ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Αριθμητικές Μέθοδοι για την υλοποίηση γραφικών με Υπολογιστή

Διπλωματική Εργασία Ειδίκευσης
στα Εφαρμοσμένα Μαθηματικά

Χειλάκος Νικόλαος

Επιβλέπουσα: Μητρούλη Μαριλένα

Σχολή Θετικών Επιστημών
Τμήμα Μαθηματικών
Εθνικόν και Καποδιστριακόν Πανεπιστήμιο Αθηνών
2021

Η παρούσα Διπλωματική Εργασία

εκπονήθηκε στο πλαίσιο των σπουδών

για την απόκτηση του

Διπλώματος Μεταπτυχιακών Σπουδών

με ειδίκευση στη

Κατεύθυνση Εφαρμοσμένα Μαθηματικά

που απονέμει το

Τμήμα Μαθηματικών

του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών

στον/στην Χειλάκο Νικόλαο

Εγκρίθηκε την 30η Σεπτεμβρίου 2021 (ημέρα/μήνας/έτος)

από Εξεταστική Επιτροπή αποτελούμενη από τους:

Ονοματεπώνυμο		Βαθμίδα	Υπογραφή
Μητρούλη Μαριλένα	(Επιβλέπων/-ουσα Καθηγητής/Καθηγήτρια)	Καθηγήτρια	
Νοτάρης Σωτήριος		Καθηγητής	
Δρακόπουλος Μιχαήλ		Επ. Καθηγητής	

Πρόλογος

Η διπλωματική αύτη εργασία, αποτελεί ένα ακόμα μέρος της ενασχόλησης μου, με τον χώρο των Γραφικών με Υπολογιστή, από τη σκοπιά του μαθηματικού και όχι του απλού χρήστη ενός μηχανήματος. Η αρχή έγινε μέσω του προπτυχιακού προγράμματος σπουδών και συνεχίζεται στο πλαίσιο των μεταπτυχιακών σπουδών μου στο τμήμα Μαθηματικών.

Στο παρόν κείμενο θα προσπαθήσω να παρουσιάσω όσο τον δυνατό καλύτερα μεθόδους από την Αριθμητική Ανάλυση, που αποτελούν τα θεμέλια στο οικοδόμημα των σύγχρονων γραφικών με υπολογιστή.

Το πρώτο κεφάλαιο είναι μια εισαγωγή στα συστήματα συντεταγμένων, στους μετασχηματισμούς στο επίπεδο και στο χώρο, καθώς και μια παρουσίαση αλγορίθμων σχεδίασης βασικών σχημάτων σε διακριτό πλέγμα εικονοστοιχείων. Ο λόγος που τα παρουσιάζω είναι για να καταδείξω ότι υπάρχουν εργαλεία, όπου τα αποτελέσματα που θα δώσουν οι μέθοδοι που θα δούμε στο κύριο κομμάτι της εργασίας, επιτρέπουν την απεικόνιση τους στις οθόνες των υπολογιστών.

Στα επόμενα κεφάλαια , που αποτελούν και το κύριο θέμα της εργασίας, μελετάμε το πρόβλημα της παρεμβολής και της προσέγγισης καμπυλών και συγκρίνουμε τα διάφορα εργαλεία.

Επέλεξα τους κώδικες των προγραμμάτων και των συναρτήσεων να τους γράψω με το υπολογιστικό πακέτο Matlab, όχι μόνο διότι το γνωρίζω αρκετά καλά, αλλά γιατί ο στόχος της εργασίας δεν ήταν να φτιαχτεί μια βιβλιοθήκη συναρτήσεων γραφικών για σχεδιασμό, υπάρχουν ήδη πολύ καλύτερες, αλλά να μπούμε στην ουσία των αλγορίθμων των μεθόδων. Και το συγκεκριμένο εργαλείο προσφέρει αυτή την δυνατότητα. Σε κάποια σημεία, όπως για παράδειγμα όταν χρειάστηκε η επίλυση κάποιου γραμμικού συστήματος, επέλεξα να χρησιμοποιήσω εργαλεία που διαθέτει το Matlab για να μείνει όσο γίνεται πιο απλός ο κώδικάς μου.

Έκρινα επίσης ότι θα ήταν καλύτερο να ενσωματώσω τους κώδικές μου με το υπόλοιπό κείμενο, και να μην τους βάλω όλους μαζί στο τέλος, γιατί μέσω αυτών παρουσιάζεται και ο αλγόριθμός της κάθε μεθόδου στο σημείο που αναφέρεται. Οι κώδικες γράφτηκαν στην έκδοση R2019a χωρίς χρήση κάποιου επιπρόσθετου toolbox και για αυτό μπορούν να τρέξουν και σε Octave.

Χειλάκος Νίκος
2021

Abstract

This Master Thesis is only a glimpse in the Computer Graphics' world and the bond with Mathematics. I will try to present some tools from Numerical Analysis, that will be the foundations of Computer Graphics.

The first chapter is an introduction to the basic mathematic tools, that we will need in our journey to this digital world. Two-Dimensional and Three-Dimensional Transformations, Projections and Algorithms for elementary shapes, such us line and circle.

Chapters Two and Three are the core of this thesis. In these we will see, how we can use Interpolation and Approximation methods to generate curves in computer graphics.

For the programming part, I have used Matlab R2019a. Not only because I am familiar with this tool, but also because the main purpose was to demonstrate the algorithms and not to make a computer graphic library. Matlab's codes are very comprehensible to read and also has a big variety of useful tools.

Cheilakos Nick
2021

Ευχαριστίες

Θα ήθελα να ευχαριστήσω την καθηγήτρια Μαριλένα Μητρούλη, εκτός της τιμής που μου έκανε να είναι η επιβλέπουσα στην εργασία μου, ήταν το μάθημα της Γραφικά με Ηλεκτρονικούς Υπολογιστές, που μου έδειξε την βαθύτερη σύνδεση μεταξύ Μαθηματικών και Computer Graphics. Η καθοδήγηση της όλα αυτά τα χρόνια ήταν πολύτιμη για εμένα και μόνο κερδισμένος μπορώ να αισθάνομαι από την συνεργασία μαζί μας.

Επίσης ευχαριστώ τους κυρίους Δρακόπουλο και Νοτάρη που μου έκαναν την τιμή να είναι μέλη της τριμελής μου επιτροπής. Αλλά και γιατί μέσω της διδασκαλίας τους, τόσο σε προπτυχιακό όσο και σε μεταπτυχιακό επίπεδο, ήρθα σε επαφή με διάφορους κλάδους της Αριθμητικής Ανάλυσης.

Κλείνοντας, ένα μεγάλο ευχαριστώ σε όσους ανθρώπους ήταν δίπλα μου και αποτέλεσαν στήριγμα στη ζωή μου όποτε τους χρειάστηκα.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

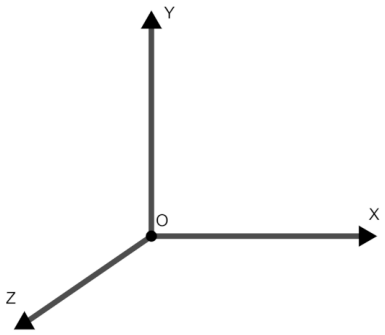
Όταν εργαζόμαστε με έναν υπολογιστή περιοριζόμαστε σε έναν πεπερασμένο διακριτό κόσμο. Είναι αυτονόητο, ότι το ίδιο ισχύει και με τα γραφικά. Καλούμαστε να σχεδιάσουμε έναν κόσμο τριών διαστάσεων σε ένα περιβάλλον δύο διαστάσεων όσο γίνεται πιο πιστά. Στο κεφάλαιο αυτό θα γίνει μια παρουσίαση των βασικών εργαλείων που μας επιτρέπουν να κάνουμε την μετάβαση από το πραγματικό κόσμο στον κόσμο που βλέπουμε στην οθόνη μας. Επίσης θα δούμε ενδεικτικά κάποιους αλγόριθμους που επιτρέπουν την δυνατόν καλύτερη απεικόνιση βασικών σχημάτων σε πλεγματικές οθόνες.

1.1 Σύστημα συντεταγμένων

Για να καθορίσουμε στο επίπεδο και στο χώρο ένα σημείο, συνηθίζουμε να χρησιμοποιούμε κατάλληλο ορθογώνιο σύστημα συντεταγμένων. Ορίζουμε ένα σημείο αναφοράς Ο σαν αρχή και σχεδιάζοντας τους αντίστοιχους ορθογώνιους άξονες με αρχή το σημείο αυτό. Ένα τέτοιο σύστημα συντεταγμένων ονομάζεται καρτεσιανό.

Για να προσδιορίσουμε ένα σημείο στο σύστημα καρτεσιανών συντεταγμένων του επιπέδου, αρκεί ένα διατεταγμένο ζεύγος πραγματικών αριθμών.

Έστω $x, y \in \mathbb{R}$ τότε το (x, y) είναι το σημείο που τέμνονται οι κάθετες που θα φέρουμε από τους άξονες Ox και Oy στα σημεία x και y. Όμοια για τον καθορισμό της θέσης ενός σημείου στο χώρο αρκεί μια τριάδα αριθμών. Ειδικότερα για την εργασία, σε ότι αφορά το καρτεσιανό σύστημα συντεταγμένων των τριών διαστάσεων, θα κάνουμε χρήση του δεξιόστροφου συστήματος.



Σχήμα 1.1: Δεξιόστροφο σύστημα συντεταγμένων

1.2 Συσχετισμένοι μετασχηματισμοί

Στα γραφικά είναι συχνά απαραίτητο να προβούμε σε αλλαγές των συντεταγμένων ενός σχήματος , για παράδειγμα αν θέλουμε να το χρησιμοποιήσουμε πολλές φορές σε μια εικόνα. Για τον λόγο αυτό χρησιμοποιούμε μετασχηματισμούς συντεταγμένων.

Ορισμός 1.1 *Συσχετισμένος συνδυασμός σημείων* $P_0, P_1, \cdots, P_n \in \mathbb{R}^3$*, είναι ένα σημείο* $P \in \mathbb{R}^3$ *που ορίζεται ως*

$$P = \sum_{i=0}^{n} a_i P_i$$

με $a_0, a_1, \cdots, a_n \in \mathbb{R}$ *και* $\sum_{i=0}^{n} a_i = 1$ *[17, σελ. 68]*

Ειδικότερα κάνουμε χρήση συσχετισμένων μετασχηματισμών, οι οποίοι διατηρούν τις βασικές ιδιότητες των σχημάτων, όπως τους λόγους των αποστάσεων την παραλληλία, τους συσχετισμένους συνδυασμούς, τις ευθείες γραμμές και τους λόγους αναλογιών.

Ορισμός 1.2 *Συσχετισμένος μετασχηματισμός, είναι ένας μετασχηματισμός που διατηρεί του συσχετισμένους συνδυασμούς.*

Δηλαδή $\Phi : \mathbb{R}^3 \mapsto \mathbb{R}^3$ *είναι συσχετισμένος αν για* $P = \sum_{i=0}^{n} a_i P_i$ *συσχετισμένο συνδυασμό τότε* $\Phi(P) = \sum_{i=0}^{n} a_i \Phi(P_i)$*. [17, σελ. 69]*

Η ιδιότητα αυτή των συσχετισμένων μετασχηματισμών, μας δίνει την δυνατότητα, αν θέλουμε για παράδειγμα να εφαρμόσουμε έναν τέτοιο μετασχηματισμό σε ένα τρίγωνο, αντί να πρέπει να τον εφαρμόσουμε σε κάθε σημείο του, να τον εφαρμόσουμε μόνο στις κορυφές του. Και επομένως να μειώσουμε το κόστος υπολογισμού για να βρούμε το μετασχηματισμού κάθε σημείου του.

Παρακάτω θα παρουσιάσουμε τους βασικούς συσχετισμένους μετασχηματισμούς για τις δύο και τις τρεις διαστάσεις. Τους λέμε βασικούς, γιατί οποιοδήποτε άλλος μετασχηματισμός προκύπτει από την σύνθεση τους. Για να μπορούμε να επιτυγχάνουμε εύκολα την σύνθεση μετασχηματισμών, θα χρησιμοποιούμε μια επιπλέον διάσταση σε σχέση με τον χώρο που θα βρισκόμαστε κάθε φορά. Με τη χρήση ομογενών συντεταγμένων μπορούμε να εκφράσουμε όλους τους βασικούς μετασχηματισμούς με πίνακες ίδιας διάστασης.

Ορισμός 1.3 *Ομογενείς συντεταγμένες ενός σημείου* $P = (x_1, x_2, \cdots, x_n)$ *του* $\mathbb{R}^n$ *για κάθε* $z \in \mathbb{R} \setminus \{0\}$*,ονομάζεται η* $(n+1)$*-αδα* $(x_1 z, x_2 z, \cdots, x_n z, z)$

Ορισμός 1.4 *Το σημείο* $P = (x, y)$ *του* $\mathbb{R}^2$ *σε ομογενείς συντεταγμένες θα το γράφουμε* $P = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$*. Ενώ το σημείο* $P = (x, y, z)$ *του* $\mathbb{R}^3$ *σε ομογενείς συντεταγμένες θα το γράφουμε* $P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^T$ *.*

Οι χρήση ομογενών συντεταγμένων μας δίνει την δυνατότητα να εκφράσουμε τους μετασχηματισμούς σε πίνακες ίδιας διάστασης και έτσι να μπορούμε να ορίσουμε και την σύνθεση τους σαν έναν απλό πολλαπλασιασμό πινάκων.

1.2.1 Μετασχηματισμοί στο επίπεδο

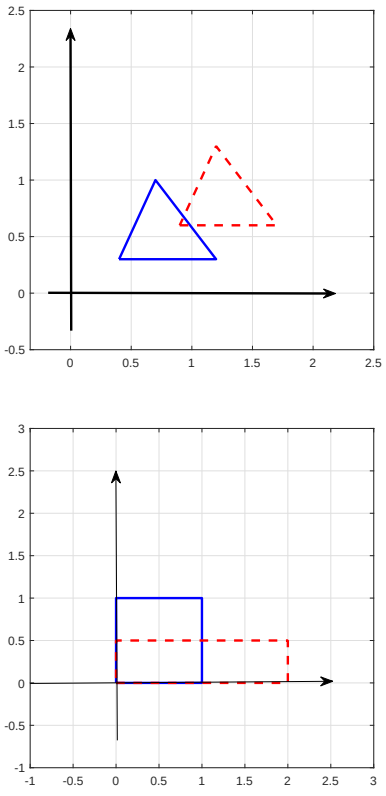
Ορισμός 1.5 *Γεωμετρικός μετασχηματισμός είναι μια 1-1 και επί συνάρτηση που το πεδίο ορισμού και το σύνολο τιμών της είναι σημεία. [2, page 202]*

Πίνακες βασικών μετασχηματισμών στο επίπεδο

Οι ακόλουθοι είναι οι βασικοί γεωμετρικοί μετασχηματισμοί στο επίπεδο σε μορφή 3×3 πινάκων.

- Μεταφορά κατά το διάνυσμα
$$\vec{v} = (x, y)$$
$$T_v = \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix}$$

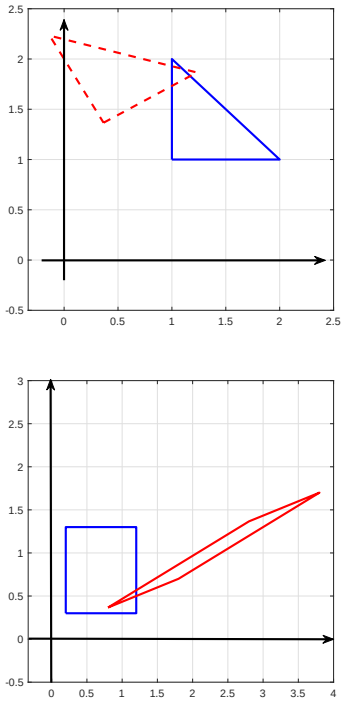
- Αλλαγή κλίμακας (Scaling) ως προς την αρχή των αξόνων, με παράγοντες αλλαγής κλίμακας a, b αντίστοιχα για τον άξονα x και y
$$S_{a,b} = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Σχήμα 1.2: Μεταφορά και Scaling. Με μπλε το αρχικό σχήμα

- Στροφή γύρω από την αρχή των αξόνων κατά γωνία θ
$$R_\theta = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Στρέβλωση (Shearing) ως προς την αρχή των αξόνων, με παράγοντες στρέβλωσης a, b κατά μήκος των αξόνων x και y αντίστοιχα
$$Sh = \begin{bmatrix} 1 & a & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



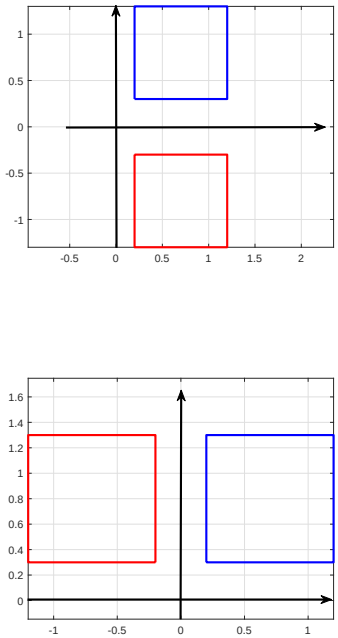
Σχήμα 1.3: Στροφή και Shearing. Με μπλε το αρχικό σχήμα

- Συμμετρία ως προς τον άξονα x

$$M_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Συμμετρία ως προς τον άξονα y

$$M_y = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Σχήμα 1.4: Συμμετρία ως προς τον άξονα x και τον y. Με μπλε το αρχικό σχήμα

1.2.2 Μετασχηματισμοί στο χώρο

Όπως στο επίπεδο έτσι και στο χώρο, κάνουμε χρήση ομογενών συντεταγμένων. Επομένως οι πίνακες των μετασχηματισμών σε αυτή την περίπτωση είναι 4×4. Οι διαφορές που παρατηρούμε εκτός από την διάσταση, είναι ότι οι συμμετρίες σε αυτή τη περίπτωση είναι ως προς τα επίπεδα που ορίζουν οι άξονες. Όπως και η στρέβλωση γίνεται κατά μήκος των αξόνων των επιπέδων. Τέλος έχουμε 3 πίνακες στροφής, αφού αυτή γίνεται γύρω από άξονα και όχι από το Ο.

Πίνακες βασικών μετασχηματισμών στο χώρο

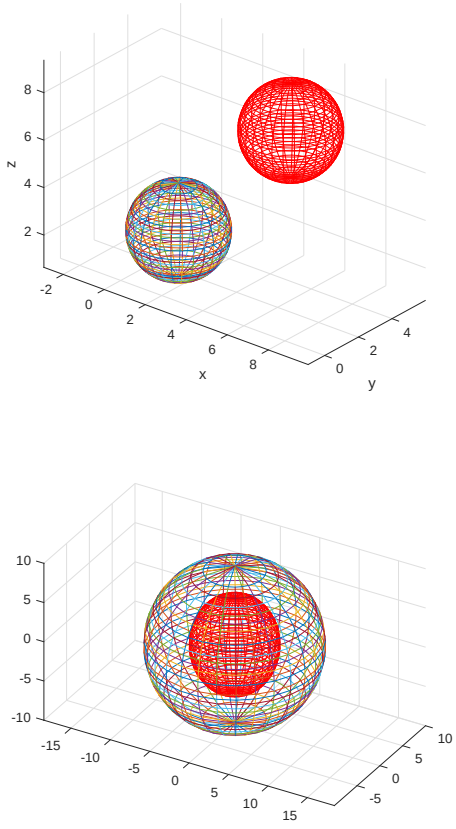
- Μεταφορά κατά το διάνυσμα

$$\vec{v} = (x, y, z)$$

$$T_v = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Αλλαγή κλίμακας (Scaling), με παράγοντες αλλαγής κλίμακας a, b, c αντίστοιχα για τους άξονες x,y και z.

$$S_{a,b,c} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Σχήμα 1.5: Μεταφορά και Scaling. Με κόκκινο το νέο σχήμα

- Στροφή γύρω από τον άξονα x κατά γωνία θ

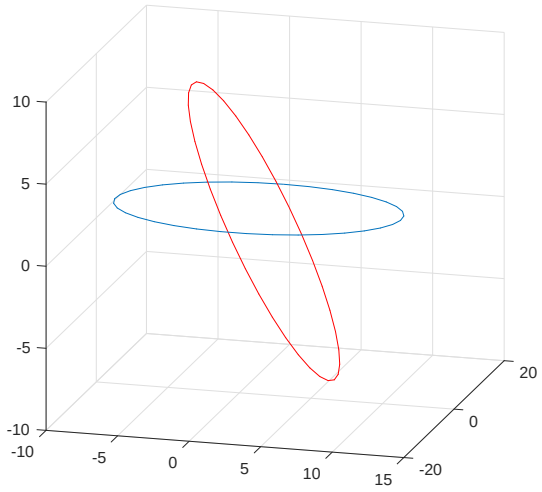
$$R_{\theta,x} = \begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Στροφή γύρω από τον άξονα y κατά γωνία θ

$$R_{\theta,y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Στροφή γύρω από τον άξονα z κατά γωνία θ

$$R_{\theta,z} = \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Σχήμα 1.6: Στροφή γύρω από άξονα. Με κόκκινο το νέο σχήμα

- Συμμετρία ως προς το επίπεδο xy

$$M_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Συμμετρία ως προς το επίπεδο xz

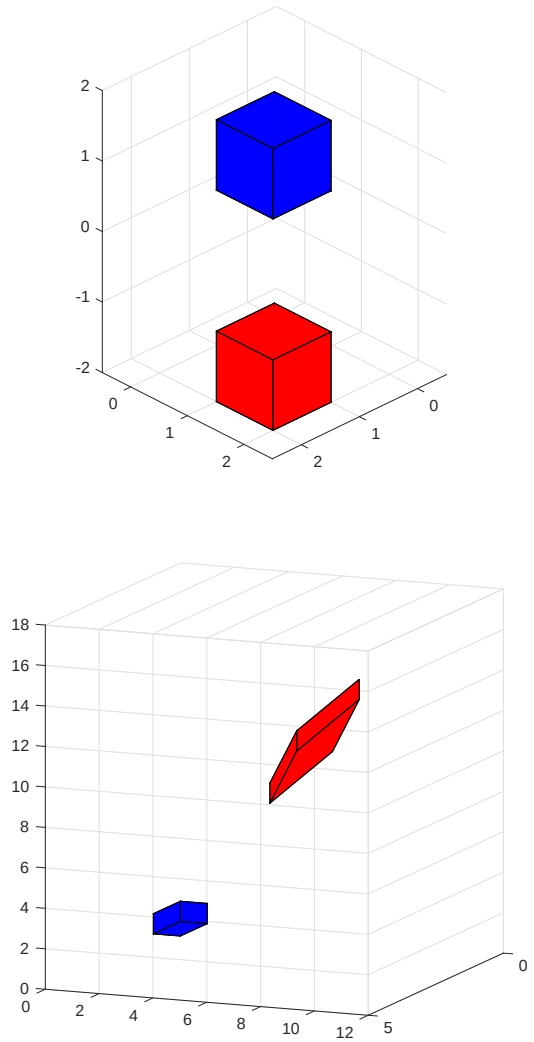
$$M_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Συμμετρία ως προς το επίπεδο yz

$$M_{xy} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Στρέβλωση (Shearing), με παράγοντες στρέβλωσης (yx, zx), (xy, zy) και (xz, yz) κατά μήκος των αξόνων των επιπέδων xy ,xz και yz

$$Sh = \begin{bmatrix} 1 & yx & zx & 0 \\ xy & 1 & zy & 0 \\ xz & yz & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Σχήμα 1.7: Συμμετρία και Shearing. Με κόκκινο το νέο σχήμα

1.2.3 Σύνθεση μετασχηματισμών

Όπως είναι προφανές ένας χρήσιμος μετασχηματισμός για μια διεργασία, σπάνια θα είναι απλά ένας από τους βασικούς, αλλά συνήθως θα προκύπτει από μια αλληλουχία αυτών.

