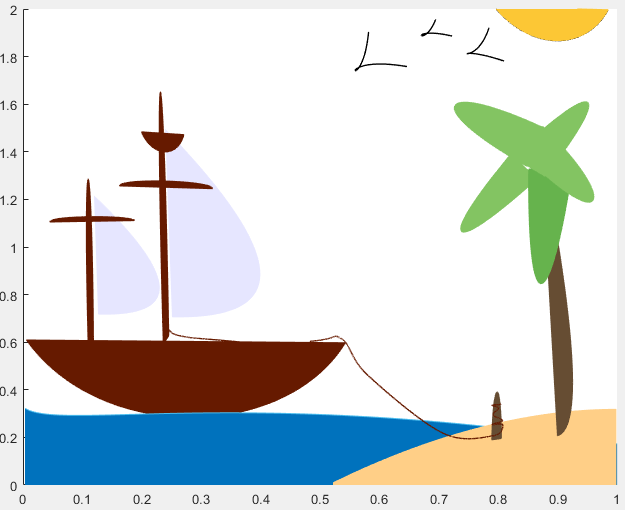
Curbe Bezier – Carribean

Nicoleta Radu, Matlab R2021b

# Design

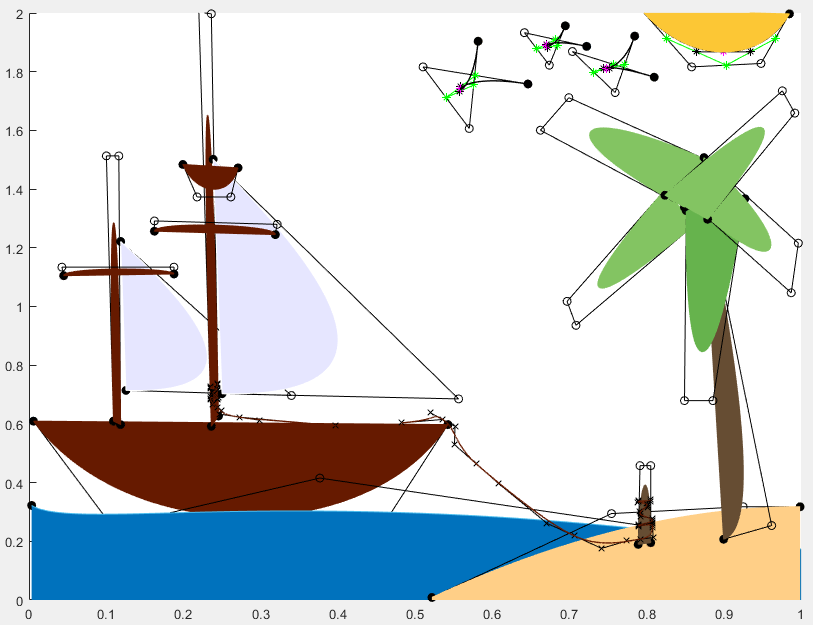
1. Peisajul

Ideea mea inițială a fost să construiesc un peisaj asemănător cu cel din Caraibe, fiind o mare fana a seriei “Pirații din Caraibe”.



1. Coordonatele XY

Punctele poligonului de control pentru fiecare dintre curbele Bezier au fost alese folosind doar intuiția mea, prin încercări și erori, folosind o funcție pe care am creat-o. In cadrul functiei este apelata funcția Matlab predefinită , “ginput”.



1. Tipul Curbelor Bezier

Tipul de curbe Bezier pe care l-am ales a fost strâns legat de cele 2 funcții, “fill” și “area”. Acestea se comporta în moduri diferite și, deși am folosit în principal funcția “fill”, am găsit diferite utilizări pentru ambele.

“Area(X,Y)” trasează valorile din Y în raport cu coordonatele X. Funcția umple apoi zonele dintre curbe pe baza formei lui Y. Acest lucru a fost util pentru desenarea oceanului și a solului insulei cu forma dorita de mine.

Produsul final are un total de *23 de curbe Bezier* de diferite tipuri, cum ar fi: pătratice, cubice și, de asemenea, de grade superioare.

# Structura

1. Functii

Pentru fiecare dintre curbele Bezier utilizate a existat o funcție separată creată special pentru tipul său. Am pus în aplicare cunoștințele acumulate de-a lungul semestrului academic și le-am folosit în avantajul meu în realizarea algoritmilor pe care îi voi prezenta ulterior. În total, am folosit 7 funcții pentru modularizarea proiectul conform cerințelor.

