsymbol{c}_{i,3} \tag{3.52}$$
$$\boldsymbol{v}_i = \tilde{L}'_{i,1} \boldsymbol{c}_{i,1} + \tilde{L}'_{i,2} \boldsymbol{c}_{i,2} + \tilde{L}'_{i,3} \boldsymbol{c}_{i,3} \tag{3.53}$$
$$\boldsymbol{w}_i = \tilde{L}''_{i,1} \boldsymbol{c}_{i,1} + \tilde{L}''_{i,2} \boldsymbol{c}_{i,2} + \tilde{L}''_{i,3} \boldsymbol{c}_{i,3}. \tag{3.54}$$

The Bézier control points $\boldsymbol{s}_l, \boldsymbol{u}_l, \boldsymbol{v}_l, \boldsymbol{w}_l, l = j, k$, and the corresponding weights can be found similarly. Next, we compute the weights of the Bézier control points $\boldsymbol{r}_k, \boldsymbol{\theta}_k$ and $\boldsymbol{\omega}$:

$$w_{\boldsymbol{r}_k} = \lambda_{i,j} w_{\boldsymbol{u}_i} + \lambda_{j,i} w_{\boldsymbol{v}_j} \tag{3.55}$$
$$w_{\boldsymbol{\theta}_k} = \lambda_{i,j} w_{\boldsymbol{w}_i} + \lambda_{j,i} w_{\boldsymbol{w}_j} \tag{3.56}$$
$$w_{\boldsymbol{\omega}} = a_i w_{\boldsymbol{w}_i} + a_j w_{\boldsymbol{w}_j} + a_k w_{\boldsymbol{w}_k}, \tag{3.57}$$

and define

$$\tilde{\lambda}_{i,j} = \frac{\lambda_{i,j} w_{\boldsymbol{u}_i}}{w_{\boldsymbol{r}_k}} \quad \tilde{\lambda}_{j,i} = \frac{\lambda_{j,i} w_{\boldsymbol{v}_j}}{w_{\boldsymbol{r}_k}} \tag{3.58}$$
$$\tilde{\lambda}'_{i,j} = \frac{\lambda_{i,j} w_{\boldsymbol{w}_i}}{w_{\boldsymbol{\theta}_k}} \quad \tilde{\lambda}'_{j,i} = \frac{\lambda_{j,i} w_{\boldsymbol{w}_j}}{w_{\boldsymbol{\theta}_k}} \tag{3.59}$$
$$\tilde{a}_l = \frac{a_l w_{\boldsymbol{w}_l}}{w_{\boldsymbol{\omega}}}, l = i, j, k. \tag{3.60}$$

Then we have

$$\begin{aligned} \boldsymbol{r}_k &= \tilde{\lambda}_{i,j}\boldsymbol{u}_i + \tilde{\lambda}_{j,i}\boldsymbol{v}_j && (3.61)\\ \boldsymbol{\theta}_k &= \tilde{\lambda}'_{i,j}\boldsymbol{w}_i + \tilde{\lambda}'_{j,i}\boldsymbol{w}_j && (3.62)\\ \boldsymbol{\omega} &= \tilde{a}_i\boldsymbol{w}_i + \tilde{a}_j\boldsymbol{w}_j + \tilde{a}_k\boldsymbol{w}_k. && (3.63)\end{aligned}$$

Similar expressions hold for $\boldsymbol{r}_l, \boldsymbol{\theta}_l, l = i, j$, and the corresponding weights. Equations (3.51)–(3.54) and (3.61)–(3.63) are convex combinations, thus numerical stability is maintained. We may conclude that all PS–spline algorithms which take convex combinations of the control points only, can be applied to NURPS in a numerical stable way.

A point on the surface

A point on a NURPS surface $\boldsymbol{s}(u)$ can now be computed as follows.

1. Find the Bézier subtriangle $\rho_l^* \in \Delta^*$ that contains u.
2. Compute the rational Bernstein–Bézier representation of the given NURPS on ρ_l^*.
3. Calculate the barycentric coordinates of u with respect to ρ_l^*, say τ.
4. Use the rational de Casteljau algorithm with respect to τ, see (3.5)–(3.12).

A tangent plane and normal vector to the surface

Once the rational Bernstein–Bézier representation $\boldsymbol{b}(\tau)$ of a NURPS surface has been determined on ρ_l^*, the calculation of tangent planes and normal vectors at $\boldsymbol{s}(u)$ is also obvious.

1. Let τ denote the barycentric coordinates of u with respect to ρ_l^*.
2. Either apply the rational de Casteljau algorithm with respect to τ, then the intermediate control points $\boldsymbol{b}^1_{1,0,0}$, $\boldsymbol{b}^1_{0,1,0}$ and $\boldsymbol{b}^1_{0,0,1}$ span a tangent plane at $\boldsymbol{s}(u)$ and the normal vector is given by

$$\boldsymbol{n}(\tau) = (\boldsymbol{b}^1_{0,1,0} - \boldsymbol{b}^1_{1,0,0}) \times (\boldsymbol{b}^1_{0,0,1} - \boldsymbol{b}^1_{1,0,0}). \qquad (3.64)$$

3. Or, determine the unit barycentric directions along the x– and the y–direction (2.18)–(2.19), calculate $\frac{\partial}{\partial u_1} = D_{d_x}\boldsymbol{b}(\tau)$ and $\frac{\partial}{\partial u_2} = D_{d_y}\boldsymbol{b}(\tau)$ by (3.16), then the tangent plane is the set of points

$$\boldsymbol{s}(u) + a\frac{\partial}{\partial u_1}\boldsymbol{s}(u) + b\frac{\partial}{\partial u_2}\boldsymbol{s}(u), \quad a, b \in \mathbb{R}, \qquad (3.65)$$

and the normal vector is

$$\boldsymbol{n}(u) = \frac{\partial}{\partial u_1}\boldsymbol{s}(u) \times \frac{\partial}{\partial u_2}\boldsymbol{s}(u). \tag{3.66}$$

The graphical display of a NURPS surface

A NURPS surface can be displayed graphically by calculating the rational Bernstein–Bézier representation on each sub–triangle ρ_l^*, and displaying those rational Bernstein–Bézier surfaces using the subdivision scheme from Section 3.2.2.

3.4 Designing with NURPS surfaces: weight control

It is a remarkable fact that, while NURBS curves and surfaces are supported by most computer aided design systems, the use of the weights is not. Often these weights are even inaccessible through the user interface of the system. In this section, we show how the NURPS weights can be used as a shape control tool. Section 3.4.1 shows that similarly to PS–spline surfaces, control triangles can be associated with NURPS surfaces that are tangent to the surface. Shape parameters are introduced in Section 3.4.2, that enable to manipulate the weights in an intuitive way. In Section 3.4.3 we show how large weights at a certain vertex result in locally planar surface sections. Section 3.4.4 finally discusses some more special effects that do not involve the weights: cusps and corners can be modeled using degenerate control triangles.

3.4.1 Control plane tangency

Control triangles can be associated with a NURPS surface by Definition 2.24, as was the case for non–rational PS–spline surfaces. They are tangent to the surface:

Theorem 3.2 *The control triangle T_i of a NURPS surface is tangent to the surface at $\boldsymbol{s}(V_i)$.*

Proof: Referring to the locality of the B–splines, it is easy to verify that the (component–wise) evaluation of $\boldsymbol{s}(u)$ and its partial derivatives at vertex V_i, using the rational algorithm, yields

$$\boldsymbol{s}(V_i) = \tilde{\alpha}_{i,1}\boldsymbol{c}_{i,1} + \tilde{\alpha}_{i,2}\boldsymbol{c}_{i,2} + \tilde{\alpha}_{i,3}\boldsymbol{c}_{i,3} \tag{3.67}$$

$$\frac{\partial \boldsymbol{s}(V_i)}{\partial u_1} = \tilde{e}_{i,1}\boldsymbol{c}_{i,1} + \tilde{e}_{i,2}\boldsymbol{c}_{i,2} + \tilde{e}_{i,3}\boldsymbol{c}_{i,3} \tag{3.68}$$

$$\frac{\partial \boldsymbol{s}(V_i)}{\partial u_2} = \tilde{d}_{i,1}\boldsymbol{c}_{i,1} + \tilde{d}_{i,2}\boldsymbol{c}_{i,2} + \tilde{d}_{i,3}\boldsymbol{c}_{i,3} \tag{3.69}$$

for some

$$\tilde{e}_{i,1} + \tilde{e}_{i,2} + \tilde{e}_{i,3} = \tilde{d}_{i,1} + \tilde{d}_{i,2} + \tilde{d}_{i,3} = 0. \tag{3.70}$$

It follows that the control triangle at V_i is tangent to the surface at $\boldsymbol{s}(V_i)$, i.e., any point $\mathbf{p}$ in the tangent plane is a barycentric combination of the control points $\mathbf{c}_{i,1}, \mathbf{c}_{i,2}, \mathbf{c}_{i,3}$:

$$\boldsymbol{p} = \boldsymbol{s}(V_i) + a\,\frac{\partial \boldsymbol{s}(V_i)}{\partial u_1} + b\,\frac{\partial \boldsymbol{s}(V_i)}{\partial u_2},\ a, b \in \mathbb{R}. \tag{3.71}$$

In the following section, we shall give the weights a geometric interpretation. It will then be possible to move the tangent point to any position within T_i by giving the weights $w_{i,j}$ a different value.

3.4.2 Shape parameters

In contrast to rational Bernstein–Bézier surfaces (see Section 3.2.3), shape parameters can be defined for NURPS. Recall that $(\tilde{\alpha}_{i,1}, \tilde{\alpha}_{i,2}, \tilde{\alpha}_{i,3})$ are the barycentric coordinates of $\boldsymbol{s}(V_i)$ with respect to $T_i(\boldsymbol{c}_{i,1}, \boldsymbol{c}_{i,2}, \boldsymbol{c}_{i,3})$. By (3.49), there is a one–one connection between these barycentric coordinates and the weights $w_{i,j}, j = 1, 2, 3$, if $w_{\boldsymbol{s}_i}$ is fixed. Consider the point $\boldsymbol{s}(V_i) = \boldsymbol{s}_i$ on the surface. It is the tangent point of T_i to $\boldsymbol{s}(u)$, and it is a Bézier control point with weight $w_{\boldsymbol{s}_i}$. Suppose this point is dragged to another location $(\tilde{\alpha}'_{i,1}, \tilde{\alpha}'_{i,2}, \tilde{\alpha}'_{i,3})$ within T_i, without changing $w_{\boldsymbol{s}_i}$. The PS–weights corresponding to the new position are found immediately as

$$w'_{i,j} = \frac{\tilde{\alpha}'_{i,j} w_{\boldsymbol{s}_i}}{\alpha_{i,j}} = \frac{\tilde{\alpha}'_{i,j} w_{i,j}}{\tilde{\alpha}_{i,j}},\quad j = 1, 2, 3. \tag{3.72}$$

This justifies that $(\tilde{\alpha}_{i,1}, \tilde{\alpha}_{i,2}, \tilde{\alpha}_{i,3})$ can act as shape parameters. Moreover, if the weight of a B–spline control point $\boldsymbol{c}_{i,j}$ is increased, the surface will be locally attracted by that control point. This corresponds to what one would intuitively expect. For example, doubling the weight $w_{i,j}$ increases the barycentric coordinate of $\boldsymbol{s}_i$ with respect to T_i, corresponding to $\boldsymbol{c}_{i,j}$ by a factor of two. Figure 3.6 illustrates this: on the left, a control triangle with all weights equal to one is used, while on the right, the weight of one of the control points has been multiplied by two.

3.4.3 Special effects: local planar sections

If the control triangles of adjacent vertices V_i, V_j, V_k are chosen to be coplanar, then the surface section $\boldsymbol{s}(u)|_{\rho_{i,j,k}}$ will be in the same plane, as a consequence of the convex hull property. However, using the weights in the NURPS representation, it is possible to achieve more local planar effects.

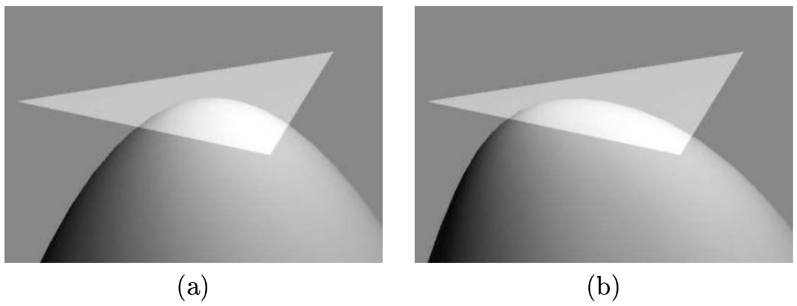


Figure 3.6: The effect increasing the weight of one of the control points. (a) All weights are equal to one. (b) The weight of the leftmost control point has been multiplied by two.

We are interested in the effect of very large weights for the control points of a particular vertex, say V_i, on the shape of the surface. Therefore, suppose that

$$w_{i,1} = w_{i,2} = w_{i,3} = W. \tag{3.73}$$

Consider the restriction of the NURPS surface $\boldsymbol{s}(u)$ to one domain triangle $\rho_{i,j,k}$ on Figure 3.5(a), and its homogeneous Bernstein–Bézier representation on Figure 3.5(b). First we will look at the Bézier sub–triangle $\rho^*(V_i, R_{i,j}, Z)$. More particularly, we want to calculate the Bézier control point $\boldsymbol{r}_k$ using the rational evaluation algorithm from Section 3.3.2. By (3.45), (3.48) and (3.73), the weight of $\boldsymbol{u}_i$ is given by

$$w_{\boldsymbol{u}_i} = W. \tag{3.74}$$

Therefore, the weight of $\boldsymbol{r}_k$ is given by

$$w_{\boldsymbol{r}_k} = \lambda_{i,j} W + \lambda_{j,i} w_{\boldsymbol{v}_j}. \tag{3.75}$$

If we define the following scalars as in (3.58):

$$\tilde{\lambda}_{i,j} = \frac{\lambda_{i,j} W}{\lambda_{i,j} W + \lambda_{j,i} w_{\boldsymbol{v}_j}} = \frac{\lambda_{i,j}}{\lambda_{i,j} + \frac{\lambda_{j,i} w_{\boldsymbol{v}_j}}{W}} \tag{3.76}$$

$$\tilde{\lambda}_{j,i} = \frac{\lambda_{j,i} w_{\boldsymbol{v}_j}}{\lambda_{i,j} W + \lambda_{j,i} w_{\boldsymbol{v}_j}} = \frac{\frac{\lambda_{j,i}}{W} w_{\boldsymbol{v}_j}}{\lambda_{i,j} + \frac{\lambda_{j,i} w_{\boldsymbol{v}_j}}{W}}, \tag{3.77}$$

then $\boldsymbol{r}_k = \tilde{\lambda}_{i,j} \boldsymbol{u}_i + \tilde{\lambda}_{j,i} \boldsymbol{v}_j$. Letting the value W increase, we find

$$\lim_{W \to \infty} \tilde{\lambda}_{i,j} = 1 \quad \text{and} \quad \lim_{W \to \infty} \tilde{\lambda}_{j,i} = 0, \tag{3.78}$$

so

$$\lim_{W\to\infty} \boldsymbol{r}_k = \boldsymbol{u}_i. \tag{3.79}$$

Likewise, one can show that

$$\lim_{w\to\infty} \boldsymbol{\theta}_k = \lim_{w\to\infty} \boldsymbol{\omega} = \boldsymbol{w}_i. \tag{3.80}$$

Recall that the convex hull of a number of points $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n$ is denoted $[\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n]$. From the convex hull property of Bernstein–Bézier surfaces, we may conclude that

$$\boldsymbol{s}(u)|_{\rho^*(V_i,R_{i,j},Z)} = [\boldsymbol{s}_i, \boldsymbol{u}_i, \boldsymbol{w}_i]. \tag{3.81}$$

A similar argument shows that

$$\boldsymbol{s}(u)|_{\rho^*(V_i,R_{k,i},Z)} = [\boldsymbol{s}_i, \boldsymbol{v}_i, \boldsymbol{w}_i] \tag{3.82}$$

and therefore

$$\boldsymbol{s}(u)|_{\rho^*(V_i,R_{i,j},Z)\cup\rho^*(V_i,R_{k,i},Z)} = [\boldsymbol{s}_i, \boldsymbol{u}_i, \boldsymbol{w}_i, \boldsymbol{v}_i]. \tag{3.83}$$

The latter surface section is in the control plane T_i. The same holds for the other triangles $\rho_j \in \Delta$ that have V_i as a vertex. To illustrate, Figure 3.7 shows a NURPS surface with very large weights at a vertex V_i with molecule number $m_i = 6$.

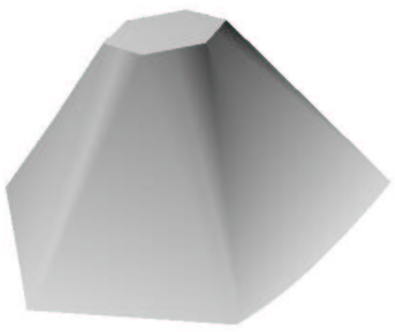


Figure 3.7: The effect of very large weights for the three vertices of one control triangle.

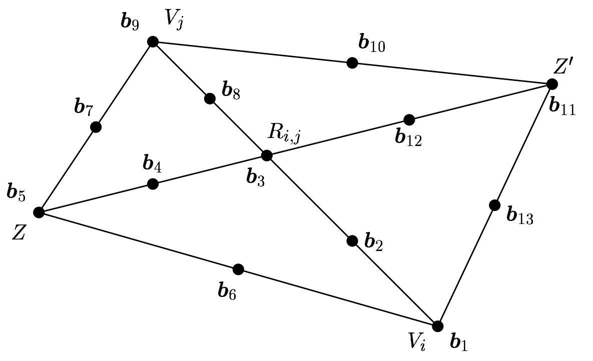


Figure 3.8: The Bézier subtriangles of a non–rational PS–spline surface, adjacent to edge V_iV_j. Also indicated are the Bézier control points.

3.4.4 Special effects: singularities and corners

Some more special effects can be achieved by letting one or more control triangles degenerate to a point or a line. They do not involve the weights $w_{i,j}$, so we shall apply them to non–rational PS–spline surfaces $\boldsymbol{s}(u)$ in this section.

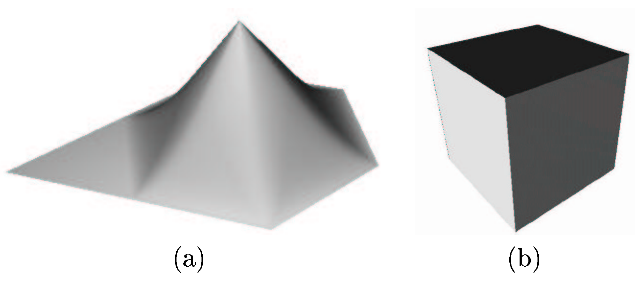


Figure 3.9: (a) A degenerate control triangle at one vertex V_i. (b) Representing corners by using several degenerate control triangles.

A cusp. Consider a non–rational PS–spline surface $\boldsymbol{s}(u)$. The Bézier subtriangles adjacent to an edge V_iV_j from Δ are shown on Figure 3.8, i.e., $\rho^*(V_i, R_{i,j}, Z)$, $\rho^*(V_i, R_{i,j}, Z')$, $\rho^*(V_j, R_{i,j}, Z)$ and $\rho^*(V_j, R_{i,j}, Z')$. Also indicated on the figure are the control points $\boldsymbol{b}_k, k = 1, \ldots, 13$, of the piecewise Bernstein–Bézier representation of the surface. A point on V_iV_j can be represented by its barycentric coordinates with respect to $\rho^*(V_i, R_{i,j}, Z)$, that is, $\sigma = (\sigma_1, \sigma_2, 0)$. The Bernstein–Bézier representation of $\boldsymbol{s}(u)$ on this sub–triangle is denoted $\boldsymbol{b}(\tau)$. The component–wise directional derivative of $\boldsymbol{b}(\tau)$ with respect to a barycentric direction $d(d_1, d_2, d_3)$ at σ is given by

$$D_d\boldsymbol{b}(\sigma) = 2\left(d_1(\sigma_1\boldsymbol{b}_1 + \sigma_2\boldsymbol{b}_2) + d_2(\sigma_1\boldsymbol{b}_2 + \sigma_2\boldsymbol{b}_3) + d_3(\sigma_1\boldsymbol{b}_6 + \sigma_2\boldsymbol{b}_4)\right).$$

Suppose that the control triangle at V_i is degenerate, i.e., $\boldsymbol{c}_{i,1} = \boldsymbol{c}_{i,2} = \boldsymbol{c}_{i,3} = \boldsymbol{C}_i \neq \boldsymbol{b}_3 \neq \boldsymbol{b}_4$. Then from the component–wise application of (2.54)–(2.57), we have

$$D_d\boldsymbol{b}(\sigma) = 2\sigma_2(d_1\boldsymbol{C}_i + d_2\boldsymbol{b}_3 + d_3\boldsymbol{b}_4), \tag{3.84}$$

hence

$$D_d\boldsymbol{b}(\sigma) = 0 \leftrightarrow \sigma_2 = 0. \tag{3.85}$$

The directional derivative is degenerate at $\sigma_2 = 0$, that is, at V_i. The surface has a singularity at that point as illustrated on Figure 3.9(a).

A corner. If the control triangle at V_j is degenerate as well, $\boldsymbol{c}_{j,1} = \boldsymbol{c}_{j,2} = \boldsymbol{c}_{j,3} = \boldsymbol{C}_j \neq \boldsymbol{C}_i$, then equation (3.84) can be written as

$$D_d\boldsymbol{b}(\sigma) = \sigma_2 d_1(\boldsymbol{C}_i - \boldsymbol{C}_j), \tag{3.86}$$

from which it is clear that the directional derivative degenerates to the zero vector at V_i and for directions d parallel to $R_{i,j}Z$; it is parallel to $\boldsymbol{C}_i - \boldsymbol{C}_j$ in all other cases. As an application, a corner can be represented along V_iV_j. For example, Figure 3.9(b) shows a cube represented as a PS–spline surface.

A curved corner. To conclude we give an example where two control triangles degenerate to a point, and a third one in between these two degenerates to a line. As shown on Figure 3.10, a curved corner can be modeled that way.

3.5 Converting from a rational Bernstein–Bézier surface to a NURPS surface

In this section we investigate the problem of finding the NURPS representation of a given rational quadratic Bernstein–Bézier surface $\boldsymbol{b}(\tau)$ on a triangle. This conversion problem does not have a unique solution. We shall deal with two possible schemes in this section. If the domain triangle is given, a so–called identical representation is given in Section 3.5.1, in which the B–spline control points and weights coincide with certain Bézier control points and weights. This is likely to be the simplest approach for the

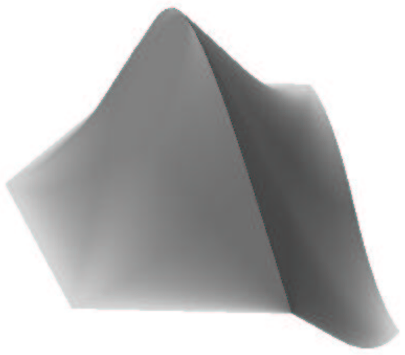


Figure 3.10: A curved corner can be modeled by letting two control triangles degenerate to a point, whereas a third one in between these two degenerates to a line.

functional case. For parametric surfaces however, the domain triangle may not be given beforehand. In that case we propose an equilateral domain triangle and will derive what we call an optimal representation in Section 3.5.2.

3.5.1 An identical representation

First of all, it is worthwhile to note that all the accompanying figures show the domain triangle in part (a) and the schematic representation of the (intermediate) Bézier control points in the homogeneous space in part (b). Also, we need the following theorem.

Proposition 3.1 *Suppose we are given a triangle $\rho_{i,j,k}(V_i, V_j, V_k)$ with centroid Z. If W_l denotes the midpoint of the side opposite to V_l, then $(Z+V_l)/2$ is the centroid of the triangle $\rho(V_l, W_m, W_n), l, m, n \in i, j, k, l \neq m \neq n$.*

The construction of a B–spline representation is based on the rational de Casteljau algorithm (see Section 3.2.1). Consider a rational quadratic Bernstein–Bézier surface $\boldsymbol{b}(\tau)$ on $\rho_{i,j,k}$ as in Figure 3.11. Subdivision at Z, the centroid of $\rho_{i,j,k}$ and at $R_{i,j}$, the midpoint V_iV_j, yields the new Bézier control points as shown on Figure 3.12. For example, the weights and 3D control points corresponding to $\tilde{\boldsymbol{b}}^1_{1,0,0}$, $\tilde{\boldsymbol{b}}^1_{0,1,0}$ and $\tilde{\boldsymbol{d}}^1_{0,0,1}$ are computed through the rational de Casteljau algorithm as follows:

$$w_{\boldsymbol{b}^1_{1,0,0}} = \frac{1}{3}\left(w_{\boldsymbol{b}_{2,0,0}} + w_{\boldsymbol{b}_{1,1,0}} + w_{\boldsymbol{b}_{1,0,1}}\right) \tag{3.87}$$

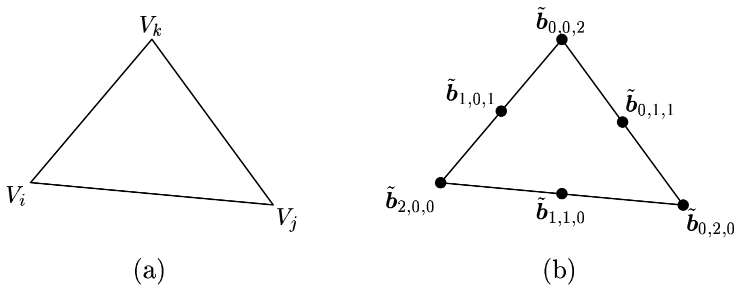


Figure 3.11: (a) A Bézier domain triangle. (b) The schematic representation of the Bézier control points.

$$w_{b^1_{0,1,0}} = \frac{1}{3}\left(w_{b_{1,1,0}} + w_{b_{0,2,0}} + w_{b_{0,1,1}}\right) \tag{3.88}$$

$$w_{d^1_{0,0,1}} = \frac{1}{2}\left(w_{b^1_{1,0,0}} + w_{b^1_{0,1,0}}\right) \tag{3.89}$$

$$b^1_{1,0,0} = \frac{\frac{1}{3}w_{b_{2,0,0}}}{w_{b^1_{1,0,0}}}b_{2,0,0} + \frac{\frac{1}{3}w_{b_{1,1,0}}}{w_{b^1_{1,0,0}}}b_{1,1,0} + \frac{\frac{1}{3}w_{b_{1,0,1}}}{w_{b^1_{1,0,0}}}b_{1,0,1} \tag{3.90}$$

$$b^1_{0,1,0} = \frac{\frac{1}{3}w_{b_{1,1,0}}}{w_{b^1_{0,1,0}}}b_{1,1,0} + \frac{\frac{1}{3}w_{b_{0,2,0}}}{w_{b^1_{0,1,0}}}b_{0,2,0} + \frac{\frac{1}{3}w_{b_{0,1,1}}}{w_{b^1_{0,1,0}}}b_{0,1,1} \tag{3.91}$$

$$d^1_{0,0,1} = \frac{\frac{1}{2}w_{b^1_{1,0,0}}}{w_{d^1_{0,0,1}}}b^1_{1,0,0} + \frac{\frac{1}{2}w_{b^1_{0,1,0}}}{w_{d^1_{0,0,1}}}b^1_{0,1,0} \tag{3.92}$$

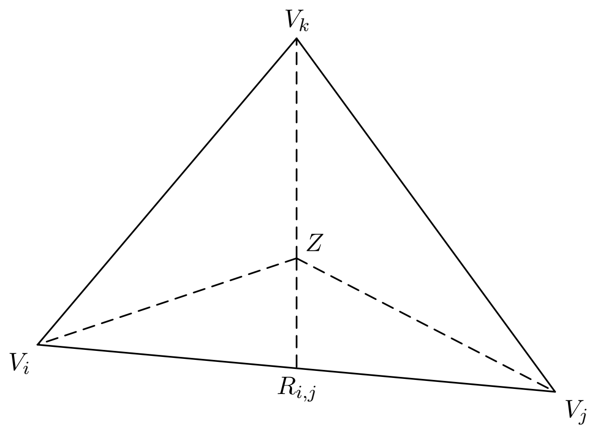
After similar subdivision of the two remaining edges, $b(\tau)$ has been refined into a piecewise rational quadratic Bernstein–Bézier surface $\bar{b}(\tau)$. Now, the six domain subtriangles thus obtained constitute a PS–refinement $\rho^*_{i,j,k}$ of the given triangle, with interior point Z and edge points $R_{i,j}, R_{j,k}$ and $R_{k,i}$ (see Figure 3.13(a)). Let $s(u)$ be the NURPS surface on $\rho^*_{i,j,k}$ with PS–triangle points

$$\begin{array}{lll} Q_{i,1} = V_i, & Q_{i,2} = R_{i,j}, & Q_{i,3} = R_{k,i}, \\ Q_{j,1} = V_j, & Q_{j,2} = R_{j,k}, & Q_{j,3} = R_{i,j}, \\ Q_{k,1} = V_k, & Q_{k,2} = R_{k,i}, & Q_{k,3} = R_{j,k}, \end{array} \tag{3.93}$$

and corresponding weights and control points

$$\begin{array}{lll} w_{i,1} = w_{b_{2,0,0}}, & w_{i,2} = w_{b_{1,1,0}}, & w_{i,3} = w_{b_{1,0,1}}, \\ w_{j,1} = w_{b_{0,2,0}}, & w_{j,2} = w_{b_{0,1,1}}, & w_{j,3} = w_{b_{1,1,0}}, \\ w_{k,1} = w_{b_{0,0,2}}, & w_{k,2} = w_{b_{1,0,1}}, & w_{k,3} = w_{b_{0,1,1}}, \end{array} \tag{3.94}$$

(a)



(b)

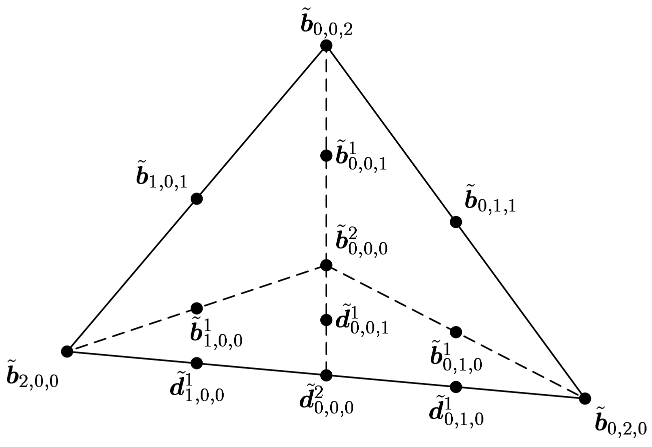

$\tilde{\boldsymbol{b}}_{0,0,2}$
$\tilde{\boldsymbol{b}}^1_{0,0,1}$
$\tilde{\boldsymbol{b}}_{1,0,1}$
$\tilde{\boldsymbol{b}}_{0,1,1}$
$\tilde{\boldsymbol{b}}^2_{0,0,0}$
$\tilde{\boldsymbol{d}}^1_{0,0,1}$
$\tilde{\boldsymbol{b}}^1_{1,0,0}$
$\tilde{\boldsymbol{b}}^1_{0,1,0}$
$\tilde{\boldsymbol{b}}_{2,0,0}$
$\tilde{\boldsymbol{d}}^1_{1,0,0}$
$\tilde{\boldsymbol{d}}^2_{0,0,0}$
$\tilde{\boldsymbol{d}}^1_{0,1,0}$
$\tilde{\boldsymbol{b}}_{0,2,0}$

Figure 3.12: (a) The Bézier domain triangles after subsequent subdivision at the centroid Z and the midpoint of edge V_iV_j. (b) The schematic representation of the (intermediate) Bézier control points resulting from the de Casteljau algorithm.

$$\begin{array}{lll} \boldsymbol{c}_{i,1} = \boldsymbol{b}_{2,0,0}, & \boldsymbol{c}_{i,2} = \boldsymbol{b}_{1,1,0}, & \boldsymbol{c}_{i,3} = \boldsymbol{b}_{1,0,1}, \\ \boldsymbol{c}_{j,1} = \boldsymbol{b}_{0,2,0}, & \boldsymbol{c}_{j,2} = \boldsymbol{b}_{0,1,1}, & \boldsymbol{c}_{j,3} = \boldsymbol{b}_{1,1,0}, \\ \boldsymbol{c}_{k,1} = \boldsymbol{b}_{0,0,2}, & \boldsymbol{c}_{k,2} = \boldsymbol{b}_{1,0,1}, & \boldsymbol{c}_{k,3} = \boldsymbol{b}_{0,1,1}. \end{array} \tag{3.95}$$

By Proposition 3.1, the PS–triangles t_i, t_j and t_k contain the corresponding PS–points; hence this is a valid B–spline representation. We proceed to prove that $\boldsymbol{s}(u) \equiv \bar{\boldsymbol{b}}(\tau)$. In order to do so, we derive the Bernstein–Bézier representation of $\boldsymbol{s}(u)$, schematically displayed on Figure 3.13(b), using the rational scheme from Section 3.3.2. For example, let us calculate the weights and control points corresponding to $\tilde{\boldsymbol{w}}_i$, $\tilde{\boldsymbol{w}}_j$ and $\tilde{\boldsymbol{\theta}}_k$.

From the geometry of the PS–refinement and the choice of the PS–triangles, we have in equation (3.39) that $\lambda_{i,j} = \frac{1}{2}$ and in (3.47) that $(L''_{i,1}, L''_{i,2}, L''_{i,3}) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. Hence

$$w_{\boldsymbol{w}_i} = \frac{1}{3}(w_{i,1} + w_{i,2} + w_{i,3}) \tag{3.96}$$

$$w_{\boldsymbol{w}_j} = \frac{1}{3}(w_{j,1} + w_{j,2} + w_{j,3}) \tag{3.97}$$

$$w_{\boldsymbol{\theta}_k} = \frac{1}{2}(w_{\boldsymbol{w}_i} + w_{\boldsymbol{w}_j}), \tag{3.98}$$

and

$$\boldsymbol{w}_i = \frac{\frac{1}{3}w_{i,1}}{w_{\boldsymbol{w}_i}}\boldsymbol{c}_{i,1} + \frac{\frac{1}{3}w_{i,2}}{w_{\boldsymbol{w}_i}}\boldsymbol{c}_{i,2} + \frac{\frac{1}{3}w_{i,3}}{w_{\boldsymbol{w}_i}}\boldsymbol{c}_{i,3} \tag{3.99}$$

$$\boldsymbol{w}_j = \frac{\frac{1}{3}w_{j,1}}{w_{\boldsymbol{w}_j}}\boldsymbol{c}_{j,1} + \frac{\frac{1}{3}w_{j,2}}{w_{\boldsymbol{w}_j}}\boldsymbol{c}_{j,2} + \frac{\frac{1}{3}w_{j,3}}{w_{\boldsymbol{w}_j}}\boldsymbol{c}_{j,3} \tag{3.100}$$

$$\boldsymbol{\theta}_k = \frac{\frac{1}{2}w_{\boldsymbol{w}_i}}{w_{\boldsymbol{\theta}_k}}\boldsymbol{w}_i + \frac{\frac{1}{2}w_{\boldsymbol{w}_j}}{w_{\boldsymbol{\theta}_k}}\boldsymbol{w}_j. \tag{3.101}$$

Taking account of (3.94)–(3.95) and comparing this with the Bézier control points and weights of $\bar{\boldsymbol{b}}(\tau)$, (3.87)–(3.92), learns that

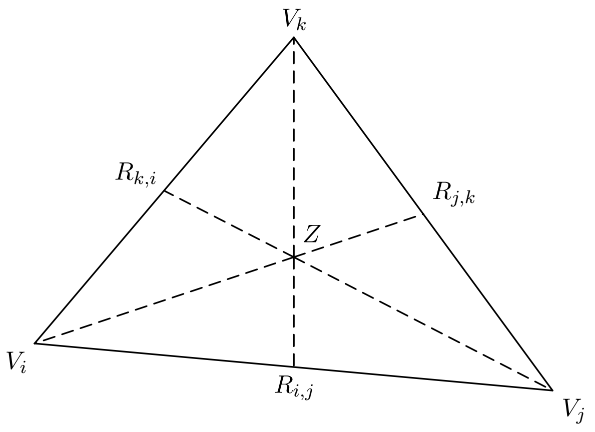
$$\begin{array}{lcl} w_{\boldsymbol{w}_i} = w_{\boldsymbol{b}^1_{1,0,0}} & & \boldsymbol{w}_i = \boldsymbol{b}^1_{1,0,0} \\ w_{\boldsymbol{w}_j} = w_{\boldsymbol{b}^1_{0,1,0}} & \text{and} & \boldsymbol{w}_j = \boldsymbol{b}^1_{0,1,0} \\ w_{\boldsymbol{\theta}_k} = w_{\boldsymbol{d}^1_{0,0,1}} & & \boldsymbol{\theta}_k = \boldsymbol{d}^1_{0,0,1}. \end{array} \tag{3.102}$$

The other weights and control points can be treated similarly. This concludes the proof.

3.5.2 An optimal representation

If the domain triangle is not given, then we may assume it to be equilateral as in Figure 3.14(a). A PS–refinement for this case can then easily be found

(a)



(b)

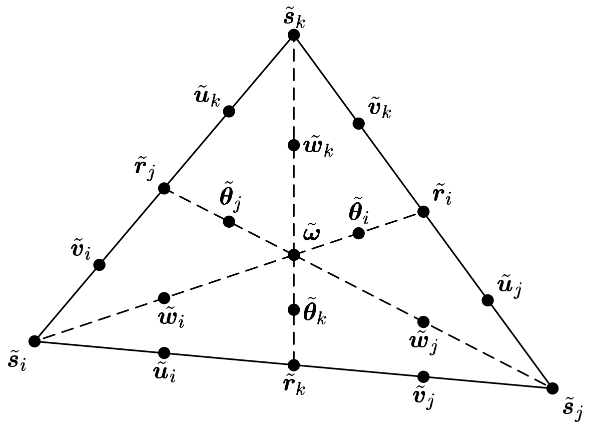


Figure 3.13: (a) The Bézier domain triangles constitute a PS–refinement of $\rho_{i,j,k}$. (b) Schematic representation of the Bézier control points.

by drawing the bisector of each angle, see Figure 3.14(b). The optimal PS–triangle for each vertex is the triangle of minimal area that contains the corresponding PS–points. It is depicted in Figure 3.15. The PS–triangle points can be found from the Bézier triangle points as

$$\begin{aligned} Q_{i,1} &= V_i, && (3.103)\\ Q_{i,2} &= \frac{2}{3}R_{i,i_3+1} + \frac{1}{3}V_i, && (3.104)\\ Q_{i,3} &= \frac{2}{3}R_{(i+1)_3+1,i} + \frac{1}{3}V_i, \quad i = 1,2,3, && (3.105) \end{aligned}$$

with $k_3 = k \bmod 3$ and $R_{i,j} = \frac{1}{2}(V_i+V_j)$. The homogeneous B–spline control points can be found from these equations by replacing $Q_{i,j}$ with $\tilde{\boldsymbol{c}}_{i,j}, j = 1,2,3$, and replacing the Bézier triangle points with the corresponding 4D Bézier control points. In a rational scheme, this becomes, for e.g. the weights and control points of V_1:

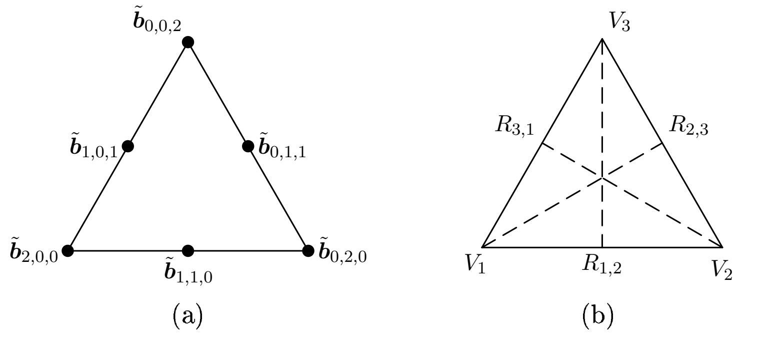
$$\begin{aligned} w_{1,1} &= w_{\boldsymbol{b}_{2,0,0}} && (3.106)\\ w_{1,2} &= \frac{2}{3}w_{\boldsymbol{b}_{1,1,0}} + \frac{1}{3}w_{\boldsymbol{b}_{2,0,0}} && (3.107)\\ w_{1,3} &= \frac{2}{3}w_{\boldsymbol{b}_{1,0,1}} + \frac{1}{3}w_{\boldsymbol{b}_{2,0,0}}, && (3.108) \end{aligned}$$

and

$$\begin{aligned} \boldsymbol{c}_{1,1} &= \boldsymbol{b}_{2,0,0} && (3.109)\\ \boldsymbol{c}_{1,2} &= \frac{\frac{2}{3}w_{\boldsymbol{b}_{1,1,0}}}{w_{1,2}}\boldsymbol{b}_{1,1,0} + \frac{\frac{1}{3}w_{\boldsymbol{b}_{2,0,0}}}{w_{1,2}}\boldsymbol{b}_{2,0,0} && (3.110)\\ \boldsymbol{c}_{1,3} &= \frac{\frac{2}{3}w_{\boldsymbol{b}_{1,0,1}}}{w_{1,3}}\boldsymbol{b}_{1,0,1} + \frac{\frac{1}{3}w_{\boldsymbol{b}_{2,0,0}}}{w_{1,3}}\boldsymbol{b}_{2,0,0}. && (3.111) \end{aligned}$$

3.6 Designing with NURPS surfaces: representing quadrics exactly

Here we present a constructive approach to the problem of representing quadric surfaces as parametric NURPS surfaces. The goal is to derive closed formulae for the control triangles of a cylinder, a cone and patches on the sphere, which is useful in CAGD systems. The calculations and verifications have been made using MAPLE [8], and will be given into detail for the case of the cylinder. Those parts of the treatments of the cone and sphere that are similar, will be given into less detail.