Θα εκμεταλλευτούμε την προσεταιριστική ιδιότητα του πολλαπλασιασμού πινάκων και θα υπολογίσουμε τον πίνακα του σύνθετου μετασχηματισμού που προκύπτει από το γινόμενό τους, και αυτόν θα εφαρμόσουμε. Επειδή δεν ισχύει η αντιμεταθετική ιδιότητα στον πολλαπλασιασμό στους πίνακες, η σειρά που κάνουμε τις πράξεις έχει σημασία.

Ορισμός 1.6 *Σαν σύνθετο μετασχηματισμό M, ορίζουμε τον μετασχηματισμό που προκύπτει από την διαδοχική εφαρμογή των βασικών μετασχηματισμών $M_1, \cdots, M_n$ Όπου M_i ο μετασχηματισμός που εφαρμόστηκε στο i-στο βήμα. Τότε*

$$M = \prod_{i=1}^{n} M_{n-i+1} = M_n * M_{n-1} * \cdots * M_2 * M_1$$

Παράδειγμα 1.1 *Έστω ότι θέλουμε να στρέψουμε κατά γωνία θ ένα αντικείμενο, όχι ως προς την αρχή των αξόνων αλλά ως προς ένα δεδομένο σημείο $P = (h, k)$. Για να βρούμε τον πίνακα αυτού του μετασχηματισμού θα χρειαστούμε 3 βήματα.*

Βήμα 1: Θα μεταφέρουμε το σημείο $P = (h, k)$ κατά διάνυσμα $\vec{v_1} = (-h, -k)$ έτσι ώστε αυτό να βρεθεί στην αρχή των αξόνων. Αυτό γίνεται με τον πίνακα μετασχηματισμού

$$T_{v1} = \begin{bmatrix} 1 & 0 & -h \\ 0 & 1 & -k \\ 0 & 0 & 1 \end{bmatrix}$$

Βήμα 2: Στη συνέχεια θα εφαρμόσουμε στροφή ως προς την αρχή των αξόνων κατά γωνία θ. Αυτό γίνεται με τον πίνακα μετασχηματισμού

$$R_\theta = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

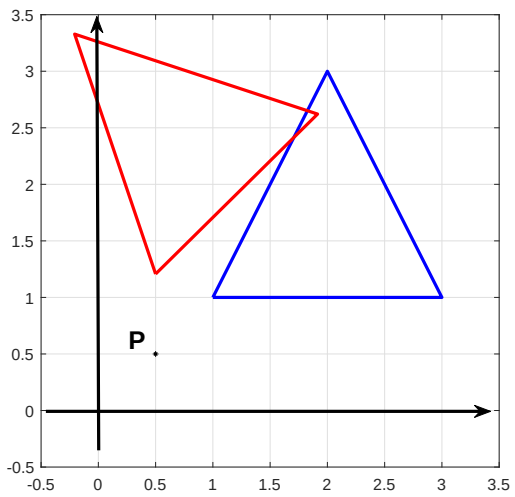
Βήμα 3: Τέλος θα μεταφέρουμε ξανά το σημείο $P = (h, k)$ κατά διάνυσμα $\vec{v_2} = (h, k)$ έτσι ώστε να βρεθεί στην αρχική του θέση. Αυτό γίνεται με τον πίνακα μετασχηματισμού

$$T_{v2} = \begin{bmatrix} 1 & 0 & h \\ 0 & 1 & k \\ 0 & 0 & 1 \end{bmatrix}$$

Ο ζητούμενος πίνακας προκύπτει ως εξής:

$$R_{P\theta} = T_{v2} * R_\theta * T_{v1} = \begin{bmatrix} 1 & 0 & h \\ 0 & 1 & k \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -h \\ 0 & 1 & -k \\ 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} cos\theta & -sin\theta & h - hcos\theta + ksin\theta \\ sin\theta & cos\theta & k - hsin\theta - kcos\theta \\ 0 & 0 & 1 \end{bmatrix}$$



Σχήμα 1.8: Στροφή 45°γύρω από το σημείο $P = (0.5, 0.5)$, με μπλε χρώμα το αρχικό σχήμα

1.3 Προβολές

Τα μοντέλα που καλείται κάποιος να διαχειριστεί στα γραφικά, είναι στη πλειονότητά τους τρισδιάστατα. Καλούμαστε λοιπόν να τα αποτυπώσουμε σε δισδιάστατες οθόνες ή σε άλλες μορφές εξόδου. Για να το πετύχουμε αυτό εφαρμόζουμε μια κατηγορία συναρτήσεων που παίρνουν σημεία του $\mathbb{R}^3$ και τα μετατρέπουν σε σημεία του $\mathbb{R}^2$.

Δύο είναι οι κύριες κατηγορίες που μας απασχολούν, και ο διαχωρισμός τους γίνεται με βάση της απόσταση του κέντρου προβολής από το επίπεδο προβολής.

1. **Προοπτική προβολή**, για πεπερασμένη απόσταση.
2. **Παράλληλη προβολή**, για άπειρη απόσταση.

1.3.1 Προοπτική προβολή

Στην προοπτική ή αλλιώς κεντρική προβολή, όπως αναφέραμε, το κέντρο προβολής **K** βρίσκεται σε πεπερασμένη απόσταση από το επίπεδο προβολής **Π** . Αν θέλουμε να προβάλουμε το σημείο **P** στο **Π** αρκεί να φέρουμε την ευθεία KP και να βρούμε το σημείο που τέμνει αυτή το **Π**. Αυτή θα είναι η εικόνα του **P** στο **Π**.

Ο προοπτικός μετασχηματισμός δεν είναι γραμμικός, αυτό δεν μας επιτρέπει να δουλέψουμε με ακριβώς τον ίδιο τρόπο με τους συσχετισμένους μετασχηματισμούς.

Έστω το κέντρο προβολής K βρίσκεται στο κέντρο των αξόνων K=(0,0,0) και το επίπεδο προβολής Π είναι κάθετο στον αρνητικό άξονα z σε απόσταση d από το K. Το σημείο $P = \begin{bmatrix} x & y & z \end{bmatrix}^T$ προβάλλεται στο $P' = \begin{bmatrix} x' & y' & d \end{bmatrix}^T$

Αν θεωρήσουμε τον πίνακα προοπτικής προβολής

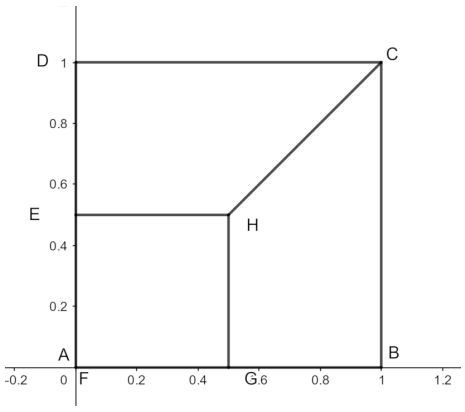
$$P_{pr} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

και τον εφαρμόσουμε στο $P = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T$ θα έχουμε

$$P' = P_{pr} * P = \begin{bmatrix} xd & yd & zd & z \end{bmatrix}^T$$

Παρατηρούμε ότι η ομογενή συντεταγμένη δεν είναι πια 1, διαιρώντας κάθε συντεταγμένη με z παίρνουμε την τελική τιμή για το P'.

$$P' = \begin{bmatrix} \frac{xd}{z} & \frac{yd}{z} & d & 1 \end{bmatrix}^T$$



Σχήμα 1.9: Προοπτική προβολή μοναδιαίου κύβου d=1

Γενικότερα μπορούμε να θεωρήσουμε προοπτικές απεικονίσεις από όλες τις δυνατές κατευθύνσεις και σε διαφορετικά επίπεδα, με τις αντίστοιχες συνθέσεις γεωμετρικών μετασχηματισμών και του πίνακα της προβολής μας.

1.3.2 Παράλληλη προβολή

Στη παράλληλη προβολή θεωρούμε ότι το κέντρο προβολής βρίσκεται σε άπειρη απόσταση από το επίπεδο προβολής. Για να την ορίσουμε χρειαζόμαστε το επίπεδο προβολής και να την κατεύθυνση προβολής. Αυτό έχει ως αποτέλεσμα να μπορούμε να διακρίνουμε δύο είδη παράλληλων προβολών. Αυτές που η κατεύθυνση προβολής είναι κάθετη στο επίπεδο προβολής και τις λέμε Ορθογραφικές και τις υπόλοιπες που τις λέμε πλάγιες.

Ορθογραφική προβολή

Συνήθως χρησιμοποιούμε για να την ορίσουμε , ένα από τα κύρια επίπεδα του χώρου σαν επίπεδο προβολής και σαν κατεύθυνση προβολής αυτή που έχει φορά προς το επίπεδο προβολής και είναι κάθετη σε αυτό. Για να ορίσουμε τον πίνακα προβολής θα θεωρήσουμε δίχως βλάβη της γενικότητας ότι το επίπεδο είναι το xy, αφού όποιο άλλο και να διαλέγαμε θα μπορούσαμε με κατάλληλους μετασχηματισμούς να καταλήξουμε στο ίδιο. Ο πίνακας που την ορίζει είναι ο

$$P_{ort} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Πλάγια προβολή

Όπως είδαμε στην περίπτωση πλάγιας προβολής, η κατεύθυνση προβολής δεν είναι κάθετη στο επίπεδο προβολής. Έστω ότι αυτή ορίζεται από το διάνυσμα $\vec{v} = (x_p, y_p, z_p)$ τότε ο πίνακας προβολής ορίζεται

$$P_{pl} = \begin{bmatrix} 1 & 0 & -\frac{x_p}{z_p} & 0 \\ 0 & 1 & -\frac{y_p}{z_p} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

1.4 Μετασχηματισμός παράστασης

Ένα από τα θεμελιώδη ζητήματα στο χώρο των υπολογιστών, είναι να καταφέρουμε να κάνουμε την μετάβαση από τον άπειρο πραγματικό μας κόσμο στον πεπερασμένο κόσμο ενός μηχανήματος. Το πρόβλημα που παρουσιάζεται όταν θέλουμε να εκφράσουμε κάποιον αριθμό του σύνολού των πραγματικών σε αριθμό μηχανής, εμφανίζεται και όταν θέλουμε να εμφανίσουμε ένα αντικείμενο από τον πραγματικό κόσμο στην οθόνη του υπολογιστή. Την διαδικασία για να μεταφέρουμε αντικείμενα από τον πραγματικό κόσμο, στον κόσμο της οθόνης μας, τη καθορίζει ο μετασχηματισμός παράστασης.

Πριν αναλύσουμε την διαδικασία, θα εισάγουμε για ευκολία τους ακόλουθους ορισμούς.

Ορισμός 1.7 *Πάνω στο επίπεδο που έχει ορίσει ο παρατηρητής και έχει προβάλει σε αυτό αντικείμενα του πραγματικού κόσμου, με την χρήση κατάλληλων μετασχηματισμών προβολής, ορίζουμε Παράθυρο Παρατηρητή (ΠΠ) το ορθογώνιο παραλληλόγραμμο, που καθορίζεται είτε από το κέντρο του και την απόστασή του από τις πλευρές του είτε από τις συντεταγμένες των κάτω αριστερά και άνω δεξιά κορυφών του.*

Ότι αντικείμενα πάνω στο επίπεδο προβολής βρίσκονται εντός του **ΠΠ** είτε πλήρως είτε μερικώς θα είναι και αυτά που θα μεταφερθούν τελικώς στην οθόνη μας. Με τη

βοήθεια μεθόδων αποκοπής θα καθοριστούν πλήρως τα μέρη των αντικειμένων που θα βρίσκονται εντός του **ΠΠ**.

Ορισμός 1.8 ***Παράθυρο Οθόνης (ΠΟ)*** *ορίζουμε το ορθογώνιο που καθορίζει τον χώρο της οθόνης μας που μπορούμε να προβάλλουμε τα γραφικά μας. Θεωρούμε ότι το κάτω αριστερό του άκρο είναι η αρχή των αξόνων και το άνω δεξιά άκρο καθορίζεται από την ανάλυση της οθόνης μας ανάλογα.*

Η μετατροπή από το ΠΠ στο ΠΟ γίνεται σε δύο βήματα.

Βήμα 1: Εφαρμόζουμε κατάλληλο μετασχηματισμό έτσι ώστε η κάτω αριστερή κορφή του ΠΠ να βρεθεί στο (0,0) και η άνω δεξιά στο (1,1). Αυτό γίνεται αν εφαρμόσουμε μεταφορά και μετά scaling.

Έστω ότι το ΠΠ ορίζεται από τις ακόλουθες συντεταγμένες των κορυφών του: (Wx_{min}, Wy_{min}) για την κάτω αριστερά και (Wx_{max}, Wy_{max}) για την άνω δεξιά. Και οι κατάλληλοι παράγοντες αλλαγής κλίμακας είναι οι :

$$WS_x = \frac{1}{Wx_{max} - Wx_{min}}$$

$$WS_y = \frac{1}{Wy_{max} - Wy_{min}}$$

Βήμα 2: Εφαρμόζουμε πάλι μετασχηματισμό αλλαγής κλίμακας έτσι ώστε η άνω δεξιά κορυφή του ορθογωνίου μας να ταυτιστεί με την αντίστοιχη κορυφή του ΠΟ αυτή την φορά. Αν η κορυφή αυτή έχει τις εξής συντεταγμένες $)(Vx_{max}, Vy_{max})$.

Οι παράγοντες αλλαγής κλίμακας θα είναι οι : $VS_x = Vx_{max}$ και $VS_y = Vy_{max}$

Τότε ο πίνακας μετασχηματισμού είναι ο

$$M_{view} = \begin{bmatrix} VS_x & 0 & 0 \\ 0 & VS_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} WS_x & 0 & 0 \\ 0 & WS_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -Wx_{min} \\ 0 & 1 & -Wy_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} VS_x * WS_x & 0 & -Wx_{min} \\ 0 & VS_y * WS_y & -Wy_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

1.5 Σχεδιασμός βασικών σχημάτων

Η συντριπτική πλειοψηφία των μεθόδων εξόδου που χρησιμοποιούμε στην εποχή μας, βασίζεται σε συσκευές που εφαρμόζουν μεθόδους Raster Scan. Με πιο απλά λόγια κάθε τι που θέλουμε να απεικονίσουμε το βλέπουμε σαν ένα είδος σύγχρονου ψηφιδωτού. Και τα αντικείμενα που θέλουμε να σχεδιάσουμε, είναι μια σύνθεση τέτοιων μικρών ψηφίδων, των pixel ή στα ελληνικά εικονοστοιχεία.

Τα pixel είναι μικρά τετράγωνα (όταν αναφερόμαστε σε οθόνες) και το πλήθος τους καθορίζει την ανάλυση της εικόνας μας. Μια εικόνα που έχει ανάλυση $m \times n$, αποτελείται από m γραμμές και n στήλες από τέτοια τετράγωνα. Για αυτό είναι και πολύ εύκολο να φανταστούμε ένα τέτοιο τετράγωνο σαν το στοιχείο ενός πίνακα, και τελικά να φανταζόμαστε κάθε εικόνα σαν έναν πίνακα.

Ορισμός 1.9 *Όταν αναφερόμαστε στη θέση ένος pixel με τις συντεταγμένες (x, y) θα εννοούμε ότι το κέντρο του τετραγώνου βρίσκεται σε αυτή τη θέση πάνω στο πλέγμα, που συνθέτει την οθόνη μας.*

Παρατήρηση 1.1 *Το **pixel** για τα γραφικά είναι ότι είναι το σημείο για τα μαθηματικά. Αν και πρακτικά έχει διαστάσεις, σε αντίθεση με το σημείο, θεωρούμε ότι είναι ισοδύναμα.*

Με αυτόν τον τρόπο απεικόνισης, χρειαζόμαστε να σκεφτούμε τρόπους για να μπορέσουμε να σχεδιάσουμε ακόμα και απλά σχήματα, όπως ένα ευθύγραμμό τμήμα ή έναν κύκλο. Καθώς οι τρόποι που γνωρίζουμε από τα μαθηματικά δεν μπορούν να βρουν άμεση εφαρμογή.

Οι αλγόριθμοι αυτοί θα πρέπει να επιλέγουν το καλύτερο δυνατό pixel έτσι ώστε το τελικό αποτέλεσμα να είναι όσο πιο κοντά στο σχήμα που θέλουμε. Αλλά ταυτόχρονα ο υπολογισμός των pixel να γίνεται γρήγορα και χωρίς να χρειάζονται πολλές πράξεις ώστε να έχουμε σφάλματα λόγω της αριθμητικής του υπολογιστή και να χρησιμοποιούν ακεραίους.

1.5.1 Αλγόριθμος Bresenham για ευθύγραμμο τμήμα

Το 1965 ο J.E. Bresenham παρουσίασε την πρώτη DDA μέθοδο (Digital Differential Analyzer) [5] για το σχεδιασμό ευθύγραμμου τμήματος. Εκεί παρουσίαζε τη μορφή που έχει ο αλγόριθμός του για ευθύγραμμα τμήματα που ανήκουν στο πρώτο οκταμόριο. Δηλαδή που η γωνία που σχηματίζουν με τον οριζόντιο άξονα είναι από 0°εώς 45°. Καθώς και τι μετατροπές που χρειάζονται να γίνουν για να μπορούμε να σχεδιάσουμε ευθύγραμμα τμήματα κάθε κλίσης.

Για το πρώτο οκταμόριο, η τεχνική του είναι σε κάθε βήμα να αυξάνουμε την x συντεταγμένη κατά 1 ενώ για την άλλη αποφασίζουμε ανάλογα με τη τιμή μιας μεταβλητής σφάλματος αν θα αυξήσουμε κατά 1 ή όχι.

Ο κώδικας που παραθέτω σε αυτό το σημείο δεν είναι ο αλγόριθμός που παρουσίασε ο Bresenham για το 1ο οκταμόριο, αλλά ένας ολοκληρωμένος αλγόριθμός που σχεδιάζει οποιοδήποτε ευθύγραμμό τμήμα ανεξαρτήτου κλίσης. Βασιζόμενος σε κάποιες παρατηρήσεις που είναι πολύ εύκολο να κάνουμε πάνω στην αρχική ιδέα.