1. Structura Programului Principal

Programul principal este format din mai multe apeluri de funcții care ajută la desenarea curbelor Bezier trecând coordonatele X și Y pe care le-am ales ca argumente funcțiilor.

Pentru a obține și a salva coordonatele alese, am apelat funcția pe care am creat-o numită "getXY" și apoi am transmis valoarea returnată de funcție celorlalte funcții de desenare. Mai multe informații despre funcția "getXY" pot fi găsite ulterior, în secțiunea "Algoritmi utilizați".

Intrucat viziunea mea pentru acest peisaj era să nu am doar linii simple, am ales să folosesc funcțiile "fill" și "area" care creează poligoane umplute din datele în X și Y folosing culoarea vertexului specificată de C. În acest fel, am reușit să creez forme și să le dau culorile corespunzătoare. Coordonatele X și Y care sunt transmise celor 2 funcții, fill și area, sunt de fapt argumentele de ieșire ale funcțiilor de desenare a curbelor Bezier.

1. Datele de Intrare

O structură similară a fost folosită pentru toate funcțiile create pentru a face dezvoltarea proiectului cât mai fluentă. Datele de intrare utilizate reprezintă o matrice de dimensiunea 2x3 sau 2x4 care conține coordonatele X si Y ale poligonului de control al fiecărei curbe Bezier. Standardul implementat a fost ca prima linie a matricei să reprezinte coordonata X, iar a doua linie coordonata Y.

Se dau ca exemple urmatoarele antene de functii:

function [P] = drawBezier\_casteljau\_cubic(bezier\_coordinates)

function [x,y] = drawBezier\_quad(bezier\_coordinates)

function [x,y] = drawBezier\_cubic(bezier\_coordinates)

function [] = drawPolygon(coordinates)

1. Datele de iesire

Datele de ieșire ale funcțiilor reprezentau coordonatele X și Y ale curbei Bezier, care sunt apoi utilizate în programul principal cu scopul de a le implementa in functiile “area” si “fill”. Ambele coordonate sunt vectori de lungime 1x3, 1x4, sau mai mare.

# Algoritmi utilizați

1. Extragere si Salvare Coordonate XY

Pentru a extrage și pentru a salva coordonatele alese, am creat funcția numită "getXY".

Am obținut coordonatele prin apelarea funcției predefinite din Matlab, "ginput", urmat de transformarea vectorilor în mod corespunzător pentru a crea matricea de coordonate. Funcția “getXY” returnează o matrice care conține coordonatele X și Y, urmând standardul folosit în tot programul. Matricea poate avea practic orice număr de coloane, dar va avea întotdeauna 2 linii.

* + - 1. Functia

function [bezier\_coordinates] = getXY(n)

% Function to return matrix with chosen XY coordonates

% Input arguments represent Bezier Curve degree

% which turns into a n+1 points taken for matrix's coordinates

% Prompt user for number of points

[x,y] = ginput(n+1)

size\_x = length(x);

size\_y = length(y);

x = reshape(x,1,size\_x);

y = reshape(y,1,size\_y);

bezier\_coordinates = [ x(1:end) ; y(1:end)];

end

1. Desenare Poligoane de control

Una dintre cerințele proiectului a fost de a afișa poligonul de control pentru fiecare curbă Bezier, așa că am decis să creez o funcție separată numită "drawPolygon".

Această funcție a fost apelată doar în interiorul corpului funcțiilor de desen. Funcția "drawPolygon" poate fi ușor comentată odată ce nu mai este necesar să se deseneze poligoanele.

function [] = drawPolygon(coordinates)

% Draws polygon based on matrix's coordinates

% Input argument representz the coordinates matrix for XY

% matrix columns

size\_coordinates = max(size(coordinates))

hold on

plot(coordinates(1,1),coordinates(2,1),'o','markerfacecolor','k')

plot(coordinates(1,size\_coordinates),coordinates(2,size\_coordinates),'o','markerfacecolor','k')

plot(coordinates(1,:),coordinates(2,:),'ko')

plot(coordinates(1,:),coordinates(2,:),'-k')

end

1. Polinoamele Bernstein

Utilizând polinoamele lui Bernstein am creat următoarele 2 funcții: “drawBezier\_cubic” și “drawBezier\_quad”. Aceste funcții funcționează în moduri foarte asemănătoare și se comportă în același mod, cu excepția faptului evident că produc curbe de grade diferite.