$\tilde{\boldsymbol{b}}_{0,0,2}$
$\tilde{\boldsymbol{b}}_{1,0,1}$
$\tilde{\boldsymbol{b}}_{0,1,1}$
$\tilde{\boldsymbol{b}}_{2,0,0}$
$\tilde{\boldsymbol{b}}_{1,1,0}$
$\tilde{\boldsymbol{b}}_{0,2,0}$
(a)
V_3
$R_{3,1}$
$R_{2,3}$
V_1
$R_{1,2}$
V_2
(b)

Figure 3.14: (a) Schematic display of rational quadratic Bernstein–Bézier surface on a uniform triangle. (b) A PS–refinement of a uniform triangle is obtained by drawing the bisector of each angle.

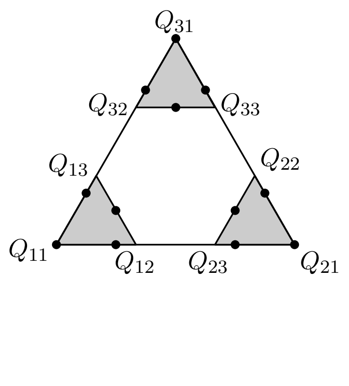


Figure 3.15: A uniform domain triangle and the optimal PS–triangles.

First we shall determine a rational quadratic Bernstein–Bézier patch on the cylinder, cone and sphere, then using (3.106)–(3.111) these can be immediately converted to a NURPS representation. We will rely on the fact that conic sections may be expressed as rational quadratic Bézier curves [30]. This allows to represent the boundaries of the patches exactly.

3.6.1 The cylinder

A cylinder

$$x^2+y^2=r^2,\quad 0\le z\le h \tag{3.112}$$

can be split up into eight isometrical triangular segments. One of these patches is shown on Figure 3.16. We will derive a rational quadratic Bernstein–Bézier representation of this segment,

$$\boldsymbol{b}(\tau)=\frac{1}{w(\tau)}\begin{pmatrix}x(\tau)\\ y(\tau)\\ z(\tau)\end{pmatrix}=\frac{\sum_{|\lambda|=2}w_\lambda \boldsymbol{b}_\lambda B^2_\lambda(\tau)}{\sum_{|\lambda|=2}w_\lambda B^2_\lambda(\tau)},\quad \tau\in\mathcal{T}. \tag{3.113}$$

Substitution into (3.112) yields

$$\left(\frac{x(\tau)}{w(\tau)}\right)^2+\left(\frac{y(\tau)}{w(\tau)}\right)^2=r^2. \tag{3.114}$$

The coordinates of the Bézier control points follow immediately from the fact that the surface is tangent to the control net at the corners of the patch:

$$\begin{array}{lll}\boldsymbol{b}_{2,0,0}=(r,0,0) & \boldsymbol{b}_{1,1,0}=(r,r,0) & \boldsymbol{b}_{0,2,0}=(0,r,0)\\ \boldsymbol{b}_{0,1,1}=\left(r,r,\frac{h}{2}\right) & \boldsymbol{b}_{0,0,2}=(r,0,h) & \boldsymbol{b}_{1,0,1}=\left(r,0,\frac{h}{2}\right).\end{array} \tag{3.115}$$

We set $w_{2,0,0}=w_{0,2,0}=w_{0,0,2}=w_{1,0,1}=1$ since the corresponding control points are on the surface. The other weights can be found by imposing that the edge curves satisfy (3.114), e.g., for the boundary curve connecting $\boldsymbol{b}_{2,0,0}$ and $\boldsymbol{b}_{0,2,0}$:

$$\begin{array}{lll} & x^2(\tau_1,1-\tau_1,0)+y^2(\tau_1,1-\tau_1,0)-r^2w^2(\tau_1,1-\tau_1,0) & = 0\\ \leftrightarrow & \tau_1^4(4r^2w_{1,1,0}^2-2r^2)+\tau_1^3(4r^2-8r^2w_{1,1,0}^2)+\tau_1^2(4r^2w_{1,1,0}^2-2r^2) & = 0\\ \leftrightarrow & w_{1,1,0} & = \frac{\sqrt{2}}{2}.\end{array}$$

Similarly we find $w_{0,1,1}=\frac{\sqrt{2}}{2}$. Back–substitution of these values into the parametric rational Bernstein–Bézier representation allows us to verify that (3.114) is satisfied for all $\tau\in\mathcal{T}$. Application of (3.106)–(3.111) immediately yields the optimal NURPS representation of this surface patch, given in the next table:

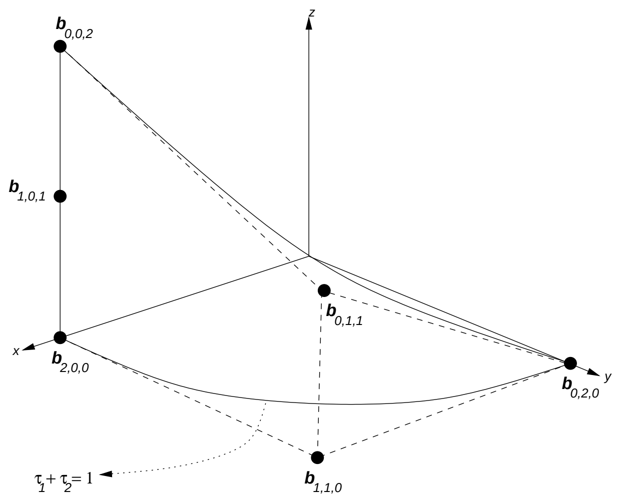


Figure 3.16: A quadratic Bernstein–Bézier patch on the cylinder with radius r, height h, and base at the $z = 0$–plane.

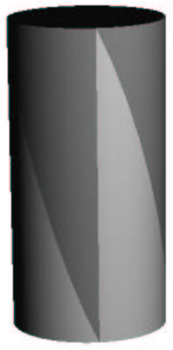


Figure 3.17: A cylinder represented as a NURPS surface. The different patches have been shaded in different shades of gray.

i	$\boldsymbol{c}_{i,1}$	$w_{i,1}$	$\boldsymbol{c}_{i,2}$	$w_{i,2}$	$\boldsymbol{c}_{i,3}$	$w_{i,3}$
1	$(r,0,0)$	1	$(r,\frac{2}{3}r,0)$	$\frac{\sqrt{2}+1}{3}$	$(r,0,\frac{1}{3}h)$	1
2	$(0,r,0)$	1	$(\frac{2}{3}r,r,\frac{1}{3}h)$	$\frac{\sqrt{2}+1}{3}$	$(\frac{2}{3}r,r,0)$	$\frac{\sqrt{2}+1}{3}$
3	$(r,0,h)$	1	$(r,0,\frac{2}{3}h)$	1	$(r,\frac{2}{3}r,\frac{2}{3}h)$	$\frac{\sqrt{2}+1}{3}$

By combining eight such isometrical patches, a complete cylinder can be described by a parametric NURPS surface, as in Figure 3.17. The patches are shown in a different shade of grey.

3.6.2 The cone

A cone

$$x^2 + y^2 = \left(\frac{h-z}{h}r\right)^2, \quad 0 \le z \le h \tag{3.116}$$

can be split up into six isometrical patches. The rational Bernstein–Bézier representation of one such patch (Figure 3.18) can be found in a similar way as for the cylinder. The NURPS representation is given in the following table:

i	$\boldsymbol{c}_{i,1}$	$w_{i,1}$	$\boldsymbol{c}_{i,2}$	$w_{i,2}$	$\boldsymbol{c}_{i,3}$	$w_{i,3}$
1	$(r,0,0)$	1	$(r,\frac{2\sqrt{3}}{9}r,0)$	$\frac{\sqrt{3}+1}{3}$	$(\frac{1}{3}r,0,\frac{2}{3}h)$	1
2	$(r,\frac{\sqrt{3}}{2}r,0)$	1	$(\frac{1}{6}r,\frac{\sqrt{3}}{6}r,\frac{2}{3}h)$	1	$(\frac{5}{6}r,\frac{7\sqrt{3}}{18}r,0)$	$\frac{\sqrt{3}+1}{3}$
3	$(0,0,h)$	1	$(0,0,h)$	1	$(0,0,h)$	1

The complete cone displayed on Figure 3.21(a) is composed of six such isometrical patches. They are shown in a different shade of grey.

3.6.3 The sphere

The sphere can be decomposed into several triangular or quadrilateral patches. Farin et al. [31] represent the octant of a sphere as a nondegenerate quartic triangular Bernstein–Bézier patch and Dietz [21] showed the existence of a tensor product patch of degree $(2, 4)$ for arbitrarily prescribed boundary circles on the sphere. However, it is not possible to represent the entire sphere by triangular piecewise rational quadratic Bernstein–Bézier patches [2]. We are therefore forced to give an incomplete representation of the sphere, but we will look for a solution that leaves only small gaps on the surface. To begin with, we try to represent the half unit sphere

$$x^2 + y^2 + z^2 = 1, \quad z \ge 0 \tag{3.117}$$

as a NURPS surface. By symmetry, an incomplete NURPS representation of the whole sphere is then easily obtained, and a generalization to spheres of radius r is also straightforward.

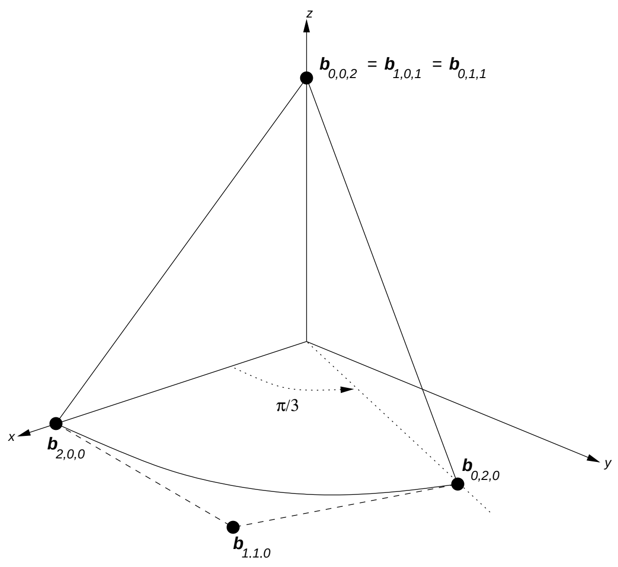


Figure 3.18: A quadratic Bernstein–Bézier patch on the cone with base radius r in the $z = 0$ plane and height h.

Consider the quadratic rational Bernstein–Bézier patch on Figure 3.19, defined by the control points $\boldsymbol{b}_\lambda, |\lambda| = 2$, and the accompanying weights. The vertices $\boldsymbol{b}_{2,0,0}$ and $\boldsymbol{b}_{0,2,0}$ are located in the $z = 0$ plane; the vertex $\boldsymbol{b}_{0,0,2}$ coincides with the north pole of the sphere. The patch is symmetric with respect to the (x, z)–plane. Consider a real value b called the "span", such that if $\boldsymbol{O}$ is the origin, then the angle $\boldsymbol{b}_{2,0,0}\hat{\boldsymbol{O}}\boldsymbol{b}_{0,2,0}$ equals $2b$. Let Q_λ denote the circle on which the boundary curve opposite to the corner control point $\boldsymbol{b}_\lambda$ lies. The patch boundaries connecting $\boldsymbol{b}_{0,0,2}$ with $\boldsymbol{b}_{2,0,0}$ and with $\boldsymbol{b}_{0,2,0}$ are chosen to lie on the two great circles $Q_{0,2,0}$ and $Q_{2,0,0}$ which intersect at the point $\boldsymbol{S}(0, 0, -1)$. The following coordinates are thereby determined:

$$\begin{array}{lll} \boldsymbol{b}_{2,0,0}(\cos b, -\sin b, 0) & \boldsymbol{b}_{0,2,0}(\cos b, \sin b, 0) & \boldsymbol{b}_{0,0,2}(0,0,1) \\ \boldsymbol{b}_{0,1,1}(\cos b, \sin b, 1) & \boldsymbol{b}_{1,0,1}(\cos b, -\sin b, 1) & \end{array} \tag{3.118}$$

Setting the weights for the corner control points equal to one, the value of $w_{0,1,1}$ is determined by imposing the boundary curve connecting $\boldsymbol{b}_{0,0,2}$ and $\boldsymbol{b}_{0,2,0}$ to be on $Q_{2,0,0}$. This yields $w_{0,1,1} = \frac{\sqrt{2}}{2}$, and by symmetry we have $w_{1,0,1} = w_{0,1,1}$. Recall from Theorem 3.1 that the boundary circles $Q_{2,0,0}$, $Q_{0,2,0}$ and $Q_{0,0,2}$ need to intersect in a common point. Therefore,

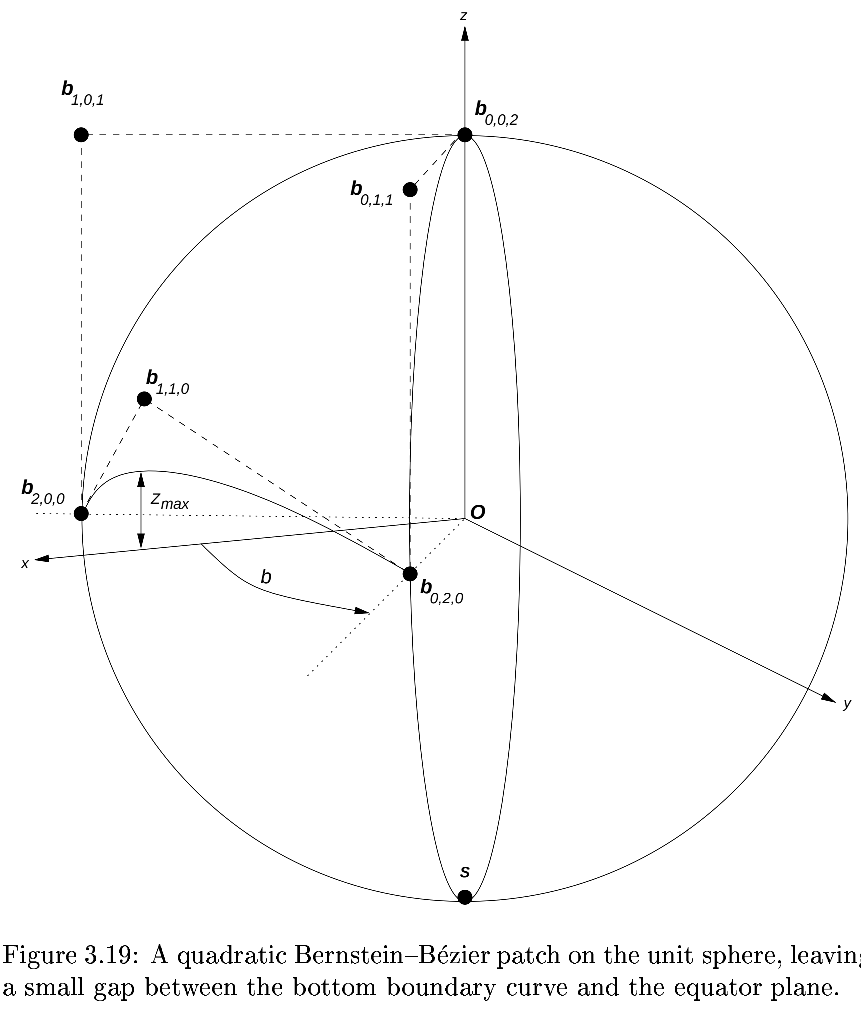


Figure 3.19: A quadratic Bernstein–Bézier patch on the unit sphere, leaving a small gap between the bottom boundary curve and the equator plane.

the remaining edge curve should be in the plane with equation

$$\begin{vmatrix} x & y & z & 1 \\ \cos b & -\sin b & 0 & 1 \\ \cos b & \sin b & 0 & 1 \\ 0 & 0 & -1 & 1 \end{vmatrix} = x - (z+1)\cos b = 0, \tag{3.119}$$

and the unknown control point $\boldsymbol{b}_{1,1,0}$ has coordinates $((\bar{z}+1)\cos b, 0, \bar{z})$ for some $\bar{z} \in \mathbb{R}$. Now recall from [21] that the sum of the angles at the patch corners equals π. Since $\boldsymbol{b}_{0,1,1}\hat{\boldsymbol{b}}_{0,0,2}\boldsymbol{b}_{1,0,1} = 2b$, and by the symmetry of the patch with respect to the (x, z)–plane, the following equation holds for the angle $\boldsymbol{b}_{1,1,0}\hat{\boldsymbol{b}}_{0,2,0}\boldsymbol{b}_{0,1,1}$:

$$\begin{aligned} \frac{\bar{z}}{\sqrt{\bar{z}^2\cos^2 b + \sin^2 b + \bar{z}^2}} &= \cos\left(\frac{\pi}{2} - b\right) & (3.120) \\ &= \sin b. & (3.121) \end{aligned}$$

Solving this equation for positive $\bar{z}$ yields $\bar{z} = \tan^2 b$. The coordinates of the last control point then are

$$\boldsymbol{b}_{1,1,0}\left(\frac{1}{\cos b}, 0, \tan^2 b\right). \tag{3.122}$$

Finally, the condition for the third boundary curve to be on the unit sphere yields $w_{1,1,0} = \cos^2 b$. Substitution of the control points and weights thus obtained in the general form of the rational Bernstein–Bézier patch (3.113) allows us to verify that the whole of the patch satisfies

$$\left(\frac{x(\tau)}{q(\tau)}\right)^2 + \left(\frac{y(\tau)}{q(\tau)}\right)^2 + \left(\frac{z(\tau)}{q(\tau)}\right)^2 = 1. \tag{3.123}$$

A patch on the sphere with radius r can be found by multiplying the coordinates of each control point with r. In general, by letting $b = \frac{\pi}{n}$, the entire sphere can be approximated by $2n$ such (isometrical) patches. Recall that we are interested in a solution that leaves only small gaps around the equator plane. Consider again the patch from Figure 3.19. The height of the gap, bounded by the lower boundary curve and the plane $z = 0$ can be measured by the maximum z–coordinate of the points along this curve. A straightforward calculation yields

$$z_{\max} = \frac{\sin^2 b}{\cos^2 b + 1}. \tag{3.124}$$

This size as a function of the span b is plotted on Figure 3.20 for $b \in \left[0, \frac{\pi}{2}\right]$. One can see that the smallest gaps can be achieved by choosing a tiny span

b, but then a large number of patches have to be used. An incomplete NURPS representation of the sphere with $b = \frac{\pi}{6}$ is given in Figure 3.21(b). The patches are shown in different shades of grey. The optimal NURPS control points and weights can be calculated using (3.103)–(3.105). That is, for a sphere with radius r:

i	$\boldsymbol{c}_{i,1}$	$w_{i,1}$	$\boldsymbol{c}_{i,2}$	$w_{i,2}$	$\boldsymbol{c}_{i,3}$	$w_{i,3}$
1	$(\frac{\sqrt{3}}{2}r, -\frac{r}{2}, 0)$	1	$(\frac{11\sqrt{3}}{18}r, -\frac{1}{6}r, \frac{2}{9}r)$	$\frac{5}{6}$	$(\frac{\sqrt{3}}{2}r, -\frac{1}{2}r, \frac{2}{3}r)$	$\frac{\sqrt{2}+1}{3}$
2	$(\frac{\sqrt{3}}{2}r, \frac{1}{2}r, 0$	1	$(\frac{\sqrt{3}}{2}r, \frac{1}{2}r, \frac{2}{3}r)$	$\frac{\sqrt{2}+1}{3}$	$(\frac{11\sqrt{3}}{18}r, \frac{1}{6}r, \frac{2}{9}r)$	$\frac{5}{6}$
3	$(0, 0, r)$	1	$(\frac{\sqrt{3}}{3}r, -\frac{1}{3}r, r)$	$\frac{\sqrt{2}+1}{3}$	$(\frac{\sqrt{3}}{3}r, \frac{1}{3}r, r)$	$\frac{\sqrt{2}+1}{3}$

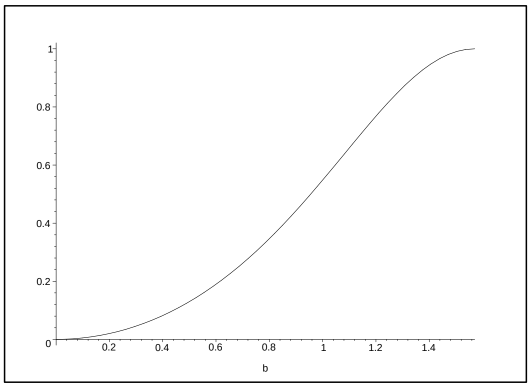


Figure 3.20: A measurement of the height of the gap bounded by the bottommost boundary curve of the derived patch on the sphere and the equator plane, as a function of the patch span b, given in radians.

3.7 Concluding Remarks

In this chapter we extended the B–spline representation of PS–spline surfaces to non uniform rational PS–spline surfaces (NURPS). First we reviewed triangular rational Bernstein–Bézier surfaces in Section 3.2. These have the convex hull property and are affine invariant, and also enjoy projective invariance. In their quadratic form they allow to represent patches on quadrics exactly, under certain conditions stated in Section 3.2.3. We also mentioned the rational de Casteljau algorithm (Section 3.2.1) which resolves numerical problems that could occur when calculating with Bernstein–Bézier surfaces in the homogeneous space. We used it to calculate a point on the surface, a tangent plane and a normal vector. An alternative for classical subdivision of rational Bernstein–Bézier surfaces was given in Section 3.2.2. It is better suited for displaying the surfaces graphically. We also

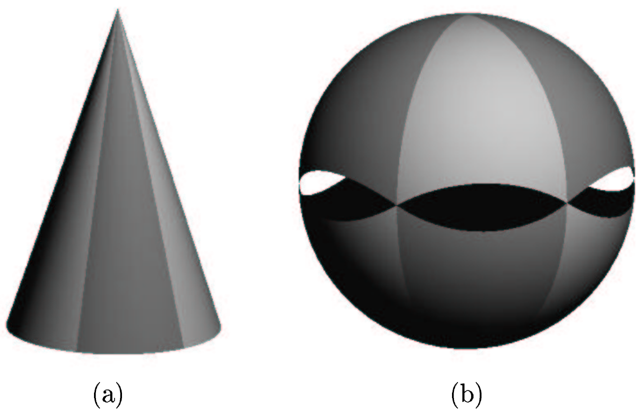


Figure 3.21: (a) A cone represented as a NURPS surface. (b) An incomplete NURPS representation of the sphere. For this particular case, a span $b = \frac{\pi}{6}$ has been used.

mentioned that shape parameters can't be defined for rational Bernstein–Bézier surfaces in the way they can for rational Bézier curves; therefore an intuitive manipulation of the weights by geometric means is not obvious.

Next, we introduced NURPS surfaces in Section 3.3. They enjoy the convex hull and affine as well as projective invariance properties, but also have local control. We have given an algorithm for calculating the piecewise rational Bernstein–Bézier representation of a NURPS surface in Section 3.3.2 and used it to determine a point on the surface, a tangent plane and a normal vector. Furthermore, this algorithm illustrated that any calculation scheme for PS–splines that uses convex combinations only, can be generalized to a numerically stable rational algorithm for NURPS, if we employ the idea behind the rational de Casteljau algorithm: avoid to work in the homogeneous space by rearranging the calculations. But the algorithm also is useful for the study of modeling special effects. Indeed, Section 3.4 shows how the NURPS weights can be used as a shape control tool. We defined control triangles and proved their tangency to the surface in Section 3.4.1. The weights were also given a geometric interpretation: there is a one–one connection between the weights of a control triangle T_i and the barycentric coordinates of the tangent point with respect to T_i. This allows to move the tangent point to any location within the control triangle: shape parameters for NURPS were born (Section 3.4.2). The weights can also be used to model local planar effects on the surface, as shown in Section 3.4.3. Hereto, we let the weights of a certain control triangle be large compared with those of the others. Finally the modeling of cusps, corners and curved corners was demonstrated in Section 3.4.4. They do not involve the weights, but are achieved by degenerate control triangles.

A given triangular rational Bernstein–Bézier surface can be converted to a NURPS surface in a number of ways, depending on the choice of the PS–triangles. In Section 3.5.1 we gave an identical representation, useful for the functional case, where the NURPS weights and control points coincide with those of the Bernstein–Bézier representation. The optimal representation on an equilateral domain triangle was given in Section 3.5.2. It can be used for parametric surfaces. Both algorithms were given in their rational, numerically stable form. The latter conversion scheme was applied in Section 3.6 where we represented quadrics as NURPS. First, rational quadratic Bernstein–Bézier patches on quadrics were derived in a constructive manner, relying on the fact that conic sections can be represented exactly as rational quadratic Bézier curves, and with objective in mind to represent the quadrics as a whole, using several isometric patches. These were then converted to NURPS surfaces. We showed that a complete cylinder and cone can be represented as composite NURPS surfaces in Sections 3.6.1 and 3.6.2. The whole sphere can not be represented as a piecewise rational quadratic Bernstein–Bézier surface, but we gave an incomplete NURPS representation in Section 3.6.3, where the user has a certain degree of control on the size of the surface gaps: increasing the number of patches of the composite form decreases the size of the gaps.

Though these representations are useful in CAD systems, a certain drawback is that they are composite. For instance, a cylinder was obtained by eight isometrical patches, but not as a single NURPS surface. This is due to the fact that the homogeneous component of the complete cylinder, represented as a piecewise rational Bernstein–Bézier surface is not C^1–continuous. Another drawback is that the sphere representation is incomplete. But, we will use this as a case for illustrating our algorithm for the polygonal hole problem in Chapter 5, where a PS–spline surface is determined that smoothly fills in a hole, given by a set of bounding surfaces.

Chapter 4

Uniform PS–splines and wavelets

4.1 Introduction

This chapter focuses on Powell–Sabin splines on regular triangulations, composed of isometrical equilateral triangles, the so–called uniform triangulation. These UPS–splines have certain advantages compared with PS–splines on arbitrary triangulations. Due to the high degree of symmetry, a PS–refinement is known in advance, and the form of the PS–triangles can be fixed (Section 4.2.2). Thus, the calculation of PS–triangles containing the PS–points at each vertex, is no longer required. Also, the basis functions are known beforehand. This enables one to implement a number of algorithms more efficiently: we will have a close look at the interpolation problem, the integration of PS–splines and the calculation of the Bernstein–Bézier representation in Section 4.3. Moreover, we present a subdivision scheme for UPS–splines in Section 4.4, that calculates the representation of a given spline $s(u)$ on a refinement of Δ in an efficient and numerically stable way. This result is generalized to a k–subdivision scheme along the edges of Δ. We consider two applications of this algorithm: a more theoretical one and a very practical one.

The first application concerns the construction of UPS–spline wavelets. The advantages of B–spline wavelets in signal and image processing have been recognized in the literature [52, 73, 74]. Their simple, explicit analytical form facilitates their manipulation. They have excellent approximation properties, allowing to get a smooth representation of a discrete signal. On the other hand, non–separable wavelet transforms on e.g. hexagonal grids have better orientation properties than their tensor product counterparts [11, 40, 49, 67, 76], which favor horizontal, vertical and diagonal features of the data. As such, they have found application in medical imaging [48, 55]. Little has been published yet on non–separable wavelets using lifting; the general ideas however are exposed in [45].

In Section 4.5, it is shown that the UPS–spline basis functions are translates and dilates of three functions. A multiresolution analysis is defined. Then, a UPS–spline wavelet transform using lifting is presented. It yields non–separable B–spline wavelets on regular triangular grids. It shall be built with the idea in mind to decompose a given UPS–spline into a low–frequency component and a number of high–frequency components. We also consider an algorithm for noise removal, based on the UPS–spline wavelet transform.

The second, more practical application of UPS–spline subdivision involves the graphical display of UPS–spline surfaces and is presented in Section 4.6. By repeated subdivision, it is possible to calculate a great number of points on a given UPS–spline surface, as well as the normal vectors at those points. A three–direction wireframe or a rendered surface can then be generated.

Finally, we note that parametric uniform Powell–Sabin splines allow to represent surfaces having three, four, five or six edges, which are the most common cases.

4.2 Uniform PS–splines

4.2.1 Isometrical lattices and uniform triangulations

Recall that the triangulation of a polygonal domain Ω is denoted "Δ". For uniform triangulations, it is useful to extend this notation with a level of refinement (or resolution). To do so, we will look at the vertices V_i, the triangles $\rho_{i,j,k}$ and the uniform triangulation Δ as parts of a refinable isometrical grid. The notation is based in part on [45]. Consider the isometrical grid constituted by three infinite sets of parallel lines which are equally spaced at distance B and are at angles $\pi/3$ to each other, the so–called grid lines (see Figure 4.1). The intersection points of these lines form the point lattice $\Gamma B\mathbb{Z}^2$ where Γ is the invertible 2×2 matrix

$$\Gamma = \begin{bmatrix} 1 & 1/2 \\ 0 & \sqrt{3}/2 \end{bmatrix}. \tag{4.1}$$

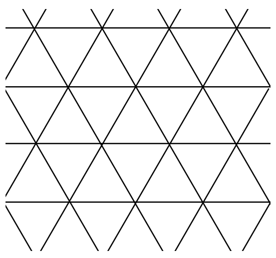


Figure 4.1: The isometrical grid.

The grid creates a number of equilateral triangular cells $\rho_{i,j,k}$, whose vertices are $V_i, V_j, V_k \in \Gamma B\mathbb{Z}^2$, and where each edge has length B. In order to simplify notation, we will refer to the grid lines, the point lattice and the set of triangular cells using one symbol: $\mathcal{K}$.

Definition 4.1 *The isometrical lattice $\mathcal{K} = \Gamma B\mathbb{Z}^2$, constituted by the grid lines, is defined as the set of lattice points V_i and triangular cells $\rho_{i,j,k}$.*

So now it is safe to write not only $V_i \in \mathcal{K}$, but also $\rho_{i,j,k} \in \mathcal{K}$, etc. If D is the 2×2 matrix with integer coefficients

$$D = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \tag{4.2}$$

we can find a sublattice of $\mathcal{K}$ as $D\mathcal{K}$. There are $|D| - 1 = 3$ distinct cosets of this first sublattice $D\mathcal{K}$ each of the form $D\mathcal{K} + k_i$, with $k_i \in \Gamma B\mathbb{Z}^2$. We know therefore that

$$\mathcal{K} = \bigcup_{i=0}^{3} (D\mathcal{K} + k_i), \tag{4.3}$$

where $k_0 = 0$ and the cosets in the union are disjoint. Figure 4.2 shows the decomposition of $\mathcal{K}$ into its four disjoint sublattices according to (4.3).

Definition 4.2 *A lattice polygon is a polygon whose vertices are points of a point lattice.*

We will further restrict this definition to lattice polygons whose edges are on the grid lines of $\mathcal{K}$. Now consider such a lattice polygon $\Omega \subset \mathbb{R}^2$ on $\mathcal{K}$ with

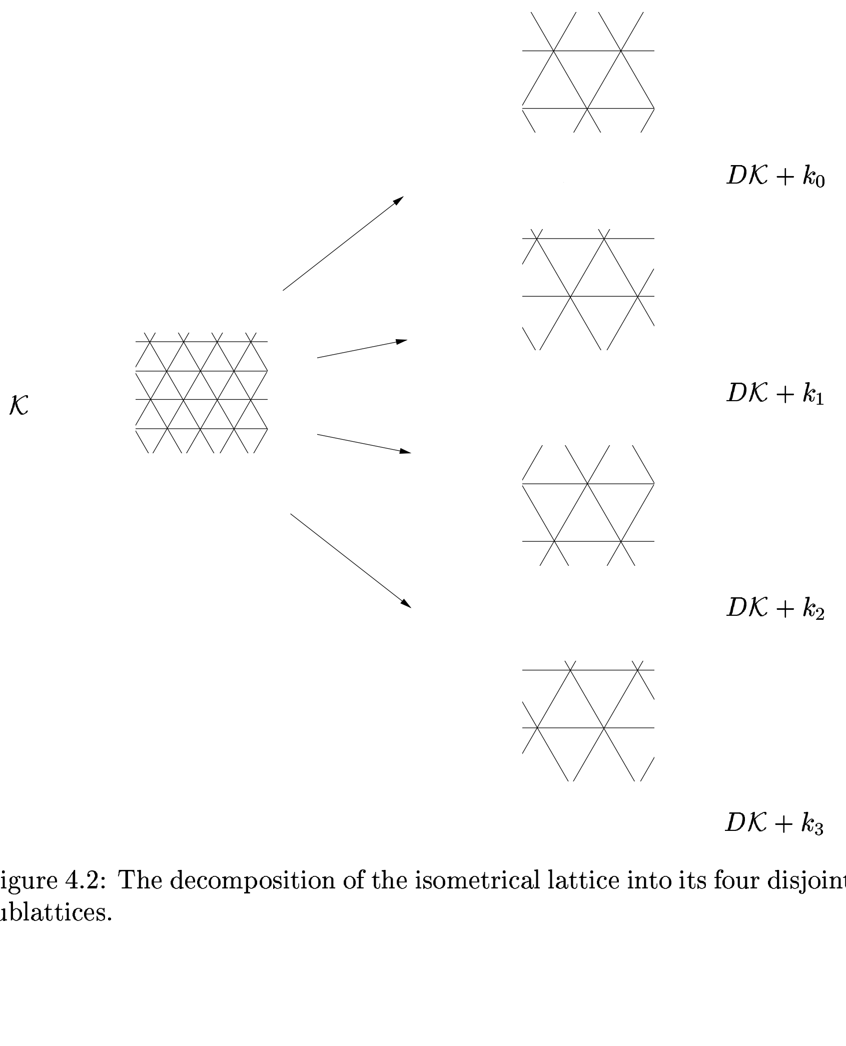


Figure 4.2: The decomposition of the isometrical lattice into its four disjoint sublattices.

boundary $\partial\Omega$. The triangular cells $\rho_{i,j,k} \in \mathcal{K}$ that are inside Ω, constitute a uniform triangulation Δ of Ω, that is, a triangulation entirely constituted of equilateral triangles. We shall refer to the length B as the base of this triangulation.

The isometrical lattice has been split up into sublattices by premultiplication with D. It can be refined by premultiplication with D^{-1}: the lattice $D^{-1}\mathcal{K}$ has its grid lines at distance $B/2$, and each triangular cell of $\mathcal{K}$ contains four cells of $D^{-1}\mathcal{K}$. We shall adopt this notation to denote refinements of uniform triangulations. So $D^{-1}\Delta$ denotes a refinement of a given triangulation Δ of a lattice polygon Ω. This refinement can be obtained by mid–edge subdivision: a new vertex is introduced in the middle of each edge, and each triangle is split into four subtriangles. Figure 4.3 shows a triangulation and its refinement $D^{-1}\Delta$.

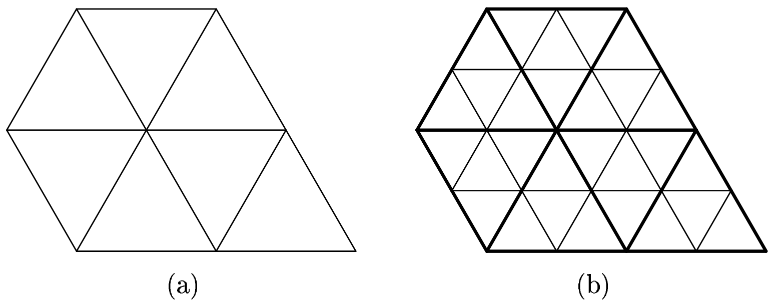


Figure 4.3: (a) A triangulation Δ and (b) its refinement $D^{-1}\Delta$.

4.2.2 A fixed form for the PS–triangles

As has already been pointed out in Section 3.5.2, a valid PS–refinement of a uniform triangulation (see Definition 2.16) is easily constructed:

Definition 4.3 *A uniform PS–refinement or UPS–refinement corresponds to the PS–refinement obtained by drawing the bisector of each angle.*

This is illustrated in Figure 4.4. Given such a UPS–refinement Δ^*, the definition of uniform PS–splines follows immediately.

Definition 4.4 *A Uniform Powell–Sabin (UPS–) spline is a PS–spline as given by (2.31) on a uniform triangulation Δ of a lattice polygon Ω on $\mathcal{K}$ with uniform PS–refinement Δ^*.*

The extension to parametric UPS–spline surfaces is also straightforward.

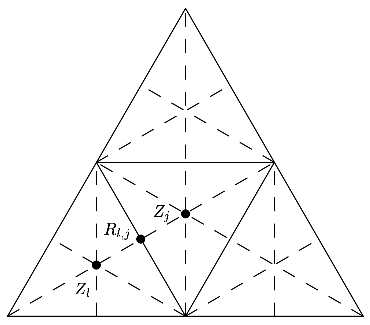


Figure 4.4: The PS–refinement of a uniform PS–triangulation (solid lines) can be found by drawing the bisector of each angle (dashed lines).

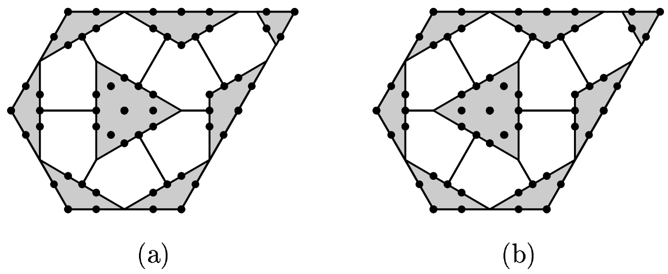


Figure 4.5: (a) The optimal PS–triangles for molecule numbers one, two, three and six. (b) A second solution for the vertex with molecule number six.

Since each angle in a uniform PS triangulation is $\pi/3$, only a limited number of molecules are possible. The maximum molecule number in Δ is six. Furthermore, all molecules having the same molecule number are identical up to a rotation with a multiple of $\pi/3$. The optimal PS–triangles can be calculated once and for all for each possible molecule and stored in a look–up table. Since there are, up to a rotation, only six possible molecules, this table has only six entries. To illustrate the idea, Figure 4.5 shows the optimal PS–triangles for the molecule numbers one, two, three and six, as obtained by solving the quadratic programming problem [19]. For molecule number six, two optimal solutions exist: compare Figure 4.5 (a) and (b). In the sequel, we will use the second one, but the choice is arbitrary. Now denote with $t(m_i)$ the optimal PS–triangle corresponding to a vertex having molecule number m_i. It is clear that any $t(k)$ with $k > m_i$ is also a valid PS–triangle for that vertex, although not optimal. As we will see, the use of $t(6)$ for all vertices will have a number of particular advantages. So rather than looking up the optimal PS–triangles in a table, we will fix from now on the PS–triangles to $t(6)$ as on Figure 4.6(a).

Property 4.1 *The fixed PS–triangle $t(6)$ is equilateral and has its vertices halfway the edges of $\mathcal{K}$. The corresponding vertex is at the barycenter of $t(6)$. So, referring to Figure 4.6(b), we have that*

$$Q_i = \frac{1}{2}(V + V_{2i-1}), \quad i = 1, 2, 3, \tag{4.4}$$

$$V = \frac{1}{3}(Q_1 + Q_2 + Q_3). \tag{4.5}$$

4.3 On calculating with UPS–splines

The computation schemes for PS–splines (Section 2.3.3) and parametric PS–spline surfaces (Section 2.3.5) can be implemented more efficiently for UPS–spline and parametric UPS–spline surfaces. We will look into detail at solving the interpolation problem, calculating the Bernstein–Bézier representation of a UPS–spline and integrating a UPS–spline.

4.3.1 The interpolation problem for UPS–splines

Recall that there is a one–one connection between the coordinates of the PS–triangle points and the triplets $(\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j})$ that denote the function value and first order partial derivative values of $B_i^j(u)$ at V_i. Since the form of the PS–triangles is fixed, so are the values of $(\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j})$ according to the next property.

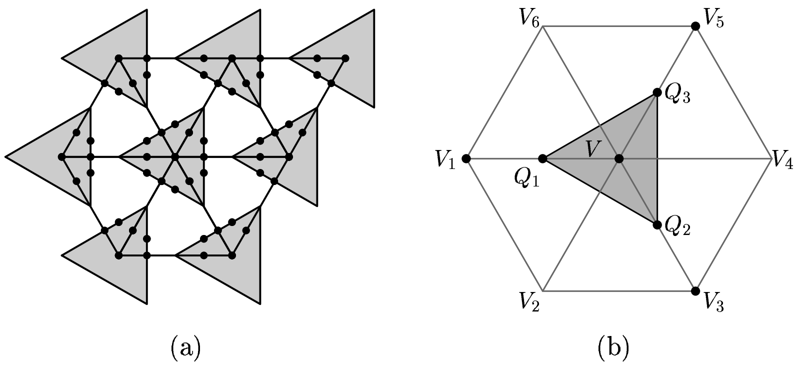


Figure 4.6: (a) A fixed form for the PS–triangles that contains the PS–points. (b) The fixed PS–triangle is equilateral and has its vertices halfway the edges of $\mathcal{K}$.