Παρατήρηση 1.2 *Κάθε ευθύγραμμό τμήμα AB ορίζεται από τα σημεία $A = (x_1, y_1)$ και $B = (x_2, y_)$. Αν θεωρήσουμε ότι το σημείο A ταυτίζεται με την αρχή των αξόνων και χωρίσουμε το επίπεδο σε οχτώ ίσα τμήματα, το τμήμα το περιέχει το σημείο B καθορίζει και το οκταμόριο που ανήκει το ευθύγραμμο τμήμα.*

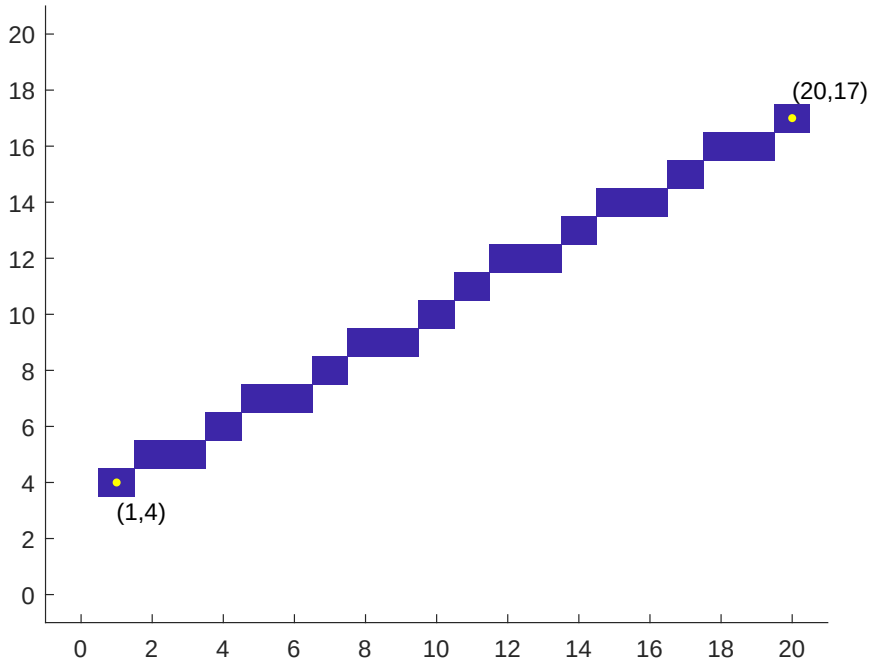
Παρατήρηση 1.3 *Στην αρχική ιδέα του Bresenham παίζει σημαντικό ρόλο η σειρά που θα δώσουμε τα σημεία A και B. Δηλαδή ποιο είναι η αρχή και ποιο το τέλος του τμήματος. Για αυτό έχουμε 8 διαφορετικές περιπτώσεις. Όμως αν απλά θέλουμε να σχεδιάσουμε ένα ευθύγραμμο τμήμα. Δεν μας ενδιαφέρει από ποια μεριά θα ξεκινήσουμε τον σχεδιασμό του. Αλλά διαλέγουμε πάντα σαν αρχή το σημείο για το οποίο η συντεταγμένη $y = min(y_1, y_2)$ τότε αντί για 8 περιπτώσεις έχουμε μόνο 4. Αφού τα οκταμόρια 5,6,7 και 8 ανάγονται στα 1,2,3 και 4 αντίστοιχα.*

Παρατήρηση 1.4 *Ο υπολογισμός των σημείων στο δεύτερο οκταμόριο γίνεται αν θεωρήσουμε ότι στη θέση των A και B, εισάγουμε τα σημεία $A_2 = (y_1, x_1)$ και $B_2 = (y_2, x_2)$ και αντί για να φωτίζουμε το pixel (x, y) που θα μας επιστρέφει ο αλγόριθμος εμείς φωτίζουμε το (y, x).*

Παρατήρηση 1.5 *Για τα οκταμόρια 3 και 4, αρκεί να υποθέσουμε ότι θέλουμε να σχεδιάσουμε το συμμετρικό ευθύγραμμο τμήμα ως προς τον άξονα y και να βάλουμε τον αλγόριθμο του Bresenham να υπολογίσει για αυτό τα σημεία, αλλά εμείς θα φωτίζουμε τα συμμετρικά των υπολογισμένων.*

Αφού λάβουμε υπόψιν μας αυτές τις παραδοχές, μπορούμε να υλοποιήσουμε σε μια συνάρτηση Matlab τον ολοκληρωμένο αλγόριθμό. Σε αυτό το σημείο πρέπει να διευκρινίσω ότι το υπολογιστικό πακέτο Matlab δεν παρέχει την δυνατότητα να αλληλοεπιδράσουμε απευθείας με τα εικονοστοιχεία της οθόνης μας. Για να δώσουμε

οπτικά το αποτέλεσμα θα σχεδιάσουμε με την βοήθεια της εντολής image τετράγωνα πλευράς 1 με κέντρο τις συντεταγμένες (x, y) του εκάστοτε προς φωτισμό pixel. Για αυτό στον κώδικά στη θέση που θα έπρεπε να είναι μια εντολή pixel(x,y) ή plot(x,y) εμείς βάζουμε image(x,y,c) η μεταβλητή c απλά έχει να κάνει με το Matlab και το χρώμα που θα βάλει στα τετράγωνα.



Σχήμα 1.10: Παράδειγμα ευθύγραμμου τμήματος, με άκρα τα σημεία (1,4) και (20,17)

Κώδικας Matlab 1.1

```
function bresline(P1,P2)
clf
hold on
dx = x2-x1;
dy = y2-y1;
d = 0; i = 0;
if dx == 0      % Vertical lines
    if y2 < y1
       t = y1; y1 = y2; y2 = t;
    end
    for y = y1:y2
        image(x1,y,0)
    end
elseif dy == 0 % Horizontal lines
    if x2 < x1
       t = x1; x1 = x2; x2 = t;
    end
    for x = x1:x2
        image(x,y1,0)
    end
elseif abs(dy) <= abs(dx) % 1st,4th,5th and 8th octant
```

```
    if x2 < x1
       t = x1; x1 = x2; x2 = t;
       t = y1; y1 = y2; y2 = t;
    end
    if (dy < 0 & dx > 0) | (dy > 0 & dx < 0)
        d = 1;
    end
    dx = abs(dx);
    dy = abs(dy);
    x = x1;
    y = y1;
    c1 = 2*dy;
    sfalma = 2*dy-dx ;
    c2 = sfalma-dx;
    while x <= x2
         if d == 0
             image(x,y,0)
         else
             image(x,y-2*i,0)
         end
         x = x+1;
         if sfalma < 0
            sfalma = sfalma+c1;
         else
            y = y+1;
            i = i+1;
            sfalma = sfalma+c2;
         end
    end
else % 2nd,3rd,6th and 7th octant
       if y2 < y1
         t = x1; x1 = x2; x2 = t;
         t = y1; y1 = y2; y2 = t;
       end
       if (dy < 0 & dx > 0) | (dy > 0 & dx < 0)
          d = 1;
       end
       dx = abs(dx);
       dy = abs(dy);
       x = x1;
       y = y1;
       c1 = 2*dx;
       sfalma = 2*dx-dy ;
       c2 = sfalma-dy;
       while y <= y2
             if d == 0
                image(x,y,0)
             else
                image(x-2*i,y,0)
             end
             y = y+1;
             if sfalma < 0
```

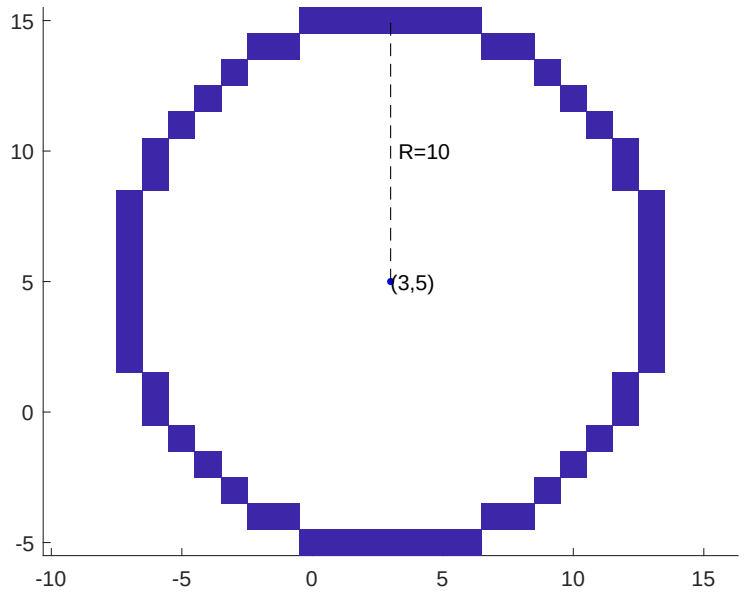
```
                sfalma = sfalma+c1;
            else
                i = i+1;
                x = x+1;
                sfalma = sfalma+c2;
            end
        end
    end
axis equal
```

Παρατήρηση 1.6 *παρατηρούμε, ο αλγόριθμος δεν χρειάζεται να κάνει πολλαπλασιασμούς και διαιρέσεις και γενικότερα να κάνει χρήση μη ακέραιων. Μόνο προσθέσεις, αφαιρέσεις και shift πράξεις. Αυτό τον καθιστά πολύ εύκολα υλοποιήσιμο σε επίπεδο hardware.*

1.5.2 Αλγόριθμος Bresenham για κύκλο

Στο σχεδιασμό κύκλου με κέντρο το (0,0) και ακτίνα R εκμεταλλευόμενοι την συμμετρικότητα του, έχουμε αρκετούς αλγόριθμούς που μπορούμε να χρησιμοποιήσουμε. Θα αναφέρουμε τη μέθοδο Bresenham [**Bres2**], όπου υπολογίζει τα pixel μόνο ενός από τα οκταμόρια που μπορούμε να τον χωρίσουμε, ειδικότερα του δεύτερου. Ξεκινάει από το σημείο $(0, R)$ και τερματίζει στο σημείο $(\frac{R}{\sqrt{2}}, \frac{R}{\sqrt{2}})$. Τα υπόλοιπα σημεία δεν χρειάζεται να υπολογιστούν ξεχωριστά, αφού προκύπτουν από 8 συμμετρίες που μπορούμε να θεωρήσουμε.

Για να σχεδιάσουμε κάποιο κύκλο με διαφορετικό κέντρο, υπολογίζουμε τα σημεία, σαν να έχει κέντρο το (0,0), και εφαρμόζουμε μετασχηματισμό μεταφοράς έτσι ώστε από το κέντρο (0,0) να βρεθούμε στο ζητούμενο. Στη πράξη βλέπουμε ότι αρκεί στις συντεταγμένες των σημείων που παράγει ο αλγόριθμός να προσθέσουμε τις αντίστοιχες συντεταγμένες του επιθυμητού κέντρου του κύκλου. Έτσι προκύπτει ο ακόλουθος αλγόριθμος υλοποιημένος σε Matlab.



Σχήμα 1.11: Παράδειγμα κύκλου, με κέντρο (3,5) και ακτίνα R = 10

Κώδικας Matlab 1.2

```
function brescircle(xc,yc,r)
x = 0; y = r;
sfalma = 3 - 2 * r;
hold on
while x <= y
    image(x+xc,y+yc,0)
    image(x+xc,-y+yc,0)
    image(-x+xc,y+yc,0)
    image(-x+xc,-y+yc,0)
    image(y+xc,x+yc,0)
    image(-y+xc,x+yc,0)
    image(y+xc,-x+yc,0)
    image(-y+xc,-x+yc,0)
    x = x + 1;
    if sfalma >= 0
        y = y - 1;
        sfalma = sfalma - 4 * y;
    end
    sfalma = sfalma + 4 * x + 2;
end
axis equal
```

1.5.3 Σχεδιασμός καμπυλών με Raster Scan μεθόδους

Την ίδια λογική που ακολουθούν οι αλγόριθμοι που είδαμε για ευθεία και κύκλο, μπορούμε να την εφαρμόσουμε και για τον σχεδιασμό καμπυλών σε μια οθόνη. Στην εργασία δεν θα μείνουμε παραπάνω στο πως σχεδιάζονται καμπύλες σε πλεγματικές οθόνες, αλλά κρίνω πρέπον να παραπέμψω σε δουλειές σαν τις [9] , [11], [12].

Κεφάλαιο 2

Καμπύλες Παρεμβολής

Όταν καλούμαστε να υλοποιήσουμε ένα σκηνικό με την χρήση γραφικών, θα χρειαστεί να σχεδιάσουμε διάφορα αντικείμενα δύο ή τριών διαστάσεων. Δεν αρκούν οι μέθοδοι σχεδιασμού των βασικών σχημάτων. Ειδικά αν θέλουμε το αποτέλεσμα να είναι πιο ρεαλιστικό.

Θα πρέπει να καταφύγουμε λοιπόν σε μαθηματικές μεθόδους για τον υπολογισμό κατάλληλων καμπυλών και επιφανειών, που με την σύνθεση τους θα μας δώσουν το επιθυμητό αποτέλεσμα. Φροντίζοντας πάντα να έχουμε όσο τον δυνατόν πιο αποτελεσματικούς αλγορίθμους.

2.1 Παραμετρικές Καμπύλες

Δύο είναι οι πιο διαδεδομένοι τρόποι για να εκφράσεις με την βοήθεια των μαθηματικών μια καμπύλή . Ο πρώτος είναι να ορίσουμε την καμπύλη μέσω της $F(x, y) = 0$, για παράδειγμα ο κύκλος ακτίνας 1 μπορεί να γραφεί $x^2 + y^2 - 1 = 0$.

Ο άλλος είναι να εκφράσουμε τις καμπύλες σε παραμετρική μορφή. Στη πράξη η απεικόνιση μιας καμπύλης με τον τρόπο αυτό, είναι σύνηθες στο χώρο των γραφικών.

Ορισμός 2.1 *Μια συνάρτηση $r : I \subseteq \mathbb{R} \to \mathbb{R}^n$ ονομάζεται παραμέτριση του $\mathbb{R}^n$.*

Ορισμός 2.2 *Ένα υποσύνολο Γ του $\mathbb{R}^3$ ονομάζεται παραμετρική καμπύλη του χώρου, όταν υπάρχει παραμέτρηση $r : I \subseteq \mathbb{R} \to \mathbb{R}^3$, τέτοια ώστε $\Gamma = \{r(t) : t \in I\}$.*

Οι παραμετρικές καμπύλες που συναντώνται στα γραφικά, βασίζονται κυρίως στα πολυώνυμα. Αυτό γιατί είναι απλές συναρτήσεις, εύκολο να υπολογιστούν με στοιχειώδης αριθμητικούς υπολογισμούς, καθώς επίσης να βρούμε τις παραγώγους τους.

Ειδικότερα δε, προτιμούμε να δουλεύουμε με τρίτου βαθμού. Αυτό γιατί καθώς ανεβαίνει ο βαθμός του πολυωνύμου και η πολυπλοκότητα του υπολογισμού τους αυξάνεται. Ενώ τα μικρότερου βαθμού αναπαριστούν είτε ευθύγραμμά τμήματα (πρώτου βαθμού), είτε παραβολές (δευτέρου βαθμού). Προφανώς, αν η καμπύλη που θέλουμε είναι κάποια από τις δύο δεν χρειάζεται να καταφύγουμε σε μεγαλύτερο βαθμό.

2.2 Καμπύλες Παρεμβολής

Ένας τρόπος υπολογισμού κατάλληλων πολυωνύμων, που θα μας δώσουν τη δυνατότητα να σχεδιάσουμε μια επιθυμητή καμπύλη. Είναι να βρούμε το πολυώνυμο παρεμβολής για δεδομένα σημεία. Για αυτό οι καμπύλες που προκύπτουν από αυτά, ονομάζονται καμπύλες παρεμβολής.

Ορισμός 2.3 *Έστω $P_1, P_2, \cdots, P_n, P_{n+1}$ σημεία, τότε μπορούμε να κατασκευάσουμε ένα πολυώνυμο που παρεμβάλλεται από τα $n+1$ αυτά σημεία. Το πολυώνυμο αυτό, το ονομάζουμε πολυώνυμο παρεμβολής. Μάλιστα είναι μοναδικό.*

Θεώρημα 2.1 (Παρεμβολή Lagrange) *[18] Έστω $x_1, x_2, \cdots, x_n, x_{n+1} \in \mathbb{R}$ ανά δυο διαφορετικά μεταξύ τους σημεία και $y_1, y_2, \cdots, y_n, y_{n+1} \in \mathbb{R}$. Τότε υπάρχει ακριβώς ένα πολυώνυμο $p \in \mathbb{P}_n$ τέτοιο ώστε*

$$p(x_i) = y_i, i = 1, 2, \cdots, n, n+1$$

Το πολυώνυμο p του θεωρήματος 2.1 μπορούμε να το υπολογίσουμε με διάφορες μεθόδους. Δύο είναι οι πιο γνωστές:

1. Η μέθοδος Lagrange
2. Η μέθοδος Newton

2.2.1 Μέθοδος Lagrange

Δεδομένων $\kappa + 1$ σημείων $(x_0, y_0), (x_1, y_1), \cdots, (x_\kappa, y_\kappa)$ όπου $\forall x_i \neq x_j$, $\forall i \neq j$ με $i, j \in \{0, \cdots, \kappa\}$. Το πολυώνυμο παρεμβολής με τη μέθοδο Lagrange παίρνει στην θέση x την τιμή :

$$L(x) = \sum_{j=0}^{\kappa} \left(y_j \prod_{\substack{0 \leq m \leq \kappa \\ m \neq j}} \frac{x - x_m}{x_j - x_m} \right) \tag{2.1}$$

Η συγκεκριμένη μέθοδος όμως, δεν βρίσκει εφαρμογή στην πράξη αφού παρουσιάζει διάφορα προβλήματα, μερικά από τα οποία αναφέρονται στην παρατήρηση που ακολουθεί.

Παρατήρηση 2.1

1. *Η αλλαγή ενός σημείου μας μας αναγκάζει να υπολογίσουμε από την αρχή το πολυώνυμο.*
2. *Επειδή είναι της μορφής $y = f(x)$ εξαρτάται άμεσα από το σύστημα αξόνων συντεταγμένων. Έτσι στην περίπτωση που εφαρμόσουμε κάποιον μετασχηματισμό αξόνων στην εικόνα μας. Η καμπύλη ενδέχεται να αλλάξει μορφή, το οποίο είναι και κάτι που δεν θέλουμε. [17]*
3. *Στον υπολογισμό του πολυωνύμου με αυτή την μέθοδο, υπεισέρχονται πολλά αριθμητικά σφάλματα.*
4. *Το πολυώνυμο παρεμβολής, είναι ανεξάρτητο της σειράς με την οποία θα δώσουμε τα σημεία παρεμβολής. Η γραφική παράσταση του θα ξεκινάει πάντα από το σημείο παρεμβολής με την μικρότερη x-συντεταγμένη και θα καταλήγει στο σημείο με την μεγαλύτερη.*

Για να δούμε στην πράξη τα όποια της μειονεκτήματα, με τη βοήθεια του Matlab θα υλοποιήσουμε μια συνάρτηση, που θα υπολογίζει την τιμή του πολυωνύμου Lagrange στην θέση x, το οποίο μπορεί να είναι είτε μια τιμή είτε ένα διάνυσμα με τόσες θέσεις όσες οι διαφορετικές τιμές του πολυωνύμου που θέλουμε να υπολογίσουμε, για n σημεία παρεμβολής που δίνονται σαν ένας $2 \times n$ πίνακας $IntP$.

Κώδικας Matlab 2.1

```
function lpf = lagrange(IntP,x)
n = size(IntP,2);
lpf = 0;
    for i=1:n
        nm = 1;
        dn = 1;
        for j=1:n
            if j~=i
                nm = nm.*(x-IntP(1,j));
                dn = dn*(IntP(1,i)-IntP(1,j));
            end
        end
        L = nm/dn;
        lpf = lpf + L * IntP(2,i);
    end
```

Παράδειγμα 2.1 *Έστω τα σημεία παρεμβολής:*

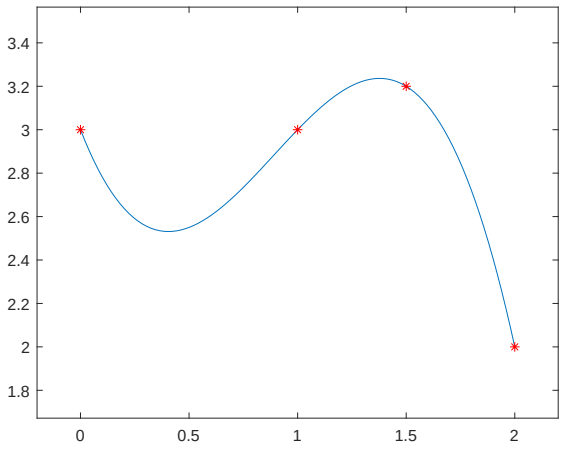
$$P_0(0,3), P_1(1,3), P_2(1.5,3.2), P_3(2,2).$$

Ζητάμε να υπολογίσουμε τις τιμές του πολυωνύμου παρεμβολής για τις τιμές της διαμέρισης του $[0,2]$ *με βήμα* 0.25.

Τότε με την βοήθεια της συνάρτησης που κατασκευάσαμε θα πάρουμε τα εξής σημεία:

$(0,3.0000), (0.2500,2.5906), (0.5000,2.5500), (0.7500,2.7344), (1.0000,3.0000),$
$(1.2500,3.2031), (1.5000,3.2000), (1.7500,2.8469), (2.0000,2.0000)$

Παράδειγμα 2.2 *Μπορούμε να σχεδιάσουμε την αντίστοιχη καμπύλη που προκύπτει αν υπολογίσουμε την ομοιόμορφη διαμέριση 100 τιμών του ίδιου διαστήματος.*



Σχήμα 2.1: Η καμπύλη του **Παραδείγματος 2.2** για 100 σημεία, με κόκκινο τα σημεία παρεμβολής.