“drawBezier\_quad” ia ca argument de intrare o matrice de dimensiune 2x3 și returnează coordonatele X și Y pentru desenarea unei curbe Bezier de gradul 2.

“drawBezier\_cubic” ia ca argument de intrare o matrice de dimensiune 2x4 și returnează coordonatele X și Y pentru desenarea unei curbe Bezier de gradul 3.

Următoarele obiecte au fost create folosind algoritmele prezentate: cadrul bărcii, catargul de prova, catargul principal, cuibul de cioară, catargul 1 și 2, solul insulei, țărușul, frunzele de palmier, trunchiul de palmier, vela principală 1 și 2 a bărcii.

* + - 1. Functia 1

function [x,y] = drawBezier\_cubic(bezier\_coordinates)

% Draws a cubic Bezier Curve using Bernstein polynomials

% Input arguments reprezent the following points:

% start,2 control,end

% time array

t = 0:0.001:1;

s = max(size(bezier\_coordinates));

% Bernstein polynomials

bernstein\_b0 = (1 - t).^3;

bernstein\_b1 = 3 .\* (1 - t).^2 .\* t;

bernstein\_b2 = 3 .\* (1 - t) .\* t.^2;

bernstein\_b3 = t.^3;

% Calculates XY coordinates

x = bezier\_coordinates(1,1)\*bernstein\_b0 + bezier\_coordinates(1,2)\*bernstein\_b1 + bezier\_coordinates(1,3)\*bernstein\_b2 + bezier\_coordinates(1,4)\*bernstein\_b3;

y = bezier\_coordinates(2,1)\*bernstein\_b0 + bezier\_coordinates(2,2)\*bernstein\_b1 + bezier\_coordinates(2,3)\*bernstein\_b2 + bezier\_coordinates(2,4)\*bernstein\_b3;

hold on

%drawPolygon(bezier\_coordinates)

plot(x,y,'b','LineWidth',0.2)

axis([0,1,0,2])

end

* + - 1. Functia 2

function [x,y] = drawBezier\_quad(bezier\_coordinates)

% Draws a quadratic Bezier Curve using Bernstein polynomials

% Input arguments reprezents the Bezier's control polygon coordinates

% The coordinates are XY (2D space) points

% time array

t = 0:0.001:1;

% number of matrix columns

s = max(size(bezier\_coordinates));

% Bernstein polynomials

bernstein\_b0 = (1 - t).^2;

bernstein\_b1 = 2 \* t.\* (1 - t);

bernstein\_b2 = t .^ 2;

% Calculates XY coordinates

x = bezier\_coordinates(1,1)\*bernstein\_b0 + bezier\_coordinates(1,2)\*bernstein\_b1 + bezier\_coordinates(1,3)\*bernstein\_b2;

y = bezier\_coordinates(2,1)\*bernstein\_b0 + bezier\_coordinates(2,2)\*bernstein\_b1 + bezier\_coordinates(2,3)\*bernstein\_b2;

hold on

%drawPolygon(bezier\_coordinates)

plot(x,y,'b','LineWidth',0.2)

axis([0,1,0,2])

end

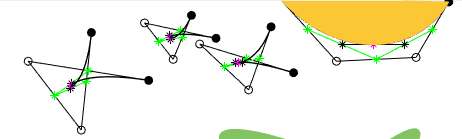
1. **De Casteljau’s Algorithm**

Pentru Algoritmul lui De Casteljau am creat 2 funcții: „drawBezier\_casteljau\_cubic” și „lerp”. Funcția numită „lerp" are o funcționalitate foarte simplă, care constă în returnarea valorilor calculate pentru 2 puncte date. Ea urmează formula matematică corespunzătoare (P = (1 - t) .\* p0 + t .\* p1) și a fost utilizată doar în cadrul funcției „drawBezier\_casteljau\_cubic”. În corpul funcției am calculat nivelurile corespunzătoare ale matricei sistolice și apoi am trasat atât curba Bezier, cât și structura sistolică.

Următoarele obiecte au fost create folosind algoritmul prezentat: soarele și pasărea 1,2 și 3.