Property 4.2 *The triplets* $(\alpha_{i,j}), (\beta_{i,j}), (\gamma_{i,j}), j = 1, 2, 3$ *do not depend on the vertex* V_i*:*

$$(\alpha_{i,1}, \alpha_{i,2}, \alpha_{i,3}) = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right), \tag{4.6}$$

$$(\beta_{i,1}, \beta_{i,2}, \beta_{i,3}) = \left(-\frac{4}{3B}, \frac{2}{3B}, \frac{2}{3B}\right), \tag{4.7}$$

$$(\gamma_{i,1}, \gamma_{i,2}, \gamma_{i,3}) = \left(0, -\frac{2\sqrt{3}}{3B}, \frac{2\sqrt{3}}{3B}\right). \tag{4.8}$$

Consider the interpolation problem (2.30). When the function value and first order derivative values at the vertices V_i are given, the B–spline coefficients of the interpolating UPS–spline can be found by simplifying (2.45):

$$\begin{aligned} c_{i,1} &= f_i - \frac{B}{2} f_{x,i} \\ c_{i,2} &= f_i + \frac{B}{4} f_{x,i} - \frac{\sqrt{3}B}{4} f_{y,i} \\ c_{i,3} &= f_i + \frac{B}{4} f_{x,i} + \frac{\sqrt{3}B}{4} f_{y,i} \end{aligned} \tag{4.9}$$

4.3.2 Calculating the Bernstein–Bézier representation of a UPS–spline

The computation of the Bernstein–Bézier representation (which is displayed schematically in Figure 4.7) of a UPS–spline on $\rho_{i,j,k}$ by (2.54)–(2.60) simplifies to

$$
\begin{aligned}
s_i &= \frac{1}{3}(c_{i,1}+c_{i,2}+c_{i,3}) && (4.10)\\
u_i &= \frac{1}{2}(c_{i,2}+c_{i,3}) && (4.11)\\
v_i &= \frac{2}{3}c_{i,3}+\frac{1}{6}(c_{i,1}+c_{i,2}) && (4.12)\\
w_i &= \frac{1}{3}c_{i,2}+\frac{2}{3}c_{i,3} && (4.13)\\
r_k &= \frac{1}{2}(u_i+v_j) && (4.14)\\
\theta_k &= \frac{1}{2}(w_i+w_j) && (4.15)\\
\omega &= \frac{1}{3}(w_i+w_j+w_k). && (4.16)
\end{aligned}
$$

Note that Δ contains triangles $\rho_{i,j,k}$ pointing upward (Figure 4.7 top) as well as downward (bottom of Figure 4.7). Since these triangles, accommodated with PS–triangles and Bézier ordinates, are each others mirror image with respect to the edge V_iV_j, the formulae are valid for both cases.

4.3.3 Integration of UPS–splines

The Bézier ordinates of a UPS–spline basis function $B_i^l(u), 1 \le l \le 3$, can be found using (4.10)–(4.16), with all $c_{i,m}=0$, except for $m=l : c_{i,l}=1$. This allows to simplify expression (2.63) for the integral of a PS–spline on $\rho_{i,j,k}$. We worked this out for the three basis functions at V_i, and the result is valid for both situations in Figure 4.7:

$$
\begin{aligned}
\int_{\rho_{i,j,k}} B_i^1(u)du &= \frac{\sqrt{3}B^2}{144} && (4.17)\\
\int_{\rho_{i,j,k}} B_i^2(u)du &= \frac{\sqrt{3}B^2}{36} && (4.18)\\
\int_{\rho_{i,j,k}} B_i^3(u)du &= \frac{7\sqrt{3}B^2}{144} && (4.19)
\end{aligned}
$$

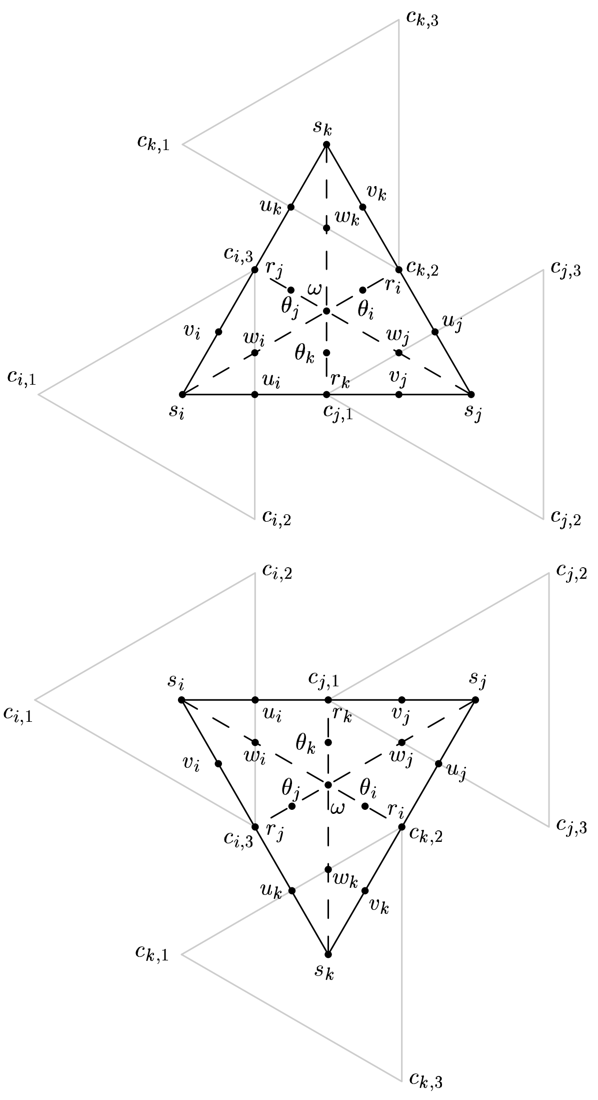


Figure 4.7: Schematic representation of the Bézier ordinates of a triangle $\rho_{i,j,k} \in \Delta$.

The integrals of the basis functions at the other vertices have the same values, but the upper index has to be rotated. For example,

$$\int_{\rho_{i,j,k}} B_j^1 = \frac{7\sqrt{3}B^2}{144}, \quad \text{etc.} \tag{4.20}$$

The integral of a basis function $B_i^l(u)$ over molecule M_i can now be derived:

$$\int_{M_i} B_i^l(u)du = \frac{\sqrt{3}B^2}{6}, \quad l = 1, 2, 3. \tag{4.21}$$

The following algorithm computes the integral of a UPS–spline as a weighted sum of the B–spline coefficients:

$$\int_{\Omega} s(u)dxdy = \sum_{i=1}^{n}\sum_{j=1}^{3} w_{i,j}c_{i,j}. \tag{4.22}$$

```
1. Set weights[] = $\{\frac{1}{36}, \frac{7}{36}, \frac{1}{9}\}$ as an array with zero–based index.

2. Determine the area $A = \frac{\sqrt{3}}{4}B^2$ of a triangle $\rho \in \Delta$.

3. Set int = 0

4. For each vertex $V_i, i = 1, \ldots, n$

   (a) For $j = 1, 2, 3$

       i. Set w = 0
       ii. For each triangle $\rho_k \in M_i$
           A. Determine, among the three possibilities, the relative ori-
              entation of $Q_{i,j}$ and $\rho_k$. This yields an index $l, 0 \leq l < 3$.
           B. Set w = w + weights[l]
       iii. Set int = int + w * $c_{i,j}$

5. Set int = int * $A$
```

The appropriate weights in (4.22) can also be determined beforehand and be referenced from a table. Figure 4.8 shows the different configurations according to molecule numbers one, two, three and six. The other situations can be mapped onto one of these by mirroring or rotating the molecule.

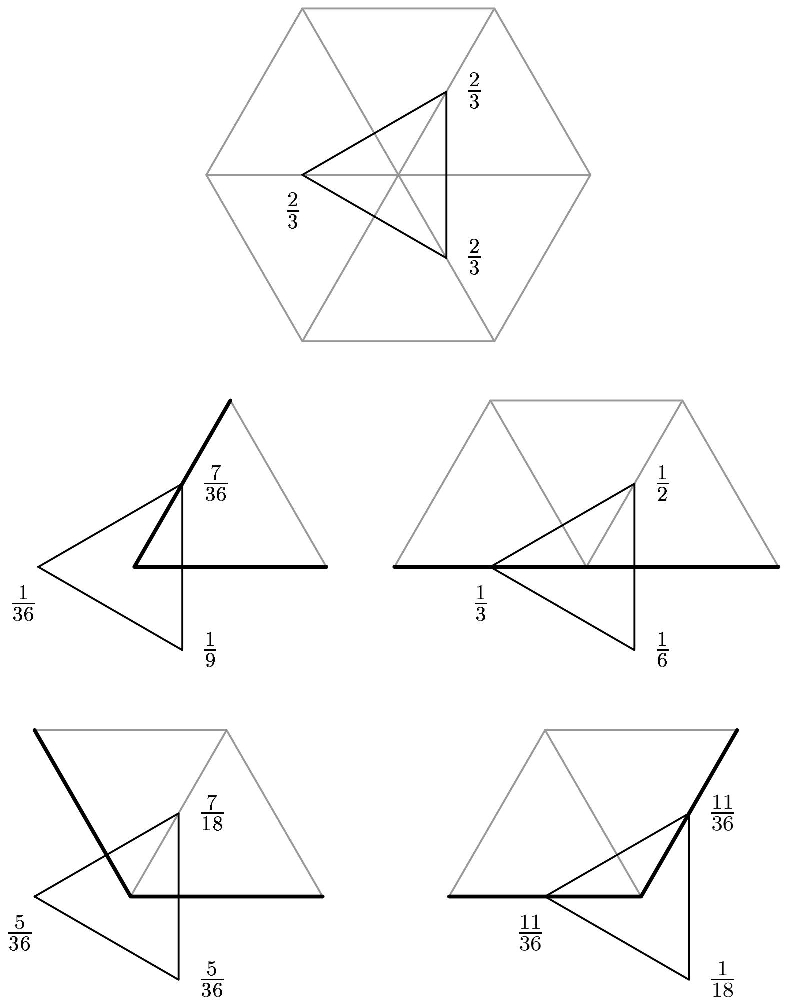


Figure 4.8: The integral of a UPS–spline can be calculated as a weighted sum of the B–spline coefficients. The weights are depicted here for each possible configuration according to molecule numbers 1, 2, 3 and 6. The weighted sum should be post–multiplied with the area of a triangle $\rho_j \in \Delta$ in order to obtain the integral.

4.4 Subdivision of UPS–splines

In this section we will derive a subdivision rule for UPS–splines: given $s(u) \in S_2^1(\Delta^*)$, find the representation of $s(u)$ on a refinement of Δ, that is, $s'(u) \in S_2^1(D^{-1}\Delta^*)$. The objective is to write the B–spline coefficients of $s'(u)$ as convex combinations of those of $s(u)$. To do so, we will first derive such a relation between the corresponding PS–triangle points in the domain plane, and then generalize this result to the control points. Indeed, since the form of the PS–triangles is fixed, they can be determined immediately for the refined triangulation $D^{-1}\Delta$. For example, Figure 4.9 shows an initial triangulation with its PS–triangles and the PS–triangles of the refinement $D^{-1}\Delta$. There are two cases to consider: the old vertices $V_i \in \Delta$ (Section 4.4.1), and the new vertices $V_i \in D^{-1}\Delta \setminus \Delta$ (Section 4.4.2). In Section 4.4.3 we will look for a more general solution on the edges of Δ.

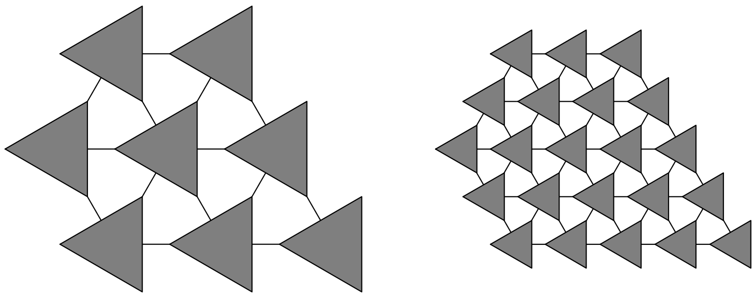

(a) (b)

Figure 4.9: (a) The fixed form of the PS–triangles for an initial triangulation and (b) for its refinement $D^{-1}\Delta$.

4.4.1 Subdivision rules for the old vertices

Consider a vertex $V_i \in \Delta$ (Figure 4.10), its PS–triangle before subdivision $t_i(Q_{i,1}, Q_{i,2}, Q_{i,3})$ and after subdivision $t_i'(Q_{i,1}', Q_{i,2}', Q_{i,3}')$. The edges of $D^{-1}\Delta \setminus \Delta$ are indicated in dashed lines. Both t_i and t_i' have a fixed form, and their PS–triangle points can be found from (4.4). It is therefore easy to write the vertices of t_i' in terms of those of t_i, e.g.,

$$Q_{i,1}' = \frac{1}{2}\left(\frac{1}{3}\left(Q_{i,1} + Q_{i,2} + Q_{i,3}\right) + Q_{i,1}\right). \tag{4.23}$$

The coordinates of $Q_{i,k}'$ are in fact the x– and y–coordinates of the control points $\boldsymbol{c}_{i,k}'$. Now, since the corresponding control triangles T_i and T_i' are coplanar, the z–coordinate of $\boldsymbol{c}'_{i,k}$ is found as the distance between $Q'_{i,k}$ and the intersection point of a line in the z–direction through $Q'_{i,k}$, and T_i. But then the barycentric coordinates of $\boldsymbol{c}'_{i,k}$ with respect to T_i are the same as those of $Q'_{i,k}$ with respect to t_i. Therefore, the same expression as (4.23) holds for the control points, so

$$\boldsymbol{c}'_{i,k} = \frac{2}{3}\boldsymbol{c}_{i,k} + \frac{1}{6}\left(\boldsymbol{c}_{i,k_3+1} + \boldsymbol{c}_{i,(k+1)_3+1}\right), \tag{4.24}$$

where $i_3 = i \bmod 3$.

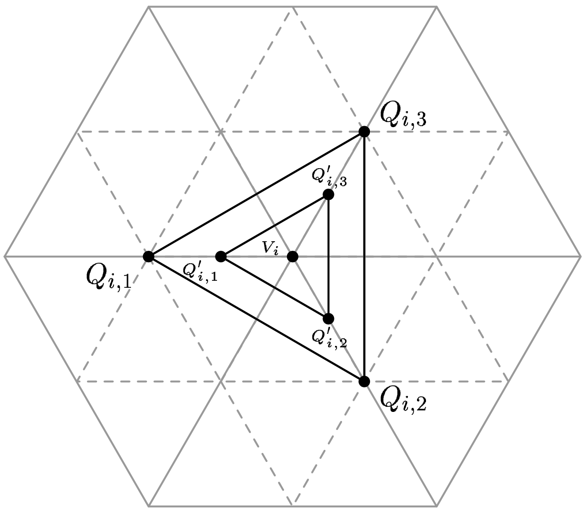


Figure 4.10: Subdivision of UPS–splines at an old vertex $V_i \in \Delta$.

4.4.2 Subdivision rules for the new vertices

Consider an edge $V_iV_j \in \Delta$, the new vertex $V_{i,j} \in D^{-1}\Delta \setminus \Delta$ which is introduced in the middle of this edge by subdivision, and the corresponding PS–triangles $t_i(Q_{i,1}, Q_{i,2}, Q_{i,3})$, $t_j(Q_{j,1}, Q_{j,2}, Q_{j,3})$ and $t_{i,j}(Q_{i,j,1}, Q_{i,j,2}, Q_{i,j,3})$. Again, the PS–triangle points of $t_{i,j}$ can be calculated using (4.4) and they can be expressed in terms of the PS–triangle points of t_i and t_j, e.g.,

$$Q_{i,j,3} = \frac{1}{2}\left(\frac{1}{2}\left(Q_{i,3} + Q_{j,3}\right) + Q_{j,1}\right) \tag{4.25}$$

The coordinates of $Q_{i,j,k}$ are the x– and y–coordinates of the control points associated with $s'(u)$. The same relations hold for the z–coordinates, as will be confirmed by a proof in a more general context further in this text. Therefore, the control points of $s'(u)$ can be calculated by the following subdivision rule:

$$\begin{aligned} \boldsymbol{c}_{i,j,1} &= \frac{1}{2}\left(\boldsymbol{c}_{i,2}+\boldsymbol{c}_{i,3}\right) && (4.26)\\ \boldsymbol{c}_{i,j,2} &= \frac{1}{2}\boldsymbol{c}_{j,1}+\frac{1}{4}\left(\boldsymbol{c}_{i,2}+\boldsymbol{c}_{j,2}\right) && (4.27)\\ \boldsymbol{c}_{i,j,3} &= \frac{1}{2}\boldsymbol{c}_{j,1}+\frac{1}{4}\left(\boldsymbol{c}_{i,3}+\boldsymbol{c}_{j,3}\right) && (4.28) \end{aligned}$$

These combinations are depicted on Figure 4.11 in dashed lines.

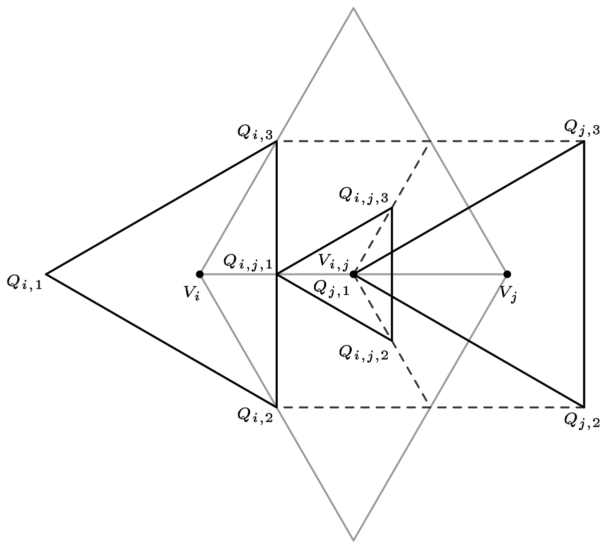


Figure 4.11: Subdivision of UPS–splines at a new vertex $V_{i,j} \in D^{-1}\Delta \setminus \Delta$.

Figure 4.12 shows an example of a UPS–spline surface before subdivision (left) and after subdivision (right). The number of control triangles increases through subdivision, and they get closer to the surface.

4.4.3 k–Subdivision along the edges of Δ

In this section we will look for a more general solution for the subdivision problem along the edges of Δ. Suppose that the initial triangulation is

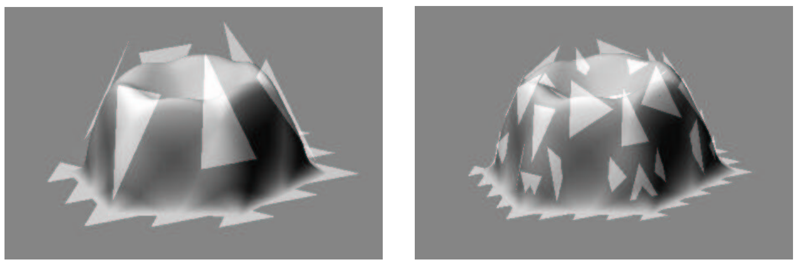


Figure 4.12: A UPS–spline surface before subdivision (left) and after subdivision (right).

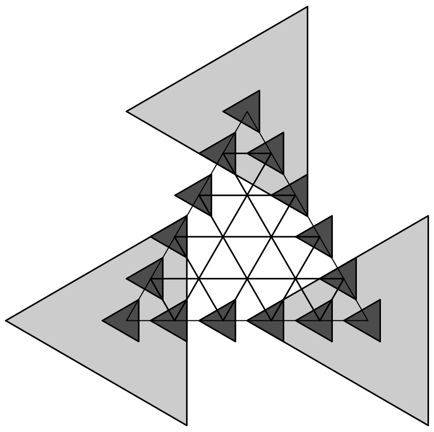


Figure 4.13: k-Subdivision along the edges with $k = 5$. The darker control triangles (after refinement) are to be found from the lighter ones (before refinement).

refined by splitting each edge into k pieces, $k \geq 2$, such that the base of the refined triangulation becomes $\frac{B}{k}$. Our aim is then to find a subdivision rule that calculates the control triangles along the edges of Δ only. For example, Figure 4.13 illustrates a refinement of an initial triangle with $k = 5$. We would like to find the B–spline coefficients corresponding to the darker shaded PS–triangles as convex combinations of the original (lighter) ones. To do so, consider the Bernstein–Bézier representation of a given UPS–spline $s(u)$ on a triangle $\rho_{1,2,3} \in \Delta$, for the two Bézier subtriangles along the edge V_1V_2, namely $\rho^*(V_1, R_{1,2}, Z)$ and $\rho^*(V_2, R_{1,2}, Z) \in \Delta^*$. Figure 4.14 denotes the Bézier subtriangles and –ordinates schematically. Let $\bar{c}_{i,j}, i = 1,2,\ j = 1,2,3$ denote the initial B–spline coefficients at V_1 and V_2. We want to calculate the B–spline coefficients of the refined UPS–spline at $W_l = \frac{k-l}{k}V_1 + \frac{l}{k}V_2$, that is, $c_{l,j}$, $l = 0, \ldots, k$, $j = 1,2,3$. If we know the values of $s(W_l)$, $\frac{\partial s}{\partial x}(W_l)$, $\frac{\partial s}{\partial y}(W_l)$, then the solution is given by (4.9) with B replaced by $\frac{B}{k}$, the base of the refined triangulation. So the problem is reduced to finding the values of $s(u)$ and its first order partial derivatives at W_l. Suppose that $0 \leq l \leq \frac{k}{2}$, such that W_l belongs to $\rho^*(V_1, R_{1,2}, Z)$. Otherwise, W_l belongs to $\rho^*(V_2, R_{1,2}, Z)$ and a similar derivation can be given. First we apply Bézier subdivision on $\rho^*(V_1, R_{1,2}, Z)$ at the point W_l: ρ^* is split up into two triangles $\rho^*(W_l, Z, V_1)$ and $\rho^*(W_l, Z, R_{1,2})$ (see Figure 4.14, top). The Bézier ordinates of $s(u)$ on the latter triangle are denoted b_λ, $|\lambda| = 2$ (see Figure 4.15). We are interested in the values of $b_{2,0,0}$, $b_{1,0,1}$ and $b_{1,1,0}$. These can be obtained by the de Casteljau algorithm:

$$\begin{aligned} b_{2,0,0} &= \frac{(k-2l)^2}{k^2}S_1 + \frac{4kl - 6l^2}{k^2}L_1 + \frac{2l^2}{k^2}L_2 && (4.29) \\ b_{1,1,0} &= \frac{k-l}{k}L_1'' + \frac{l}{k}L_2'' && (4.30) \\ b_{1,0,1} &= \frac{k-l}{k}L_1 + \frac{l}{k}L_2 && (4.31) \end{aligned}$$

We know that $s(W_l) = b_{2,0,0}$, so, now, we only need to find the partial derivatives. Let $b(\tau)$ denote the Bernstein–Bézier representation of $s(u)$ on $\rho^*(W_l, Z, R_{1,2})$. Then

$$s(u) = b(\tau) = \sum_{|\lambda|=2} b_\lambda B_\lambda^2(\tau_1, \tau_2, 1 - \tau_1 - \tau_2), u = (x,y) \in \rho^*(W_l, Z, R_{1,2}) \qquad (4.32)$$

and taking account of (2.1), application of the chain rule yields:

$$\frac{\partial s}{\partial x}(u) = \left(\frac{-2k}{(k-2l)B}\right) \sum_{|\lambda|=2} \frac{\partial B_\lambda^2}{\partial \tau_1}(\tau_1, \tau_2, 1 - \tau_1 - \tau_2), \qquad (4.33)$$

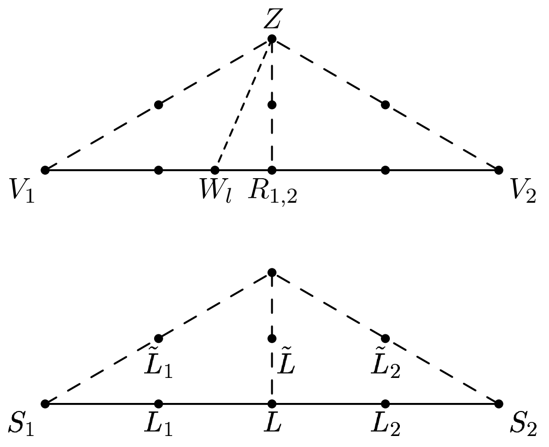


Figure 4.14: The Bernstein–Bézier representation of $s(u)$ on the Bézier subtriangles along the edge V_1V_2. Top: the Bézier subtriangles $\rho^*(V_1, R_{1,2}, Z)$ and $\rho^*(V_2, R_{1,2}, Z)$. Bottom: schematic representation of the Bézier ordinates.

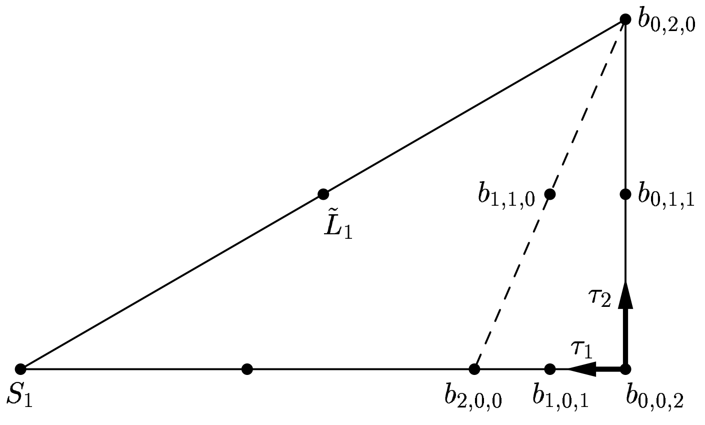


Figure 4.15: Bézier Subdivision at W_l: schematic representation of the Bézier ordinates.

$$\frac{\partial s}{\partial y}(u) = \left(\frac{2\sqrt{3}}{B}\right) \sum_{|\lambda|=2} \frac{\partial B^2_\lambda}{\partial \tau_2}(\tau_1, \tau_2, 1-\tau_1-\tau_2), \tag{4.34}$$

so the partial derivative values at V_W are

$$\begin{aligned}\frac{\partial s}{\partial x}(W_l) &= \frac{-4k}{(k-2l)B}(b_{2,0,0} - b_{1,0,1}), &(4.35)\\ \frac{\partial s}{\partial y}(W_l) &= \frac{4\sqrt{3}}{B}(b_{1,1,0} - b_{1,0,1}). &(4.36)\end{aligned}$$

Substituting these values in (4.9) and taking account of (4.10)–(4.16), the B–spline coefficients of the refined UPS–spline at W_l can be written in terms of the original ones at V_1 and V_2:

$$\begin{aligned}c_{l,1} &= \frac{(k-2l)(k-2l+2)}{3k^2}\bar{c}_{1,1} + \frac{k^2+2kl-5l^2-k+5l}{3k^2}(\bar{c}_{1,2}+\bar{c}_{1,3})\\ &+ \frac{4l(l-1)}{3k^2}\bar{c}_{2,1} + \frac{l(l-1)}{3k^2}(\bar{c}_{2,2}+\bar{c}_{2,3}), &(4.37)\\ c_{l,2} &= \frac{(k-2l)(k-2l-1)}{3k^2}\bar{c}_{1,1} + \frac{k^2+2kl-5l^2+2k-4l}{3k^2}\bar{c}_{1,2}\\ &+ \frac{k^2+2kl-5l^2-k-l}{3k^2}\bar{c}_{1,3} + \frac{2l(2l+1)}{3k^2}\bar{c}_{2,1} + \frac{l(l+2)}{3k^2}\bar{c}_{2,2}\\ &+ \frac{l(l-1)}{3k^2}\bar{c}_{2,3}, &(4.38)\\ c_{l,3} &= \frac{(k-2l)(k-2l-1)}{3k^2}\bar{c}_{1,1} + \frac{k^2+2kl-5l^2-k-l}{3k^2}\bar{c}_{1,2}\\ &+ \frac{k^2+2kl-5l^2+2k-4l}{3k^2}\bar{c}_{1,3} + \frac{2l(2l+1)}{3k^2}\bar{c}_{2,1} + \frac{l(l-1)}{3k^2}\bar{c}_{2,2}\\ &+ \frac{l(l+2)}{3k^2}\bar{c}_{2,3}, \quad l = 0, 1, \ldots, \frac{k}{2}. &(4.39)\end{aligned}$$

For $k = 2$ and $l = 0, 1$, these formulae confirm the subdivision rule that we earlier derived in (4.24) and (4.26)–(4.28). In a straightforward manner we can verify that the sum of the coefficients in each of the right hand sides is equal to one, so the combinations are barycentric. In order to prove that these combinations are also convex, we have to check the sign of each of the numerators of the coefficients for $k \geq 2$, $0 \leq l \leq \frac{k}{2}$, $k, l \in \mathbb{N}$. Each of them is a quadratic function in l, so this can be done in a straightforward way. The conclusion is that the numerators are positive for $k \geq 2$, $0 \leq l \leq \frac{k}{2}$, $k, l \in \mathbb{N}$, and the right hand sides of (4.37)–(4.39) are indeed convex combinations of the original B–spline coefficients at V_1 and V_2. A similar derivation can now be given for the case $\frac{k}{2} \leq l \leq k$. We only give the resulting subdivision rule here:

$$\begin{aligned} c_{l,1} &= \frac{(l-k-1)(l-k)}{k^2}(\bar{c}_{1,2}+\bar{c}_{1,3}) + \frac{-3k^2-6k+12kl+8l-8l^2}{3k^2}\bar{c}_{2,1} \\ &+ \frac{l(l-1)}{3k^2}(\bar{c}_{2,2}+\bar{c}_{2,3}), \end{aligned} \tag{4.40}$$

$$\begin{aligned} c_{l,2} &= \frac{(k-l)^2}{k^2}\bar{c}_{1,2} + \frac{(k-l)(k-l-1)}{k^2}\bar{c}_{1,3} \\ &+ \frac{-3k^2+12kl+3k-4l-8l^2}{3k^2}\bar{c}_{2,1} + \frac{l(l+2)}{k^2}\bar{c}_{2,2} \\ &+ \frac{l(l-1)}{3k^2}\bar{c}_{2,3}, \end{aligned} \tag{4.41}$$

$$\begin{aligned} c_{l,3} &= \frac{(k-l)(k-l-1)}{k^2}\bar{c}_{1,2} + \frac{(k-l)^2}{k^2}\bar{c}_{1,3} \\ &+ \frac{-3k^2+12kl+3k-4l-8l^2}{3k^2}\bar{c}_{2,1} + \frac{l(l-1)}{3k^2}\bar{c}_{2,2} \\ &+ \frac{l(l+2)}{k^2}\bar{c}_{2,3}. \end{aligned} \tag{4.42}$$

It is also possible to derive subdivision rules for the new vertices that are interior to the triangles of Δ, but this would lead to very complex expressions, and for our purposes we only need k–subdivision along the edges, as will become clear in Chapter 5.

4.5 UPS–Spline Wavelets

In this section we present the UPS–spline wavelet transform. As is customary in the wavelet literature, we will talk about a signal $s(u)$ rather than a UPS–spline. Such a signal can be thought of as consisting of value and partial derivative information $(f_i, f_{x,i}, f_{y,i})$ at the vertices V_i of a uniform triangulation, which by (4.9) uniquely defines a UPS–spline $s(u)$. In Section 4.5.1, we will prove that the basis functions of $S_2^1(D^{-j}\Delta)$ are translates and dilates of three functions. In Section 4.5.2 we will define a multiresolution analysis spanned by these basis functions. Section 4.5.3 introduces a lifting scheme, that splits a given signal into high– and low frequency components step by step, giving rise to the UPS–spline wavelet transform in Section 4.5.4, where the wavelets are also introduced. In Section 4.5.5 we will make some suggestions for further exploring the multi–wavelet aspects of UPS–spline wavelets. Section 4.5.6 concludes with an algorithm for noise removal, based on thresholding.

4.5.1 A closer look at the basis functions

Theorem 4.1 *The B–spline basis functions on $D^{-j}\Delta$ are translates and dilates of three basis functions on Δ.*

Proof

1. Recall that a basis function $B_i^j(u)$ on Δ is the unique solution of the interpolation problem (2.30) with all $(f_l, f_{x,l}, f_{y,l}) = (0,0,0)$ except for $(f_i, f_{x,i}, f_{y,i}) = (\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j}) \neq (0,0,0)$. By the uniformity of the PS–triangles, the triplets $(\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j})$ are the same at each vertex V_i, see also (4.6)–(4.8). Therefore, all the basis functions $B_k^j(u), k = 1, \ldots, n$, on Δ are translates of three functions:

$$\Phi(u) = \begin{bmatrix} B_i^1(u) \\ B_i^2(u) \\ B_i^3(u) \end{bmatrix} \tag{4.43}$$

   for some $i \in \{1, \ldots, n\}$. The three basis functions are shown on Figure 4.16.

2. The Bézier ordinates of $\Phi(u)$ can be calculated by (4.10)–(4.16). From these formulae it is clear that the Bézier representation of the B–splines is independent of the base of the triangulation, B.

3. Consider the basis functions $\Phi(u)$ on Δ and $\Phi^j(u)$ on $D^{-j}\Delta$. The domains of these functions, restricted to their support, are equilateral hexagons with edge length B, resp. $2^{-j}B$. Hence, they are equal up to an affine transformation.

4. The invariance of Bernstein–Bézier polynomials under affine parameter transformations finally leads to the conclusion that the basis functions on all refinements $D^{-j}\Delta$ of $\Delta \in \mathcal{K}$ are translates and dilates of three functions on Δ:

$$\Phi_k^j(u) = \Phi(D^j u + k), \quad k \in \mathcal{K}. \tag{4.44}$$

■

Definition 4.5 *The array of basis functions $\Phi(u)$ is called the multi–scaling function, and each of its components is called a scaling function.*

The dilation equation, which expresses the multi–scaling function in terms of translates and dilates of itself, can be found by applying the subdivision

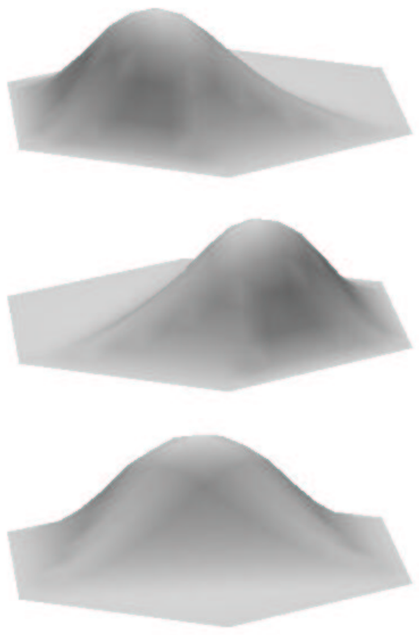


Figure 4.16: The three basis functions on Δ

scheme given by (4.24) and (4.26)–(4.28) directly to the UPS–spline basis functions.

$$\begin{aligned}
\Phi(u) &= \begin{bmatrix} 2/3 & 1/6 & 1/6 \\ 1/6 & 2/3 & 1/6 \\ 1/6 & 1/6 & 2/3 \end{bmatrix} \Phi(Du - \bar{k}_0) \\
&+ \begin{bmatrix} 0 & 1/2 & 1/2 \\ 0 & 1/4 & 0 \\ 0 & 0 & 1/4 \end{bmatrix} \Phi(Du - \bar{k}_1) \\
&+ \begin{bmatrix} 1/4 & 0 & 1/2 \\ 0 & 1/4 & 1/2 \\ 0 & 0 & 0 \end{bmatrix} \Phi(Du - \bar{k}_2) \\
&+ \begin{bmatrix} 1/4 & 0 & 0 \\ 1/2 & 0 & 1/2 \\ 0 & 0 & 1/4 \end{bmatrix} \Phi(Du - \bar{k}_3) \\
&+ \begin{bmatrix} 0 & 0 & 0 \\ 1/2 & 1/4 & 0 \\ 1/2 & 0 & 1/4 \end{bmatrix} \Phi(Du - \bar{k}_4) \\
&+ \begin{bmatrix} 1/4 & 0 & 0 \\ 0 & 1/4 & 0 \\ 1/2 & 1/2 & 0 \end{bmatrix} \Phi(Du - \bar{k}_5) \\
&+ \begin{bmatrix} 1/4 & 1/2 & 0 \\ 0 & 0 & 0 \\ 0 & 1/2 & 1/4 \end{bmatrix} \Phi(Du - \bar{k}_6), \qquad (4.45)
\end{aligned}$$

where $\bar{k}_0 = 0$, and

$$\begin{array}{lll}
\bar{k}_1 = (-B/2, 0)^T, & \bar{k}_2 = \left(-B/4, -\sqrt{3}B/4\right)^T, & \bar{k}_3 = \left(B/4, -\sqrt{3}B/4\right)^T, \\
\bar{k}_4 = (B/2, 0)^T, & \bar{k}_5 = \left(B/4, \sqrt{3}B/4\right)^T, & \bar{k}_6 = \left(-B/4, \sqrt{3}B/4\right)^T.
\end{array}$$

4.5.2 UPS–splines: a multiresolution

Now we can decompose $L_2(\mathbb{R}^2)$ into a sequence of nested subspaces $\mathcal{V}_j = S_2^1(D^{-j}\Delta^*) \subset L_2(\mathbb{R}^2), j \in \mathbb{Z}$, such that

$$\mathcal{V}_j \subset \mathcal{V}_{j+1}, \; j \in \mathbb{Z}, \qquad (4.46)$$

the closure of their union is $L_2(\mathbb{R}^2)$

$$\overline{\lim_{j\to\infty} \mathcal{V}_j} = \overline{\bigcup_{j\in\mathbb{Z}} \mathcal{V}_j} = L_2(\mathbb{R}^2) \qquad (4.47)$$

and their intersection contains only the zero function

$$\lim_{j\to-\infty} \mathcal{V}_j = \bigcap_{j\in\mathbb{Z}} \mathcal{V}_j = \{0\}. \tag{4.48}$$

A UPS–spline $s(u),\ u \in \mathbb{R}^2$ that belongs to one of these subspaces $\mathcal{V}_j$ satisfies the dilation and translation property:

$$s(u) \in \mathcal{V}_j \Leftrightarrow s(Du) \in \mathcal{V}_{j+1},\ j \in \mathbb{Z},\ u \in \mathbb{R}^2, \tag{4.49}$$

$$s(u) \in \mathcal{V}_0 \Leftrightarrow s(u+k) \in \mathcal{V}_0,\ k \in \mathcal{K},\ u \in \mathbb{R}^2. \tag{4.50}$$

Equations (4.46)–(4.50) form the definition of a multiresolution analysis.

4.5.3 The lifting scheme

The lifting scheme [68, 69, 70] (Figure 4.17) is a method for constructing a wavelet transform. It can be used to create wavelet transforms on regular sampling intervals or grids (the first generation wavelets) as well as on irregular samplings, meshes, manifolds,... (the second generation wavelets, which are more flexible). Lifting has a number of advantages over the classical method of constructing wavelets which relies on the Fourier transform. One of these advantages is the fact that the inverse transform can easily be found by running the scheme backward, from right to left, changing each $+$ into a $-$ and vice versa, and inverting each scaling operation (see also [45]). The wavelet transform is a change of basis. It decomposes the given signal $s(u)$ in a space $\mathcal{V}_j$ into a number of components, capturing either the high frequency, or the low frequency characteristics of $s(u)$. Those components live in complementary subspaces of $\mathcal{V}_j$. The basis functions of the low frequency component are called the scaling functions, and the basis functions of the high frequency component are called the wavelet functions. Lifting yields biorthogonal wavelets [13].

In this section we will construct a first generation wavelet transform for UPS–splines. Our lifting scheme starts from a trivial wavelet transform (the split) and enhances its properties using a number of lifting steps. In our construction we use two lifting steps: a prediction step and an update step. Those will be designed with the goal in mind to decompose a UPS–spline into a number of low– and high frequency components. We will also ensure that the original signal can be recovered from its wavelet transform (this is called perfect reconstruction).

Split The trivial wavelet transform splits the given signal into four parts, according to the disjunct sublattices $D\Delta + k_j$ of the original triangulation Δ.

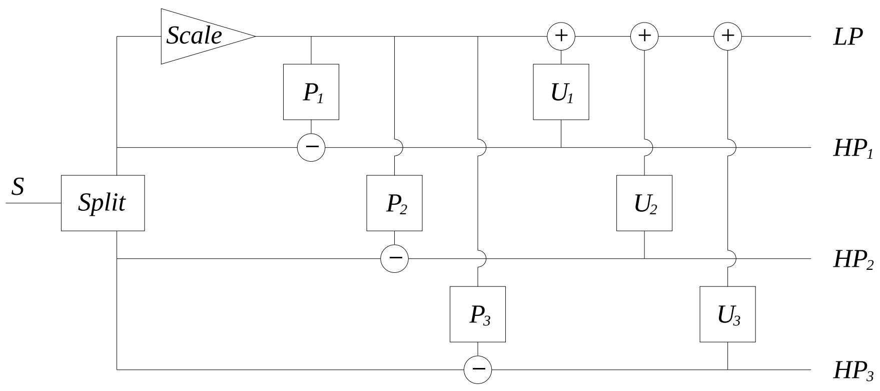


Figure 4.17: The lifting scheme starts with a trivial wavelet i.e. just splitting the signal in samples on the four distinct sublattices. It then gradually builds a new wavelet with improved properties using lifting steps.

Prediction Consider the sub–signal on the zeroth sublattice $D\Delta$ on the upper branch in Figure 4.17. We try to predict the signals on the other sublattices starting from this sub–signal. In order to do so, we first scale the control triangles on $D\Delta$ by the inverse transform of (4.24). This brings them to a lower resolution level. From this level, we use the subdivision rule (4.26)–(4.28) to estimate the values on the other sublattices. We call this the prediction operator P_i for the ith sublattice. Figure 4.18 illustrates the idea: $D\Delta$ is drawn in solid lines; the other sublattices are drawn in dashed lines. The control triangles are displayed schematically as large triangles for $D\Delta$ (after the scaling step) and as small triangles for $D\Delta + k_i, i = 1, 2, 3$. In between each pair of adjacent large triangles, a small triangle can be predicted by subdivision. To continue this lifting step, each prediction $P_i(s)$ is subtracted from the real values on $D\Delta + k_i$. The difference tells how much the prediction fails to be correct.

Definition 4.6 *The detail coefficients or wavelet coefficients are defined as the difference between the real values on $D\Delta + k_i$ and the predictions $P_i(s)$.*

For quadratic polynomials, the prediction will always be exact and the wavelet coefficients are equal to zero. Referring to the wavelet terminology, we say that the UPS–spline wavelet transform has three dual vanishing moments. The upper branch of the scheme now contains the low–frequency component of the signal s, that is, $LP(s)$.