2.2.2 Μέθοδος Newton

Το πολυώνυμο παρεμβολής με τη μέθοδο Newton [18]

$$p(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + \cdots + a_n(x-x_0)(x-x_1)\cdots(x-x_{n-1}) \quad (2.2)$$

όπου οι συντελεστές $a_i, i = 0, \cdots, n$ υπολογίζονται αναδρομικά:

$$\begin{aligned} p(x_0) = y_0 &\rightarrow a_0 = y_0 \\ p(x_1) = y_1 &\rightarrow a_1 = \frac{y_1 - a_0}{x_1 - x_0} \\ p(x_2) = y_2 &\rightarrow a_2 = \left(\frac{y_2 - a_0}{x_2 - x_0} - a_1\right)/(x_2 - x_1) \end{aligned} \quad (2.3)$$

κ.ο.κ.

Τους συντελεστές a_i μπορούμε να τους υπολογίσουμε με την χρήση διαιρεμένων διαφορών.

Ορισμός 2.4 *Έστω $f \in C[a,b]$, $x_0, x_1, \cdots \in [a,b]$,$x_i \neq x_j$ για $i \neq j$. Τότε ορίζουμε επαγωγικά ως προς i:*

$$\begin{aligned} &\Delta^0(x_0)(f) := f(x_0) \\ &\Delta^i(x_0, \cdots x_i)(f) := \frac{\Delta^i(x1, \cdots, x_i)(f) - \Delta^{i-1}(x_0, \cdots, x_{i-1}(f)}{x_i - x_0}, i \geq 1. \end{aligned} \quad (2.4)$$

Ο αριθμός $\Delta^i(x_0, \cdots x_i)(f)$ λέγεται διαιρεμένη διαφορά τάξεως i της f ως προς τα σημεία $x_0, \cdots, x_i$. [18]

Βλέπουμε λοιπόν ότι $a_i = \Delta^i(x_0, \cdots x_i)(f), i = 0, \cdots, n$. Και επομένως μπορούμε να γράψουμε τον ακόλουθο κώδικα Matlab όπου αφού κατασκευάσουμε τον πίνακα διαφορών που αντιστοιχεί στα σημεία παρεμβολής, που δίνονται σαν ένας $2 \times n$ πίνακας $IntP$, στην συνέχεια μπορούμε να υπολογίσουμε τις τιμές του πολυωνύμου για το x, το οποίο μπορεί να είναι είτε μια τιμή είτε ένα διάνυσμα με τόσες θέσεις όσες οι διαφορετικές τιμές του πολυωνύμου που ζητάμε.

Κώδικας Matlab 2.2

```
function npf = newton(IntP,x)
n = size(IntP,2);
DN = zeros(n,n);
DN(:,1) = IntP(2,:);
for j = 2:n
    for i = 1:n-j+1
        DN(i,j) = (DN(i+1,j-1)-DN(i,j-1))/...
            (IntP(1,i+j-1)-IntP(1,i));
    end
end
m = length(x);
npf = zeros(1,m);
a = DN(1,:);
for i = 1:m
    xd = x(i)-IntP(1,:);
    npf(i) = a(1);
    for j = 1:n-1
        npf(i) = npf(i) + a(j+1)*prod(xd(1:j));
    end
end
```

Παρατήρηση 2.2 *Η μέθοδος Newton οδηγεί στο ίδιο πολυώνυμο παρεμβολής, αφού αυτό είναι μοναδικό όπως προκύπτει και από το θεώρημα 2.1.*

Αυτό μπορούμε να το επαληθεύσουμε αν καλέσουμε την συνάρτηση newton με τα αντίστοιχα δεδομένα που έχουμε στο **Παράδειγμα 2.1**.

Σε αυτή την περίπτωση, θα πάρουμε ακριβώς τα ίδια 9 σημεία, που πήραμε και στο παράδειγμα.

$$(0, 3.0000), (0.2500, 2.5906), (0.5000, 2.5500), (0.7500, 2.7344), (1.0000, 3.0000),$$
$$(1.2500, 3.2031), (1.5000, 3.2000), (1.7500, 2.8469), (2.0000, 2.0000)$$

Εύλογα θα αναρωτηθούμε, ποιος ο λόγος που καταφεύγουμε σε αυτό τον τρόπο; Η απάντησή είναι, ότι ο τρόπος με τον οποίο αυτό υπολογίζεται θεραπεύει κάποιες από τις αδυναμίες της μεθόδου Lagrange.

Για παράδειγμα μπορεί με ευκολία να εκφραστεί σε παραμετρική μορφή. Το οποίο μας είναι εξαιρετικά χρήσιμο στην περίπτωση που θέλουμε να σχεδιάσουμε με ευκολία, μια καμπύλη που θα παρεμβάλει σημεία στον χώρο. Κάτι που δεν μπορούμε να πετύχουμε είτε με την μέθοδο Lagrange είτε με την μη παραμετρική αναπαράσταση της μεθόδου Newton όπου μένουμε στο επίπεδο.

2.2.3 Παραμετρική Μέθοδος Newton

Έστω τα σημεία $P_0 = (x_0, y_0, z_0), P_1 = (x_1, y_1, z_1), \cdots, P_n(x_n, y_n, z_n)$ και έστω οι κόμβοι

$$t_0 = 0 < t_1 < \cdots < t_{n-1} < t_n = 1$$

Τότε κάθε σημείο πάνω στην καμπύλη που εκφράζει το πολυώνυμο παρεμβολής μπορεί να υπολογιστεί μέσω του τύπου

$$P(t) = \sum_{i=0}^{n} N_i(t) a_i \tag{2.5}$$

Όπου τα $N_i(t)$ εξαρτώνται μόνο από τους κόμβους μας και όχι από τα σημεία που έχουμε. $N_0 = 1, N_i = (t - t_0)(t - t_1) \cdots (t - t_{i-1})$ για $i = 1, \cdots, n$. Και για τον υπολογισμό των συντελεστών a_i έχουμε τις ακόλουθες εξισώσεις, που προκύπτουν από τον τύπο πολυωνύμου.

$$\begin{aligned}
&P_0 = P(t_0) = a_0 \\
&P_1 = P(t_1) = a_0 + a_1(t_1 - t0) \\
&P_2 = P(t_2) = a_0 + a_1(t_1 - t0) + a_2(t_2 - t_0)(t_2 - t_1) \\
&\vdots \\
&P_n = P(t_n) = a_0 + \cdots
\end{aligned} \tag{2.6}$$

και όπως ήδη έχουμε αναφέρει τα a_i μπορούν να υπολογιστούν αναδρομικά.

$$\begin{aligned}
a_0 &= P_0 \\
a_1 &= \frac{P_1 - P_0}{t_1 - t_0} \\
a_2 &= \frac{P_2 - P_0 - \frac{(P_1 - P_0)(t_2 - t_0)}{(t_1 - t_0)}}{(t_2 - t_0)(t_2 - t_1)} = \frac{\frac{P_2 - P_1}{t_2 - t_1} - \frac{P_1 - P_0}{t_1 - t_0}}{t_2 - t_0} \\
&\vdots
\end{aligned} \tag{2.7}$$

Παρατήρηση 2.3 *Όπως είναι εύκολα αντιληπτό, όταν χρησιμοποιούμε την παραμετρική μέθοδο Newton οι συντελεστές $a_i \notin \mathbb{R}$ αλλά έχουν την ίδια διάσταση με τα σημεία παρεμβολής.*

Ορισμός 2.5 *Η διαιρεμένη διαφορά των κόμβων t_i, t_j συμβολίζεται με $[t_i t_j]$ και ορίζεται ως $[t_i t_j] := \frac{P_i - P_j}{t_i - t_j}$*

Και μπορούμε να εκφράσουμε τους συντελεστές στην ακόλουθη μορφή

$$\begin{aligned} a_0 &= P_0 \\ a_1 &= \frac{P_1 - P_0}{t_1 - t_0} = [t_1 t_0] \\ a_2 &= [t_2 t_1 t_0] = \frac{[t_2 t_1] - [t_1 t_0]}{t_2 - t_0} \\ a_3 &= [t_3 t_2 t_1 t_0] = \frac{[t_3 t_2 t_1] - [t_2 t_1 t_0]}{t_3 - t_0} \\ &\vdots \\ a_n &= [t_n \cdots t_1 t_0] = \frac{[t_n \cdots t_1] - [t_{n-1} \cdots t_0]}{t_n - t_0} \end{aligned} \tag{2.8}$$

Με μερικές αλλαγές στον κώδικα της συνάρτησης που δημιουργήσαμε για την μέθοδο Newton παίρνουμε την συνάρτηση που υλοποιεί την παραμετρική μορφή της. Στην περίπτωσή αυτή θα πρέπει να δώσουμε επιπλέον σαν είσοδο τους κόμβους tn της παραμέτρισης μας, στους οποίους αντιστοιχούν τα σημεία παρεμβολής. Επίσης αυτή την φορά, αντί για τιμές x δίνουμε τιμές $t \in [0, 1]$. Σε αντίθεση με την προηγούμενη συνάρτηση που μας επέστρεφε ένα διάνυσμα με τις τιμές $p(x)$, εδώ επιστρέφει έναν πίνακα που το πλήθος των γραμμών του εξαρτάται από την διάσταση των σημείων παρεμβολής.

Κώδικας Matlab 2.3

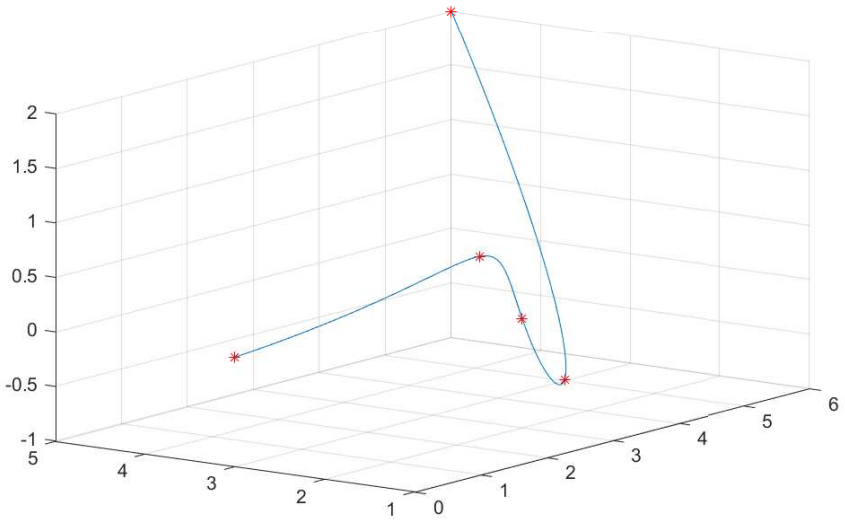
```
function npf = newton_p(t,IntP,tn)
[dim,n] = size(IntP);
m = length(t);
npf = zeros(dim,m);
for d = 1:dim
    DN = zeros(n,n);
    DN(:,1) = IntP(d,:);
    for j = 2:n
        for i = 1:n-j+1
            DN(i,j) = (DN(i+1,j-1)-DN(i,j-1))/...
                (tn(i+j-1)-tn(i));
        end
    end
    a = DN(1,:);
    for i = 1:m
        td = t(i)-tn;
        npf(d,i) = a(1);
        for j = 1:n-1
            npf(d,i) = npf(d,i) + a(j+1)*prod(td(1:j));
        end
    end
end
```

Από την στιγμή που εκφράζουμε σε παραμετρική μορφή την μέθοδο Newton, άμεσα αποκτάμε κάποια πλεονεκτήματα που δεν είχαμε προηγουμένως. Έχουμε αναφέρει ότι μπορούμε να παρεμβάλουμε σημεία στον $\mathbb{R}^3$, ή σε χώρους μεγαλύτερης διάστασης.

Παράδειγμα 2.3 *Έστω τα σημεία παρεμβολής:*

$$P_0(0,3,0), P_1(1,1,1), P_2(3,2,0), P_3(5,3,-1), P_4(6,5,2)$$

και έστω οι αντίστοιχοι κόμβοι $t_i = \frac{i}{4}, i = 0, \cdots, n$. *Τότε όπως μπορούμε να δούμε στο γράφημα που ακολουθεί, καταφέρνουμε να σχεδιάσουμε μια καμπύλη στον χώρο, όπου διέρχεται από τα σημεία παρεμβολής.*



Σχήμα 2.2: Η καμπύλη του **Παραδείγματος 2.3**, με κόκκινο τα σημεία παρεμβολής.

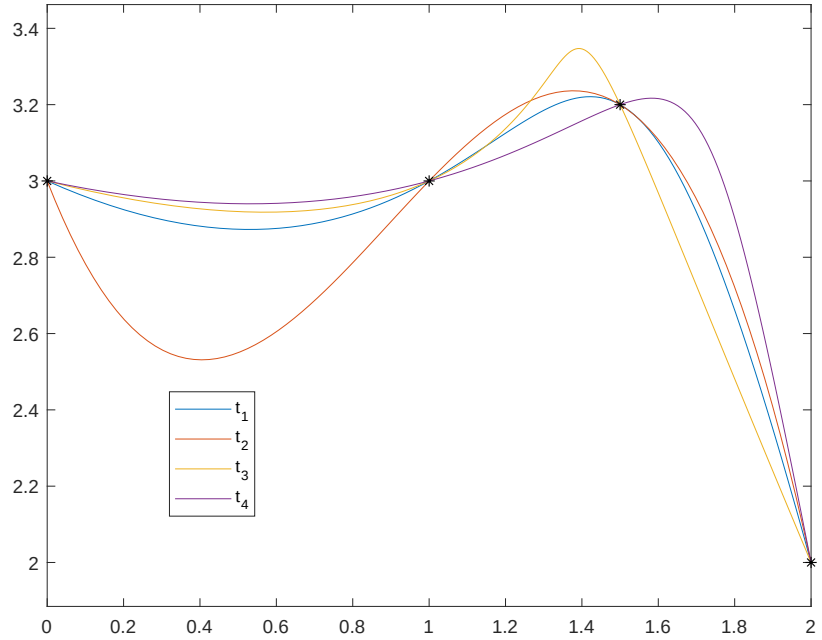
Το μεγαλύτερο πλεονέκτημα που έχουμε εκφράζοντας το πολυώνυμο σε παραμετρική μορφή, είναι ότι εκτός από τα σημεία παρεμβολής εξαρτάται και από την επιλογή των κόμβων. Κάτι που μας επιτρέπει να πάρουμε διαφορετικές καμπύλες που διέρχονται από τα ίδια σημεία, απλά και μόνο αλλάζοντας την θέση των κόμβων στο διάστημα $[0,1]$.

Παράδειγμα 2.4 *Έστω τα σημεία παρεμβολής :*

$$P_0(0,3), P_1(1,3), P_2(1.5,3.2), P_3(2,2).$$

Έστω και οι διαμερίσεις του διαστήματος $[0,1]$*:*

1. $t_1 \in \{0, 1/3, 2/3, 1\}$
2. $t_2 \in \{0, 0.5, 0.75, 1\}$
3. $t_3 \in \{0, 0.25, 0.75, 1\}$
4. $t_4 \in \{0, 0.25, 0.5, 1\}$



Σχήμα 2.3: Οι καμπύλες του **Παραδείγματος 2.4**. Με μαύρο αστερίσκο τα σημεία παρεμβολής.

Παρατήρηση 2.4 *Το **Παράδειγμα 2.4** μας κάνει όμως να αναρωτηθούμε, αφού στο **Θεώρημα 2.1** εμείς αναφέραμε ότι το πολυώνυμο που προκύπτει από δεδομένο πλήθος σημείων παρεμβολής, είναι μοναδικό. Πως γίνεται από το ίδιο σύνολο σημείων, να έχουμε στο παράδειγμα μας 4 διαφορετικές πολυωνυμικές καμπύλες που να διέρχονται από τα ίδια σημεία;*

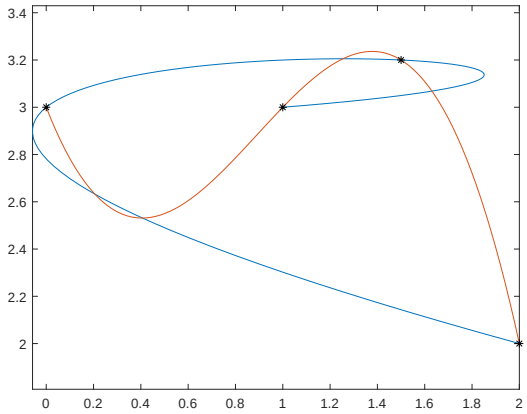
Η απάντηση είναι απλή. Δεν έχουμε τα ίδια σημεία παρεμβολής. Από την στιγμή που προσθέσαμε τους κόμβους της παραμέτρισης, αλλάξαμε αυτόματα και τα σημεία παρεμβολής. Τα σημεία μας δεν ανήκουν πια στον $\mathbb{R}^2$ αλλά στον $\mathbb{R}^2 \times [0,1]$ με επιπλέον συντεταγμένη την τιμή του t. Και έτσι η παραμετρική μέθοδος Newton εξακολουθεί να πληροί την μοναδικότητα που αναφέρει το θεώρημα.

Παρατήρηση 2.5 *Η επιπλέον πληροφορία που έχουμε μέσω των κόμβων, μας δίνει την δυνατότητα, να καταργήσουμε το πρόβλημα που αναφέραμε σαν τέταρτο, στην **Παρατήρηση 2.1**. Δηλαδή ότι το πολυώνυμο παρεμβολής, είναι ανεξάρτητο της σειράς με την οποία θα δώσουμε τα σημεία παρεμβολής. Και ότι η γραφική παράσταση του θα ξεκινάει πάντα από το σημείο παρεμβολής με την μικρότερη x-συντεταγμένη και θα καταλήγει στο σημείο με την μεγαλύτερη.*

Την σειρά με την οποία θα διέλθει η καμπύλη μας από τα σημεία παρεμβολής, δεν θα την καθορίζει πια η x-συντεταγμένη, αλλά οι κόμβοι. Η καμπύλη θα ξεκινάει από το σημείο που αντιστοιχεί στον κόμβο με τιμή $t = 0$ και θα καταλήγει σε αυτό με κόμβο $t = 1$.

Παράδειγμα 2.5 *Έστω οι κόμβοι $t \in \{0, 1/3, 2/3, 1\}$, με σημεία παρεμβολής*

1. $P_0(0,3), P_1(1,3), P_2(1.5,3.2), P_3(2,2)$.
2. $P_0(1,3), P_1(1.5,3.2), P_2(0,3), P_3(2,2)$.



Σχήμα 2.4: Με κόκκινο η καμπύλη για την πρώτη διάταξη των σημείων, ενώ με μπλε η αντίστοιχη καμπύλη για την δεύτερη διάταξη. Με μαύρο είναι τα σημεία παρεμβολής.

Παρατήρηση 2.6 *Στο δεύτερο σημείο της* ***Παρατήρησης 2.1****, γίνεται αναφορά ότι η καμπύλη αν της εφαρμόσουμε μετασχηματισμό αξόνων ενδέχεται να αλλάξει μορφή η καμπύλη μας. Αυτό σημαίνει ότι αν εφαρμόσουμε στα σημεία παρεμβολής κάποιο μετασχηματισμό και τα καινούρια σημεία που θα προκύψουν, τα χρησιμοποιήσουμε σαν σημεία παρεμβολής για την Lagrange, η καμπύλη που θα προκύψει ενδέχεται να αλλάξει μορφή. Κάτι που αν εφαρμόσουμε την παραμετρική Newton δεν θα γίνει.*

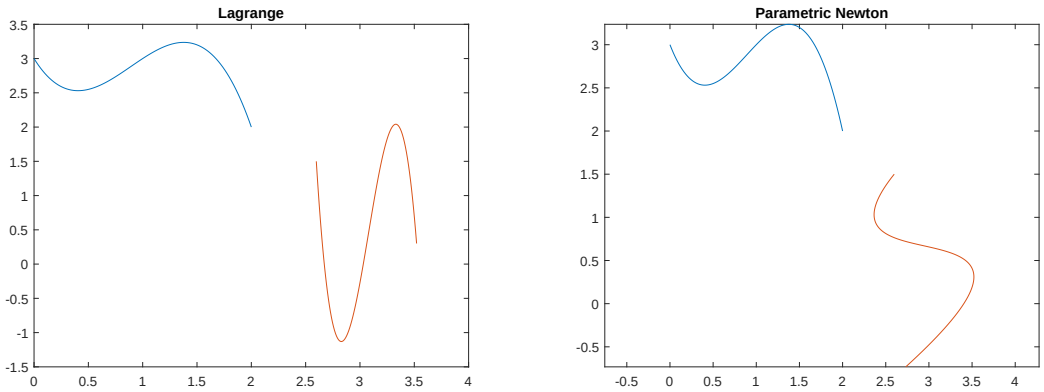
Παράδειγμα 2.6 *Χρησιμοποιούμε τα σημεία παρεμβολής:*

$$P_0(0,3), P_1(1,3), P_2(1.5,3.2), P_3(2,2).$$

και τους εφαρμόζουμε στροφή κατά γωνία $\theta = pi/3$. Η καινούρια τετράδα σημείων που προκύπτει είναι:

$$P_0(2.5981,1.5), P_1(3.0981,0.634), P_2(3.5213,0.301), P_3(2.731,-0.7321).$$

Για αυτά υπολογίζουμε εκ νέου το πολυώνυμο παρεμβολής με την μορφή Lagrange αλλά και με την παραμετρική μορφή της μεθόδου Newton. Το αποτέλεσμα το βλέπουμε στο σχήμα που ακολουθεί.