Fig(1)



Fig(2)

1. Functia 1

function [P] = drawBezier\_casteljau\_cubic(M)

% De Casteljau's algorithm for drawing Bezier Curves

% Input arguments reprezent the following points:

% start,2 control points, end

% De Casteljau's calculation for finding the curve's point

A = lerp(M(:,1),M(:,2));

B = lerp(M(:,2),M(:,3));

C = lerp(M(:,3),M(:,4));

D = lerp(A,B);

E = lerp(B,C);

P = lerp(D,E);

% Calculations for systolic matrix

t = 1/2;

level\_1 = zeros(2,3);

level\_2 = zeros(2,2);

level\_3 = zeros(2,1);

for i=1:3

level\_1(:,i)=M(:,i).\*(1-t)+M(:,i+1).\*t; % level 1

end

for i=1:2

level\_2(:,i)=level\_1(:,i).\*(1-t)+level\_1(:,i+1).\*t; % level 2

end

level\_3(:,1) = level\_2(:,1).\*(1-t)+level\_2(:,2).\*t % level 3

hold on

drawPolygon(M) % Function that draws poligon

% systolic matrix

plot(level\_1(1,:),level\_1(2,:),'-g\*')

plot(level\_2(1,:),level\_2(2,:),'-k\*')

plot(level\_3(1,:),level\_3(2,:),'m\*')

% Bezier Curve

plot(P(1,:),P(2,:),'k','LineWidth',0.2)

axis([0,1,0,2])

end

1. Functia 2

function [P] = lerp(p0,p1)

% Lerping - implemented for De Casteljau's Algorithm

% Part of calculating & drawing a Bezier Curve

t = 0:0.01:1;

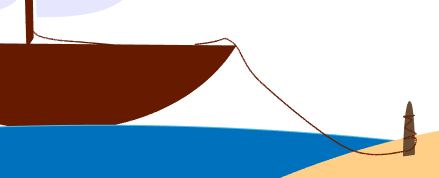
P = (1 - t) .\* p0 + t .\* p1;

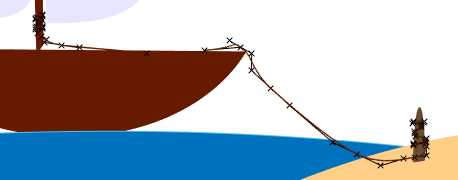
End

1. F-Mill

Intrucat ni s-a cerut să construim o funcție care să asigure continuitatea geometrică de clasă G1 a unei curbe Bezier spline, am creat o funcție care face exact acest lucru. Am ales să implementez această funcție folosind algoritmul F-Mill . De asemenea , cele 2 curbe Spline Bezier conțin cel puțin 3 segmente de curbe Bezier.

Folosind acest algoritm am creat 2 curbe Spline Bezier care reprezintă o bucată de frânghie legată de un țăruș. Am încercat să dau impresia de continuitate între cele 2 frânghii, dar ele sunt, de fapt, 2 curbe Spline Bezier diferite.





# Modificarea unui punct

1. **Punctul modificat**

Am ales una dintre cele 2 curbe Spline Bezier pe care le-am creat pentru a prezenta proprietatea de modificare locală a valorii unuia dintre punctele poligonului de control a unei curbe Bezier. Doar valoarea coordonatei Y a fost modificată, deoarece am încercat să cobor poziția "frânghiei" în acel punct specific al figurii.

Valoarea modificată este localizată la a doua linie, a treia coloană din matricea poligonului de control. Astfel, valoarea modificată a fost de la 0,463557 la 0,25.

1. **Rezultatul**

Judecând rezultatele modificării locale a spline-ului bezier în punctul menționat, s-ar părea că singura parte afectată a fost cea din vecinătatea punctului modificat. Conform rezultatului modificării, nicio altă parte a curbei Spline Bezier sau a locației punctelor poligonului de control nu a fost afectată.

**Matricea originala**:

[0.4827 0.5357 0.5795 0.7062 0.7915 0.8076 0.7892 0.8076 0.7915

0.6035 0.6152 **0.4636** 0.2187 0.2070 0.2536 0.2536 0.2711 0.2945

0.8007 0.7892 0.8053 0.7938 0.8076

0.3236 0.3353 0.3411 0.3703 0.3469]

**Matricea dupa schimbarea valorii de pe linia a doua, coloana a 3 a.**

[0.4827 0.5357 0.5795 0.7062 0.7915 0.8076 0.7892 0.8076

0.6035 0.6152 **0.2500** 0.2187 0.2070 0.2536 0.2536 0.2711

0.7915 0.8007 0.7892 0.8053 0.7938 0.8076

0.2945 0.3236 0.3353 0.3411 0.3703 0.3469]