Definition 4.7 *The scaling coefficients are defined as the coefficients on* *the zeroth sublattice, $D\Delta$.*

Thinking of the given signal $s(u)$ as a UPS–spline, the scaling coefficients are B–spline coefficients of a UPS–spline on $D\Delta$, that is, a coarser triangulation than the one we started from. If $s(u) \in \mathcal{V}_j$, then $LP(s) \in \mathcal{V}_{j-1}$. Having completed this lifting step, it is possible to recover the original signal: add the predictions $P_i(s)$ to the wavelet coefficients on $D\Delta + k_i, i = 1, 2, 3$, and scale the control triangles on $D\Delta$ using (4.24).

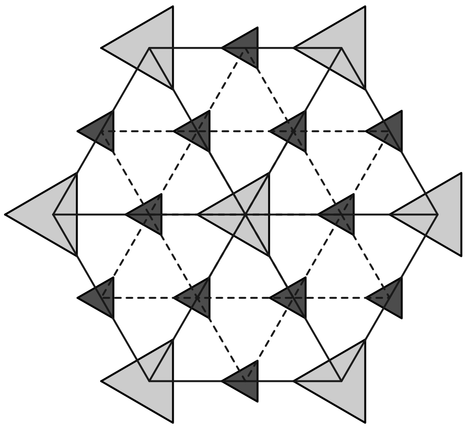


Figure 4.18: The prediction step: $D\Delta$ is drawn in solid lines, the three other sublattices are drawn in dashed lines. The large triangles correspond to the control triangles on $D\Delta$ after the scaling step. In between each such pair of adjacent triangles is a small triangle on a sublattice $D\Delta + k_i$, denoting the corresponding control triangle after the split. The latter one can be predicted from the larger pair by subdivision.

Definition 4.8 *A complementary sequence is obtained by setting one detail coefficient on the right hand side of the lifting scheme to one, all other coefficients to zero, and running the scheme backwards.*

Update In general, the signal on $D\Delta$, which results from the split and prediction steps, does not have the same average as the original signal. In a final lifting step, we make sure that the DC–component of the signal is maintained in the low–frequency component. In the wavelet terminology we say that the wavelet transform has one primal vanishing moment. This goal can be achieved by adding update operators U_i, which update the signal on $D\Delta$ from the values on the other sublattices. Our update step will try to keep things as simple as possible. Nevertheless, some suggestions for more advanced updates are made in Section 4.5.5. For now, we propose that a detail coefficient corresponding to a control triangle T_l, on some branch $D\Delta + k_i$, updates its two neighbour control triangles T_m and T_n on the zeroth sublattice. These are the control triangles from which T_l was predicted in the previous step. More exactly, a coefficient $c_{l,j}$ will increase the values of $c_{m,j}$ and $c_{n,j}$ with an amount $k * c_{l,j}$. The value of k is determined such that the average of the scaling coefficients is maintained, or equivalently, such that the average of the detail signals is zero. To do so, it is sufficient to require that the average of the complementary sequences is zero [71]. If we ignore the boundary cases for now, the value of k can be determined as illustrated on Figure 4.19.

1. **Figure 4.19(a)**: one wavelet coefficient is set to one, in this case $c_{l,1}$. All other coefficients are zero.

2. **Figure 4.19(b)**: inverse update. The neighbour control triangles along this branch are updated, i.e., $c_{m,1} \leftarrow c_{m,1} - k * c_{l,1}$ and $c_{n,1} \leftarrow c_{n,1} - k * c_{l,1}$.

3. **Figure 4.19(c)**: inverse prediction. The figure shows the situation before the scaling step. Since the latter does not affect the sum of the coefficients, it will be ignored here. Imposing the sum of the coefficients in the complementary sequence to vanish yields

$$k = \frac{1}{8}. \tag{4.51}$$

The same value of k is found for the other complementary sequences, corresponding to $c_{l,2} = 1$, to $c_{l,3} = 1$, and to the other HP–branches.

Problems will however arise when an update is made in the neighbourhood of $\partial\Delta$, such that if we draw the control triangles affected by the inverse lifting scheme, this will only be a subset of the ones on Figure 4.19. The complementary sequence will of course be a subset of what is shown on Figure 4.19(c), so the value of k for which the sum of its coefficients vanishes will be different from $k = 1/8$. Nevertheless, the same approach as before can be applied for any possible boundary case. It appears then that a fixed value of k is found for each possible coefficient on $D\Delta$ that can be updated, no matter from which detail wire $D\Delta + k_i$ the update step was initiated. Therefore, there are only four different boundary cases, as was the case for the integration of the B–splines (Section 4.3.3). Figure 4.20 shows the different situations that occur. Any other case can be mapped onto one of these by rotating and mirroring. The figure gives a value of k for each coefficient on $D\Delta$, for example, for $c_{m,j}$. If an update of $c_{m,j}$ is made from

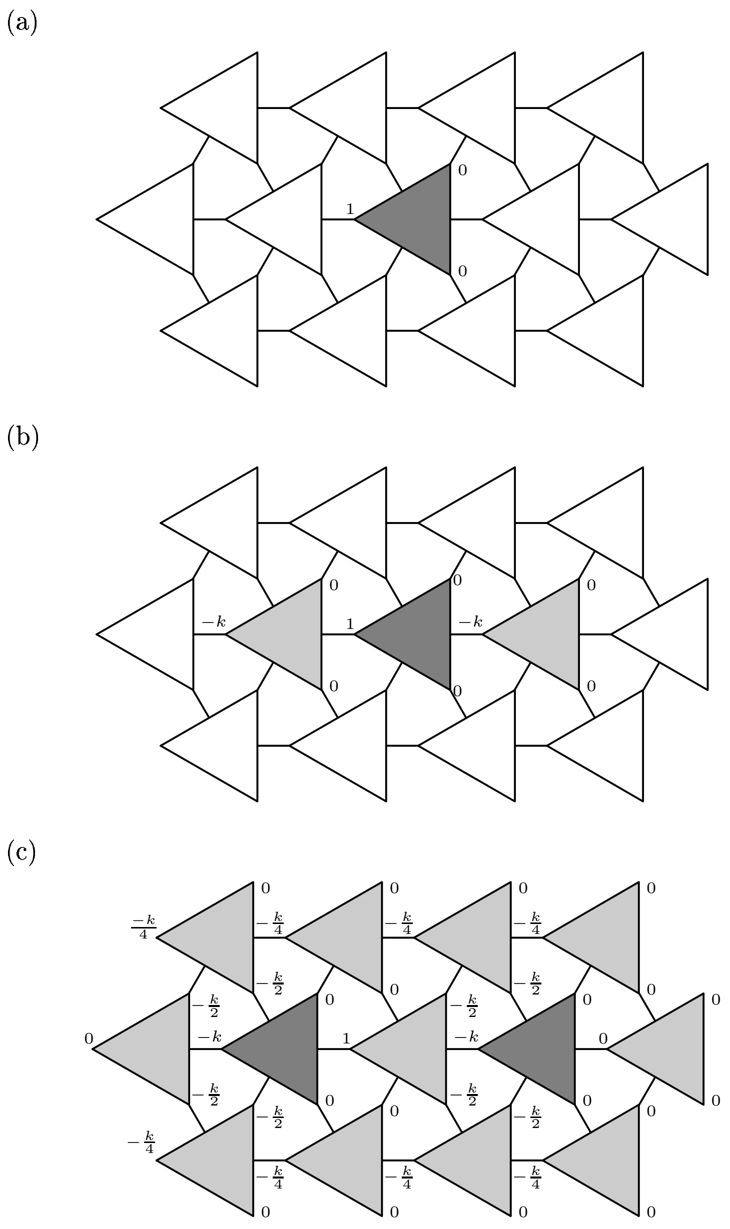


Figure 4.19: Calculation of the update weights. (a) One wavelet coefficient is set to one. (b) Situation after inverse update. (c) Situation after inverse prediction and before inverse scaling.

a detail coefficient $c_{l,j}$ on $D\Delta + k_i$, then $c_{m,j}$ has to be increased by $k * c_{l,j}$. This concludes the design of our update step.

4.5.4 The UPS–spline wavelet transform

In the previous section we designed a wavelet transform that decomposes a given signal $s(u)$ into a low–frequency component $LP(s)$ and three high frequency components $HP_i(S)$. The same scheme can be applied repeatedly to the low frequency component, filtering again out the details and yielding a yet smoother version of the original signal, $LP(LP(s))$. In general, we have an n–step wavelet transform.

Running the inverse wavelet transform starting from a sequence of zeroes and only one scaling coefficient equal to one, yields a scaling function (see Definition 4.5). Repeating the inverse transform over and over yields in the limit a control net converging to that B–spline basis function. In fact, this is nothing else than repeated subdivision.

In a similar way, one finds the basis functions corresponding to the detail signals:

Definition 4.9 *A UPS–spline wavelet in its B–spline representation is obtained by setting one detail coefficient to one, all the other ones to zero, and applying the inverse wavelet transform. The wavelets corresponding to branch $D\Delta + k_i$ are denoted $\psi_{i,j}(u), j = 1, 2, 3$, and the corresponding multi–wavelet is $\Psi_i(u) := [\psi_{i,1}(u), \psi_{i,2}(u), \psi_{i,3}(u)]^T$.*

Repeating the inverse transform ad infinitum yields a control net converging to the wavelet. Figure 4.21 shows a wavelet, (a) seen from the top, (b) seen from aside.

Property 4.3 *A wavelet whose support lies completely within Ω, has a zero integral:*

$$\int_\Omega \psi_{i,j}(u)dxdy = 0 \tag{4.52}$$

Indeed, by (4.22) with $w_{i,j} = \frac{2}{3}$ for all the nonzero coefficients, and the fact that the sum of the complementary sequence vanishes, the integral equals zero. Figure 4.21(b) clearly shows the parts of a wavelet above and beneath the plane $z = 0$. Note that the integral of a wavelet near the border of Ω is not necessarily zero.

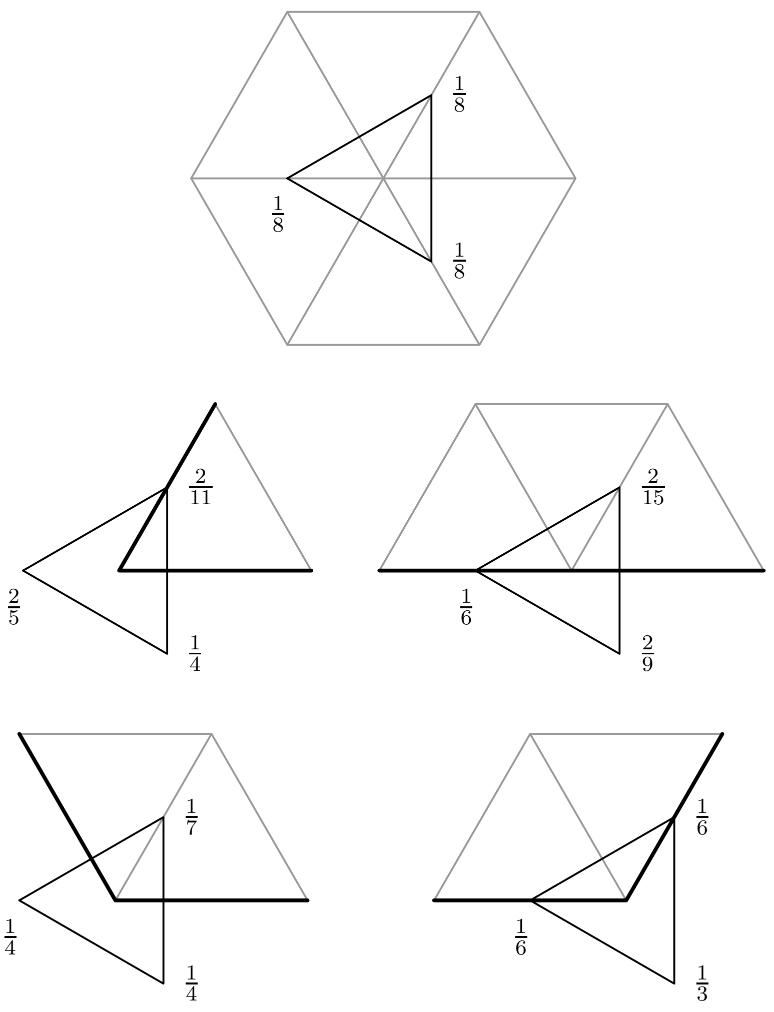


Figure 4.20: The update factor k is fixed for each coefficient $c_{m,j} \in D\Delta$ that has to be updated, no matter from which detail coefficient the update step is initiated. The update weights for all possible configurations are given. If an update of $c_{m,j}$ is made from a detail coefficient $c_{l,j}$ on $D\Delta + k_i$, then $c_{m,j}$ has to be increased by $kc_{l,j}$.

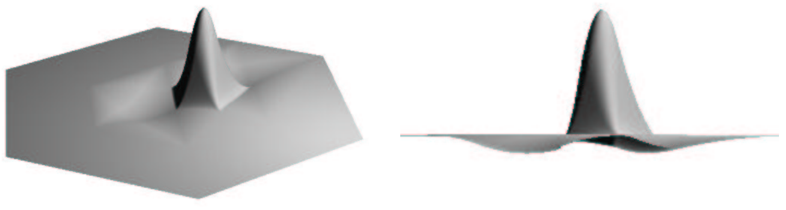


Figure 4.21: A UPS–spline wavelet. (a) View from the top. (b) Sideview, clearly showing the parts above and beneath the plane $z = 0$.

We already noticed that the space $\mathcal{V}_j$ is spanned by the translates and dilates of $\Phi(u)$, i.e.

$$\Phi_{j,k}(u) = \Phi(D^j u - k) \quad \text{with} \quad k \in \mathcal{K}. \tag{4.53}$$

The three high frequency components are each captured in subspaces $\mathcal{W}_{i,j}$ spanned by the translates and dilates of $\Psi_i(u)$, i.e.,

$$\Psi_{i;j,k}(u) = \Psi_i(D^j u - k) \quad \text{with} \quad k \in \mathcal{K}. \tag{4.54}$$

These spaces form the complement of $\mathcal{V}_j$ in $\mathcal{V}_{j+1}$

$$\mathcal{V}_j \oplus \left(\bigoplus_{i=1}^{3} \mathcal{W}_{i;j} \right) = \mathcal{V}_{j+1}. \tag{4.55}$$

Suppose the signal s belongs to $\mathcal{V}_J$, i.e., has resolution level J. It can be decomposed by the repeated wavelet transform such that the eventual lowest resolution level is L. The expansion of s then reads

$$s = \sum_{j=L}^{J-1} \sum_{k \in D^j \mathcal{K}} \sum_{i=1}^{3} W_{i;j,k} \Psi_{i;j,k} + \sum_{k \in D^L \mathcal{K}} C_{L,k} \Phi_{L,k}, \quad W_{i;j,k}, C_{L,k} \in \mathbb{R}^{3\times 1}. \tag{4.56}$$

Finally we note that it is possible to write a dilation equation for the wavelet functions too, similar to (4.45) for the scaling functions. To do so, one can use the complementary sequence and multiply each coefficient with the corresponding scaling function.

4.5.5 Exploring the multi–wavelet aspects

In the standard first generation wavelet theory, the scaling functions are translates and dilates of one function. We already pointed out that the

UPS–spline basis functions are translates and dilates of three scaling functions, which constitute the multi–scaling function (4.43). Correspondingly, the wavelets according to each branch are three–component multi–wavelets. The extension of standard wavelets to multi–wavelets has been investigated in the literature, see e.g. [12, 14, 35, 56] and the references therein. Multi–wavelets have the advantage that the matrix coefficients in the dilation equation allow to satisfy several properties at the same time, e.g., orthogonality, symmetry, short support, continuity and reproduction of constant and linear functions. Without going into details, we will show here how exploring the multi–wavelet aspects of UPS–spline wavelets can result in more complex designs of the update–operator, with additional features as compared to our simple update–step.

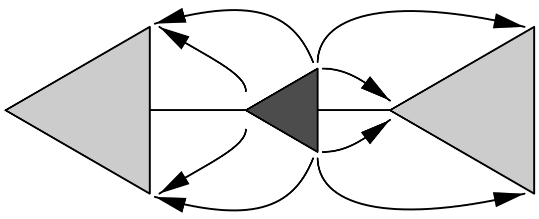


Figure 4.22: A more advanced update step: each detail coefficient updates exactly those scaling coefficients from which it was predicted.

Suppose that the lifting scheme is implemented as a wiring diagram. Since there are in fact three coefficients associated to each branch, such a diagram would have to replace each branch with three wires, say, the LP–wires and the HP_i–wires, each carrying a coefficient $c_{i,j}$. The key observation then is that with multi–wavelets, it becomes possible for a detail coefficient of a certain wavelet $\psi_{l,j}(u), j = 1,2,3$, (that is, on the jth HP_l–wire), to contribute to a scaling coefficient on another LP–wire than the jth. Thus, update steps of the form $c_{m,j} \leftarrow c_{m,j} + k_{j,1} * c_{l,1} + k_{j,2} * c_{l,2} + k_{j,3} * c_{l,3}$ are possible. This could be useful, for example, to design a wavelet transform whose wavelets near the border of Ω have also a zero integral. Suppose we want an update step that not only maintains the average of the signal over the LP–branch, but additionally has the property that every wavelet has a zero integral. Since these are two linear conditions on the complementary sequence, we need at least two degrees of freedom in our update step. We suggest that in this case, a detail coefficient might update exactly those scaling coefficients from which it was predicted before. Figure 4.22 illustrates the idea using arrows to represent the update–contributions.

The boundary cases now become much more complex and numerous. For example on Figure 4.19(a), if any of the edges of the triangulation shown lies on the border of Ω, it is a boundary case. Moreover, the scaling step can't be omitted as we did in our derivation of the value of k. In this context, it is also worth noting that a common approach to resolve these boundary cases for first generation wavelets is to mirror the given data about the boundary, and act as if there was no boundary at all [6, 7]. Finally, we note that the prediction step of the lifting scheme does take advantage of the multi–wavelet aspects of UPS–spline wavelets. The same applies to the scaling step: it can be written as

$$\left\{\begin{array}{lcl} c'_{i,1} & = & \frac{2}{3}c_{i,1} + \frac{1}{6}c_{i,2} + \frac{1}{6}c_{i,3} \\ c'_{i,2} & = & \frac{1}{4}c'_{i,1} + \frac{5}{8}c_{i,2} + \frac{1}{8}c_{i,3} \\ c'_{i,3} & = & \frac{1}{5}c'_{i,1} + \frac{1}{5}c'_{i,2} + \frac{1}{40}c_{i,3}. \end{array}\right. \tag{4.57}$$

Using this formulation, the scaling step in Figure 4.17 can be unraveled in a wiring diagram (Figure 4.23), from which it is clear that each coefficient of the multi–scaling function contributes to the new value of each other coefficient.

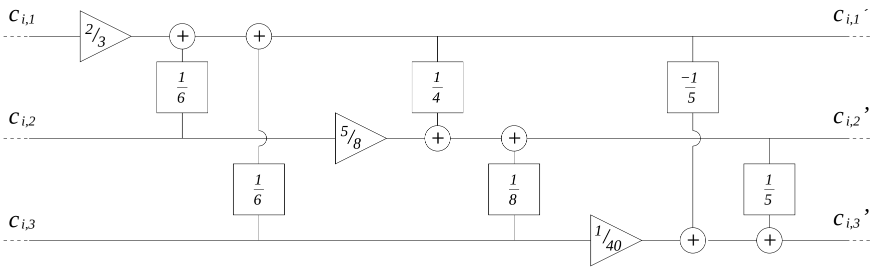


Figure 4.23: Detail of the scale step factored in lifting steps.

4.5.6 A thresholding algorithm for noise removal

As an application, we use the newly developed UPS–spline wavelet transform in an algorithm for smoothing a noisy surface via thresholding. This is a classical application in the field of data processing, see, e.g., [24, 25, 26, 27, 39]. It starts with a wavelet transform of a given signal. Then, some of the wavelet coefficients are changed, and finally a backward wavelet transform yields the denoised signal. In order to manipulate the coefficients, they are divided in a number of classes according to a certain criterion. In most cases, this is a binary criterion: a first class contains the important coefficients, and a second one those that are affected by noise. A straightforward criterion is the absolute value of the coefficients: those that are close to zero contain little information and are relatively strongly influenced by noise. The method assumes that the signal can be represented well by a small number of large coefficients. A thresholding procedure will classify all the wavelet coefficients below a threshold ϵ as noisy; and will replace them by zero. The coefficients above the threshold are decreased in absolute value by ϵ for soft thresholding, or are left untouched in the hard thresholding approach.

The selection of an appropriate threshold is a central issue in this context, and certainly not a trivial one. It is a matter of finding the right balance between a too small threshold (yielding too noisy results) and a too large one that removes important signal features. In most cases, statistical properties of a group of coefficients are used to estimate the optimal value of the corresponding threshold ϵ. This optimal value minimizes the mean square error of the denoised result, as compared with the unknown noise-free coefficients. A well known method for example, uses the minimizer of the Generalized Cross Validation (GCV) function [37, 38, 81].

Suppose we are given a parametric UPS–spline $\boldsymbol{s}(u)$ on Δ,

$$\boldsymbol{s}(u) = \begin{bmatrix} s_x(u) \\ s_y(u) \\ s_z(u) \end{bmatrix} = \sum_{i,j} \boldsymbol{c}_{i,j} B_i^j(u), \quad (u) \in \Omega, \quad \boldsymbol{c}_{i,j} \in \mathbb{R}^3. \tag{4.58}$$

Note that the wavelet transform can be applied component–wise. Suppose that the number of triangles ρ_j along each edge of Ω is, up to a constant, the same power of two, say 2^f. Such a surface can be constructed, for example, by applying f subdivision steps to an initial parametric PS–spline surface, and editing the control points afterwards to modify the surface locally. We call this surface $\boldsymbol{s}^f(u)$. Now we describe the thresholding algorithm.

1. The algorithm begins with a forward wavelet transform with exactly f steps. This means that, during the first step, $\boldsymbol{s}^f$ is decomposed into $\boldsymbol{s}^{f-1} = LP(\boldsymbol{s}^f)$, and a series of wavelet coefficients for each component in the parametric representation, say $\boldsymbol{W}^1 = (W_x^1, W_y^1, W_z^1)$, with, e.g., $W_x^1 = \{HP_1(s_x), HP_2(s_x), HP_3(s_x)\}$. As we know, $\boldsymbol{s}^{f-1}$ is a parametric UPS–spline surface that is a smoother version of the initial one. On the other hand, $\boldsymbol{W}^1$ captures the high frequency parts of $\boldsymbol{s}^f$. During the next step, the same decomposition is applied to the low frequency part: $\boldsymbol{s}^{f-1} \leftarrow (\boldsymbol{s}^{f-2}, \boldsymbol{W}^2)$, etc. After f steps, the result is a surface $\boldsymbol{s}^0(u, v)$, together with f series of wavelet coefficients $\boldsymbol{W}^k, k = 1, \ldots, f$.

2. The second phase of the algorithm consists of f thresholding steps. During each such step $k = 1 \ldots, f$, the user can provide a triple of real positive thresholds $(\epsilon_x^k, \epsilon_y^k, \epsilon_z^k)$. All the wavelet coefficients belonging to $W_l^{f-(k-1)}$ which have an absolute value smaller than $\epsilon_l^k, l = x, y, z$, are replaced by zeros. For the remaining wavelet coefficients of $W_l^{f-(k-1)}$ we consider two possibilities.

   Hard thresholding leaves the wavelet coefficients with an absolute value higher than the threshold unchanged.

   Soft thresholding decreases the absolute value of the remaining wavelet coefficients from $W_l^{f-(k-1)}$ with ϵ_l^k, leaving their sign unchanged.

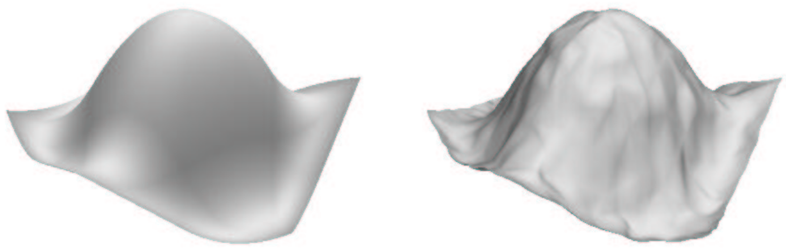
   Applying the inverse wavelet transform on the thresholded coefficients, denoted $\bar{\boldsymbol{W}}^{f-(k-1)}$, results in the surface $\bar{\boldsymbol{s}}^k$, which is, together with $\boldsymbol{W}^{f-k}$, the input for the next thresholding step. After f steps, a denoised surface $\bar{\boldsymbol{s}}^f$ at the same resolution level as the initial one results.

We have tested this algorithm on a noisy UPS–spline surface, which was obtained by subdividing a given smooth surface f times, after which its B–spline coefficients $c_{i,j}$ were replaced by $c_{i,j} + \delta c_{i,j}$ with $\delta c_{i,j}$ uniformly distributed on a user specified interval $[-c, c]$. Figure 4.24 shows the original surface (left) and the surface after adding noise to the coefficients (right). Figure 4.25 shows the output for $f = 3$, with ϵ_l^k equal to 10% of the maximal wavelet coefficient of $W_l^{f-(k-1)}$ in absolute value, for $l = x, y, z$. The result for hard thresholding is displayed left and the result for soft thresholding right. At first sight, soft thresholding yields the better results. The following table shows the percentage of wavelet coefficients that have been eliminated during each soft thresholding step. The figures indicate that our wavelet transform may be useful for data compression of large UPS–splines as well.

k	S_x	S_y	S_z
1	27.78%	22.23%	13.89%
2	48.41%	50.00%	38.10%
3	73.50%	75.43%	74.79%

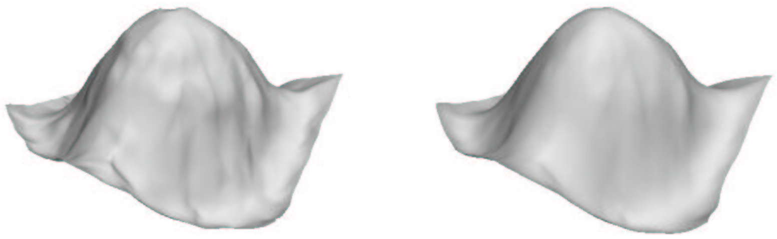
4.6 Graphical display of UPS–spline surfaces

It is customary to visualize bicubic B–spline surfaces on rectangular domains with two-direction wireframe models. As an application of formulae



(a) (b)

Figure 4.24: (a) The original surface. (b) The noisy surface.



(a) (b)

Figure 4.25: The output for (a) hard thresholding and (b) soft thresholding, both after three steps.

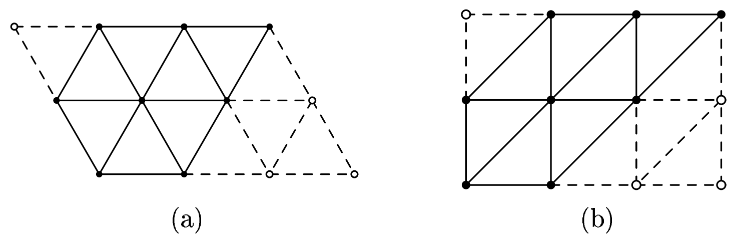


Figure 4.26: (a) An extended triangulation of Δ is obtained by adding the minimum number of triangles such that it becomes a parallelogram with its base line shifted to the right. (b) An associated grid is obtained by recasting the extended triangulation into a rectangle.

(4.24)–(4.28) we show how a three–direction wireframe model of the PS–spline surface can be constructed. Efficient local matrix operations make the algorithm suited for hardware implementation. By avoiding plain evaluations, we don't need to convert representation (2.31) to its Bernstein–Bézier form.

4.6.1 A matrix structure

From the uniform triangulation Δ we can deduce a naturally associated matrix structure containing control triangle data.

Step 1. Extend the triangulation with the smallest number of triangles so that a parallelogram with its baseline shifted to the right arises, and denote it with $\tilde{\Delta}$. Figure 4.26(a) shows the extension of a five-sided domain triangulation with dashed lines. New vertices have been introduced, so we have to distinguish between

- real vertices (black dots)
- pseudo–vertices (circles)

Step 2. Think of this triangulation as a movable grid, and recast it into a rectangle (see Figure 4.26b). Call this the associated grid R, with r horizontal lines and c vertical lines.

Step 3. With each vertex of the grid, associate a 3×3 matrix B, called the control block:

- for a real vertex V_i the control block is given by

$$\left[\; \boldsymbol{c}_{i,1} \quad \boldsymbol{c}_{i,2} \quad \boldsymbol{c}_{i,3} \;\right], \tag{4.59}$$

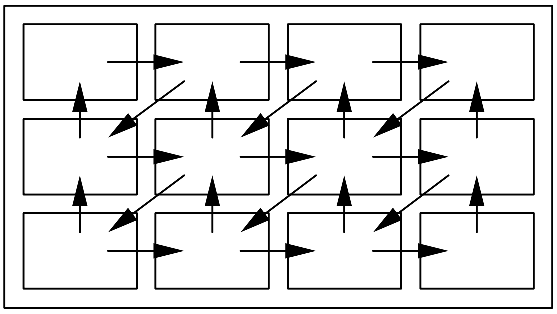


Figure 4.27: The associated matrix structure M id obtained by placing a matrix B at each vertex of the triangulation, containing the control point coordinates. Each edge is replaced by an arrow.

- for a pseudo–vertex, the control block is a 3×3 zero matrix.

Step 4. All those tiny matrices, located at the position of the corresponding grid point, constitute a block matrix structure M with r blockrows and c blockcolumns. Replacing the edges of the associated grid R by arrows, we obtain the associated matrix structure M of the grid (see Figure 4.27). The arrows go in counterclockwise direction if the arrows are considered as edges of domain triangles pointing upward.

Suppose that the triangulation is subdivided into $D^{-1}\Delta$. Then a new structure M' should be deduced, which is the matrix structure associated with the extended refined triangulation $D^{-1}\tilde{\Delta}$, using local matrix operations on M. Thinking of the B–matrices as control triangles and the arrows as edges, it is clear that the subdivision process causes new B'–matrices to appear:

- at the position of the original B–matrices,
- halfway the arrows.

More particularly, if an arrow in the matrix structure points from the control block associated with V_i to the one of V_j, then formulae (4.26)–(4.28) apply immediately. Figure 4.28 shows the structure M'; the B'–matrices corresponding to the arrows have been shaded. The number of rows and columns increase as

$$r' = 2r - 1 \qquad c' = 2c - 1. \tag{4.60}$$

To distinguish the real from the pseudo–vertices, let the first and last position of a real vertex in row i be denoted by $k_{i,1}$ resp. $k_{i,2}$. They evolve

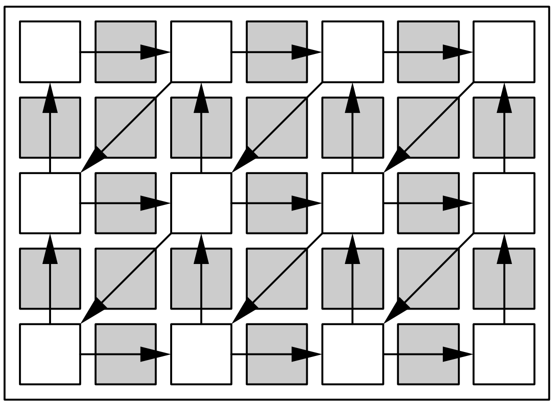


Figure 4.28: Matrix structure M'.

like

$$k'_{2i-1,j} = 2k_{i,j} - 1,\ j = 1, 2,\ i = 1, \dots, r, \tag{4.61}$$

and

$$k'_{2i,j} = \frac{1}{2}(k'_{2i-1,j} + k'_{2i+1,j}),\ j = 1, 2,\ i = 1, \dots, r-1. \tag{4.62}$$

4.6.2 Local matrix operations

The local matrix operations allowing to calculate B' from B can be derived from (4.24)–(4.28), i.e.

$$B'_{2k-1,2l-1} = B_{k,l}\begin{bmatrix} \frac{2}{3} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{2}{3} \end{bmatrix},\ k = 1, \dots, r,\ l = 1, \dots, c$$

$$B'_{2k-1,2l} = B_{k,l}\begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{4} & 0 \\ \frac{1}{2} & 0 & \frac{1}{4} \end{bmatrix} + B_{k,l+1}\begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} \end{bmatrix}, \quad \begin{matrix} k = 1, \dots, r \\ l = 1, \dots, c-1 \end{matrix}$$

$$B'_{2k,2l-1} = B_{k+1,l}\begin{bmatrix} \frac{1}{4} & \frac{1}{2} & 0 \\ 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{4} \end{bmatrix} + B_{k,l}\begin{bmatrix} \frac{1}{4} & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{4} \end{bmatrix}, \quad \begin{matrix} k = 1, \dots, r-1 \\ l = 1, \dots, c \end{matrix}$$

$$B'_{2k,2l} = B_{k,l+1}\begin{bmatrix} \frac{1}{4} & 0 & \frac{1}{2} \\ 0 & \frac{1}{4} & \frac{1}{2} \\ 0 & 0 & 0 \end{bmatrix} + B_{k+1,l}\begin{bmatrix} \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix}, \quad \begin{matrix} k = 1, \dots, r-1 \\ l = 1, \dots, c-1 \end{matrix}$$

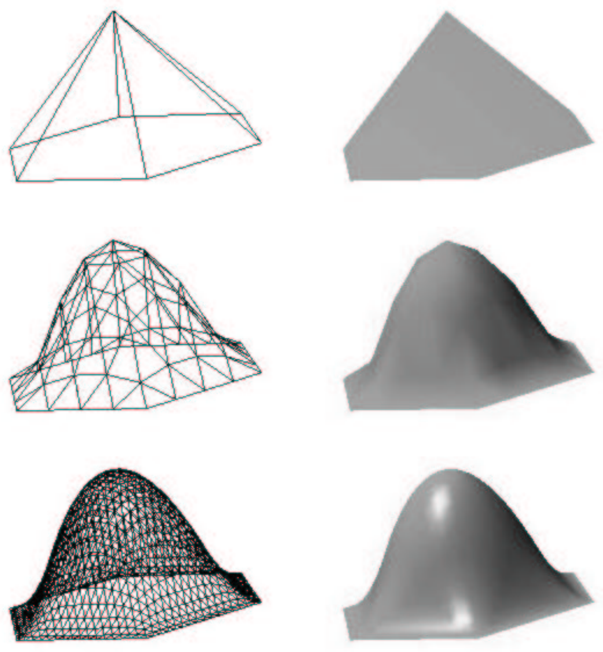


Figure 4.29: A three–direction wireframe and rendered PS–spline surfaces for an increasing number of refinement steps, zero, two and four.

4.6.3 Construction of the wireframe

After a few refinement steps, the points on the PS–spline surface are generated as follows:

```
**for** $i = 1, \dots, r$
  **for** $j = 1, \dots, c$
    $p(i,j) = B_{ij} \left(\frac{1}{3}\ \frac{1}{3}\ \frac{1}{3}\right)^T$
```

The actual data that eventually will be displayed, are obtained by selecting the real vertices from the extended triangulation and joining them, yielding a wireframe. Figure 4.29 shows the surface at consecutive refinement steps as a wireframe (left) and with the polygons shaded (right).

The information to calculate the normal vectors n_i at the tangent points is readily available as well. This is useful for rendering surfaces. Furthermore, note that we only use convex combinations of the control points. Therefore, an extension to the graphical display of rational UPS–spline surfaces, similar to the rational algorithms of Chapter 3, where we avoided to work in the homogeneous space, can be made in a straightforward manner.

4.6.4 Graphical display of UPS–spline surfaces with non uniform PS–triangles

Recall from Section 3.6 that we derived rational PS–spline representations of the cylinder, cone, and the sphere (though not complete). These were composite NURPS surfaces, where each patch was given in its optimal representation on an equilateral domain triangle (see Figure 3.15). If we can find the representation of these surfaces using uniform PS–triangles instead of optimal ones, then it is possible to display them graphically with the rational extension of our visualization algorithm. To do so, we first derive general formulae for replacing a PS–triangle of a given PS–spline. Then we apply this to the case of replacing an optimal PS–triangle with a uniform one. The component–wise extension for PS–spline surfaces is then obvious.

Consider a PS–spline $s(u)$ on Δ, and the PS–triangle $t_i(Q_{i,1}, Q_{i,2}, Q_{i,3})$ at V_i with corresponding triplets $(\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j}), j = 1,2,3$, and B–spline coefficients $c_{i,1}, c_{i,2}, c_{i,3}$. Recall from Section 2.3.2 that from the locality of the B–splines, we have

$$\begin{bmatrix} \alpha_{i,1} & \alpha_{i,2} & \alpha_{i,3} \\ \beta_{i,1} & \beta_{i,2} & \beta_{i,3} \\ \gamma_{i,1} & \gamma_{i,2} & \gamma_{i,3} \end{bmatrix} \begin{bmatrix} c_{i,1} \\ c_{i,2} \\ c_{i,3} \end{bmatrix} = \begin{bmatrix} s(V_i) \\ \frac{\partial s(V_i)}{\partial x} \\ \frac{\partial s(V_i)}{\partial y} \end{bmatrix}. \tag{4.63}$$

Now suppose that the triangle $t'_i(Q'_{i,1}, Q'_{i,2}, Q'_{i,3})$ also contains the PS–points of V_i, then it is possible to replace t_i with t'_i. The α–, β– and γ–triplets corresponding to t'_i are directly obtained by (2.37)–(2.40). Furthermore, let $c'_{i,1}, c'_{i,2}, c'_{i,3}$ denote the B–spline coefficients of $s(u)$ with respect to t'_i. Then an expression similar to (4.63) can be written for t'_i as well. Elimination of the right hand sides allows to calculate the unknown coefficients by

$$\begin{bmatrix} c'_{i,1} \\ c'_{i,2} \\ c'_{i,3} \end{bmatrix} = \begin{bmatrix} \alpha'_{i,1} & \alpha'_{i,2} & \alpha'_{i,3} \\ \beta'_{i,1} & \beta'_{i,2} & \beta'_{i,3} \\ \gamma'_{i,1} & \gamma'_{i,2} & \gamma'_{i,3} \end{bmatrix}^{-1} \begin{bmatrix} \alpha_{i,1} & \alpha_{i,2} & \alpha_{i,3} \\ \beta_{i,1} & \beta_{i,2} & \beta_{i,3} \\ \gamma_{i,1} & \gamma_{i,2} & \gamma_{i,3} \end{bmatrix} \begin{bmatrix} c_{i,1} \\ c_{i,2} \\ c_{i,3} \end{bmatrix}. \tag{4.64}$$

Replacing the optimal PS–triangles with uniform ones. The precise values of $(\alpha_{i,j}), (\beta_{i,j}), (\gamma_{i,j}), (\alpha'_{i,j}), (\beta'_{i,j}), (\gamma'_{i,j}), j = 1,2,3$, depend on the relative orientation of t_i and t'_i. We give an example for the configuration

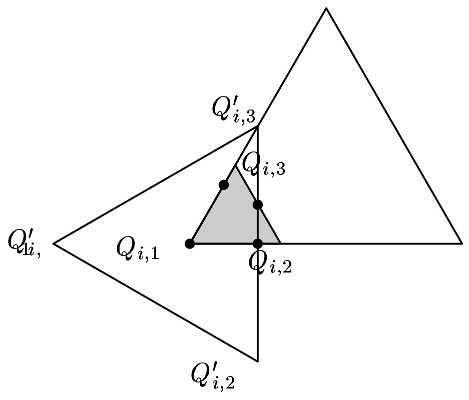


Figure 4.30: Replacing the optimal PS–triangle $t_i(Q_{i,1}, Q_{i,2}, Q_{i,3})$ with a uniform one, $t'_i(Q'_{i,1}, Q'_{i,2}, Q'_{i,3})$

on Figure 4.30. The values for the optimal PS–triangle $t_i(Q_{i,1}, Q_{i,2}, Q_{i,3})$ are given by

$$(\alpha_{i,1}, \alpha_{i,2}, \alpha_{i,3}) = (1, 0, 0)\,, \tag{4.65}$$

$$(\beta_{i,1}, \beta_{i,2}, \beta_{i,3}) = \left(-\frac{3}{B}, \frac{3}{B}, 0\right), \tag{4.66}$$

$$(\gamma_{i,1}, \gamma_{i,2}, \gamma_{i,3}) = \left(-\frac{\sqrt{3}}{B}, -\frac{\sqrt{3}}{B}, \frac{2\sqrt{3}}{B}\right), \tag{4.67}$$

and those for the uniform PS–triangle are given by (4.6)–(4.8). Application of (4.64) yields

$$\begin{bmatrix} c'_{i,1} \\ c'_{i,2} \\ c'_{i,3} \end{bmatrix} = \begin{bmatrix} 5/2 & -3/2 & 0 \\ 1 & 3/2 & -3/2 \\ -1/2 & 0 & 3/2 \end{bmatrix} \begin{bmatrix} c_{i,1} \\ c_{i,2} \\ c_{i,3} \end{bmatrix}. \tag{4.68}$$

Note that these barycentric combinations are not convex. This is what we could expect because t'_i does not ly within t_i. Nevertheless they allow to convert the optimal representation of the rational PS–spline surfaces on quadrics, given in Section 3.6, to a uniform representation using exact symbolic algebra. The graphical display based on UPS–spline subdivision is then possible. Figure 4.31 shows UPS–spline patches on a cylinder (left) and a sphere (right), together with the control triangles. Note that the figures from Chapter 3 which showed a cylinder (Figure 3.17) and a cone and a sphere (Figure 3.21) have been obtained by calculating the uniform rational UPS–spline representation as well.