Σχήμα 2.5: Αποτελέσματα για το **Παράδειγμα 2.6**. Με μπλε η αρχική καμπύλη, με κόκκινο αυτή μετά την στροφή.

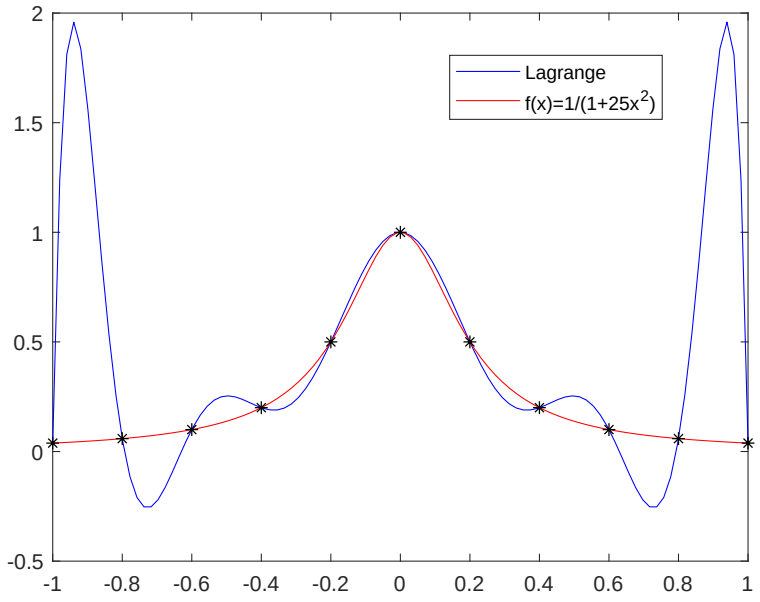
Γίνεται αντιληπτό από την εικόνα, ότι όταν χρησιμοποιήσαμε τα μετασχηματισμένα σημεία παρεμβολής που παράγει η μέθοδος Lagrange,είχαμε αλλοιώσεις , κάτι που δεν συνέβη στην άλλη περίπτωση, που χρησιμοποιήσαμε την παραμετρική εκδοχή της μεθόδου Newton.

2.2.4 Πολυώνυμο Παρεμβολής Μεγάλου βαθμού

Όπως είδαμε και στο θεώρημα 2.1, ο βαθμός του πολυωνύμου παρεμβολής εξαρτάται από το πλήθος των σημείων παρεμβολής. Για την ακρίβεια αν έχουμε n σημεία, το πολυώνυμό θα είναι βαθμού $n-1$.

Στα γραφικά και όχι μόνο, θα χρειαστεί να σχεδιάσουμε καμπύλες που θα παρεμβάλλονται από μεγάλο το πλήθος σημεία. Αυτό θα έχει σαν αποτέλεσμα να πρέπει να κατασκευάσουμε και αντίστοιχου βαθμού πολυώνυμα. Δυστυχώς όσο ανεβαίνει ο βαθμός, τόσο εμφανίζονται προβλήματα στην καμπύλη που παράγεται.

Παράδειγμα 2.7 *Έστω ότι θέλουμε να σχεδιάσουμε την καμπύλη που παρεμβάλει τα σημεία που οι συντεταγμένες x προκύπτουν από την ομοιόμορφη διαμέριση του διαστήματος $[-1,1]$ με βήμα 0.2. Ενώ οι συντεταγμένες y δίνονται από τον τύπο $y=\frac{1}{1+25x^2}$.*



Σχήμα 2.6: Με μπλέ η καμπύλη παρεμβολής για το **Παράδειγμα 2.7**. Με κόκκινο η καμπύλη που προκύπτει με από τον τύπο της f του παραδείγματος. Με μαύρο είναι τα σημεία παρεμβολής.

Όπως βλέπουμε από το σχήμα 2.2.4, ενώ η γραφική παράσταση που δίνει η $f(x)=\frac{1}{1+25x^2}$ είναι μια αρκετά ομαλή καμπύλη, αυτή που προκύπτει από το πολυώνυμο παρεμβολής με τη μέθοδο Lagrange εμφανίζει πολλές ταλαντώσεις στα άκρα. Το οποίο δεν είναι επιθυμητό όταν προσπαθούμε να υλοποιήσουμε ένα αντικείμενο στα γραφικά. Το ίδιο θα συμβεί και αν υπολογίσουμε το πολυώνυμό με την μέθοδο Newton αφού έχουμε το ίδιο πολυώνυμό. Καταλήγουμε λοιπόν στο συμπέρασμα, ότι θα πρέπει να μείνουμε σε μικρότερου βαθμού πολυώνυμα. Κατά προτίμηση μέχρι τρίτου.

Δεν γίνεται όμως να αναπαραστήσουμε μια οποιαδήποτε καμπύλη, με ένα πολυώνυμο τρίτου βαθμού έτσι ώστε και να έχουμε εύκολο τρόπο υπολογισμού της, και να επιτύχουμε το επιθυμητό αποτέλεσμα. Για αυτό η λύση είναι να περιγράψουμε κατά τμήματα την καμπύλη, χρησιμοποιώντας παραμετρικές καμπύλες κυβικών πολυωνύμων.

2.3 Καμπύλες Spline

2.3.1 Spline

Ορισμός 2.6 *Μια πραγματική συνάρτηση $S(x), [a,b] \subseteq \mathbb{R} \to \mathbb{R}$ και $a = x_0 < x_1 < \cdots < x_n = b$ ένας διαμερισμός του $[a,b]$. Καλείται spline βαθμού K αν πληροί τις ακόλουθες ιδιότητες:*

- $S \in C^{K-1}([a,b])$
- $S\,|_{x_i,x_{i+1}} = s_i$ *πολυώνυμο βαθμού το πολύ* K

Παρατήρηση 2.7 *Επομένως οι κυβικές spline είναι C^2 συναρτήσεις, όπου κατά τμήματα είναι πολυώνυμα, το πολύ τρίτου βαθμού.*

2.3.2 Παρεμβολή Hermite

Ένα βασικό μειονέκτημα που παρουσιάζουν οι μέθοδοι που ήδη αναφέραμε, είναι ότι σε περίπτωση που η καμπύλη που προκύπτει από τα σημεία που δίνουμε, δεν μας ικανοποιεί, η αλληλεπίδραση μας με αυτήν μπορεί να γίνει μόνο μέσω της προσθήκης νέων σημείων, κάτι που αυξάνει τους υπολογισμούς μας, ή αλλαγή της παραμέτρισης.

Παρατήρηση 2.8 *Το να αλλάξουμε κάποια σημεία, δεν αποτελεί πάντα λύση. Αφού μπορεί να είναι απαραίτητο η καμπύλη μας να περνάει από τα συγκεκριμένα, ούτε η αλλαγή της παραμέτρισης θα μας δίνει πάντα το επιθυμητό αποτέλεσμα.*

Επομένως προκύπτει η ανάγκη για μια μέθοδο που θα επιτρέπει στον χρήστη να προβαίνει εύκολα σε μετατροπές στην καμπύλη, μέσω του ελέγχου των παραμέτρων της. Η παρεμβολή Hermite είναι μια τέτοια μέθοδος. Ο λόγος που είναι πιο εύκολο να αλληλοεπιδράσουμε στην καμπύλη που παράγει αυτή η μέθοδος, είναι ότι δεν εξαρτάται μόνο από τα σημεία, αλλά και από την παράγωγο σε αυτά τα σημεία. Επομένως μπορούμε να μεταβάλλουμε την καμπύλη χωρίς να αλλάξουμε τα σημεία, αλλά επεμβαίνοντας στα εφαπτόμενα σε αυτά διανύσματα.

Έτσι μπορούμε να ορίσουμε μια καμπύλη ως ένα πολυώνυμο τρίτου βαθμού που οι συντελεστές του εξαρτώνται από 2 σημεία και 2 εφαπτόμενα διανύσματα.

Αν λοιπόν εκφράσουμε την πολυωνυμική καμπύλη σε παραμετρική μορφή:

$$\begin{aligned} &P(t) = at^3 + bt^2 + ct + d, t \in [0,1] \Rightarrow \\ &P'(t) = 3at^2 + 2bt + c, , t \in [0,1] \end{aligned} \tag{2.9}$$

Επομένως αν γνωρίζουμε τις ποσότητες, P_1, P_2, P_1', P_2' τότε μπορούμε, μέσω των ακόλουθων εξισώσεων να υπολογίσουμε τους συντελεστές a, b, c, d

$$\begin{aligned} &P(0) = P_1 \Rightarrow a \cdot 0^3 + b \cdot 0^2 + c \cdot 0 + d = P_1 \\ &P(1) = P_2 \Rightarrow a \cdot 1^3 + b \cdot 1^2 + c \cdot 1 + d = P_2 \\ &P'(0) = P_1' \Rightarrow 3a \cdot 0^2 + 2b \cdot 0 + c = P_1' \\ &P'(1) = P_2' \Rightarrow 3a \cdot 1^2 + 2b \cdot 1 + c = P_2' \end{aligned} \tag{2.10}$$

Αυτό το σύστημα εξισώσεων λύνεται εύκολα και έχουμε την εξής λύση

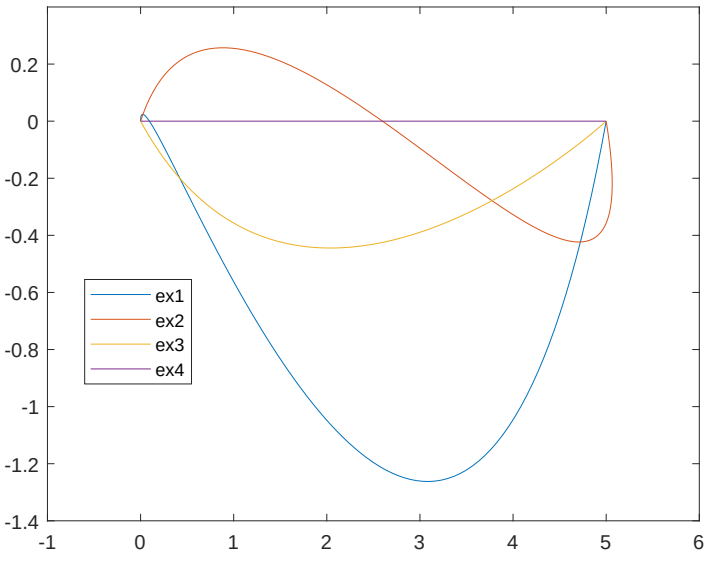
$$\begin{aligned} a &= 2P_1 - 2P_2 + P_1' + P_2' \\ b &= -3P_1 + 3P_2 - 2P_1' - 2P_2' \\ c &= P_1' \\ d &= P_1 \end{aligned} \tag{2.11}$$

Αυτό μας δίνει και την δυνατότητα να υλοποιήσουμε μια ρουτίνα στο Matlab που να υπολογίζει την τιμή του πολυωνυμικού τμήματος. Χρησιμοποιώντας απλούς πολλαπλασιασμούς διανυσμάτων και πινάκων, μιας και η καμπύλη $P(t)$ μπορεί να εκφραστεί από την εξής σχέση:

$$P(t) = (t^3, t^2, t, 1) \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} P_1 \\ P_2 \\ P_1' \\ P_2' \end{pmatrix}, t \in [0,1] \tag{2.12}$$

Κώδικας Matlab 2.4

```
function hpf = Hermite(P1,P2,dfP1,dfP2,N)
d = length(P1);
hpf=zeros(d,N);
t = linspace(0,1,N);
T = [t.^3;t.^2;t;ones(1,N)]';
M = [2 -2  1 1;
    -3 3 -2 -1;
    0 0 1 0;
    1 0 0 0];
for i = 1:d
    P = [P1(i),P2(i),dfP1(i),dfP2(i)]';
    hpf(i,:)=(T*M*P)';
end
```



Σχήμα 2.7: Παραδείγματα παρεμβολής Hermite με ίδια σημεία παρεμβολής, αλλά διαφορετικά εφαπτόμενα διανύσματα.

Δεν χρειάζεται να δούμε μεγαλύτερου βαθμού καμπύλες παρεμβολής Hermite και ο λόγος όπως εξηγήσαμε, είναι ότι στα γραφικά θα προτιμούμε να δουλεύουμε με πολυώνυμα μέχρι τρίτου βαθμού. Για να σχεδιάσουμε καμπύλες πιο σύνθετες, με την βοήθεια μόνο τέτοιου βαθμού πολυωνύμων θα καταφύγουμε σε μεθόδους παρεμβολής με κυβικές Spline.

2.3.3 Παρεμβολή με κυβικές Spline

Στη μέθοδο παρεμβολής με κυβικές spline, καλούμαστε να κατασκευάσουμε μια ομαλή καμπύλη που διέρχεται από n δοσμένα σημεία.

Η βασική αρχή της μεθόδου είναι, δεδομένων $P_1, P_2, \cdots, P_{n-1}, P_n$ σημείων του χώρου , να ορίσουμε $n-1$ ζευγάρια της μορφής $(P_i, P_{i+1}), i \in \{1, 2, \cdots, n-1\}$. Όπου για το κάθε τέτοιο ζεύγος θα υπολογίσουμε το κυβικό πολυώνυμο Hermite, έτσι ώστε ολόκληρη η καμπύλη μας να είναι C^2. Τα σημεία μας είναι δεδομένα, επομένως για να επιτύχουμε την επιθυμητή ομαλότητα πρέπει να βρούμε τα άγνωστα εφαπτόμενα διανύσματα.

Έστω λοιπόν ότι έχουμε τα τμήματα $P_k(t) = [P_k, P_{k+1}]$ και $P_{k+1}(t) = [P_{k+1}, P_{k+2}]$ για να έχουμε την ομαλότητα που θέλουμε όταν μεταβαίνουμε από το ένα τμήμα στο άλλο πρέπει $P'_k(1) = P'_{k+1}(0)$. Άρα για κάθε σημείο P_k υπάρχει μόνο ένα εφαπτόμενο διάνυσμα P'_k και αφού έχουμε n σημεία, θα έχουμε και n εφαπτόμενα διανύσματα. Επειδή θέλουμε η καμπύλη μας να είναι C^2 θα χρειαστούμε να υπολογίσουμε $n-2$ δεύτερες παραγώγους στα αντίστοιχα εσωτερικά σημεία. Αυτό θα μας οδηγήσει στην εύρεση $n-2$ το πλήθος αγνώστους. Τα δύο διανύσματα που μένουν για να συμπληρωθεί ο αριθμός, για τον σχεδιασμό της επιθυμητής καμπύλης, θα δίνονται από τον χρήστη. Και θα τα αποκαλούμε συνοριακές συνθήκες της μεθόδου.

Αφού η καμπύλη μας αποτελείται από κατά τμήματα πολυώνυμα τρίτου βαθμού της μορφής $P(t) = at^3 + bt^2 + ct + d$, η δεύτερη παράγωγος θα είναι το πολυώνυμό πρώτου βαθμού,

$$P^{(2)} = 6at + 2b. \tag{2.13}$$

Έστω το τμήμα της καμπύλης που περιέχει τα σημεία P_k, P_{k+1}, P_{k+2}. Για το εσωτερικό σημείο P_{k+1} ισχύει το εξής:

$$\begin{aligned} &P_k^{(2)}(1) = P_{k+1}^{(2)}(0) \Longleftrightarrow \\ &6a_k \cdot 1 + 2b_k = 6a_{k+1} \cdot 0 + 2b_{k+1} \end{aligned} \tag{2.14}$$

Αν τώρα εκφράσουμε τις ποσότητες $a_k, a_{k+1}, b_k, b_{k+1}$ συναρτήσει των 3 σημείων και των αντίστοιχων εφαπτόμενων διανυσμάτων όπως είδαμε στο σύστημα εξισώσεων 2.11, τότε οδηγούμαστε στην εξίσωση:

$$\begin{aligned} &6\left(2P_k - 2P_{k+1} + P'_k + P'_{k+1}\right) + 2\left(-3P_k + 3P_{k+1} - 2P'_k - 2P'_{k+1}\right) = \\ &6\left(2\left(P_{k+1} - P_{k+2}\right) + P'_{k+1} + P'_{k+2}\right) \Longleftrightarrow \\ &P'_k + 4P'_{k+1} + P'_{k+2} = 3(P_{k+2} - P_k) \end{aligned} \tag{2.15}$$

Και καταλήγουμε στο παρακάτω γραμμικό σύστημα $n-2$ εξισώσεων όπου έχουμε για αγνώστους, τις παραμέτρους $P'_k, k = 1, \cdots n$.

$$n-2 \begin{cases} \begin{pmatrix} 1 & 4 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 4 & 1 & \cdots & 0 \\ \vdots & \cdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 1 & 4 & 1 \end{pmatrix} \begin{pmatrix} P'_1 \\ P'_2 \\ \vdots \\ P'_n \end{pmatrix} = \begin{pmatrix} 3(P_3 - P_1) \\ 3(P_4 - P_2) \\ \vdots \\ 3(P_n - P_{n-2}) \end{pmatrix} \end{cases} \tag{2.16}$$

Αν τώρα δώσουμε αρχικές τιμές στις παραμέτρους P'_1, P'_n και επιλύσουμε το γραμμικό σύστημα με μια αριθμητική μέθοδο, όπως για παράδειγμα απαλοιφή Gauss με μερική οδήγηση, τότε θα έχουμε υπολογίσει και τις υπόλοιπες παραμέτρους που μας χρειάζονται για τον καθορισμό των συντελεστών των $n-1$ πολυωνύμων 3ου βαθμού Hermite. Και άρα θα μπορούμε να προχωρήσουμε στον σχεδιασμό της κυβικής Spline που θέλουμε να κατασκευάσουμε.

Την διαδικασία αυτήν, μπορούμε να την περιγράψουμε με τον παρακάτω αλγόριθμό, τον οποίο και θα υλοποιήσουμε αμέσως μετά.

Αλγόριθμός 2.1

1. *Εισαγωγή των n σημείων $P_1, \cdots, P_n$*
2. *Εισαγωγή των 2 εφαπτόμενων διανυσμάτων P_1', P_n'*
3. *Επίλυση συστημάτων $n-2$ αγνώστων για τον υπολογισμό των υπόλοιπων εφαπτόμενων διανυσμάτων έτσι ώστω να έχουμε την ομαλότητα που επιθυμούμε*
4. *Υπολογισμός των $n-1$ καμπύλων παρεμβολής Hermite για τα $n-1$ διαδοχικά ζευγάρια σημείων με τα αντίστοιχα εφαπτόμενα διανύσματα*

Με βάση αυτόν, κατασκευάζουμε την συνάρτηση της Matlab που ακολουθεί. Η οποία μας υπολογίζει τα σημεία των $n-1$ καμπυλών παρεμβολής Hermite για τα $n-1$ διαδοχικά ζευγάρια σημείων. Δέχεται σαν είσοδο, τα σημεία παρεμβολής IP, τα οποία δίνονται σαν πίνακας $n \times 2$, το πρώτο και το τελευταίο εφαπτόμενο διάνυσμα, που τα εισάγουμε σαν 2 διανύσματα και τέλος, τον πλήθος των σημείων που θέλουμε να υπολογίσουμε για κάθε ένα από τα τμήματα που θα παράξουν την τελική καμπύλη.

Κώδικας Matlab 2.5

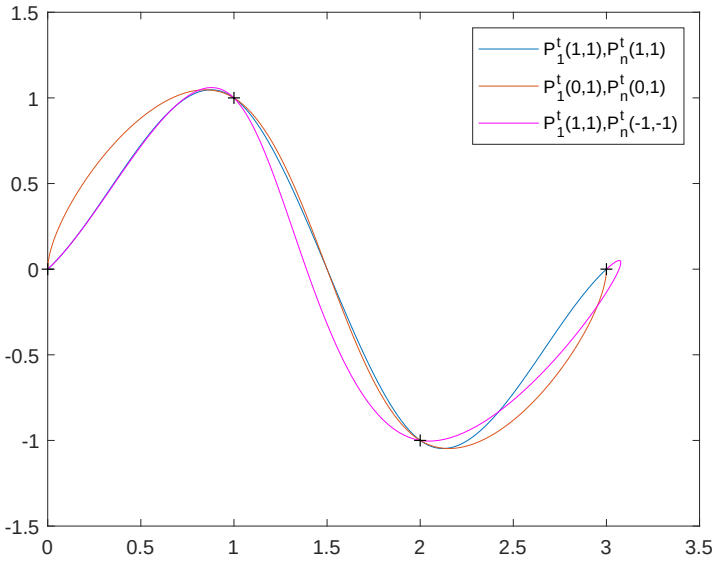
```
function SPoints = qubicspline(IP,dP1,dPn,np)
[m,n] = size(IP);
DP = zeros(m,n);
DP(:,1) = dP1';
DP(:,n) = dPn';
A = diag(4*ones(1,n-2))+diag(ones(1,n-3),1)+ ...
 diag(ones(1,n-3),-1);
for i = 1:m
    dP = zeros(n-2,1);
    dP(1) = 3*(IP(i,3)-IP(i,1))- dP1(i);
    dP(n-2) = 3*(IP(i,n)-IP(i,n-2))- dPn(i);
    for j = 2:n-3
        dP(j) = 3*(IntP(i,j+2)-IntP(i,j));
    end
    DP(i,2:n-1) = A\dP;
end
SPoints=Hermite(IP(:,1),IP(:,2),DP(:,1),DP(:,2),np);
for i = 2:n-1
    HP=Hermite(IP(:,i),IP(:,i+1),DP(:,i),DP(:,i+1),np);
    SPoints = [SPoints HP(:,2:end)];
end
```

Παράδειγμα 2.8 *Έστω τα σημεία παρεμβολής:*

$$P_0(0,0), P_1(1,1), P_2(2,-1), P_3(3,0)$$

και τα ακόλουθα εφαπτόμενα διανύσματα:

1. $P_1'(1,1), P_n'(1,1)$
2. $P_1'(0,1), P_n'(0,1)$
3. $P_1'(1,1), P_n'(-1,-1)$



Σχήμα 2.8: Οι κυβικές spline του παραδείγματος 2.8 με μαύρο + τα σημεία παρεμβολής

Φυσικές Κυβικές Spline

Στην περίπτωση που δεν γνωρίζουμε ή δεν επιθυμούμε να εισάγουμε τα 2 εφαπτόμενα διανύσματα, τα οποία όμως μας δίνουν την δυνατότητα να έχουμε έλεγχο των καμπυλών που παράγονται με τα συγκεκριμένα σημεία παρεμβολής, μπορούμε να θέσουμε εξ΄ αρχής τα $P_1''(0) = P_{n-1}''(1) = 0$. Αυτές οι συνοριακές συνθήκες, ονομάζονται φυσικές.

Η συνθήκη $P_1''(0) = 0$ μας δίνει μέσω της εξίσωσης 2.9 ότι $-3P_1+3P_2-2P_1'-P_2' = 0$. Από το οποίο προκύπτει $P_1' = \frac{3}{2}(P_2 - P_1) - \frac{1}{2}P_2'$.

Αντίστοιχα από το $P_{n-1}''(1) = 0$ και την εξίσωση 2.11 $\Rightarrow P_n' = \frac{3}{2}(P_n - P_{n-1}) - \frac{1}{2}P_{n-1}'$. Έτσι οδηγούμαστε στο ακόλουθο γραμμικό σύστημα με $n-2$ εξισώσεις και άλλους τόσους αγνώστους:

$$\begin{pmatrix} 1 & 4 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 4 & 1 & \cdots & 0 \\ \vdots & \cdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 1 & 4 & 1 \end{pmatrix} \begin{pmatrix} \frac{3}{2}(P_2 - P_1) - \frac{1}{2}P_2' \\ P_2' \\ \vdots \\ \frac{3}{2}(P_n - P_{n-1}) - \frac{1}{2}P_{n-1}' \end{pmatrix} = \begin{pmatrix} 3(P_3 - P_1) \\ 3(P_4 - P_2) \\ \vdots \\ 3(P_n - P_{n-2}) \end{pmatrix} \tag{2.17}$$

Το οποίο μπορούμε να το δούμε και σε αυτή την μορφή:

$$\begin{pmatrix} \frac{7}{2} & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 4 & 1 & \cdots & 0 \\ \vdots & \cdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 1 & \frac{7}{2} \end{pmatrix} \begin{pmatrix} P_2' \\ P_3' \\ \vdots \\ P_{n-1}' \end{pmatrix} = \begin{pmatrix} 3(P_3 - P_1) - \frac{3}{2}(P_2 - P_1) \\ 3(P_4 - P_2) \\ \vdots \\ 3(P_n - P_{n-2}) - \frac{3}{2}(P_n - P_{n-1}) \end{pmatrix} \tag{2.18}$$

και σε συνδυασμό με τις σχέσεις:

$$\begin{aligned} P_1' &=' \frac{3}{2}(P_2 - P_1) - \frac{1}{2}P_2' \\ P_n' &= \frac{3}{2}(P_n - P_{n-1}) - \frac{1}{2}P_{n-1}' \end{aligned} \tag{2.19}$$

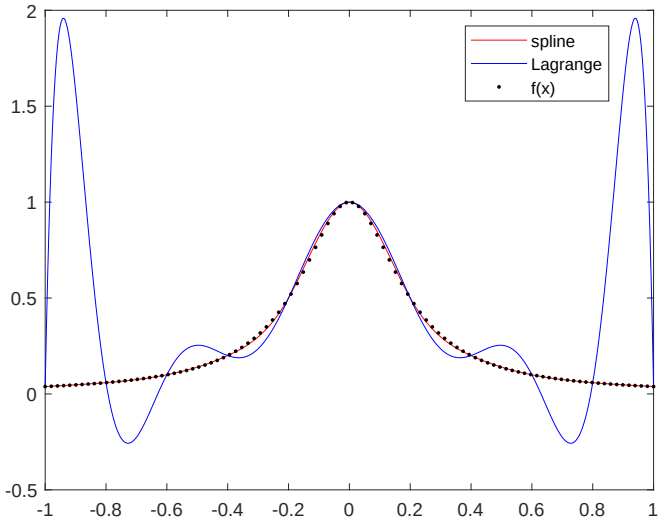
Υπολογίζουμε όλες τις τιμές για τα εφαπτόμενα διανύσματα και μπορούμε να υλοποιήσουμε την παρακάτω συνάρτηση.

Κώδικας Matlab 2.6

```
function SPoints = splinerelax(IP,np)
[m,n] = size(IP);
DP = zeros(m,n);
A = diag([7/2 4*ones(1,n-4) 7/2])+ ...
    diag(ones(1,n-3),1)+diag(ones(1,n-3),-1);
X = 3*[IP(:,2)-IP(:,1) IP(:,n)-IP(:,n-1)]/2;
for i = 1:m
    dP = zeros(n-2,1);
    dP(1) = 3*(IP(i,3)-IP(i,1))- X(i,1);
    dP(n-2) = 3*(IP(i,n)-IP(i,n-2))- X(i,2);
    for j = 2:n-3
        dP(j) = 3*(IP(i,j+2)-IP(i,j));
    end
    DP(i,2:n-1) = A\dP;
    DP(i,1) = X(i,1)-DP(i,2)/2;
    DP(i,n) = X(i,2)-DP(i,n-1)/2;
end
SPoints = Hermite(IP(:,1),IP(:,2),DP(:,1),DP(:,2),np);
for i = 2:n-1
    HP = Hermite(IP(:,i),IP(:,i+1),DP(:,i),DP(:,i+1),np);
    SPoints = [SPoints HP(:,2:end)];
end
```

Παράδειγμα 2.9 *Έστω τα σημεία παρεμβολής, που είχαμε και στο* ***Παράδειγμα*** *2.7. Αλλά αυτή την φορά, εκτός από την μέθοδο Lagrange θα εφαρμόσουμε και μέθοδο κυβικής spline με φυσικές συνοριακές συνθήκες.*

Με το παράδειγμά αυτό, θα δούμε πως ξεπερνάμε το πρόβλημα της ταλάντωσης στα άκρα της καμπύλης, αν έχουμε μεγάλο αριθμό σημείων παρεμβολής, σε αντίθεση με τις αρχικές μας μεθόδους.



Σχήμα 2.9: Υλοποίηση για το **Παράδειγμα** 2.9

Κυκλικές Κυβικές Spline

Εκτός από τις φυσικές συνοριακές συνθήκες μπορούμε να εισάγουμε και άλλες ανάλογα με τον τύπο καμπύλης που θέλουμε. Για παράδειγμα, ιδανικές συνθήκη για κλειστές καμπύλες είναι η λεγόμενη και cyclic end condition.Η συνθήκη αυτή, απαιτεί ότι $P_1' = P_n'$ και $P_1'' = P_n''$.

Από την $P_1'(0) = P_{n-1}'(1)$ και την εξίσωση 2.9 παίρνουμε:

$$c_1 = 3a_{n-1} + 2b_{n-1} + c_{n-1} \tag{2.20}$$

ενώ από την $P_1''(0) = P_{n-1}''(1)$ και την εξίσωση 2.13 έχουμε:

$$2b_1 = 6a_{n-1} + 2b_{n-1} \tag{2.21}$$

Αν τώρα αφαιρέσουμε τις εξισώσεις 2.20 και 2.21 και σε συνδυασμό με την εξίσωση 2.11 θα οδηγηθούμε στην σχέση:

$$\begin{aligned} &P_1' - 3\left(-3P_1 + 3P_2 - 2P_1' - P_2'\right) = \\ &-3\left(2P_{n-1} - 2P_n - P_{n-1}' + P_n'\right) + P_{n-1}' \Rightarrow \\ &5P_1' + 3P_n' = 6\left(P_2 - P_1 + P_n - P_{n-1}\right) - \left(P_2' + P_{n-1}'\right) \end{aligned} \tag{2.22}$$

Και χρησιμοποιώντας ότι $P_1' = P_n'$

$$P_1' = P_n' = \frac{3}{4}\left(P_2 - P_1 + P_n - P_{n-1}\right) - \frac{1}{4}\left(P_2' + P_{n-1}'\right) \tag{2.23}$$

Και από εκεί να πάμε στο γραμμικό σύστημα $n-2$ εξισώσεων με $n-2$ αγνώστους:

$$\begin{pmatrix} 15/4 & 1 & 0 & 0 & \cdots & 0 & -1/4 \\ 0 & 1 & 4 & 1 & 0 & \cdots & 0 \\ \vdots & \cdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & 1 & 4 & 1 \\ -1/4 & 0 & \cdots & \cdots & 0 & 1 & 15/4 \end{pmatrix} \begin{pmatrix} P_2' \\ P_3' \\ \vdots \\ P_{n-2}' \\ P_{n-1}' \end{pmatrix} = \begin{pmatrix} 3(P_3 - P_1) - \frac{3}{4}(P_2 - P_1 + P_n - P_{n-1}) \\ 3(P_4 - P_2) \\ \vdots \\ 3(P_{n-1} - P_{n-3}) \\ 3(P_n - P_{n-2}) - \frac{3}{4}(P_2 - P_1 + P_n - P_{n-1}) \end{pmatrix} \tag{2.24}$$

Παρατήρηση 2.9 *Παρατηρούμε ότι σε αντίθεση με τα προηγούμενα συστήματα, που ήταν τριδιαγώνια, εδώ ο πίνακας έχει στις θέσεις $(1,n)$ και $(n,1)$ μη μηδενική τιμή.*

Αφού λύσουμε το σύστημα 2.24 με την βοήθεια της σχέσης 2.23 θα υπολογίσουμε και την τιμή για τα P_1', P_n'. Και μπορούμε να προχωρήσουμε στην υλοποίηση στο Matlab.

Κώδικας Matlab 2.7

```
function SPoints = splinecyclic(IP,np)
[m,n] = size(IP);
DP = zeros(m,n);
A = diag([15/4 4*ones(1,n-4) 15/4])+ ...
    diag(ones(1,n-3),1)+diag(ones(1,n-3),-1);
```

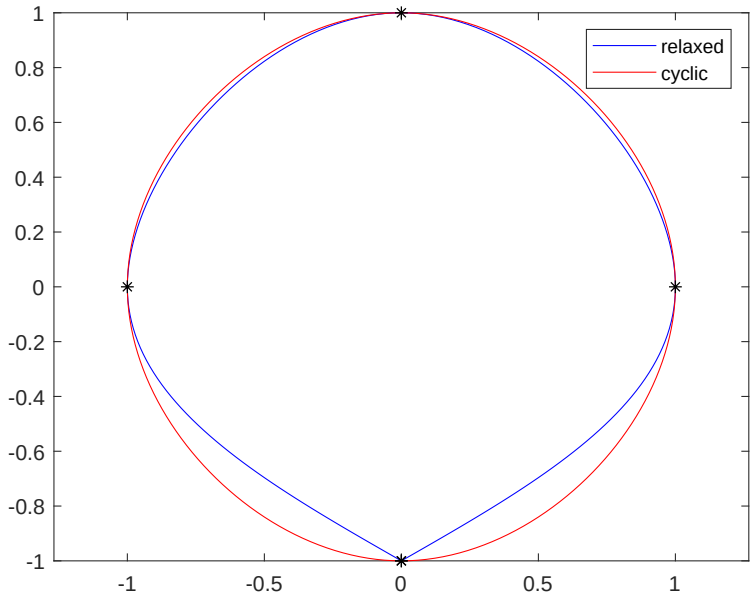
```
A(1,n-2) = -1/4;
A(n-2,1) = -1/4;
X = 3*(IP(:,2)-IP(:,1)+ IP(:,n)-IP(:,n-1))/4;
for i = 1:m
    dP = zeros(n-2,1);
    dP(1) = 3*(IP(i,3)-IP(i,1))-X(i);
    dP(n-2) = 3*(IP(i,n)-IP(i,n-2))-X(i);
    for j = 2:n-3
        dP(j) = 3*(IP(i,j+2)-IP(i,j));
    end
    DP(i,2:n-1) = A\dP;
    DP(i,1) = X(i)-(DP(i,2)+DP(i,n-1))/4;
    DP(i,n) = DP(i,1);
end
SPoints = Hermite(IP(:,1),IP(:,2),DP(:,1),DP(:,2),np);
for i = 2:n-1
    HP = Hermite(IP(:,i),IP(:,i+1),DP(:,i),DP(:,i+1),np);
    SPoints = [SPoints HP(:,2:end)];
end
```

Παράδειγμα 2.10 *Έστω τα σημεία παρεμβολής:*

$$P_1(0,-1), P_2(1,0), P_3(0,1), P_4(-1,0), P_5(0,-1)$$

Είναι προφανές ότι το ζητούμενο είναι να σχεδιασθεί μια κλειστή καμπύλη αφού το πρώτο και το τελευταίο σημείο παρεμβολής συμπίπτει.

Θα χρησιμοποιήσουμε Φυσικές κυβικές spline και Κυκλικές κυβικές spline για να σχεδιάσουμε καμπύλες που θα περνάνε από αυτά τα πέντε σημεία παρεμβολής.

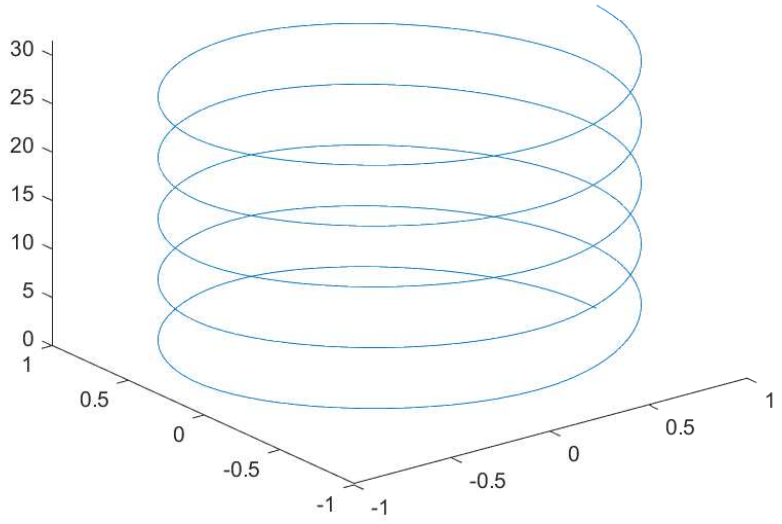


Σχήμα 2.10: Υλοποίηση για το **Παράδειγμα** 2.10

Παρατηρούμε ότι οι κυκλικές συνοριακές συνθήκες μας δίνουν ένα πιο ομαλό αποτέλεσμα, για μια κλειστή καμπύλη.

Η μέθοδος αυτή εκτός από κλειστές καμπύλες, δουλεύει και για καμπύλες που παρουσιάζουν μια περιοδικότητα, χωρίς υποχρεωτικά να είναι κλειστές. Όπως θα δούμε στο επόμενο παράδειγμα.

Παράδειγμα 2.11 *Η παραμετρική συνάρτηση* $F(t) = (\cos t, \sin t, t)$ *περιγράφει μια έλικα. Αν πάρουμε την ομοιόμορφη διαμέριση* t *του διαστήματος* $[0, 10\pi]$ *και βήμα* $\pi/2$*. Και εφαρμόσουμε κυκλικές κυβικές spline. Τότε θα πετύχουμε να αναπαραστήσουμε την έλικα στον χώρο, όπως φαίνεται και στο σχήμα που ακολουθεί.*



Σχήμα 2.11: Έλικα

2.3.4 Άλλα Είδη Spline

Εκτός από τις κυβικές spline των οποίων τις συνοριακές συνθήκες είδαμε. Υπάρχουν και πολλές άλλες που χρησιμοποιούνται στα γραφικά, όπως για παράδειγμα, οι αντικυκλικές (anticyclic) όπου $P_1' = -P_n'$ και $P_1'' = -P_n''$ ή άλλου τύπου καμπύλες όπως οι Akima spline που τις ανέπτυξε ο Hiroshi Akima στα τέλη της δεκαετίας του 60 [10] στην προσπάθειά του να πετύχει ένα αποτέλεσμα που θα προσομοίωζε όσο γίνεται περισσότερο μια ομαλή καμπύλη σχεδιασμένη στο χέρι.

Ακόμα αν και δεν είναι τυπικά spline υπάρχουν και άλλες κατά τμήματα κυβικές πολυωνυμικές καμπύλες, που χρησιμοποιούνται στα γραφικά. Και λέμε ότι τυπικά δεν είναι spline, μιας και είναι μόνο C και όχι C^2 όπως απαιτεί ο ορισμός. Ένα τέτοιο παράδειγμα είναι οι καμπύλες Catmull-Rom Curves όπου το πολυώνυμό κάθε τμήματος, προκύπτει από την σύνθεση δύο παραβολών.

Έκρινα όμως καλό, να μη παρουσιάσω άλλες καμπύλες παρεμβολής και να προχωρήσω στο επόμενο κεφάλαιο, που θα δούμε καμπύλες προσέγγισης.

Κεφάλαιο 3

Καμπύλες Προσέγγισης

Οι μέθοδοι που αναφέραμε στο προηγούμενο κεφάλαιο, αυτές δηλαδή που βασίζονται σε μεθόδους παρεμβολής, είναι σίγουρα εξαιρετικές όταν το σχήμα που θέλουμε να σχεδιάσουμε προέρχεται από πειραματικά δεδομένα ή μέσα από μαθηματικούς υπολογισμούς. Τέτοια είναι τα φτερά ενός αεροπλάνου ή τα μηχανικά μέρη ενός αντικειμένου. Αλλά υπάρχουν και περιπτώσεις που η αναγκαιότητα να γνωρίζουμε τα σημεία που θα διέρχεται το σχήμα αλλά και τα αντίστοιχα εφαπτόμενα διανύσματα, μας στερεί την αισθητική από το σχήμα μας και ενδεχομένως να μην μπορεί να τυποποιηθεί.

Παραδείγματα από τον πραγματικό κόσμο μπορούμε να βρούμε στον σχεδιασμό ενός επίπλου, ενός ποτηριού ή της επιφάνειας ενός αμαξιού. Για αυτές τις περιπτώσεις κάνουμε χρήση μιας άλλης μεθόδου παραγωγής καμπυλών. Τις καμπύλες που δημιουργούν αυτές οι μέθοδοι τις ονομάζουμε καμπύλες προσέγγισης.

Στις καμπύλες αυτού του είδους, ο χρήστης εισάγει σημεία ελέγχου της καμπύλης, τα οποία δεν είναι απαραίτητό να ανήκουν σε αυτήν. Συνήθως μόνο το πρώτο και το τελευταίο αποτελούν μέρος της.

Οι καμπύλες που προκύπτουν από μεθόδους προσέγγισης, αποτελούν εναλλακτική επιλογή από αυτές που προκύπτουν από μεθόδους παρεμβολής. Αλλά στην πράξη τις προτιμούμε μιας και οι αλγόριθμοι υπολογισμού τους παρουσιάζουν μικρότερη πολυπλοκότητα από αυτή των αντίστοιχων μεθόδων παρεμβολής.

3.1 Καμπύλες Bezier

Οι καμπύλες Bezier είναι οι πιο γνωστές καμπύλες προσέγγισης. Οφείλουν το όνομά τους στον μαθηματικό Pierre Bezier, οποίος πρώτος δημοσίευσε την δουλειά του πάνω σε αυτές. Αν και πρωτοπόρος στην μελέτη τους ήταν ο επίσης μαθηματικός Paul de Faget de Casteljau. Αποτελούν ένα πολύ σημαντικό εργαλείο σχεδιασμού στα γραφικά, και προέκυψαν μέσα από την ανάγκη μεγάλων αυτοκινητοβιομηχανιών να σχεδιάσουν τα προϊόντα τους. Και οι δύο δούλευαν σε δύο από τις πιο γνωστές Γαλλικές αυτοκινητοβιομηχανίες.

Στο κεφάλαιο αυτό θα μιλήσουμε αναλυτικότερα για καμπύλες Bezier. Θα δούμε τις ιδιότητές τους καθώς και αλγόριθμους υπολογισμού τους.

3.1.1 Πολυώνυμα Bernstein

Οι καμπύλες Bezier είναι παραμετρικές καμπύλες $P(t)$ που εκφράζονται μέσω πολυωνυμικών συναρτήσεων με παράμετρο t. Ο βαθμός του πολυωνύμου εξαρτάται από το πλήθος των σημείων ελέγχου.