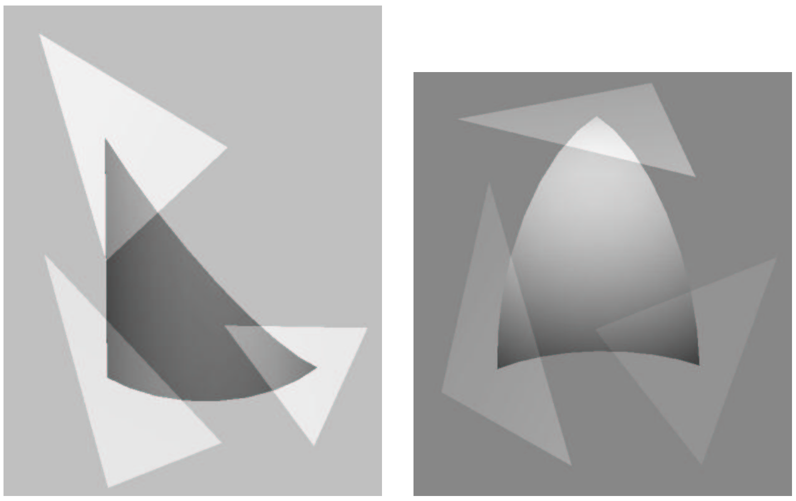


Figure 4.31: Uniform PS–spline patches on the cylinder (left) and sphere (right) together with the control triangles.

4.7 Concluding Remarks

In this chapter we studied PS–splines on uniform triangulations, the so–called UPS–splines. Although they allow to represent surfaces with three, four, five and six edges on convex domains only, we showed that UPS–splines have a number of advantages compared with general PS–splines: they can be implemented and manipulated more efficiently; they have multiresolution properties and graphical display based on subdivision is possible.

In Section 4.2.1 we introduced the notation "$D^{-j}\Delta$" for triangulations at the jth refinement level. We defined UPS–splines in Section 4.2.2, and showed that it is possible to fix the form of the PS–triangles in advance. Solving a quadratic programming problem for each vertex at runtime is then avoided. This leads to more efficient calculations with UPS–splines, as shown in Section 4.3. We have given explicit formulae for solving an interpolation problem where function value and derivatives at each vertex are given, for calculating the Bernstein–Bézier representation, and for integrating UPS–splines. In Section 4.4 we presented a subdivision scheme for UPS–splines, allowing to represent a given UPS–spline on a refinement of the initial triangulation. Our subdivision scheme uses simple convex combinations of the original control points only, so it is an efficient and numerically stable algorithm. We extended this scheme to general k–subdivision along the edges, which will be useful in the context of the polygonal hole problem in Chapter 5. Two applications of subdivision have been studied.

The first one, in Section 4.5, was the development of a UPS–spline wavelet transform using lifting. It combines the advantages of B–spline wavelets, non-separable wavelets and multi–wavelets: the explicit form allows for an efficient evaluation of the wavelets and scaling functions, they have better orientation properties than tensor product wavelets, and they allow for more flexibility when designing lifting steps, such that several desirable wavelet properties can be obtained at once. We proposed a simple update step maintaining the average of the signal over the low frequency component, and proved that, ignoring boundary cases, the integral of the corresponding wavelets is zero. Furthermore, we made some suggestions for enhancing the design of the update step such that the wavelets near the boundary also have a zero integral. We explicitly formulated the wavelet decomposition in Section 4.5.4. A practical application of the UPS–spline wavelet transform was given in Section 4.5.6: we presented an algorithm for noise removal based on thresholding. This can be useful for smoothing UPS–spline surfaces on refined triangulations, without changing the global shape. Such a noisy surface may be obtained if a UPS–spline is used to fit a set of data, using an algorithm that recursively refines the initial triangulation. Our experiments showed that soft thresholding yielded the better results compared to hard thresholding, and a large percentage of wavelet coefficients can be eliminated without modifying the global shape of the UPS–spline surface. This indicates that our wavelet transform may be useful for data compression of large UPS–splines as well. In this context it is worth noting that Vanraes et al. very recently developed a subdivision scheme for general PS–splines [77]. In view of Section 4.5, it has the potential of leading to the construction of second generation, non–separable B–spline multi–wavelets on arbitrary triangulations.

Finally, in Section 4.6 we gave a second application of subdivision: a number of points on a UPS–spline surface can be generated efficiently, as well as the normal vectors to the surface at those points, allowing to display UPS–spline surfaces either as a three–direction wireframe, or as a lit surface. We also proposed a matrix structure for representing UPS–splines, that is suitable for hardware implementation. We will see in Chapter 6 that this representation leads to a very efficient graphical display in OpenGL. Rational UPS–spline surfaces and surface patches on equilateral triangles in their optimal representation can be displayed as well. For the first, the drawing algorithm can be extended to its rational form in a similar way as we did for the calculation of the Bézier representation in Section 3.3.2. For the latter, we derived a general formula for replacing a PS–triangle by another valid one, and applied it for calculating the uniform representation of an optimal surface patch.

In Chapter 6 we will demonstrate the application of UPS–splines in CAGD with our software prototype, PS–Surf. It relies on the data structure presented here, and features, among other things, interactive manipulation of PS–spline surfaces, subdivision, thresholding, and real-time graphical display.

Chapter 5

UPS–splines for the polygonal hole problem

5.1 Introduction

A classical problem in CAGD is to fill in a hole, bounded by a set of surfaces. This problem has been addressed in the literature. A number of authors determine a patch by blending surfaces that meet the boundary conditions at certain curves: for example in [9], a pentagonal surface patch, joining adjacent rectangular patches with C^1–continuity is presented. In [32] an n–sided patch is found by extending a given C^k patch complex around the hole. For the case of cubic boundary curves, [10] calculates a C^1 cubic triangular spline patch, interpolating the curves exactly and approximating the boundary derivatives on the hole. By energy minimization, the bumpiness of the patch is reduced. A solution with a G^1 subdivision surface patch appeared in [50]. In this chapter, we present an algorithm for filling in a three, four, five or six–sided hole with a UPS–spline surface. We demonstrate that using composite patches, it is possible to fill a gap bounded by two curves as well. Our algorithm differs from most of the solutions in the literature in that it makes no assumptions on the surrounding surfaces, and therefore it is generally applicable. On the other hand, the filling patch will meet the given boundary curves only approximately. The input of our algorithm (see Figure 5.1) consists of the boundary curves $\boldsymbol{p}$ which join at their endpoints, parameterized on the unit interval. Furthermore, the user should provide the unit tangent vector $\vec{\gamma}$ to the boundary curves at any point, and the unit normal vector $\vec{n}$ to the surrounding surface at any curve point except at the endpoints, where the tangent vectors of the joining curves are needed only (see Figure 5.1). For other (interior) curve points, our algorithm will

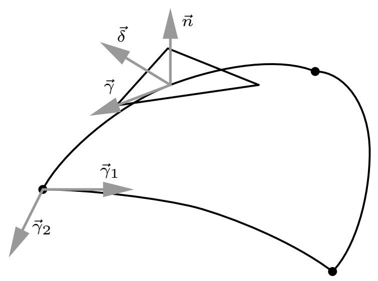


Figure 5.1: The user supplied data consists of the bounding curves $\boldsymbol{p}$, the unit tangent vector $\vec{\gamma}$ and the unit normal $\vec{n}$ to the surrounding surface. From these, the unit cross–boundary tangent vector $\vec{\delta}$ is computed.

calculate a unit vector $\vec{\delta} = \vec{\gamma} \times \vec{n}$, called the (unit) cross–boundary tangent vector. It shall be referred to as if it were provided by the user. In Section 5.2 we will show how the control triangles along the boundary of the filling patch can be determined such that the user supplied curves are interpolated in a number of points, and approximated in between them. This yields the so–called fitting equations. These are used in Section 5.3 to determine a solution iteratively.

Having determined the control triangles of the filling patch along the boundary of the polygonal hole, Section 5.4 addresses the problem of finding the interior control triangles of the filling patch. First, in Section 5.4.1, we have a look at how the quality of the different approaches can be compared relying on surface interrogation tools. Then, in Section 5.4.2, some more or less intuitive approaches are discussed. The remainder of Section 5.4 focuses on a mathematically more soundly justified, but more expensive method which tries to maximize the fairness of the filling patch. This goes back to another common problem in CAGD, namely the generation of aesthetically appealing surfaces, subject to certain technical requirements such as interpolation to a given data set [44, 61, 80]. The latter constraints are often easy to formulate in mathematical terms, but this is not immediately the case for quality criteria measuring the well shaped–ness of surfaces. Motivated by physical models of elastic membranes or thin plates [53], the variational approach measures the lack of smoothness of a surface by the value of some energy functional. Fairing methods then minimize this objective function; it is assumed that surfaces of minimal energy are fair. A fairing method that is based on intrinsic surface properties only, avoids that the parameterization influences the resulting outer fairness. Unfortunately, intrinsic geometric quantities such as the curvature measures $(\kappa_1 + \kappa_2)^2, |\kappa_1\kappa_2|$ and $\kappa_1^2 + \kappa_2^2$ (where κ_1 and κ_2 are the principal curvatures) depend on the coefficients in a nonlinear way and their evaluation is rather complex. Therefore, these often are approximated by a combination of second order derivatives, such as the energy of a thin elastic plate [51]. In Section 5.4.3 we review a measurement for the fairness of a PS–spline surface [20] and present a constrained fairing method for UPS–splines that is not intrinsic, but that is closely affiliated with the nature of PS–spline surfaces and that is directly applicable to the polygonal hole problem, enabling us to calculate the interior part of the patch, maximizing the global fairness of the surface.

The various approaches for calculating the interior are illustrated using an example, based on the incomplete NURPS representation of the sphere from Chapter 3, and are compared throughout Section 5.4. Section 5.6 concludes with some more examples: blending two intersecting cylinders and placing a cap on top of a prism.

5.2 The fitting equations

In this section, we will investigate how the filling patch, a UPS–spline surface $\boldsymbol{s}(u)$ on a uniform triangulation Δ, can approximate the user supplied curves along its boundary, $\partial\Omega$. Section 5.2.1 gives the piecewise Bernstein–Bézier representation of the bounding curves. In Section 5.2.2 we show how interpolation of the user supplied data at $V_i \in \partial\Delta$ can be achieved, leaving some degrees of freedom. These are used in Section 5.2.3 to fit the given data in between the interpolation points, yielding the fitting equations.

5.2.1 The piecewise Bernstein–Bézier representation

The Bernstein–Bézier representation of $\boldsymbol{s}(u)$ along an edge V_iV_j of $\partial\Delta$, (Figure 5.2) can be written in terms of the B–spline control points using (4.10)–(4.12):

$$\boldsymbol{s}(V_i) = \boldsymbol{p}_i = \frac{1}{3}(\boldsymbol{c}_{i,1} + \boldsymbol{c}_{i,2} + \boldsymbol{c}_{i,3}) \tag{5.1}$$

$$\boldsymbol{s}(V_j) = \boldsymbol{p}_j = \frac{1}{3}(\boldsymbol{c}_{j,1} + \boldsymbol{c}_{j,2} + \boldsymbol{c}_{j,3}) \tag{5.2}$$

$$\boldsymbol{u}_i = \frac{1}{2}(\boldsymbol{c}_{i,2} + \boldsymbol{c}_{i,3}) \tag{5.3}$$

$$\boldsymbol{u}_j = \frac{2}{3}\boldsymbol{c}_{j,1} + \frac{1}{6}(\boldsymbol{c}_{j,2} + \boldsymbol{c}_{j,3}) \tag{5.4}$$

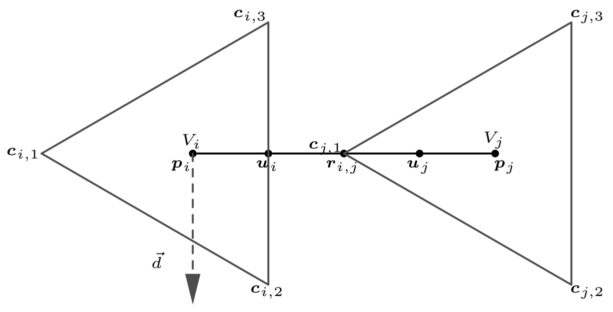


Figure 5.2: The Bézier points of $\boldsymbol{s}(u)$ along an edge V_iV_j of $\partial\Delta$.

$$\boldsymbol{r}_{i,j} \quad = \quad \frac{1}{2}(\boldsymbol{u}_i + \boldsymbol{u}_j). \tag{5.5}$$

Assuming a (counterclockwise) ordering of the boundary vertices $V_i \in \partial\Delta$, the edge curve from $\boldsymbol{s}(V_i)$ to the next adjacent point $\boldsymbol{s}(V_j)$ will be denoted $\boldsymbol{e}_{i,j}(u)$. It is a piecewise quadratic Bézier curve, which means that $\boldsymbol{p}_i$, $\boldsymbol{r}_{i,j}$ and $\boldsymbol{p}_j$ are surface points, and that $\boldsymbol{u}_i - \boldsymbol{p}_i$ and $\boldsymbol{p}_j - \boldsymbol{u}_j$ are tangent to the surface at $\boldsymbol{p}_i$, resp. $\boldsymbol{p}_j$.

5.2.2 Interpolating UPS–splines and degrees of freedom

Recall that the user provides a number of boundary curves $\boldsymbol{p}$, unit tangent vectors $\vec{\gamma}$ and cross–boundary tangent vectors $\vec{\delta}$. In this section we show how to interpolate those data at the vertices $V_i \in \partial\Delta$. The given values at V_i will be denoted $(\boldsymbol{p}_i, \vec{\gamma}_i, \vec{\delta}_i)$. There are two cases to consider: corner vertices and inner vertices of the boundary.

Inner vertices $V_i \in \partial\Delta$

In order to obtain interpolation we determine a control triangle T_i in the tangent plane spanned by $\boldsymbol{p}_i + a\vec{\gamma}_i + b\vec{\delta}_i$, $a, b \in \mathbb{R}$, such that $\boldsymbol{s}(V_i) = \boldsymbol{p}_i$. So there are three conditions.

1. Curve point interpolation is simply expressed by (5.1).
2. Furthermore, we let the tangent to $\boldsymbol{e}_{i,j}$ at V_i be parallel to $\vec{\gamma}_i$:

$$\boldsymbol{u}_i - \boldsymbol{p}_i = \frac{1}{6}(\boldsymbol{c}_{i,2} + \boldsymbol{c}_{i,3}) - \frac{1}{3}\boldsymbol{c}_{i,1} = \frac{1}{2}\alpha_i\vec{\gamma}_i, \tag{5.6}$$

where α_i is a scaling factor.

3. Next, we need the cross–boundary tangent vector of $\boldsymbol{s}(u)$ at V_i to be parallel with $\vec{\delta}_i$. Mapping the cross–boundary vector $\vec{d}$ in the domain plane (see Figure 5.2) onto the control triangle yields a vector parallel with $\boldsymbol{c}_{i,2} - \boldsymbol{c}_{i,3}$:
$$\boldsymbol{c}_{i,2} - \boldsymbol{c}_{i,3} = 2\beta_i \vec{\delta}_i, \tag{5.7}$$
where β_i is again a scaling factor.

Solving (5.1), (5.6) and (5.7) for $\boldsymbol{c}_{i,j}$ in terms of the unknown α_i and β_i (further called the α– and β–factors) yields
$$\left\{\begin{array}{lcl} \boldsymbol{c}_{i,1} & = & \boldsymbol{p}_i - \alpha_i \vec{\gamma}_i \\ \boldsymbol{c}_{i,2} & = & \boldsymbol{p}_i + \frac{\alpha_i}{2}\vec{\gamma}_i + \beta_i \vec{\delta}_i \\ \boldsymbol{c}_{i,3} & = & \boldsymbol{p}_i + \frac{\alpha_i}{2}\vec{\gamma}_i - \beta_i \vec{\delta}_i . \end{array}\right. \tag{5.8}$$

These equations ensure that $\boldsymbol{s}(u)$ interpolates the given data at $V_i \in \partial\Delta$, and leave us two degrees of freedom per vertex (α_i and β_i). These scaling factors are related to the size of the control triangle. For example, subdivision by (4.24) divides α_i and β_i by a factor of 2.

Corner vertices $V_i \in \partial\Delta$

Consider a sharp corner on Figure 5.3. The data corresponding to the outgoing edge are $(\boldsymbol{p}_i, \vec{\gamma}_1, \vec{\delta}_1)$ and those of the incoming edge are $(\boldsymbol{p}_i, \vec{\gamma}_2, \vec{\delta}_2)$. We determine a control triangle T_i in the tangent plane spanned by $\boldsymbol{p}_i + a\vec{\gamma}_1 + b\vec{\gamma}_2$, $a, b \in \mathbb{R}$, such that $\boldsymbol{s}(V_i) = \boldsymbol{p}_i$. There are three conditions.

1. Curve point interpolation is again expressed by (5.1).

2. We let the tangent to the incoming curve be parallel to $\vec{\gamma}_2$:
$$\boldsymbol{u}_2 - \boldsymbol{p}_i = \frac{1}{6}(\boldsymbol{c}_{i,1} + \boldsymbol{c}_{i,2}) - \frac{1}{3}\boldsymbol{c}_{i,3} = \frac{1}{2}\alpha_2\vec{\gamma}_2, \tag{5.9}$$
where α_2 is a scaling factor.

3. We let the tangent to the outgoing curve be parallel to $\vec{\gamma}_1$:
$$\boldsymbol{u}_1 - \boldsymbol{p}_i = \frac{1}{6}(\boldsymbol{c}_{i,2} + \boldsymbol{c}_{i,3}) - \frac{1}{3}\boldsymbol{c}_{i,1} = \frac{1}{2}\alpha_1\vec{\gamma}_1, \tag{5.10}$$
where α_1 is again a scaling factor.

The solution of (5.1), (5.9) and (5.10) for $\boldsymbol{c}_{i,j}$ in terms of the unknown α_1 and α_2 yields

$$\left\{\begin{array}{lcl} \boldsymbol{c}_{i,1} & = & \boldsymbol{p}_i - \alpha_1\vec{\gamma}_1 \\ \boldsymbol{c}_{i,2} & = & \boldsymbol{p}_i + \alpha_1\vec{\gamma}_1 + \alpha_2\vec{\gamma}_2 \\ \boldsymbol{c}_{i,3} & = & \boldsymbol{p}_i - \alpha_2\vec{\gamma}_2 \end{array}\right. \tag{5.11}$$

These equations express interpolation at the corner point V_i and leave two degrees of freedom, α_1 and α_2. Similar expressions can be found for the case when the corner at V_i is obtuse. There are two possibilities, depicted on Figure 5.4. Note that any corner of a uniform triangulation can be mapped onto one of these three cases (Figures 5.3 and 5.4) by rotating and mirroring.

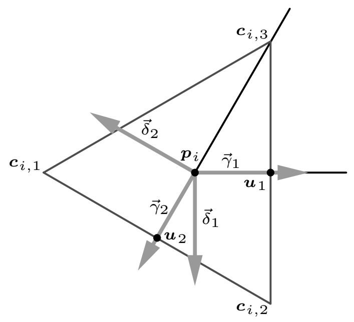


Figure 5.3: Interpolation at the corner vertices: a sharp corner.

The β–factors at the corner vertices

We considered two types of degrees of freedom: first, the scaling factors of the $\vec{\gamma}$–vectors (α–factors), and second, those of the $\vec{\delta}$–vectors (β–factors). The control triangle at an inner vertex of $\partial\Delta$ is determined by one degree of freedom of each type. The control triangle at a corner vertex is determined by two degrees of freedom of the first type. However, we can also write relations between the control points of T_i at a corner vertex and the $\vec{\delta}$–vectors, containing β–factors. Assuming that $\vec{\gamma}_1, \vec{\gamma}_2, \vec{\delta}_1$ and $\vec{\delta}_2$ are coplanar, we have

$$\boldsymbol{c}_{i,2} - \boldsymbol{c}_{i,3} = 2\beta_1\vec{\delta}_1 \tag{5.12}$$

$$\boldsymbol{c}_{i,1} - \boldsymbol{c}_{i,2} = 2\beta_2\vec{\delta}_2. \tag{5.13}$$

This allows us to calculate the β–factors at the corner vertices, when the α–factors are known. Combining (5.11), (5.12) and (5.13) yields

$$\beta_1 = \alpha_2(\vec{\gamma}_2 \cdot \vec{\delta}_1), \quad (5.14)$$
$$\beta_2 = -\alpha_1(\vec{\gamma}_1 \cdot \vec{\delta}_2). \quad (5.15)$$

One may imagine cases where $\vec{\gamma}_1 || \vec{\gamma}_2$, such that $\beta_1 = \beta_2 = 0$ according to (5.14)–(5.15). This indicates that the junction point is not a real corner, and it should be signaled to the user, who may try to join the two curves and replace them by a single one. For obtuse corners, the β–factors are related to the α–factors in the same way.

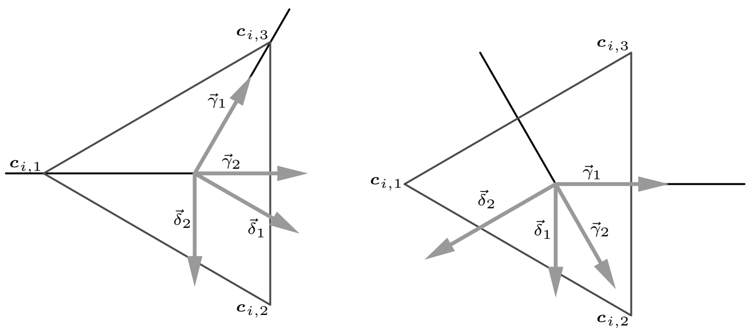


Figure 5.4: Interpolation at the obtuse corners of the patch.

5.2.3 The fitting equations

We will now use the degrees of freedom that we have determined, in order to fit the user supplied data in between each pair of adjacent interpolating vertices $V_i, V_j \in \partial\Delta$. First, the α–factors at V_i and V_j will be determined by trying to interpolate the curve $\boldsymbol{p}$ at the edge midpoint $V_{i,j} = \frac{1}{2}(V_i + V_j)$. Let $\boldsymbol{p}_{i,j}$ be the given curve point to be fitted. Using (5.6), the fitting condition $\boldsymbol{r}_{i,j} = \boldsymbol{p}_{i,j}$ through (5.5) results in

$$\begin{aligned} \alpha_i \vec{\gamma}_i - \alpha_j \vec{\gamma}_j &= 4\boldsymbol{p}_{i,j} - 2(\boldsymbol{p}_i + \boldsymbol{p}_j) \\ &= \boldsymbol{q}_{i,j}, \end{aligned} \quad (5.16)$$

where $\boldsymbol{p_{i,j}}$ is the given curve point and $\boldsymbol{q}_{i,j}$ is a shorthand notation. Next, the β–factors at V_i and V_j are obtained by fitting the cross–boundary tangent vector at $V_{i,j}$. Suppose that β_i and β_j are given, and that by subdivision a control triangle at $V_{i,j}$ is calculated. Using (4.27),(4.28) and (5.7), the subdivision rule can be expressed in terms of β–factors and cross–boundary tangent vectors:

$$\beta'_{i,j}\vec{\delta}'_{i,j} = \frac{1}{4}(\beta_i\vec{\delta}_i + \beta_j\vec{\delta}_j), \tag{5.17}$$

where $\vec{\delta}'_{i,j}$ is the cross–boundary tangent vector to the UPS–spline surface $\boldsymbol{s}(u)$ at $V_{i,j}$. The fitting condition for the cross–boundary tangent vector is then given by

$$\beta_{i,j}\vec{\delta}_{i,j} = \frac{1}{2}(\beta_i\vec{\delta}_i + \beta_j\vec{\delta}_j). \tag{5.18}$$

5.3 The algorithm

In this section we will use the fitting equations for iteratively determining a solution to the polygonal hole problem. We will restrict the figures illustrating the algorithm to the case of a triangular hole, although it is immediately applicable to cases with four, five and six boundary curves as well (see Section 5.5).

The idea is to calculate, during a pre–iteration step, an initial solution which is smooth, but in general not close enough, and to refine this approximation iteratively in order to obtain a better fit to the given curves. This refinement phase operates on the boundary of Ω. At each step, Δ is refined by mid–edge subdivision. The control triangles from the previous steps are scaled by (4.24) (they already interpolate the given data and scaling them doesn't change this), and new ones are calculated in between, using the fitting equations. After the last iteration, we have a solution which is close enough on $\partial\Omega$. The calculation of the interior control triangles may be postponed until then, or they may be computed and updated together with the solution on the boundary (see Section 5.4). Figure 5.5 (from left to right, top to bottom) illustrates this: imagine a pre–iteration step, two refinement steps and a post–iteration step actually filling the hole. The control triangles added during a particular step have been shaded.

5.3.1 An initial solution

The initial solution (Figure 5.5, top left) is obtained by imposing interpolation conditions of the form (5.11) at the corners of the patch. The degrees of freedom (two α–factors per vertex) are found by solving (5.16) in the least squares sense for each edge V_iV_j of the patch. If we assume that $\vec{\gamma}_i \neq \pm\, \vec{\gamma}_j$, then

$$\alpha_i \;=\; \frac{1}{D}\left((\vec{\gamma}_i \cdot \boldsymbol{q}_{i,j}) - (\vec{\gamma}_j \cdot \boldsymbol{q}_{i,j})(\vec{\gamma}_i \cdot \vec{\gamma}_j)\right) \tag{5.19}$$

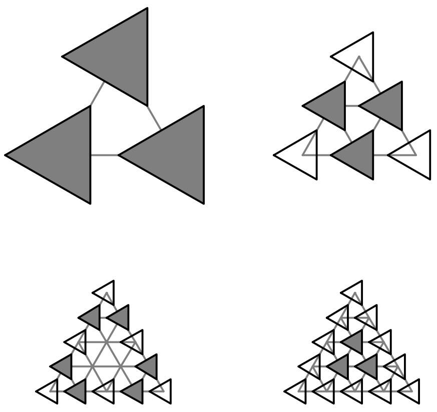


Figure 5.5: Consecutive iteration steps from left to right, top to bottom. The first step involves the calculation of the initial solution. It is followed by two iteration steps, during which the approximation on the boundary is refined, and one post–iteration step for calculating the interior control triangles.

$$\alpha_j \quad = \quad \frac{1}{D}\left(-(\vec{\gamma}_j \cdot \boldsymbol{q}_{i,j}) + (\vec{\gamma}_i \cdot \boldsymbol{q}_{i,j})(\vec{\gamma}_i \cdot \vec{\gamma}_j)\right), \tag{5.20}$$

where $D = 1 - (\vec{\gamma}_i \cdot \vec{\gamma}_j)^2 \neq 0$. If, on the contrary, $\vec{\gamma}_i = \pm\, \vec{\gamma}_j$, then (5.16) has no solution in the least–squares sense. First, consider the case when $\vec{\gamma}_i || (\boldsymbol{p}_j - \boldsymbol{p}_i)$ and $\vec{\gamma}_j = \vec{\gamma}_i$. It is plausible then to assume that $\boldsymbol{s}_{i,j}$ is a straight line from $\boldsymbol{s}(V_i)$ to $\boldsymbol{s}(V_j)$. The α–factors can then be determined as follows (see Figure 5.2):

$$\boldsymbol{u}_i = \boldsymbol{p}_i + \frac{\alpha_i}{2}\vec{\gamma}_i \quad \text{and} \quad ||\boldsymbol{u}_i - \boldsymbol{p}_i|| = \frac{1}{4}||\boldsymbol{p}_j - \boldsymbol{p}_i|| \tag{5.21}$$

such that

$$\alpha_i = \alpha_j = \frac{1}{2}||\boldsymbol{p}_j - \boldsymbol{p}_i||. \tag{5.22}$$

An initial solution for other cases where $\vec{\gamma}_i = \vec{\gamma}_j$ (see Figure 5.6) or $\vec{\gamma}_i = -\vec{\gamma}_j$ (see Figure 5.7) can not be determined unambiguously relying on the fitting equations. In order not to over–complicate the calculation of the initial solution by introducing more and more exceptional cases, every time that $\vec{\gamma}_i = \pm\vec{\gamma}_j$ we determine α_i and α_j using (5.22). The boundary curve $\boldsymbol{e}_{i,j}$ is then a piecewise Bézier curve that interpolates the given data $(\boldsymbol{p}_i, \vec{\gamma}_i)$ and

$(\boldsymbol{p}_j, \vec{\gamma}_j)$ at V_i and V_j (see Figures 5.6 and 5.7), and is exact in the case of a straight line.

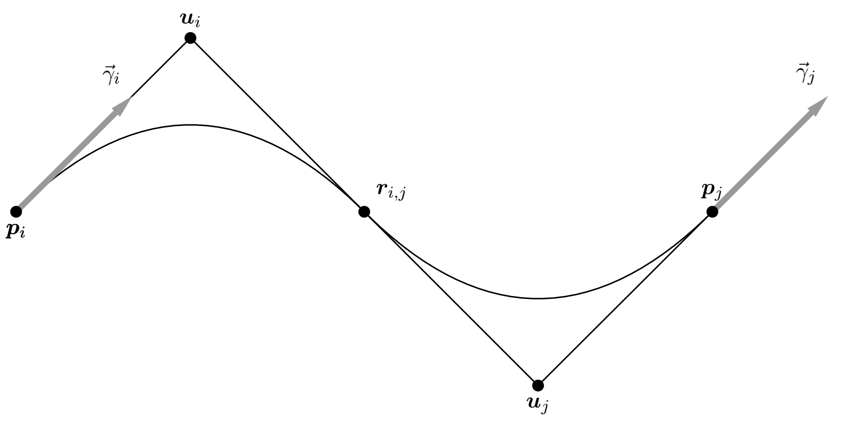


Figure 5.6: An initial solution for the case when $\vec{\gamma}_i = \vec{\gamma}_j$.

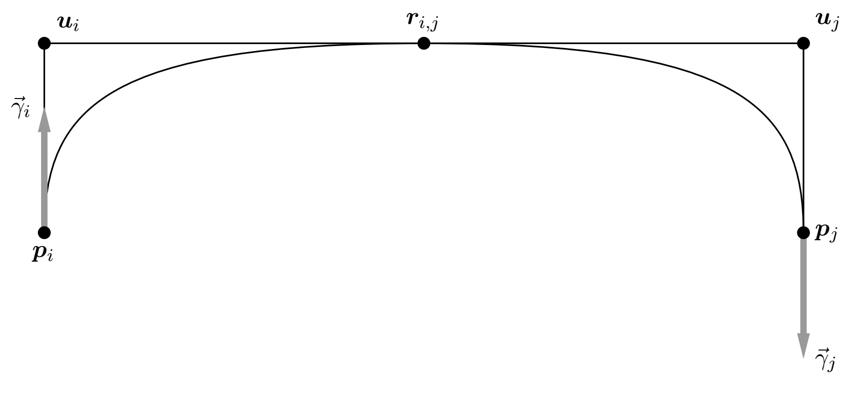


Figure 5.7: An initial solution for the case when $\vec{\gamma}_i = -\vec{\gamma}_j$.

5.3.2 The iteration step

First the control triangles from the previous steps are scaled by subdivision. This is simply done by scaling down the α– and β–factors: $\alpha_i \leftarrow \frac{\alpha_i}{2}$ and $\beta_i \leftarrow \frac{\beta_i}{2}$, for each $V_i \in \partial\Delta$.

Next, a new control triangle is created in between any two adjacent vertices at the coarser level. This situation is illustrated on Figure 5.8, where the shaded triangles are known. The middle control polygon interpolates the given data at $\boldsymbol{s}(V_k)$, $V_k = \frac{1}{2}(V_i + V_j)$. It has two degrees of freedom, α_k and β_k. Consider the α–factor first. In order to obtain a better fit, we try to interpolate the user defined boundary curve at $V_{i,k} = \frac{1}{2}(V_i + V_k)$ and $V_{k,j} = \frac{1}{2}(V_k + V_j)$. Referring to (5.16), this yields a set of fitting equations

$$\left\{ \begin{array}{rcl} \alpha_i \vec{\gamma}_i - \alpha_k \vec{\gamma}_k & = & \boldsymbol{q}_{i,k} \\ \alpha_k \vec{\gamma}_k - \alpha_j \vec{\gamma}_j & = & \boldsymbol{q}_{k,j} \end{array} \right. \tag{5.23}$$

where α_i and α_j are known. Thus, α_k can be obtained as the least-squares solution of (5.23):

$$\alpha_k = \frac{1}{2}(\vec{\gamma}_k \cdot (\alpha_i \vec{\gamma}_i - \alpha_j \vec{\gamma}_j + \boldsymbol{q}_{k,j} - \boldsymbol{q}_{i,k})). \tag{5.24}$$

The β_k–factor is found by fitting the cross–boundary vectors at $V_{i,k}$ and $V_{k,j}$. Referring to (5.18), this yields the following set of fitting equations:

$$\left\{ \begin{array}{rcl} \beta_{i,k} \vec{\delta}_{i,k} & = & \frac{1}{2}(\beta_i \vec{\delta}_i + \beta_k \vec{\delta}_k) \\ \beta_{k,j} \vec{\delta}_{k,j} & = & \frac{1}{2}(\beta_k \vec{\delta}_k + \beta_j \vec{\delta}_j). \end{array} \right. \tag{5.25}$$

This is an overdetermined system where β_i and β_j are known. It can be solved for β_k in the least–squares sense while the unknowns $\beta_{i,k}$ and $\beta_{k,j}$ are not used. This yields

$$\beta_k = \frac{\beta_i(\vec{\delta}_i \cdot \vec{\delta}_{i,k})(\vec{\delta}_k \cdot \vec{\delta}_{i,k}) + \beta_j(\vec{\delta}_j \cdot \vec{\delta}_{k,j})(\vec{\delta}_k \cdot \vec{\delta}_{k,j}) - \beta_i(\vec{\delta}_i \cdot \vec{\delta}_k) - \beta_j(\vec{\delta}_j \cdot \vec{\delta}_k)}{2 - (\vec{\delta}_k \cdot \vec{\delta}_{i,k})^2 - (\vec{\delta}_k \cdot \vec{\delta}_{k,j})^2}. \tag{5.26}$$

If $\vec{\delta}_{i,k} = \vec{\delta}_k = \vec{\delta}_{k,j}$, for example when the boundary curve is the intersection of the surrounding surface and a plane, this system has no solution in the least–squares sense. An appropriate value for the β_k factor can then easily be obtained by subdivision. Referring to (5.18), β_k is then given by

$$\beta_k = \frac{1}{2}\left(\vec{\delta}_k \cdot (\beta_i \vec{\delta}_i + \beta_j \vec{\delta}_j)\right). \tag{5.27}$$

Once that α_k and β_k are determined, the control points $\boldsymbol{c}_{k,1}, \boldsymbol{c}_{k,2}$ and $\boldsymbol{c}_{k,3}$ can be calculated by an expression of the form (5.8). At the end of the iteration step, the interpolation conditions are satisfied at V_i, V_j and V_k, and approximately at the points $V_{i,k}$ and $V_{k,j}$.

5.3.3 The stopping criterion

In our implementation, the user can choose between a fixed number of iterations, or he can iterate until the maximum residual is smaller than a

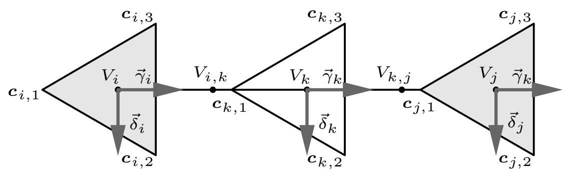


Figure 5.8: The iteration step: Δ is subdivided. The control triangles at the original vertices are scaled, and the α– and β–factors of the new control triangles are found by solving the fitting equations.

certain value ϵ. Therefore, each time we solve the fitting equations (5.23), the residuals $r_{i,k} = \|\boldsymbol{q_{i,k}} - \alpha_i\vec{\gamma}_i + \alpha_k\vec{\gamma}_k\|$ and $r_{k,j}$ similarly, are calculated. The maximum residual for each edge is stored in memory and updated at each refinement step. If at the current refinement level, for a certain edge, $r_{i,j}^{\max} < \epsilon$, then the solution on this edge is subdivided during the next refinement step, instead of calculating the new control triangles by solving the fitting equations again and again.

5.4 Finding the interior control points

Untill now, we haven't addressed the question of how to determine the interior control triangles. Recall that in general, the objective is to determine a smooth patch, fitting the given boundary curves well enough. In Section 5.4.2 we will describe some more or less intuitive approaches. They will try to find a compromise that combines the smoothness of the initial solution with the good approximation properties of the final solution on the boundary in an efficient way. The remainder of the section however discusses a less cheap, but mathematically more sound approach: constrained fairing of UPS–spline surfaces. In Section 5.4.3 the fairness of a UPS–spline surface is expressed in mathematical terms. Optimizing the fairness of the filling patch involves finding the minimizer of a certain objective function, subject to a number of constraints which express that the boundary control triangles are already given. But first we will have a look at how the quality of these different methods can be compared.

We will discuss the various possibilities by the help of an example. In Section 3.6.3 we derived a NURPS representation for patches on the sphere. Figure 5.9, left, shows two such patches, constituting a sphere sextant with a hole around the equator plane. This hole will be filled by two triangular patches, as shown in Figure 5.9, right. Their common boundary curve is

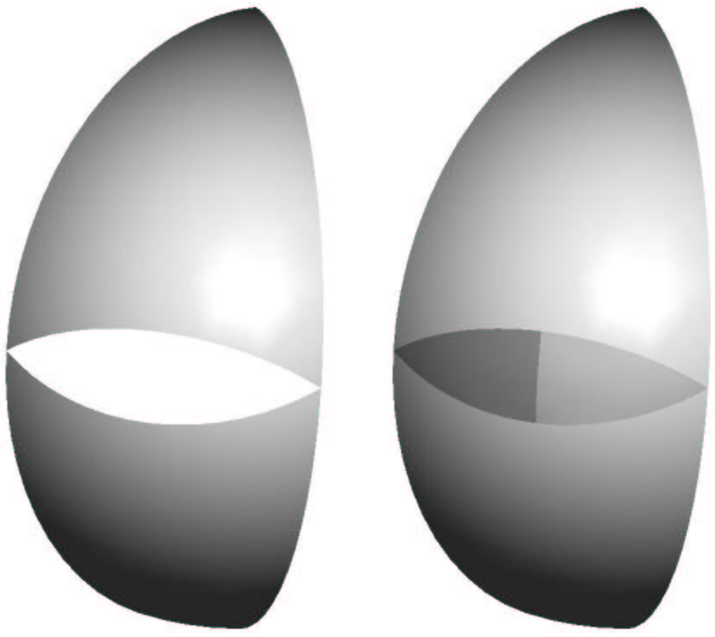


Figure 5.9: The hole and the triangular patches.

fitted to the great circle bisecting the hole.

5.4.1 UPS–spline surface interrogation

A surface may look perfect in its wireframe and its shaded representation, nevertheless it may have certain imperfections. Curve and surface interrogation tools are used to detect those imperfections, in order to check whether a design satisfies its functionality and aesthetic shape requirements. A variety of surface interrogation algorithms have been developed [33, 54], which emphasize different geometric surface properties, such as geometric continuity, special points, convexity conditions, strong curvature variations,... We shall explain the surface interrogation tools that we have implemented for UPS–spline surfaces here without going into detail. For further information, we refer to any standard textbook on differential geometry, or to the works mentioned above.

A common tool for interrogating surfaces is the use of curvature plots. The most important curvature types for surfaces are the Gaussian curvature K and the mean curvature H. For a given UPS–spline surface $\boldsymbol{s}(u), u = (u_1, u_2) \in \Omega$, these can be calculated as (see, e.g., [54])

$$K = \frac{LN - M^2}{EG - F^2} \tag{5.28}$$

and

$$H = \frac{NE - 2MF + LG}{EG - F^2} \tag{5.29}$$

with

$$\begin{aligned}
E &= \frac{\partial \boldsymbol{s}(u)}{\partial u_1} \cdot \frac{\partial \boldsymbol{s}(u)}{\partial u_1} && (5.30)\\
F &= \frac{\partial \boldsymbol{s}(u)}{\partial u_1} \cdot \frac{\partial \boldsymbol{s}(u)}{\partial u_2} && (5.31)\\
G &= \frac{\partial \boldsymbol{s}(u)}{\partial u_2} \cdot \frac{\partial \boldsymbol{s}(u)}{\partial u_2} && (5.32)\\
L &= \boldsymbol{n}(u) \cdot \frac{\partial^2 \boldsymbol{s}(u)}{\partial u_1^2} && (5.33)\\
M &= \boldsymbol{n}(u) \cdot \frac{\partial^2 \boldsymbol{s}(u)}{\partial u_1 \partial u_2} && (5.34)\\
N &= \boldsymbol{n}(u) \cdot \frac{\partial^2 \boldsymbol{s}(u)}{\partial u_2^2}. && (5.35)
\end{aligned}$$

For the evaluation of the partial derivatives we refer to Section 2.3.3. The calculation of the normal vectors $\boldsymbol{n}(u)$ was discussed in Section 2.3.5.

The Gaussian and mean curvatures are related to the principal curvatures κ_1 and κ_2 at the surface point under consideration:

$$K = \kappa_1 \kappa_2 \quad \text{and} \quad H = \frac{1}{2}(\kappa_1 + \kappa_2). \tag{5.36}$$

Another useful type of curvature is the absolute curvature $\kappa_{\text{abs}} = |\kappa_1| + |\kappa_2|$. It may be used in cases where Gaussian and mean curvature do not offer enough information, that is, for cylindric surfaces, resp. minimal surfaces. Finally we note that Gauss's Theorema Egregium states that the Gaussian curvature depends on the intrinsic geometry of the surface, and, e.g., not on the parameterization. In order to generate curvature plots, the curvature κ (which can be any of the previously mentioned curvatures, and in general lies anywhere between $-\infty$ and $+\infty$) is normalized to κ_{norm} in the interval $[-1, 1]$ using

$$\kappa_{\text{norm}} = \text{sgn}(\kappa)(1 - e^{-C||\kappa||}), \tag{5.37}$$

where C is a factor for controlling the contrast in the visualization that should be determined interactively [72]. This simplifies mapping the curvature to a color code. For example, in order to generate plots in black and white, we let the color vary linearly from black for $\kappa_{\text{norm}} = -1$ to white for $\kappa_{\text{norm}} = 1$. We shall compare the results of the different approaches for determining interior control triangles using Gaussian curvature plots, directly on the surface. As for the detection of non–smooth curvature variations, the pictures of the other curvature types are similar. Light grey regions will indicate elliptic points; darker grey zones will indicate hyperbolic points. Strong curvature variations may be detected by non–smooth transitions in the curvature plots. Figure 5.10 shows the Gaussian curvature plot of a B–spline basis function. It is a typical plot for smooth piecewise polynomial splines. Note that in our example, we try to fill in a hole in a sphere. Ideally, the filling patch would be on the sphere too, and its curvature plot would be monotonous. We know that this is impossible since the filling patch is non–rational. The purpose however remains to find a patch with an even curvature plot.