Ορισμός 3.1 *Η καμπύλη Bezier που προκύπτει από $P_0, P_1, \cdots, P_n$ σημεία ελέγχου, εκφράζεται μέσω ενός πολυωνύμου $P(t) \in \mathbb{P}_n$.*

$$P(t) = \sum_{i=0}^{n} P_i \cdot B_i, 0 \leq t \leq 1. \tag{3.1}$$

Παρατήρηση 3.1 *Δηλαδή όπως μπορούμε να δούμε, η καμπύλη προκύπτει από το άθροισμα των σημείων ελέγχου που το καθένα πολλαπλασιάζεται με ένα βάρος B_i. Και αφού όπως αναφέραμε οι Bezier είναι παραμετρικές καμπύλες με παράμετρο t και αφού προφανώς τα σημεία ελέγχου $P_0, P_1, \cdots, P_n$ είναι σταθερά, καταλήγουμε στο συμπέρασμα, ότι τα βάρη $B_i, i = 0, 1, \cdots, n$ είναι συναρτήσεις της παραμέτρου.*

Τα βάρη αυτά λοιπόν, δεν προκύπτουν με κάποιο τυχαίο τρόπο αλλά καθορίζονται από τα σημεία ελέγχου που θέλουμε να έχουν μεγαλύτερη βαρύτητα στην καμπύλη. Για παράδειγμα για τιμές t κοντά στο 0, πιο πολύ θα επηρεάζει το πρώτο σημείο ελέγχου και το τελευταίο δεν θα παίζει καθόλου ρόλο.

Και ο Bezier και ο de Casteljau επέλεξαν σαν συναρτήσεις βάρους τα πολυώνυμα Bernstein που ακολουθούν την επιθυμητή λογική.

Ορισμός 3.2 *Το πολυώνυμο Bernstein $B_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i}$, όπου $\binom{n}{i} = \frac{n!}{t!(n-i)!}$.*

Τα οποία μπορούν να υπολογιστούν μέσω της συνάρτησης:

Κώδικας Matlab 3.1

```
function b = bernstein(i,n,t)
a = nchoosek(n,i)*t^i;
b = a*(1-t)^(n-i);
```

Ειδικότερα ο Bezier στη διαδικασία για να καταλήξει σε αυτά έθεσε κάποιες προϋποθέσεις για τις συναρτήσεις βάρους. [1]

1. Πρέπει να εξασφαλίζουν ότι η καμπύλη θα διέρχεται από το πρώτο και το τελευταίο σημεία ελέγχου.

2. Το εφαπτόμενο διάνυσμα στο πρώτο σημείο ελέγχου πρέπει να είναι το $P_0' = P_1 - P_0$ και το αντίστοιχο διάνυσμα για το τελευταίο το $P_n' = P_n - P_{n-1}$.

3. Πρέπει να αθροίζουν στην μονάδα για κάθε t.

4. Το $P(k)(0)$ εξαρτάται μόνο από το P_0 και τα υπόλοιπα k γειτονικά του σημεία.

5. Πρέπει να είναι συμμετρικές ως προς t και $(1-t)$.

6. Πρέπει να εξασφαλίζουν ότι η καμπύλη δεν εξαρτάται από το σύστημα συντεταγμένων.

7. Εξασφαλίζουν ότι βρίσκεται πάντα στην εντός του κυρτού πολυγώνου που σχηματίζουν τα σημεία ελέγχου

Και πράγματι τα πολυώνυμα Bernstein τις πληρούν.

3.1.2 Ιδιότητες καμπύλης Bezier

Οι προϋποθέσεις που πληρούν τα πολυώνυμα Bernstein υπαγορεύουν στην πράξη και τις ιδιότητες των καμπυλών Bezier.[2]

1. Αν πάρουμε τα σημεία ελέγχου με την αντίθετη σειρά, δηλαδή την ξεκινήσουμε από το P_n και πάμε στο P_0 η καμπύλη μας θα μείνει η ίδια.

2. Η καμπύλη μας μένει αναλλοίωτη σε συσχετισμένους μετασχηματισμούς. Και επομένως αν θέλουμε να την μετασχηματίσουμε, αρκεί να εφαρμόσουμε τον μετασχηματισμό στα σημεία ελέγχου.

3. Η καμπύλη βρίσκεται πάντα μέσα στο κυρτό πολύγωνο που σχηματίζουν τα σημεία ελέγχου. Γιατί στα πολυώνυμα Bernstein ισχύει ότι $\sum_{i=0}^{n} B_{n,i}(t) = 1, \forall t$ και $0 \leq B_{n,i}(t) \leq, t \in [0,1]$.

4. Αν ορίσουμε μια νέα παράμετρο $t_1 = a + (b-a)t$ έτσι ώστε η παράμετρος της καμπύλης αντί να είναι στο διάστημα $[0,1]$ ανήκει στο διάστημα $[a,b]$ η καμπύλη μας θα παραμείνει αναλλοίωτη.

5. Η παράγωγος της καμπύλης Bezier δίνεται από τον τύπο

$$P'(t) = n \sum_{i=0}^{n-1} B_{n-1,i}(t)(P_{i+1} - P_i) \tag{3.2}$$

6. Τα βάρη της καμπύλης $B_{n,i}(t)$ παρουσιάζουν μέγιστο για $t = \frac{i}{n}$.

7. Ενώ μια αλλαγή σε ένα σημείο ελέγχου στην ουσία αλλάζει όλη την καμπύλη, εξαιτίας της μορφής που έχουν τα βάρη. Η αλλαγή παρατηρείται εντονότερα στα σημεία της που επιδρά περισσότερο εκείνο το σημείο ελέγχου.

Με την χρήση του ορισμού 3.1 και του κώδικά 3.1 καταλήγουμε στην εξής συνάρτηση:

Κώδικας Matlab 3.2

```
function Points = Bezier(CP,np)
t = linspace(0,1,np);
n = size(CP,2)-1;
Points = zeros(size(CP,1),np);
for k = 1 : length(t)
        h = 0;
        for i = 0 : n
        h = h + bernstein(i,n,t(k))*CP(:,i+1);
        end
        Points(:,k) = h;
end
```

3.1.3 Αλγόριθμος του de Casteljau

Ο τρόπος υπολογισμού μιας καμπύλης Bezier με βάση τον τύπο του ορισμού, μπορεί να είναι άμεσος, αλλά αποτελεί έναν όχι αποδοτικό αλγόριθμό για να υλοποιηθεί σε έναν υπολογιστή. Αφού θα συσσωρεύονται αριθμητικά σφάλματα, από την προσπάθεια υπολογισμού των διωνυμικών συντελεστών αλλά και από τις υψώσεις σε δύναμη των παραμέτρων. Στην πράξη λοιπόν εφαρμόζουμε τον αλγόριθμο του deCasteljau.

Αλγόριθμός 3.1 *Αλγόριθμος του de Casteljau*

1. *Εισάγουμε τα σημεία ελέγχου* $P_0, P_1, \cdots, P_n$.
2. *Για το* t *που θέλουμε να υπολογίσουμε το σημείο της καμπύλης, θέτουμε*
$$P_i^0(t) = P_i, i = 0, 1, \cdots, n.$$
3. *Για κάθε* r *από 1 εώς* n *υπολογίζουμε τα:*
$$P_i^r(t) = (1-t) \cdot P_i^{r-1}(t) + t \cdot P_{i+1}^{r-1}(t), i = 0, 1, \cdots, n-r$$
4. *Το σημείο της καμπύλης για την τιμή* t *που θέλουμε να υπολογίσουμε είναι το* $P_0^n(t)$.

Τον οποίο αλγόριθμό μπορούμε να τον υλοποιήσουμε στο Matlab με την παρακάτω συνάρτηση:

Κώδικας Matlab 3.3

```
function BP = Casteljau(CP,np)
[m,n] = size(CP);
P = CP';
BP = zeros(m,np);
t=linspace(0,1,np);
for j = 1:np
    P = CP';
    for r = 2:n
        for i = 1:n-r+1
            P(i,:) = (1-t(j))*P(i,:)+t(j)*P(i+1,:);
        end
    end
BP(:,j) = P(1,:)';
end
```

3.1.4 Σύγκριση Αλγορίθμων Καμπυλών Bezier

Παράδειγμα 3.1 *Έστω τα σημεία ελέγχου:*

$$P_0(0,0), P_1(2,2), P_2(4,0), P_3(6,-2), P_4(8,0)$$

Και εφαρμόζουμε τις 2 συναρτήσεις που φτιάξαμε, ζητώντας να υπολογίσουν 5 σημεία της καμπύλης Bezier.

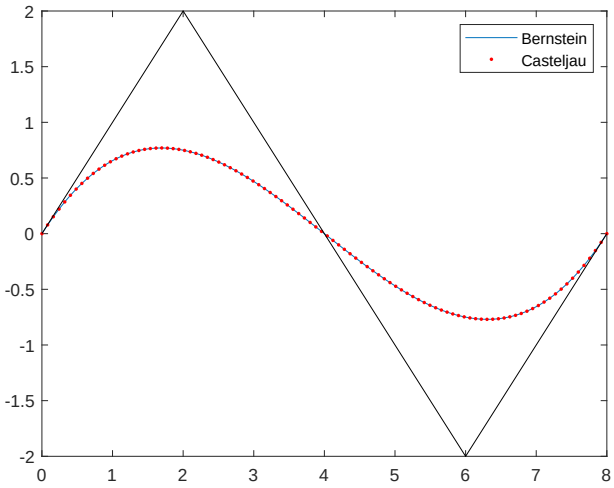
Και οι δύο θα βγάλουν τα ίδια σημεία για την καμπύλη μας. Τα οποία είναι τα :

$$B_0(0,0), B_1(2,0.75), B_2(4,0), B_3(6,-0.75), B_4(8,0)$$

Όπως βλέπουμε στο παράδειγμα 3.1 και στο σχήμα 3.1.4, και οι 2 τρόποι μπορούν να μας υπολογίσουν τα σημεία μιας καμπύλης Bezier με δεδομένα σημεία ελέγχου.

Από πλευράς αριθμητικών σφαλμάτων ήδη αναφέραμε ότι ο αλγόριθμός του de Casteljau έχει πλεονέκτημα απέναντι στη χρήση του τύπου, αφού έχει να κάνει μόνο με προσθέσεις και πολλαπλασιασμούς και όχι τόσες σύνθετες πράξεις σε σύγκριση με τον ορισμό.

Τι γίνεται όμως από τη πλευρά της πολυπλοκότητας ; Μήπως με την χρήση ενός υπολογιστικού πακέτου όπως το Matlab, που έχει αρκετές έτοιμες ρουτίνες βέλτιστα υλοποιημένες , συμφέρει να χρησιμοποιούμε τον κώδικα 3.2; Το ερώτημα αυτό μπορεί να απαντηθεί αν τρέξουμε το παρακάτω πρόγραμμα.



Σχήμα 3.1: Με μπλε γραμμή η καμπύλη όπως προκύπτει χρησιμοποιώντας τον τύπο με τα πολυώνυμα Bernstein ενώ τα κόκκινα σημεία προκύπτουν με de Casteljau. Με μαύρο βλέπουμε το πολύγωνο των σημείων ελέγχου.

Κώδικας Matlab 3.4

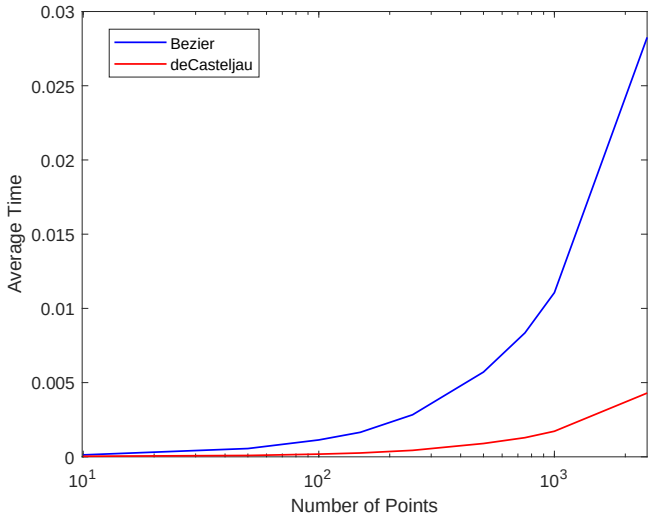
```
n = [10 50 100 150 250 500 750 1000 2500];
x =length(n);
TimeB = zeros(1,x);
TimeC = zeros(1,x);
for k = 1:x
        for i = 1:100

                CP = [0 2 4 6 8 10;randi(10,1,6)-5];
                tic;BP = Casteljau(CP,n(k));tc=toc;
                TimeC(k) = TimeC(k)+tc;
                tic;BP = Bezier(CP,n(k));tb=toc;
                TimeB(k) = TimeB(k)+tb;
        end
end
s1 = 'Number of Points';
s2 = 'Definition';
s3 = 'deCateljau';
s4 = 'Speed up factor'
TimeB = TimeB/100;
TimeC = TimeC/100;
fprintf('%s \t %s \t %s \t %s \n',s1,s2,s3,s4)
for i = 1:x
    fprintf('\t %d \t \t \t %f  \t \t  %f  \t \t  %4.2f \
       n',n(i),TimeB(i),TimeC(i),TimeB(i)/TimeC(i))
end
semilogx(n,TimeB,n,TimeC)
xlabel(s1)
ylabel('Average Time')
legend('Bezier','deCasteljau')
```

Το συγκεκριμμένο script φτιάχνει 6 τυχαία σημεία ελέγχου κάθε φορά, και υπολογίζει με βάση αυτά διάφορα το πλήθος σημεία της καμπύλης. Ξεκινώντας με το πλήθος 10 μέχρι και τα 2500 σημεία, ανά κλίση των συναρτήσεων. Για κάθε ένα από αυτά τα πλήθη σημείων, το κάνει 100 φορές. Και στο τέλος υπολογίζει τον μέσο χρόνο που χρειάστηκαν και οι 2 συναρτήσεις, για να τα βρουν. Στην συνέχεια μας εμφανίζει στην οθόνη έναν πίνακα με αυτά τα αποτελέσματα, και ένα γράφημα με τους χρόνους.

Παρατήρηση 3.2 *Με βάση λοιπόν τα αποτελέσματα που, καταλήγουμε στο συμπέρασμα, ότι όχι μόνο έχει λιγότερες σύνθετες πράξεις ο αλγόριθμός του de Casteljau και επομένως θα έχουμε και λιγότερα αριθμητικά σφάλματα. Αλλά είναι και σαφώς πιο γρήγορος. Ειδικότερα όταν χρειάζεται να υπολογίσουμε αρκετά σημεία μιας καμπύλης Bezier. Αυτό με την πρώτη ματιά στους 2 κώδικες 3.2 και 3.3 ενδεχομένως να ξενίζει μιας και φαίνεται σαν ο κώδικάς που βασίζεται στον ορισμό των καμπυλών να κάνει λιγότερα βήματα. Όμως μην ξεχνάμε ότι είμαστε σε περιβάλλον Matlab. Και η χρήση της εντολής nchoosek όπου υπολογίζει τους συνδυασμους μας κρύβει βήματα που δεν είμαστε σε θέση να τα μετρήσουμε. Και για αυτό επιλέγουμε σαν μέτρο σύγκρισης τον χρόνο.*

Εδώ βλέπουμε τα αποτελέσματα που επέστρεψε το Script:

$N.Points$	$Definition$	$deCasteljau$	$Speedupfactor$
10	0.000113	0.000023	4.90
50	0.000588	0.000092	6.39
100	0.001347	0.000208	6.46
150	0.001621	0.000251	6.47
250	0.002704	0.000414	6.54
500	0.005404	0.000826	6.54
750	0.008189	0.001255	6.52
1000	0.010783	0.001641	6.57
2500	0.027010	0.004125	6.55



Σχήμα 3.2: Διάγραμμα χρόνου υπολογισμού των 2 μεθόδων

3.1.5 Ομαλή Συνένωση Καμπυλών Bezier

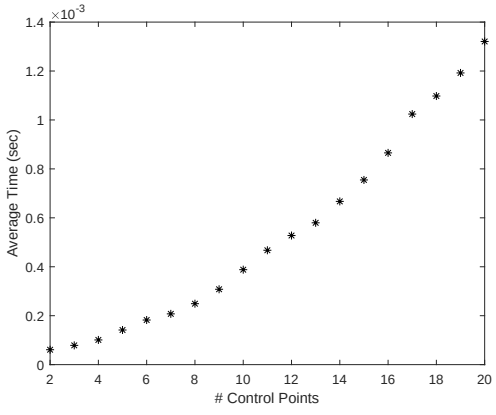
Από τον ορισμό της, μια καμπύλη Bezier μπορεί να είναι όσο μεγάλου βαθμού επιθυμούμε. Αρκεί να εισάγουμε τον επιθυμητό αριθμό σημείων ελέγχου. Όμως όσο μεγαλώνει ο βαθμός της, ανεβαίνει και η πολυπολοκότητα στον υπολογισμό των σημείων της καμπύλης. Κάτι που γίνεται άμεσα αντιληπτό, εάν εφαρμόσουμε τον ακόλουθο πειραματισμό.

Παράδειγμα 3.2 *Θα υπολογίσουμε για τυχαία σημεία ελέγχου διαφορετικού πλήθους σε κάθε βήμα, πόσο χρόνο χρειάζεται ο αλγόριθμός του deCasteljau για να υπολογίσει 100 σημεία της καμπύλης. Για κάθε βαθμό καμπύλης, θα δοκιμάσουμε 500 τυχαία σύνολα σημείων, και θα καταλλήξουμε στον μέσο χρόνο υπολογισμού για κάθε βαθμό καμπύλης. Αυτό θα το κάνουμε χρησιμοποιώντας το παρακάτω script στο Matlab.*

Κώδικας Matlab 3.5

```
ncp = 20;
TimeC = zeros(1,ncp-1);
for k = 2:ncp
        for i = 1:500
        CP = [0:k-1 ;randi(10,1,k)-5];
                tic;BP = Casteljau(CP,100);tc=toc;
                TimeC(k-1) = TimeC(k-1)+tc;
        end
end
s1 = '# Control Points';
s2 = 'Time';
TimeC = TimeC/500;
fprintf('%s \t %s \n',s1,s2)
for i = 1:ncp-1
    fprintf('\t %2d \t \t \t %f \n',i+1,TimeC(i))
end
plot(2:ncp,TimeC,'k*')
xlabel(s1)
ylabel('Average Time (sec)')
```

Τα αποτελέσματα φαίνονται στο σχήμα που ακολουθεί.



Σχήμα 3.3: Μέσος χρόνος υπολογισμού καμπύλης Bezier ανάλογα με το πλήθος των σημείων ελέγχου.

Όπως λοιπόν στις καμπύλες παρεμβολής επιλέξαμε να μείνουμε σε μικρού βαθμού πολυώνυμα παρεμβολής, έτσι και στις μεθόδους προσέγγισης επιλέγουμε να δουλέψουμε με πολυώνυμα το πολύ μέχρι τρίτου βαθμού.

Οπότε γυρνάμε πάλι στο κομμάτι των καμπυλών spline και στην απαίτηση για συνέχεια τουλάχιστον C^{n-1} αν ο βαθμός της καμπύλης είναι n.

Έστω λοιπόν ότι έχουμε $CP_i, i = 1, \cdots, 7$ σημεία ελέγχου, και θέλουμε ομαλή συνένωση δύο καμπυλών Bezier. Στην ουσία θα δημιουργήσουμε δύο τετράδες σημείων ελέγχου. Την τετράδα $[CP_1, CP_2, CP_3, CP_4]$ και την τετράδα $[CP_4, CP_5, CP_6, CP_7]$. Από αυτές και με βάση τον τύπο 3.1 έχουμε ότι η πρώτη καμπύλη είναι η $P_{c1}(t)$ και η δεύτερη η $P_{c2}(t)$.

Επομένως για να έχουμε C^2 πρέπει:

1. $P_{c1}(1) = Pc2(0)$.
2. $P'_{c1}(1) = P'_{c2}(0)$.
3. $P''_{c1}(1) = P''_{c2}(0)$.

Το πρώτο ισχύει αφού $P_{c1}(1) = CP_4 = Pc2(0)$. Για την πρώτη παράγωγο εχουμε αναφέρει στην εξίσωση 3.2 ότι

$$P'(t) = n \sum_{i=0}^{n-1} B_{n-1,i}(t)(P_{i+1} - P_i)$$

Επομένως $P'_{c1}(1) = 3(CP_4 - CP_3)$ και $P'_{c2}(0) = 3(CP_5 - CP_4)$, τα οποία πρέπει να είναι ίσα. Έτσι έπεται ότι:

$$3(CP_4 - CP_3) = 3(CP_5 - CP_4) \Leftrightarrow CP_5 = 2CP_4 - CP_3 \tag{3.3}$$

Άρα το CP_5 θα πρέπει να είναι συνευθειακό με τα CP_3, CP_4 και μάλιστα το CP_4 θα βρίσκεται στο μέσω του ευθύγραμμου τμήματος που θα ορίζουν τα CP_3, CP_5.

Για την ισότητα στις δεύτερες παραγώγους θα χρειαστεί πρώτα να δούμε ποιά είναι η παράγωγος στα άκρα μιας καμπύλης Bezier.

$$P''(t) = (P'(t))' = \left(n \sum_{i=0}^{n-1} B_{n-1,i}(t)(P_{i+1} - P_i) \right)' = n(n-1) \sum_{i=0}^{n-2} B_{n-2,i}(t)(P_{i+1} - P_i) \tag{3.4}$$

Η εξίσωση 3.4 μας δίνει για τις 2 καμπύλες μας στο κοινό τους άκρο,

$$\begin{aligned} &P''_{c1}(1) = 3 \cdot 2(CP_4 - 2CP_3 + CP_2) = 3 \cdot 2(CP_4 - 2CP_5 + CP_6) = P''_{c2}(0) \Rightarrow \\ &(CP_4 - 2CP_3 + CP_2) = (CP_4 - 2CP_5 + CP_6) \Rightarrow \end{aligned} \tag{3.5}$$

Αυτό σημαίνει ότι αν προεκτίνουμε τα 2 ευθύγραμμα τμήματα που ορίζουν τα $\overline{CP_2CP_3}$ και $\overline{CP_5CP_6}$ θα τέμνονται σε ένα κοινό σημείο. Εστω το σημείο αυτό είναι το MP, από τις σχέσεις μας έπεται ότι

$$|MPCP_3| = |CP_2CP_3| \tag{3.6}$$

Αλλά και ότι

$$|MPCP_5| = |CP_5CP_6| \tag{3.7}$$

Από τις εξισώσεις 3.6 και 3.7 έπεται ότι και το σημείο CP_6 θα εξαρτάται από τα σημεία ελέγχου της πρώτης καμπύλης. Και τελικά το μόνο σημείο ελέγχου που θα

είναι ανεξάρτητο από τα αντίστοιχα σημεία ελέγχου της πρώτης καμπύλης θα είναι το τελευταίο της.

Στην περίπτωση της ομαλή συνένωσης τριών κυβικών καμπυλών $P_1(t), P_2(t), P_3(t)$, η κατάσταση αρχίζει και γίνεται προβληματική. Και αυτό γιατί χάνουμε κάθε δυνατότα να χειριστούμε ανεξάρτητα τα κομμάτια της Spline Bezier.Έστω ότι οι καμπύλες $P_1(t), P_3(t)$ βρίσκονται στα άκρα της Spline μας. Αυτες μπορούμε να τις προσαρμόσουμε ανάλογα με το τι είναι επιθυμητό για εμάς, αλλάζοντας τα σημεία ελέγχου που τις δημιουργούν. Επειδή όμως θέλουμε να υπάρχει ομαλή συνένωση, η καμπύλη $P_2(t)$ θα καταλλήγει τελείως εξαρτημένη από τις άλλες δύο. Αφού τα σημεία ελέγχου της θα είναι αυστηρά καθορισμένα από την απαίτηση μας για ομαλότητα και άρα δεν θα έχουμε κανέναν τοπικό έλεγχο πάνω σε αυτήν.

Καταλήγουμε στο συμπέρασμε λοιπόν, ότι οι κυβικές καμπύλες Bezier δεν αποτελούν κατάλληλο εργαλείο στην περίπτωση που θέλουμε μέσω αυτών να δημιουργήσουμε μια spline Bezier που θα αποτελείτε από μικρού βαθμού πολυώνυμα όταν χρειαζόμαστε να περιγράψουμε μια πιο σύνθετη καμπύλη που θέλουμε να σχεδιάσουμε. Επομένως αν θέλουμε να δουλέψουμε με Bezier , αναγκαστικά πρέπει να πάμε σε χρήση καμπυλών μεγαλύτερου βαθμού κάθε φορά, που έχει όμως όλα τα μειονεκτήματα που περιγράψαμε στην αρχή της παραγράφου σχετικά με τον υπολογισμό τους αλλά και την απουσία τοπικού ελέγχου. Αφού αν αλλάξουμε ένα σημείο ελέγχου μιας καμπύλης στη προσπάθειά μας να πετύχουμε το ζητούμενο σχέδιο, είμαστε υποχρεωμένοι να υπολογίσουμε την καμπύλη από την αρχή για όλα τις τα σημεία.

3.2 B-Spline

Στα σύγχρονα προγράμματα σχεδιασμού γραφικών, επικρατούν οι B-spline μέθοδοι, που ξεπερνάνε τα όποια προβλήματα έχουν οι καμπύλες Bezier. Συγκεκριμένα οι καμπύλες B-spline έχουν τοπικό έλεγχο, και είναι μπορούμε πιο εύκολα να πάρουμε τον βαθμό συνέχειας που επιθυμούμε, ανάμεσα στα τμήματά της.

Οι καμπύλες B-spline είναι καμπύλες προσέγγισης, και ως εκ τούτου έχουν σημεία ελέγχου, αλλά επίσης ο χρήστης είναι απαραίτητο να εισάγει και επιπλέον δεδομένα για τον υπολογισμό τους. Τους λεγόμενους κόμβους,που ανάλογα αν είναι ομοιόμορφα κατανεμημένοι ή όχι δίνουν και άλλες κατηγορίες από καμπύλες.

Αν και ήταν γνωστές από την δεκαετία του 40, τις B-spline άρχισαν να τις εισάγουν στα γραφικά την δεκαετία του 70, με πρωτοπόρο τον Richard Riesenfeld με την διδακτορική του διατριβή [15]το 1973.

Ορισμός 3.3 *Έστω $U = \{t_0, \cdots, t_m\}$ μια μη φθίνουσα ακολουθία πραγματικών αριθμών. Τα t_i καλούνται κόμβοι. Και το U διάνυσμα κόμβων. [3]*

Ορισμός 3.4 *Την i-οστή B-Spline συνάρτηση βάσης βαθμού p την συμβολίζουμε $N_{i,p}(t)$ και ορίζεται αναδρομικά ως εξής*

$$
\begin{aligned}
N_{i,0}(t) &= \begin{cases} 1 & ,\epsilon\alpha\nu \quad t_i \le t < t_{i+1} \\ 0 & ,\alpha\lambda\lambda\iota\omega\varsigma \end{cases} \\
N_{i,p}(t) &= \frac{t - t_i}{t_{i+p} - t_i} N_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}(t)
\end{aligned}
\tag{3.8}
$$

[3]

Ορισμός 3.5 *Η καμπύλη B-Spline βαθμού p ορίζεται:*

$$C(t) = \sum_{i=0}^{n} N_{i,p}(t) P_i, 0 \leq t \leq 1$$

Όπου P_i τα σημεία ελέγχου και $N_{i,p}(t)$ οι B-Spline συναρτήσεις βάσεις βαθμού p, που είναι ορισμένες στο διάνυσμα κόμβων $U = \{t_0, \cdots, t_m\}$.

Για να υπολογίσουμε το σημείο της B-Spline για συγκεκριμένο t, κάνουμε τα εξής βήματα.

1. Προσδιορίζουμε που βρίσκεται το t σε σχέση με το διάνυσμα των κόμβων μας.
2. Υπολογίζουμε τις μη μηδενικές συναρτήσεις βάσεις μας για αυτό το κομμάτι.
3. Τις πολλαπλασιάζουμε με τα αντίστοιχα σημεία ελέγχου και τα αθροίζουμε.

3.2.1 Αλγόριθμός του de Boor

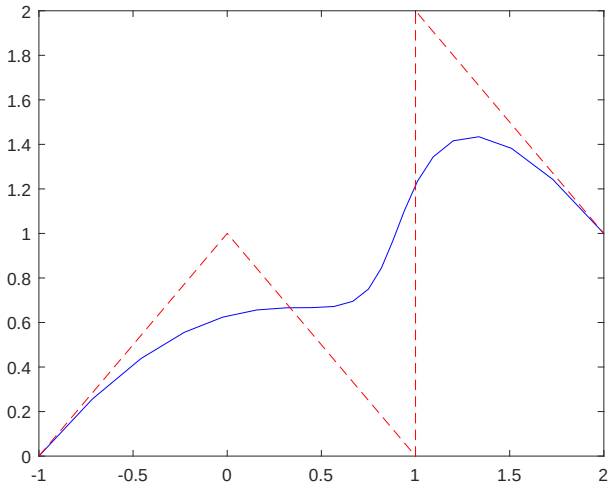
Αντίστοιχα με τον αλγόριθμο του de Casteljau για τον υπολογισμό ενός σημείου μιας καμπύλης Bezier, για τον υπολογισμό των σημείων μιας καμπύλης B-Spline χρησιμοποιούμε τον αλγόριθμό του de Boor [14]. Θα παρουσιάσω μια υλοποιησή του σε Matlab, η οποία μας δίνει καμπύλη που παρεμβάλει το πρώτο και το τελευταίο σημείο ελέγχου.

Όπου t είναι η τιμή στο διάστημά μας για την οποία θέλουμε να βρούμε το σημείο της καμπύλης. p ο βαθμός των τμημάτων της καμπύλης. CP τα σημεία ελέγχου και U το διάνυσμα των κόμβων.

Κώδικας Matlab 3.6

```
function P = deboor(CP,p,U,t)
dim = size(CP,1);
CP=CP';
if x == U(1,end)
    I=length(U)-p-1;
else
    id=find(U(p+1:(end-p))>t);
    I=id(1)+p-1;
end
cdb=CP((I-p):(I),:);
for i=1:p
    cnt=0;
    for j=I-p+i:I
        a=(x-U(j))/(U(j+p+1-i)-U(j));
        cnt=cnt+1;
        if a~=inf && isnan(a)==0
           for d = 1:dim
             cdb(cnt,d)=(1-a)*cdb(cnt,d)+a*cdb(cnt+1,d);
           end
        end
     end
end
P = cdb(1,:)';
```

Παράδειγμα 3.3 *Έστω τα σημεία ελέγχου* $P_1(-1,0), P_2(0,1), P_3(1,0), P_4(1,2), P_5(2,1)$ *και θέλουμε να σχεδιάσουμε την καμπύλη B-Spline βαθμού 3 με διάνυσμα κόμβων* $U = (0,0,0,0,0.5,1,1,1,1)$. *Το αποτέλεσμα το βλέπουμε στην εικόνα που ακολουθεί.*

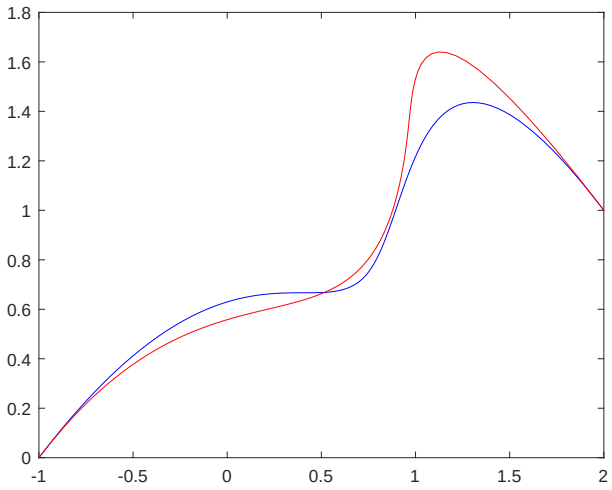


Σχήμα 3.4: Με μπλε η καμπύλη όπως προκύπτει χρησιμοποιώντας το κώδικα, ενώ με κόκκινο το πολύγωνο των σημείων ελέγχου.

3.2.2 Πλεονεκτήματα B-Spline σε σχέση με της Bezier

Τι κερδίζουμε όμως με τις B-Spline; Μέσω κάποιων απλών και γρήγορων παραδειγμάτων θα δούμε μερικά από τα πλεονεκτήματα.

Παράδειγμα 3.4 *Έστω τα σημεία ελέγχου* $P_1(-1,0), P_2(0,1), P_3(1,0), P_4(1,2), P_5(2,1)$ *όπως παραδείγματος 3.3. Θέλουμε να σχεδιάσουμε την καμπύλη B-Spline βαθμού 3 με διάνυσμα κόμβων* $U1 = (0,0,0,0,0.5,1,1,1,1)$. *Αλλά και την καμπύλη με διάνυσμα κόμβων* $U2 = (0,0,0,0,0.8,1,1,1,1)$.



Σχήμα 3.5: Με μπλε η καμπύλη όπως προκύπτει από το διάνυσμα κόμβων $U1$, ενώ με κόκκινο η καμπύλη όπως προκύπτει από το διάνυσμα κόμβων $U2$.

Αν θεωρήσουμε ότι έχουμε δεδομένα τα σημεία ελέγχου και δεν τα αλλάζουμε, η καμπύλη Bezier που θα προκύψει θα είναι μοναδική. Κάτι που προφανώς μας περιορίζει. Αυτό δεν συμβαίνει με τις B-Spline, αφού δεν εξαρτώνται μόνο από τα σημεία ελέγχου αλλά και από το διάνυσμα κόμβων. Επομένως, αν δώσουμε άλλο διάνυσμα κόμβων θα πάρουμε διαφορετική καμπύλη. Με αυτό τον τρόπο αποκτούμε και τοπικό έλεγχο στην καμπύλη μας, αφού πειράζοντας τους κόμβους καθορίζουμε τα διαστηματα που οι συναρτήσεις βάσεις θα μηδενίζονται και που θα συμμετέχουν στον υπολογισμό των σημείων της καμπύλης μας.

Παράδειγμα 3.5 *Αν γράψουμε το παρακάτω πολύ απλό κώδικά Matlab, ο οποίος παράγει τυχαία 10 σημεία ελέγχου. Και συγκρίνουμε τον μέσο χρόνο για τον υπολογισμό 1000 σημείων μια κυβικής καμπύλης B-Spline με διάνυσμα κόμβων που προκύπτει από την εντολή* $U = [0, 0, 0, linspace(0, 1, 7), 1, 1, 1]$ *και μιας ενάτου βαθμού καμπύλης Bezier, που προκύπτει από 1000 επαναλήψεις του πειράματος.*

Κώδικας Matlab 3.7

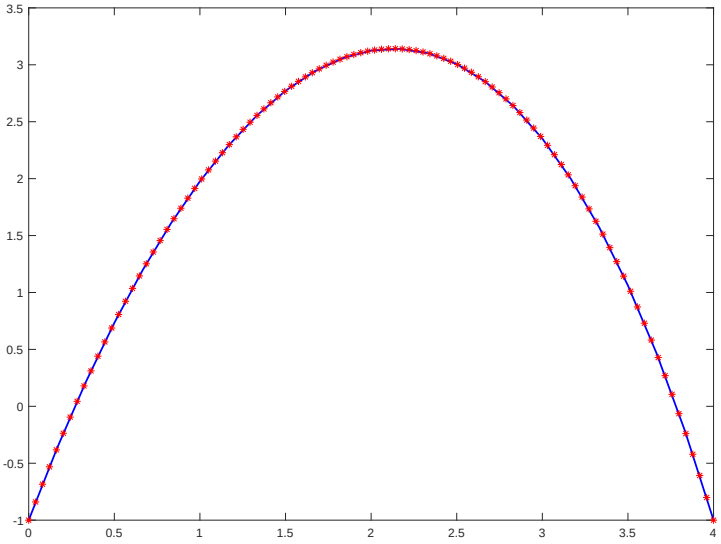
```
TimeBs = 0;
TimeBz = 0;
for i=1:1000
    CP=[0:9;randi(10,1,10)-5];
    BS = zeros(2,1000);
    U=[0 0 0 linspace(0,1,7) 1 1 1];
    tic;
    j=1;
    for t =linspace(0,1,1000)
        BS(:,j) = deboor(CP,3,U,t);
        j=j+1;
    end
    tbs = toc;
    TimeBs = TimeBs + tbs;
    tic;
    BP = Casteljau(CP,1000);
    tbp = toc;
    TimeBz = TimeBz + tbp;
end
disp('Average Time for B-Spline')
disp(TimeBs/1000)
disp('Average Time for Bezier')
disp(TimeBz/1000)
disp('Speed up factor')
disp((TimeBz/1000)/(TimeBs/1000))
```

Θα πάρουμε σαν μέσω χρόνο για τον αλγόριθμο του de Boor 0.0026 ενώ για τον αλγόριθμό του Casteljau 0.0037 ,δηλαδή περίπου 43 τις εκατό πιο πολύ. Και είναι κάτι λογικό, μιας και στην μια περίπτωση μένουμε σε πολυώνυμα μέχρι τρίτου βαθμού, ενώ στην άλλη πάμε σε υπολογισμό πολυωνύμων μέχρι ενάτου.

Παράδειγμα 3.6 *Έστω ότι θέλουμε να σχεδιάσουμε την B-Spline καμπύλη τετάρτου βαθμού που προκύπτει από 5 σημεία ελέγχου και με διάνυσμα κόμβων* $U = (0, 0, 0, 0, 0, 1, 1, 1, 1, 1)$. *Και την αντίστοιχη καμπύλη Bezier τετάρτου βαθμού από τα ίδια σημεία ελέγχου.*

Κώδικας Matlab 3.8

```
CP=[0:4;randi(10,1,5)-5];
n=4;
U = [zeros(1,5) ones(1,5)];
BS = zeros(2,25);
j=1;
for t =linspace(0,1,25)
    BS(:,j) = deboor(CP,4,U,t);
    j=j+1;
end
BP = Casteljau(CP,100);
plot(BS(1,:),BS(2,:),'b',BP(1,:),BP(2,:),'r*')
```



Σχήμα 3.6: Με μπλε η καμπύλη B-Spline, ενώ με κόκκινα αστεράκια η καμπύλη Bezier.

Παρατήρηση 3.3 *Βλέπουμε λοιπόν ότι οι δύο καμπύλες ταυτίζονται. Αυτό συμβαίνει γιατί οι B-Spline είναι η γενίκευση των Bezier. Οι συναρτήσεις βάσης για την ειδική αυτή μορφή κόμβων είναι ίσες με τα πολυώνυμα Bernstein*

Στην αρχή του κεφαλαίου ανέφερα τους λόγους για τους οποίους οι καμπύλες προσέγγισης αποτελούν στην πράξη καλύτερο εργαλείο σχεδιασμού από τις καμπύλες παρεμβολής στα χέρια ενός επαγγελματία. Κλείνοντας βλέπουμε ότι οι καμπύλες B-Spline είναι στην ουσία η ραχοκοκαλιά όλων αυτών των επαγγελματικών προγραμμάτων. Μιας και προσφέρουν γρήγορη υλοποίηση και τοπικό έλεγχο κατά τον σχεδιασμό με τρόπο εύχρηστο και γρήγορο.

Όλες οι μέθοδοι που είδαμε μπορούν με σχετικά απλές μετατροπές, από μεθόδους υπολογισμού σημείων καμπυλών, να εξελιχθούν σε μεθόδους υπολογισμού σημείων επιφανειών.

Βιβλιογραφία

Αγγλική Βιβλιογραφία

[1] Pierre Bezier. *The Mathematical Basis of the UNISURF CAD System*. Butterworth-Heinemann, 1986. ISBN: 0408221755.

[2] David Salomon. *The Computer Graphics Manual*. Springer Publishing Company, Incorporated, 2011. ISBN: 0857298852, 9780857298850.

[3] Les Piegl and Wayne Tiller. *The NURBS book*. 2nd. Springer Verlag, 1997. ISBN: 3-540-61545-8.

[4] Jerry Van Aken and Mark Novak. «Curve-drawing algorithms for Raster displays». In: *ACM Transactions on Graphics* 4 (2 Apr. 1985).

[5] Jack Bresenham. «Algorithm for Computer Control of a Digital Plotter». In: *IBM Syst. J.* 4.1 (Mar. 1965), pp. 25–30. ISSN: 0018-8670.

[6] M. L. V. Pitteway. «Algorithm for drawing ellipses or hyperbolae with a digital plotter». In: *The Computer Journal* 10 (3 Mar. 1967).

[7] Donald Hearn and M. Pauline Baker. *Computer Graphics*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1986. ISBN: 0-13-165382-2.

[8] G. Farin, J. Hoschek, and M.-S. Kim. *Handbook of Computer Aided Geometric Design*. 1st. North Holland & IFIP, 2002. ISBN: 0444511040, 9780444511041.

[9] Gabriel Taubin. «Distance approximations for rasterizing implicit curves». In: *ACM Transactions on Graphics* 13 (1 Jan. 1994).

[10] Hiroshi Akima. «A New Method of Interpolation and Smooth Curve Fitting Based on Local Procedures». In: *J. ACM* 17.4 (Oct. 1970). ISSN: 0004-5411.

[11] Pavel Emeliyanenko, Nicola Wolpert, and Kurt Mehlhorn. «Visualization of Points and Segments of Real Algebraic Plane Curves». In: 2007.

[12] Alois Zingl. «A Rasterizing Algorithm for Drawing Curves». Master Thesis. Technikum Wien, 2012.

[13] Hiroshi Akima. «A New Method of Interpolation and Smooth Curve Fitting Based on Local Procedures». In: *J. ACM* 17.4 (Oct. 1970), pp. 589–602. ISSN: 0004-5411. DOI: 10.1145/321607.321609. URL: https://doi.org/10.1145/321607.321609.

[14] Carl de Boor. *A practical guide to splines; rev. ed.* Applied mathematical sciences. Berlin: Springer, 2001. ISBN: 0-387-95366-3.

[15] Richard Franklin Riesenfeld. «Applications of B-spline Approximation to Geometric Problems of Computer-aided Design.» AAI7408299. PhD thesis. Syracuse, NY, USA, 1973.

[16] Duncan Marsh. *Applied Geometry for Computer Graphics*. 2nd. Berlin, Heidelberg: Springer-Verlag London, 2005. ISBN: 1852338016.

Ελληνική Βιβλιογραφία

[17] Θεοχάρης Θεοχάρης και Αλέξανδρος Μπεμ. *Γραφικά. Αρχές & Αλγόριθμοι*. Συμμετρία, 1999. ISBN: 978-960-11-0004-3.

[18] Γεώργιος Ακρίβης και Βασίλειος Δουγαλής. *Εισαγωγή Στην Αριθμητική Ανάλυση*. Πανεπιστημιακές Εκδόσης Κρήτης, 2002. ISBN: 978-960-524-022-6.

[19] Λεωνίδας Τσίτσας. *Εφαρμοσμένος Διανυσματικός Απειροστικός Λογισμός*. Συμμετρία, 2003. ISBN: 978-960-266-152-9.