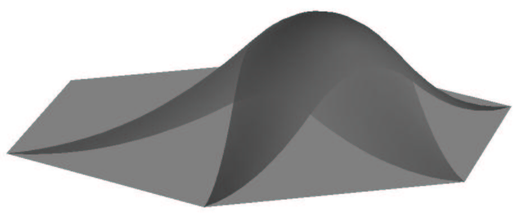


Figure 5.10: The Gaussian curvature plot of a B–spline basis function.

5.4.2 Interior control triangles: the intuitive approach

In this section, we will try to combine the initial solution (as given in Section 5.3.1), which is smooth, and the solution on the boundary, which approximates the given curves well, for calculating the interior control triangles.

Copy from initial solution The interior control points are obtained directly from the initial solution by subdivision. This guarantees that the interior of the patch is smooth. A disadvantage however is that the interior of the initial solution in general has no connection with the shape of the edge curves, since it interpolates the given curves at the corners of the patch only. For example, Figure 5.11 shows a situation where a triangular filling patch is calculated, given three boundary curves with their tangent and normal vectors. The initial solution is planar, because the tangent vectors at the corner vertices are coplanar. Two of the boundary curves are within the same plane, but the third one is not. Therefore the initial solution fails to fit the latter boundary curve. It has only its endpoints in the plane of the initial solution. When applying subsequent subdivision, all of the interior control triangles remain in the plane of the initial solution, while the ones on the boundary will tend to follow the given curves, as in Figure 5.11, top (the rendered surface is given left, the Gaussian curvature plot right), while the user would rather expect a filling patch as in the bottom of the figure. Clearly, this method can cause unwanted artifacts near the

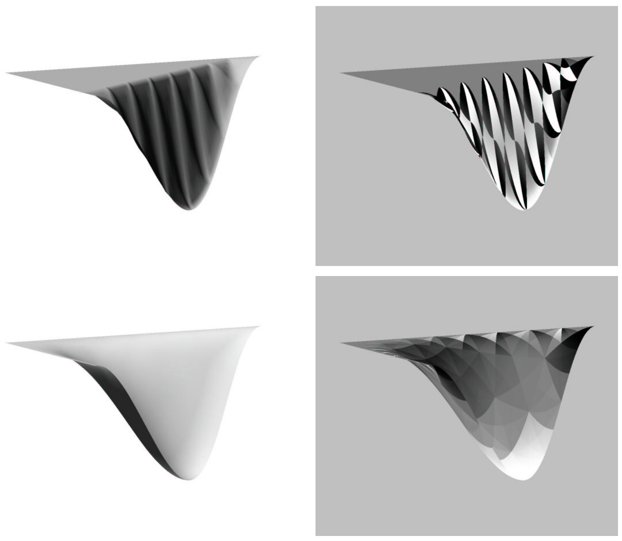


Figure 5.11: The "Copy from initial" method fails if the initial solution does not fit the given boundaries well. Top: the output of the "Copy from initial" method for a case where the initial solution is planar. Bottom: The output that the user would expect. Left: the rendered surfaces, Right: Gaussian curvature plots.

boundary. On Figure 5.12 it is used to fill the hole in the sphere sextant. The curvature plot indicates strong curvature variations near the boundary. The next option will therefore take edge features into account towards the interior of the patch.

Average surrounding control triangles This method is performed as a post–iteration step, i.e., after the solution on the boundary has been obtained. We will fill the hole gradually by calculating a ring of control triangles during each pass, going from the edge towards the inner of the

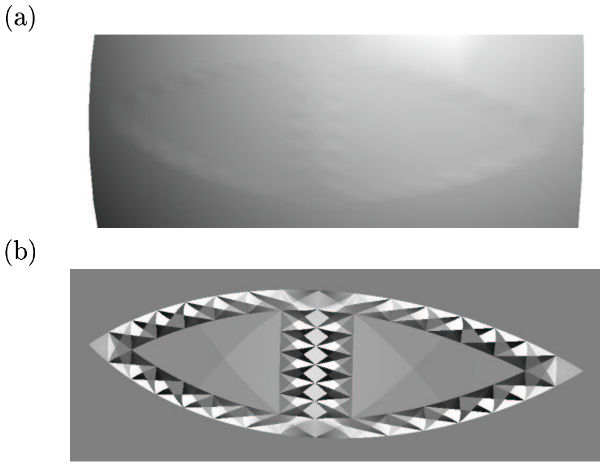


Figure 5.12: The interior control triangles are obtained by copying them from the initial solution. (a): Rendered surface, (b): Gaussian curvature plot.

patch. Figure 5.13 shows an example where each ring has a different shade of grey. At each step, a control triangle of the current ring is obtained by averaging six surrounding control triangles. These come from the initial solution, or, if possible, from a previously calculated ring. Edge features are now smoothed out towards the inner of the patch. However, there is a main disadvantage to this approach, if averaging is applied after the last iteration step: the unwanted artifacts mentioned before are now repeated for every ring, smoothed out towards the inner of the surface, as shown on Figure 5.14. The curvature plot shows that high curvature variations occur all over the patch.

Instant update A good compromise would be to take edge features into account before we finish iterating. This can be accomplished by subdividing the initial solution at each refinement step, but, we always overwrite its edge with the most recent boundary approximation. By doing so, the edge features are taken into account as soon as they become available in the approximation, and the interior control triangles are calculated together with the ones on the boundary, instead of during a post–iteration step. The results of this strategy are depicted in Figure 5.15. The curvature plot indicates less variation, but still there are some regions with a non–smooth behaviour.

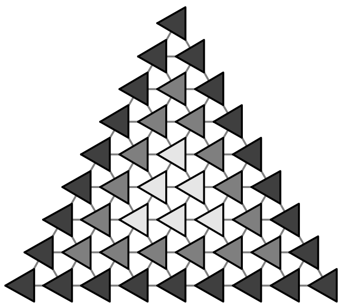


Figure 5.13: Averaging: the interior control points are calculated gradually during a post–iteration step. In each pass, a ring of control triangles is calculated, going from the boundary towards the inner of the patch. Each control triangle is found by averaging the six surrounding triangles. The rings are shaded in different tones of grey.

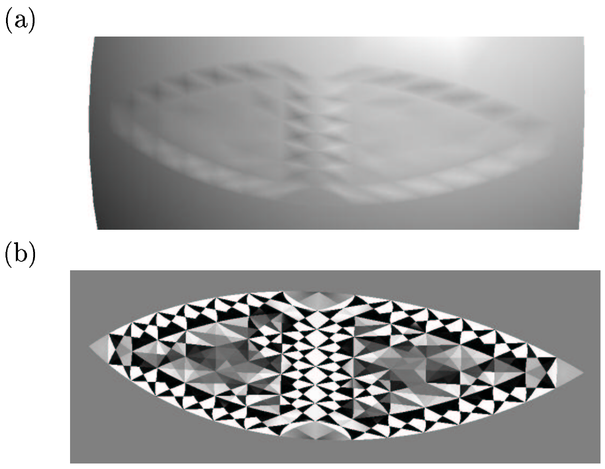


Figure 5.14: Calculation of the interior control triangles by averaging the six surrounding ones. (a): Rendered surface, (b): Gaussian curvature plot.

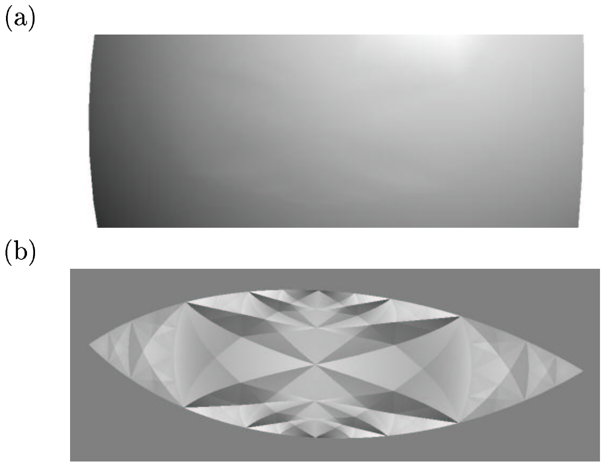


Figure 5.15: Calculation of the interior control triangles by subdivision of the initial solution, where the boundary has been overwritten with the most recent approximation. (a): Rendered surface, (b): Gaussian curvature plot.

Bridging Finally, a different approach based on k–subdivision (see Section 4.4.3) tries to build bridges between opposite control triangles on the patch boundaries. It uses information of the final solution on the boundary only, not of the initial solution. It is an approach that has to be performed during a post–iteration step. There are three main phases in this algorithm according to the direction in which the bridges are build. First, we have the horizontal direction, then we build bridges in the southeast–northwest direction and finally there is the southwest–northeast direction. Figure 5.16 illustrates the idea for the horizontal direction. For each horizontal row in the triangulation, labeled $k = 2, \ldots, 7$, a bridge of interior control triangles is determined by k–subdivision on the boundary control triangles. First, the boundary control triangles (shaded darker) have to be scaled up appropriately, because otherwise the interior control triangles would not be at the right refinement level. This is done by the inverse transform of (4.37)–(4.39), with $l = 0$ and the value of k as indicated in Figure 5.16. Then the horizontal bridge of interior control triangles is determined, again by (4.37)–(4.39) for $0 < l \leq k/2$, and by (4.40)–(4.42) for $k/2 < l < k$. The bridges of interior control triangles have been shaded alternating by light and dark tones in Figure 5.16 for clarity. The same idea can now be applied to the southeast–northwest direction and the southwest–northeast direction. At the end, there are three control triangles for each interior vertex V_i, one for each bridging direction. These can be averaged, delivering the final result. Figure 5.17 shows the resulting surface. Curvature variations are less prominent than in the first two cases, but still they are present all over the surface.

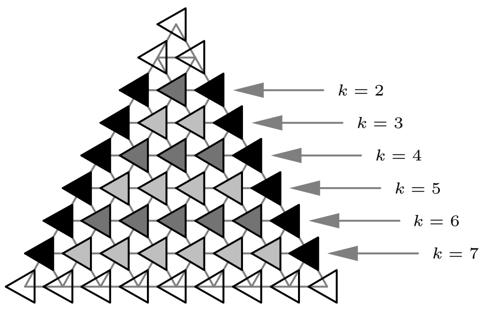


Figure 5.16: Bridging in the horizontal direction: in between each pair of opposite control triangles on the left and right boundary, a horizontal bridge of interior control triangles is build using k–subdivision. First, each pair of boundary control triangles has to be scaled up appropriately such that the new ones are on the right level of refinement. The horizontal bridges are shaded alternatingly.

Note that in any case the user can change the interior control triangles afterwards, and still he has a C^1–continuous filling patch, fitting the specified edge curves, with the demanded precision.

Smoothing via thresholding with boundary constraints Note that the surfaces that come out of the polygonal hole algorithm are wavelet transformable because they have been obtained after several subdivision steps followed by modifications of the control points. It is therefore possible to apply the thresholding algorithm from Section 4.5.6 in order to smooth the interior of the patches obtained in the preceding sections. However, the wavelet coefficients corresponding to the control triangles on and next to the boundary should not be thresholded, otherwise we would loose the good approximation of the boundary curves after the update step. Therefore, we build in our thresholding algorithm the option to leave the boundaries untouched. However, this is rather a severe restriction. Our experiments show that for appropriate thresholds a small improvement at the center of the patch can be achieved, but it is very limited. For example, Figure 5.18 shows a curvature plot of the filling surface using the bridging method. Recall from Figure 5.9 that it consists of two triangular patches. The left

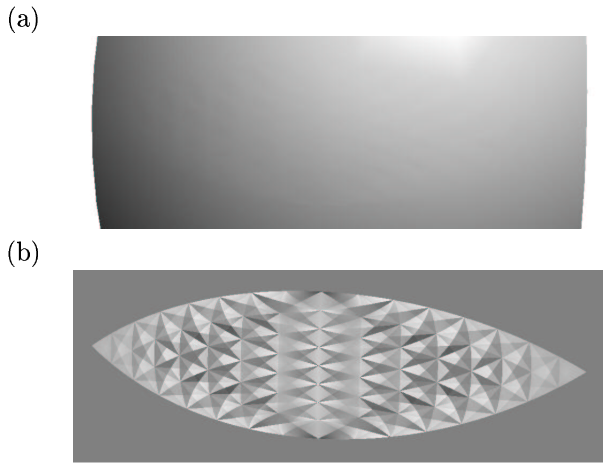


Figure 5.17: Calculation of the interior control triangles by building bridges between opposite boundary control triangles using k–subdivision. (a): Rendered surface, (b): Gaussian curvature plot.

patch has been smoothed by thresholding, and the right one has been left untouched for comparison. There is little improvement, at the interior of the patch only.

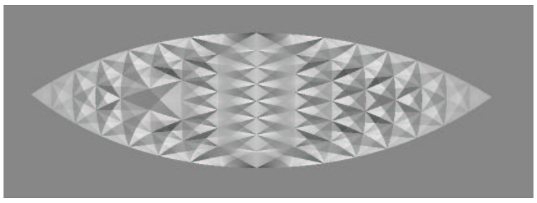


Figure 5.18: Smoothing via thresholding with boundary constraints.

5.4.3 The fairing problem for UPS—splines

Dierckx et al. [20] adapted the philosophy of the tensor product smoothing spline algorithm [16, 18] for deriving smoothness conditions for PS–splines in a least–squares surface fitting problem. The idea was to measure the lack of smoothness by imposing conditions for $s(u)$ to be a single polynomial on Ω. Consider the triangle $\mathcal{T}(V_1, V_2, Z)$ divided into two subtriangles from the PS–refinement, say $\rho^*(V_1, R_{1,2}, Z)$ and $\rho^*(V_2, R_{1,2}, Z)$, with $R_{1,2} = \lambda_1 V_1 + \lambda_2 V_2$. Denote the Bézier ordinates of $s(u)$ on $\rho^*(V_1, R_{1,2}, Z)$

and $\rho^*(V_2, R_{1,2}, Z)$ as on Figure 5.19. Since $s(u) \in S_2^1(\Delta^*)$, it is C^1–continuous. In order for $s(u)$ to be a single polynomial on Ω, we impose a C^2–continuity condition across $ZR_{1,2}$. By (2.24), this is

$$\lambda_1^2(b_{2,0,0} - 2b_{1,1,0} + b_{0,2,0}) - \lambda_2^2(b_{0,2,0} - 2\hat{b}_{1,1,0} + \hat{b}_{0,2,0}) = 0. \qquad (5.38)$$

Note that this equation only involves the Bézier ordinates on the edge V_1V_2, although it ensures C^2–continuity across $ZR_{1,2}$. Likewise, if there is a similar meeting of two triangles $\rho^*(V_1, R_{1,2}, Z^*)$ and $\rho^*(V_2, R_{1,2}, Z^*)$, the same condition will also ensure C^2–continuity across $Z^*R_{1,2}$. A similar one involving the Bézier ordinates on ZZ^* will ensure C^2–continuity across $V_1R_{1,2}$ and $V_2R_{1,2}$. Therefore, there will be one such condition for each edge in Δ and one condition for each line joining the interior points Z_j and Z_l of adjacent triangles ρ_j and $\rho_l \in \Delta$, for $s(u)$ to be a single polynomial on Ω.

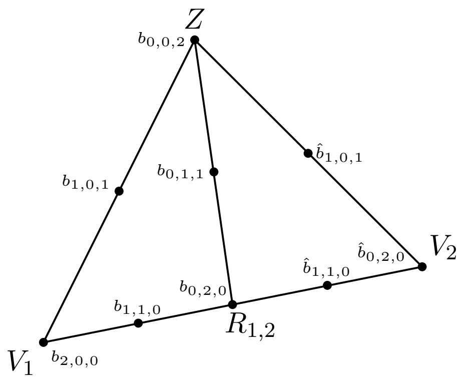


Figure 5.19: C^2 continuity across the common edge $ZR_{1,2}$.

We will now look for a solution to the following problem: consider a UPS–spline surface on a triangulation Δ of Ω. The control triangles at the border vertices $V_i \in \partial\Delta$ are supposed to be known, and the remainder of the control triangles is to be determined such that the resulting surface is fair. This can be applied for calculating the interior control triangles in the polygonal hole problem.

To begin with, we examine the particular case that Δ is a molecule M having molecule number $m = 6$ (Figure 5.20). The vertices are denoted V (at the center) and V_i, $i = 1, \ldots, 6$ (adjacent to V, at the border). The figure shows the PS–triangles at each vertex, but we will think of those as schematic representations of the control triangles. Now suppose that the control points $\boldsymbol{c}_{i,j}, i = 1, \ldots, 6, j = 1, 2, 3$ are given (corresponding to the lighter shaded control triangles). The problem, then, is to find the control triangle $T(\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3)$ at V such that the resulting surface is fair. This clearly is a constrained fairing problem; the constraints being the set of given control triangles at the vertices adjacent to V. The fairing criterion will be to minimize the jump in second order derivatives across the interior edges of Δ and the PS–refinement lines of Δ^*. To do so, we impose each component

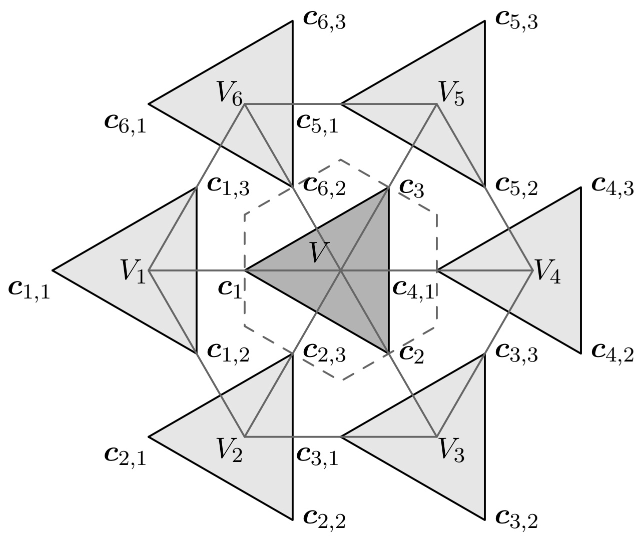


Figure 5.20: The fairing problem for UPS–spline surfaces on a molecule with $m = 6$: given the control points of the border control triangles (shaded lighter), determine the control points of the inner control triangle (shaded darker) so as to obtain a fair surface.

of $\boldsymbol{s}(u)$ (2.64) to be a single polynomial on M, by writing down one condition of the form (5.38), for each edge VV_i (using Bézier control points instead of Bézier ordinates), and a similar one for the PS–refinement line intersecting that edge (the latter are drawn as dashed lines on the figure). For example, let us have a look at VV_1 and the PS–refinement line Z_1Z_2 intersecting it (Figure 5.21). The Bézier control points along these PS–refinement lines are denoted $\boldsymbol{b}_j, j = 1, \ldots, 9$, and are displayed schematically in the picture. They can be expressed in terms of the B–spline control points at V and V_1

by (4.10)–(4.16):

$$\begin{aligned}
\boldsymbol{b}_1 &= \frac{1}{3}(\boldsymbol{c}_1 + \boldsymbol{c}_2 + \boldsymbol{c}_3) && (5.39)\\
\boldsymbol{b}_2 &= \frac{2}{3}\boldsymbol{c}_1 + \frac{1}{6}(\boldsymbol{c}_2 + \boldsymbol{c}_3) && (5.40)\\
\boldsymbol{b}_3 &= \frac{1}{3}\boldsymbol{c}_1 + \frac{1}{12}(\boldsymbol{c}_2 + \boldsymbol{c}_3) + \frac{1}{4}(\boldsymbol{c}_{1,2} + \boldsymbol{c}_{1,3}) && (5.41)\\
\boldsymbol{b}_4 &= \frac{1}{2}(\boldsymbol{c}_{1,2} + \boldsymbol{c}_{1,3}) && (5.42)\\
\boldsymbol{b}_5 &= \frac{1}{3}(\boldsymbol{c}_{1,1} + \boldsymbol{c}_{1,2} + \boldsymbol{c}_{1,3}) && (5.43)\\
\boldsymbol{b}_6 &= \frac{2}{9}(\boldsymbol{c}_{1,3} + \boldsymbol{c}_{6,2} + \boldsymbol{c}_1) + \frac{1}{9}(\boldsymbol{c}_{1,2} + \boldsymbol{c}_{6,1} + \boldsymbol{c}_3) && (5.44)\\
\boldsymbol{b}_7 &= \frac{1}{3}(\boldsymbol{c}_1 + \boldsymbol{c}_{1,3}) + \frac{1}{6}(\boldsymbol{c}_3 + \boldsymbol{c}_{1,2}) && (5.45)\\
\boldsymbol{b}_8 &= \frac{1}{3}(\boldsymbol{c}_1 + \boldsymbol{c}_{1,2}) + \frac{1}{6}(\boldsymbol{c}_2 + \boldsymbol{c}_{1,3}) && (5.46)\\
\boldsymbol{b}_9 &= \frac{2}{9}(\boldsymbol{c}_{1,2} + \boldsymbol{c}_{2,3} + \boldsymbol{c}_1) + \frac{1}{9}(\boldsymbol{c}_{1,3} + \boldsymbol{c}_{2,3} + \boldsymbol{c}_2). && (5.47)
\end{aligned}$$

The C^2–continuity condition (5.38) along VV_1 can now be written in terms of the B–spline coefficients:

$$\boldsymbol{c}_1 = -\frac{1}{3}\boldsymbol{c}_{1,1} + \frac{2}{3}\boldsymbol{c}_{1,2} + \frac{2}{3}\boldsymbol{c}_{1,3}, \qquad (5.48)$$

and the same holds for the C^2–continuity condition along Z_1Z_2:

$$\frac{2}{9}\boldsymbol{c}_2 - \frac{2}{9}\boldsymbol{c}_3 = -\frac{2}{9}\boldsymbol{c}_{1,2} + \frac{2}{9}\boldsymbol{c}_{1,3} - \frac{2}{9}\boldsymbol{c}_{6,2} - \frac{1}{9}\boldsymbol{c}_{6,1} + \frac{2}{9}\boldsymbol{c}_{2,3} + \frac{1}{9}\boldsymbol{c}_{2,1}. \qquad (5.49)$$

This process can be repeated for each edge $VV_i, i = 1, \ldots, 6$, and the PS–refinement lines intersecting them, yielding an overdetermined system with twelve equations in three unknowns,

$$A\boldsymbol{c} = R, \quad A \in \mathbb{R}^{12\times 3}, \quad \boldsymbol{c} = \begin{bmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \\ \boldsymbol{c}_3 \end{bmatrix}, \quad R \in \mathbb{R}^{12\times 1}. \qquad (5.50)$$

Solving this system in the least squares sense amounts to minimizing the extent to which the C^2–continuity conditions (5.38) are violated, or, minimizing the jump in the second order derivatives of the components of $\boldsymbol{s}(u)$, across the edges and the PS–refinement lines. That is,

$$A^T A\boldsymbol{c} = A^T R = L = \begin{bmatrix} L_1 \\ L_2 \\ L_3 \end{bmatrix} \qquad (5.51)$$

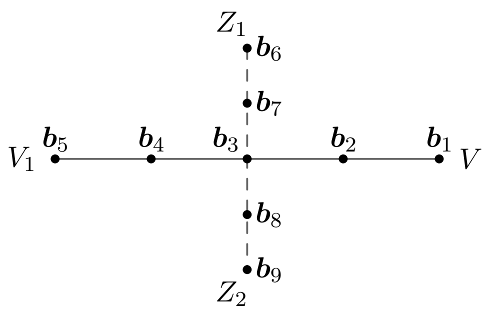


Figure 5.21: Two smoothness conditions exist for each edge VV_j, in this case VV_1. They involve a relation between the Bézier control points along the edge, and one between the Bézier control points along the PS–refinement line intersecting the edge.

with

$$A^TA = \begin{bmatrix} \frac{178}{81} & -\frac{8}{81} & -\frac{8}{81} \\ -\frac{8}{81} & \frac{178}{81} & -\frac{8}{81} \\ -\frac{8}{81} & -\frac{8}{81} & \frac{178}{81} \end{bmatrix}, \tag{5.52}$$

and therefore the solution is

$$\begin{bmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \\ \boldsymbol{c}_3 \end{bmatrix} = \begin{bmatrix} \frac{85}{186}L_1 + \frac{2}{93}(L_2 + L_3) \\ \frac{85}{186}L_2 + \frac{2}{93}(L_3 + L_1) \\ \frac{85}{186}L_3 + \frac{2}{93}(L_1 + L_2) \end{bmatrix}. \tag{5.53}$$

Fairing on a molecule may be useful for improving the averaging method from Section 5.4.2. Indeed, instead of calculating an interior control triangle by averaging the six surrounding ones, it can be determined by (5.53). The results for our test example are depicted on Figure 5.22. It shows some improvement compared with averaging (Figure 5.14), but there still are high curvature variations. We shall later use equation (5.53) for determining an initial solution in the case of polygonal holes with five or six edges.

This algorithm can now be extended to UPS–spline surfaces on uniform triangulations Δ. Let us consider an example in Figure 5.23. It depicts a triangular domain with a number of interior control triangles (shaded lighter), which are to be determined, and a number of border triangles (shaded darker) which are given. An overdetermined system expressing the

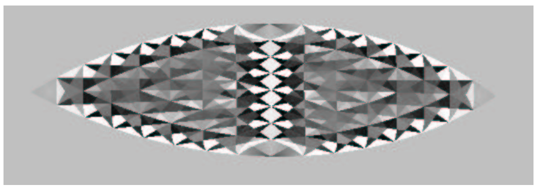


Figure 5.22: Calculation of the interior control triangles by improved averaging.

requirement that each component of $s(u)$ is a single polynomial on Ω can be determined as follows: a relation of the form (5.48) and (5.49) exists for

1. each edge connecting two interior vertices (dashed lines). This equation does not contribute to the right hand side of the system,
2. each edge connecting an interior vertex with one on the border (bold lines). Such an equation does contribute to the right hand side of the system.

For the case of Figure 5.23, let k be the number of subdivision steps that has been applied to the initial domain triangle Ω. Suppose that $k > 2$, otherwise there are no interior vertices. Then the number of border vertices along one edge of the domain triangle is given by $2^k + 1$. Straightforward calculation shows that the number of unknowns is equal to $9(2^{k-1} - 1)(2^k - 1)$ whereas the number of equations is equals $18(2^{k-1} - 1)(2^k - 3)$, so this is an overdetermined system for $k > 2$. We implemented the Givens orthogonalization method to determine the least–squares solution of the system, because it allows to process one equation at a time, keeping memory requirements low, and it allows to exploit the sparsity of the system: indeed each row has at most eight nonzero entries, whereas the number of unknowns for Figure 5.23 equals 189. Figure 5.24 shows the results for calculating the interior of the filling patch by fairing. There is a clear distinction between the curvature behaviour near the boundary and the inner of the patch, but no dramatic curvature variations occur.

Conclusion

We have examined a number of methods to calculate the interior control triangles. A generally applicable solution seems hard to obtain in the absence of further information about the interior of the patch. If the user is interested in a curvature plot with a smooth behaviour, we can recommend the fairing method. It yields a mathematically well founded solution that is optimal with respect to a certain smoothness criterion. However, the much

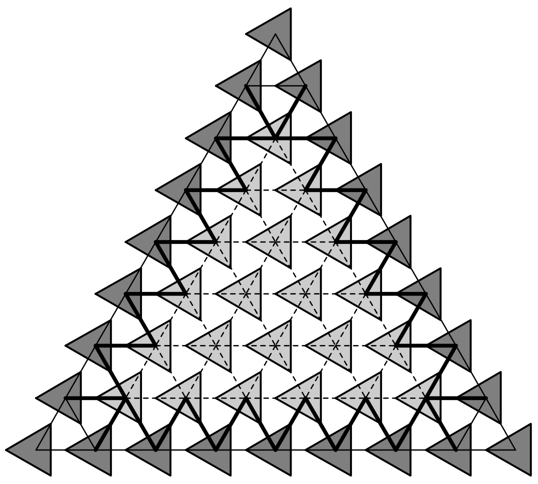


Figure 5.23: The fairing problem for a UPS–spline surface. Given the border (darker) control triangles, the interior control triangles are determined such that the surface is fair. Two C^2–continuity conditions can be written down for each bold edge (involving known and unknown control points) and for each dashed line (involving unknown control points only), yielding an overdetermined system.

cheaper "instant update" method also performs well, and may be used in cases where fairing causes the interior of the patch to be too smooth. We will encounter an example of this case in Chapter 6.

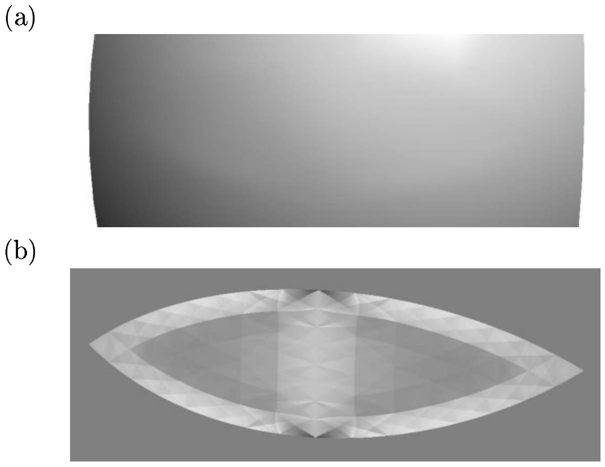


Figure 5.24: Calculation of the interior control triangles by fairing. (a): Rendered surface, (b): Gaussian curvature plot.

5.5 A note on the number of edges

Throughout the previous Section, we used the example of the incomplete sphere representation from Chapter 3 to illustrate our algorithm for filling a polygonal hole. This shows that it can be used for holes bounded by two and three curves. However, it is immediately applicable to problems with four, five and six boundary curves as well. Figure 5.25 shows the configuration of the initial solution for each of these cases. If we are working with five edges, there are two edges having a control triangle at their midpoint (shaded darker). This requires a small modification to the calculation of the initial solution for those edges. The α–factors are obtained by solving (5.23) to the unknown α_i, α_j and α_k. The β–factors of the control polygons at the corners are obtained as usual by (5.14)–(5.15); for the darker shaded polygons one can apply (5.27). Another problem is that for the cases of five and six boundary curves, an interior control triangle (unshaded on Figure 5.25) has to be calculated for the initial solution. This can be done using fairing on the corresponding molecule by (5.53).

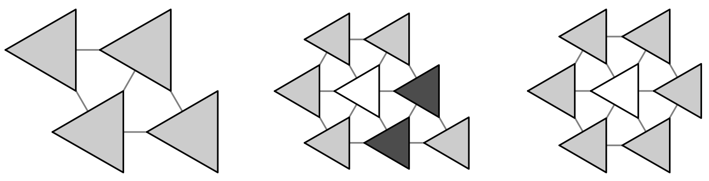


Figure 5.25: Cases with four, five and six boundary curves can be handled as well.

5.6 Further applications

The applicability of our algorithm is not restricted to the polygonal hole problem. In fact, many other situations can be thought of where a number of curves is given, as well as unit tangent vectors and unit normal vectors along these curves, and a surface patch needs to be found bounded by these curves. To illustrate this, we applied our algorithm to the blending problem for two intersecting cylinders, and for placing a cap on top of a prism.

5.6.1 Blending two cylinders

Our first example is illustrated on Figure 5.26. The two intersecting cylinders are shown above, the middle figure shows the three–sided blending patches using a different color, and the bottom figure shows the blended result as a whole. Figure 5.27 shows how a triangular blending patch is obtained. Consider two cylinders C_1 (about the y–axis) and C_2 (about the x–axis). The two cylinders intersect each other in a point $\boldsymbol{t}$ on the z–axis (which is orthogonal to the plane of the figure), and in a point $\boldsymbol{b}$ in the $z = 0$ plane. Consider two planes π_1 and π_2 through the z–axis, symmetric with respect to the bisector of the first quadrant, at an angle θ. The plane π_1 intersects C_1 along an ellipse E_1 which has a point $\boldsymbol{c}_1$ in the plane $z = 0$. Similarly, the plane π_2 intersects C_2 along an ellipse E_2, which has a point $\boldsymbol{c}_2$ in the plane $z = 0$. The curves bounding the blending patch are then determined as follows.

1. $\boldsymbol{p}_{1,2}$ is the segment of E_2 going from $\boldsymbol{t}$ to $\boldsymbol{c}_2$. The normal vectors along this curve are the normal vectors to C_2.

2. $\boldsymbol{p}_{2,3}$ is a quadratic Bézier curve in the $z = 0$ plane with control points $\boldsymbol{c}_2, \boldsymbol{b}$ and $\boldsymbol{c}_1$. The normal vectors are the vectors in the $z = 0$ plane that are normal to this curve.

3. $\boldsymbol{p}_{3,1}$ is the segment of E_1 going from $\boldsymbol{c}_1$ to $\boldsymbol{t}$. The normal vectors along this curve are the normal vectors to C_1.

The cylinders are blended together by eight such patches.

5.6.2 A cap on top of a prism

In the second example, a cap is placed on top of a prism using three triangular patches, each bounded by one edge of the prism on one side, and having two by two the same bounding curve for the other two sides. These curves are quadratic Bernstein–Bézier curves. Their control polygon consists of two legs. The first leg lies in the extension piece of a ribbon of the prism, while the second leg is parallel to the base plane of the prism and has its endpoint at the junction point of the three filling patches. Figure 5.28 shows the top of the prism (left), and an illustration of three patches constituting a cap in a different colour (right). In fact, this is the initial solution. In the point where the three patches join, they have the same normal vector. Figure 5.29 shows the result using the "instant update" method for calculating the interior control points. Figure 5.30 shows the result using fairing. For both cases, the Gaussian curvature plots are included.

5.7 Concluding Remarks

In this chapter, we gave a generally applicable solution for the polygonal hole problem. The user should only provide a set of boundary curves joining at their endpoints, their unit tangent vectors, and the unit normal vectors to the surrounding surface. The algorithm then calculates a UPS–spline surface patch that fills the gap. It is directly applicable to cases with three, four, five and six boundary curves, and by using composite patches a case bounded by two curves has been demonstrated. It fills the gap in the incomplete NURPS representation of the sphere from Chapter 3.

We first derived the so–called fitting equations in Section 5.2: interpolation of the given data at the vertices of $\partial\Delta$ leaves some degrees of freedom, which can be used to fit the data in between the interpolation points. Then, we sketched the algorithm in Section 5.3. It consists of three phases: first, an initial solution that fits the given data at the patch corners is calculated. Next, the solution is refined iteratively until the given data on the boundary are fitted close enough. Finally, the interior control triangles are determined, either during a post–iteration step, or together with the solution on the boundary. This has been investigated in Section 5.4. We have given a standard method for evaluating the quality of surfaces using surface

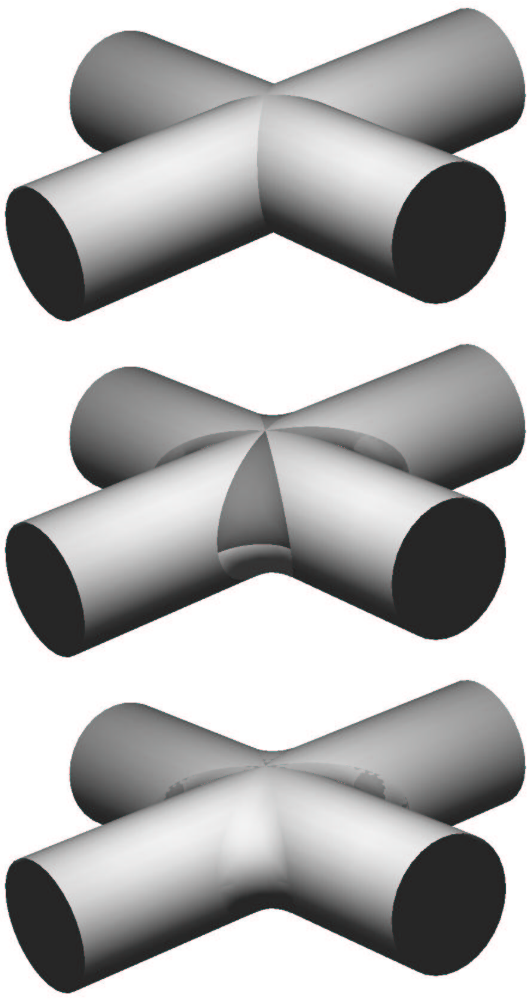


Figure 5.26: Blending two intersecting cylinders. Top: the intersecting cylinders. Middle: The blending patches are rendered in a different colour. Bottom: the blended surface as a whole.

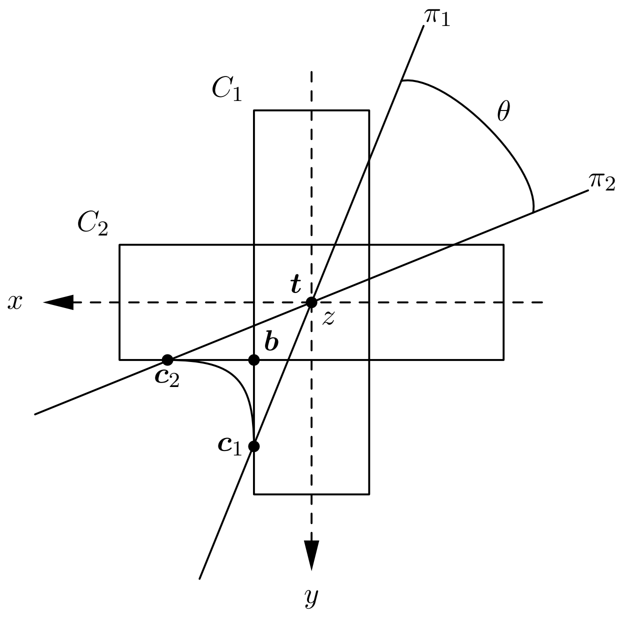


Figure 5.27: Blending two intersecting cylinders: determining the bounding curves of the patch

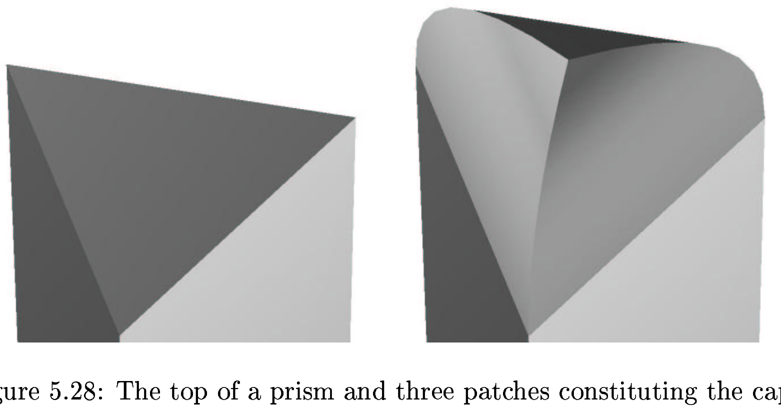


Figure 5.28: The top of a prism and three patches constituting the cap.

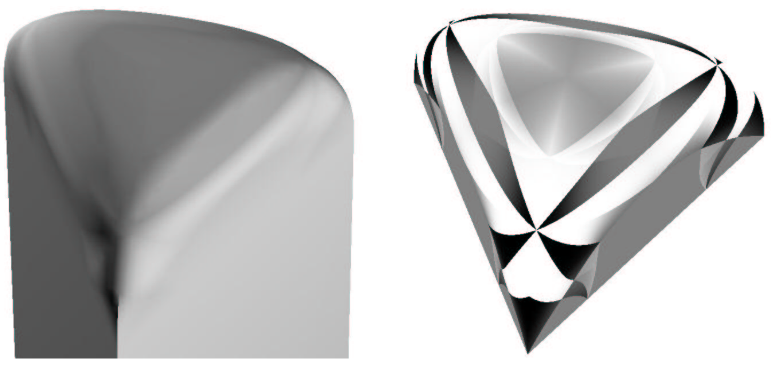


Figure 5.29: A cap on the prism using the "instant update" method. Left: rendered surface, Right: Gaussian curvature plot.

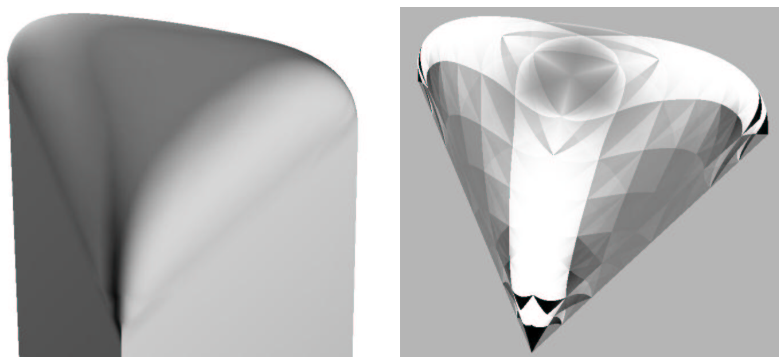


Figure 5.30: A cap on the prism using fairing. Left: rendered surface, Right: Gaussian curvature plot.

interrogation tools in Section 5.4.1. We showed how to calculate the Gaussian, mean, principal and absolute curvatures of UPS–spline surfaces, and how to map them to a color code. Then, we presented a number of cheap, intuitive methods to determine the interior control points in Section 5.4.2, using either the initial solution, the final solution on the boundary, or both. We compared them using Gaussian curvature plots. The filling patches are wavelet transformable. We investigated the application of the thresholding algorithm with boundary constraints for smoothing the interior of the patches. In Section 5.4.3 we reviewed a method for measuring the fairness of PS–spline surfaces, leading to an algorithm for fairing UPS–spline surfaces, given their boundary control triangles. This allows to calculate the interior control triangles, maximizing the fairness of the surface as a whole. It yielded the smoothest curvature plots in our experiments, with the "instant update" method as a worthwhile, cheap alternative. We explicitly gave a solution of the fairing problem for the case when Δ is a molecule M with molecule number $m = 6$. This is useful for determining an initial solution for cases with five or six boundary curves. Finally we showed in Section 5.6 that the applicability of our algorithm is not restricted to the polygonal hole problem, but it can be used as well for, e.g., blending problems, or placing a cap on top of a prism. In Chapter 6 we will use the polygonal hole algorithm to replace a degenerate NURBS patch in a realistic CAD model with a triangular UPS–spline surface.

Chapter 6

PS–surf: a CAD software prototype

6.1 Introduction

Together with our research on PS–splines in CAGD, we have written a software package that illustrates the main results. It consists of an interactive CAD application, and some underlying libraries. It supports designing with rational UPS–spline surfaces. In this chapter we motivate the choices that we have made regarding the data structure, in order to obtain efficiency in the graphical display. We also describe the global structure and the main features of our software project, and use it in a number of practical applications.

For the graphical part, we use the OpenGL graphics system, a software interface to graphics hardware that is widespread in computer graphics applications. It offers a number of geometric primitives and routines to generate realistic pictures. Sometimes we will give technical details in the form of pseudo–code, thereby referring to OpenGL–routines. However, a warning is in place because these examples are often simplified versions of the real OpenGL routines, which may have a lot more parameters that are not relevant in this discussion. Therefore, the OpenGL pseudo–code should not be copied literally to real implementations. An excellent guide for programming in OpenGL is given in [83], also called “the red book”.

In Section 6.2, we investigate object–oriented (OO–) data representations for PS–spline surfaces on arbitrary triangulations in an interactive environment. First, a decomposition into a number of classes is given in Section

6.2.1. Then we use this data structure for displaying PS–spline surfaces in OpenGL. It appears that the performance of the display is greatly influenced by the number of OpenGL function calls, and the redundant processing of vertices that are common to neighbour polygons. Vertex arrays are introduced as a tool to reduce these numbers. In Section 6.2.3, we show that the block matrix structure for UPS–spline surfaces from Section 4.6.1 has certain advantages compared with the OO–decomposition. Furthermore, its regularity can be exploited to even further reduce the number of function calls and redundant vertex specifications, as explained in Section 6.2.4. Section 6.3 discusses the structure and main functionality of PS–surf. First, the modular components are described in Section 6.3.1, and in Section 6.3.2 we illustrate how the OO–architecture yields an extendible and yet controllable piece of software, by exploiting the advantages of inheritance. Section 6.3.3 lists the main functionality that is built into PS–surf. It can be divided into two categories: a class of general features regarding the display of geometric objects, and a class of features concerning UPS–spline surfaces, which illustrate the results of our research. We also give a number of screen–shots as an illustration. Section 6.4 presents a number of realistic CAD models using UPS–spline surfaces in two practical applications. First, we integrate our polygonal hole algorithm in a reverse engineering environment, where geometric models are generated from physical prototypes. It allows to represent a CAD model using NURBS surfaces and UPS–spline surfaces all together. In our example, a driving–mirror of a car, we used the latter instead of the degenerate triangular NURBS patches that otherwise are unavoidable in systems that support NURBS representations only. Secondly, in Section 6.4.2, we give an algorithm for approximating a given NURBS surface with a UPS–spline surface, based on component–wise interpolation of function and partial derivative values. This is illustrated with a UPS–spline model of the back of a car seat.

6.2 Representing PS–spline surfaces in an interactive environment

In this section we investigate data representations for PS–spline surfaces, and their role in the graphical display using OpenGL.

6.2.1 An object–oriented decomposition

An object oriented design for representing PS–spline surfaces can be obtained by decomposing them into their main components, and isolating these into separate classes. Let us start with representing the domain triangulation Δ. Figure 6.1 shows a class diagram. A triangulation is represented as an array of triangles and an array of vertices. Each instance of class `Triangle` contains three integer values v_1, v_2, v_3, which are indices of entries in the vertex array. Another possibility is to store pointers to the vertices instead of their indices, but for our purposes the indices are better suited (see Section 6.2.2). Furthermore, an instance of class `Vertex` has a boolean property which tells whether it is a boundary vertex or not. This information structure has to be extended with information about the

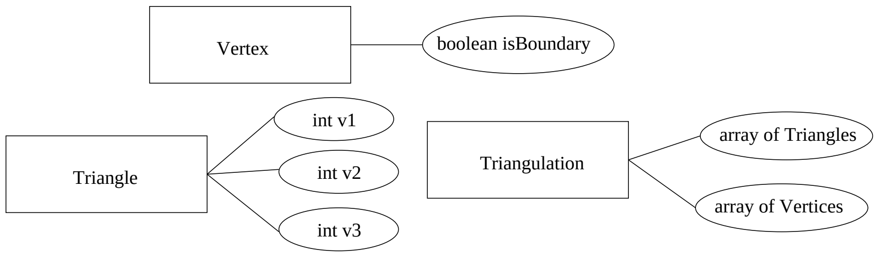


Figure 6.1: An object–oriented data structure for the triangulation Δ: initial configuration.

topology of the triangulation, i.e., for each vertex V_i, we have to be able to find the triangles $\rho_l \in M_i, l = 1, \ldots, m_i$, and for each triangle $\rho_{i,j,k}$, we have to be able to find its neighbour triangles sharing the edges V_iV_j, V_jV_k and V_kV_i with it. This can be done by introducing a class `Molecule`, and by extending the classes `Triangle` and `Vertex` as on Figure 6.2. A molecule is characterized by an array of pointers to triangles. A triangle has three pointers to its neighbours $nb1$, $nb2$ and $nb3$, and each vertex contains an object of class `Molecule`. The data structure for Δ can be completed by

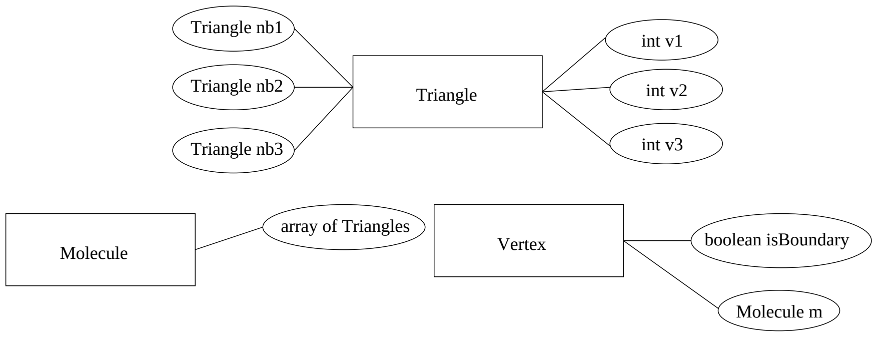


Figure 6.2: An object–oriented data structure for the triangulation Δ: information about the topology has been added.

adding a representation for the PS–triangles. This is illustrated in Figure 6.3: the class `PS-Triangle` has three members of class `2DPoint`, and each vertex contains an instance of class `PS-Triangle`.

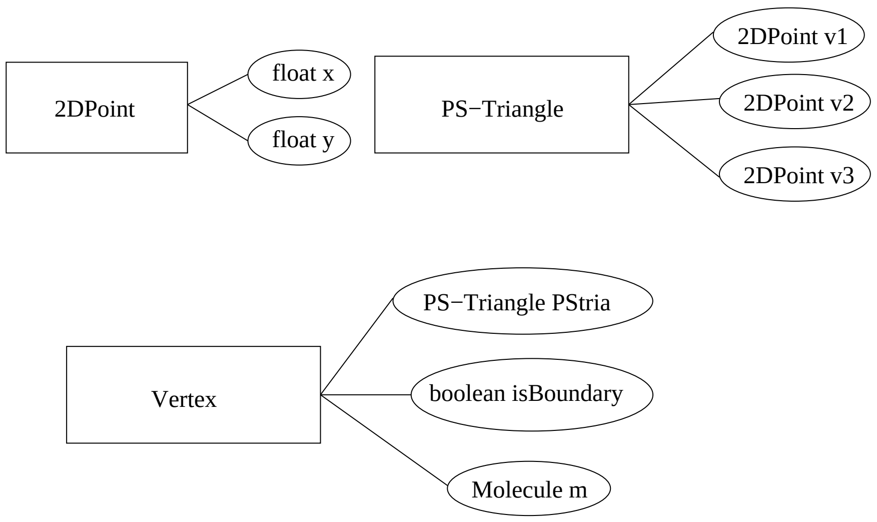


Figure 6.3: An object–oriented data structure for the triangulation Δ: information about the PS–triangles has been added.

This representation of Δ is now extended with the information about the control net (See Figure 6.4), in order to obtain a complete representation of a PS–spline surface. Therefore, two classes are added: first there is class `Point` for representing 3D points in the Euclidean space, and second there is a class `Control Triangle` that has three `Point` objects among its members. Finally, an instance of class `Control Triangle` is added to each vertex. Note that NURPS surfaces can be represented by extending the definition of class `Point` with a weight.

6.2.2 Displaying PS–spline surfaces in OpenGL

We are particularly interested in the graphical display of PS–spline surfaces using OpenGL. Suppose that a PS–spline surface is displayed by drawing the triangles that connect the tangent points of the control triangles (usually after a few subdivision steps). This can be achieved in OpenGL, using the above data structure, as follows:

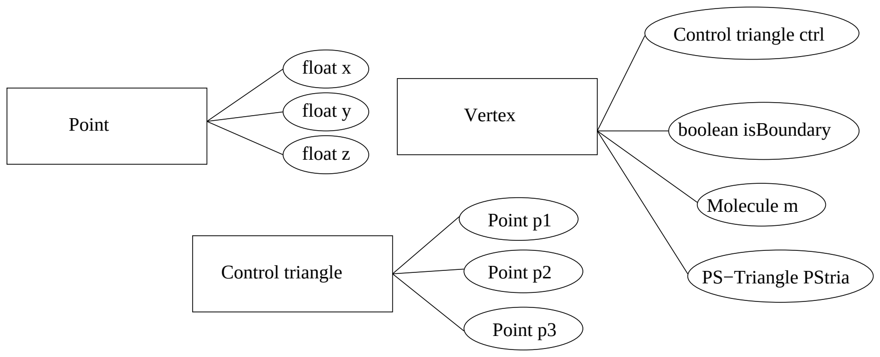


Figure 6.4: An object–oriented data structure for representing PS–spline surfaces is obtained by adding information about the control triangles.

```
glBegin(GL_TRIANGLES);
   For each Triangle in Triangulation
      For each Vertex in Triangle
         Calculate the tangent point p
          of Vertex.ctrl;
         glVertex(p);
      Next Vertex
   Next Triangle
glEnd();
```

Drawing a set of triangles in OpenGL is initiated by calling the routine `glBegin()` with parameter `GL_TRIANGLES` as the drawing primitive, and is ended by calling `glEnd()`. In between, the vertices of the triangles have to be specified one after another using `glVertex()`. Each three consecutive calls determine a triangle. It is clear that the latter routine is called $3t$ times, where t denotes the number of triangles $\rho \in \Delta$, and that each vertex V_i has to be specified m_i times. This means that rather many function calls have to be performed, and that there is a redundant processing of vertices that are shared between adjacent polygons. Therefore, this approach may not be preferable in an environment where surfaces have to be manipulated interactively, and efficiency in the graphical display is crucial. OpenGL offers a number of routines that help to reduce the number of function calls as well as the redundant processing of vertices. These make use of so–called vertex arrays. The idea is to put the data that have to be drawn in an array, and pass that array to OpenGL using a single routine. Then, instead of specifying coordinates of vertices between `glBegin()` and `glEnd()`, the array–indices of the vertices are passed. Here we take advantage of the fact that our class Triangle stores the array indices of the vertices rather than pointers. First a vertex array is constructed:

```
p = new float[3*n];
int i = 0;
For each Vertex in array of Vertices
   Calculate the tangent point (x,y,z)
    of Vertex.ctrl;
   p[i++] = x; p[i++] = y; p[i++] = z;
Next Vertex
```

Then, when the surface is to be displayed, the vertex array is passed to OpenGL and is dereferenced:

```
glVertexPointer(p);
For each Triangle in Triangulation
   int indices[] = {Triangle.v1, Triangle.v2, Triangle.v3};
   glDrawElements(GL_TRIANGLES, indices);
Next Triangle
```

Note that as long as the surface isn't modified, the same vertex array may be used for displaying it from different viewpoints. Moreover, geometric transformations can be added to the perspective projection matrix, allowing for example that the surface is rotated interactively, without recalculating the vertex array.

6.2.3 The block matrix representation for UPS–spline surfaces

While the data representation of Section 6.2.1 is general, and allows to display surfaces in OpenGL using vertex arrays easily, it may not be the most efficient option when dealing with UPS–spline surfaces. Recall from Section 4.6.1 that UPS–spline surfaces can be represented using a block matrix structure by extending the polygon Ω to a parallelogram, thereby introducing a number of phantom vertices, then recasting it into a rectangle, and putting a 3×3 matrix at the position of each vertex that carries the coefficients of the control points, or zero elements for the pseudo–vertices. This, together with two arrays of integers which indicate the first and last position of the real vertices for each row, suffices to represent UPS–spline surfaces. Since the domain triangulation and the PS–triangles are known beforehand, these do not have to be stored explicitly. However, a picture of Δ may be useful for interactively selecting control triangles by mouse–clicking the corresponding PS–triangle. In that case, a similar block matrix structure may be used for representing Δ and the PS–triangles.

Clearly, this is not an object–oriented representation, but it has a number of advantages compared with the general data structure from Section 6.2.1, mainly because topologic information is inherent to the matrix structure.

1. For a given block entry, it is immediately clear whether the corresponding vertex is on the boundary or not, and which vertices are adjacent to it.
2. No information about the domain triangles has to be stored explicitly. The triangles can be read of immediately from the matrix structure. For a given triangle, its neighbours are directly recognized.
3. Given a vertex, the triangles of its molecule can be read of directly from the structure.

Furthermore, this block matrix structure is suitable for hardware implementations. We will show in the next section that its regularity allows to take even more advantage of vertex arrays in OpenGL.

6.2.4 Displaying UPS–spline surfaces using OpenGL

Recall from Section 6.2.2 that vertex arrays can be used in OpenGL to reduce the number of function calls and the redundancy in processing vertices that are common to adjacent polygons. However, from the drawing algorithm it can be seen that for displaying t triangles, still $3t$ vertex indices have to be passed to OpenGL, and the index of each vertex V_i is again passed m_i times, where m_i equals six for the interior vertices of Δ. By exploiting the regularity of the uniform triangulation, these numbers can be reduced further. The idea is to avoid drawing one triangle at a time, but rather use a strip of triangles (this concept is illustrated on Figure 6.5) as the drawing primitive. These are chosen in a way that permits them to be read of easily from the block matrix structure. In order to illustrate

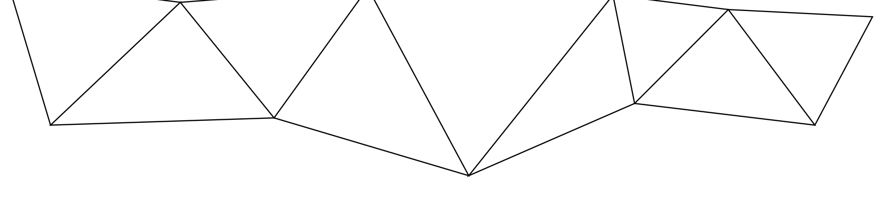


Figure 6.5: A strip of triangles.

this, let us take a simple example in Figure 6.6 that has only two triangle strips. The figure shows a triangulation having eight vertices, together with its extended grid. Consider the two strips of triangles formed by the sets of vertices

```
 int strip1[] = {4, 1, 5, 2, 6, 3};
 int strip2[] = {4, 7, 5, 8, 6};
```

and suppose that the vertex array, containing the corresponding surface points has been calculated as in Section 6.2.2. Then the surface can be drawn using

```
glVertexPointer(p);
glDrawElements(GL_TRIANGLE_STRIP, strip1);
glDrawElements(GL_TRIANGLE_STRIP, strip2);
```

Now each vertex of the top and bottom rows of the block matrix structure is passed only once to OpenGL, and the other ones are passed exactly two times. This reduces the number of function calls and the redundant processing of vertices. Also, as long as the triangulation Δ, and the number of subdivision steps used for generating points on the surface remain the same, the strip definition does not need to be modified. If the surface is modified by changing one control point, only a small number of entries in the vertex array have to be updated. The same idea that has been illustrated here for two strips of triangles, has been implemented in general in our software prototype PS–surf for displaying UPS–spline surfaces. Its efficiency allows to manipulate them interactively even on a low–end personal computer.

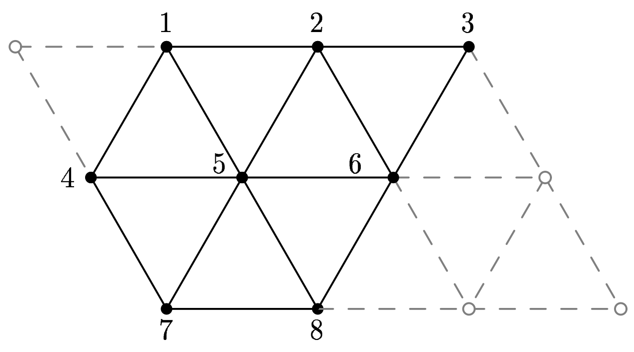


Figure 6.6: A simple triangulation having two triangle strips: the first one is defined by the vertices $\{4, 1, 5, 2, 6, 3\}$ and the second one by $\{4, 7, 5, 8, 6\}$.

6.3 PS–surf

In this section we discuss the architecture of our software prototype. We shall discuss the global modular design, how the advantages of the OO–approach were exploited, and we shall briefly summarize its main functionality.

6.3.1 The modular approach

Although we refer in general to PS–surf as the application that illustrates the use of UPS–spline surfaces, it is only one part of the set of modules that have been developed. In this set, PS–surf fulfills the role of user application, by implementing a framework that allows the user to organize objects into a hierarchical tree structure, to view them in 3D and to manipulate them through custom dialogues, menus and buttons. It is written for the Microsoft Windows operating system and heavily relies on the common user interface controls available in the Microsoft Foundation Classes (MFC). The real work, however, is done in a number of portable libraries underlying PS–surf. These are illustrated in the dependence diagram of Figure 6.7, and are briefly discussed below.

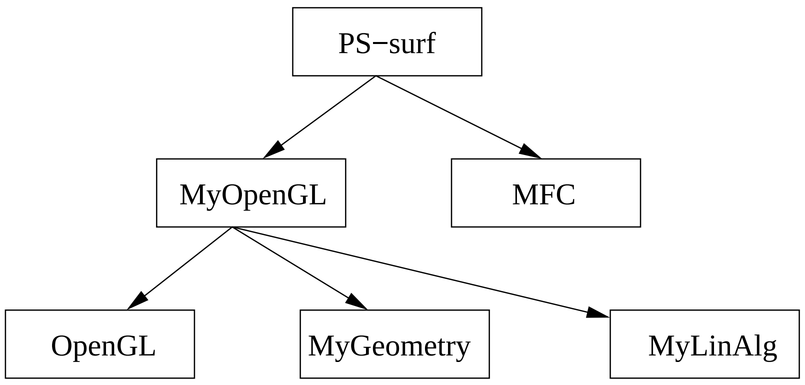


Figure 6.7: The dependence diagram of the modular PS–surf components.

MyOpenGL. This library offers an object–oriented shell on top of the OpenGL library for creating realistic images of 3D rendered objects. It has classes for

- geometric objects, for example, a cube. These can be grouped into composite entities called scenes, which can be nested, but otherwise act just as normal geometric objects.

- views on objects, which may be obtained by perspective or orthogonal projections.
- viewpoints which tell where the viewer is located and in which direction he looks.
- materials. These encapsulate properties such as the colour, shininess, transparency,...
- light sources. They can be added to scenes.

It contains a number of commonly used geometric objects from the glut library [43] such as a cone, a sphere and a teapot, as well as classes that represent rational UPS–spline surfaces and NURBS surfaces. It also implements an interface that allows to read NURBS surfaces from IGES files [75], which is a common file format for exchanging CAD models.

MyGeometry. This library offers a number of auxiliary routines, such as for the calculation of a dot product and a vector product of two vectors. Furthermore, a number of `FITPACK` [17] routines for evaluating and differentiating tensor product PS–splines has been imported into this library (see Section 6.4.2).

MyLinAlg. This library offers a class for working with overdetermined systems, and finding a solution in the least squares sense using Givens orthogonalization.

6.3.2 The object–oriented approach

In Section 6.2, we showed that in order to obtain a data structure for UPS–spline surfaces that easily led to an efficient visualization in OpenGL, the object–oriented design philosophy wasn't pushed too far. However, PS–surf exploits the advantages of object–oriented design in other areas, where this is useful. In this section we consider four examples to illustrate this.

All the classes that represent geometric objects inherit from a common parent class, `CGLObject` (see Figure 6.8). For example, `CGLNURBS` (representing NURBS surfaces) and `CGLScene` (which is a placeholder for an unlimited number of geometric objects). The class `CGLObject` provides functionality for calculating an appropriate viewpoint, for toggling its visibility, and for applying basic geometric transformations such as translations and rotations, using OpenGL. Also, a number of materials that can optionally be specified by the inheriting class upon construction, are assigned to a `CGLObject`–derived model. For example, the class `CGLUPSSpline` (that represents a

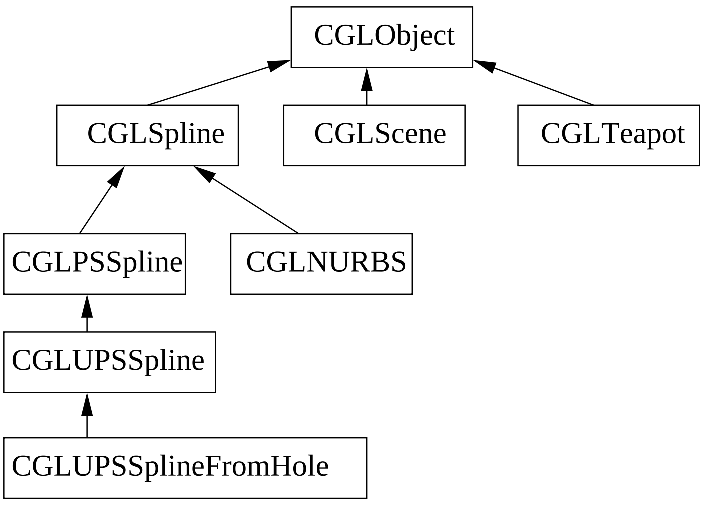


Figure 6.8: Inheritance diagram for `CGLObject`-derived classes.

rational UPS–spline surface) uses two different materials: one for the surface and one for the control triangles. Furthermore, all `CGLObject`–derived classes can automatically be organized in a hierarchical tree structure, they can be rendered and rotated interactively in a 3D view, and can be manipulated through a dialogue–based user interface for, e.g., changing material properties. `CGLObject` is an abstract class, and as such specifies a signature for implementing the rendering routines in derived classes. All that a deriving class has to implement in order to be accessible through the PS–surf user interface are routines for calculating a bounding box, rendering a wireframe and rendering the object as a solid model. Optionally, the class can provide a routine for generating a curvature plot of the underlying surface. This first example shows how our object–oriented design takes advantage of inheritance from a powerful base class.

Secondly, the design can be be extended in the future towards PS–spline surfaces on arbitrary triangulations. These can be integrated smoothly into the existing user interface. Therefore, an abstract class `CGLPSSpline` has been defined that is a common ancestor of `CGLUPSSpline` and a future class for representing general PS–spline surfaces (see Figure 6.8). This abstract class defines the signature for a number of routines that, once implemented, automatically integrate general PS–spline surfaces in the user interface of PS–surf. For example, all `CGLPSSpline`–derived classes can be moved to a 2D domain view pane using drag-and-drop. In this window, PS–triangles can be double–clicked and a dialog appears that allows to manipulate the corresponding control triangle and weights. Furthermore, all `CGLPSSpline`–derived classes that implement a number of routines related to the wavelet transform, automatically make use of a dialogue that shows the spectrum of the wavelet coefficients and allows to perform the thresholding algorithm. Finally, a signature has been provided for routines that allow to write a picture of the domain triangulation and its PS–triangles to disk, in picTEX [1] or TEXdraw [41] format.

Our third example shows that PS–surf can be extended towards new CAD applications, taking advantage of the existing features as much as possible. The integration of the polygonal hole algorithm is such an example. The class `CGLUPSSplineFromHole` allows to calculate a UPS–spline surface filling in a polygonal hole, given a set of boundary curves, their unit tangent vectors and unit normal vectors to the surrounding surfaces, as in Chapter 5. This class inherits directly from `CGLUPSSpline`, as shown on Figure 6.8, so it enjoys all the properties we already mentioned.

Finally we note that the classes `CGLNURBS` and `CGLPSSpline` have a common ancestor, `CGLSpline` (See Figure 6.8). It implements part of the functionality that is common to all spline based surface representations, such as the possibility to show or hide the control net and the option to specify the number of subdivision steps that is to be performed for the graphical display. This allows to access these features for both NURBS surfaces and UPS–spline surfaces, which are very different mathematical representations, through a common interface.

6.3.3 The main functionality

General features

1. The application window has three main components: a tree view for organizing the geometric models hierarchically, a 3D view, and a pane in which the domain picture of UPS–spline surfaces is displayed, that is, the triangulation together with the PS–triangles (See Figure 6.9).

2. Geometric objects can be added to an initially empty scene. Objects can be grouped into sub–scenes, which act as composite objects.

3. Geometric transformations such as rotations, translations and scaling operations can be applied to any geometric object by the transformation dialogue shown in Figure 6.10.

4. Material properties can be assigned to simple and composite geometric objects. The material properties dialogue (Figure 6.11) allows to specify the ambient, diffuse and specular colour components, the colour of emitted light, a degree of transparency, and the specular exponent. Interesting materials can be stored in a library in the tree view, using drag-and-drop. They can also be dragged from the library and dropped onto an object in order to change its material.

5. Each geometric object in the tree structure can be displayed from its own viewpoint. Initially, an appropriate viewpoint is calculated automatically by dragging the object to the 3D view pane, but the viewer can move and turn his head within the scene in any direction. Interesting viewpoints can be stored in a library in the tree view using drag-and-drop. They also can be dragged from the library and dropped onto an object in order to change its viewpoint.

6. Light sources can be added to a scene. The properties of a lightsource can be altered using the lightsource dialogue. The main colour characteristics are the ambient, diffuse and specular colour components. A lightsource can be directional or positional, it may act as a spotlight or an omni–directional lightsource, and its intensity can be attenuated linearly and quadraticly. A sphere may be drawn in the scene at the location of the lightsource.

7. Each objects visibility in the 3D view pane can be toggled. An object can be rendered as a wireframe or as a lit surface. In the latter case, the user can choose between Gouraud shading or flat shading. The user can also render a curvature plot on the surfaces, using Gaussian, mean, principal or absolute curvatures, and he can define his own curvature to colour mapping.

8. Objects can be moved interactively in the 3D view pane, by mouse movements.

9. NURBS surfaces can be added to a scene by importing them from an IGES file.

10. The picture in the 3D view can be exported to a bitmap or to the clipboard. The root scene with all its objects and the material and viewpoint libraries can be permanently stored to and retrieved from a binary file.

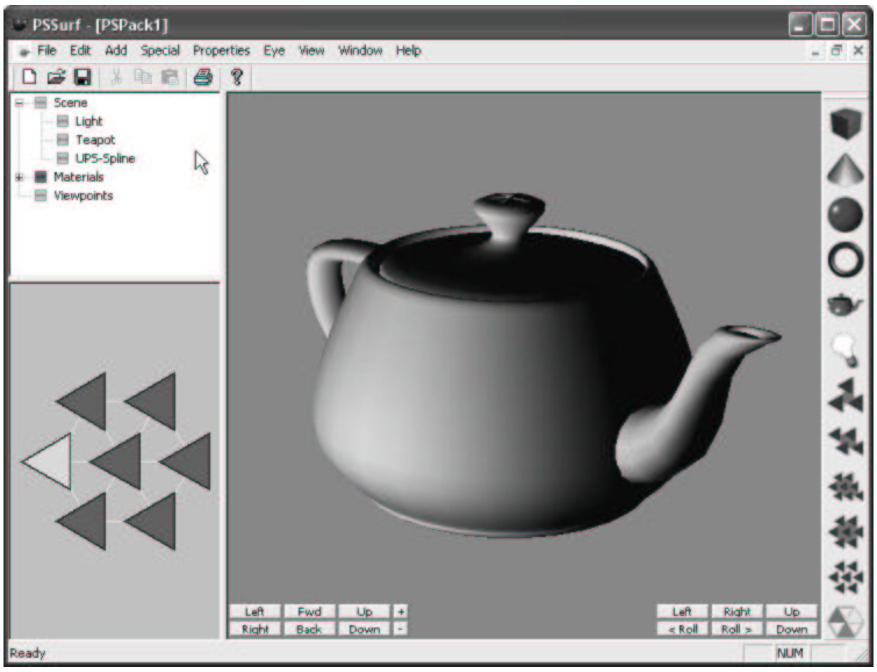


Figure 6.9: The user interface consists of three main components: a 3D view pane, a tree view, and a 2D domain view pane.

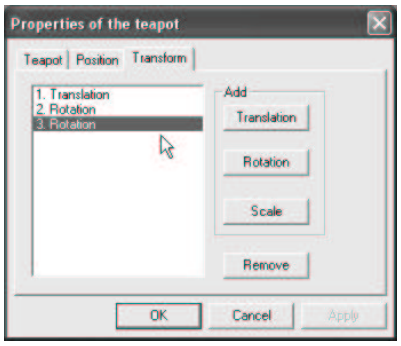
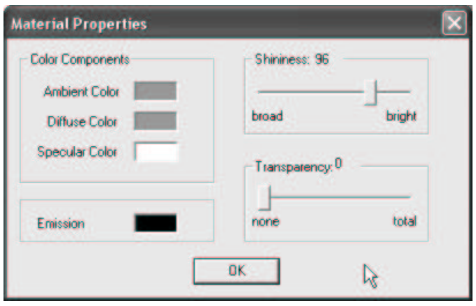


Figure 6.10: Geometric transformations can be applied to the objects.


Material Properties

Shininess: 96

Transparency: 0

Figure 6.11: Material properties are set using the material dialogue.

Features concerning UPS–spline surfaces

1. UPS–spline surfaces can be added to a scene. The domain polygon can be user–specified, or can be selected from a list containing the most common cases of three, four, five and six edges.

2. The domain picture of a UPS–spline surface can be obtained by dragging the surface from the tree view and dropping it onto the domain view pane. It allows to select PS–triangles by mouse–clicking them. A double click opens a dialogue in which the control points and weights of the corresponding control triangle can be altered, see Figure 6.12.

3. Constrained fairing of UPS–spline surfaces as explained in Section 5.4.3 is provided.

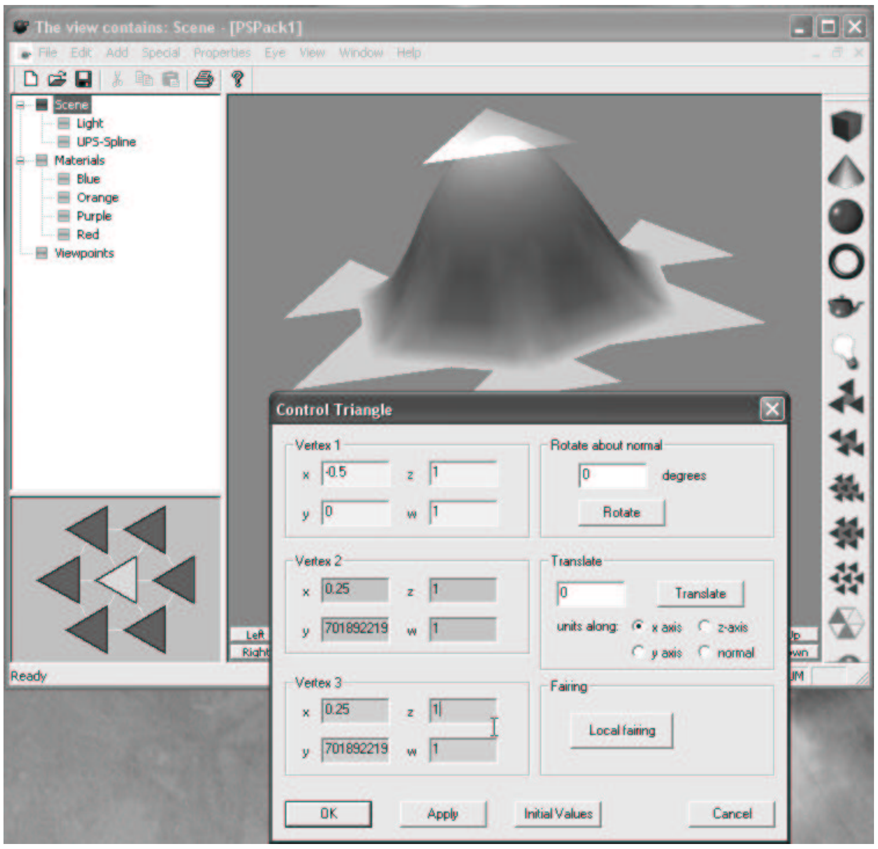
4. Each UPS–spline surface can be subdivided in order to obtain better local control, as explained in Section 4.4.

5. The UPS–spline wavelet transform (Section 4.5.4) can be applied to UPS–spline surfaces that have been subdivided at least once. PS–surf plots the wavelet coefficient spectrum for each surface component, and allows to apply the thresholding algorithm from Section 4.5.6 interactively, step by step, while the user can supply values for the thresholds of each component (see Figure 6.13).

6. UPS–spline surfaces can be obtained as the output of the polygonal hole algorithm from Chapter 5. The user can determine an appropriate solution by controlling the number of iteration steps interactively (See Figure 6.14). During each step he can choose a method for determining the interior control triangles as explained in Section 5.4.

7. A UPS–spline surface can be obtained as well by fitting the boundary curves and normal vectors of a NURBS surface, again relying on the polygonal hole algorithm.

8. A UPS–spline surface can be obtained as the approximation of a given NURBS surface (see Section 6.4.2).



Vertex 1: x -0.5, z 1, y 0, w 1

Vertex 2: x 0.25, z 1, y 701892219, w 1

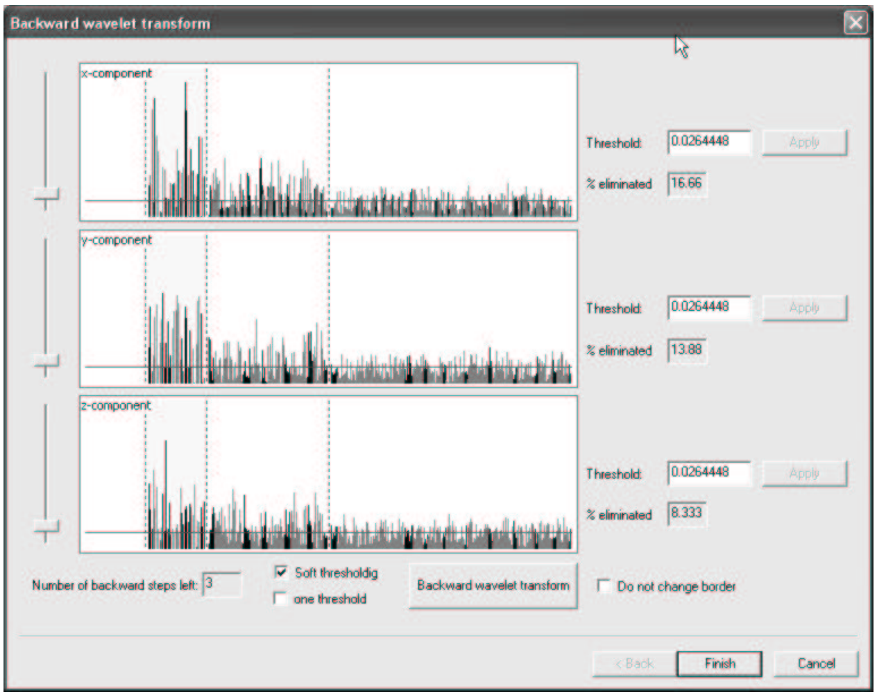
Vertex 3: x 0.25, z 1, y 701892219, w 1

Rotate about normal: 0 degrees

Translate: 0, units along: [✓ x axis] z-axis y axis normal

Fairing

Figure 6.12: The control points corresponding to the selected PS–triangle can be modified. The results are immediately shown in the 3D view pane. Each control point has a color code reflected in the edit boxes as well as on the selected control triangle in the 3D view pane.



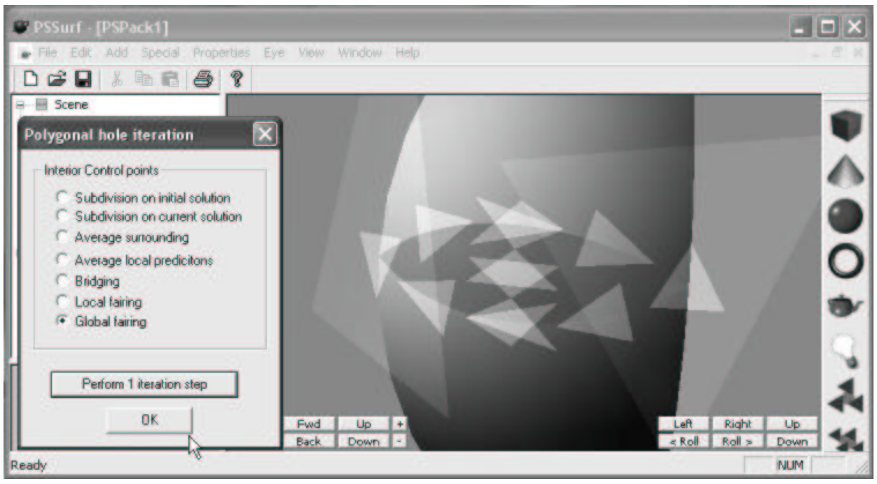
Threshold: 0.0264448
% eliminated 16.66

Threshold: 0.0264448
% eliminated 13.88

Threshold: 0.0264448
% eliminated 8.333

Number of backward steps left: 3
[✓] Soft thresholdig
[ ] one threshold
[ ] Do not change border

Figure 6.13: The wavelet transform dialogue shows the spectrum of the wavelet coefficients for each component. The backward wavelet transform can be applied in conjunction with thresholding.



Polygonal hole iteration

Interior Control points

- Subdivision on initial solution
- Subdivision on current solution
- Average surrounding
- Average local predicitons
- Bridging
- Local fairing
- [✓ Global fairing]

Figure 6.14: The polygonal hole iteration dialog allows to perform iteration steps interactively and to specify the method for calculating the interior control triangles. The results are immediately shown in the 3D view pane.

6.4 Applications in CAD modeling

6.4.1 Reverse Engineering

The use of physical models is an integral part of each design cycle during product development. However, a CAD–description of such models is needed before the production phase can begin. The translation of a physical model into a geometric model is called reverse engineering, and involves, for example, the generation of a mathematical surface model starting from a point cloud or a set of curves. In [42], an integrated approach to the modeling of free–form surfaces in reverse engineering is presented. It is mainly based on the NURBS representation for surfaces, because of its wide support in available CAD packages. In order to cope with surface patches that do not have a rectangular geometry, the author developed methods for calculating degenerate NURBS surfaces starting from curves or point clouds. By letting two vertices of the rectangular domain coincide, a triangular surface patch can be represented. However, this approach has a number of disadvantages. The surface normal at the degeneration point is undefined, and this may be a problem for cutting tools in CAM systems. Furthermore, the control point density in the neighbourhood of the degeneration point is relatively much higher compared with the rest of the surface, causing local surface wrinkles. In this section we recall an example from the cited work, where a CAD model of the driving–mirror of a car is obtained by reverse engineering techniques. However, we replaced the three degenerate NURBS surfaces in the model with triangular UPS–spline surfaces obtained via the polygonal hole algorithm from Chapter 5, starting from the given boundary curves and surface normals of the degenerate patch. Recall that our algorithm does not need normal vectors at the corners of the patch, so the fact that the given patch is degenerate at one corner is not a problem. Figure 6.15 shows one of these triangular UPS–spline surface patches in a different colour than the surrounding NURBS surfaces, together with its control triangles. Figure 6.16 compares the wireframes of the degenerate NURBS patch (which is a refined control net) and the UPS–spline patch. It illustrates that the control point density in the neighbourhood of the degeneration point is higher than elsewhere. The wireframe of the UPS–spline surface is much more regular. Figure 6.17 shows the resulting CAD model of the whole mirror as a rendered surface. The interior control triangles were obtained by the "instant update" method, which performed better than fairing in this case. The latter method yielded an interior which was too flat. These examples show that it is possible to let NURBS surfaces and UPS–spline surfaces live together in a CAD environment, each having their own advantages.

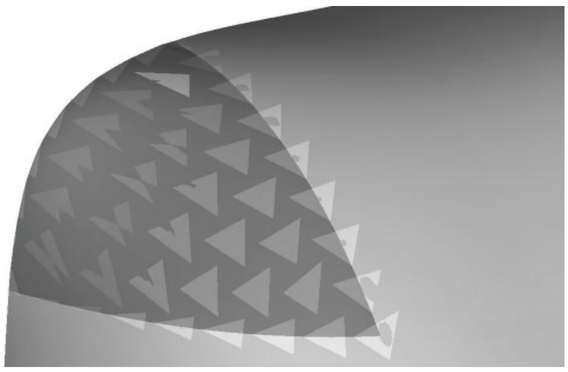


Figure 6.15: Close up of one of the triangular UPS–spline surface patches in a different colour than the surrounding NURBS surfaces.

6.4.2 Approximation of a NURBS surface with a UPS–spline surface

As a second application, we consider the approximation of a given NURBS surface

$$\boldsymbol{S}(\bar{u},\bar{v}), (\bar{u},\bar{v}) \in R = [\bar{u}_0, \bar{u}_0 + W] \times [\bar{v}_0, \bar{v}_0 + H], \qquad (6.1)$$

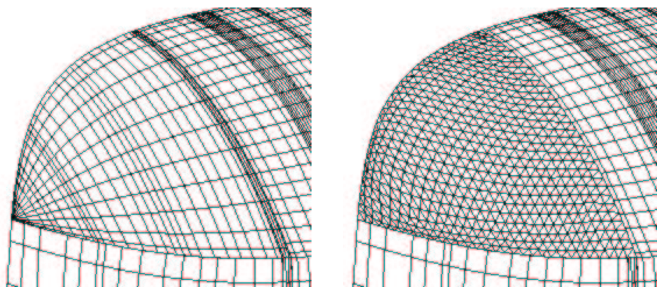


Figure 6.16: A comparison of the wireframe models using degenerate NURBS patches (left) and triangular UPS–spline surfaces (right).

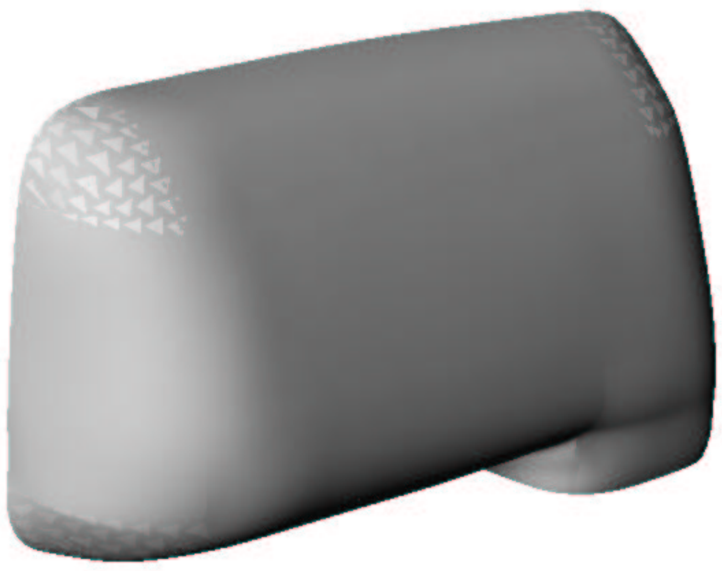


Figure 6.17: The CAD model of a driving mirror, using combined NURBS and UPS–spline surface patches.

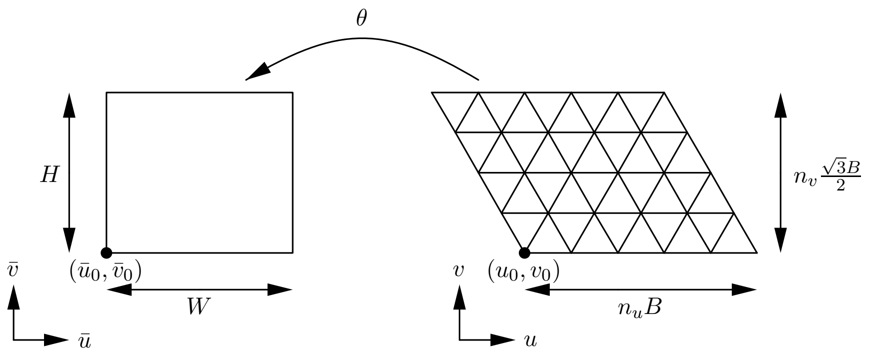


Figure 6.18: Left: The domain R of a NURBS surface $\boldsymbol{S}(\bar{u}, \bar{v})$. Right: The domain Ω of a UPS–spline surface $\boldsymbol{s}(u, v)$ and its triangulation Δ.

by a UPS–spline surface $\boldsymbol{s}(u, v)$ on a four–sided domain Ω. This can be done by component–wise interpolation of the function values and partial derivatives of $\boldsymbol{S}(\bar{u}, \bar{v})$. The domain R is displayed on Figure 6.18 (left) and Ω is depicted on the right. Let $(n_v + 1)$ and $(n_u + 1)$ denote the number of rows, resp. columns of the triangulation Δ of Ω, and recall that each triangle of Δ has edge length B. Consider the map

$$\theta : \Omega \to R : \begin{cases} \bar{u} &= \bar{u}_0 + \frac{W}{n_u B}(u - u_0) + \frac{\sqrt{3}}{3}\frac{W}{n_u B}(v - v_0) \\ \bar{v} &= \bar{v}_0 + \frac{2\sqrt{3}}{3}\frac{H}{n_v B}(v - v_0), \end{cases} \tag{6.2}$$

then $\boldsymbol{s}(u, v)$ is completely determined by imposing

$$\boldsymbol{s}(V_i) = \boldsymbol{S}(\theta(V_i)) \tag{6.3}$$

$$\frac{\partial \boldsymbol{s}(V_i)}{\partial u} = \frac{\partial \boldsymbol{S}(\theta(V_i))}{\partial u} \tag{6.4}$$

$$\frac{\partial \boldsymbol{s}(V_i)}{\partial v} = \frac{\partial \boldsymbol{S}(\theta(V_i))}{\partial v}, \tag{6.5}$$

for all $V_i \in \Delta$. Component–wise application of (4.9), together with the chain rule applied to (6.2) yields

$$\boldsymbol{c}_{i,1} = \boldsymbol{S}(\theta(V_i)) - \frac{W}{2n_u}\frac{\partial \boldsymbol{S}(\theta(V_i))}{\partial \bar{u}} \tag{6.6}$$

$$\boldsymbol{c}_{i,2} = \boldsymbol{S}(\theta(V_i)) - \frac{H}{2n_v}\frac{\partial \boldsymbol{S}(\theta(V_i))}{\partial \bar{v}} \tag{6.7}$$

$$\boldsymbol{c}_{i,3} = \boldsymbol{S}(\theta(V_i)) + \frac{W}{2n_u}\frac{\partial \boldsymbol{S}(\theta(V_i))}{\partial \bar{u}} + \frac{H}{2n_v}\frac{\partial \boldsymbol{S}(\theta(V_i))}{\partial \bar{v}}. \tag{6.8}$$

We have implemented this method in PS–surf. For the evaluation and differentiation of the given NURBS surfaces, we relied on `FITPACK` [17]. The appropriate routines of this curve and surface fitting package written in `FORTRAN 77`, have been translated to `C` and imported in our `MyGeometry` library, and could be applied directly since the weights of the given NURBS model were not used. In order to determine an appropriate value of n_u and n_v, we tried to use about the same number of control points for both the NURBS and the UPS–spline models. Suppose that $k_{\bar{u}}$ and $k_{\bar{v}}$ denote the number of control points of the NURBS model in the $\bar{u}-$, resp. $\bar{v}-$direction. Then it is plausible to determine n_u and n_v as

$$n_u \approx \frac{\sqrt{3}}{3} k_{\bar{u}} - 1, \quad n_v \approx \frac{\sqrt{3}}{3} k_{\bar{v}} - 1, \tag{6.9}$$

such that the number of control points in the UPS–spline model is equal to $3(n_u + 1)(n_v + 1) \approx k_{\bar{u}} k_{\bar{v}}$. We approximated a given NURBS model of the back of a car seat by UPS–spline surfaces. Figure 6.19 shows the head restraint with its control triangles. Figure 6.20 shows the UPS–spline model of the back of the seat as a whole.

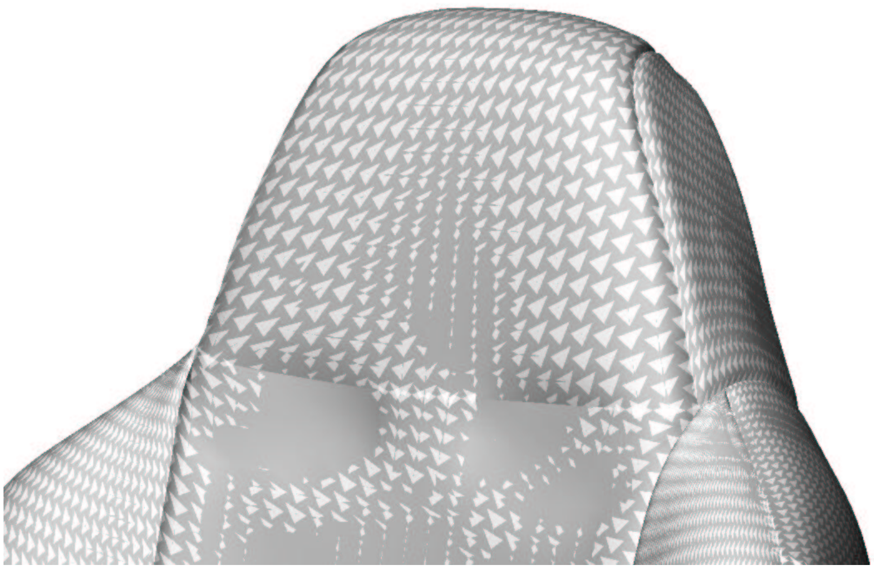


Figure 6.19: Close–up of the head restraint of the car seat, together with the control triangles.

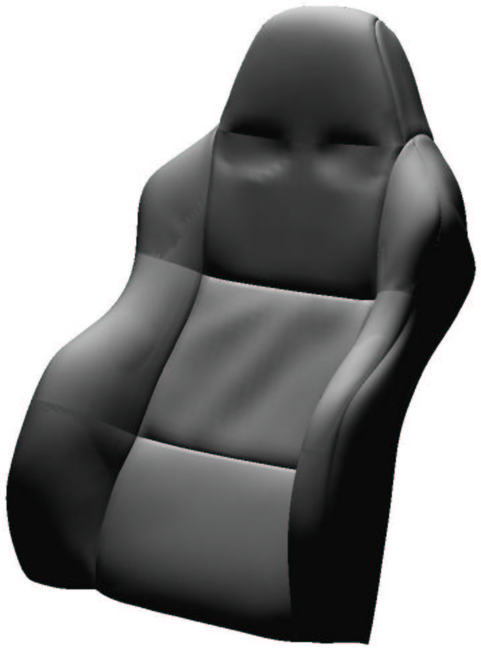


Figure 6.20: CAD model of the back of a car set, using UPS–spline surfaces.

6.5 Concluding Remarks

In this chapter, we presented our software prototype PS–surf, that has been developed for demonstrating our research results. It is a computer program for designing with rational UPS–spline surfaces. We gave a generally applicable object–oriented design for representing PS–spline surfaces in Section 6.2.1 and discussed the graphical display with OpenGL in Section 6.2.2. The performance could be improved by the use of vertex arrays. These allow to reduce the number of function calls and the redundant processing of vertices. We showed in Section 6.2.3 that the block matrix structure for UPS–spline surfaces, that was given in Section 4.6.1, has a number of advantages compared with a generally applicable object–oriented approach because it inherently contains information about the topology of the triangulation. Moreover, its regularity has been exploited in order to even further reduce the number of OpenGL function calls and the redundant processing of vertices in Section 6.2.4, without calculating additional information. We also gave a general description of the modular architecture of our software project in Section 6.3.1. A number of portable libraries have been implemented, with on top an interactive application for the Microsoft Windows operating system. Section 6.3.2 described a number of classes from the main library, `MyOpenGL`, into more detail. It illustrates how the advantages of object–oriented programming have been successfully exploited. Inheritance from a powerful base class `CGLObject`, allows to introduce new geometric objects with a minimal effort. On the other hand, inheritance from a more specialized class allows to extend PS–surf with new CAD applications that use complex geometric entities such as UPS–spline surfaces. In both cases, a great deal of code is reused. We also demonstrated the use of abstract classes, for example to prepare PS–surf for an extension towards PS–spline surfaces on arbitrary triangulations in the future, maybe using the representation from Section 6.2. This future class will be smoothly integrated in the existing user interface by inheritance from `CGLPSSpline`. Section 6.3.3 gives an overview of the possibilities that PS–surf has to offer together with some screen–shots. There are a number of general features for modeling with geometric entities, but the main part consists of the functionality that illustrates the use of UPS–spline surfaces. In Section 6.4, we have used our software prototype in realistic CAD modeling examples. First, we integrated our polygonal hole algorithm into a reverse engineering context, such that NURBS surfaces and UPS–spline surfaces can be used together to describe a model. This avoids that degenerate NURBS patches have to be used. We illustrated this with the example of a driving–mirror of a car. A second application approximates a given NURBS surface with a UPS–spline surface. We derived formulae for control points of a UPS–spline surface that interpolates the functional and partial derivative values of the given NURBS surface component–wise. As an example, we gave a UPS–spline representation of the back of a car seat. It shows that UPS–spline surfaces can be used to describe complex geometric models.

Chapter 7

Conclusions

This thesis studies the use of Powell–Sabin splines in their normalized B–spline representation in CAGD. This chapter will give our general conclusions arranged in the following order:

1. the importance of Powell–Sabin splines in CAGD,
2. the main results of this thesis,
3. suggestions for future research.

The importance of Powell–Sabin splines in CAGD

PS–splines in their normalized B–spline representation have a number of advantages compared to tensor product B–splines and NURBS, todays standard in CAD applications, and to the theoretically well elaborated Bernstein–Bézier representation of polynomials on triangles:

1. PS–splines are defined over arbitrary conforming triangulations of polygonal domains, so they allow to represent surfaces with an arbitrary number of edges. Tensor product B–splines and NURBS are restricted to a rectangular geometry.
2. Local adaptivity is possible using an appropriate triangulation, in contrast to the regular panel structure that the knots impose on the rectangular domain of tensor product B–splines and NURBS.
3. In their normalized B–spline representation, PS–splines perform better than triangular Bernstein–Bézier polynomials, and they enjoy similar properties as those that have made NURBS so popular:

   (a) association of the B–spline coefficients with control triangles that are tangent to the surface,
   (b) affine invariance,
   (c) local control, a property that triangular Bernstein–Bézier polynomials do not share,
   (d) the convex hull property,
   (e) they have global C^1–continuity regardless the choice of the B–spline coefficients, while continuity conditions for complexes of triangular Bernstein–Bézier polynomials are hard to implement.

4. The B–spline basis can be constructed by solving a number of small–scale quadratic programming problems.

5. PS–splines can be obtained as the solution of an interpolation problem where functional and derivative values are given at the vertices of the triangulation. This allows to approximate known shapes by PS–spline surfaces.

6. The Bernstein–Bézier representation of PS–splines can be calculated in an efficient and numerically stable way. Therefore, we can rely on the well elaborated theory of triangular Bernstein–Bézier polynomials, and efficient calculations schemes such as the de Casteljau algorithm.

The main results of this thesis

NURPS: an extension of PS–spline surfaces. We considered the generalization of PS–spline surfaces to their rational form, i.e. NURPS surfaces, and showed that this representation has all the features to make it attractive in CAGD. We have given a numerically stable algorithm for calculating the rational Bernstein–Bézier representation of NURPS surfaces, based on the idea behind the rational de Casteljau algorithm. Moreover, we have illustrated that any algorithm for PS–splines that uses convex combinations only, can be extended to a numerically stable version for NURPS surfaces. When the rational Bernstein–Bézier representation is known, the rational de Casteljau algorithm from Farin can be used to calculate a point, a tangent vector and a normal vector to the surface. In the NURPS representation, each control point has a weight assigned to it. Remarkably, the weights of the widespread NURBS representation are rarely used in commercial packages. We have shown how the NURPS weights can be used as a shape control tool in a totally transparent way by the concept of shape parameters. Until now, no such means is known for triangular rational Bernstein–Bézier surfaces. Furthermore, the weights allow to achieve local planar sections. Continuing the modeling of special effects, we studied degenerate control triangles. Those do not directly involve the weights, but they allow to model cusps, corners and curved corners. We also investigated the representation of quadric surfaces as NURPS, and gave explicit formulae for a complete cylinder and a cone. Since it is impossible to describe the complete sphere by rational quadratic surface patches, we gave an incomplete representation that allows the user to control the size of the gaps that occur around the equator plane.

UPS–splines: a particular case of PS–splines. We studied the case of triangulations composed entirely of equilateral triangles. This has a number of advantages compared to general PS–splines.

1. A PS–refinement is readily obtained.
2. The form of the PS–triangles can be fixed. This avoids having to solve a quadratic programming problem for each vertex at runtime.
3. The B–spline functions are known beforehand.

All this leads to very efficient calculations with UPS–splines. We have given algorithms for the interpolation problem, for calculating the Bernstein–Bézier representation and for the integration of UPS–splines. We also gave a subdivision scheme that allows to represent a given UPS–spline on a refinement of the initial triangulation. It is an efficient and numerically stable algorithm that calculates the new B–spline coefficients as convex combinations of the original ones. A more general case, k–subdivision was also considered. In order not to over–complicate our formulae, we restricted ourselves to k–subdivision along the edges of the triangulation, which is satisfactory for our purposes in the context of the polygonal hole problem. We gave two applications of subdivision, the UPS–spline wavelet transform and the graphical display of UPS–spline surfaces.

The UPS–spline wavelet transform. We presented a new wavelet transform on a regular triangular grid. The observation that the UPS–spline basis functions are translates and dilates of three functions, led to the definition of a multi–resolution analysis for UPS–splines. Then, we designed a lifting scheme that decomposes a UPS–spline into a low–frequency part and a number of high–frequency components, capturing the details. For the prediction step, we used our subdivision scheme. A simple update step was proposed, that maintains the average of the signal over the low–frequency component. The basis functions of the high–frequency components are the UPS–spline wavelets. We proved that the wavelets that do not cross the boundary of the domain, have a zero integral. Our wavelets combine the advantages of B–spline wavelets, non–separable wavelets and multi–wavelets. For example, the multi–wavelet aspects were explored by making some suggestions for improving the update step, such that the wavelets satisfy several properties at once. We also described the UPS–spline wavelet transform. As an application we gave a thresholding algorithm for noise removal of UPS–spline surfaces. It allows to smooth a given surface without changing the global shape. Our experiments showed that soft thresholding is more favorable than hard thresholding, but the selection of appropriate threshold values is not straightforward.

Displaying NURPS and UPS–spline surfaces. The graphical display of NURPS surfaces can be done by calculating the rational Bernstein–Bézier representation using our rational conversion scheme, followed by our alternative subdivision scheme for rational Bernstein–Bézier surfaces, that is better suited for graphical purposes than the classical one. A number of points on the surface is then obtained. We also investigated the graphical display of UPS–spline surfaces thoroughly. We have given a calculation scheme that generates a number of points on the surface using repeated subdivision. These can be connected as a three direction wireframe. The surface normals at those points are readily available as well, which makes the rendering of realistic images possible. Our algorithm can be extended to rational UPS–spline surfaces in a straightforward manner. The data structure that we used has proven to be very efficient in combination with OpenGL, because it allows to reduce the number of function calls and the redundant processing of vertices by using vertex arrays, without having to calculate additional information.

Approximation and conversion. A solution for the approximation of a NURBS surface with a UPS–spline surface, by component–wise interpolation of functional and partial derivative values, has been given. We also derived several conversion schemes throughout this dissertation.

1. The conversion of a NURPS surface to a composite rational Bernstein–Bézier surface. Numerical stability was obtained by avoiding to work in the homogeneous space, inspired by the idea behind the rational de Casteljau algorithm.
2. The conversion of a given rational Bernstein–Bézier surface to a NURPS surface. There are many possible ways to do this, depending on the choice of the PS–triangles. We have given a

so–called identical representation in which the B–spline control points coincide with the Bernstein–Bézier control points. It is suited for the functional case. For the parametric case where a domain triangle is not known, the optimal conversion scheme may be better suited. It uses optimal PS–triangles on an equilateral domain triangle. Here too, the rational versions of these schemes enhance numerical stability.

3. We also studied the replacement of a PS–triangle of a given PS–spline with another valid one. This has been applied to obtain the uniform representation of an optimal PS–spline on an equilateral domain triangle. The rational extension allowed to calculate rational UPS–spline representations of surfaces that were given in their optimal NURPS representation. These could then be displayed using the rational extension of our visualization scheme based on subdivision. For example, our representation of the cylinder, cone and sphere has been visualized.

The polygonal hole problem. We presented an algorithm for calculating a UPS–spline surface that fills a hole, bounded by a set of surfaces. Because there are no assumptions on the mathematical representation of the surrounding surfaces, it is generally applicable. The boundary curves are fitted approximately, which certainly makes sense for practical purposes. The user input consists of evaluation routines for the boundary curves, their tangent vectors and normal vectors to the surrounding surfaces, parameterized on the unit interval. First an initial solution which is smooth is determined. It is then refined iteratively on the boundary, by interpolating the given data in a number of points. This leaves some degrees of freedom, which allow to fit the curves in between. Finally, when the approximation on the boundary is close enough, the interior control points are determined. We have first given a number of intuitive approaches to do so, based on subdivision, averaging and k–subdivision. We also studied the smoothing of these results using our thresholding algorithm. A different, mathematically more sound approach, was given by an algorithm for fairing UPS–spline surfaces. The quality of these alternatives has been compared using Gaussian curvature plots. From our experiments, we can conclude that a generally applicable method is hard to find. From the possibilities that we examined, the fairing method and the cheaper "instant update" method delivered the best results. The choice depends on what the user wants. For example, when filling the hole in the sphere, we wanted to obtain a very smooth interior, so we preferred the fairing method; but when calculating triangular patches for the driving mirror model, the "instant update" method was better suited because fairing yielded a surface that was too flat. Our solution to the polygonal hole problem has been shown to be applicable to other cases as well, e.g., blending problems and placing a cap on top of a prism.

Fairing UPS–spline surfaces. We reviewed a measurement of the fairness of UPS–splines, and used it to solve a constrained fairing problem: given a UPS–spline surface, find the interior control triangles such that the surface is fair, leaving the control triangles on the boundary untouched. We gave a solution that minimizes the component–wise jump in second order derivatives across the PS–refinement lines, as the least–squares solution of an overdetermined system. The particular case where the triangulation is a molecule with molecule number six has been solved explicitly. It is useful for determining an initial solution for polygonal holes with five or six edges.

PS–surf: a CAD software prototype. We have written a software package that illustrates the results of this thesis. It allows modeling with rational UPS–spline surfaces, and it can import existing NURBS models from IGES files. As for the graphical display of the rational UPS–spline surfaces, the rational extension of our visualization algorithm is used in conjunction with OpenGL. We opted for the block matrix data structure because it inherently contains topological information of the triangulation, and it allows to readily use some OpenGL primitives that enhance the efficiency of the display. The number of functions calls and the redundant vertex processing are thereby reduced to a minimum. Globally, PS–surf consists of a number of portable libraries with on top of it a menu–driven, dialogue–based window application. As for the internal structure, the advantages of the object–oriented design have been illustrated, in particular the use of inheritance and abstract classes. Completely new geometric entities or new applications for existing objects can be smoothly integrated reusing a great deal of code. PS–surf is ready for the extension towards general PS–spline surfaces, and a suggestion for an object–oriented data structure has been made. We also used PS–surf in realistic modeling examples. First, a model of a driving-mirror was given that uses NURBS and UPS–spline surfaces all together, avoiding degenerate NURBS patches. Secondly, the back of a car seat was represented using UPS–spline surfaces with three and four edges, by approximating a given NURBS CAD model. This shows that UPS–spline surfaces can be used to model complex, realistic shapes.

Suggestions for future research

1. It is well known that parametric continuity is often too severe to impose smoothness conditions. For example, in the case of B–spline curves, it can be relaxed to geometric continuity, also called visual smoothness. This even yields some extra degrees of freedom, which give the designer more flexibility. It is called the β–spline representation. It is an open question whether the C^1–continuity conditions for PS–splines can be relaxed to G^1 geometric continuity, and whether this yields extra degrees of freedom as in the case of β–splines.

2. Regarding our thresholding algorithm, it would be interesting to have a procedure for selecting an appropriate threshold, maybe based on well–known techniques such as the method of the generalized cross validation function.

3. The update step of our lifting scheme is kept as simple as possible. Yet, the multi–wavelet character of UPS–spline wavelets allows to design much more complex update boxes. In that way, wavelets can meet several conditions at once, e.g., orthogonality, symmetry, short support, continuity, polynomial reproduction,... We have already made some suggestions about how the UPS–spline wavelets near the boundary can be imposed to have a zero integral. It should be further investigated what the possibilities in this respect are, and where this can be useful in practical applications.

4. Our polygonal hole algorithm calculates a surface that interpolates the given boundary curves in a number of points. The control triangles at those interpolation points have two degrees of freedom: the β–and the α–factors. The latter are determined such that the boundary curves are fitted in between, and the β–factors are determined by fitting the cross–boundary tangent vectors. It can be checked that rescaling the β–factors does not influence the boundary of the filling patch at all, but merely has an effect on the interior of the patch. It is worth to investigate whether, while the α–factors are used to fit the boundary curves as close as possible, the β–factors can be used to obtain a smooth interior of the patch.

5. Another extension of our polygonal hole algorithm could be the use of rational UPS–spline surfaces instead of non–rational ones. It seems an interesting, but surely non–trivial challenge to exploit the use of the weights as to obtain a smooth interior of the filling patch.

6. Also the use of convexity constraints (see [82]) for determining the interior of the filling patch is worth investigating.

7. The most obvious, but certainly not the most uninteresting possibility for future research, is the extension of our results regarding UPS–splines towards general PS–splines. With this respect, we should note that after a fruitful collaboration, very recently Vanraes et al. [77] derived a subdivision scheme for PS–spline surfaces on arbitrary triangulations. In view of the results of this dissertation, it has a great potential for research topics:

   (a) The graphical display of general PS–splines can be implemented more efficiently than with our subdivision scheme for rational Bernstein–Bézier surfaces. However, the data structure will be much more complex than our block matrix structure and overhead should be kept to a minimum. Therefore, the regularity that naturally arises after some subdivision steps should be incorporated in the data structure, and for optimizing performance in OpenGL by keeping functions calls and redundant vertex processing to a minimum.

   (b) Subdivision for PS–splines can be used as a prediction step for a general PS–spline wavelet transform. Such a transform would yield second generation non–separable B–spline multi–wavelets. Their properties are worth a thorough investigation. For example, we may expect that the irregular structure of the triangulation will give raise to different orientation properties than is the case for UPS–spline wavelets. Furthermore, the applicability of the PS–spline wavelet transform in classical domains as compression and de–noising in geometric processing is worth to be investigated.

   (c) Willemans [82] studied constrained fitting problems with Powell–Sabin spline surfaces. It would be interesting to combine these results with subdivision techniques. For example, a procedure for data fitting using general PS–splines, based on repeated subdivision (with or without constraints), would be very useful. If through this process, a molecule M_i would appear that contains no data points, this is not a problem because the control triangle at V_i can be determined by subdivision of the previous solution. Such an algorithm would yield a surface that has a PS–spline wavelet transformable structure, which is useful for e.g. application of the thresholding algorithm as a smoothing step.

We believe that PS–splines have a future in CAGD and geometric processing, and hope that this dissertation is a first step in their fundamental and practical elaboration.

Bibliography

[1] picTeX package available at www.ctan.org, manual for sale at Wichura@Galton.Uchicago.EDU.

[2] C. Bangert and H. Prautzsch. Circle and sphere as rational splines. *Neural, Parallel and Scientific Computations*, 5:153–162, 1997.

[3] C. Bangert and H. Prautzsch. Quadric splines. *Computer Aided Geometric Design*, 16:497–515, 1999.

[4] W. Boehm. Some remarks on quadrics. *Computer Aided Geometric Design*, 10:231–236, 1993.

[5] W. Boehm and D. Hansford. Bézier patches on quadrics. In G. Farin, editor, *NURBS for Curve and Surface design*, chapter 1, pages 1–14. SIAM, 1991.

[6] C. Brislawn. Classification of symmetric wavelet tranforms. Technical report, Los Alomos, 1993.

[7] C. Brislwan. Classification of nonexpansive symmetric extension transforms for multirate filterbanks. *Appl. Comput. Harmon. Anal.*, 3:337–357, 1996.

[8] B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan, and S. M. Watt. *Maple V Language Reference Manual.* Springer Verlag, 1992.

[9] P. Charrot and A. Gregory. A pentagonal surface patch for computer aided geometric design. *Computer Aided Design*, 1:87–94, 1984.

[10] C. K. Chui and M.-J. Lai. Filling polygonal holes using C^1 cubic triangular spline patches. *Computer Aided Geometric Design*, 17:297–307, 2000.

[11] A. Cohen and J.-M. Schlenker. Compactly supported bidimensional wavelet bases with hexagonal symmetry. *Constr. Approx.*, 9:209–236, 1993.

[12] M. Cotronei, L. Montefusco, and L. Puccio. Multiwavelet analysis and signal processing. *IEEE Transactions on Circuits and Systems II*, 1997.

[13] I. Daubechies and W. Sweldens. Factoring wavelet transforms into lifting steps. *J. Fourier Anal. Appl.*, 4(3):245–267, 1998.

[14] G. M. Davis, V. Strela, and R. Turcajová. Multiwavelets construction via the lifting scheme. In *Wavelet Analysis and Multiresolution Methods.* Marcel Dekker, 1999.

[15] O. Davydov and L. L. Schumaker. On stable local bases for bivariate polynomial spline spaces. *Constr. Approx.*, 18:87–116, 2002.

[16] P. Dierckx. An algorithm for surface fitting with spline functions. *IMA J. Num. Anal.*, 1:267–283, 1981.

[17] P. Dierckx. Fitpack user guide part 2: surface fitting routines. Technical Report TW 122, Departement of Computer Science, Katholieke Universiteit Leuven, Belgium, 1989.

[18] P. Dierckx. *Curve and surface fitting with splines.* Clarendon Press, Oxford, 1995.

[19] P. Dierckx. On calculating normalized Powell–Sabin B–splines. *Computer Aided Geometric Design*, 15:61–78, 1997.

[20] P. Dierckx, S. Van Leemput, and T. Vermeire. Algorithms for surface fitting using Powell–Sabin splines. *IMA Journal of Numerical Analysis*, 12:271–299, 1992.

[21] R. Dietz. *Rationale Bézier–Kurven und Bézier–Flächenstücke auf Quadriken.* PhD thesis, Technische Hochschule Darmstadt, 1995.

[22] R. Dietz, J. Hoschek, and B. Jüttler. An algebraic approach to curves and surfaces on the sphere and on other quadrics. *Computer Aided Geometric Design*, 10:211–229, 1993.

[23] R. Dietz, J. Hoschek, and B. Jüttler. Rational patches on quadric surfaces. *Computer Aided Design*, 27(1):27–40, 1995.

[24] D. L. Donoho, I. M. Johnstone, G. Kerkyacharian, and D. Picard. Wavelet shrinkage: Asymptopia? *J. R. Statist. Soc. B.*, 57(2):301–337, 1995.

[25] David L. Donoho. Wavelet shrinkage and w.v.d.: A 10-minute tour.

[26] David L. Donoho. De-noising by soft-thresholding. *IEEE Transactions on Information Theory*, 41(3):613–627, 1995.

[27] David L. Donoho and Iain M. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81(3):425–455, 1994.

[28] G. Farin. Bézier polynomials over triangles and the construction of piecewise C^r polynomials. TR 91, Brunel University, Uxbridge, England, 1980.

[29] G. Farin. Triangular Bernstein–Bézier patches. *Computer Aided Geometric Design*, 3(2):83–127, 1986.

[30] G. Farin. *Curves and Surfaces for CAGD, a practical guide.* Academic Press, fourth edition, 1996.

[31] G. Farin, B. Piper, and A. J. Worsey. The octant of a sphere as a nondegenerate triangular Bézier patch. *Computer Aided Geometric Design*, 4:329–332, 1987.

[32] J. A. Gregory, V. K. H. Lau, and J. M. Hahn. High order continuous polygonal patches. In G. Farin, H. Hagen, and H. Noltemeier, editors, *Geometric Modelling.* Springer–Verlag Wien, 1993.

[33] H. Hagen, S. Hahmann, T. Schreiber, Y. Nakajima, and P. Holleman-Grundstedt. Surface interrogation algorithms. *IEEE Computer Graphics and Applications*, 12(5):53–60, September 1992.

[34] W. Hohenberger and T. Reuding. Smoothing rational B–spline curves using the weights in an optimization procedure. *Computer Aided Geometric Design*, pages 837–848, July 1995.

[35] D. Huang, Q. Sun, and W. Wang. Multiresolution generated by several scaling functions. *Adv. Math. (China)*, 26, 1997.

[36] Sanchez-Reyes J. A simple technique for NURBS shape modification. *IEEE Computer Graphics and Applications*, pages 52–59, January–February 1997.

[37] M. Jansen and A. Bultheel. Multiple wavelet threshold estimation by generalized cross validation for images with correlated noise. *IEEE Trans. on Image Proc.*, 8(7):947–953, 1999.

[38] M. Jansen, M. Malfait, and A. Bultheel. Generalized cross validation for wavelet thresholding. *Signal Processing*, 56(1):33–44, January 1997.

[39] Maarten Jansen. *Noise Reduction by Wavelet Thresholding*, volume 161 of *Lecture Notes in Statistics*. Springer Verlag, 2001.

[40] Xiang-Gen K. and B. W. Suter. Construction of Malvar wavelets on hexagons. In *Appl. Comput. Harmonic Anal*, volume 3, pages 65–71. 1996.

[41] Peter Kabal. *TEXdraw*, second edition, 1995.

[42] A. Kerstens. *Een geïntegreerde aanpak voor het modelleren van CAM–bewerkbare vrije–vorm oppervlakken in een reverse engineering omgeving*. PhD thesis, Kath. Universiteit Leuven, 1999.

[43] Mark J. Kilgard. *The OpenGL Utility Toolkit (GLUT) Programming Interface*. Silicon Graphics, api version 3 edition, November 1996.

[44] Leif Kobbelt. Discrete fairing. In *Proceedings of the Seventh IMA Conference on the Mathematics of Surfaces*, pages 101–131, 1997.

[45] J. Kovačević and W. Sweldens. Wavelet families of increasing order in arbitrary dimensions. *IEEE Trans. Image Proc.*, 9(3):480–496, March 2000.

[46] M.-J. Lai and L. L. Schumaker. Macro-elements and stable local bases for splines on Clough–Tocher triangulations. *Numer. Math.*, 88:105–119, 2001.

[47] M.-J. Lai and L. L. Schumaker. Macro-elements and stable local bases for splines on powell-sabin triangulations. *Math. Comp.*, 2002. to appear.

[48] A. Laine and S. Schuler. Hexagonal wavelet processing of digital mammography. In M. H. Loew, editor, *Medical Imaging 1993: Image Processing*, volume 1898 of *Spie Proceedings*, pages 559–573, 1993.

[49] A. Laine and S. Schuler. Hexagonal QMF banks and wavelets. In M. Akay, editor, *Time-Frequency and Wavelet Transforms in Biomedical Engineering*. IEEE Press, 1997.

[50] A. Levin. Filling an n–sided hole using combined subdivision schemes. In A. Cohen, C. Rabut, and L.L. Schumaker, editors, *Curve and Surface Design: Saint–Malo 1999*. Vanderbilt University Press, Nashville, 2000.

[51] N.J. Lott and D.I. Pullin. Methods for fairing B–splne surfaces. *Computer Aided Design*, 20(10):597–604, 1988.

[52] F. Muller, P. Brigger, K. Illgner, and M. Unser. Multiresolution approximation using shifted splines, 1998.

[53] H. Nowacki and D. Reese. Design and fairing of ship surfaces. In R.E. Barnhill and W. Boehm, editors, *Surfaces in computer aided geometric design*, pages 121–134. North–Holland, Amsterdam, 1983.

[54] Nicholas M. Patrikalakis and Takashi Maekawa. *Shape Interrogation for Computer Aided Design and Manufacturing.* Springer Verlag, 2000.

[55] R. S. Pfisterer and F. Aghdasi. Hexagonal wavelets for the detection of masses in digitised mammograms. volume 3813 of *Spie Proceedings*, 1999.

[56] G. Plonka and V. Strela. From wavelets to multiwavelets. In M. Dahem, T. Lyche, and L. Shumaker, editors, *Mathematical Methods for Curves and Surfaces II*, pages 1–25. Vanderbilt University Press, Nashville, TN, 1998.

[57] M.J.D. Powell and M. Sabin. Piecewise quadratic approximations on triangles. *ACM Transactions on Mathematical Software*, 3:316–325, 1977.

[58] H. Qin and D. Terzopoulos. D–NURBS: a physics–based framework for geometric design. *IEEE Transactions on Visualisation and Computer Graphics*, 2(1):85–96, 1996.

[59] L. Ramshaw. Blossoming: a connect-the-dots approach to splines. Technical report, Digital Systems Research Center, Palo Alto, CA, 1987.

[60] P. Sablonnière. Error bounds from Hermite interpolation by quadratic splines on a α–triangulation. *IMA Journal of Numerical Analysis*, 7:495–508, 1987.

[61] R. Schneider and L. Kobbelt. Geometric fairing of irregular meshes for free form surface design. *Computer Aided Geometric Design*, to appear.

[62] L. L. Schumaker. On the dimension of spaces of piecewise polynomials in two variables. In W. Schempp and K. Zeller, editors, *Multivariate Approximation Theory*, pages 396–412. Birkhäuser, Basel, 1979.

[63] L. L. Schumaker. Bounds on the dimension of spaces of multivariate piecewise polynomials. *Rocky Mountain Journal of Mathematics*, 14:251–264, 1984.

[64] L.L. Schumaker and W. Volk. Efficient evaluation of multivariate polynomials. *Computer Aided Geometric Design*, 3(2):149–154, 1986.

[65] T. Sederberg and D. Anderson. Steiner surface patches. *IEEE Comp. Graphics Appl.*, 5:23–36, 1985.

[66] X. Shi, S. Wang, W. Wang, and R. H. Wang. The C^1–quadratic spline space on triangulations. Report 86004, Department of Mathematics, Jilin University, Changchun, 1986.

[67] E. P. Simoncelli and E. H. Adelson. Non-separable extensions of quadrature mirror filters to multiple dimensions. volume 78 of *Proc. IEEE*, pages 652–663, 1990.

[68] W. Sweldens. A new philospohy in biorthogonal wavelet constructions. In A. F. Laine and M. Unser, editors, *Wavelet Applications in Signal and Image Processing III*, volume 2569 of *Spie proceedings*, pages 68–79, 1995.

[69] W. Sweldens. The lifting scheme: A custom-design construction of biorthogonal wavelets. *Appl. Comput. Harmon. Anal.*, 3(2):186–200, 1996.

[70] W. Sweldens. The lifting scheme: A construction of second generation wavelets. *SIAM J. Math. Anal.*, 29(2), 1997.

[71] W. Sweldens and P. Schröder. Building your own wavelets at home. In *Wavelets in Computer Graphics*, pages 15–87. ACM SIGGRAPH Course notes, 1996.

[72] H. Theisel. *Vector Field Curvature and Applications.* PhD thesis, University of Rostock, Computer Science Department, 1996.

[73] M. Unser. Ten good reasons for using spline wavelets. In *Proceedings of the SPIE Conference on Mathematical Imaging: Wavelet Applications in Signal and Image Processing V*, volume 3169, pages 422–431, San Diego CA, USA, July 30-August 1 1997.

[74] M. Unser. Splines: A perfect fit for signal and image processing. *IEEE Signal Processing Magazine*, 16(6):22–38, November 1999.

[75] US Product Data Association. *IGES, An American National Standard. Initial Graphics Exchange Secification 5.3.*

[76] G. Uytterhoeven and A. Bultheel. The red-black wavelet transform. Technical Report 271, Department of Computer Science, K.U.Leuven, Belgium, 1997.

[77] E. Vanraes, J. Windmolders, A. Bultheel, and P. Dierckx. Subdivision for Powell–Sabin spline surfaces. Technical Report TW345, Kath. Univeriteit Leuven, Department of Computer Science, 2002. submitted to CAGD.

[78] R.-H. Wang. The dimension and basis of spaces of multivariate splines. *Journal of Compuational and Applied Mathematics*, 12 and 13:163–177, 1985.

[79] M. Weiyin. *NURBS–based CAD modelling from measured points of physical models.* PhD thesis, Katholieke Universiteit Leuven, 1994.

[80] W. Welch and A. Witkin. Free form shape design using triangulated surfaces. In *Computer Graphics (Proc. SIGGRAPH '94)*, volume 28, pages 247–256, 1994.

[81] N. Weyrich and G. T. Warhola. De-noising using wavelets and cross validation. In S. P. Singh, editor, *Approximation Theory, Wavelets and Applications*, volume 454 of *NATO ASI Series C*, pages 523–532. 1995.

[82] K. Willemans. *Smoothing scattered data with a constrained Powell–Sabin spline surface.* PhD thesis, Katholieke Universiteit Leuven, 1996.

[83] M. Woo, J. Neider, and T. Davis. *OpenGL programming guide. The official guide to learning OpenGL, version 1.1.* Addison–Wesley developers press, 1997.

Publications

1. Windmolders J., and P. Dierckx, *Subdivision of uniform Powell-Sabin splines*, Computer Aided Geometric Design 16, 301-315, 1999.

2. Windmolders J., and P. Dierckx, *From PS-splines to NURPS*, proceedings of Curve and Surface Fitting, Saint-Malo 1999, Albert Cohen, Christophe Rabut, and Larry L. Schumaker (eds.), Vanderbilt University Press, Nashville, 2000.

3. Windmolders J., and P. Dierckx, *NURPS for special effects and quadrics*, proceedings of Mathematical Methods for Curves and Surfaces, Oslo 2000, Tom Lyche and Larry L. Schumaker (eds.), Vanderbilt University Press, 2001.

4. Windmolders, J. and P. Dierckx, *Uniform Powell-Sabin splines for the polygonal hole problem*, Proceedings of Algorithms for Approximation IV (I. Anderson J. Levesly, ed.), 2001.

5. J. Windmolders, E. Vanraes, P. Dierckx, A. Bultheel, *Uniform Powell-Sabin spline wavelets*, to appear in J. Comput. Appl. Math. (May 2003).

6. E. Vanraes, J. Windmolders, A. Bultheel, P. Dierckx, *Dyadic and sqrt(3) subdivision for uniform Powell Sabin splines*, Proceedings of the Sixth International Conference on Information Visualisation (D. Williams, ed.), IEEE Computer Society, 2002, pp. 639-643.

7. E. Vanraes, J. Windmolders, P. Dierckx, A. Bultheel, *Subdivision of Powell-Sabin spline surfaces*, 2002, submitted to CAGD.